ISBN: 978-93-48620-19-4

Fundamentals of Data Engineering Concepts

Editors:
Dr. Sumit Chopra
Er. Ramandeep Kaur
Er. Gagandeep Singh

Bhumi Publishing, India
First Edition: April 2025

Fundamentals of Data Engineering Concepts

(ISBN: 978-93-48620-19-4)

Editors

Dr. Sumit Chopra

Associate Professor, GNA University, Phagwara

Er. Ramandeep Kaur

M. Tech. Student, GNA University, Phagwara

Er. Gagandeep Singh

Assistant Professor, GNA University, Phagwara

Bhumi Publishing

April 2025

Title: Fundamentals of Data Engineering Concepts

Editors: Dr. Sumit Chopra, Er. Ramandeep Kaur, Er. Gagandeep Singh

First Edition: April 2025

ISBN: 978-93-48620-19-4

Published by:

Bhumi Publishing

BHUMI PUBLISHING

Nigave Khalasa, Tal – Karveer, Dist – Kolhapur, Maharashtra, INDIA 416 207

E-mail: bhumipublishing@gmail.com

PREFACE

In the era of digital transformation, data has emerged as the new currency, powering decision-making across industries and shaping the future of technology. At the heart of this data-driven revolution lies data engineering—a discipline dedicated to designing, building, and maintaining systems that enable the collection, storage, processing, and analysis of vast and varied datasets. This book, "Fundamentals of Data Engineering Concepts", is crafted to introduce readers to the core principles and practices of this crucial field.

The book is structured to provide a foundational understanding of data engineering, beginning with an overview that sets the stage for more detailed explorations. It walks the reader through essential topics including sources and types of data, various file formats, data repositories, pipelines, and integration platforms. These chapters offer a clear view of how raw data is transformed into usable, structured information.

Further sections delve into the tools used for big data processing and data storage techniques that ensure reliability and scalability. The importance of data processing and efficient information retrieval is also highlighted, providing insights into how data is refined and accessed in real-world scenarios.

To give readers a broader perspective, the book introduces concepts like regression analysis, demonstrating how data engineering supports predictive modeling. It concludes with an engaging chapter on data visualization, emphasizing the power of graphical representation in communicating complex data insights effectively.

This volume serves as an essential resource for students, educators, and professionals who are beginning their journey into the world of data engineering. With a balanced approach combining theoretical foundations and practical insights, it aims to inspire a deeper understanding and appreciation of the systems and strategies that underpin modern data infrastructures.

We trust that this book will not only support academic learning but also encourage innovation and curiosity in one of the most impactful fields of our time.

- Editors

TABLE OF CONTENT

CHAPTER 1

INTRODUCTION TO DATA ENGINEERING

Ramandeep Kaur[1], Rubina Choudhary[2] and Gagandeep Singh[3]

[1,3]GNA University, Phagwara

[2]Lovely Professional University, Jalandhar

1.1 Definition:

Data engineering is a sub-discipline of data management that concentrates on creating, building, and overseeing scalable and effective data infrastructure. It raises to the collection, storage, processing, and transformation of raw data into structured formats that are available for analysis, machine learning, and decision-making. Data engineers construct data pipelines that facilitate the automatic transfer of data from diverse origins (including databases, APIs, IoT devices, and streaming platforms) into integrated storage entities, like data warehouses or data lakes.

Data engineers use ETL (Extract, Transform, Load) and ELT (Extract, Load, Transform) processes, big data frameworks (Apache Spark and Hadoop), and cloud platforms (Amazon Web Services (AWS), Azure, and GCP) and workflow automation tools (Apache Airflow and Prefect) to offer reliable and efficient automation. They mask, cleanse, normalize, and transform data to be fit for business use, addressing data quality issues.

Moreover, it is also essential in the realms of data governance, security, and compliance as it ensures the implementation of encryption, access control, and monitoring mechanisms for sensitive data. By streamlining data storage and retrieval processes, data engineers help organizations extract insights more quickly and enhance decision-making capabilities. They lay the groundwork for the most sophisticated data science, business intelligence, and AI applications, making sure that the data is organized, clean, and available for iterative analysis.

In this chapter, you will learn about the various roles and responsibilities of data engineers and how it works to support data science. This chapter will introduced the various tools used by data engineers as well as the different areas of the technology that will proficient in to become a data engineer.

This chapter will cover the following main topic:

- Data engineering lifecycle
- What Data engineers do
- Data engineering versus data science
- Data engineering tools

1.2 Data engineering lifecycle

The data engineering lifecycle is mainly divided into multiple stages as shown in fig. 1.1 ensuring there is organized flow of data from raw collection to final consumption. This life cycle is critical for building a scalable, reliable, and efficient data infrastructure that can power analytics, machine learning, and business intelligence. Each step of the data engineering lifecycle is explained in detail below.

1.2.1 Data Ingestion

Data ingestion is the first step of the data engineering lifecycle, being the process of extracting raw data from different sources and putting it into a central repository for further computation. These sources may

include databases, APIs, log files, IoT devices, and streaming services. Based on the use case we can ingest data in two ways a) Batch: In batch mode, data is collected for processing at scheduled times. b) Real Time Streaming: In Real time streaming data is ingested continuously and data is processed as it arrives. Data ingestion is the first phase in this process, where data is either sent over a network or added from a file to a database or data reservoir.

1.2.2 Data Storage

After ingesting data, it needs to be stored in suitable storage, where it can be scalable, secure, and easy to access. Hence the storage is decided based on the structure of the data and its use case. The main difference between data warehouse (Amazon Redshift or Google BigQuery) and data lakes (AWS S3 or Azure Data Lake) is that the main purpose of data warehouse is to store structured data only with analytics in mind, while data lakes can hold raw, structured, semi-structured and unstructured data. Also NoSQL databases such as MongoDB or Ajapche Cassandra are used for the storage of a high volume of unstructured data. One of the key elements to storing large datasets lies in how efficient the storage solution is in enabling quick retrieval and processing while ensuring the integrity of the data.

1.2.3 Data Processing & Transformation

It will take lot of time in preprocessing raw data, which can be messy and unstructured, before getting to look into it for analytics or machine learning. Data transformation and processing of various formats of data through the processes of cleaning, duplicates removal, normalization, aggregation, and enrichment. One common way to process data for analysis is a process called ETL (Extract, Transform, Load) or ELT (Extract, Load, Transform). Apache Spark and Hadoop are batch processing frameworks for large scale processing, while Apache Flink and Kafka Streams are for real-time processing (that is, continuous data transformation). This hashing so level checks and ensures that the data flowing through is accurate, consistent, and all of the correct format for downstream applications.

1.2.4 Data Orchestration & Workflow Management

The process of automating and managing data pipeline processes is known as data orchestration which guarantees the seamless functioning of the data workflows. This includes scheduling tasks, resolving dependencies, and fail handling. Workflow management tools such as Apache Airflow, Prefect and AWS Step Functions allow data engineers to define workflows, track their execution, and ensure that pipelines are executed in the right order. Reducing the manual effort involved, streamlining the entire pipeline pipeline, but also ensuring that data processing tasks are executed reliably and at the right moment.

1.2.5 Data Governance & Security

Data engineering has a significant component of data privacy, security, and compliance. Data governance includes enforcing policies for data access, line aging, and metadata management. Sensitive information is addressed through security measures such as encryption (for data both at rest and in transit), access control, and authentication mechanisms that limit access and prevent breaches. For organizations processing personal or sensitive data, adhering to regulations such as GDPR, CCPA, and HIPAA is a necessity. Good governance and security frameworks safeguard the misuse of data and advanced data remains trusted and compliant with the law.

1.2.6 Data Monitoring & Quality Management

To enable trusted insights and decision making data quality has to stay high. Data Monitoring and Quality Management—continuously tracking data accuracy, consistency, and completeness to identify anomalies or corruption. Tools such as Great Expectations and Monte Carlo aid in validating data

schema, tracking missing values, and finding inconsistencies in the data. Performance monitoring also guarantees that data pipelines work smoothly and efficiently, without any bottlenecks under preparation. This process ensures that enterprises have clean and high-quality data that they can trust in analytics and machine learning use cases.

1.2.7 Data Delivery & Consumption

Data Delivery and Consumption: This is the main step of data life cycle where the processed data is delivered to users for consumption. Data is delivered via business intelligence (BI) tools like Tableau and Power BI for reporting and visualization, APIs for real-time accessibility, or machine learning models for predictive analytics depending on the application. Create visual exploratory user interface: Instead of naked query results, use a visual exploratory technique (explsore tool)Visual datasets explorers like Presto, Trino or Google BigQuery It ensures that meaningful insights can be derived and data-driven strategies can be executed by the stakeholders.- data analyst, scientist, executives, etc.

These steps essentially form the data engineering lifecycle and are interrelated, providing a seamless, secure flow of data from collection to actionable insight.

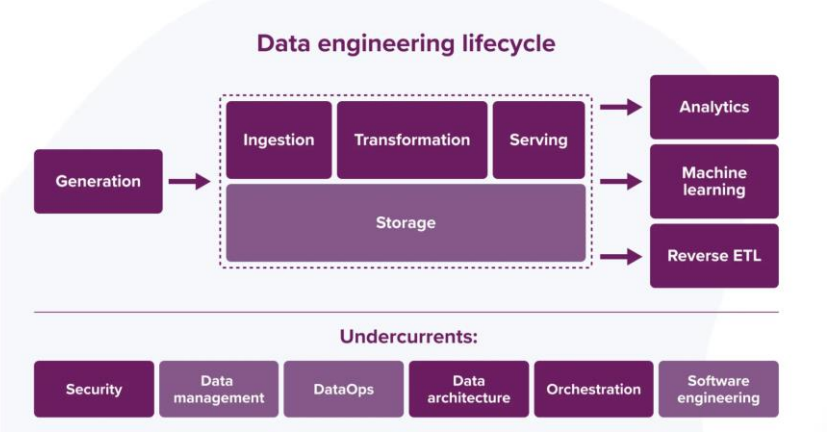


Fig. 1.1: Data engineering lifecycle

1.3 Roles and responsibilities of Data Engineers

Following are some roles and responsibilities of data engineer:

1. **Data Pipeline Development:** Data engineers create ETL (Extract, Transform, Load) pipelines that transport data from multiple sources (databases, APIs, log files, IoT devices) into a unified storage solution such as a data warehouse, data lake, or cloud storage. They help ensure that data is extracted, transformed (cleaned, formatted, and structured), and loaded correctly before its processing. Depending on the business needs, these pipelines could be batch-based, i.e., scheduled processing or real-time, i.e., continuous data streaming.
2. **Data Storage & Management:** Data storage and handling big datasets are the keys for efficient maximum reproducibility. Again, in terms of data storage, data engineers work with relational databases (PostgreSQL, MySQL, SQL Server), NoSQL databases (such as MongoDB, Cassandra), and cloud-based data solutions (AWS S3, Google BigQuery, Azure Data Lake), which store structured and unstructured data. They craft schemas, tune indexing, shard data, optimize query efficiency, lowering latency and improving performance for analytic and machine-learning applications.
3. **Processing & Optimization in Big Data:** Big data frameworks like Apache Spark, Hadoop, and Kafka are also required for dealing with large-scale data. Data engineers use distributed computing

techniques to process large data sets efficiently. By using parallel processing, caching mechanisms and optimized file formats (Parquet, Avro, ORC), you speed up data workflows and save the cost of processing and storage. They also make sure data pipelines can scale as the amount of data increases.

4. **Cloud Computing & Infrastructure Management:** As cloud-based data platforms take over, data engineers are building scalable, cost-efficient data solutions by using services such as AWS (Redshift, Lambda, Glue), Azure (Synapse, Data Factory), and Google Cloud (BigQuery, Dataflow). You configure auto-scaling, set up serverless architectures, and integrate with cloud-based tools to ensure the availability, reliability, security, and disaster recovery of data pipelines.
5. **Data Governance, Security & Compliance:** One of their key responsibilities is to ensure the security of data and compliance with regulations. Data engineers construct encryption, access control mechanisms (IAM, role-based access control), and audit logging to prevent sensitive information. They also enforce data anonymization, masking, and retention policies to ensure compliance with GDPR, HIPAA, CCPA, and SOC 2. They also have features for data lineage and metadata management for tracking movement and changes to the data.
6. **Working Together With Data Science & Business Teams:** Data engineers collaborate with data scientists, analysts, and business stakeholders to deliver clean, structured, and high-quality data for analytics, reporting, and machine learning models. They make sure the data is in the correct format, pre-processed in a timely manner, and stored so that it can be queried and retrieved quickly. They work together to provide businesses with valuable insights, enhance decision-making, and create predictive models.

1.4 Data engineering versus Data science

Table 1: Difference between data engineering and data science

Data Engineering	Data Science
It primary focus on building and maintaining data infrastructure, pipelines and storage system.	It primary focus on analysing data, applying machine learning, and generating insights for decision-making.
It works with raw, semi-structured, and structured data, transforming it into usable formats.	It works with cleaned, structured data to apply AI, analytics, and visualization techniques.
Strong expertise in SQL, Python, Scala, Java, and shell scripting.	Proficient in python, R, SQL and AI-related libraries
It uses distributed computing frameworks like Hadoop, spark, and kafka for managing large-scale data.	It works on big data analytics, but usually on processed data provided by engineers.
It prepares and optimized data for machine learning by ensuring quality and scalability.	It implements machine learning models, deep learning algorithms, and AI solution.
It designs and manages cloud-based storage and processing solutions using AWS, Azure, and Google cloud.	It uses cloud-based ML and AI tools for model deployment and data analytics.
It implements data security, encryption, role-based access control (RBAC), GDPR, HIPAA compliance.	It delivers predictive models, insights, reports, and AI-powered solutions.

1.5 Data engineering Tools

Focusing on this aspect, data engineers use different tools to extract, transform, load, store, administer, and secure data. These tools are classified according to the role they play in the data pipeline lifecycle:

1.5.1 Data Ingestion tools: They aid in collecting raw data from a large number of sources — databases, IoT devices, logs, APIs, or even streaming platforms.

a. **Apache Kafka:** A distributed real-time event streaming platform that takes high-velocity data from applications, sensors, and logs, allowing businesses to process and analyze it in real-time.

b. **Apache flume**: High-throughput solution for log data collection, aggregation, and centralized log storage in big-data environments.AWS Kinesis A cloud real-time data ingestion tool for collecting any data stream and preparing it for analysis AI and machine learning.

c. **Google Cloud Sub:** Messaging service that allows for real-time event-driven architectures, enabling asynchronous data transfer across disparate systems.

1.5.2 ETL (Extract Transform Load) & Data Pipeline Tools: These tools assist data movement downstream of the pipeline along with basic cleaning and transformation, as well as enrichment tasks.

a. **Apache Airflow**: A modular ETL orchestration engine that lets engineers define, monitor, and schedule ETL pipelines using directed acyclic graphs (DAGs)

b. **Luigi:** An ETL tool written in Python that makes it easy to build complex pipelines and makes sure they run only when data dependencies are satisfied.

c. **Apache NIFI:** A data flow automation tool designed to automate the flow of data between different systems.

d. **Talend**: An easy-to-use, low-code ETL tool for data integration and transformation. It is extensively used for migration of data, governance, and quality.

e. **Informatica Power Center**: A powerful ETL tool for enterprise data integration, that organizations utilize to extract, transform, and load large data volumes with great efficiency very often.

1.5.3 Tools for Storing and Managing Data: It assists in storing and managing structured, semi-structured, and unstructured data in an efficient way.

a. **Amazon S3**: A cloud-based object storage that is highly scalable and can be used for data lakes, backups, and analytics workloads.

b. **Google BigQuery**: Serverless data warehouse enables very rapid SQL queries for any amount of data without managing any infrastructure.

c. **Snowflake:** A high-level multi-cloud data warehouse service with serverless and scalable on-demand resources for analytics and machine learning.

d. **Azure Data Lake:** It is a cloud-based storage service built for big data applications, offering superior parallel processing capabilities.

e. **PostgreSQL / MySQL / Microsoft SQL Server:** Relational Databases used to store your data in a structured manner with the ability to query and index your structured data using SQL.

1.5.4 Big Data Processing Tools: They operate on batch and real-time data in distributed computing environments.

a. **Apache Spark**: A fast, distributed computing framework that speeds up big data analytics, streaming and machine learning on big datasets.
b. **Apache Hadoop**: A framework that allows for distributed storage and parallel processing across clusters of computers, making it perfect for the big data application.
c. **Presto:** A distributed SQL query engine designed for low-latency queries on large datasets stored in data lakes and warehouses
d. **Dask:** A flexible parallel computing library for analytics, that enables the implementation of parallelism in Python code.

1.5.5 Work Automation & Data Orchestration Tools: They automate, monitor, and schedule complex data workflows, making sure they send right task at right time.

a. **Apache Airflow**: A Task Scheduler that designs the workflows with DAG (Directed Acyclic Graphs) and schedules and monitors the pipelines.
b. **Prefect**: A data pipeline orchestration tool that provides a modern approach to making workflows reliable, offers fault tolerance and integration with the cloud infrastructure.
c. **Dagster:** An open-source data orchestrator for running modular, testable, and scalable data pipelines

1.5.6 Data Integration & API Tools: Such tools allow for easy connectivity to several data sources along with applications.

a. **dbt (Data Build Tool)**: A modern open-source transformation workflow that enables engineers to structure their own data from raw cloud warehouse models in a matter of minutes.
b. **MuleSoft:** One of the most widely used integration platforms that is commonly used for API management and to connect various enterprise applications in the organization.
c. **Apache Camel**: A lightweight integration framework that allows developers to send and transform data between systems using built-in connectors.

CHAPTER 2

DATA SOURCES AND TYPES OF DATA

Navjot Kaur Basra[1], Davinder Singh[2] and Gagandeep Singh[3]

[1,3]GNA University, Phagwara

[2]LKCTC, Jalandhar

2.1 Types of data

Data is produced from various sources, it can be in different formats based on its format, type and usage. It is important to know these types of data so that we can effectively store, process and analyze it in data engineering. There are four major data types and those are text, video, audio, and biological data.

2.1.1 Text Data

One of the forms of data that everyone uses is text data, which are structured, semi-structured, and unstructured textual data. It could be from documents, emails, social media, website, books, chat logs, etc. Text data is utilized by enterprises and organizations for natural language processing (NLP), sentiment analysis, document classification, and search engine optimization (SEO).

Various Sources: Websites, emails, news articles, social media posts, interaction with chatbots, database records.

2.1.2 Video Data

Video data is a well three-dimensional structured data that is in great demand in numerous industries, e.g. entertainment, surveillance, social media, health care, and autonomous systems. It is recorded via cameras, drones, cellphones and streaming services. Use of Video analytics in Object Detection, Facial Recognition, Motion Tracking, Content Recommendation

Various Sources: CCTV surveillance, YouTube, social media (TikTok, Instagram Reels), video conference platforms (Zoom, Ms Teams).

2.1.3 Audio Data

It uses sound chunks, music, environmental noises, and various other sounds. It is applied in areas such as speech recognition, voice assistants, music streaming, and sound classification. Moreover, with the evolution in AI, audio data is playing an essential role in voice-based authentication, virtual assistance, and live speech translation.

Various sources: podcasts, call center recordings, music streaming services (Spotify, Apple Music), and voice assistants (Alexa, Google Assistant, Siri).

2.1.4 Biological Data

Biological data is information derived from biological systems, the study of organisms and genetics. Highly utilized in sectors like biotechnology, healthcare, pharmaceuticals, and genomics research. Advancements in bioinformatics led to the analysis of large-scale biological datasets ranging from predicting diseases, drug discovery, and personalised medicine.

Various Sources: DNA sequencing machines, EHRs (Electronic Health Record Systems), medical imaging (MRI, X-rays), wearables (Fitbit, Apple Watch)

2.2 Other type of data:

2.2.1 Structured data:

Structured data is highly organized & stored in a predefined structure. This means it is organized in tables with columns and rows, with each column assigned a data type. Structured data is easy to search, retrieve and analyze using SQL queries, as it follows a fixed schema. Such data is generally utilized in banking, retail, healthcare, and enterprise-grade applications that need to handle transactional and operational data in an efficient manner.

For Example: If bank exchange store customer data, it will be in a structured manner in field like Customer ID, Customer Name, Account Number, Balance, Transaction History etc. The data is stored in an SQL database like MySQL or PostgreSQL, enabling rapid data retrieval for financial reporting and fraud detection.

2.2.2 Semi Structured data:

Semi-structured data does not have to be tabular but it does have tags or markers or metadata that organizes elements. The structuring is not strict, or has no schema as is the case with structured data. It follows some sort of organization and is more flexible and scalable. One common type of data that requires dynamic processing is commonly found in APIs, web data, emails, and log files.

For example, JSON (JavaScript Object Notation) files for web applications saves user data in a semi-structured format. For example, an e-commerce user JSON object might have some properties like:

```
{
“user_id”: 01,
“name”: “deep”,
“purchase_history”: [
{“product”: “Laptop”, “price”: 1200, “date”: “2025-02-10”},
{“product”: “Smartphone”, “price”: 800, “date”: “2025-01-04”},
]
}
```

2.2.3 Unstructured data:

Unstructured data, does not adhere to any particular model or structure, making it challenging to store and process using traditional database systems. Such data involves unstructured data such as text documents, images, videos, audio recordings, social media content, and demands high-level AI, NLP and big data skills. In fields such as media, healthcare, cybersecurity, and artificial intelligence, it is critical.

For example: Medical imaging in healthcare diagnostics produces unstructured data like X–rays, MRI scans, CT scans, etc. These images generate the valuable patient data, but need to be scanned by deep learning and image recognition models to detect illnesses. TensorFlow and OpenCV are traditional tools that process this data to detect anomalies in medical images and support them in diagnosis.

CHAPTER 3

DIFFERENT TYPES OF FILE FORMAT

Shruti[1], Neharika Sharma[2] and Mamta Bansal[3]

[1,2,3]GNA University, Phagwara

3.1 Overview

From a data science perspective file formats are the ways in which data is stored, structured and encoded, which is essential during the stages of processing, analyzing and sharing data. Different file formats are there to make the storage more efficient, faster retrieval, platform and application inter-compatibility and computational efficiency. Based on the structure of data in these formats, they can be grouped as follows: structured formats (eg− CSV, Excel) that hold data in tabular form to facilitate access to the data, semi-structured formats (like JSON, XML) which embed metadata to capture hierarchy, and unstructured formats (images, videos, text files, etc) that need to be processed using specialized techniques.

Big data formats such as Parquet, Avro, and ORC are optimized for large data processing in distributed environments like Hadoop and Apache Spark, allowing for efficient compression and parallel processing. The choice of a file format is critical in data engineering, machine learning and cloud computing workflows as it depends on considerations like data size, processing time, compatibility, intended purpose, etc. Choosing the right file format, be it Avro, ORC, Parquet, or another, to store our raw data, helps for efficient storage, optimal analytical tool integration and performance optimization in dealing with structured, semi-structured, and unstructured data.

3.2 Common File formats

3.2.1 CSV (Common Separated Values):

Common Separated Values (CSV) is a text file format for separating data by comma. It is a very popular format due to its simplicity, compatibility, and human-readable format. But CSV files do not contain information about metadata (like data types and relationships), and therefore CSV files can become inefficient for large-scale processing. CSV is often used for exporting data sets from SQL databases, as well as spreadsheets (Excel), and machine learning data sets (from Kaggle & UCI repository).

Advantages:

1. CSV files are easy to create and read using various text editors and spreadsheet software.
2. Supports nearly every programming languages and databases.
3. Ideal for small to medium-sized data sets.

Disadvantages:

1. It does not implement data types, relationships, or compression, so it doesn't scale well while dealing with too much data
2. Does not work well with nested or hierarchical data.

Python code to read a csv file in pandas:

```
import pandas as pd
df= pd.read_csv("file_path / file_name.csv")
print(df)
```

3.2.2 Excel(XLSX):

XLSX files are Excel files, which store structured information across multiple sheets, with formulas, charts, and formatting. It is commonly used in business analytics, finance, and reporting. Excel files, on the other hand, support data validation, pivot tables, and macros, allowing for more robust operations on small datasets than you get with CSV.

Advantages:

1. Leverages formulas, conditional formatting, and multiple sheets.
2. Use graphical visualization tools such as charts and pivot tables.

Disadvantages:

1. Not great for mass processing or automation.
2. Requires Excel or some library like pandas and openpyxl for Python-based manipulation

Python code to read xlsx file in pandas:

```
import pandas as pd
df= pd.read_excel ('file_path\\name.xlsx')
print(df)
```

3.2.3 JSON (JavaScript Object Notation)

JSON (JavaScript Object Notation) is a lightweight data interchange format. And because it stores data in key-value pairs and nested structures, it can be parsed easily in languages like Python and JavaScript. Web applications use JSON to exchange data between front-end and back-end systems (e.g.,user profiles, shopping carts).

Advantages of JSON:

1. Easy to read and parse in python, JavaScript, and many other languages.
2. It is flexible for Hierarchical and Nested Data.
3. Used in Rest APIs, MongoDB, cloud storage etc.

Disadvantages of JSON:

1. It takes more space then binary.
2. It doesn't support binary data or complex types natively.
3. It requires additional tools for efficient storage.

Python code to read json file in pandas:

```
import pandas as pd
df = pd.read_json('File_path \\ File_Name.json')
print(df)
```

3.2.4 ZIP

It is commonly used to group files and folders into one file to save space. It performs lossless compression which means that during compression the data is not lost. ZIP is often used to store and back up files and for faster data transfer. It is commonly used for zipping big datasets before distribution which helps to save disk space.

Advantages of ZIP:

1. It reduce the size of files so it helps to free up the disk space.
2. It helps to store multiple files and directories in a single archive.
3. It can be encrypted and password protected which results in more security.

Disadvantages of ZIP:

1. Certain file types don't compress well (e.g., already-compressed files like JPEG).
2. Files need to be extracted before they are usable.
3. ZIP file corruption can be tricky.

Python code to read zip file in pandas:

```
import pandas as pd
    df = pd.read_csv('file_path \\ file_name.zip')
    print(df)
```

3.2.5 PDF(Portable Document Format):

PDF stands for Portable Document Format, a fixed-layout document format created by Adobe that is used to present documents in a manner independent of application software, hardware, and operating systems. It can include text, images, hyperlinks, forms, and multimedia content. It is mainly used for official reports, contracts, legal documents, research papers, e-books and manuals.

Advantages:

1. It has consistent formatting or have same appearance or any device or operating system.
2. It is more secure as it is password protection, can be encrypted and digital signature can also be include.
3. All interactive elements including forms, annotations, hyperlinks and multimedia content.
4. It support OCR (Optical Character Recognition).

Disadvantages:

1. It is difficult to edit without any specialized software.
2. PDFs containing images and other multimedia are of large size which require more space.
3. It is not ideal for storing raw data or machine-readable text.

3.2.6 HTML (Hyper Text Markup Language):

HTML stands for Hyper text markup language and it is a markup language for web pages. HTML is used to describe the structure and content of a web page with the help of elements and tags. It operates with CSS for styling and JavaScript for interactivity.

Advantages:

1. It works on all web browsers.
2. It is text based format so it loads quickly.
3. It support multimedia files also.

Disadvantages:

1. HTML will be interpreted differently by different browsers.
2. It require additional technologies like CSS and JavaScript for interactivity.
3. It can vulnerable to cross-site scripting attacks if it is not handled properly.

Python code to read html file in pandas:

```
import pandas as pd
a= pd.read_html ('file_path\\name.html')
print (a)
```

CHAPTER 4

DATA REPOSITORIES, PIPELINES, AND INTEGRATION PLATFORMS

Simran[1], Jeevanjot Singh Chagger[2] and Rajesh Sharma[3]

[1,3]GNA University, Phagwara

[2]Sant Baba Bhag Singh University, Jalandhar

4.1 Overview

A data repository is a centralized source of data that stores data to be retrieved, managed, organized, and maintained in a structured, semi-structured, or unstructured format. It acts as a data lake in which enterprises can hold large quantities of data for analysis, retrieval, sharing and processing. Because of this, data repositories are vital for data engineering, analytics, machine learning, and business intelligence, as they provide efficient data management, security, and accessibility.

It enhanced data governance. A structured data repository helps in maintaining data integrity and consistency. They also allow for scalability, enabling organizations to work with large datasets, facilitating big data processing. There are different types of data repositories, i.e., data warehouse, data lake, databases, metadata repository and data mart based on the purpose and nature of data.

Data repositories enable data storage and processing in modern applications, lying at the core of every decision-making, research, and AI-powered solution, though the need for them has escalated exponentially over time. They follow industry best practices that make data easily accessible, optimize the performance of the data pipelines and allow easy integration with analytical or machine learning models. Data Warehouses and Data Lakes serve different purposes in the modern data ecosystem and must be chosen according to considerations such as data structure, scalability, security requirements, and analytical needs.

4.2 Types of data repositories:

4.2.1 Data Warehouse

A data warehouse is a centralized data management system that collects and stores a lot of pre-processed and structured data from disparate sources. A data warehouse is an architecture (as shown in fig.4.1) specifically designed for business intelligence (BI), reporting, and analytics, in contrast to transactional databases that are focused on day-to-day operations. It shows organizations how historical information can be analysed to discover trends and make business decisions.

Features Of data warehouse:

a. Subject-Oriented

Unlike transaction data, a data warehouse is organized by business domains like sales, finance and customer service. This allows organizations to analyze trends, create reports, and make data-driven decisions more easily without the complexity of operational data.

Example: A retail firm might have separate systems for online orders, in-store purchases, and customer loyalty programs. A data warehouse combines all sales data, making it easier to analyze customer buying patterns across multiple channels.

b. Integrated

A data warehouse collects data from multiple sources, such as databases, customer relationship management (CRM), enterprise resource planning (ERP) systems, and external sources. This process helps to ensure the data is consistent, accurate, and standardized before entering the storage.

For example: A multinational corporation may use different database systems (SQL Server, Oracle, MySQL) in different regions. A data warehouse aggregates data from all of these, giving you a single view of the company's performance.

c. Time-Variant

A data warehouse stores historical data over the long term, enabling organizations to follow changes, analyze patterns, and make strategic decisions based on previous performance. Unlike operational databases that only keep records of the most current entries, data warehouses retain older data, even if the original information has been updated or deleted.

Example: A bank can examine customer loans paid in the past decade and use data thereby stored in the warehouse to create risk factors and defaults potential.

d. Non-Volatile

The data is not modified or deleted once it is loaded into a data warehouse. It ensures the stability and consistency of data for reporting and analysis. Warehouse is designed for reading and querying the data and not for frequent updating of records as is the case with transactional databases.

Example: Patient medical records in healthcare data warehouses, which shouldn't change to maintain an accurate history for doctors and researchers.

e. Optimized for Analytics

A data warehouse is designed to provide strategic insights and analytics, rather than processing transactions. Indexes, partitioning and aggregations included for optimizing the query performance for users to gain insights in short time.

Example: If the same financial institution extracts stock market data and run complex queries to detect trends, anticipate future stock prices, and make investment recommendations.

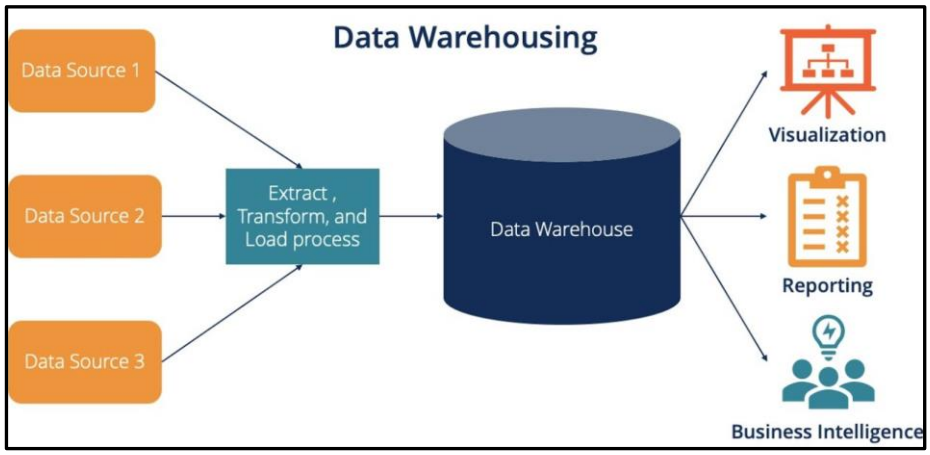


Fig. 4.1: Data Warehousing

Key features:

1. **Centralized Storage**

The data repository is a consolidated storage of data from multiple different sources. As opposed to the scattered data across different systems, it organizes all information in one place thereby

significantly enhancing data accessibility, consistency and organization. Reduce data fragmentation and simplify management

Example: A global company aggregates customer data from various countries in a central data warehouse to gain insights into global sales trends.

2. Scalability

Data scales are evolving with an increase in the volume of the data. Cloud-based solutions for building data repositories are common nowadays, owing to scalable storage available as per the need and does not involve any hardware purchase for storage expansion. This is even more critical when it comes to big data applications.

Example: An e-commerce platform expands its data warehouse to accommodate more customer transaction data during the holidays.

3. Data Security and Access Control

Security The security is the one of the most important to keep in mind in data management. They. Data repositories implement access control mechanisms, authentication protocols, encryption techniques, and audit logs to safeguard sensitive data from unauthorized access, breaches, and cyber-attacks.

Example: Patient records can be adequately stored by the healthcare provider in an encrypted database that protects the data from being accessed by unauthorized medical personnel and at the same time, remain compliant with HIPAA regulations.

4. Data Integration & Interoperability

The data warehouse should connect to various data sources like databases, cloud platforms, APIs, IoT (Internet of Things) devices, and third-party applications. This allows for smooth data interchange and avoids data silos, letting organizations integrate multiple datasets to achieve deeper insights.

Example: A financial institution brings together stock market data, customer portfolios, and transaction history into a single repository for predictive analysis.

Advantages:

1. It helps to improve decision making as it is reliable source for business insights.
2. It combines multiple data sources into a unified format.
3. Data warehouse stores historical data for long-term data analysis.
4. Data Warehouse can manage large data sets and scale as the business requirements scale.
5. It offers access control and data protection mechanisms.

Disadvantages:

1. The implementation cost of data warehouse is high.
2. It is only suited or batch processing not for real-time updates.
3. It is hard to adjust to new data sources or schema evolution.
4. The configuration in data warehouse should be effective and it requires frequent updates.

4.2.2 Data marts

A data mart is a smaller, subject-oriented data warehouse that is designed to serve a particular business function or department as shown in fig.4.2. Companies use this to answer specific reporting or analytical requirements and it offers an aggregated view of data in a limited way. Data marts: Smaller than data

warehouses, these hold relevant data for a highly specific group of users, for example, sales, marketing, or finance. They are domain-specific, they are scoped to an area, which might be sales, customer data, product information, etc. The data will be structured, transformed and optimized for querying and analysis in the domain.

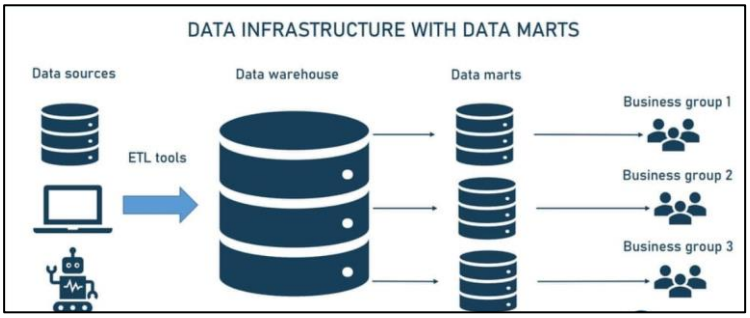


Fig. 4.2: Data Marts

Types of Data Marts:

1. Independent Data Mart; 2) Dependent Data Mart; 3) Hybrid Data Mart

1. Independent Data Mart:

The data mart is a kind of independent data mart created and maintained independently of the data warehouse as shown in fig.4.3. It is designed for the unique requirements of an individual business unit or department. Independent data marts are often smaller and much more quickly and easily deployed. Really, you were not limited by the constraints of a centralized data warehouse. However, if it is replicated across multiple data marts, this could result in data redundancy and inconsistency.

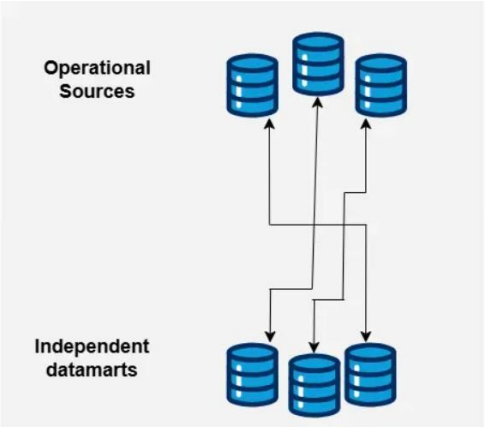


Fig. 4.3: Independent Data Mart

2. Dependent Data Mart:

A dependent data mart is created directly from a data warehouse. It extracts a portion of the data from the data warehouse and reformats it to cater to the requirements of a particular industry as shown in fig.4.4. They benefit from the data provisioning on the data, data quality and data consistency ensured by the data warehouse, and enable organization-wide data consistency through the establishment of a single source of truth for all of the data. They are typically created for specific reporting and analysis needs and often refresh from the data warehouse. Dependent data marts take advantage of the data warehouse as their primary source of data, ensuring data consistency and preventing data duplication.

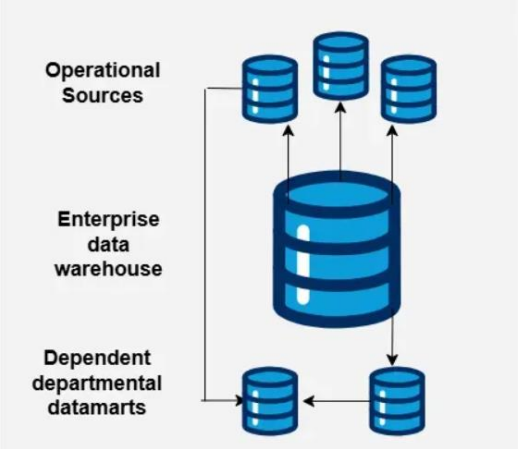


Fig. 4.4: Dependent Data Mart

3. **Hybrid Data Mart:**

A hybrid data mart includes both independent and dependent data mart components. In addition to integrating and standardizing the central data, it also integrates the supplementary data sets specific to the individual business unit or department as shown in fig.4.5. Hybrid data marts provide the advantages of both strategies because they combine flexibility and agility for department-specific requirements with maintenance of both integrity and uniformity of commonly used data from the data warehouse. This strategy balances localized data management with a centralized data management.

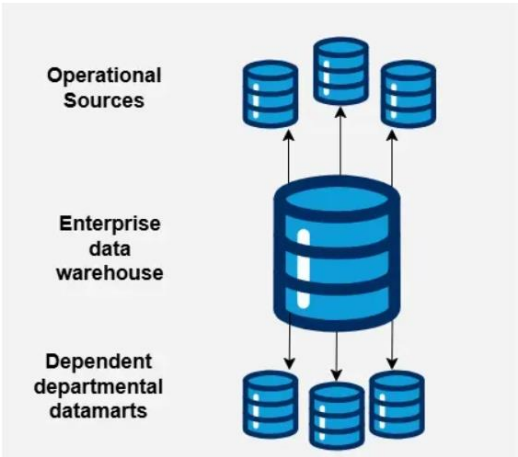


Fig. 4.5: Hybrid Data Mart

Structure of Data Mart:

1. **Star**

Star schema is a common data mart structure based on the dimensional model. It consists of multiple dimension tables around one core fact table. The fact table contains numerical data or metrics about a particular business process or subject matter, such as sales or inventory. Dimensional tables provide contextual or descriptive information about the data stored in the fact table. Normally each dimension table represents a certain attribute or facet of the data, such as time, location, products, or customers. The fact table and dimension tables are linked via primary-key and foreign-key associations forming a star-like structure. In this format, users can easily slice and dice data across multiple dimensions which enables good querying and analysis.

2. Snowflake

The snowflake model is a dimensional model extension that provides a more normalized data structure. The structure further normalizes dimension tables by splitting them into a number of related tables. This normalization will remove data redundancy in case of complex hierarchies or when a dimension has lots of properties. The snowflake model, on the other hand, can complicate search and data integration processes.

Advantages:

1. Data marts are built for particular business units to retrieve data quickly.
2. It is cheap as compared to data warehouse.
3. It is quick and easy to implement.
4. It reduce security risks as data access is limited to specific teams.

Disadvantages:

1. It can result in siloed data sources that makes it hard to analyse data enterprise-wide.
2. Storage costs rise as data is repeated in various data marts.
3. A specific business function is served, therefore missing a holistic view at the company level.
4. Merging multiple data marts into a unified system can be quite complex.
5. Must be regularly updated and monitored to ensure relevance and functionality.

4.3 Difference between data warehouse and data marts:

Table 2: Difference between data warehouse and data marts

Data warehouse	**Data mart**
It is a central repository that stores large volumes of structured data from multiple sources for the entire organizations.	It is a subset of a data warehouse focused on a specific department or business unit.
It covers all business functions.	It is team-specific that tailored to a particular function.
It collects data from multiple internal and external sources.	It usually extract data from a data warehouse or a few selected sources.
It is large in size as it store historical and current data	It is smaller in size as it contains only relevant data for a department.
Implementation time for data warehouse is long as it requires significant planning and resources.	Implementation time for data marts are shorter then data warehouse as it focuses on limited data and users.
It is expensive to build and maintain due to storage, processing and infrastructure needs.	It is cost-effective since it covers a smaller dataset.
It is more flexible but requires structured data and predefined schemas.	It is more rigid as it is designed for specific use cases.
It requires complex security and access controls for multiple users.	It is easier to manage security since access is limited to specific departments.

4.4 Data lakes

A Data Lake is a repository, centralised storage point, that retains a huge volume of data in its native, raw state. Unlike the hierarchical data warehouse that stores data in files or folders, the data lake has a flat architecture and object storage to store the data. Object storage stores the data in such a way that it is tagged with accompanying metadata and assigned a unique identifier, making it easier to locate the relevant data in a region or retrieve data by using the given identifier — thereby improving performance as shown in fig.4.6. Data lakes take advantage of cheap object storage and open formats so that many applications can use the data.

Data Lakes evolved to overcome the challenges with Data Warehouses. Although data warehouses deliver high performance, scalable analytics to businesses, they are expensive, proprietary, and do not handle any of the modern use cases that most businesses are looking to solve for. Data lakes are typically used to centralize all of an organization's data in one place, where it can be stored "as is," without needing to place a schema (i.e., a formal structure for how the data is organized) in place in advance as a data warehouse does. Raw data can be ingested and stored in a data lake, next to an organization's structured, tabular data sources (like database tables) as well as intermediate data tables that are generated during the refining of raw data. Unlike typical databases and data warehouses, data lakes can handle all types of data — including unstructured and semi-structured data, such as images, video, audio and documents — that are vital for today's machine learning and advanced analytics use cases.

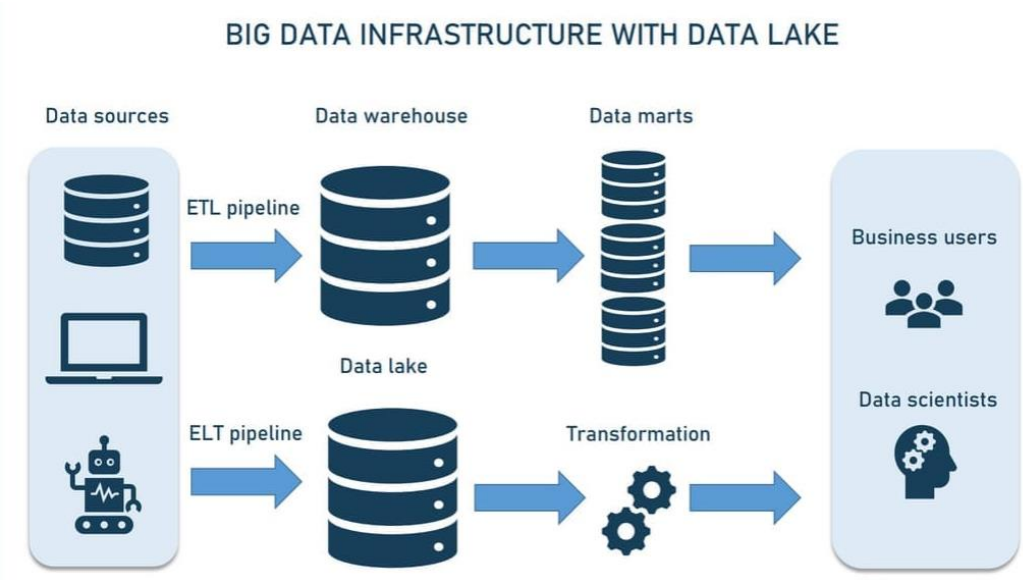


Fig.4.6: Data lake infrastructure

4.5 Data pipelines:

Data Pipelines automate the end-to-end flow of data, unlike manual data handling which are prone to human errors. They can do this through real-time (streaming) or batch data processing as per the needs of the business. Modern data pipelines ingest data of all types — structured, semi-structured, and unstructured — from a diverse set of sources, including databases, APIs, IoT sensors, log files, social media, and cloud services. A data pipeline is a series of data processing steps. It can be used for business intelligence, analytics, and machine learning applications and helps organizations scale to collect, process, and store data.

A modern data pipeline contains multiple consecutive phases like data ingestion, transformation, validation, storage, and monitoring, etc. By providing a unified view of enterprise data across disparate sources, data integration simplifies the process of combining data for analysis.

4.5.1 Data pipeline hierarchy:

The hierarchy clarifies the distinct stages that data traverses, ending in informed decision-making. Data pipeline staircase diagram data flows. In every step the previous one is the cornerstone of data raw input that gives valuable insights. Here are further details on every step in the hierarchy (as shown in fig.4.7):

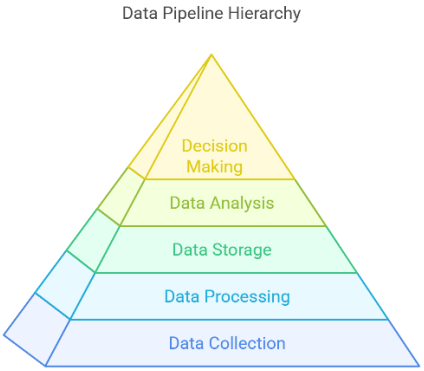


Fig. 4.7: Data Pipeline Hierarchy

a. Data Collection

Data collection forms the basis of the data pipeline where raw data is collected from multiple sources like databases, APIs, log files, IoT (Internet of things) sensors, social media, and web scraping. This data can be in a structured, semi-structured, or unstructured format, and it can be gathered in real-time or batch. At this stage, ensuring the data is complete, accurate, and reliable is critical, as mistakes in the collection of data can affect the entire pipeline. Collecting data usually leads to duplicates, missing data and inconsistent data which need to be resolved in the later stages.

b. Data Processing

The collected data is processed (cleaned and transformed) into a structured format, ready for further analysis. Data cleaning (duplicate removal, missing value management), transformation (normalization, aggregation), and integration (combining multiple datasets) are all part of data processing. Honing and formatting raw data for a streamlined storage and retrieval process is the next step in the extract-load-transform pipeline. Depending on the requirements of the business, the latency in processing data is determined, with real-time processing for time-sensitive use cases and batch processing for larger data sets.

c. Data Storage

The cleaned data is then stored into be suitable storage systems like databases, data warehousing, data lake, or cloud storage solutions after the processing. Different type of data serve different purpose hence different storage would be used. Structured data is stored in databases (like SQL, NoSQL), Analytical data is loaded into data warehouses for business intelligence, and Data lakes for large volumes of raw and unstructured data. Sound data governance, security, and backup processes need to be put in place to maintain the integrity of data and ease access for analysis.

d. Data Analysis

Now, the stored data is evaluated to come up with useful conclusions. This includes descriptive analytics (providing a summary of past data), diagnostic analytics (discovering patterns and correlations), predictive analytics (using machine learning to forecast trends) and prescriptive analytics (using insights to suggest actions). Data is processed and visualised using tools such as SQL, Python, R, Power BI and Tableau. Monitoring data allows companies to understand consumer behaviour and market patterns leading to operational efficiencies and better business decisions.

e. Decision Making

The analysis helps organizations make data-driven business decisions, which is the top level of the data pipeline hierarchy. For example, decision-making can be manual (business leaders interpreting reports) or automated (AI-driven decision-making systems). They assist in strategy planning, operational optimization, fraud detection, personalized marketing, and process automation. By creating an efficient data pipeline, companies can open their doors to better-quality insights, maximizing utility for growth, differentiation, and well-being.

4.5.2 Types of data pipelines:

1. **ETL**
2. **ELT**

1. ETL(Extract transform load)

ETL is the batch-based data integration method and in ETL data is extracted from multiple systems, transformed into a structure, and then loaded into a data warehouse for analytical processing (as illustrated in fig.4.8) . This approach is widely applied within business intelligence (BI) and reporting, where having structured and cleaned data is crucial for decision making.

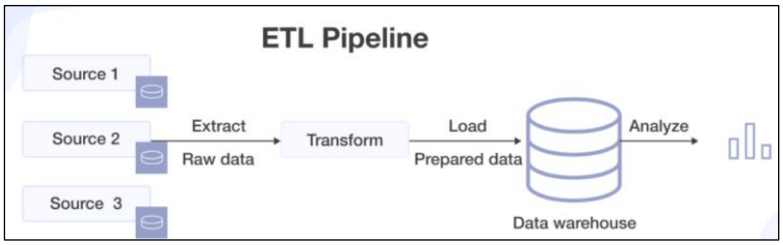


Fig. 4.8: ETL Pipeline

Steps in ETL Process:

Extract

The first step is to extract raw data from different sources such as databases, APIs, cloud storage, log files, and third-party applications. These come in the form of SQL databases as structured data, semi-structured data like JSON or XML files, and unstructured data that can be text documents and images. ETL is a structured process, so its focuses is on extracting structured or semi-structured data so that it is compatible with the data warehouse.

Transform

After extraction, the data is passed through various transformation processes until it is clean, structured and useful for analytical purposes. This includes, cleaning, standardization, normalization, aggregation, enrichment, etc. This process can involve a range of tasks such as filling missing values, removing duplicate records, converting data types, and correcting inconsistencies. In many cases , transforming the

data before loading it into the data warehouse makes sure that the data warehouse is well-formed and populated with high-quality data, which is ready for reporting and business intelligence.

Load

This is the last step where the transformed data is loaded to the data warehouse such as Amazon Redshift, Google BigQuery, Snowflake, Microsoft Synapse Analytics etc. The cleaned and structured nature of the data also allows it to be immediately queried, reported, and visualized using platforms like Tableau, Power BI, and Looker. Data loading is usually done in scheduled batches (daily, weekly, or monthly) in order to keep the warehouse up to date.

ELT:

ELT is a modernized data integration approach to extract data first, load it straight into a data lake or cloud data warehouse, and transform it in the desired schema as necessary(as shown in fig.4.9). In contrast to ETL, which transforms data before loading it into storage, ELT initially loads raw data and transforms it afterward. This approach is popular in big data processing, stream analytics, and machine learning (ML) workloads.

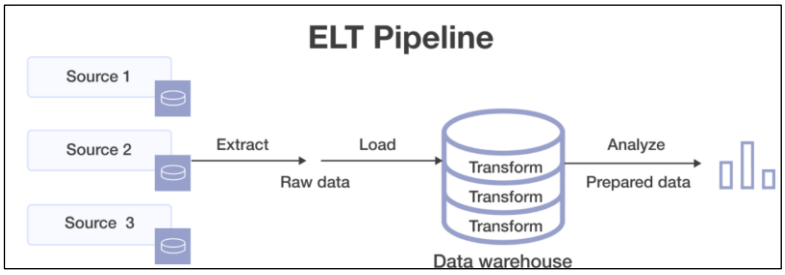


Fig. 4.9: ELT Pipeline

Steps in ELT Process:

Extract

Just like with ETL, the extraction part entails gathering data from various sources, including databases, APIs, logs, cloud services, and IoT (Internet of things) devices. In ELT, however, all raw data is extracted and stored without transformation so that organizations can subsequently analyze all available information, not just what has been transformed.

Load

Unlike ETL which transforms the data first, ELT loads all the raw, unprocessed data into a high-volume and scalable storage system like Amazon S3, Azure Data Lake, or Google Cloud Storage. As cloud storage is cheaper and targets high scalability, an organization can deposit big volumes of raw data without filtration and pre-processing, so ELT suits to big data and real-time projects very well.

Transform

Once it is stored, all transformations happen inside the storage system using cloud-based compute. Transformations are done on demand using SQL, Python, Apache Spark, or native cloud tools like AWS Glue and Google Dataflow. Only the needed data is queried and transformed when analysts must do so, making ELT more adaptable and scalable for complex analytics and AI-driven use cases.

4.6 Data integration platforms:

Data Integration platforms are software applications that drive the interconnection, amalgamation, and transformation of data from a number of sources into a common and well-organized format. These

platforms make them easy for data to get mobilized and ensure that the data from various databases, cloud services, API, and applications can be combined and processed well. Automating ETL/ELT processes allows organizations to streamline data workflows, increase data accessibility, improve decision-making, and create better enterprise-wide decision making process.

4.6.1 Essential Features of Data Integration Platforms

Data Connectivity

A good data integration platform should be able to connect to a variety of data sources such as relational databases (MySQL, PostgreSQL, SQL Server), cloud storage (AWS S3, Google Cloud Storage), NoSQL databases (MongoDB, Cassandra), APIs, and on-premise systems. This ability provides the functionality to extract and synchronize data between different environments without manual intervention, thereby allowing events to access real-time and historical data.

ETL & ELT Capabilities

ETL and ELT functionalities are offered by most integration platforms. ETL is a commonly used process in data warehousing, where data is transformed before being loaded into the target system to ensure data quality. With ELT, however, raw data is first loaded into a data warehouse or Data Lake and then the transformations are applied as needed. This makes data ready for analytics and reporting as organizations can efficiently work with structured, semi-structured, and unstructured data.

Real-time & Batch Processing

However, data integration platforms can work with both real-time data in motion and batch data at rest. For applications such as fraud detection, IoT(Internet of Things) analytics, and customer behaviour tracking, real-time processing is essential, as data should be processed immediately. Batch processing, however, is useful for regularly scheduled updates, such as daily sales summaries or monthly performance analysis. This provides businesses with the flexibility to pick and choose between the approaches as per their operational needs.

Cleaning and transforming the data

One of the critical functions of integration platforms is to ensure data accuracy and consistency. These platforms typically include capabilities for data cleansing, such as dealing with missing values, eliminating duplicates, standardizing formats, and rectifying inconsistencies. For cleaning it is important that data transformation operations like filtering, aggregating, normalizing, and enriching are applied sequentially so that the final dataset is formed, structured, and meaningful for analysis. Inaccurate data can result in unreliable insights and poor decision-making without suitable cleansing.

Scalability & Cloud Support

Data integration is key to improving such potential problems, but modern data integration platforms are extremely scalable, which means they can keep up with promoting increasing volumes of data to meet business demands. Cloud solutions consisting of AWS Glue, Google Dataflow, and Microsoft Azure Data Factory allow your data team to work on terabytes of rows and columns without worrying about the infrastructure limitations. Auto-scaling is also offered by these platforms, so businesses can sweep up data spikes without fearing too much about overspending.

Security & Compliance

Security is paramount, as data integration deals with sensitive business and customer information. These include data encryption, role-based access control (RBAC), audit logs, compliance management, etc

which safeguards data from unauthorised access and cyber threats. They also assist with compliance to industry regulations like GDPR, HIPAA, and CCPA, thus allowing businesses to maintain data integrity and avoid penalties.

Automation & Workflow Management

They automate complex data workflows with tools for scheduling and monitoring data pipelines. Automating these workflows ensures that data gets extracted, transformed and loaded without any intervention from the operations team, lowering the operational overhead. Error handling and monitoring dashboards are also provided whereby users will be alerted if the data pipeline fails, allowing for seamless operation and faster downtime.

4.6.2 Advantages of data integration platforms:

Improves Data Consistency

These tools bring data from multiple sources under one roof in a single, unified format eliminating inconsistencies and discrepancies resulting from manual data processing. This ensures that every business unit has access to the same version of data, supporting accurate reporting and reliable insights.

Enhances Decision-Making

And with instant access to real-time and historical data, organizations can dramatically accelerate data-driven decision-making. Underpin them is Data Integration platforms that help Business Intelligence tools to get clean, process data that helps companies find trends and improve operations as well as customer experiences.

Saves Time & Resources

The automation of the data extraction, transformation, and loading process saves time and effort that would have been spent on manual data processing. Prescriptive data movement means IT teams don't have to develop individual scripts anymore, freeing them up for higher level priorities. This also reduces the operational cost by removing duplicate processes and reducing errors.

Enables Scalability

An adaptable data integration platform can easily deal with increasing volumes of data without degrading performance. As such, cloud solutions provide elastic scalability where businesses can scale computing resources up or down based on demand. It allows organizations to scale up data without needing expensive infrastructure updates.

Compatible with Cloud & Hybrid Environments

Modern companies have hybrid environments where on-premise and cloud-based systems work. Data integration platforms act as a bridge between disparate data models within and beyond the organization, which allows smooth movement of data between different ecosystems, a needs in other to leverage cloud computing and when legacy systems are still in use.

4.6.3 Challenges of data integration platforms:

Complex Setup & Maintenance

It requires technical knowhow to deploy and configure a data integration platform. Firms must accurately map data flows, configure security policies and manipulate infrastructure, which can be difficult and time-consuming.

Data Security Risks

With data traversing several different systems, it is subject to unauthorized access, breaches, and cyber threats. Businesses need to use strong security features to secure private information, including data masking, encryption, and authentication.

High Costs

Building, maintaining, and updating enterprise-level data integration platforms can cost a fortune, as you may have to shell out for software licenses, rent cloud storage, and potentially hire highly skilled professionals. Cloud-based solutions provide the pay-as-you-go pricing, but when data volume and processing requirements increase, so do the costs.

Performance Bottlenecks

Integrating data from multiple sources is a resource-intensive process. If data is not processed optimally, slow data ingestion and latency issues can hinder business operations. To combat this, organizations need to ensure their platform serves parallel processing and distributed processing.

4.6.4 Popular data integration Platforms:

Table 3: Popular data integration platforms

Platform	Key Features	Best for
Talend	Open-source ETL, cloud & on-premise integration	General ETL and data governance
Informatica PowwerCenter	AI-driven data management, robust ETL capabilities.	Enterprise BI & Big Data
Apache Nifi	Real-time data streaming, automation workflows	IOT & real-time analytics
AWS Glue	Serverless ETL, integration with AWS ecosystem	Big data & AI workloads
Google cloud dataflow	Real-time & batch data processing, scalable pipelines.	Machine learning and artificial intelligence

CHAPTER 5

BIG PROCESSING TOOLS

Suruchi[1], Prerna Kutlehria[2] and Ramandeep Kaur[3]

[1,2,3]GNA University, Phagwara

5.1 Overview

Big Data is a term that can describe data, but it seems more appropriate to say that it describes a set of Big data, which are very huge data that cannot be effectively handled, processed and analyzed using the traditional database management tools due to their size, speed and diversity. This data is generated from various sources like social media, IoT devices, business transactions, health records, etc. In this digital era, Big Data matters as companies leverage it for meaningful insights, opportunities, to enhance operation efficiency, improve customer experience, and drive innovation.

5.2 Key characteristics of big data (3V's):

Volume (Size of Data)

Volume can be described as the large volume of data that is generated, collected, and stored in an instant. Huge datasets created by businesses, social media platforms, financial institutions, and IoT devices need scalable storage and processing power. Traditional databases are not designed to store large-scale data, so distributed storage options such as Hadoop Distributed File System (HDFS), cloud storage platforms (such as AWS S3, Google Cloud Storage), and data lakes have become popular. Companies need Big Data solutions to stop dealing with explicit sets and start analyzing large bunches of them for intelligent decision-making. A good example is Facebook, which handles 4 petabytes of data daily, while terabytes of transaction data from retail businesses are stored for market analysis.

Velocity (speed of the data processing)

Velocity is how fast data is generated and needs to be processed. As the number of real-time applications grows, organizations must process high-speed data streams for fast & accurate decision-making. Stock market transactions, for example, average millions of trades per second and need real-time analytics to identify market trends and thwart fraud. Likewise, in smart cities, IoT devices produce constant sensor data, which needs to be processed instantaneously for automated traffic management. Poor velocity management risks missed opportunity and poor information.

Variety

Variety translated to the different formats, sources and structures of data being generated. Where traditional databases have structured data, Big Data entails structured data, semi-structured data, and unstructured data from various sources. The data that is structured, such as relational databases and spreadsheets, is easy to process and store. But, semi-structured data like JSON, XML, and log files need particular tools for processing. Unstructured data such as images, videos, social media posts, and e-mails are much harder the process given they are not quite as straightforward in nature. The data extracted from multiple sources is then processed using NoSQL databases (MongoDB, Cassandra), AI-based analytics, and Natural Language Processing (NLP) models, enabling the business to obtain insights from various data formats. Medical system that processes structured patient electronic records, semi-structured medical reports, and unstructured MRI images together to provide accurate diagnosis and treatment.

5.3 Big data processing tools:

Big data processing tools are specialized software frameworks that enable non-technical users to access and manage vast volumes of structured, semi-structured, and unstructured data in an efficient manner. Big data cannot be managed using traditional databases that cannot store the size, velocity, and types of big data. These tools are built on the principles of distributed computing and parallel processing, breaking up larger tasks into smaller processes that can run faster and in a more scalable way. They assist organizations in deriving actionable insights from raw dataset, facilitating decision making, automation, predictive analytics, etc.

5.3.1 Important Characteristics of Big Data Processing Tools

Scalability

The tools for big data is aimed at workloads getting more and more without losing performance. They scale horizontally, allowing the addition of computing nodes to read and distribute the data processing load as needed. This aspect of scalability is critical for enterprises managing petabytes of data and ensuring that, as the size of these datasets increases, the system continues to respond and perform efficiently. In contrast to traditional databases that may struggle to perform under enormous loads, big data tools can scale their infrastructure dynamically according to need.

Fault Tolerance

Because big data systems are distributed across multiple machines, there will be failures. This means that these tools are equipped with fault-tolerant mechanisms that provide redundancy of data and automatic recovery. Capabilities such as data replication, check pointing, and self-healing clusters enable systems to operate uninterrupted even if some nodes go down. Hadoop's HDFS (Hadoop Distributed File System), for instance, copies the same data across servers, so if you lose a machine, the data lives on another server.

Real-Time & Batch Processing

There are two principal processing modes for big data tools:

Batch processing: where you periodically process large batches of data. This makes it perfect for analysis and reporting on historical data. Apache Hadoop MapReduce is an example, which applies batch processing over distributed clusters on data.

Real-Time Processing: Tools in real time process streams and each time new data comes in. As such, it is critical for applications like fraud detection, or stock market analysis, or IoT data processing, enabling businesses to monitor, analyze, and react to data in real time. Real-time data streaming itself is typically facilitated using enterprise tools such as Apache Kafka, and streaming frameworks such as Apache Flink.

Distributed Computing

To process big data you will use distributed architecture, where data is split into smaller pieces and processed in parallel among many servers. This greatly improves efficiency because several tasks will be performed at the same time, rather than in sequential order. Analytics frameworks such as Apache Spark utilize Resilient Distributed Datasets (RDDs) to leverage memory optimization for performing high speed analytics across a distributed cluster of machines.

Multi-Format Data Handling

Modern big data tools are built to deal with multiple types of data, including:

Structured Data: Data stored in RDBMS (MySQL, PostgreSQL, etc.)

Semi-structured, as in JSON, XML, and log files which have some organizational structure.

Unstructured Data: images, videos, social media posts and raw text.

The ability to manage constantly changing data types has made big data tools indispensable for organizations running with various data origins like IoT devices, social media, cloud storage, and enterprise applications.

5.4 Big data processing techniques:

5.4.1 Various Tools

a. HDFS (Hadoop distributed file system)

Without further ado, let us before head to the HDFS (Hadoop distributed file system) let us know what is the file system. A file system is a Data structure or method which we used to manage file on disk space in an operating system. It means it permit the user to stay manage and get data from the local disk.

NTFS (New Technology File System) and FAT32(File Allocation Table 32) are examples of the windows file system. Some older versions of windows use FAT32 but can be used on all versions of windows xp. Just like windows, we have the such file system in Linux OS ext3, ext4, etc.

What is DFS?

DFS stands for distributed file system, it is the concept of storing the file in multiple nodes in distributed way. DFS basically gives abstraction for a single mega system whose storage is equal to all nodes in a cluster combined.

For example, if you have a DFS of 4 different machines of size say 10TB then you can store something around say 30TB across this DFS as you have a combined Machine of size 40TB. Such that the 30TB data is spread across this with the blocks distribution of data (as illustrated in fig.5.1).

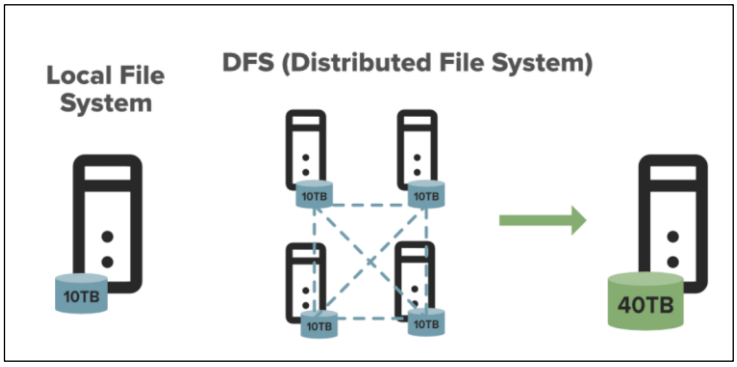


Fig. 5.1: Distributed File System

Why we need DFS?

Very large data sets are typically organized with a distributed storage system, or named a Distributed File System (DFS). For instance, in a Local File System (LFS), all the data is stored in a single machine, in DFS the data is being distributed over multiple servers. In the DFS network, each node adds its own storage capacity, resulting to greater overall storage space and performance.

As in the image, in the Local File System (LFS), there is just one storage unit of size 10 TB. When we mention the DFS system, it is actually formed by four interconnected servers with a 10TB storage drive. As we can see (as shown as image) by dividing and combining the storage capacity, the total available is 40TB. It helps appropriate resource allocation, parallel processing, and fault tolerance (as defined in fig.5.2 and fig.5.3).

This internally developed system has a key advantage over traditional centralized storage systems because of its design structure. HDFS (Hadoop Distributed File System), Google File System (GFS), Amazon S3 are examples of popular DFS implementations.

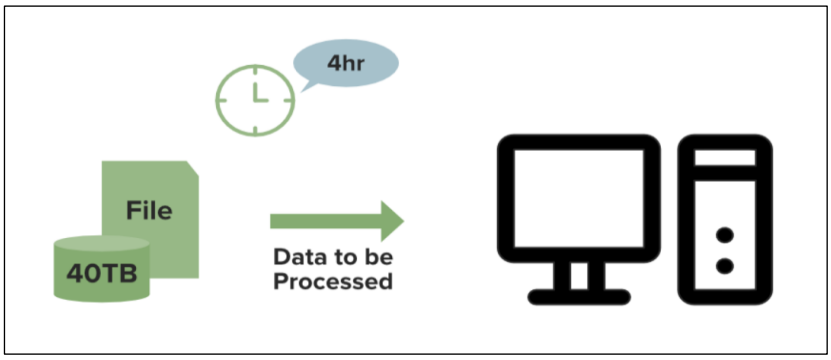


Fig. 5.2: Local file System Processing

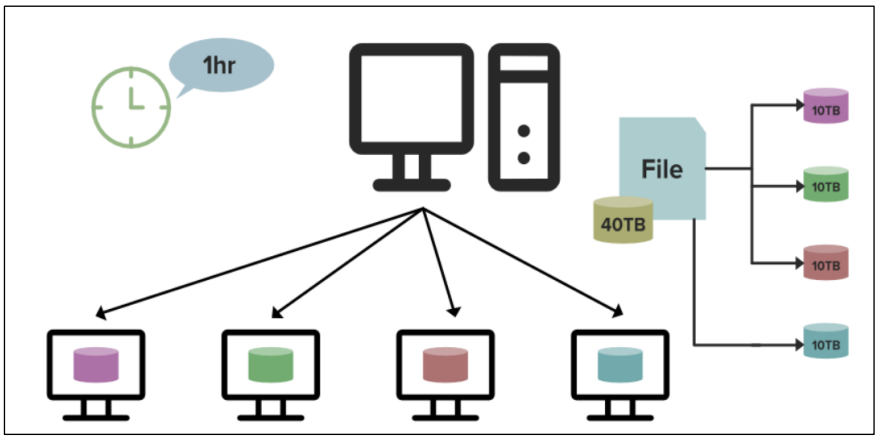


Fig. 5.3: Distributed file system processing

b. HDFS:

Hadoop Distributed File System (HDFS) is a distributed, fault tolerant and scalable file system to store and process huge data sets on multiple machines. This is the fundamental part of the Apache Hadoop ecosystem. HDFS enables businesses to distribute structured, semi-structured, and unstructured data over various nodes in a cluster while maintaining reliability and high availability.

Key features:

Distributed Architecture

The master-slave architecture is in the form of HDFS is where the NameNode is the master and the DataNodes are the slaves. The NameNode manages metadata including the location of the files and permissions to access data and the DataNodes holds actual data blocks. By distributing data across multiple nodes in a cluster, the load is balanced so that no individual machine will become overloaded. The architecture allows for data to be processed in parallel, making HDFS a natural fit for big data applications.

Fault Tolerance

HDFS has one of the most important functions of gracefully recovering from hardware failure. It does so by using a replication mechanism, which means that every piece of data is copied several times (the default replication factor is 3). HDFS automatically retrieves data from other available copies if one or

more DataNodes fail, which guarantees high availability and small data loss. That is why HDFS can be called a reliable mechanism for handling big data which is distributed over different clusters.

Scalability

HDFS provides high scalability, enabling organizations to grow the storage capacity by adding more nodes into the system. HDFS, unlike any other file system, is not limited by storage; it is 'horizontal scaled', and meaning that you can just add more commodity hardware to add more storage and computing power instead of upgrading your existing system. It is inexpensive and designed for petabytes of data.

High Throughput

HDFS is designed for handling massive amounts of data. It is a system intending for batch jobs, jobs where data is processed in one go. To not random access small files, HDFS supports high-bandwidth streaming access to application data, and is well-suited for the distributed data processing such as log processing, machine learning, and big data analytics. This allows for scaling operations out quickly and at an efficient cost.

Data Replication

Data blocks are automatically replicated into multiple nodes to provide redundancy and reliability. Each file is split into fixed-size blocks (usually, either 128MB or 256MB) by default and stored three times on separate nodes. This means that to serve the reads for data, multiple copies of it can be retrieved, since there should be carbon copies available for replicating the data, leading to no data loss and serving read performance. Also, in case of node failure the system rebalance and replicas of lost data automatically to keep data integrity intact.

Write-Once, Read-Many Model

HDFS uses a write-once, read-many access model, which means that once a file has been written to the system it cannot be changed, only read multiple times. This is great for big data use cases where you must perform computations over an entire dataset that can be very large and is immutable. Because data is immutable, this model reduces the complexity of managing data consistency, mitigating risks of corruption from simultaneous changes. Instead of changing any file users can add new (create new file with the added data) which supports better retrieval of data with better read.

Architecture of HDFS:

HDFS is a distributed file system designed for large-scale data storage and processing, comprising a master-slave architecture. It is built from various parts which collaborate to achieve features like fault tolerance, scalability, and high throughput data access. Hadoop works o MapReduce algorithm which is a master-slave architecture, HDFS has following nodes (as shown in fig.5.4):

a. NameNode (Master)
b. DataNode (Slave)

a. **NameNode:** NameNode acts as Masters in a Hadoop cluster that directs the Datanode (Slaves). Namenode is primarily responsible for holding the Metadata that is simply data about the data. For example, meta data can be the transaction logs that record the user activity in a Hadoop cluster.Meta Data can also refer to file name and size, and location information such as Block number, and Block IDs that the Namenode maintains for a more efficient communication with the DataNode. Namenode provides the operation to the DataNodes like delete, create, Replicate, etc.

Being NameNode as a Master it should have a High RAM or Processing power to maintain or guide all the slaves in a Hadoop cluster. All the slaves i.e. DataNodes send heartbeat signals and block reports to Namenode.

b. **DataNode:** DataNodes are the slaves. DataNodes are primarily used to store the data in a Hadoop Cluster, the number of DataNodes is between 1 to 500 or more than 500 also in which your Hadoop cluster has More data Can be stored. As a result, the DataNode should have a high storing capacity to store a large number of file blocks. Datanode performs create, delete, etc according to the instruction provided by NameNode

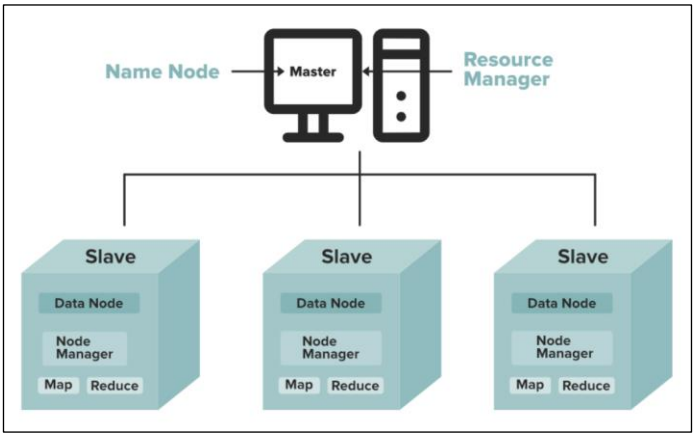


Fig. 5.4: HDFS architecture

c. **Hadoop:**

Hadoop is an **open-source framework** developed by the **Apache Software Foundation (ASF)** that enables **distributed storage** and **processing of large datasets** across clusters of commodity hardware. It is designed to handle **big data** efficiently by leveraging a **parallel computing** approach. Hadoop is fault-tolerant, scalable, and flexible, making it an essential tool for organizations dealing with **structured, semi-structured, and unstructured data.**

The framework allows businesses to **store and process petabytes of data** without relying on expensive high-performance hardware. Instead, it distributes workloads across multiple machines, ensuring **high availability** and **fault tolerance.** Hadoop's architecture is based on the **MapReduce programming model**, which divides data processing tasks into smaller, parallelizable units, improving efficiency and speed.

Key Features:

a. Distributed Architecture

Hadoop is a distributed architecture that enables a single set of data to be stored and processed across multiple machines as an alternative to a single high-performance server. It is based on a master slave architecture, in which NameNode (master) stores the metadata and tracks the location of files stored, while the DataNodes (slaves) contains the actual data. This design enables Hadoop to scale horizontally, allowing it to accommodate large volumes of data on cluster of common hardware. Hadoop is a distributed framework, it is highly scalable and resilient, so we can easily manage our enormous data.

b. Scalability

It is horizontal scalable .Hadoop can add more machines (known as nodes) to the cluster when a significant volume of data is generated. Hadoop's scalability enables organizations to expand their infrastructure without incurring the high costs associated with proprietary hardware upgrades. It allows

Hadoop to scale from terrabytes, to petabytes, even exabyte worth of data by simply splitting it into many nodes. It allows organizations to scale their data processing capabilities without sacrificing performance or forcing them into a sub sequential hardware bind.

c. High Availability and Fault Tolerance

Hadoop's fault-tolerance mechanism is one of its strongest suite features, meaning that even if some hardware components fail, the data is still accessible. Visualization create an Extensible File System (HDFS) Hadoop does this by replicating the data blocks to multiple nodes in the cluster to improve fault tolerance. The default replication factor here is three, so a block of data is stored on three different machines. In case a node fails, Hadoop automatically fetches the data from the other available replicas and ensures that no data is lost. The prominent high availability feature guarantees accessibility to critical data even in the event of hardware failures.

Giant Population Size & CPU Processing

Hadoop is a high throughput framework designed for batch workloads, running tasks in a parallel manner. It uses the MapReduce programming model that lets data be partitioned into multiple smaller pieces and execute them in parallel on different nodes. Unlike sequential execution, Hadoop's parallel processing speeds up data analytics and complex computations. This is also perfect for working with huge datasets in areas like scientific research, financial analysis, and large-scale business intelligence.

Cost-Effective

You can use Hadoop to process a large amount of data very easily because it is an open-source framework, so organizations can use it without paying any licensing fees. Unlike traditional big data solutions that require costly proprietary software and high-class hardware top run, Hadoop can work fine on a low-cost machine (commodity hardware) . With this, it provides a convenient way for businesses to store and analyze garage datasets without significant infrastructure costs. This approach maintains a low-cost recovery policy while maximizing the storage and processing capabilities and hence Hadoop is an economical way for handling Big Data.

Data Replication for Reliability

The word Hadoop can guarantee data reliability and durability by having its data replication mechanism. Implicitly, 3 replicas of the data blocks are distributed across the nodes in the cluster. This means that even if several machines go down at once, the data is still available and safe. Moreover, the NameNode keeps an ongoing check on DataNodes and when the data becomes inconsistent, it re-replicates data check. It is essential for disaster recovery and through this feature any organization does not have to lose critical information due to hardware failure.

Write-Once, Read-Many Model

Hadoop uses a write-once, read-many approach, which implies that when data have been stored in the HDFS (Hadoop Distributed File System), it cannot be modified. Rather, if it needs to be updated, it needs to create a new version of that data. This helps out with data consistency and concurrency issues. The write-once model is especially useful for large-scale analytical workloads where data is downloaded once and read many times to generate insights, making Hadoop very efficient for batch processing, as well as historical analysis of data.

Support for Cloud Computing

Hadoop integrates smoothly with cloud, such as Amazon AWS, Google Cloud, as well as Microsoft Azure enabling the organizations to implement on-demand computing resources. Amazon EMR (Elastic

MapReduce) and Google Dataproc are cloud-based Hadoop solutions allowing companies to process big data at scale without the need for on-premises hardware. This allows companies to provision computing resources based on workload demands, providing flexibility, scalability, and cost savings.

Hadoop ecosystem:

Hadoop Ecosystem is a platform or collection of multiple services to address the big data challenges. It covers Apache based projects and a suite of commercial tools and solutions. The essential components of Hadoop include HDFS, MapReduce, YARN, and Hadoop Common Utilities (as shown in fig.5.5). The rest of the tools or solutions work in supporting or augmenting these core managed elements. These various tools work together to perform a variety of functions including data absorption, analytical processing, data storage and maintenance etc.

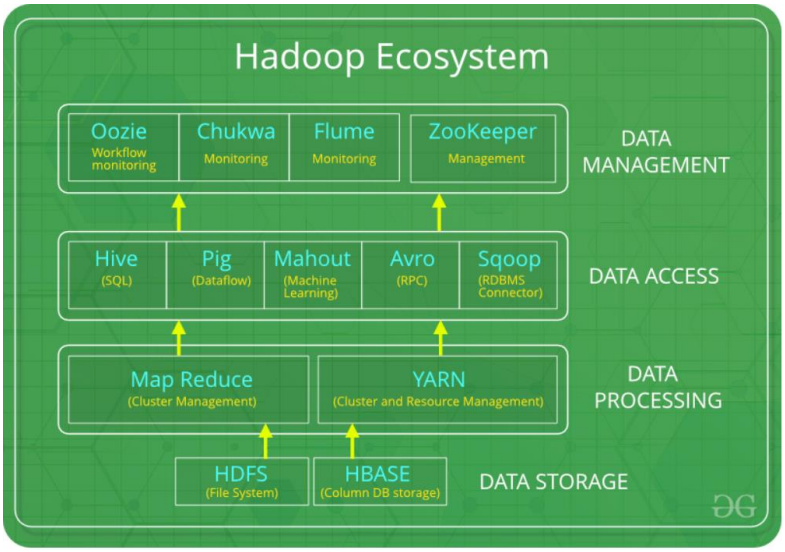


Fig. 5.5: Hadoop Ecosystem

Components:

1. HDFS(Hadoop Distributed File system):

Hadoop Distributed File System (HDFS) is the core storage layer in the Hadoop ecosystem, and it is purposely built to store and process large volumes of structured and unstructured data across a cluster of machines. This is based on a master-slave architecture which consists of a NameNode (master) that stores metadata (data location information) and a DataNodes (slaves) that stores the data. HDFS by design is fault-tolerant, and it keeps many copies of data blocks on many nodes in case of node failure even if one or more fails data is still available. The replication mechanism makes it more reliable, and adds more integrity to the data. Furthermore, HDFS is cost-efficient because it operates on standard hardware components, enabling businesses to store massive data sets at a very low price. Acting as the backbone to coordinate between software clusters across hardware, HDFS allows for the scalable storage of big data and provides the basis of the Hadoop ecosystem, giving it the ability to work in unison.

2. YARN(Yet Another Resource Negotiator):

The resource management layer of Hadoop, which is responsible for managing and scheduling computing resources across clusters. As it provides CPU, memory, and bandwidth to different applications, it makes sure that cluster resources are used efficiently. YARN has three main components:

Resource Manager: It manages the overall allocation of resources, ensuring that applications receive the required processing power.

Node Manager: Runs on every node and is responsible for monitoring the resource usage i.e CPU, memory and returning it to the Resource Manager.

Application manager: The Application Manager serves as a medium, passing on information between the Resource Manager and the Node Managers, allowing resources to be allocated and deallocated on an as-needed basis.

Hadoop also has the capability of supporting multiple processing frameworks due to YARN and this makes it more flexible and adaptable to different Big Data applications.

3. MapReduce:

A MapReduce is the programming model that allows you to process a large amount of data on a Hadoop cluster, in parallel and distributed across a cluster of nodes. It is made up of two main functions, Map() and Reduce().

Map() Function: The map function takes in the input data and filters, sorts, and transforms it into a list of key-value pairs. After that these key-value pairs are grouped based on some similar attributes.

Reduce(): The reduce function processes the clustered data produced by the map stage, performing summary, computation, or aggregation to produce the end processed data.

So to enable the scalable processing of the huge data sets the processing over the distributed nodes (MapReduce) has been used. Despite being one of the initial processing models added to Hadoop, fresh technologies such as Apache Spark enable quicker in-memory processing which many companies explore for real-time analytics.

4. Apache Pig

Apache Pig, which was created by Yahoo, is high-level data flow scripting platform that helps to process and analyze the large data sets. The language it uses is called Pig Latin, which is like SQL but made for working with big data transformations. Pig allows developers to write scripts for data analysis without having to hand-write Java-based MapReduce programs. Pig compiles these scripts into a series of MapReduce jobs, which are then run in the background in an efficient and seamless manner. Highly extensible, users can define their own functions for processing. Because Pig hides the complex inner workings of the MapReduce programming model, it is a very popular tool among Data engineers for data preprocess Apache Hive

Apache Hive is an open-source data warehouse infrastructure built on top of Hadoop that provides data summarization and ad hoc querying capabilities using a SQL-like scripting language called Hive Query Language (HQL). It enables SQL-interested analysts to query Hadoop without having to write any MapReduce code. Highly scalable, enabling batch processing through real-time query execution Hive has two main components:

5. Apache Mahout

Mahout is a library for scalability and Distributed Computing writing machine learning applications in the Hadoop ecosphere. It includes libraries for clustering, recommendation systems, classification, and collaborative filtering. Mahout enables organizations to create intelligent applications that utilize machine learning to replicate the way data, in combination with patterns and users, repeat. Mikhail increases Hadoop's use by allowing for efficient distribution of computations over vast datasets, making it invaluable for use cases including predictive analytics, fraud detection, and recommendation engines in domains such as e-commerce, finance, and social media.

6. Apache Spark

By far one of the best big data processing engines, supporting real-time data analytics and different workloads, such as batch, streaming, machine learning, and graph processing. In contrast to Hadoop MapReduce, which writes intermediate data to disk, Spark processes data in memory, and is therefore orders of magnitude faster. Spark is commonly utilized for real-time event processing, fraud detection, sentiment analysis, and large-scale AI applications. Hadoop has been integrated with many other tools but its versatility and integrated nature make it one of the most powerful tools built in the big data ecosystem.

7. Apache HBase

HBase (from Apache) is a NoSQL database that operates on large amounts of sparse and unstructured data. It's built on ideas from Google's BigTable, and works atop HDFS, offering random, real-time access to large datasets. This is especially helpful in cases where in HBase fast lookup and retrieval of small amount of data from huge database is needed. Kafka is commonly utilized for use cases like log processing, IoT sensor data management, and real-time analytics. HBase is built on top of Hadoop so is not capable of processing SQL queries like relational databases but offers a rich schema that can manage dynamic and high-velocity data.

Other components:

Solr & Lucene

Solr and Lucene services for search and indexations. Lucene is a search library in Java that provides functions such as indexing text and search functionality, such as spell-checking and complex queries. It is built on lucene and is a high-performance, scalable search platform optimized for large-scale applications. They are commonly used in search engines, enterprise content management, and information retrieval systems.

Apache Zookeeper

Zookeeper is a distributed application coordination and synchronization service. Zookeeper: In Hadoop, Zookeeper is a centralized service which allows configuration management, leader election and messaging between various components. This addressed issues such as causing Hadoop clusters to be more stable in handling failures and enhanced the stability and resilience of Hadoop clusters by resolving challenges faced by cluster management, distributed locking, and service discovery. It provides a robust, centralized repository of metadata that ensures large-scale Hadoop installations are reliable and efficient.

Apache Oozie

Oozie is a workflow scheduler system which is used to schedule the job execution in Hadoop. It allows users to set up workflows and dependencies between tasks, so that complex jobs run in a set order. It is particularly employed for scheduling ETL (Extract, Transform, Load) procedures, coordinating massive data pipelines, and automating Hadoop job processing.

Oozie is primarily used in two major job types:

Oozie Workflow Jobs: Jobs whose executing tasks need to be done in a specific order.

Oozie Coordinator Jobs: It runs on the basis of data availability in the cluster or some external stimulus.

5.5 HIVE

Apache Hive is an SQL-like query processing and data warehousing system built on top of Hadoop. It enables users to conduct structured query processing on vast datasets kept in HDFS (Hadoop Distributed File System). Hive, which was developed at Facebook, was born from the problem of querying large

datasets stored in Hadoop, and eventually it formed an open-source project under the Apache Software Foundation.

It is especially helpful for data analysts and business intelligence developers specialised with SQL but without knowledge of writing the advanced MapReduce programs. Hive enables users to write queries in HQL (a SQL-like language, tailored for dealing with massive amounts of data) rather than writing Java-based MapReduce queries.

Key features:

a. **SQL like query langauge (HiveQL):** Hive offers a HiveQL (Hive Query Language) similar to traditional SQL, so any user can write queries without having to learn Hadoop internals. HiveQL supports SQL-like constructs such as SELECT, JOIN, GROUP BY, ORDER BY, and aggregation functions. It also provides support for the creation of custom User Defined Functions (UDFs) that can extend its functionality.

b. **Scalability and Performance:** Hive helps in handling large-scale datasets efficiently. It uses Hadoop's ability to process data in parallel to run queries in a distributed fashion. Add More Nodes in Hive Cluster to Scale It to Petabytes of Data Although optimized for batch processing, it can be integrated or can work with Apache Tez or Spark to achieve better performance during real-time analytics.

c. **Schema-on-Read Model:** Hive is a Schema-on-Read approach (i.e. there is no need for data to be structured before loading into Hive). Instead, the schema is enforced when reading the data. This enables organizations to store semi-structured and unstructured data (JSON, Avro, Parquet, etc.) and query it without needing to build definitions ahead of time.

d. **Seamless Integration with Hadoop Ecosystem:** Hive has a strong integration with Hadoop ecosystem and operating with the HDFS, MapReduce, YARN, and HBase. It uses HDFS for data storage, and MapReduce, Apache Tez, or Apache Spark as execution engines for processing the data. It also integrates with BI (Business Intelligence) tools such as Tableau, Power BI and Apache Zeppelin via JDBC/ODBC drivers.

e. **Partitioning and Bucketing:** Hive supports Partitioning and Bucketing to improve query performance and minimize data scanning:

   Partitioning: Splits large tables into smaller, logical partitions based on the values of certain columns (date based partitions, etc.). This allows us to avoid scanning the full dataset and only scan the relevant partitions which reduces query execution time.

   Bucketing: Data in partitions is broken further down into smaller groups (buckets) based on hashing of the column values. If the data is needed to be joined in a very efficient manner or if to be sampled, the bucket can be used.

Hive architecture:

Apache Hive is a data warehousing solution built on top of Hadoop, which use an SQL-like query language to store and manage large data sets over HDFS (Hadoop distributed file system) or HBase. A query language similar to SQL, called Hive Query Language (HQL), is used to allow users to process structured and semi-structured data.

The Hive architecture has several components (as defined in fig.5.6) which are essential for executing queries, managing data, and storing it efficiently. We'll go through each element in detail.

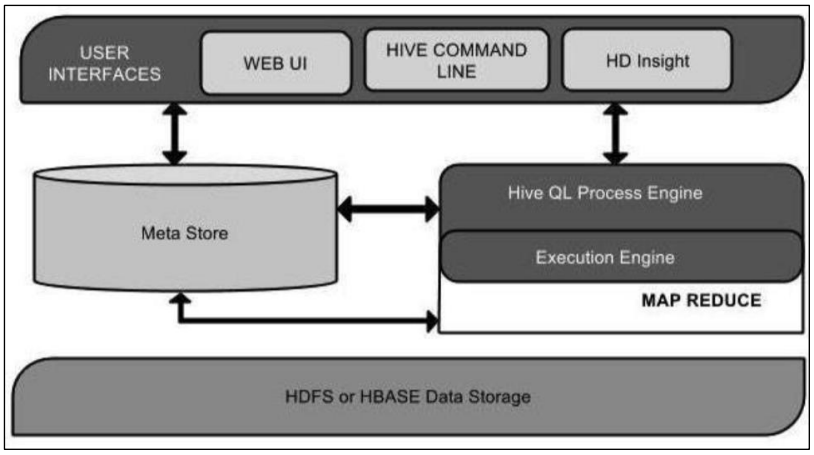


Fig. 5.6: Hive architecture

Components:

1. User interface(top layer)

Hive architecture always starts from the User Interface Layer, where users will communicate with Hive to access the query and manage the data. Hive provides different interfaces for different uses. Following are some of interfaces:

- **a. Web UI (User Interface):** The web-based graphical interface enabling users to query Hive via browser. Users are able to query, check the running jobs, and see the outcome of the query visually. This facilitates a more point-and-click style of interaction rather than command-line usage for business analysts and users.
- **b. Hive Command Line Interface (CLI):** It is a command-line tool that enables submitting HiveQL queries, running data management commands, and fetching query results. This mode is popular among developers and data engineers for batch processing and automation tasks.
- c. **HDInsight**: It is a cloud-based interface from Microsoft azure that allows the users to run the hive queries on Hadoop clusters. It's cloud service integration allows for scalable and efficient data processing.

2. Metadata Storage Layer(Metastore):

This is an essential part of the hive which contains metadata (data about data). It acts as a registry for Hive tables, and allows the system to locate, organize, and process data effectively.

Functions of the Metastore:

1. It store information about metadata of tables, databases, partitions, schemas, column types and their storage locations.
2. Keeps a mapping from logical table names to physical file locations (hdfs or hbase)
3. Enables schema evolution, allowing users to make changes to tables without having an impact on data that is stored there.
4. This makes sure that the data is consistent across the various parts of Hive.

How Metastore Works:

a. Hive receives queries from the User Interface Layer.

b. HiveQL Process Engine Queries Metastore for Schema and table Information

c. The next stage is the Execution Engine which executes the query using the metadata obtained from the Metastore.

The Metastore is typically hosted on an external relational database like MySQL, PostgreSQL, or Apache Derby, so that it can provide fast access to metadata.

3. HiveQL Process Engine (Query processing Layer):

Managing Hive processes that handle HiveQL (Hive Query Language) queries and convert them into execution plans that Hadoop can run is the HiveQL Process Engine.

Functions of HiveQL process engine:

a. Query Parsing: When a user enters a query, the engine parses that query, checking for syntax errors and validating it against the metadata from the Metastore.

b. Query optimization: The engine parses the query, checks for syntax, logical, and semantic correctness, and builds an execution plan in a Data Structure.

c. Logical Plan Generation: This is the step where the query is translated into an execution plan which defines how the data should be processed.

HiveQL Example:

```
SELECT cust_id , SUM(purchase_amount)
FROM sales_data
WHERE purchase_date > '2024-05-01'
GROUP BY customer_id;
```

In this above query, the query engines checks if the sales_data exists in the metastore and it determine that sales_data is stored in HDFS and decide how to process and retrieve the data and the execution engine takes over to run the query efficiently that sum up the purchase_amount for the given cust_id and purchase_date.

4. Execution Engine(Processing Layer):

The execution engine runs the queries and interacts with Hadoop's computational framework. It receives the optimized execution plan from HiveQL Process Engine and executes it on Hadoop.

How it works:

1. It compiles HiveQL queries to low-level MapReduce, Tez, or Spark jobs.

2. Then it submit the jobs to Hadoop to get executed.

3. It observes job execution, then fetches the results on the job after processing has completed.

4. It will interact with HDFS or HBase to read/write data efficiently.

Various Execution Modes:

a. MapReduce (Default Mode): Query Execution Engine divides the query into Map and Reduce tasks and distributes them across multiple nodes in the cluster.

b. Apache Tez (Fast Alternative): Tez is a more efficient framework than MapReduce with less number of jobs and faster as soon as the query is run, it simplifies the process for the faster execution of systems.

c. Apache Spark (Real-time Processing): Spark is used when we need interactive querying and real-time analytics rather than batch processing.

The Execution Engine gives Hive the power to work with the most distributed computing framework which can handle terabytes of datasets in an efficient way.

5. Storage layer(HDFS or HBase Data Storage):

Hive manages and retrieves data that Layers into Storage Layer. Hive does not retain data on its own, but it forms the integer over the HDFS or HBase.

HDFS (Hadoop Distributed File System): It is main storage system for hive. It stores structured, semi-structured and unstructured data across multiple distributed nodes. HDFS ensures the fault tolerance and scalability by replicating data across multiple machines.

HBase(Hadoop Database): It is NoSQL database that supports real-time data access. It is widely used when random read/write opertaions are required instead of batch processing. It allows for querying of large datasets at high speed.

5.6 Spark:

Apache Spark is an open-source distributed computing system that can be used for big data processing and analytics. Unlike traditional Hadoop MapReduce, which process data in a series of stages with intermediate writes to disk, Spark computations are done in-memory, which makes them orders of magnitude faster. Apache Spark is a fast and general engine for big data processing, with built-in modules for streaming, SQL, machine learning and graph processing.

Key features:

1. **Speed and In-Memory Processing:** One key benefit of Spark is its in-memory computation model, which avoids repeated disk I/O between stages of processing, resulting in much faster processing than Hadoop MapReduce, which always writes intermediate results to disk. Unlike systems that need to spill state to disk, Spark caches intermediate results in memory, reducing overhead and improving efficiency. For iterative ML tasks and analytics, Spark has been shown to be up to a hundred times faster than Hadoop.
2. **Unified engine for data processing:** Spark engine is designed to address various data workloads on a single platform. Its provides batch processing with Spark Core, stream processing with Spark Streaming, querying with Spark SQL, machine learning capabilities with MLlib, and graph computation with GraphX. It removes the requirement of using different tools to do what they do between two different platforms.
3. **Fault Recovery and Reliability:** It ensures fault tolerance through Resilient Distributed Datasets (RDDs). RDDs also keep track of the lineage (history) of the transformations, which means, after a failure, Spark can restore lost data at the lost data partition. A worker node can crash, and Spark doesn't need a check pointed version of the data, it can reconstruct the lost partitions from the transformations that were recorded.
4. **Lazy Evaluation and the Query Optimization process:** Instead, Spark works on Lazy evaluation, where transformations are not cued instantly—they're logged and optimized prior to execution. After some action is triggered (e.g. results are collected), Spark builds a optimized DAG (a Directed Acyclic Graph) by merging multiple transformations and minimize unnecessary computations. This increases efficiency and shortens execution time.
5. **Support of Different Storage Systems:** Spark work with many storage solutions: HDFS, Apache Hadoop HBase, Amazon S3, Google Cloud Storage, Azure Blob Storage, Apache Cassandra. This provides a high level of flexibility in Spark which can adapt to varying enterprise environments.

6. **Static Control and Parallel Computing:** Small or big, Spark can run on a single machine or on a 1000 node cluster. It works in Standalone mode as well as with cluster management solutions like Hadoop YARN, Apache Mesos, and Kubernetes. It is also very much optimized for cloud platforms like AWS, Microsoft Azure, and Google Cloud.

5.6.1 Difference between apache hive and apache spark:

Table 4: Difference between apache hive and apache spark

Apache Hive	Apache Spark
Apache hive is a data warehouse infrastructure built on the top of the Hadoop that primarily designed for querying and managing structured data. It translates SQL-like queries into MapReduce jobs for execution on Hadoop.	Apache Spark is a distributed data processing engine that provides fast computation for large-scale data processing. It supports both batch and real-time data analytics in-memory computing.
Hive primarily works on batch processing where queries are executed as MapReduce jobs, making it efficient for handling large volumes of data. However, it is not optimized for real-time processing.	Spark supports both batch and real-time processing. With spark streaming, it can process real-time data streams efficiently, making it more flexible compared to hive.
Hive is slower compared to spark because it relies on disk-based MapReduce operations, which involve multiple read/write operations on HDFS. The execution time depends on the complexity of the queries.	Spark is significantly faster because it processes data in-memory, reducing the need for frequent disk read/write operations. It can perform iterative computations much more efficiently.
Hive uses HiveQL, which is similar to SQL, making it easy for data analyst and SQL users to work with. It does not require advanced programming skills.	Sparks supports structured, semi-structured and unstructured data, allowing it to handle diverse data formats like JSON, CSV, Parquet, and even streaming data from kafka.
Since Hive uses SQL-like queries, it is easy to learn and use for people familiar with SQL. It is widely used by data analysts for querying large datasets.	Sparks requires programming knowledge in languages like Scala or python, making it more developer-centric. However, Spark SQL allows SQL-like querying, making it easier for analysts.
Hive does not have built-in machine learning capabilities. It is mainly designed for data warehousing and querying rather than advanced analytics.	Spark provides built-in machine learning libraries, enabling scalable and fast machine learning model training on big data.
Hive is more cost-effective since it uses disk-based storage, reducing memory requirements. It is suitable for businesses focusing on low-cost batch processing.	Spark requires high memory (RAM) for in-memory processing, which can make it more expensive in terms of infrastructure costs. However, it significantly improves performance for critical applications.
Hive is preferred when dealing with large-scale structured data for batch processing and data warehousing. It is useful for report generation, business intelligence, and ETL workflows.	Spark is preferred for applications that requires fast data processing, real-time analytics, machine learning, and AI. It is widely used in big data analytics, financial services, fraud detection, and IoT applications.

5.7 Impact of big data on data engineering

Big Data has undoubtedly changed the data engineering landscape by providing organizations the ability to glean insights from large amounts of structured, semi-structured, and unstructured data. Classic data management approaches are inadequate to the scale, velocity, and variety of contemporary data. Consequently, organizations have turned to sophisticated data engineering solutions that utilize distributed storage, parallel computing, real-time processing, and AI-powered analytics. Now we can discuss what are the effects of Big Data on data engineering in more detail:

a. Distributed Storage Systems from Traditional Databases:

Traditional databases, such as RDBMS (Relational Database Management Systems), are data structures that have well-defined schemas. But with the proliferation of semi-structured and unstructured data, traditional databases could no longer keep up. The world of Big Data brought distributed storage solutions like HDFS (Hadoop Distributed File System), Amazon S3, Apache HBase, and Google Cloud Storage where a few petabytes of data are spread across hundreds of nodes. They offer scalability, which enables organizations to scale their infrastructure as data grows. They also provide fault tolerance by replicating data across multiple nodes, giving high availability. It was the switch to the distributed storage that made data storage cost-efficient and feasible with large volumes.

b. Emergence of Distributed Computing Frameworks

Handling large amounts of data is enabled through distributed computing, which breaks the tasks down into smaller tasks on a separately distributed system to allow increased efficiency. Parallel data processing is performed on data arranged in distributed framework by utilizing various frameworks like Apache Hadoop, Apache Spark, and Apache Flink which can reduce processing time dramatically. The MapReduce model of Hadoop is quite popular for batching processing, but Spark allows for in-memory computing, making it up to 100 times faster in iterative tasks. Apache Flink focuses on stream processing, a key for any real-time analytics. This very idea enabled big data processing of terabytes of data, in minutes which fuels the developments of big data analytics and machine learning.

c. NextGen Data Pipeline: Real-Time Data Processing and Streaming Analytics

Traditionally, businesses worked with batch processing, in which data was collected and processed at specific intervals. But current applications need for processing real-time data to take instantaneous decisions on real-time data onboarding. Technologies like Apache Kafka, Apache Flink, and Spark Streaming enable organizations to process continuous data streams from IoT sensors, financial transactions, and social media feeds. Consider, for example, that banks rely on real-time analytics for fraud detection, hunting down suspicious transactions as they are processed. The same thing goes for predictive maintenance in manufacturing that relies on IoT data to identify possible fails ahead of time. Real-time data processing has enhanced operational efficiency and decision making enormously.

d. Data Pipeline and ETL Workflows Evolution

The paradigm of ETL has seen a tremendous shift with Big Data and is now mostly being replaced with ELT models. In ELT, data is stored as raw data in a data lake and then transformed on demand. While this approach, in turn, allows more flexibility and scalability. Tools such as

Apache Airflow, Apache NiFi, and AWS Glue automate and orchestrate complex data workflows, providing seamless movement of data between systems. They assist organizations in working with large-scale data integration in a near real-time, automated, and consistent manner, so that manual intervention can be largely avoided leading to data consistency.

e. Increased Usage of NoSQL Databases

With the advent of Big Data, NoSQL databases approached to make way for structured, semi-structured, and unstructured data. Unlike SQL databases, NoSQL databases like MongoDB, Apache Cassandra, and Amazon DynamoDB offer high-speed read/write processes, schema flexibility, and horizontal scalability. These properties suit them well for processing real-time and heavy workload like a recommendation engine, social media analysis, and Internet of things (IoT) apps. NoSQL databases have gained enormous significance in the final data architectures since they enable organizations to shop and sign facts more effectively than traditional relational databases.

f. Challenges of Data Governance, Security, and Compliance

With the increase in data volume, organizations have to ensure that their data governance, data security as well as regulatory compliance. Business regulations like GDPR, CCPA, HIPAA insist on data privacy and access control. To protect sensitive information, data security can be enforced through a mechanism that implements encryption, access control policies, data masking, etc. Cleaning is not the only Transition needed, in fact you have features like Data lineage and auditing that are important, to track the modifications made in your data and get the details of who, how and when it was changed. Security tools like Apache Ranger, Apache Atlas, or AWS Lake Formation assist organizations in implementing security policies and preserving data integrity. The increased focus on data governance has made it a key responsibility of data engineers.

g. The Cloud Data Engineering Takeover

If you are not aware, cloud computing has transformed data engineering with immediate scalability, convenient pricing and integrated data. Data is stored, processed, and analyzed in any cloud-based data platform such as AWS, Google Cloud, and Microsoft Azure, which organizations are increasingly implementing. Cross-Cloud Data ManagerCloud data lakes and warehouses such as Amazon S3, Google BigQuery, and Azure Synapse Analytics makes it easier for companies to use on-demand databases to administer huge datasets without the financial burden of investing in expensive on-premise infrastructure. Serverless computing can be further enabled by cloud platforms which reduce operational overhead and improve agility. Big Data processing is now accessible to businesses of all sizes thanks to the transition to cloud-based data engineering.

h. Integrating AI and machine learning into Data Engineering

The rise of big data providing the need for advancements in machine learning (ML) and artificial intelligence (AI) required to work hand in hand with data pipelines. Since you are one of the most best machine learning models, it needs massive quality dataset to train and predict. On such teams, data engineers are critical to create ML-ready pipelines, automate data preprocessing, and scale the deployment of models. Model Ops (Machine Learning Operations) is a discipline that is

emerging with a focus on deployment, monitoring, and continuous refinement of the models. MLflow, TensorFlow, and Apache Mahout are the tools that help enterprises build AI powered applications quickly. By incorporating ML in data engineering pipelines, organizations are speeding up the innovation process in several domains.

i. Revolutionizing Data Management with DataOps

A methodology similar to DevOps — DataOps encourages people & processes within data engineering to work together in the most effective manner. However, DataOps practices target CI/CD (Continuous Integration/Continuous Deployment) for data pipelines, monitoring, and real-time optimization. DataOps focuses on improving the quality of data, reducing errors, and delivering data faster through the use of version control, containerization (Docker, Kubernetes), and automated testing. This method involves great utility for organizations dealing with complex data workflows, as it ensures data remains accurate, consistent, and reliable during its lifecycle.

j. Making Data Democratic: Self Service Analytics

Big Data is encouraging organizations to allow non-technical users to access and analyze data with self-service analytics tools – Tableau, Power BI, Apache Superset, etc. They usually provide intuitive dashboards and drag and drop interfaces with no need for SQL or programming knowledge for business teams to explore data, create reports, and derive insights. The advent of data democratization reduced the waits for ad-hoc reports from the data engineering teams and helped the employees across departments make data-driven decisions fast and efficiently.

CHAPTER 6

DATA PREPROCESSING

Debjit Mohapatra[1], Gagandeep Singh[2] and Simran[3]

[1,2,3]GNA University, Phagwara

6.1 Overview

Data preprocessing serves as an absolutely vital phase within the overall data analysis workflow. This significant process encompasses the detailed conversion of unrefined, raw data into a well-structured, orderly, and functional format that is suitable and ideal for various machine learning algorithms and analytical endeavors. The integrity and quality of the data play an absolutely crucial role in obtaining precise, reliable, and trustworthy outcomes, and effective preprocessing plays a pivotal role in ensuring that the dataset remains devoid of discrepancies, inaccuracies, and superfluous, unnecessary information. In this comprehensive chapter, we will delve deeply into the essential stages and processes of data preprocessing, which include data cleansing, imputing missing values, eliminating noise, selecting relevant features, reducing dimensionality, and normalizing the data effectively. Each step in this essential process contributes significantly to enhancing the quality and usability of the dataset, thus ensuring that subsequent analysis is as accurate and informative as possible.

6.2 Data Transformation Techniques:

6.2.1 Cleaning Data

Data cleaning is an essential and fundamental procedure aimed at identifying and rectifying or eliminating various segments of data that may be corrupt, inaccurate, or entirely irrelevant. When working with raw data, it is common to encounter a variety of errors, inconsistencies, and outliers, all of which can significantly compromise the overall performance and accuracy of machine learning models. Consequently, it becomes imperative to actively resolve these issues to guarantee the quality of the data that will be utilized in subsequent analyses. The following tasks are standard practices in the data cleaning process, and they serve to improve not only the integrity but also the reliability and overall robustness of the dataset:

a. Handling Duplicates

The presence of duplicate records can significantly distort analytical outcomes and hinder the effective training of models used for data analysis. By systematically identifying and methodically eliminating these duplicates, one can ensure that each unique data point remains distinct and separate from others. This careful process guarantees that every data point contributes uniformly and appropriately to the overall analytical process, ultimately leading to more accurate results and better-informed decision-making.

Example:

Consider a dataset of customer information where some customers have been entered more than once.

Customer id	Name	Age	Email
1	John	28	John.48@gmail.com
2	Smith	30	Smsith5678@gmail.com
1	John	28	John.48@gmail.com
3	Alice	29	Alice.b67@gmail.com

In this example, the record for John is duplicated. Removing the duplicate ensures that the dataset is accurate.

b. Correcting Errors

The dataset is likely to contain numerous errors, including typographical mistakes, inaccurate figures, and various discrepancies. A clear illustration of this issue is found within a specific column that is designated for age, which could potentially include negative values—an anomaly that is simply logically untenable. It is absolutely essential to thoroughly address these inaccuracies through careful correction or complete elimination to ensure the overall data integrity and reliability for analysis and decision-making purposes.

Example:

Consider a dataset with an age column containing negative values.

Customer id	Name	Age	Email
1	Doe	28	Doe56@gmail.com
2	John	-30	John.b45@gmail.com
3	Smith	29	Smith7869@gmail.com

The negative age for John is an error. Correcting this error involves either removing the record or replacing the negative value with a plausible one.

c. Standardizing Formats

Data frequently appears in a wide variety of inconsistent formats, which can encompass dates being presented in a wide range of different styles or text displayed in numerous variations of letter cases. By taking the essential steps to standardize these differing formats consistently across the board, we can achieve a significantly higher level of consistency and uniformity throughout the entire dataset. This methodical approach not only improves the overall quality of the data substantially but also enhances its usability greatly, making it increasingly easier for users to analyze and interpret the information effectively and efficiently.

Example:

Consider a dataset with dates in different formats.

Customer id	Name	Date of birth
1	John	1990-05-15
2	Smith	15/05/1990
3	alice	May 15,1990

Standardizing the date format to YYYY-MM-DD ensures consistency.

Removing Irrelevant Data

To greatly enhance the overall clarity and focus of the analysis being conducted, it is absolutely essential to meticulously eliminate those columns or rows that do not contribute meaningfully to

the overall evaluation process, such as unique identifiers or various forms of extraneous metadata that do not add value. This thoughtful and deliberate reduction in complexity will significantly facilitate a much more streamlined and effective interpretation of the data involved, making it easier to draw meaningful conclusions and insights from the information gathered.

Example:

Consider a dataset with customer information including a unique identifier and metadata.

Customer id	Name	Age	Email	Metadata
1	John	25	John.io@gmail.com	X1
2	Smith	28	Sam234@gmail.com	X2
3	Alice	30	Alice789@gmail.com	X3

The "Metadata" column is irrelevant for customer analysis and should be removed.

6.2.2 Missing Data Imputation

Incomplete data represents a significant and prevalent challenge in real-world datasets that researchers and analysts often face. This occurrence may stem from various factors, including errors that occur during the data gathering process, malfunctioning sensors that fail to record information accurately, or the unavailability of specific information when needed. Neglecting to address missing data can result in skewed or partial analyses that ultimately lead to misleading conclusions. Multiple methodologies and techniques exist for managing missing data, including imputation methods, data augmentation strategies, and complete case analysis. Each of these approaches has its own strengths and weaknesses that must be considered carefully depending on the context of the analysis and the extent of the missing information.

a. **Removing Missing Data**

When the dataset in question contains only a small number of missing values, it may often be considered acceptable to eliminate the corresponding rows or columns from the analysis without encountering any significant repercussions or negative consequences for the overall outcome. Nevertheless, one must intelligently recognize and appreciate that this method carries with it the inherent risk of discarding potentially valuable information, which could play a crucial role in the overall analytical process and the insights derived from it. Therefore, while this approach may seem convenient in the short term, it is absolutely imperative to carefully weigh the decision and thoroughly consider the potential impacts it might have on the results, interpretations, and conclusions drawn from the data. This thoughtful consideration is essential to ensure the integrity and reliability of the analytical findings.

Example:

Consider a dataset with missing values.

Customer id	Name	Age	Email
1	John	25	John.54@gmail.com
2	Jane		Jae34@gmail.com
3	Alice	29	

Removing rows with missing values would result in:

Customer id	Name	Age	Email
1	John	25	John.54@gmail.com

b. **Imputation**

Imputation refers to the specialized and systematic process of replacing missing or absent values in a given dataset with estimates that are derived or computed based on existing, available data. This vital and critical procedure is frequently employed and applied in statistical analysis and data science to significantly enhance the overall quality of datasets. Its primary purpose is to prevent any potential biases or inaccuracies that may arise from simply ignoring these missing values during analysis. To achieve this, various advanced methods of imputation have been developed, each specifically designed to maintain the integrity and usability of the data, thereby ensuring more accurate and reliable analysis and results. Some widely used methods include:

1. **Mean/Median/Mode Imputation**

   Substituting absent values in a dataset with the statistical measures of central tendency—specifically focusing on the mean, median, or mode—of the respective column serves as a commonly used imputation technique in the crucial step of data preprocessing. This methodological approach allows for the retention of data integrity while effectively mitigating the potential biases that may arise from the exclusion of entire records due to these missing elements within the dataset. Each of these statistical measures carries distinct implications for the analysis: the mean, while useful, is particularly sensitive to extreme or outlier values, which can skew results; conversely, the median offers a greater level of robustness against such outliers, providing a more reliable central value in the presence of skewed distributions. Meanwhile, the mode reflects not just any value but the most frequently occurring value in the dataset, which can provide insights into the data's characteristics and trends. Therefore, selecting the most appropriate substitute value for the missing elements can significantly influence subsequent analyses and the interpretations that arise from them, making it crucial to thoroughly consider the distribution characteristics of the data in question. Ultimately, understanding the nuances among these measures is essential for achieving accurate and meaningful outcomes in data analysis.

   Example:

   Consider a dataset with missing age values.

Customer id	Name	Age	Email
1	John	28	John.54@gmail.com
2	Jane		Jae34@gmail.com
3	Alice	29	Alice.b34@gmail.com

The mean age is (28 + 29) / 2 = 28.5. Imputing the missing value with the mean:

Customer id	Name	Age	Email
1	John	28	John.54@gmail.com
2	Jane	28.5	Jae34@gmail.com
3	Alice	29	Alice.b34@gmail.com

2. K-Nearest Neighbors (KNN) Imputation

Utilizing the values acquired from the nearest neighbors within a specific dataset enables researchers and analysts to accurately infer and predict data points that are currently absent or missing from the comprehensive analysis. This advanced methodology greatly enhances the capability to achieve a more thorough and holistic understanding of the dataset by effectively addressing, filling in, and rectifying the gaps in the existing information. By implementing this innovative technique, one can significantly improve data continuity and integrity, which ultimately leads to richer, more detailed, and more reliable insights. As a result, the overall analytical process benefits immensely, ensuring that conclusions drawn from the dataset are well-informed and comprehensive. Through this approach, the potential for uncovering valuable patterns and trends within the data is markedly increased, boosting the overall effectiveness of analysis and decision-making.

Example:

Consider a dataset with missing age values.

Customer id	Name	Age	Email
1	John	28	John.54@gmail.com
2	Jane		Jae34@gmail.com
3	Alice	29	Alice.b34@gmail.com

Using KNN imputation, the missing age for Jane is estimated based on the ages of the nearest neighbors (John and Alice).

3. Regression Imputation

Utilizing regression models is fundamentally crucial for achieving precise and highly accurate estimations of values that might potentially be absent or inadequately represented in the existing dataset. This methodological approach plays a significant role in ensuring both the completeness of the data and its overall reliability. By effectively leveraging these models, one can proficiently predict and fill in the gaps that might otherwise compromise the integrity of the data analysis process. Through this method, analysts can improve their insights and enhance the robustness of their conclusions, ultimately leading to more informed decision-making and strategic planning that can benefit various applications across different fields.

Example:

Consider a dataset with missing age values.

Customer id	Name	Age	Email
1	John	28	John.54@gmail.com
2	Jane		Jae34@gmail.com
3	Alice	29	Alice.b34@gmail.com

A regression model can be trained on the available data to predict the missing age for Jane Smith.

4. **Forward/Backward Fill**

   In the context of analyzing time-series data, the intricate process of imputing missing values can be effectively carried out by employing either the value that directly precedes the missing entry or the one that immediately follows it in the established chronological sequence of values. This particular method is advantageous as it not only allows for a more robust and reliable estimation of the missing data points but also plays a critical role in maintaining the overall integrity and coherence of the entire dataset. By utilizing the surrounding values, we significantly minimize the adverse impact of missing data on the analysis, thereby ensuring that the resultant dataset remains as complete and informative as possible, ultimately enhancing the quality of insights derived from the data. This practice is essential in various analytical frameworks where consistency and accuracy are paramount, assisting researchers and analysts in making well-informed decisions based on comprehensive information.

   Example:

   Consider a time-series dataset with missing values.

Data	Value
2023-01-01	10
2023-01-02	
2023-01-03	12

Using forward fill:

Data	Value
2023-01-01	10
2023-01-02	10
2023-01-03	12

6.2.3 Noise Elimination

Noise is defined by erratic or extraneous variations that can greatly disrupt the integrity of any given dataset, which can considerably hinder the identification of significant patterns and relationships within the data being analyzed. Such noise may arise from a wide array of sources, including inaccuracies in measurement instrumentation, human errors that occur during the data entry process, or even influences from external environmental conditions that may affect the information collected. Addressing and removing noise from a dataset is absolutely critical for significantly enhancing the overall quality and reliability of the data that we work with and analyze on a routine basis. Several common and effective techniques and methodologies for successful noise reduction include:

a. **Smoothing**

   Utilizing a diverse range of various algorithms, such as moving averages in conjunction with low-pass filters, plays a significant role in effectively reducing and mitigating the noise that frequently emerges in time-series data. This crucial process ultimately results in much clearer insights and a notably more accurate analysis overall. By ensuring that the data we depend on is more reliable and useful, we enhance its value for informed decision-making. When we apply these methods thoughtfully, we can extract essential patterns and trends, promoting better understanding and facilitating more effective strategic planning in various contexts.

Example:

Consider a time-series dataset with noise.

Data	Value
2023-01-01	10
2023-01-02	12
2023-01-03	15
2023-01-04	11
2023-01-05	14

Applying a moving average with a window size of 3:

Data	Value
2023-01-01	10
2023-01-02	12
2023-01-03	12.33
2023-01-04	12.67
2023-01-05	13.33

b. Binning

The process of categorizing data into discrete intervals, which is commonly referred to as bins, entails the systematic organization of data points into well-defined ranges. This method is essential for structuring dispersed information in a way that is both coherent and insightful. Following this categorization, we subsequently substitute individual values with the mean or median of those specific bins. This approach effectively serves to minimize variability within the entire dataset, leading to a more uniform set of data. As a result, it enables clearer insights and interpretations by significantly reducing noise and emphasizing noticeable trends. By incorporating this method into our analysis, we enhance the overall quality and reliability of the findings to a considerable extent. This intricately structured process ultimately aids in improving decision-making and analytical outcomes.

Example:

Consider a dataset with age values.

Customer id	Name	Age
1	John	28
2	Jane	34
3	Alice	29
4	Bob	35
5	Carol	30

Grouping ages into bins of 10 years and replacing with the bin mean:

Customer id	Name	Age	Binned age
1	John	28	30
2	Jane	34	35
3	Alice	29	30
4	Bob	35	35
5	Carol	30	30

Outlier Detection

The process of identifying and effectively eliminating outliers, which are those specific data points that exhibit a significant and often notable deviation from the overall dataset, is critical and essential for maintaining the integrity of the data within any analytical framework. Various methods, including the Z-score method, the Interquartile Range (IQR) technique, and even advanced clustering algorithms, can be effectively employed to detect, analyze, and manage these outliers efficiently and accurately. By applying these diverse methods, analysts can significantly enhance the overall quality of their data analysis processes and ensure that their results are not only more reliable but also more meaningful in the context of the particular study or project. This rigorous approach to outlier detection ultimately leads to improved decision-making based on a more accurate understanding of the dataset and its underlying patterns.

Example:

Consider a dataset with age values:

Customer id	Name	Age
1	John	28
2	Jane	34
3	Alice	29
4	Bob	35
5	Carol	100

The age 100 is an outlier. Using the IQR method, we can identify and remove this outlier.

6.2.4 Feature Selection and Dimensionality Reduction

Feature selection and dimensionality reduction are fundamental methodologies that are widely employed in the realm of data analysis, particularly to significantly decrease the quantity of input variables found in a dataset. This practice is critical and plays a vital role due to the numerous complications that can arise from dealing with high-dimensional data, which often lead to issues such as overfitting, increased computational demands, and hindered interpretability of the model. By utilizing these essential techniques, analysts can effectively streamline the data, improving model performance and making the results much easier to understand. In this way, dimensionality reduction not only simplifies the data management process but also enriches the quality of insights that can be derived. Ultimately, these improvements contribute to more effective and informed data-driven decision-making, allowing organizations and researchers to make better choices based on their findings.

a. Feature Selection

Feature selection is an essential and vital process that involves not just identifying but also carefully choosing a well-defined subset of the most relevant and pertinent features that will be utilized effectively during the model training phase. Typical approaches that are frequently employed in this intricate process encompass:

1. **Filter Methods**

   Through the application of a broad spectrum of diverse statistical methodologies, which include, but are not limited to, correlation coefficients, chi-square exercises, and assessments of mutual information, one can proficiently and effectively rank a wide array of various features that may be critical to research and analysis. This systematic and comprehensive approach ultimately

culminates in the identification of the most effective features that warrant additional evaluation and deeper scrutiny for a more nuanced understanding. By leveraging these robust analytical techniques, researchers are significantly better equipped to discern and understand the features that exert the greatest and most pertinent influence on the desired outcomes. This rigorous process not only facilitates more accurate analyses but also leads to more informed and insightful decisions that can drive successful results in various fields of study.

Example:

Consider a dataset with multiple features.

Customer id	Age	Income	Spending score	Email
1	28	50000	85	John.32@gmail.com
2	34	60000	92	Smith354@gmail.com
3	25	55000	80	Alice.34@gmail.com

Using correlation to select features, we might find that "Age" and "Income" are highly correlated with "Spending Score".

2. **Wrapper Methods**

Evaluating a multitude of different combinations of features through the employment of a sophisticated machine learning model entails a thorough and comprehensive understanding of the specific subsets that are capable of generating the most optimal performance outcomes. This intricate and detailed process can be exemplified in numerous scenarios that involve the implementation of advanced techniques, such as recursive feature elimination, which systematically identifies and eliminates the less important features from consideration. This particular approach not only streamlines the selection process significantly but also ultimately enhances the model's predictive accuracy while concurrently improving efficiency in data processing on a substantial level. By meticulously fine-tuning the selection of features, practitioners and data scientists can ensure that the most relevant and impactful data is utilized, which leads to more reliable predictions and consistently better overall results in diverse machine learning applications across different fields and industries.

Example:

Utilizing the highly effective method of recursive feature elimination plays an especially crucial role in the precise identification of the most optimal and beneficial subset of features that can significantly enhance the overall accuracy of a predictive model across various contexts and applications. This meticulous process is essential for improving model performance and ensuring far more reliable and consistent predictions in a wide range of scenarios.

3. **Embedded Methods**

The process of feature selection is an essential and critical component of various model training methodologies that are employed widely in the field of data science and machine learning. This significant process encompasses a diverse range of techniques, including popular and well-known methods such as Lasso regression and decision trees, which are recognized for their efficacy. It involves systematically identifying, evaluating, and meticulously choosing the most relevant variables that significantly contribute to the predictive power and capability of the model. By addressing these aspects, it greatly enhances the overall performance and interpretability of the

model, which in turn allows practitioners and data scientists to make more informed decisions based on the valuable results and insights derived from the thorough analysis. Ultimately, effective feature selection can lead to enhanced model accuracy, efficiency, and robustness, proving to be a pivotal aspect of the modeling process.

Example:

Lasso regression serves as a highly valuable and effective technique for feature selection in statistical modeling by systematically reducing certain coefficients in an intelligent manner to exactly zero. This process facilitates the identification and emphasis of the most significant features present within a dataset. By focusing only on these key features, Lasso regression improves model interpretation and enhances predictive performance.

b. Dimensionality Reduction

Dimensionality reduction is a highly specialized and essential process that involves the careful transformation of complex and high-dimensional data into a lower-dimensional space characterized by a significantly reduced number of dimensions. This transformation is accomplished while ensuring that the original structure, patterns, and relationships inherent within the data are preserved to the greatest extent possible. Various common methodologies and techniques that are frequently utilized in this nuanced process include:

1. Principal Component Analysis (PCA)

A linear transformation that effectively and efficiently maps an array of data points onto orthogonal axes, which are specifically referred to as principal components, is precisely designed to maximize the representation of variance that is found within the entire dataset. This innovative and well-structured approach not only allows for a much clearer and more comprehensive understanding of the underlying structure of the data but also facilitates significant dimensionality reduction while still preserving the most relevant and crucial information that is necessary for thorough and rigorous analysis. By employing this highly effective method, it becomes considerably easier to visualize complex and multifaceted data, thus allowing analysts to identify patterns, relationships, and anomalies that would otherwise remain obscured and hidden in the higher dimensions of the dataset. This proves to be incredibly valuable in various fields where data interpretation and understanding are critical for informed decision-making processes. Additionally, this transformation aids in reducing computational costs and improving efficiency in handling large datasets, thereby enhancing the overall analytical capabilities of the involved systems and methodologies.

Example:

Utilizing Principal Component Analysis (PCA) on a dataset characterized by a myriad of features and numerous variables effectively serves to considerably diminish its dimensional complexity. Such a substantial reduction in dimensionality creates a situation that becomes markedly easier to analyze and interpret the data at hand. This streamlined process allows researchers and analysts to extract clearer insights and gain a deeper understanding from the information that is presented, thereby facilitating a more effective analysis of the dataset and leading to potentially valuable conclusions based on the findings.

2. t-Distributed Stochastic Neighbor Embedding (t-SNE)

A highly advanced and sophisticated approach that effectively utilizes a comprehensive range of complex and intricate non-linear principles to accurately represent a diverse and varied array of intricate data sets. This meticulous and detailed methodology is particularly well-suited to handling and processing data characterized by significant high dimensionality, presenting it in visually engaging and impactful two-dimensional or even three-dimensional formats. By incorporating such advanced techniques and frameworks into the data processing pipeline, this innovative approach ultimately enhances both comprehension and understanding of the information when it is presented to the audience. It makes the data substantially more accessible and considerably easier to interpret and analyze, regardless of the varying and often challenging contexts in which it may be encountered or utilized. Such a method offers substantial benefits when it comes to interpreting the complexities inherent in modern data sets, providing clear insights and fostering a deeper comprehension of the data relationships, trends, and patterns that may not be immediately obvious. As a result, this expanded capability not only aids analysts in their work but also enables stakeholders to make informed decisions based on comprehensive analyses, bridging the gap between intricate data complexities and practical, actionable insights that are valuable across numerous applications and industries.

Example:

Utilizing t-Distributed Stochastic Neighbour Embedding (t-SNE) serves as a powerful means for visualizing clusters that exist within high-dimensional datasets. This advanced dimensionality reduction technique is remarkably effective in retaining local similarities, all while successfully separating distinct data clusters. Such capabilities enable a clearer and more insightful interpretation of the complex relationships that are inherent in the data being analyzed. The t-SNE algorithm adeptly captures the intricate structure of the data by transforming similarities into probabilities, which significantly aids in uncovering the underlying geometric structure that resides within the high-dimensional space when it is projected into lower dimensions. This process not only enhances data visualization but also supports more informed decision-making based on the revealed patterns and structures.

3. Autoencoders

Models that utilize advanced neural networks are specifically crafted to effectively gain a much more compact and efficient representation of intricate and nuanced data through their complex and sophisticated learning processes. These insightful learning processes meticulously take into account a wide variety of dimensions, attributes, and relationships inherent within the data, allowing for a much deeper understanding and insightful extraction of essential information that can be widely utilized across various applications and contexts. By leveraging these advanced techniques and methodologies, we can achieve remarkable and superior performance in critical tasks such as classification, prediction, and pattern recognition, ultimately leading to more informed decisions and enhanced outcomes in numerous fields, industries, and specialized domains. As a result, these models are becoming indispensable tools for researchers, businesses, and practitioners alike, enabling them to harness the power of data to drive innovation, efficiency, and effectiveness in their respective areas of work.

Example:

Utilizing an autoencoder for the clear and specific objective of effectively achieving dimensionality reduction across a vast array of diverse types of image data sets proves to be exceptionally advantageous for various applications in data analysis and machine learning. The ability to compress high-dimensional data into lower dimensions while retaining essential features is a significant benefit in many fields.

c. Normalization

Normalization refers to the method of systematically adjusting numerical features in datasets so that they conform to a standardized, specified range. This important process is exceptionally crucial for algorithms that exhibit a significant sensitivity to the magnitude of input variables, particularly those that utilize gradient-based optimization techniques or distance-dependent algorithms. Normalization ensures that each feature contributes equally to the distance measures and calculations, thereby significantly enhancing the overall performance of various machine learning models. Widely employed normalization methods include Min-Max scaling, Z-score standardization, and robust scaling, among others. Each of these prevalent methods has its own unique advantages, depending on the specific characteristics of the data being analyzed and the particular requirements of the model in use. By implementing normalization, data scientists can improve model accuracy and training speed, leading to more efficient and effective computational processes in the realm of machine learning.

a. Min-Max Scaling

Scaling variables to a standardized range, which is frequently defined as the interval [0, 1], is an extensively recognized and widely adopted preprocessing method found in the expansive realm of data analysis and statistics. This effective and essential technique aims to ensure that all features contribute equally and fairly to the entire analysis process by systematically normalizing the data within a clearly defined and standardized interval. By performing this normalization, it not only helps mitigate the risk of bias that could potentially emerge from disparities in the scale of the features, but it also significantly enhances the overall performance of various algorithms that might be sensitive to the magnitudes of the input values being utilized. Furthermore, this strategic normalization facilitates more accurate comparisons among features, allowing for a more reliable and consistent interpretation of the results derived from subsequent analyses. Ultimately, this critical process plays a crucial role in refining the quality of insights that can be extracted from the data, leading to more informed and strategic decision-making based on the outcomes of the analytical methods employed. Through this meticulous process, analysts can ensure a level playing field for all features in their datasets, thereby optimizing the effectiveness and accuracy of their data-driven conclusions.

Example:

Consider a dataset with age values.

Customer id	**Name**	**Age**
1	John	28
2	Smith	34
3	Brown	29

Applying min-max scaling:

Customer id	Name	Age	Scaled Age
1	John	28	0.0
2	Smith	34	1.0
3	Brown	29	0.1667

b. Z-Score Normalization

Standardizing features is a critical process that involves adjusting data values to achieve a mean of 0 and a standard deviation of 1. This essential procedure typically entails subtracting the mean from each individual feature value and subsequently dividing the result by the standard deviation. By implementing these adjustments, we significantly enhance the comparability of different features, which is particularly important when these features are measured on varied and inconsistent scales. Such normalization is crucial across a wide range of analytical models, as it can lead to improved convergence rates in various optimization algorithms. Moreover, this standardization contributes to more reliable and valid results in the realm of predictive analytics, enabling better performance of machine learning models and enhancing their effectiveness. By ensuring that the data is on a consistent scale, we create a solid foundation for further analysis, allowing us to derive meaningful insights and make informed decisions based on the data at hand.

Example:

Consider a dataset with age values:

Customer id	Name	Age
1	John	28
2	Smith	34
3	Brown	29

Applying Z-score normalization:

Customer id	Name	Age	Scaled Age
1	John	28	-1.2247
2	Smith	34	1.2247
3	Brown	29	0.0

c. Robust Scaling

By employing both the median and the interquartile range for the critical and important purpose of feature scaling, we can significantly reduce the overall impact that outliers present in the dataset might have on our analysis. This specific and well-established methodology effectively mitigates the potential distortion and bias that extreme values can introduce into the dataset, thereby greatly enhancing the robustness, accuracy, and overall reliability of the analysis that is conducted. As a direct result of utilizing these techniques, this analytical approach leads to the achievement of results that are more dependable, consistent, and trustworthy across a variety of analytical contexts and applications. This ultimately fosters a greater level of confidence in the conclusions drawn from the data, allowing researchers and analysts to make informed decisions based on msore stable and less skewed insights derived from their datasets.

Example:

Consider a dataset with age values.

Customer id	Name	Age
1	John	28
2	Smith	34
3	Brown	29

Applying robust scaling:

Customer id	Name	Age	Scaled Age
1	John	28	-0.5
2	Smith	34	1.5
3	Brown	29	0.0

Conclusion:

Data preprocessing serves as a critical foundational stage in any comprehensive project that relies on data for insightful analysis and relevant interpretation. This essential process involves a multitude of techniques such as data cleaning, which plays a crucial role in ensuring that the dataset is completely free from errors and inconsistencies, effectively addressing missing values through various methods of imputation or by the removal of incomplete records. Additionally, it involves the removal of irrelevant noise that could potentially skew results and mislead analysts. Selecting pertinent features that are specifically relevant to the problem at hand is equally important, as it helps to focus the analysis on the most impactful variables. Moreover, reducing dimensionality is a key step that serves to simplify the dataset, making it more manageable and easier to interpret. Another fundamental aspect includes normalizing the data to ensure that it fits well within a specific range, thereby improving the accuracy of algorithms that depend on such scaled values. Collectively, these essential practices prepare the dataset effectively for subsequent and reliable analysis, as well as for efficient modeling. They contribute not only to the enhancement of overall data quality but also significantly improve the performance and interpretability of various machine learning models. Therefore, dedicating adequate time and resources to this vital and indispensable phase of data preprocessing is absolutely critical for the successful derivation of accurate, reliable, and ultimately valuable insights from the dataset, allowing researchers and analysts to make well-informed decisions based on solid evidence.

CHAPTER 7

INTRODUCTION TO DATA STORAGE

Sumit Chopra[1], Debjit Mohapatra[2] and Navjot Kaur Basra[3]

[1,2,3]GNA University, Phagwara

7.1 Overview

Data storage is one of the key components of data engineering as it helps one interact between data creation with data flow, data management and data retrieval. With the rise in data volume as well as data formats, secure storage systems contribute to maintaining and ensuring quality and accessibility for respective data.

Storage plays an important role in the management and organization of large amounts of data sets in a way that eases the processing and analysis. It can be structured like SQL databases, semi-structured such as JSON, Avro, Parquet, or unstructured like logs, images, and videos. Therefore, engineers build huge storage solutions that guarantee their scalability, reliability, and performance in distributed environments. Cloud-based storage solutions such as Amazon S3, Google Cloud Storage, and data lakes became essential for the big data handling. Properly storing data is a must in our world to utilize the value of data for more sophisticated data analysis.

Table 5: Key Definitions and Concepts Related to Data Storage

Term	Definition
Data	Raw, unprocessed information that can be processed to generate meaningful insights.
Database	An organized collection of interrelated data items that facilitates efficient storage and retrieval of information.
Schema	A structured framework or blueprint defining the organization of data within a database.
Entity-Relationship Diagram (ER Diagram)	A visual representation of the relationships among different tables within a database.

7.1.1 Importance of efficient data storage

The storage of data efficiently is important for large-scale data systems in terms of fast access, scalability, and costs. All fulfill optimal performance through fast data retrieval and processing, contributing a great deal to analytics and real-time applications. Oriented properly, storage solutions favor data integrity, security, and regulatory compliance. It's also a good management process which ensures disaster recovery that provides business continuity and the prevention of data loss. Data volume just keeps growing; a well-optimizing balancing act between performance on one side and reliability with cost needs to be preserved always in a modern data-driven environment that further elaborate in fig.7.1.

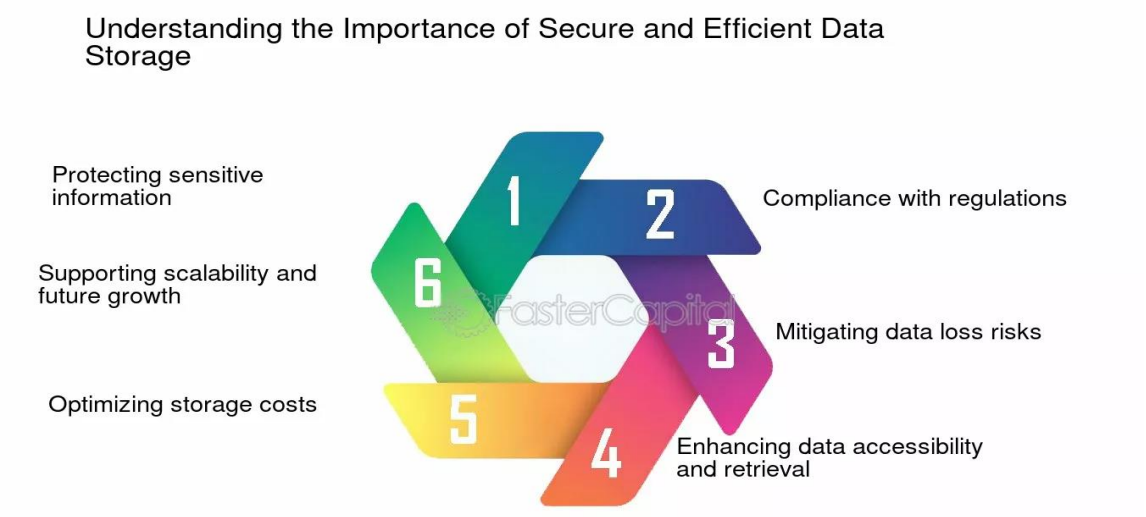


Fig. 7.1: Importance of Secure and Efficient Data

a. **Protecting Sensitive Information**: Organizations handle huge amounts of confidential information, such as personally identifiable information (PII), financial records, and intellectual property, making strong security a critical aspect to take care of.

- Encryption is a process of transforming intelligible plaintext into an unintelligible cipher text, readable by the intended receivers only. To encrypt a message, there are two things required, algorithm and key. Based on the number of keys used, encryption generally falls into two classes: first, symmetric or private key encryption, where the same key is employed at both the sending and receiving side; and second, asymmetric or public key encryption, where there are two different keys used on the sending and receiving side.
- Access control forms one of the techniques of security for providing integrity and confidentiality. Its main task is to regulate the sharing of resources or information. Access control really refers to whether certain user commands have rights to some specific data. Access control policies define the users' permission in order to provide security.

b. **Compliance with regulations:** Data compliance management involves every practice, process, and policy that takes place during the data compliance lifecycle. This assures that your organization understands and acts upon data protection laws, privacy regulations, industry standards, and internal policies that are relevant to them.

Its purpose is to ensure compliance with various data privacy legislation, such as GDPR in Europe, CCPA in California, or industry-specific standards like HIPAA in the field of healthcare. The intent of this data compliance management is to protect sensitive data assets, maintain their integrity and security, and avoid legal or financial penalties.

c. **Mitigating Data loss risks:** Understanding the causes and implications of data loss is essential for organizations to enhance data security and protect against financial and reputational damage. Implementing robust backup solutions, advanced data security measures, and employee training programs are key strategies to prevent data loss incidents.

d. **Enhancing Data Accessibility and Retrieval:** Data accessibility defines the ease with which users within an organization may extract and apply relevant data. To ensure all employees who

require data to do their jobs can retrieve and use it, data must be easy to access, regardless of an employee's technical skills.

e. **Optimizing Storage cost:** Data storage costs refer to the expenses involved in keeping data on various storage media-such as hard drives, solid-state drives (SSDs), cloud storage, and tape storage. Different factors will influence these costs, such as the data volume, the type of storage media utilized, data access frequency, and the redundancy level required. As unstructured data continues to be emitted in ever-increasing quantities, the cost of its storage remains one of the most tremendously important considerations for many organizations.

f. **Scalability and Future Growth:** As the volume of data keeps growing rapidly, scalable data storage and processing systems are a requisite for future-proofing data engineering pipelines. Designing for scalability involves creating systems capable of adapting to growing workloads without deliberating performance bottlenecks or exorbitant costs.

- **Cloud-Based Storage & Data Lakes -** Utilizing scalable cloud storage solutions like AWS S3, Google Cloud Storage, and Azure Blob Storage set data lakes on top and enable flexible management of huge data sets.
- **Distributed & Scalable Databases -** Employ distributed databases such as Apache Cassandra, Amazon DynamoDB, or Google Bigtable, and horizontal scaling can be realized for growth in data.

7.1.2 Types of Data:

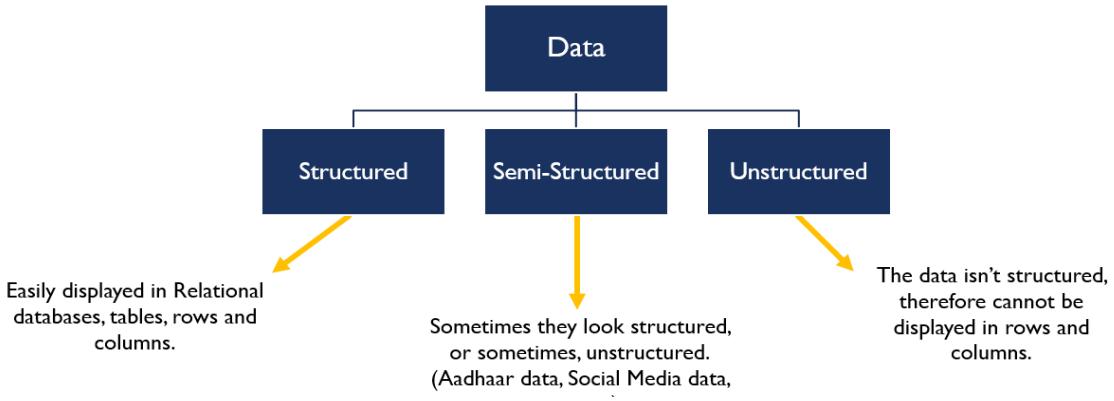


Fig. 7.2: Types of data

a. **Structured Data:** This describes data which is inflexible and rigid, most commonly fixed into tabular forms, and made suitable to be stored in data warehouses, spreadsheets, and especially relational databases. This means that it is organized according to a specific schema model, which is basically a grid of cells structured into rows and columns that serves to create easy entry, search, retrieval, and analysis of data thematically related.

Features of Structured Data:

- Predefined format: arranged within fixed-column tables of data.
- Easily searchable: can be queried through SQL.
- Highly structured: stored within a relational database, e.g., MySQL, PostgreSQL and MS SQL Server(not possible with large amounts of unstructured data).
- Efficient processing: fast retrieval supported by access paths.

- Scalable: while increasing in data, it will be maintained in structure by adding new rows or new columns.

b. Semi-Structured Data: Semi-structured data lacks a strict tabular form as with structured data; however, the data is further organized using tags, markers, or metadata. It, therefore, lies between structured and unstructured data, containing parts of each one. Approximately organized data, also called loosely organized data, usually has some forms of structure in it (for instance, tags, keys or metadata), even if it does not resemble tabular form at all. Unlike structured data found in relational databases, semi-structured data is generally found in JSON, XML, or NoSQL databases, depending on their flexibility in their format.

Features of semi-structured Data:

- Partially Organized: This is a description often adopted with thelasely or partly organized data. This usually has some structure contained within-the (e.g., tags, keys, or metadata)-although a different form other than tables does not appear.
- Flexible schema: This means that the data structure may vary according to the record, the records may present some form of flexibility rather than the rigidity of schema form.
- Easier to process than unstructured data: This would still take some processing time in contrast to structured data but has some notion of identifiers, allowing parsing and querying to happen.
- Use of either hierarchical or graph-based storage for structuring: Generally, these could either be in some format of JSON, XML, or even NoSQL in nature.
- Querying methods: a querying method of the conventional SQL mode cannot be applied to the processing of this type of data; XPath, XQuery, NoSQL query languages, etc. are tools that carry out the operations.

c. Unstructured Data: Unstructured data refers to any information that has no defined model, format, or structure. Unstructured data is raw information thus need to apply advanced techniques for further analysis- including Machine learning, NLP, and AI. This, when put together with structured data usually living within relational databases, or semi-structured data, like those having some form of organization (for instance XML or JSON), best describes, in general, the analysis of hidden unstructured data, which has a more advanced form.

Features of Unstructured Data:

- No Fixed Format: This means lack of defined rules, which in essence thus will logically not allow it to be stored in traditional databases.
- Diverse and Complex: Information could be in the form of texts, pictures, videos, or audios. High volume and Growth- Accounts for over 80% of the world's store of data, and it is constantly growing. ·
- Difficult to Query and Analyze: Artificial Intelligence, Natural Language Processing, and Deep Learning need to be used by complex tools to pull out the data that actually makes sense.
- Dense with Information: Such documents contain important indications concealed within the raw text, speech, or multimedia content.

7.1.3 On-Premise vs. Cloud Storage in Data Engineering

Data engineers must decide between on-premise and cloud storage based on the scale of size, cost, security, and maintenance. Each of the options has advantages and certain trade-offs depending on the business needs and infrastructure capabilities.

a. **On-Premise Storage** It involves hosting data within an organization's own data center, demanding additional hardware, maintenance, and management of security. A list of pros and cons concerning cloud storage includes:

Advantages:

- Full Control: Organization has total control over data security, access, and configurations.
- Performance & Low Latency: Faster data access within internal networks hence beneficial for high performance applications.
- Regulatory Compliance: It helps meet the strict sovereignty and privacy regulations for data.
- Predictable Costs: Fix infrastructure investments without fluctuating cloud price

Disadvantages

- High Initial Costs: It requires a larger upfront in hardware, cooling, and infrastructure.
- Limited Scalability: To expand storage capacity requires the purchasing of additional hardware and the setting up of additional hardware.
- Maintenance & Management: Requires dedicated IT teams for updates, security, and massive general disaster recoveries.

b. **Cloud storage** It refers to a third-party network that the storage cloud provides, for example, AWS S3, Google Cloud Storage, Azure Blob Storage, etc., that enables modern flexible and scalable storage. A list of pros and cons concerning cloud storage includes:

Advantages

- Scalability & Flexibility: Each Cloud service is able to scale big storage up and down as needed, and does not require the upfront investment.
- Cost-effective: Pay-as-you-go pricing models allow cost to vary due to load.
- Automated Maintenance: Security, updates, and system reliability are handled by the cloud provider.
- Disaster Recovery & High Availability: Built-in redundancy and global distribution enhance data protection.

Disadvantages

- Data Security & Compliance: Sensitive data has to be extremely well encrypted with good access controls to meet some regulations.
- Network Latency: Big workloads might slower data retrieval as compared to on-premise storage.
- Ongoing Costs: The long-term cost of cloud creates concerns when the volume of transfer and storage usage remains high.

7.2 Databases

Database is a structured system for storing data, which allows for effective data management, retrieval, and processing for various applications. These databases are deemed important for large-scale datasets,

where data consistency should be ensured while optimizing performance for analytics, transactions, and reporting. Types of databases: Relational Databases (RDBMS), NOSQL databases.

7.2.1 Relational Databases (RDBMS)

A relational database is simply a type of database in which data points are related. Relational databases are based on the relational model that somewhat intuitively and easily represents the data in a tabular form. Rows in a relational database represent records; Columns of the table hold attributes of the data. Each record is often provided with a value for each attribute thus; it becomes easy for one to know the relationship between the data points.

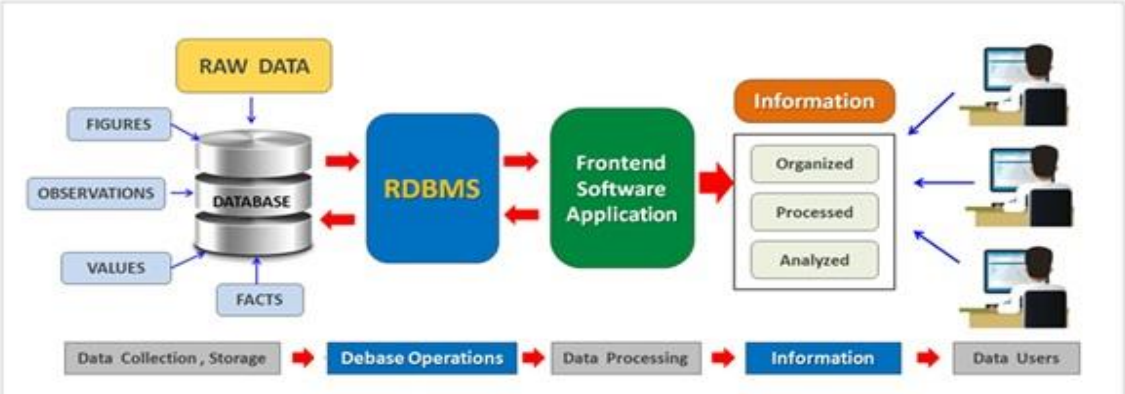


Fig. 7.3: Relational databases

The relational database was developed in the 1970s by E.F. Codd of IBM, allowing in return any table to be related to another table using a common attribute. Other than using hierarchical structures to organize data, Codd proposed a refocus into the data model where data should be stored, approached, and related in tables without reorganizing tables containing that data. Visualize the relational database as varied collections of spreadsheet files to help organize, manage and relate to data in the right concepts of businesses. In this sense, every "spreadsheet" in the relational database model is a table storing information in columns (attributes) and rows (records/tuples). Each of these attributes (colunms) defines a data type, and a record or row contains the value of the specific data type. All tables in a relational database have a field called a primary key, which is the unique identifier of a row, while each row can be used to create a relationship between different tables using a foreign key-reference to a primary key of another existing table(as shown in fig.7.3).

Components of RDBMS:

A Relational Database Management System (RDBMS) consists of several key components that work together to store, organize, manage, and retrieve structured data efficiently. Below are the essential components of an RDBMS, along with their roles and significance in data engineering.

1. **Tables (Relations)**
   - A table is the fundamental structure in an RDBMS where data is stored.
   - It consists of rows (records) and columns (fields/attributes) with a predefined schema.
   - Each table represents an entity in a relational model (e.g., a "Student" table storing student details).

Example: Student Table

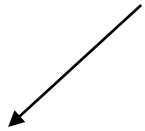
Student_id	Name	Age	Course
1	Sanya	22	IT
2	Prince	21	CS
3	Alice	26	AI

2. **Primary Key**

- **A Primary Key is a unique identifier for each row in a table.**
- **Ensures that no two rows have the same key value, maintaining data integrity and uniqueness.**
- **The primary key column cannot be NULL or duplicate.**

Example: In the "Students" table, Student_ID can be the primary key.

Primary key



Student_id	Name	Age	Course
1	Sanya	22	IT
2	Prince	21	CS
3	Alice	26	AI

3. **Foreign Key (FK)**

- A Foreign Key is a column that establishes a relationship between two tables.
- It refers to the Primary Key of another table, enabling referential integrity.
- Ensures consistency by restricting deletion or modification of referenced data.

Example: A "Courses" table with a Course_ID as a Primary Key can be referenced in the "Students" table as a Foreign Key.

Course_id	Course_Name
C001	Computer Science
C002	Artificial Intelligence
C003	Information Technology

The Foreign Key in "Students" would reference Course_ID from "Courses":

Student_id	Name	Age	Course_id(FK)
1	Sanya	22	C003
2	Prince	21	C001
3	Alice	26	C002

ACID Properties

The primary properties of a relational database management system are as follows. The shortened word for those qualities is ACID. To preserve the integrity of a database management system, these four properties will be adhered to either before or after a transaction occurs in a database.

Atomicity involves operations.

Abort: Database changes are not saved in the database because they are not visible.

Commit: Database modifications are successfully saved and visible.

Consistency: To keep the database system in a consistent state at all times, the integrity constraints must be correctly maintained either before or after a transaction occurs at the database. Integrity limitations include a record's data type, foreign key, and primary key in a database. Therefore, the database should verify that the data is correct when a transaction occurs.

Isolation: Database transactions ought to be isolated from one another. After the transaction is completed successfully, it ought to be made public. If not, a lot of problems would occur. Every transaction that takes place in the database is distinct from every other transaction. It is forbidden for another transaction to access data or records that were performed during a specific transaction.

Durability: When a database is updated, it will stay in the random-access memory (RAM) for a predetermined amount of time before being transferred to the database. Databases should update themselves with the most recent updates in the event of a memory failure to prevent the memory from crashing. This property's only function is to guarantee that updates made through transactions are final until they are altered by authorised individuals.

Advantages of RDBMS:

- **Data Integrity & Security**: Enforces strict data validation rules.
- **Scalability & Performance Optimization**: Supports indexing, partitioning, and replication.
- **Standardized Querying**: Uses SQL, making data extraction efficient and accessible.
- **Multi-User Access & Concurrency Control**: Manages concurrent transactions safely.
- **Business Intelligence & Analytics**:Supports reporting, visualization, and decision-making processes.

Popular RDBMS Example

- **MySQL** – Open-source, widely used for web applications.
- **PostgreSQL** – Advanced, supports JSON, spatial data, and extensibility.
- **Oracle Database** – Enterprise-grade, optimized for high-volume transactions.
- **Microsoft SQL Server** – Integration with Windows ecosystem and enterprise solutions.

7.2.2 NOSQL Databases

A database management system (DBMS) called NoSQL, or "Not Only SQL," is made to manage large amounts of unstructured and semi-structured data. NoSQL databases are perfect for contemporary applications that need real-time data processing because they offer flexible data models and facilitate horizontal scalability, in contrast to traditional relational databases that employ tables and pre-defined schemas.

Why use NoSQL?

NoSQL databases lack a universal query language, in contrast to relational databases, which employ Structured Query Language. Rather, every NoSQL database type usually has a different query language. Strong consistency and organised links between data are ensured by traditional relational databases, which adhere to the ACID (Atomicity, Consistency, Isolation, and Durability) principles.

Types of NOSQL Databases:

1. Key-value: A database type that stores data as a collection of key-value pairs is called a key-value data store. Every data item in this kind of data storage is recognised by a unique key, and the value that goes with that key can be anything—a string, number, object, or even another data structure.

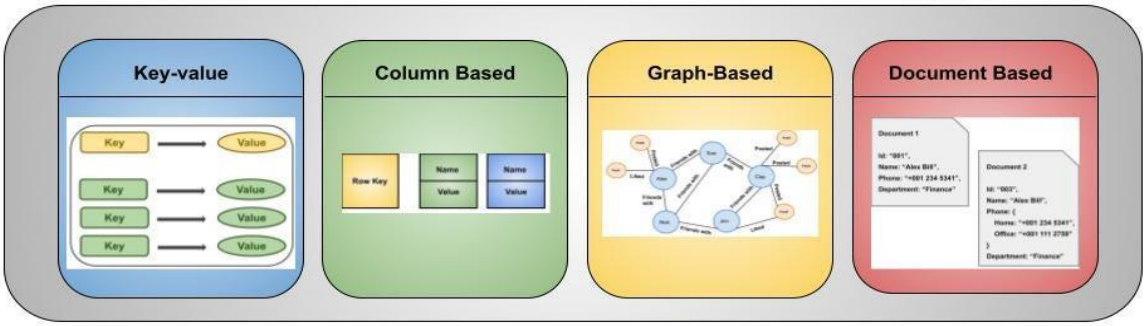


Fig. 7.4: Types of NOSQL Databases

Document based: Information is kept in documents in a document database as elaborate in. Usually, each document is a nested key-value structure. A collection in a document database is comparable to a table in a relational database, with the exception that no one schema is applied to all documents. The values can be atomic data types or more complicated elements like lists, arrays, nested objects, or child collections (example shown in fig.7.5).

Key	Document
101	{ "ID": "1001", "ItemsOrdered":[{ "ItemID": "1", "Quantity": "2", "cost": "1000", }, { "ItemID": "1001", "Quantity": "2", "cost": "1000", }] , "OrderDate": "05/11/2019" }
102	"ID": "1002", "ItemsOrdered":[{ "ItemID": "2890", "Quantity": "11", "cost": "10000", }] "OrderDate": "05/11/2019" }

Fig. 7.5: Document based NOSQL Database

2. **Column Based:** It is a kind of NoSQL database that is very flexible and scalable since it stores data in columns rather than in rows.

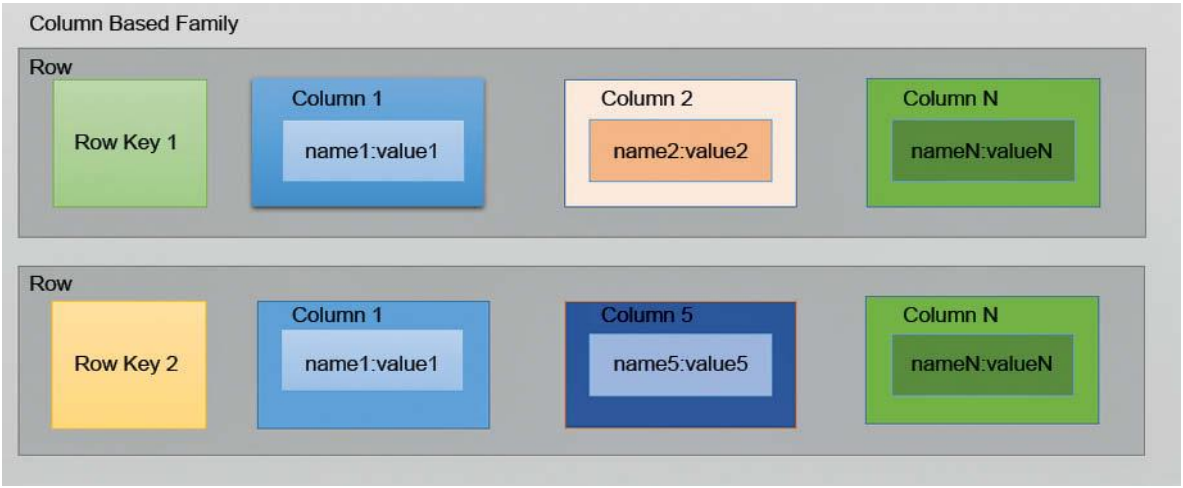


Fig. 7.6: Column based NOSQL Database

Data is arranged into column families—groups of columns with similar characteristics—in a broad column data storage. In a broad column data store, every row is uniquely identified by a row key, and the columns in the row are further divided into column names and values (as shown in fig.7.6).

4. Graph based: Highly linked data can be stored and queried using graph databases. Entities, also called nodes or vertices, and the connections between them, sometimes called edges, can be used to model data. In graph databases, the kind or strength of the relationships also has important implications (as shown in fig.7.7.)

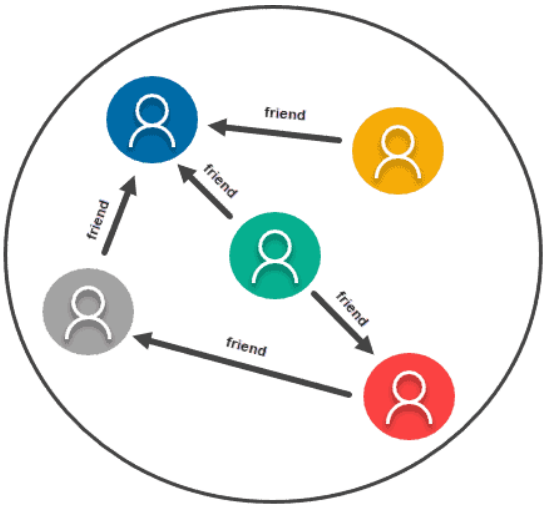


Fig. 7.7: Graph based NOSQL Database

In the current scenario, we are creating mountains of data, which need to be stored and manipulated accurately. Data storage and retrieval are very important in software development, and we have been relying on RDBMS, which performed well in these aspects. It can query and retrieve an extensive amount of data from the database, but it can't handle big data. In recent times, we started creating lots of images and files, which were very bulky, and it became tedious to handle and manipulate these types of data using RDBMS.

The schema, which specifies the constraints for the data, helped RDBMS to store the data properly. This schema specified the way in which the data can be stored, which helped in the fast retrieval of data but was limited to data structures. To solve the limitations of RDBMS with the schema, there are many alternatives like NoSQL and JSON-supported databases, which can easily handle bulky or non-schema data. In the upcoming sections, we will first start with a detailed study of databases. What is data? There are multiple ways to store data; for example, we can store our data in files, and this is the most basic way to store data. Software programs create innumerable files: source files, data files, config files, etc. It became very tedious to handle the data, regardless of how powerful the modern-day computer was, and we needed to search the records very quickly.

7.3 ER diagrams:

ER diagrams:

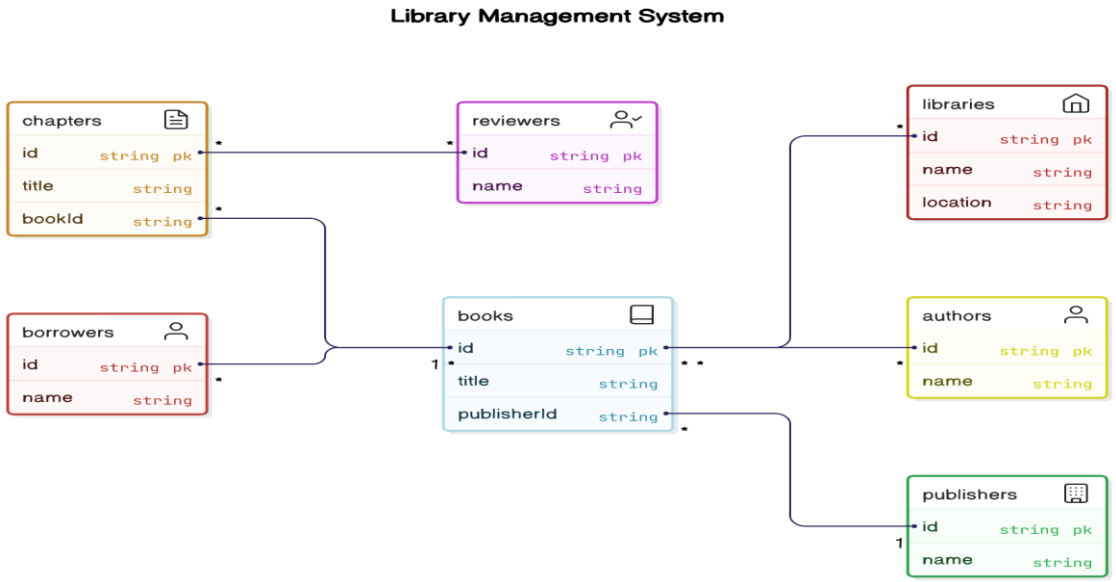
An ER Diagram is a flowchart-like entity that illustrates how entities like individuals, objects, or ideas interact with each other within a system. In general terms, the phrase bibliographic ER diagrams ought to encompass similar tools in other areas of information and engineering systems, research, and education. An ERD or ER Model uses a standardized set of symbols like rectangles, diamonds, ovals, and lines to connect these symbols to represent how entities and relationships with attributes relate to each other. They are similar to grammatical syntax, where the entities are equivalent to nouns and relationships are akin to

verbs. ER diagrams are similar to a data structure diagram (DSD), which is interested in relationships between entities' elements rather than the relationships that occur between entities themselves. ER diagrams are also often utilized with data flow diagrams (DFDs), which graphically depict information flow for processes or systems.

Example: This is an ER diagram for a Library Management System. It depicts entities such as Book, Author, Publisher, Chapter, Reviewer, Library, and Borrower, and their relationships.

Entities and Relationships Modeled:

1. **A Book** can have multiple **Authors**, and an Author can write multiple Books (**M: N**).
2. **A Book** is published by one **Publisher** (**1:M**).
3. **A Book** consists of multiple **Chapters**, but a Chapter belongs to one Book (**1:M**).
4. **A Chapter** can be reviewed by multiple **Reviewers**, and a Reviewer can review multiple Chapters (**M: N**).
5. **A Book** can be stored in multiple **Libraries**, and a Library contains multiple Books (**M: N**).
6. **A Borrower** can borrow multiple Books from a Library (**M: N**).



7.3.1 History of ER diagrams

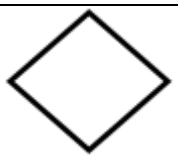
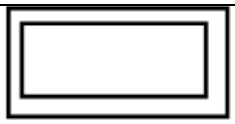
ER modeling for database design was created by Peter Chen in the 1970s. He is now on the faculty at Carnegie-Mellon University in Pittsburgh. In 1976, while working as an assistant professor at the Sloan School of Management at MIT, he wrote one of the only articles on the subject: "The Entity-Relationship Model: Toward a Unified View of Data." A more general conceptualization of the interconnection of things dates back to the writings of philosophers in ancient Greece, particularly Aristotle, Socrates and Plato. It is further modernized by the efforts of 19th and 20th Century philosopher-logicians such as Charles Sanders Peirce and Gottlob Frege. Charles Bachman and A.P.G. Brown were already working with the immediate predecessors of Chen's method by the 1960s and 1970s. Charles Bachman created a form of Data Structure Diagram; a method that was named the Bachman Diagram. Brown wrote about modeling real-world systems. James Martin had ERD improvements. Chen's, Bachman's, Brown's, Martin's and other work also lent itself to the eventual creation of the Unified Modeling Language currently widely used for designing software.

7.3.2 Symbols Used in ER Model

An ER Model is used to model the logical view of the system from the perspective of the data, and it consists of these symbols:

i. **Rectangles** are used to denote entities in the ER Model.
ii. **Ellipses** denote attributes in the ER Model.
iii. **Diamonds** denote relationships between entities.
iv. **Lines** denote attributes to entities and entity sets to other kinds of relationships.
v. **Double ellipses** denote multi-valued attributes.
vi. **Double rectangles** denote weak entities.

Table 6: Representation of various symbol used in ER diagram

Figures	Symbols	Representation
Rectangle		Entities in the ER Model.
Ellipse		Attributes in the ER Model.
Diamond		Relationships between entities.
Lines		Attributes to entities and entity sets to other entity types.
Double ellipse		Multi-valued attributes.
Double rectangle		Weak Entity.

7.3.3 Components of ER Diagram

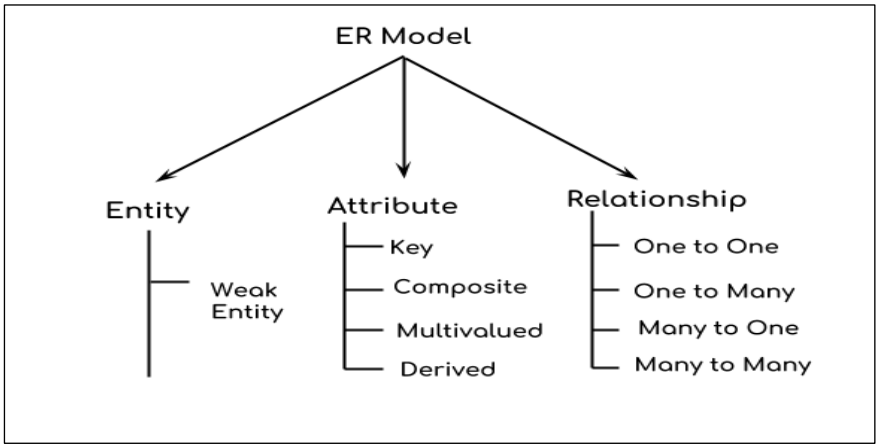
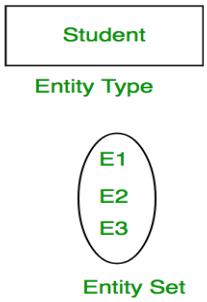


Fig. 7.9: Components of ER Diagram

i. Entity:

An Entity can be an object of physical existence - for instance, a specific individual, vehicle, house, or employee - or it can be an object of conceptual existence, like a company, a position, or a university course.

ii. Entity Set:



An entity is an object of an entity type, and an entity set is a collection of all such entities. For instance, E1 is an entity of the entity type of student, whereas the collection of all students is referred to as the entity set. In a diagram, the entity type is shown by:

An entity set can be shown in an ER Diagram but entities cannot be shown in an ER Diagram because they are columns and rows in the relation and the ER representation is a graphical representation of the data.

Types of Entity:

There are two types of entity:

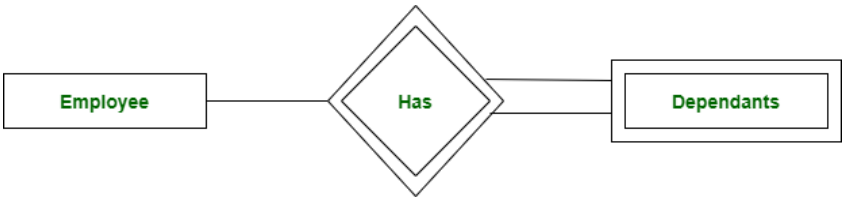
a. Strong Entity:

A Strong Entity is an entity with a primary Attribute. Strong Entity does not rely on any other Entity within the Schema. It contains a primary key, which assists in identifying it uniquely and is shown in the form of a rectangle. These are referred to as Strong Entity Types.

b. Weak Entity:

Some classes of entities there exist for which a set of identifier attributes may not be determinate, capable of uniquely differentiating every instance from others of its type. They are called weak entity types.

Example: There is some personal information (spouse name, children name etc.) for a dependent of the employee for an employer, stored by it. It can never be managed alone since there exist no independent entities named dependent to hold separate meanings. Hence, Dependent would be a Weak Entity Type whereas Employee would be an Identifying Entity type for Dependent i.e. Strong Entity Type. Whereas weak entity types are shown in a double rectangle, the participation of weak entity types is complete. The relationship between the weak entity type and the identifying strong entity type is referred to as an identifying relationship which is shown by using a double diamond.



i. Attributes:

The attributes are utilized to define the entity-type. For example, Roll_No, Name, DOB, Age, Address, or Mobile_No are the attributes that define the entity-type Student. In ER diagram, the attribute is indicated by the oval shape.

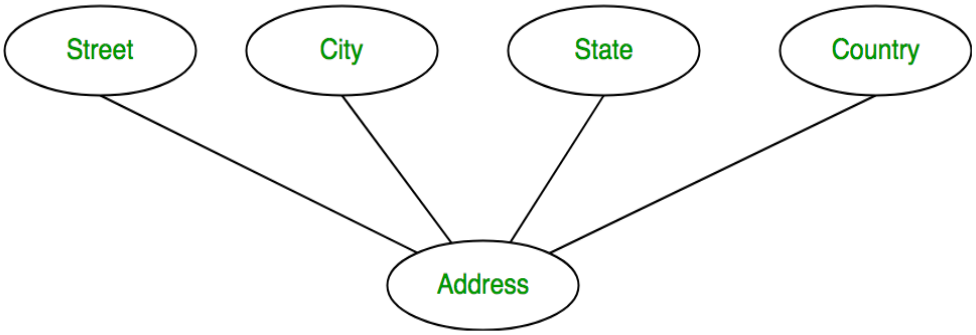
Types of Attributes:

a. Key Attributes:

A property which can individually specify every entity within the entity set is typically referred to as the key attribute. For instance, Roll_No may be unique for every student. A key attribute is represented in the ER diagram by an oval with a background line.

b. Composite Attributes:

A composite attribute contains multiple other attributes. For example, Street, City, State, and Country are all the attributes of the Address attribute of the student Entity type. In an ER diagram, the composite attribute is represented by an oval that contains other ovals.



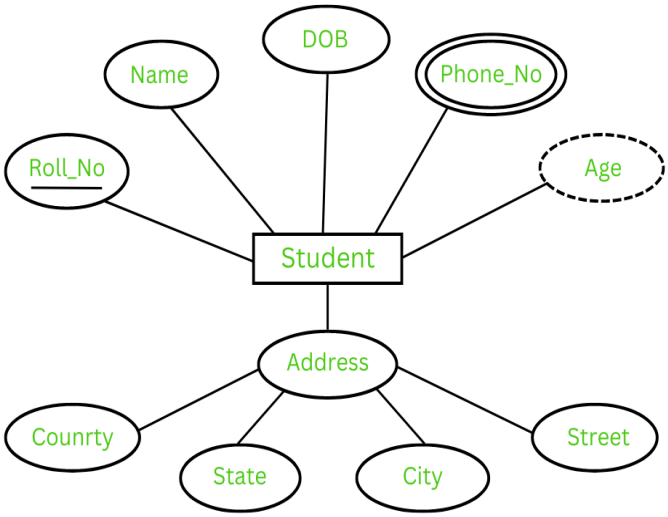
c. Multi-valued Attributes:

The property of possessing more than one value for an entity, e.g., Phone_No, maybe more than one for a student. A double oval is used in the ER diagram to denote a multivalued attribute.

d. Derived Attributes:

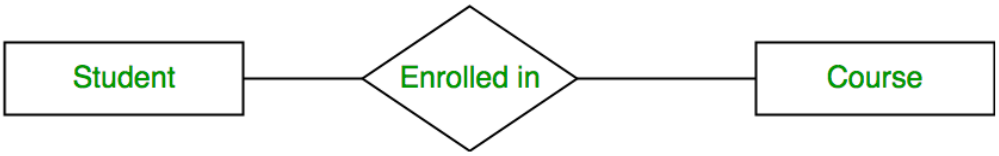
A property that can be derived from other properties of the entity type is referred to as a derived property. For instance, Age (computable from DOB). On the ER diagram, the derived property is denoted by a dashed oval.

The Complete Entity Type Student with its Attributes can be represented as:

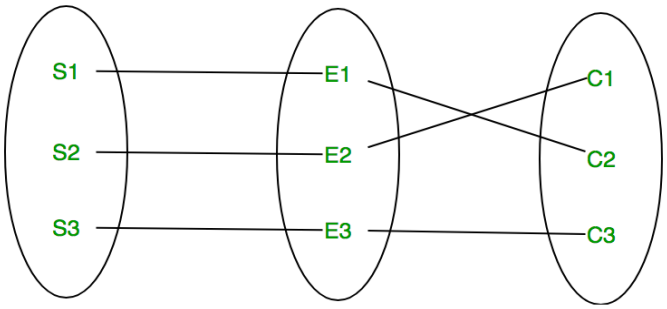


ii. Relationship Type and Relationship Set:

A relationship type defines the connection between different entity types in a database model. For instance, the relationship type 'Enrolled in' illustrates the connection between two entity types: Student and Course. In an Entity-Relationship (ER) diagram, this relationship type is represented by a diamond shape, with lines connecting it to the respective entity types involved.



A relationship set is a collection of similar relationships about a specific relationship type. For example, in the relationship set based on 'Enrolled in,' we might identify that Student S1 is enrolled in Course C2, Student S2 is enrolled in Course C1, and Student S3 is enrolled in Course C3. Each of these instances reflects the broader relationship defined by 'Enrolled in.'



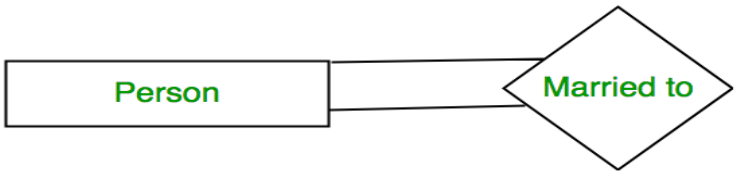
iii. Degree of a Relationship Set:

The degree of a given relation set is represented by the number of denoting different entity sets.

i. Unary Relationship:

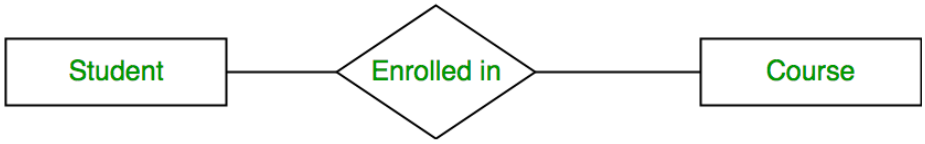
When only one entity set is involved in the relationship, it is termed a unary relationship. An example of this could be found in the instance of a person being

married to another person. In this case, the entity set consists of individuals, and the relationship describes a connection between one entity and another within the same set.



ii. Binary Relationship:

A relationship that involves only two entities is called the binary relationship. So, a (Student is enrolled in a Course).



iii. Ternary Relationship:

A ternary relationship is one in which three groups of entities have a relationship in common.

iv. N-ary Relationship:

An n-ary relationship is where there are n entities set participating within a relationship.

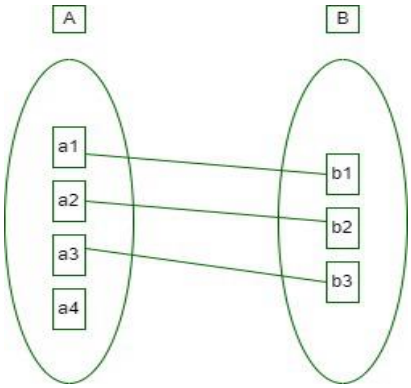
iv. Cardinality:

Cardinality is defined as the number of times an entity in an entity set participates in a relationship set. Cardinality appears in different forms:

1. One-to-One:

One-to-one cardinality applies when each entity in each entity set can take part only once in the relationship. Let's say that a male can marry one female, and a female can marry one male. Therefore, the relationship will be one-to-one. The total number of tables that can be used in this is two.

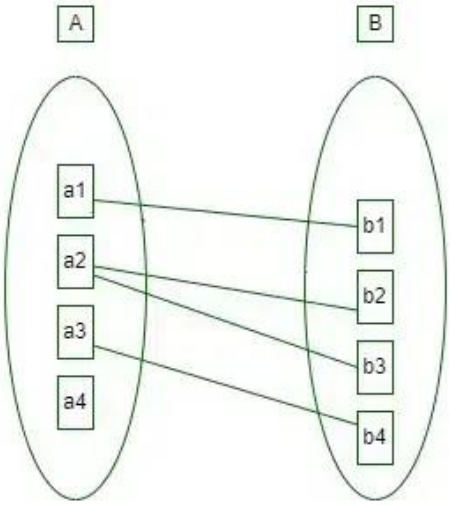
Using different sets;



2. **One-to-Many:**

In a one-to-many mapping, an entity can and will relate with many other entities, but the number of tables that can be used is always only two. Suppose one surgeon department can have unique doctors. Thus, Cardinality=1toM. It means one department can have many Doctors. Type 3 Derivatives: The maximum number of tables.

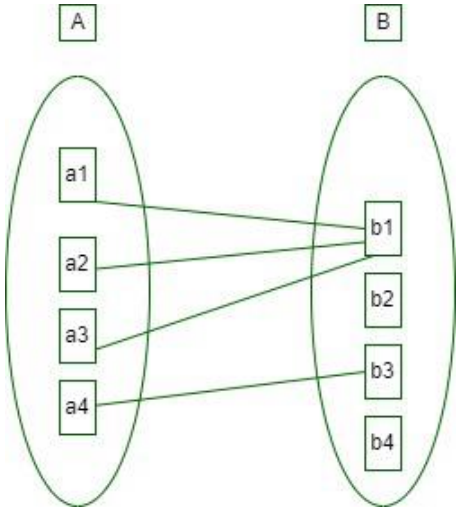
Using sets, one-to-many cardinality can be represented as:



3. **Many-to-One:**

It is the relationship of cardinality many to one when entities in one entity set can be present just once in the relationship set and entities in other entity sets can take part more than once in the relationship set. Assume that one student can only take one course and one course broadly many students can take up. Thus, cardinality will be n to 1. This means that for one course, there can be n students but for one student, there can only be one course. Here, we have a possible number of tables = 3.

Using Sets, it can be represented as:

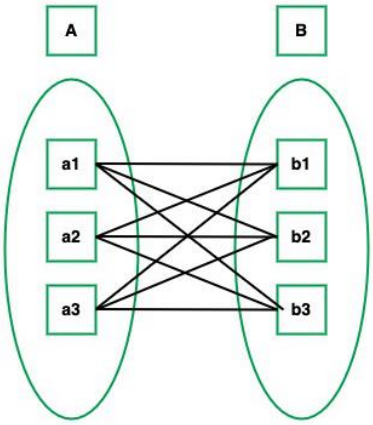


In this case, each student is taking only 1 course but 1 course has been taken by many students.

4. **Many-to-Many:**

If entities of all entity sets can participate in the relationship more than once, then the cardinality is termed as many to many. A student can take multiple courses, and a course can be taken by many students, so the relationship will be many to many. In this can be maximum of 3 tables.

Using Sets, it can be represented as:



In this example, student S1 is enrolled in C1 and C3, and Course C3 is enrolled in S1, S3, and S4. So, it is a many-to-many relationship.

7.3.4 Uses of entity relationship diagrams:

i. **Database Design:**

ER diagrams present modeling and design logics and business rules about a relational database, which are usually expressed in a logical data model and sometimes even in a physical data model, depending on the specific technology to be implemented. In software engineering work, an ER

diagram is usually a primary procedure leading to determining requirements for an information systems project. It is then used as a secondary means to model a particular database or databases. Each relational database has an equivalent relational table and could essentially be expressed to make that sort of sense.

ii. Business information systems:

Diagrams for designing or analyzing relational databases generally used in business processes may also apply. Any business process with fielded data that encompasses entities, actions taken on them and interplay among themselves can benefit from a relational database. It may streamline processes, make finding information easier and ultimately improve results.

iii. Education:

ER Diagrams are the best candidate for a data structure blueprint, while the databases are presumably utilized for articulation concepts of structured information for learning and their subsequent retrieval.

iv. Research:

Since so much of the research is dependent on organized data directly, ER diagrams have a reasonable chance to give one direction in establishing beneficial databases to work on the data.

7.4 SQL

A standardized language for programming named SQL (Structured Query Language) is employed to work with and manage relational databases. It is required for activities such as database structure creation, record updating, and querying data. Due to its simplicity and strong features, SQL is commonly applied in data engineering, data analysis, and application development.

7.4.1 Uses of SQL

SQL is important to the structuring and querying of data in databases due to its Support for so many various technologies. SQL is vital to historical relational databases (RDBMS) as well as modern-day technologies such as blockchain, artificial intelligence, and machine learning. It makes users interact with data, whether it is stored in structured RDBMSs or other types of databases, since it seamlessly interoperates with DBMSs (Database Management Systems).

1. **Web development:**

Used on sites and applications developed using frameworks like Django, Node.js, and Ruby on Rails to handle user information, commerce transactions, and content management.

2. **Data science & Analytic**

Large dataset questioning, cleaning, and analysis are all performed using data science and analytics. SQL is utilized by analysts to create insights and reports that inform business decisions.

3. **AI & Machine Learning**

Helps in preparing and structuring the data to be used to train AI algorithms and machine learning models. It is utilized in data cleansing, transformation, and extraction.

4. **Cloud and Big Data**

To support effortless data administration and querying, SQL is integrated with cloud databases (such as Amazon RDS and Microsoft Azure SQL) and big data systems (such as Apache Hive).

7.4.2 Syntax and Elements of SQL

a) Data Definition Language (DDL): DDL statements determine the structure of database objects, such as tables, indexes, and schemas. Some of the important DDL statements are:

CREATE TABLE: Creates a brand new table. All the table names, column names, data types, and constraints are defined through the CREATE TABLE statement. The constraints that ensure data integrity are PRIMARY KEY, NOT NULL, UNIQUE, and FOREIGN KEY.

Syntax: CREATE TABLE table_name

Example: CREATE TABLE Employees
(Employe ID INT Primary Key, Employe Name VARCHAR(40), Country
VARCHAR(40), phone int (10));

ALTER TABLE: Created modifications to a current table structure. ALTER TABLE is used to add, drop, or modify columns. It can also be used to apply constraints.

Example: ALTER TABLE employees ADD COLUMN hire_date DATE;
ALTER TABLE employees MODIFY COLUMN salary DECIMAL(12, 2);

TABLE TRUNCATE: The TRUNCATE TABLE command is used for the purpose of quickly deleting all of the information from a table but leaving its structure intact.

Example: TRUNCATE TABLE employees;

DROP TABLE: A table and all that it contains are erased permanently using this statement.

Example: DROP TABLE employees;

b) Data Manipulation Language (DML): Management of table data is done through DML commands. Common DML commands include:

INSERT: Adds new data. The INSERT command can be utilized to insert values for all or a selection of the columns.

Example: INSERT INTO employees (id, name, department, salary)
VALUES (1, 'John ', 'Finance', 75000.0);

UPDATE: Existing records are modified via the UPDATE command. Only designated rows are altered when the WHERE clause is used.

Example: UPDATE employees
SET salary = salary * 1.10
WHERE department = 'IT';

DELETE: This command deletes particular records. Take care while using the WHERE clause to prevent inadvertently removing every entry.

Example: DELETE FROM employees WHERE department = 'Finance';

c) Data Query Language (DQL): DQL selects data through the SELECT statement. Typical operations are:

- Selecting records with WHERE.
- Ordering data with ORDER BY.

SELECT Command: SELECT statements can return single columns or all columns (*). It is the most versatile and widely used SQL command.

Syntax: SELECT column1, column2 FROM table_name WHERE condition;

Example: Selecting Specific Column:

SELECT id, name FROM employees;

d) Data Control Language (DCL):DCL statements control access to manage data security. Typical DCL statements are:

GRANT: The GRANT command is employed to grant particular privileges to users or roles.

Syntax: GRANT privilege_name ON object_name TO user_name;

Example: GRANT SELECT, INSERT ON employees TO user1;

This grants user1 permission to perform SELECT and INSERT on the employees table.

REVOKE: The REVOKE command cancels the permissions already granted.

Syntax: REVOKE privilege_name ON object_name FROM user_name;

Example: REVOKE INSERT ON employees FROM user1;

This removes INSERT permissions from user1 on the employees table.

e) Transaction Control Language (TCL):TCL maintains database transaction consistency. Some of the important TCL commands are:

COMMIT: COMMIT stores all the changes done in the ongoing transaction to the database.

Example: BEGIN;

UPDATE employees SET salary = salary * 1.10 WHERE department = 'Sales';

COMMIT;

ROLLBACK: ROLLBACK cancels changes since the last COMMIT or SAVEPOINT.

Example: BEGIN;

UPDATE employees SET salary = salary * 1.10 WHERE department = 'Sales';

ROLLBACK;

This command reverts the salary update.

SAVEPOINT: SAVEPOINT enables partial rollbacks to a specified point in a transaction.

Example: BEGIN;

UPDATE employees SET salary = salary * 1.10 WHERE department = 'Sales';

SAVEPOINT sp1;

UPDATE employees SET salary = salary * 1.05 WHERE department = 'IT';

ROLLBACK TO sp1; -- Rolls back only the IT department update

COMMIT;

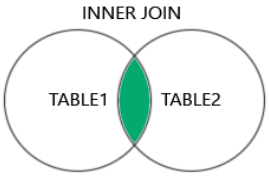
7.4.3 SQL Joins

By establishing logical relationships between tables, the SQL JOIN clause permits you to query and retrieve data from multiple tables. Utilizing common key values that are common to several tables, it can retrieve data from several tables simultaneously. SQL JOIN can be applied with multiple tables. While it can be used with other clauses, the most typical method of filtering data retrieval is to use JOIN in conjunction with the WHERE clause.

Types of SQL Joins:

1 Inner Join
2 Left Join
3 Right Join
4 Full Outer Join

1. **Inner Join:** If the condition is satisfied, the INNER JOIN keyword returns all rows from all tables. Where the condition is satisfied, i.e., the value in the common field is the same, the keyword will join all columns in both tables to form the result-set.



Syntax:

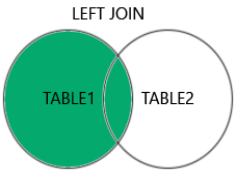
```
SELECT a.column1, b.column2
FROM tableA a
INNER JOIN tableB b
ON a.common_column = b.common_column;
```

Example:

```
SELECT employees.name, departments.department_name
FROM employees
INNER JOIN departments
ON employees.department_id = departments.id;
```

Results: only Employees who belongs to a department will be shown.

2. **Left Join:** All of the left side table rows are returned when the right side table rows are matched by the LEFT JOIN. If there are no matching rows in the right-side table, the result-set will contain null. A left join can also be referred to as a left outer join.



Syntax:

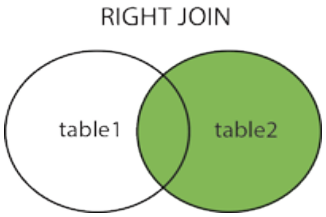
```
SELECT a.column1, b.column2
FROM tableA a
LEFT JOIN tableB b
ON a.common_column = b.common_column;
```

Example:

```
SELECT employees.name, departments.department_name
FROM employees
LEFT JOIN departments
ON employees.department_id = departments.id;
```

3. **Right Join:** RIGHT JOIN will return all rows of the table on the right side of the join and the corresponding matching rows for the table on the left side of the join. RIGHT JOIN is identical to LEFT JOIN apart from rows that do not have a corresponding row on the left, for these rows, result-set will be null.. RIGHT JOIN is another name for

RIGHT OUTER JOIN.



Syntax:

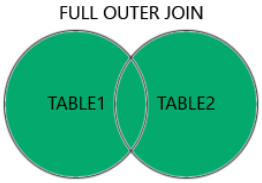
```
SELECT a.column1, b.column2
FROM tableA a
RIGHT JOIN tableB b
ON a.common_column = b.common_column;
```

Examples:

```
SELECT employees.name, departments.department_name
FROM employees
RIGHT JOIN departments
ON employees.department_id = departments.id;
```

Results: All departments will be shown, even those without assigned employees.

4. **Full Outer Join:** The results of the LEFT JOIN and RIGHT JOIN are combined to create the FULL JOIN result set. Every row from both tables will be included in the result-set. The result-set will have NULL values for the rows where there is no match.



Syntax:

```
SELECT a.column1, b.column2
FROM tableA a
FULL JOIN tableB b
ON a.common_column = b.common_column;
```

Example:

```
SELECT employees.name, departments.department_name
FROM employees
FULL JOIN departments
ON employees.department_id = departments.id;
```

Results: All employees and all departments will be displayed, even those without matches.

7.4.4 Functions

SQL functions provide a powerful and convenient method for data analysis. Using these functions in your queries, you can deepen and improve the accuracy of your insights, converting raw data into actionable intelligence.

1. **Date and time function:** date function performs operation on date values.

NOW(): The NOW() function returns the current date and time.

Syntax: SELECT NOW( );

CURDATE(): CURDATE function returns the current date.

Syntax: SELECT CURDATE( );

CURTIME(): Returns the current time.

Syntax: SELECT CURTIME( );

DATE_ADD( ): DATE_ADD function adds a specified interval to a date.

Syntax: SELECT DATE_ADD( );

DATE_DIFF(): DATE_DIFF function calculate the differences between the two dates.

Syntax: SELECT DATE_DIFF(day, '2018-01-13', '2018-01-03');

2. Aggregate Function: SQL Aggregate Functions are essential to summarize and analyze huge volumes of data.. These functions are applied across multiple rows in a table, collapsing them into a solitary, more informative result.

Different Aggregate Function:

SUM( ): Calculate the total of a numeric columns.

Syntax: SELECT SUM(Column_name) From table_name;

AVG( ): AVG( ) function calculate the average value.

Syntax: SELECT AVG(Column_name) From table_name;

COUNT( ): COUNT( ) counts the no of rows.

Syntax: SELECT COUNT(Column_name) From table_name;

MAX( ): MAX( ) returns the largest value.

Syntax: SELECT MAX(Column_name) From table_name;

MIN ( ): MIN ( ) returns the smallest value**.**

Syntax: SELECT MIN(Column_name) From table_name:

Examples:

SELECT department, AVG(salary) AS avg_salary
FROM employees
GROUP BY department;

3. String Function: SQL String Functions are very useful functions that allow us to format, modify and pick certain text data sections from our database. They are needed to perform activities such as joining text fields, sanitizing data, and matching strings.

Different String Function:

CONCAT( ): Two or more strings can be concatenated (combined) into a single string using the CONCAT() function. When we wish to combine fields, such as first and last names, into a whole name, it is helpful.

Example: SELECTCONCAT('Ranjit' , ' ', 'Singh') As Full Name;

UPPER( ): UPPER( ) function converts texts to uppercase.

Example: SELECT UPPER(' string') As UpperCase ;

LOWER( ): LOWER( ) function converts texts to lowercase.

Example: SELECT LOWER(' STRING') As LowerCase ;

SUBSTRING( ): SUBSTRING( ) Function Extract part of a string.

Example: SELECT SUBSTRING(' string', 1,4);

REVERSE( ): Reverse the characters in the string.

Example: SELECT REVERSE(‘ String’) ;

REPLACE( ): The REPLACE() function substitutes occurrences of a substring in a string with another substring.

Example: SELECT REPLACE(‘ good morning’, ‘morning’ , ‘ night’);

4. Numeric function: SQL Numeric Functions are utilized to execute operations on numeric data types like INT, FLOAT, DECIMAL, DOUBLE, and so on. These function assist you in working with numbers in several ways, such as arithmetic operations, rounding, formatting, and aggregation of data.

Different numeric function:

ABS( ) Absolute Function: ABS() turns negative numbers into positive ones by returning the number's absolute value, or the number without the sign.

Example: SELECT ABS(56);

FLOOR():FLOOR( ) function round a numbers down to the nearest integer.

Syntax: SELECT FLOOR (number);

Example: SELECT FLOOR (51.67);

ROUND( ): ROUND( ) function Rounds a number to the specified number of decimals.

Syntax: SELECT ROUND(number, decimal_palces);

Example: SELECT ROUND(12.876, 3);

CEIL( ): CEIL( ) function round a number up to the nearest integer.

Syntax: SELECT CEIL(number);

Example: SELECT CEIL(16.78);

SQRT( ) square root: SQRT( ) function returns the square root of the number.

Syntax: SELECT SQRT(number);

Example: SELECT SQRT(7);

LOG( ) Logarithm: Returns the natural logarithm of a number.

Syntax: SELECT LOG(number);

Example: SELECT LOG(99);

The basis of data engineering is SQL. Scalable data pipelines, ensuring data integrity, and efficient handling of large datasets are possible by data engineers if they understand its fundamentals , advanced concepts, and performance optimization strategies. Data management and transformation are enhanced when SQL and data engineering systems are integrated.

7.5 Stored Proceduces:

A Stored Procedure is a collection of one or more SQL statements that are precompiled and saved in a database. Stored procedures are created to accomplish certain tasks, including data manipulation, reporting, validation, and data conversion. Rather than repeatedly writing and running SQL queries, stored procedures enable users to encapsulate complex logic in a database object, making database operations more efficient and reusable.

Stored procedures help when several applications or users have to execute the same database operations. Rather than running individual queries, a stored procedure can be called by its name

and with suitable parameters. This saves redundancy and enhances performance by reducing the SQL code transmitted to the database server.

7.5.1 Key Elements of a Stored Procedure:

1. Procedure Name

All stored procedures should be named uniquely within the database. This name acts as an identifier to call the procedure when necessary. Selecting a descriptive and meaningful name for a stored procedure improves code readability and maintainability. The name should be consistent with database naming rules and must explicitly state the function of the procedure. For instance, a procedure to get employee details may be named GetEmployeeDetails.

For example:

CREATE PROCEDURE GetEmployeeDetails

2. Parameters

Stored procedures facilitate the use of parameters, which provide flexibility through dynamic execution. Rather than depending on hardcoded values, parameters allow stored procedures to receive varying inputs at runtime, making them reusable across different situations. This avoids having to write multiple queries for various values, enhancing efficiency and maintainability. Parameters are especially helpful when filtering, calculations, or condition-based operations are needed, as they enable a single stored procedure to process several cases dynamically.

Parameters in stored procedures are of three types: input parameters, output parameters, and both. Input parameters are employed to feed values into the procedure, which in turn affects its execution. They are used as substitutes for variables like customer IDs, employee IDs, or date intervals. For instance, an input parameter can be utilized to bring details of a particular employee by passing their EmployeeID rather than creating individual queries for every employee. The stored procedure executes the input value and returns the required data.

Conversely, output parameters enable stored procedures to pass values back to the calling application or program. Such parameters are employed when one value is required to be returned as opposed to a complete result set. For example, a stored procedure may accept a department ID as an input parameter and return the number of employees in that department through an output parameter. This method is typically applied in reporting and business logic implementations where summary data, calculations, or status indicators must be returned.

Stored procedures can also utilize both input and output parameters simultaneously, allowing more interactive data processing. In such a scenario, an input parameter supplies the data required for processing, and an output parameter delivers the calculated result. For instance, a stored procedure can take an OrderID as input and deliver the total order amount as output. This approach is useful in e-commerce applications, where totals, discounts, or taxes need to be calculated in real time.

For example:

CREATE PROCEDURE GetEmployeeDetails @EmployeeID INT

3. BEGIN and END Block

A stored procedure typically includes a BEGIN and END block, which define its boundaries. This block encapsulates all SQL statements within the procedure, ensuring they execute as a single unit. It also enhances readability and maintainability, particularly for complex procedures that contain multiple

operations.

For example:

```
BEGIN
 SELECT * FROM Employees WHERE EmployeeID = @EmployeeID;
END;
```

4. SQL Statements

The inner working of a stored procedure lies in the SQL commands it executes. These commands can include SELECT, INSERT, UPDATE, DELETE, MERGE, as well as procedural statements like loops and conditions. The SQL inside the procedure is precompiled, which enhances performance. This precompilation enables efficient manipulation and retrieval of data, reducing the need for repeated query execution and improving overall system efficiency.

For example:

```
SELECT * FROM Customer WHERE City = 'New York'
```

5. Return Value

A stored procedure can return an integer to indicate the execution status. The returned value is frequently used to denote success or failure. A value of 0 usually indicates successful execution, and non-zero values represent an error or special condition. This function is especially important in application programming, where proper error handling is essential.

For example:

```
RETURN 0;
```

6. Transaction Management

Stored procedures facilitate transaction management, maintaining data consistency and integrity. Transactions employ commands such as BEGIN TRAN, COMMIT, and ROLLBACK to treat multiple operations as one unit. In case an operation fails, the transaction can be rolled back to avoid partial updating. This is vital for banking, order processing, and financial applications where precision is essential.

For example:

```
BEGIN TRAN
 UPDATE Accounts SET Balance = Balance – 500 WHERE AccountID = 1;
 UPDATE Accounts SET Balance = Balance + 500 WHERE AccountID = 2;
COMMIT TRAN;
```

7. Error Handling

Stored procedures can have error handling provisions to ensure that failures do not lead to disruptions. A TRY...CATCH block is generally used to trap errors and handle them gracefully. Upon encountering an error, appropriate actions such as logging the error, displaying a message, or rolling back a transaction can be performed. This enhances the reliability of database operations and improves overall system stability.

For example:

```
BEGIN TRY
INSERT INTO Orders (orderID, CustomerID, Amount) VALUES (101, 1, 500);
END TRY
```

```
BEGIN CATCH
 PRINT 'Error Occurred';
END CATCH;
```

8. Execution of a stored procedure

After a stored procedure is defined, it must be called to carry out the specified database operation. This is achieved by employing the EXEC or EXECUTE statement, followed by the procedure name and any necessary parameters. Calling stored procedures rather than writing explicit queries provides consistency and eliminates redundant query execution.

For Example:

```
EXEC GetEmployeeDetails @EmloyeeID = 102;
```

7.5.2 Types of SQL Stored Procedures:

Depending on their use and capabilities, SQL stored procedures are divided into several categories:

1.System Stored Procedures: The SQL Server provides these pre-existing stored procedures for administrative duties. Examples are sp_rename to rename database objects and sp_help to see database object information.

2.User-Defined Stored Procedures: These are stored procedures created by the user that execute specific tasks.For example, creating a process to find the total sales for a particular product category.

3.Extended stored procedures: which may be implemented in alternate languages like C or C++, provide the capability to execute routines from outside SQL Server. External utilities are generally made available in SQL Server via extended procedures.

7.5.3 Benefits and Use Cases of Stored Procedures

A database is an essential component that can efficiently store and manage data, enabling various operations such as inserting new data, retrieving existing information, updating records, and deleting unnecessary data. It fulfills a fundamental requirement for any application or analytical process, emerging as a solution that addresses common problems associated with the retrieval and storage of data. Databases are designed with several key elements, including fields, records, and tables, particularly in the case of relational databases, and they can be accessed concurrently by multiple users. A table, specifically, represents a structured collection of related data that is neatly organized into rows and columns, forming a cohesive grouping of data parts known as fields. Additionally, a schema serves as an overarching framework that encompasses a collection of database objects, defining the organization of the data within a database. Ultimately, the database's contents are consistently organized and maintained in a structured format made up of tables.

ER-Diagram is a widely used database model presented in a clear and engaging visual format that significantly aids in understanding complex data systems. An ER Diagram effectively illustrates the types of data that are stored within a database, which are typically organized into structured tables, and it demonstrates the intricate relationships that exist between that data. Furthermore, an ER-Diagram is capable of representing a variety of complex systems such as a university, a hospital, a business organization, and many more diverse entities. Essentially, any intricate and multifaceted system can be accurately depicted in the ER-Diagram format, making it an invaluable and versatile tool for database design and comprehensive analysis. The ability to visually represent relationships also enhances

communication among stakeholders, facilitating better decision-making and clearer insights into the underlying structure of the database.

SQL, which stands for Structured Query Language, represents a highly effective and powerful tool that grants users the ability to seamlessly communicate with databases in a clear and structured manner. A sentence formulated using SQL is specifically designed and meticulously crafted to query a database and perform a wide range of operations on the valuable data that is stored within it. The queries that are constructed are integral to the functioning of the database, as they provide precise and unambiguous instructions to the database, effectively guiding it on how to respond accurately to various data requests. This systematized and organized approach ensures that all the critical information is securely stored in the database system. Therefore, whenever specific information is needed by users or applications, it must be retrieved from the database using appropriate and well-structured SQL queries. Furthermore, SQL is also extensively employed to input new data into the database, making it crucial for managing and maintaining databases effectively and efficiently. Any information that is to be added, manipulated, or updated has to be generated through the SQL statements and subsequently stored within the database structures. In today's fast-paced digital landscape, databases play a vital and significant role in storing and retrieving a multitude of information and data types. The various SQL commands serve as an effective means to convey specific instructions to the database, thus facilitating efficient data management while ensuring data integrity and reliability.

Stored Procedures are essentially a well-defined collection of SQL statements that work together to achieve a specific output by performing various conditions and operations. The set of functions that can be executed by stored procedures includes inserting data into a table, retrieving data from a table, updating existing records in the table, and deleting data that is no longer needed from the table. These procedures are made up of specific SQL queries and statements, and they need to be compiled on a server where the associated database resides. Once compiled, the Stored Procedures are securely stored within the database and subsequently executed precisely as required. One of the key advantages of using stored procedures is that they significantly reduce network traffic while also enhancing security. In many scenarios, there arises a necessity to update data in a table or to summarize the data contained in that table whenever particular events transpire. In such situations, all necessary activities can be efficiently handled by crafting stored procedures and executing them. Some notable benefits of using stored procedures, along with detailed use cases that have been effectively developed in the PostgreSQL database system, will be described further.

7.6 Indexing

Indexing is a fundamental technique used in database management systems to enhance the speed of data retrieval. Without indexing, searching for a specific record in a large dataset would require scanning each record sequentially, which is highly inefficient. An index creates a separate data structure that maintains references to the actual data, allowing queries to locate records quickly without having to search the entire dataset. By structuring the index efficiently, databases can significantly reduce the number of disk I/O operations, leading to improved performance and reduced query execution times.

Indexes are essential for large-scale applications where real-time or near-instantaneous access to data is required. They are particularly useful in scenarios where searches, lookups, and sorting operations are frequently performed. However, despite their benefits, indexes introduce additional storage requirements

and maintenance overhead since they need to be updated whenever data is inserted, updated, or deleted. Thus, database administrators must strategically decide which columns to index to balance performance improvement and resource consumption.

7.6.1 Types of Indexing

Indexing can be categorized into different types based on the attributes and structure used:

a. Primary Index

A **primary index** is created on the primary key of a table, ensuring that records remain unique and sorted according to the indexed attribute. Since primary indexes directly correspond to the primary key, they eliminate duplicate entries and optimize query execution plans. Because the index enforces uniqueness, retrieval operations are significantly faster as each lookup returns at most one record. However, primary indexes require careful maintenance, as updates or insertions can result in expensive reorganization of the index structure.

b. Secondary Index

Unlike the primary index, a **secondary index** is built on non-primary key attributes to enhance search efficiency. This type of index allows for fast retrieval based on commonly queried columns that are not part of the primary key. Secondary indexes are particularly useful for analytical queries that involve filtering based on non-unique attributes, such as a customer's name or an order status. However, secondary indexes require additional storage space and may lead to increased maintenance overhead since any modification to the data must also update the associated index.

c. Clustered Index

A **clustered index** organizes the actual data within the index structure itself, meaning that the physical order of rows in the database corresponds to the order of the index. Because the data is directly tied to the index, clustered indexes improve the performance of range queries and sequential data access. However, only one clustered index can exist per table since the data itself is sorted accordingly. The clustered index significantly improves search performance for queries that involve range-based filtering, such as date-based queries. However, modifying clustered index columns requires reorganizing the data, which can be expensive in terms of processing time and storage.

d. Non-clustered Index

A **non-clustered index** is an index that stores pointers to the actual data instead of physically organizing it. This type of index allows for multiple indexes on the same table, improving search efficiency while maintaining data organization separately. Non-clustered indexes are beneficial when multiple attributes need to be indexed for fast lookups. However, non-clustered indexes introduce additional storage overhead and may slow down write-heavy operations since each modification must update multiple index structures.

7.6.2 Indexing Strategies and Techniques

Database management systems use different indexing strategies based on data characteristics and query patterns:

a. Single-Level Indexing

Single-level indexing involves creating a simple lookup table that maps keys to their corresponding data locations. This approach is effective for small datasets but becomes inefficient as the dataset grows, leading to increased lookup times.

b. Multi-Level Indexing

To optimize large datasets, **multi-level indexing** introduces a hierarchical index structure where the first level contains pointers to second-level indexes, which in turn point to actual data locations. This technique reduces the number of disk accesses required for data retrieval, improving overall performance.

c. Hash-Based Indexing

In **hash-based indexing**, data is mapped to a hash table where each key is transformed into a fixed-length value that determines its storage location. Hash-based indexing allows for constant-time retrieval operations but is less efficient for range queries since hashed values do not maintain a sorted order.

d. Tree-Based Indexing

Tree-based indexing, particularly using **B-Trees and B+ Trees**, organizes index data in a balanced hierarchical structure, ensuring efficient searches, insertions, and deletions. These trees allow for logarithmic time complexity operations and are widely used in modern databases.

7.6.3 Advantages of Indexing

- **Faster Data Retrieval:** One of the primary advantages of indexing is the significant improvement in data retrieval speed. Since indexes reduce the number of records that need to be scanned, query execution times are minimized, leading to enhanced performance. Indexed searches are much faster than full table scans, making them essential for applications where users expect real-time responses.
- **Optimized Query Execution:** Indexing optimizes query execution plans by allowing the database engine to determine the most efficient way to retrieve data. This results in better performance and optimized resource utilization.
- **Reduction in Disk I/O Operations:** Another key benefit is the reduction in the number of disk I/O operations. Without an index, every query would require reading through the entire dataset, leading to increased resource consumption. By leveraging indexing, databases can minimize unnecessary reads and writes, conserving computing power and improving overall efficiency.
- **Improved Sorting and Filtering:** Indexing also plays a vital role in sorting and filtering operations, ensuring that ordered results can be retrieved with minimal computational effort.

7.6.4 Disadvantages of Indexing

- **Additional Storage Requirements:** Despite its benefits, indexing comes with certain drawbacks. One major disadvantage is the additional storage space required. Each index creates a separate data structure that must be stored alongside the actual data, increasing the overall disk space consumption.
- **Performance Overhead in Updates:** Maintaining indexes incurs performance overhead, particularly during insert, update, and delete operations. Every time a record is modified, the corresponding index must be updated, which can slow down write-heavy applications.
- **Excessive Indexing Can Degrade Performance:** If an excessive number of indexes are created on a table, it can lead to performance degradation rather than improvement. The database may spend more time maintaining indexes rather than executing queries efficiently.
- **Index Fragmentation:** Another issue is index fragmentation, where frequent updates can cause indexes to become less efficient over time, necessitating periodic maintenance.

7.7 B+ Tree

The **B+ Tree** is a self-balancing tree data structure commonly used in database indexing and file systems. It extends the concept of the **B-Tree** but differs in structure and efficiency. B+ Trees improve search, insertion, and deletion operations while ensuring that data remains sorted and easily accessible. Their balanced nature prevents excessive disk I/O operations, making them particularly useful for large-scale databases where performance and scalability are critical concerns.

B+ Trees provide a hierarchical structure that enables fast lookups as visualize in fig.7.10, as well as efficient range queries. Unlike B-Trees, where both keys and values are stored at internal and leaf nodes, B+ Trees store values only in the leaf nodes while internal nodes act as a guide to the correct leaf node. This property makes B+ Trees highly efficient in database indexing, as they allow for fast sequential access to data while maintaining logarithmic search time complexity.

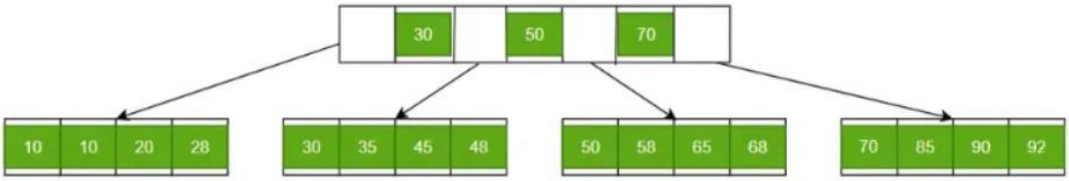


Fig. 7.10: B+ tree

7.7.1 Structure of B+ Tree

A B+ Tree consists of two types of nodes:

a. Internal Nodes

Internal nodes in a B+ Tree contain keys and act as decision points, directing the search towards the correct leaf node. They do not store actual data but instead provide a pathway for traversing the tree efficiently. Each internal node contains multiple keys and pointers to child nodes, ensuring that the tree remains balanced at all times.

b. Leaf Nodes

Leaf nodes store the actual data records in a sorted order. Unlike internal nodes, leaf nodes maintain pointers to their neighboring leaf nodes, enabling efficient range queries and sequential data retrieval. This linked structure allows for fast iteration over a range of values without having to traverse back up the tree.

c. Root Node

The root node is the topmost node in the B+ Tree. It can either be an internal node or, in the case of very small trees, a leaf node. The root node plays a crucial role in maintaining balance within the tree by ensuring that all paths from the root to the leaf nodes are of equal length.

7.7.2 Operations in B+ Tree

B+ Trees support several key operations that are fundamental to database indexing and file systems.

a. Searching in a B+ Tree

Searching in a B+ Tree is an efficient process due to its hierarchical structure. Given a search key, the algorithm starts at the root node and follows the appropriate pointer to a child node based on the range in which the key falls. This process continues until a leaf node is reached, where the key is either found or determined to be absent. The logarithmic height of the tree ensures that the number of comparisons remains minimal, making searches highly efficient.

***b*. Insertion in a B+ Tree**

Insertion in a B+ Tree follows a structured process to maintain balance. When a new key is inserted, it is added to the appropriate leaf node in sorted order. If the leaf node becomes overfull (i.e., exceeds the predefined maximum number of keys), it is split into two nodes, and the middle key is promoted to the parent node. This recursive process continues until all nodes are balanced, ensuring that the tree remains optimized for future searches.

c. Deletion in a B+ Tree

Deletion in a B+ Tree is handled carefully to maintain balance and efficiency. When a key is deleted, the algorithm checks if the leaf node still contains the minimum required number of keys. If a deficiency occurs, keys may be borrowed from a neighboring node or merged with another node to maintain balance. This ensures that the structure of the tree remains optimized and prevents excessive fragmentation.

d. Range Queries in B+ Tree

One of the biggest advantages of B+ Trees over traditional B-Trees is their efficiency in handling range queries. Since all actual data is stored in the leaf nodes, and each leaf node is linked to its adjacent node, sequential traversal of records is straightforward. This feature is particularly useful in databases where queries involve retrieving records that fall within a specific range, such as finding all employees with salaries between two values.

7.7.3 Advantages of B+ Tree

- **Efficient Search Performance:** The hierarchical structure of B+ Trees ensures logarithmic time complexity for search operations, making them highly efficient for large databases.
- **Optimized Range Queries:** The linked structure of leaf nodes allows for fast and efficient range queries, unlike other tree structures that require repeated traversals.
- **Balanced Tree Structure:** B+ Trees are self-balancing, ensuring that all paths from the root to the leaf nodes are of equal length. This prevents skewed tree growth and maintains optimal search performance.
- **Efficient Storage Utilization:** Since internal nodes store only keys and not data, B+ Trees make efficient use of disk space, reducing storage overhead.
- **Minimized Disk I/O Operations:** Due to their balanced nature and hierarchical traversal, B+ Trees reduce the number of disk accesses required for searches and updates, improving overall performance in database systems.

7.7.4 Disadvantages of B+ Tree

- **Insertion and Deletion Overhead:** While B+ Trees maintain balance automatically, the process of insertion and deletion can be complex and require node splitting or merging, leading to performance overhead.
- **Higher Memory Usage:** Maintaining additional pointers for linked leaf nodes increases memory consumption compared to simpler data structures like binary search trees.
- **Complex Implementation:** The self-balancing nature of B+ Trees introduces implementation complexity, requiring careful management of node structures and tree balance during updates.
- **Slower Updates Compared to Hash Indexing:** For workloads with frequent insertions and deletions, hash-based indexing may provide better performance since B+ Trees require restructuring when nodes become overfull or underfull.

7.7.5 Use Cases of B+ Tree

B+ Trees are widely used in various applications, particularly in database management systems and file systems.

1. **Database Indexing**

Most modern databases use B+ Trees for indexing due to their efficient search and range query capabilities. Relational database management systems (RDBMS) such as MySQL, PostgreSQL, and Oracle utilize B+ Trees to optimize query performance.

2. **File Systems**

Many file systems, including NTFS (Windows) and EXT4 (Linux), use B+ Trees to organize and store metadata efficiently. The hierarchical structure allows for fast lookup of file attributes and directory entries.

3. **Key-Value Stores**

NoSQL databases and key-value stores, such as Berkeley DB and LevelDB, use B+ Trees for indexing and storage management. The ability to handle range queries efficiently makes B+ Trees a preferred choice for large-scale data storage solutions.

7.8 MongoDB

MongoDB is a widely used NoSQL database designed to handle large volumes of unstructured or semi-structured data. Unlike traditional relational databases that use tables and rows, MongoDB stores data in flexible, JSON-like documents, making it highly scalable and adaptable to various use cases. This document-oriented approach allows for seamless data retrieval, modification, and indexing, making MongoDB a preferred choice for modern applications that require high performance and flexibility.

MongoDB is built on a distributed architecture, enabling horizontal scaling and high availability. It is particularly useful for applications that require rapid development, real-time analytics, and the ability to handle diverse data formats. MongoDB provides built-in features such as automatic sharding, replication, and indexing, making it an efficient solution for handling big data and cloud-based applications.

7.8.1 Architecture of MongoDB

MongoDB follows a client-server architecture where multiple clients can interact with the database server to store and retrieve data. The core components of MongoDB's architecture include:

a. **Documents**

MongoDB stores data in **documents** rather than tables. A document is a JSON-like structure that consists of field-value pairs, where the values can be of various data types, including arrays, embedded documents, and binary data. This flexibility allows MongoDB to handle complex and hierarchical data structures efficiently.

b. **Collections**

Documents in MongoDB are grouped into **collections**, which are analogous to tables in relational databases. Unlike tables, collections do not enforce a fixed schema, allowing documents within a collection to have different structures. This schema-less nature provides greater adaptability and makes it easier to evolve data models over time.

c. MongoDB Server

The MongoDB server is responsible for managing database operations, processing queries, and ensuring data consistency. It handles requests from clients and coordinates data storage and retrieval operations efficiently.

d. **Sharding and Replication**

MongoDB supports **sharding** and **replication** to ensure scalability and fault tolerance:

- **Sharding** enables horizontal scaling by distributing data across multiple servers, improving read and write performance.
- **Replication** ensures high availability by maintaining multiple copies of data across different nodes, preventing data loss in case of server failures.

7.8.2 Indexing in MongoDB

Indexing in MongoDB enhances query performance by allowing faster data retrieval. Without indexes, MongoDB must scan every document in a collection to find matching records, which can be inefficient for large datasets. MongoDB provides various types of indexes to optimize searches, including:

1. Single Field Indexes

Single field indexes are the simplest form of indexes in MongoDB, defined on a single field of a document. They greatly improve query performance while querying for certain values in a collection. For instance, if there is a MongoDB collection holding user data with fields like name, email, and age, indexing the name field permits the database to retrieve documents quickly based on an individual's name. If there were no index, MongoDB would need to search through all of the documents in the collection to obtain matches, which is slow with large amounts of data. With the use of a single field index, db.users.find({name: "John Doe"}) runs a lot faster, enhancing efficiency.

2. Compound Indexes

Compound indexes are established on more than one field within a document, maximizing queries for filtering or sorting data on more than one basis. Compound indexes are very useful when conditions exist in multiple fields in queries because they avoid the use of individual indexes for each field. For instance, suppose a collection contains employee records with department, salary, and hireDate fields. If queries often search for employees in a particular department and salary range, a compound index on department and salary would enable MongoDB to return results more quickly. Compound indexes also support sort operations, so they are beneficial for queries that need ordered results, like getting the top-paid employees in a department.

3. Multi-Key Indexes

MongoDB automatically creates multi-key indexes when it indexes fields that have arrays. Because arrays can store more than one value, MongoDB creates an index entry for each value in the array, making it efficient to search within array elements. This kind of indexing is especially helpful when working with documents that contain lists or collections in a single field. For instance, in a database where every document corresponds to a product with a tags array field (e.g., ["electronics", "smartphone", "Android"]), a multi-key index on tags allows quick searches for products with a certain tag. Without this index, searching within an array field would involve scanning all documents, rendering queries much slower for large data sets.

4. Geospatial Indexes

MongoDB has geospatial indexes, which are meant to improve location-based queries. These indexes facilitate quick searching for local places and hence are critical in applications that involve mapping, location tracking, or geographic search capability. Indexing coordinates held in a GeoJSON format or pre-existing coordinate pairs, MongoDB makes it possible to query for operations like finding restaurants within a particular radius, nearest service centers, or points of interest around the user's current location. For instance, an application that provides real-time navigation assistance can use a geospatial index on a location field containing latitude and longitude coordinates to efficiently retrieve nearby destinations. Without geospatial indexes, such queries would require extensive computation, leading to performance issues in applications with large datasets.

7.8.3 CRUD Operations in MongoDB

MongoDB supports basic **Create, Read, Update, and Delete (CRUD)** operations:

a. Creating Documents

New records in MongoDB are created by inserting documents into collections. This is accomplished through the use of the insertOne() function in order to add one document or the insertMany() function in order to add multiple documents simultaneously. In contrast to relational databases, MongoDB stores data in BSON (Binary JSON) format, which supports flexible and schema-less storage of data. When a document is inserted, MongoDB automatically generates a unique _id field as an identifier where none is supplied.

For example:

Adding a new user to **users** collection can be done as follow:

db.users.insertOne({ name: “Alice”, age: 25, email: “ alice@example.com” });

b. Reading Documents

Data is retrieved from MongoDB collections by using the find() method, which permits querying on certain criteria. The findOne() method gives a one matching document, whereas find() retrieves more than one document matching the query. MongoDB's flexible query language makes filtering documents on field values, conditions, and even complex expressions possible.

For example:

Retrieving all documents from **users** collection can be done as follow:

Db.users.find();

Db.users.findOne ({name: “Alice” }); // a filter condition is applied

c. Updating Documents

MongoDB offers several ways to update documents in a collection. The updateOne() function updates the first document that matches, whereas updateMany() updates all documents that match the given conditions. The replaceOne() function can also be employed to replace a document completely. Updates usually involve update operators like $set, $inc, and $push to dynamically update certain fields.

For example:

Db.users.updateOne({ name: “Alice” }, { $set: { age: 26 } });

d. Deleting Documents

MongoDB supports the deletion of unnecessary documents through the use of the deleteOne() and deleteMany() functions. The deleteOne() function deletes one document that satisfies the given conditions, whereas the deleteMany() function deletes all the documents satisfying the given condition. These processes keep the data in the database clean and up to date.

For example:

Db.users.deleteOne({ name: "Alice" });

7.8.4 Advantages of MongoDB

- **Flexible Schema:** MongoDB's document-based model allows for dynamic and flexible data structures, making it easier to handle evolving requirements.
- **High Scalability:** The ability to distribute data across multiple nodes through sharding ensures that MongoDB can handle large volumes of data efficiently.
- **Fast Data Retrieval:** Indexing mechanisms optimize search queries, reducing the time required to find relevant documents.
- **Replication and Fault Tolerance:** Data redundancy ensures high availability and prevents data loss in case of server failures.
- **Support for Complex Data Types:** The ability to store arrays, nested documents, and binary data makes MongoDB suitable for handling diverse data formats.

7.8.5 Disadvantages of MongoDB

- **Higher Memory Consumption:** The lack of a fixed schema and document-based storage leads to increased memory usage compared to relational databases.
- **Complex Transactions:** Although MongoDB supports transactions, they are not as robust as those in traditional SQL databases, making it less suitable for applications requiring strict ACID compliance.
- **Indexing Overhead:** While indexes improve performance, they also require additional storage and increase memory consumption.
- **Limited Joins:** MongoDB does not support complex joins like relational databases, requiring developers to denormalize data to optimize queries.
- **Not Ideal for Small Datasets:** The benefits of MongoDB's architecture are most evident for large-scale applications. For small datasets, relational databases may be more efficient.

7.8.6 Use Cases of MongoDB

MongoDB is widely used in various industries due to its scalability and flexibility. Common use cases include:

a. Web Applications

Many modern web applications use MongoDB for content management, user authentication, and real-time analytics. Its ability to handle large amounts of user-generated data makes it ideal for platforms like social media and e-commerce.

b. Big Data and Analytics

MongoDB is well-suited for handling big data applications due to its ability to store and process large datasets efficiently. Companies use it for real-time analytics, fraud detection, and recommendation systems.

c. Internet of Things (IoT)

IoT applications require databases that can process and store large volumes of sensor data in real-time. MongoDB's scalability and support for time-series data make it a preferred choice for IoT-based applications.

d. Content Management Systems (CMS)

Many CMS platforms leverage MongoDB to store and retrieve dynamic content efficiently. Its schema-less design allows for easy modifications and updates to content structures.

7.9 Client-Server Architecture

Client-server architecture is a fundamental model in computer networking and distributed systems, where multiple clients communicate with a central server to request and retrieve data or services. This architecture is widely used in various applications, including web services, database management, email communication, and cloud computing. The client-server model facilitates efficient resource sharing, central management, and scalability, making it a cornerstone of modern computing systems.

The client-server architecture works by dividing the system into two primary entities: **clients** and **servers**. Clients initiate requests, while servers process these requests and send responses. The interaction occurs over a network, which can be a local area network (LAN) or a wide area network (WAN), including the internet.

7.9.1 Components of Client-Server Architecture

The client-server model consists of several key components that define its functionality:

1. Client

The client is a device or application that requests services from a server. Clients can be web browsers, mobile applications, or desktop applications. The primary responsibilities of a client include:

- **User Interaction:** Providing a user interface for interaction.
- **Request Generation:** Sending requests to the server for data or services.
- **Processing Responses:** Displaying or utilizing the received data.

Examples of clients include web browsers like Chrome or Firefox accessing websites, mobile apps retrieving data from cloud storage, and email applications connecting to mail servers.

2. Server

The server is a powerful computer or program that provides requested services to multiple clients. It manages resources, processes client requests, and sends back appropriate responses. A server's primary functions include:

- **Data Storage and Management:** Storing and managing databases, files, and user information.
- **Request Processing:** Handling multiple client requests concurrently.
- **Security and Access Control:** Ensuring only authorized clients can access sensitive data.

Examples of servers include web servers (Apache, Nginx), database servers (MySQL, MongoDB), and application servers hosting software applications.

3. Network

The network is the communication medium that connects clients and servers. It can be a local network (LAN) or a global network (Internet). The efficiency of client-server communication depends on network speed, security, and reliability.

4. **Protocols**

Client-server communication follows specific protocols that define how data is exchanged. Common protocols include:

- **HTTP/HTTPS:** Used for web communication between clients (browsers) and web servers.
- **FTP (File Transfer Protocol):** For transferring files between clients and servers.
- **SMTP (Simple Mail Transfer Protocol):** Used for sending emails between mail servers.
- **TCP/IP (Transmission Control Protocol/Internet Protocol):** The fundamental protocol suite for internet communication.

7.9.2 Types of Client-Server Architectures

The client-server model can be categorized into different architectures based on the distribution of processing tasks:

1. **Two-Tier Architecture**

In a two-tier architecture, the client directly communicates with the server. The client handles the presentation layer, while the server manages data processing and storage. This model is simple but lacks scalability.

Example: A desktop application that directly connects to a database server.

2. **Three-Tier Architecture**

The three-tier architecture introduces an additional layer, known as the application server, between the client and database server. The three layers include:

1. **Presentation Layer (Client):** Handles user interactions.
2. **Business Logic Layer (Application Server):** Processes requests and applies business rules.
3. **Data Layer (Database Server):** Stores and manages data.

This architecture improves scalability, security, and performance.

Example: A web application where the frontend (browser) communicates with an API (application server), which then queries a database.

3. **Multi-Tier (N-Tier) Architecture**

In multi-tier architectures, additional layers such as caching, load balancing, and microservices are introduced to improve efficiency and performance. This model is commonly used in cloud-based and enterprise applications.

Example: E-commerce websites like Amazon, which use multiple layers for managing user requests, transactions, and inventory.

7.9.3 Advantages of Client-Server Architecture

1. **Centralized Management**
    - Servers centralize data storage and resource management, simplifying maintenance and administration.
2. **Scalability**
    - New clients can be easily added without disrupting the existing system.
3. **Efficient Resource Utilization**
    - Servers optimize processing power and storage, reducing redundancy.
4. **Improved Security**
    - Access control and authentication mechanisms protect data from unauthorized access.

5. **Data Consistency**
    - A single, authoritative server ensures data consistency and integrity across multiple clients.
6. **Easier Maintenance**
    - Updates and patches can be applied centrally to the server rather than on individual clients.

7.9.4 Disadvantages of Client-Server Architecture

1. **Single Point of Failure**
    - If the server crashes, all connected clients lose access to services.
2. **High Initial Cost**
    - Setting up and maintaining servers requires significant investment in hardware and software.
3. **Network Dependency**
    - The efficiency of client-server communication depends on network reliability and speed.
4. **Scalability Challenges**
    - While scalable, extreme traffic loads can lead to bottlenecks and require load balancing solutions.
5. **Security Risks**
    - Centralized servers are prime targets for cyber-attacks, requiring robust security measures.

7.9.5 Use Cases of Client-Server Architecture

1. **Web Applications**
    - Websites and cloud applications follow a client-server model where browsers (clients) communicate with web servers.
2. **Email Services**
    - Email clients interact with mail servers using SMTP, IMAP, and POP3 protocols.
3. **Database Management Systems**
    - Applications connect to database servers to retrieve and manipulate data.
4. **Cloud Computing**
    - Cloud platforms host applications and provide services to remote clients over the internet.
5. **Online Gaming**
    - Multiplayer games use game servers to synchronize player actions in real-time.

CHAPTER 8

INFORMATION RETRIEVAL IN DATA ENGINEERING

Jasmeet Kaur[1], Manpreet Kaur[2] and Vikramjit Parmar[3]

[1,2,3]GNA University, Phagwara

8.1 Introduction to Information Retrieval

8.1.1 Understanding Information Retrieval

Information Retrieval (IR) is the discipline focused on locating information within extensive data sources, encompassing documents, databases, and online resources. Unlike standard database searches that provide exact matches, IR systems strive to retrieve the most pertinent information according to user input. These systems are crucial for effectively managing and extracting knowledge from both unstructured and semi-structured data.

At its core, IR hinges on interpreting, ranking, and retrieving relevant content from large repositories. This significance makes it a vital area in computer science, artificial intelligence, and data engineering. As digital content expands rapidly, IR methodologies have become essential for various applications such as web search engines and recommendation platforms.

Key Components of Information Retrieval

1. Acquisition

Acquisition is the very first step in the information retrieval process. It usually entails harvesting relevant documents and textual content from various web sources. In this process, we involve automated programs named web crawlers that can systematically scan and gather data from any website. Such crawlers navigate through hyperlinks, collect text-based content, and store it in a database for further investigation. Sources of acquired data might include news articles, research papers, blogs, and even structured documents, in a quest for complete collection of information that's retrievable.

2. Representation

This stage of representation is concerned, therefore, with the structuring or arrangement of the collected data as a way of facilitating their easy retrieval. A major method of representation is indexing, through which the documents so classified are essentially provided with the keywords and metadata associated with them. Indexing may be automatic, where an algorithm looks for important words using metrics such as frequency and importance, or manual, where a human in the capacity of an editor assigns relevant terms on their own to the documents they index. Besides this, representation involves abstracting, where the documents are summarized to make for a quick grasping of their contents, and bibliographic descriptions, which provide info on, minimally, the author, title, publication source, and other metadata. This ordered representation serves the purpose of ensuring swifter and surer information retrieval.

3. File Organization

Efficient file organization of the information is very important in the aspect of search speed and accuracy. Two basic methods of file organization are-

- **Sequential File Organization:** Documents are stored complete records, and thus retrieval is simple but quite slow for large datasets. For each document, in order to retrieve them, one has to

examine all documents, which is applicable for small collections and not efficient for larger-scale retrieval systems.

- **Inverted File Organization:** It stores document references with lists of terms for improved search efficiency. Rather than the whole document, it provides a list of keys with document references. This allows the system to directly locate all records with a specific keyword in them, greatly increasing the speed of retrieval.
- **Hybrid Approach:** A hybrid approach is automatically applied by many systems in which a sequential structure is used to track smaller and mostly accessed records, while for massive data retrieval an inverted index structure is used.

4. Query Processing

The process for information retrieval begins with a user entering a query that embodies the user's intent. Such queries can range from simple keywords to very complex search expressions. While these queries resemble structured database queries, in reality, they more typically return sets of results that are usually not just one. The system processes the query by matching it to the indexed data and ranking it according to their relevance. Natural language processing, and semantic query understanding, may reveal user intent with more granularity in advanced IR systems. The details of scoring models involved in ranking State features including keyword frequency or document relevance or contextual meaning. All these components are crucial in making sure the users can find accurate, and relevant results when looking for information. By constantly fine-tuning the steps involved in this process, modern search engines and retrieval systems can thus provide faster, more precise, and user-friendly search experiences.

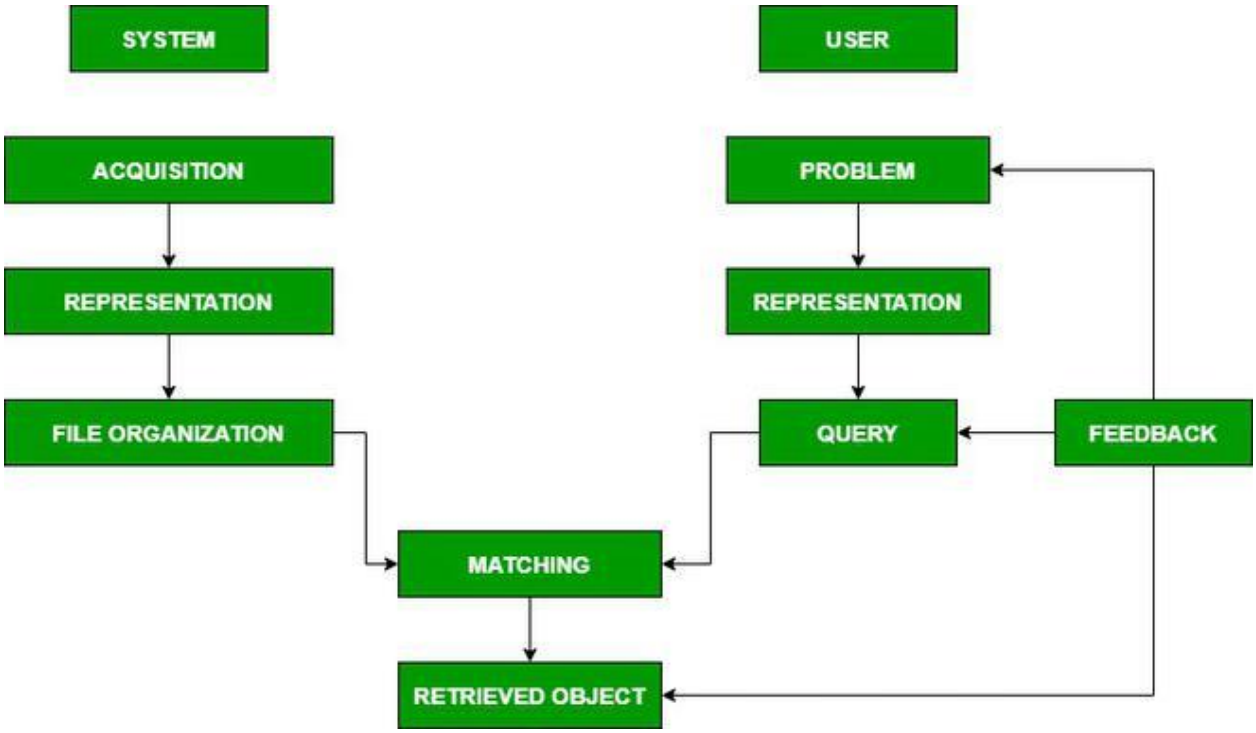


Fig. 8.1: Components of Information Retrieval

8.1.2 Importance of Information Retrieval

The importance of Information Retrieval (IR) has been established for managing and accessing huge amounts of data in different areas, with all the key research efforts focusing on methods to search,

categorize, and retrieve the numerous or relevant pieces of information from huge repositories. Below given are the key features emphasizing the importance of IR in different areas:

1. **Efficient Data Access**

The rapid growth of digital data in every sector has a tendency to create a need for finding relevant information at an accelerated pace. The IR systems fine-tune the retrieval process through indexing and ranking techniques so that users can have the best aligned results in fractions of seconds. Whether searching databases, online libraries, or file systems, IR in searching enables one to be more productive as time spent gathering specific information is drastically reduced.

2. **Search Engines**

Most techniques in today's search engines-cum-retrieval systems (Google, Bing, Yahoo, to name a few)-use IR techniques to go over a good number of web pages and return relevant results. For assuring accurate and contextually relevant search results, they analyze huge data on the web with keyword matching, semantic analysis, and ranking mechanisms involved. Without IR, fetching what is precise and particular would have been almost impossible on the web, making the task of searching lengthy and nonproductive.

3. **Enterprise and Academic Applications**

- Business and Enterprise Knowledge Management: Organizations are responsible for generation of vast amounts of data on the inside in the form of reports, emails, customer records, etc. Effective management and retrieval of crucial business information through IR systems improve decision-making and operational effectiveness.
- Academic Research: IR-based digital libraries and databases (e.g., Google Scholar, IEEE Xplore, PubMed) are largely relied upon by researchers to access scholarly articles, conference papers, and journals, thereby easing knowledge discovery and speeding up research advances.

4. **Natural Language Processing (NLP)**

The merging of information retrieval and natural language understanding examines AI-based applications that are based on human language processing. The developments in this area include the following:

- **Chatbots and Virtual Assistants:** AI-powered assistants, such as Siri, Alexa, and Google Assistant, employ IR to find relevant responses for driven user interaction.
- **Automated Customer Support:** Many have adopted AI chatbots that utilize IR strategies for retrieving solutions from a knowledge base to cut down the number of human interventions needed while also improving the efficiency of customer service.

5. **Recommendation Systems**

Within e-commerce, entertainment, and social media platforms are powered by information retrieval recommendation engines. Examples include:

- **E-commerce (Amazon, eBay):** It provides product proposals based on what users have searched for, their preferences, and prior interactions.
- **Streaming Services (Netflix, Spotify and YouTube):** It provides personalized movie, music, and video recommendations using user activity for analysis.
- **Social Media (Facebook, Instagram, Twitter):** It customizes an individual user feed based on user engagement and current interests.

6. **Big Data Analytics**

In information retrieval, very often, the foundation is to process and analyze large volumes of data. As such, it supports:

- **Data Mining:** Extraction of meaningful patterns and insights from these unstructured data sources, including social media, logs, and web traffic.
- **Business Intelligence:** Analyzing trends, editing customer feedback, and analyzing market behavior for osteo-physical assistance to facilitate and proper business decisions.
- **Healthcare and Scientific Research:** Facilitating retrieval of patient records, genomic data, and scientific papers for doctors and researchers.

8.1.3 Real World Applications of Information Retrieval

Information retrieval (IR) is being applied broadly across various domains to make things easier for information access, streamline processes, and improve decision-making. From web search engines to specialized domains like healthcare, legal research, and social media, it's one of the most prominent implementations of IR.

Some of the prominent apps of IR are:

1. **Web Search Engines**

Search engines like Google, Bing, and DuckDuckGo rely on information retrieval techniques to fetch the most suitable results from billions of web pages. These search engines deploy advanced algorithms such as:

- **Indexing:** Information retrieval crawlers will scan the entire web and store the information in a structured form so it can be retrieved quickly.
- **Ranking Algorithms:** PageRank and other machine-learning-based ranking algorithms prioritize results based on their relevance, popularity, and behavior of the users.
- **Query Expansion and Suggestions:** The search engines make the user experience better by doing such things as giving them suggestions for autocompleting words, suggesting synonyms, and related queries.

By the help of IR, these platforms allow users to retrieve accurate information within milliseconds, which is very efficient for users in navigation of websites.

2. **E-commerce Websites**

E-commerce portals such as Amazon, eBay, and Alibaba leverage information retrieval to enhance product searches, recommendations, and customer experiences. The various IR techniques used in e-commerce are as follows:

- **Keyword Matching & Semantic Search:** Users can find the most relevant products via the search terms.
- **User Behavior Analysis:** Suggestions for personalized recommendations based on past purchases, wishlists, and browsing history.
- **Visual Search and Voice Search:** Some platforms incorporate image recognition into IR-based AI models and enable voice assistants to search for products efficiently.

In general, information retrieval enhances customer engagement and thus boosts sales by augmenting search and recommendation systems.

3. **Digital Libraries and Scholarly Searching**

Academic databases like Google Scholar, IEEExplore, PubMed, and SpringerLink use techniques of information retrieval(IR) to help academicians find research papers, books and scholarly articles. Techniques of Information Retrieval in digital libraries include:

- **Metadata-based Indexing:** Organizing content based on titles, authors, abstracts and keywords for efficient retrieval.
- **Citation Analysis:** Ranking articles based on citation counts; this assists researchers in pinpointing influential papers.
- **Full Text Search and Semantic Retrieval:** Searching allows users to search entire documents, ensuring access to relevant literature.

These services open the gates of vast academic stores to researchers, enabling them to get information as quickly as possible.

4. **Healthcare and Biomedical Research**

IR in the medical field and healthcare is key, including the process of extracting vital information from large data sets. Some use cases include:

- **Electronic Health Records (EHRs):** Epic, Cerner; the respective companies have utilized IR to quickly yield patient records, allowing them to employ those in helping doctors make data-driven decisions.
- **Medical Literature Search:** Equipped with tools like PubMed and ClinicalTrials.gov, professionals in the medical field gather clinical studies, drug research and treatment protocols.
- **Disease and Drug Discovery:** Biomedical Research leverages IR to analyze genomic data, drug interactions and diseases trends for faster discoveries.

5. **Social Media and Online Platforms**

The likes of Facebook, Twitter, Instagram, LinkedIn and YouTube use IR techniques for building personalized experiences for their users. Key applications include:

- **Content Recommendation:** IR refers to the process of suggesting relevant posts, videos and articles based on user engagement and slight preference.
- **Targeted Advertising:** Ads are shown via algorithms using IR to analyze user behavior, interest and demographics.
- **Trend Analysis & Sentiment Detection:** Platforms use IR integrated with NLP for trending topic monitoring, fake news detection and engagement of public sentiment on different issues.

Through this, social media websites enhanced content discovery, increase user engagement, and generate revenue from advertisement.

6. **Legal and Compliance Research**

Information retrieval is used by law firms, government agencies, and regulatory bodies to search and analyze legal documents, case laws, and compliance records. Applications include:

- **Legal Case Search:** Westlaw and LexisNexis allow attorneys to efficiently find precedents and case laws.
- **Compliance Monitoring:** Thus, organizations may use information retrieval to monitor changes in regulatory issues and ensure compliance.

- **Contract Analysis:** There is an AI model, powered by IR, that analyses legal contracts on risk assessment and compliance validation.

IR would reduce both the time expended on manual document review by legal counsel and bolster the accuracy of legal decision-making.

8.1.4 Difference between Data Retrieval and Information Retrieval

While data retrieval and information retrieval share a common goal of fetching stored information, they differ in their approach and intended use:

Table 7: Difference between Data Retrieval and Information Retrieval

Feature	Data Retrieval	Information Retrieval
Focus	Extract data matches	Relevant information
Query Type	Structured (SQL Queries)	Unstructured or semi-structured
Example	Fetching records from a database	Search engines, Recommendation systems
Processing	Deterministic	Probabilistic and Heuristic

While regular relational databases perform data retrieval in a manner where queries yield specific data records, IR systems implement ranking mechanisms and probabilistic models leading to retrieving the most relevant document or result after ascertaining user intents.

8.1.5 Evolution of Information Retrieval

This practice has undergone several evolutions over time, passing through primitive manual indexing methods to search strategies fueled by artificial intelligence:

- **1950s-1960s:** The first developments of keyword-based searches and Boolean retrieval models.
- **1970s-1980s**: Probabilistic and vector space models were proposed to augment ranking and relevance.
- **1990s**: The ascendance of web search engines and techniques for large-scale indexing.
- **2000s:** PageRank and machine learning-based ranking systems were introduced.
- **2010s-present:** Advances in deep learning, neural search, and AI-based semantic search models, such as BERT and GPT.

Because of the complex characteristics and increasing volume of digital information, continuous innovations have been borne out in IR technologies that improved efficiency and given birth to search and recommendation systems more centered on users.

8.2 Index Construction

Index construction is about logically and practically organizing data, which aids the rapid and efficient retrieval of relevant information. An index can be conceptualized as a document collection within the realm of information retrieval in a type of roadmap connecting the terms to the occurrences of words within those documents. This avoids the need for searching sequentially through the document or catalog. After all, indexing stands paramount for large-scale search engines, databases, and information retrieval systems as it alters query response times drastically and increases search relevancy. The usually used indexing form is the inverted index, which represents the mapping of words to its containing documents, which allows for a quick lookup and rankings. Unlike typical databases that rely on structured indexing (for example, B-Trees or Hash Indexes), IR systems make use of more advanced indexing techniques dealing with word frequency and relevance scores, thus ranking documents accordingly.

The creation of indexes that function well goes through diverse processes, which include but are not limited to tokenization, normalization, removal of stop words, lemmatization, or stemming, and compression. To thrive in the digital world, modern indexing techniques should provide a structured mechanism for cataloging documents such that new or modified documents get indexed in real time without degrading the existing performance of the system. Distributed indexing frameworks like Apache Lucene, Elasticsearch, and Solr facilitate parallel processing and partitioned dataset management for constructing large datasets over multiple servers, fostering scalability and fault tolerance. Furthermore, these major developments enable indexing methods through machine learning and AI, thereby developing semantic search that grants the search engines the ability to realize the searcher's intent even beyond simple keyword matching. Nevertheless, due to the explosive growth of data, optimization in index construction becomes a huge challenge to information retrieval since a trade-off exists between storage efficiency and a trade-off between processing speed and retrieval accuracy.

8.2.1 What Is an Index?

The index is a specialized data structure designed specifically to improve searching efficiency by relating terms to their locations within a collection of documents. The main purpose of the index is to help the search engine quickly point to potentially relevant terms without needing to scan through the entire documents, thus minimizing the complexity and latency of the search.

Major advantages of indexing include:

- **Quick Retrieval:** The search engine can fetch results in seconds because of pre-processed data.
- **Optimized Storage Use:** Indexes save memory by ordering the data.
- **Reduced Workload:** Unneeded information is, of course, not scanned, thereby minimizing the workload.
- **Better Ranking and Scoring:** Based on the indexed terms, relevance scores can be assigned for the documents.

It is most commonly used for the fundamental mechanisms of search engines, databases, and large information systems for accomplishing queries in a quick and precise manner.

8.2.2 Types of Index

Various kinds of indexes can be worked on, which depend upon the kind of search and retrieval operation occurring. Below are a few of them that are common:

1. Inverted Index

An inverted index is one of the most popular indexing methods, mainly in search engines and text-based databases. The inverted index maps a word (a term) to the document in which it occurs, allowing a quick retrieval of the relevant content.

An inverted index structure could be mentioned as follows:

- **Vocabulary:** A listing of all distinct terms in the collection of documents.
- **Posting List:** For each term, a listing of document IDs (and, sometimes, positions) in which the target term occurs.

Example:

Given three documents:

- **Doc1:** "The cat sat on the mat."
- **Doc2:** "The dog sat on the mat."

- **Doc3:** "The cat and the dog played together."

An inverted index would look like this:

Term	**Documents**
cat	Doc1, Doc3
dog	Doc2, Doc3
sat	Doc1, Doc2
mat	Doc1, Doc2
played	Doc3

Advantages:

- Fast retrieval of keyword-based searches.
- Boolean queries (AND, OR, NOT operation).
- Efficient for large-scale text search engines like Google and Bing.

2. Forward Index

A forward index is a mapping of document IDs and word lists, instead of mapping words to documents as in an inverted index.

Structure of forward index:

- Each document receives a unique document ID.
- Words are indexed in the order they appear in the document, typically with word frequencies and word position boundaries.

Example:

Document	**Words (with frequency)**
Doc1	(cat:1, sat:1, mat:1)
Doc2	(dog:1, sat:1, mat:1)
Doc3	(cat:1, dog:1, played:1)

Advantages:

- Useful for ranking documents on the basis of term frequency and importance of words.
- Supports phrase searching and context-aware retrieval.
- Works nicely in concert with inverted indexes to score documents.

Disadvantages:

- Conducting searches may be generally slower for large data sets as it may require each document to be scanned.
- Not optimized toward Boolean retrieval like an inverted index.

3. Suffix Trees

A suffix tree is a specific indexing structure that helps in substring searching, pattern matching, and text compression. It has widespread applications in bioinformatics, plagiarism detection, and DNA sequence analysis.

Structure of Suffix Tree:

- A tree representation where a node contains all the suffixes of a text.
- Every branch represents a substring of the original text.
- Enables fast substring matching in O(m) time, where m is the length of the query.

Example:

For the string **"banana"$**, the suffix tree would store:

- "banana"
- "anana"
- "nana"
- "ana"
- "na"
- "a"

Applications:

- **DNA Sequence Matching:** In bioinformatics, to compare genome sequences.
- **Plagiarism Detection:** It helps in detecting similar patterns in a text.
- **Text Autocompletion:** Prefix-based searching for efficient requests in search engines.

Advantages:

- Very fast substring searches (versus brute-force methods).
- Modular to reduce the redundancy in the second step.

Disadvantages:

- High memory requirement in the case of big databases nowadays.
- The construction of a suffix tree is hard and time-consuming.

4. Signature Files

A signature file is an alternative indexing method employed in large document databases. It generates "signatures," which are short, readable representations for each document based on the words they contain, rather than connecting words to documents like an inverted index.

How it Works:

- Each document is transformed into a bit-string (or signature), based on the presence of words.
- Subsequently, bit-wise operations are performed to rapidly filter the documents.
- Once a match is found, the actual document is scanned for a final check.

Example:

If we have three documents:

- **Doc1:** "Information retrieval is efficient."
- **Doc2:** "Efficient algorithms improve retrieval speed."
- **Doc3:** "Speed and efficiency matter in retrieval."

A simplified signature file representation might look like this:

Document	**Signature (Bit-String)**
Doc1	10101001
Doc 2	11001010
Doc 3	11110000

A search query for **"efficient retrieval"** would check the bit patterns for matching **bits set to 1**, quickly narrowing down candidate documents before performing a detailed search.

Advantages:

- Memory efficient compared to an inverted index method.

- Fast filtering for approximate searches.

Disadvantages:

- False positives need to be additionally verified; thus, it can slow retrieval time down.
- Is not suitable for exact keyword search like an inverted index.

Indexes are the backbone of efficient information retrieval as they provide fast and scaling methods of search solutions. Right indexing strategy improves search engine, databases, and information systems performance by reducing search latency and hence, enhance user experience.

8.2.3 Steps in Index Construction

1. Tokenization

Tokenization is the process of breaking a large text document into individual **words, terms, or tokens** that serve as units of indexing.

Process of Tokenization:

- The text is **split based on spaces, punctuation, or special characters**.
- Each word is treated as a **separate entity** for indexing.
- Complex tokenization may involve **handling contractions, hyphenation, and compound words**.

Example:

Original text: "Information retrieval improves efficiency in searches."

After tokenization: ["Information", "retrieval", "improves", "efficiency", "in", "searches"]

Challenges:

- **Multi-word expressions:** "New York" should be treated as a single token, not two words.
- **Apostrophes & Hyphenation:** Handling words like **"can't" → ["can", "not"]** or **"state-of-the-art" → ["state", "of", "the", "art"]**.
- **Different languages:** Some languages (e.g., Chinese, Japanese) do not use spaces, requiring special segmentation techniques.

2. Normalization

Normalization ensures **uniformity in text representation** by converting words to a consistent format. This helps in improving search accuracy.

Techniques Used:

1. **Case Folding:** Converts text to **lowercase** to avoid case sensitivity issues.
    - Example: "Search" and "search" should be treated as the same term.
2. **Stemming:** Reduces words to their **root form** by removing suffixes.
    - Example: "running" → "run", "retrieval" → "retriev".
    - Common algorithms: **Porter Stemmer, Snowball Stemmer**.
3. **Lemmatization:** Converts words to their **dictionary base form** (more accurate than stemming).
    - Example: "better" → "good", "running" → "run".
    - Uses linguistic knowledge and **WordNet** databases.

Example:

Before normalization: ["Running", "retrievals", "Efficiently", "Improvement"]

After normalization: ["run", "retriev", "efficient", "improve"]

Benefits:

1. Reduces **redundancy** by treating similar words as a single term.
2. Improves **matching accuracy** in search queries.

3. Stopword Removal

Stopword removal eliminates **common words** (e.g., "the," "and," "is") that **do not contribute to the meaning** of the search query.

Why Remove Stopwords?

- **Reduces index size** by eliminating unnecessary terms.
- **Improves retrieval speed** by focusing on important keywords.

Example:

Original text: "The quick brown fox jumps over the lazy dog."

After stopword removal: ["quick", "brown", "fox", "jumps", "lazy", "dog"]

Challenges:

- Some stopwords **may be important** in certain contexts.
- Example: "To be or not to be" → If "be" is removed, the meaning is lost.
- Some modern search engines **keep stopwords** to improve natural language search.

4. Indexing

Indexing involves **storing the processed text in a structured format** that enables **quick search and retrieval**.

Types of Indexes Used:

- **Inverted Index:** The most common type, mapping words to documents.
- **Forward Index:** Stores document-wise token lists.
- **Suffix Trees & Tries:** Used for substring searches.

Example (Inverted Index):

For the documents:

- **Doc1:** "Information retrieval improves search."
- **Doc2:** "Search engines use retrieval techniques."

Term	**Documents**
Information	Doc1
retrieval	Doc1, Doc2
imposes	Doc1
search	Doc1, Doc2
engines	Doc2

Benefits:

1. Enables **fast keyword lookup**.
2. Reduces **computational cost** of searching.
3. Supports **Boolean and ranked queries**.

5. Compression

Compression techniques reduce **storage requirements** while maintaining **fast access to index data**.

Techniques Used:

1. **Huffman Encoding:**

- Assigns **shorter binary codes** to frequent terms.
- Example: "retrieval" (high frequency) might be stored as **"010"**, while "complex" (low frequency) as **"1101001"**.

2. **Delta Encoding:**

- Stores **differences between consecutive numbers** instead of absolute values.
- Example: Instead of storing **[100, 105, 110, 120]**, store **[100, +5, +5, +10]**.

3. **Bitwise Compression (Run-Length Encoding):**

- Stores **sequences of repeated values efficiently**.
- Example: **"000011110000"** becomes **"4-0s, 4-1s, 4-0s"**.

Advantages:

1. Reduces **disk space usage**.
2. Improves **query response time** (smaller index = faster lookups).
3. Enables **efficient caching in memory**.

6. Updating Indexes

Search engines need to **dynamically update** indexes as **new content is added or removed**.

Key Challenges in Index Updates:

- **Real-time indexing:** Websites and social media require instant updates.
- **Handling deletions:** Old or outdated content must be removed efficiently.
- **Incremental indexing:** Instead of rebuilding the entire index, only modified documents are updated.

Techniques Used:

1. **Dynamic Indexing:**

- Updates indexes **without rebuilding them from scratch**.
- Uses **delta updates** (small incremental changes).

2. **Merge-Based Indexing:**

- Maintains a **small active index** for recent updates.
- Periodically merges the **small index with the main index**.

3. **Real-Time Search Indexing (RTSI):**

- Used by Google, Bing, and Twitter.
- Uses **in-memory indexing** to update results within **milliseconds**.

Example:

- A new blog post is published → Search engine **adds it to the index immediately**.
- An outdated article is removed → Search engine **removes its reference from the index**.

Benefits:

1. Enables **fast content discovery** (Google updates within seconds).
2. Ensures **search results remain fresh and relevant**.
3. Reduces **index rebuilding time and costs**.

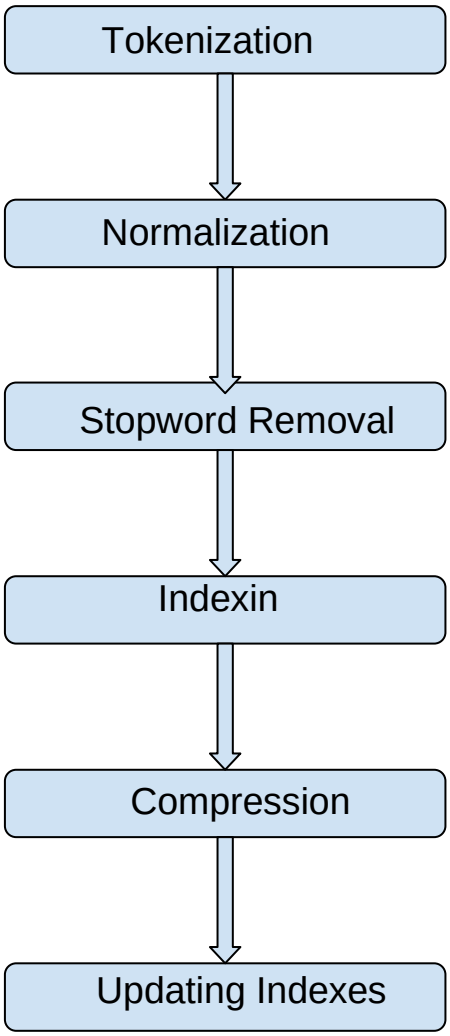


Fig. 8.2: Steps in Index Construction

8.2.4 Distributed Indexing

Modern search engines handle large amounts of data, and for that, it becomes necessary to distribute the indexing among multiple servers. Distributed indexing is just how such a distribution occurs for the efficient storage of and retrieval of indexed data since the work will be handled by different nodes. Such an arrangement enhances scale-up and fault tolerance, where slowdowns or failures on the system do not occur due to an excessive load.

In order to execute distributed indexing, search engines use advanced technologies like Apache Lucene, Elasticsearch, and Solr, which allow horizontal scalable indexing with speed and accuracy. Technologies that enable distributed index building and storage which allow fast search execution are therefore used. In the use of distributed indexing, search engines can handle billions of web pages, user-generated content, and structured databases quickly, producing results in milliseconds.

8.2.5 Issues in Index Construction

Index construction is a basic process in information retrieval that poses various challenges. Some of the most critical issues therein include:

1. Scalability;
2. Dynamic Updates;
3. Space Efficiency;

4. Multi-Language Support.
5. Fault Tolerance

- **Scalability:** Scalability refers to how a system that deals with an immense amount of data can handle the growing dataset in terms of searchability. That is, some search engines have to deal with massive datasets containing billions of documents. Of course, the more documents there are to be indexed, the more capability the system has to scale. In addressing this problem, distributed indexing, that is, partitioning data across multiple servers for parallel processing, does exist-to a certain extent-to remove the bottleneck. Still, coordination and maintenance of these partitions, together with consistency maintenance, is a big challenge.
- **Dynamic Updates:** Information in the Internet is exposed to constant change; Web pages are updated, deleted, or newly created. Search engines, therefore, must support real-time indexing, which enables new content to be indexed as instantly available; this refers to the immediacy to index without a total re-indexing. This means handling frequent updates efficiently and ensuring queries return up-to-date results.
- **Space Efficiency:** At the same time, another major issue regarding index construction is space efficiency. Keeping such an index requires vast disk space. Search engines optimize the trade-off between index compression and query performance. A highly compressed index saves storage space, but searches would be slower because of decompression overhead. The retaining of uncompressed data increases storage requirements at the same time speed of retrieval would be retained. So, the challenge lies in maintaining a right balance between these two.
- **Multi-Language Support:** The web exists in a polyglot world through character sets and different writing systems. The indexing system therefore should efficiently incorporate several varied linguistic structures word variations encoding formats. Some languages have such a morphological engine minutiae embedded into their own structure (like German Compound words), while others such as Chinese do not even use spaces between their words. So, multilingual and language issues become another critical requirement pertaining to search indexing.
- **Fault Tolerance:** A distributed indexing scheme needs to offer extremely high fault tolerance. More than one server can go down, but the search engine must still be able to function properly, with no data loss or performance degradation. Certain fault tolerance mechanisms, such as replication (which involves creating multiple copies of an index) and data recovery techniques, make sure that the systems are always reliable and available in the event of storage device failures or even network outages.

8.2.6 Indexing in Modern Search Engines

To improve speed, accuracy, and scalability, modern search engines are equipped with a variety of sophisticated indexing techniques. Here are just a few of the most effective:

- **Sharding:** Sharding means the division of the index into smaller pieces, which are stored among many servers. A shard includes a subset of the total indexed reports, so it becomes possible to process many shards in parallel. This increases the search efficiency tremendously and allows for great scalability of the data being processed. For example, when one billion web pages are

indexed by a search engine, it might break them into ten shards, thus holding average 100 million pages. Sharding saves one single server from being overwhelmed.

- **Replication:** If there is a failure of the server on which any particular shard is present, access to that shard is not available. This is why search engines create multiple copies of each index for redundancy and reliability. A replica of the shard on some other server can step into action and take over to help avoid service unavailability. Another side of replication is that it helps improve performance during query handling. This is due to the fact that, with several copies of the index in action, a concurrent approach to execution can be achieved, reducing time taken and thus increasing response time.
- **Caching:** Caching is one technique that creates a memory space for the most frequent search results to enable faster single-time response time. Instead of trying to recalculate and compute search results every time the user issues a query, the search engine retrieves results from cache that were computed beforehand for this particular search query. Very useful with popular queries that receive repetitive searches because this saves a lot of computational power and improves user experience.
- **Machine Learning-Based Index Optimization:** On the machine learning side, use cases in search engine optimization will be AI-driven and dynamically adjust things based on observed behaviors. ML model after this scoring will review the queries and user interactions to derive indexing strategies. To illustrate, we may allow AI to reveal those documents it thinks are more likely much more relevant for these queries and to modify the priority for indexing accordingly. This kind of adaptive indexing lets you find out the most relevant stuff to act on much more quickly and thus increases the relevance of the results returned, together with user satisfaction.

Hence, Distributed indexing forms the very heart of modern search engines, enabling them to handle vast amounts of data in a practical way. Following different strategies such as sharding, replicating, caching, and machine-learning optimization, search engines can offer high-speed, scalable, and fault-tolerant search experiences. Nevertheless, index construction comes with challenges, including scalability, dynamic updates, space efficiency, multi-language support, and fault tolerance. Addressing these challenges is important in keeping the search process fast, accurate, and reliable, given the increasing volumes of information in the digital world.

8.3 Scoring Models

Scoring model is considered to be one of the crucial ingredients of an information retrieval system. This model is used to determine which document in response to a user's query is most relevant for inclusion, based on that document's content, structure, and context or meaning. Each document is assigned a numerical score based on its content, context, and relational fit to the query so that documents in the returned output are ordered or ranked. General aim is to make sure relevancy-wise document with good score appears at the top of Search Engine Result Page (SERP).

8.3.1 Popular Document Scoring Methods

1. **TF-IDF (Term Frequency-Inverse Document Frequency):** TF-IDF stands for Term Frequency-Inverse Document Frequency. And it is actually a measure of document scoring and ranking used in information retrieving and search engines. In simple words, it helps understand the importance of a term inside a document about the entire collection of documents itself. This,

more specifically, identifies words very relevant to a specific document while simultaneously reducing the collective influence of very overused words present across almost every document.

The first part is TF (Term Frequency), which works by how often a term occurs in a given document since the higher the number of occurrences, the more important that word is for understanding the content of the document. Very frequent words like "the," "and,"" or "is" appear across almost all documents, rendering them almost useless in distinguishing documents from one another.

To counter this, we have the IDF (Inverse Document Frequency), which aims to determine how rare a term is in a whole collection of documents. If the word is going to appear in many documents, it receives a low score for its inverse document frequency and thus does not contribute much weight to the document's ranking. Conversely, if it appears in very few documents, a very high weight is set on it, showing its importance in those documents. The final TF-IDF score is calculated as: These three components combined form the calculation of the TF-IDF score of a particular word in a single document out of any documents.

The final TF-IDF score is calculated as:

$$\text{TF-IDF} = \text{TF} \times \text{IDF}$$

where:

- **TF (Term Frequency)** = (Number of times the term appears in the document) / (Total number of terms in the document)
- **IDF (Inverse Document Frequency)** = log (Total number of documents / Number of documents containing the term)

The scoring method has found broad applications in search engines, text mining, and natural language processing for ranking documents according to relevance to a user's query. It ensures that rare yet meaningful words are given more weight, thereby enhancing the quality and accuracy of search results.

A simple example leads us to three plain documents with the following:

Doc1: "Artificial intelligence and machine learning."

Doc2: "Machine learning upgrades artificial intelligence."

Doc3: "Deep learning is a subclass of artificial intelligence."

Since the word "machine" appeared two times in Doc1, one time in Doc2, and never in Doc3, that means Doc1 will have higher TF-IDF code than the others.

Strengths:

- Simple and efficient keyword ranking.
- Best for small-scale search engines.

Weaknesses:

- It does not take into account the arrangement of words and semantics.
- It cannot deal with synonyms and context-based meaning.

2. **BM25 (Best Matching 25):** BM25 (Best Matching 25) is one of the newer and advanced algorithm for document scoring. It keeps term saturation and document length normalization, which was developed based on the limitation of TF-IDF, while improving the ranking of

documents. BM25 thus becomes one of the most effective ranking functions in modern search engines.

How BM25 Improves TF-IDF:

- **Term Saturation:** One basic problem with TF-IDF is that it assumes a linear relationship between term frequency and relevance. The more a word recurs in a document, the further it becomes important within it. However, practically, this is not true, and the importance of a document is clustered as a function of automatic term frequency cutoff makes it add unfair weight to spanning documents. BM25 tries to lessen this a little by having a saturation function: in simple terms, BM25 makes extra repetitions of terms contribute less and less to the overall score after a certain point. It brings its advantages when viewing the results by not favoring documents that actually just repeat a term excessively by fair scanning and aiming for ranking of results.
- **Document Length Normalization:** Another deficiency with TF-IDF is that as the length of a document increases, so do the term frequencies, which puts them at an unfair advantage for scoring. Longer documents will contain larger quantities of words, and naturally, they could take a lead over shorter documents not with regard to the actual relevance to the query, but rather the sheer number of words contained therein. BM25 constitutes this by normalizing document lengths, which is achieved by altering the term frequency according to document length. This permits a fair evaluation of both short and long documents, enabling concisely written content not to be placed at an incorporated disadvantage.

BM25 FORMULA

BM25 refines TF-IDF by using the following scoring function:

$$\text{BM25} = \sum \left(\frac{\text{TF} \times (k+1)}{\text{TF} + k(1 - b + b \cdot \frac{L}{L_{avg}})} \times \text{IDF} \right)$$

where:

- **TF (Term Frequency)**: Number of times the term appears in the document.
- **IDF (Inverse Document Frequency)**: Weight assigned based on how rare the term is across documents.
- **k (Saturation Parameter)**: Controls how much term frequency affects the score (typically set between 1.2 and 2.0).
- **b (Length Normalization Parameter)**: Adjusts for document length (typically set to 0.75).
- **L (Document Length)**: Total number of words in the document.
- **L_avg (Average Document Length)**: Average word count of all documents in the collection.

The integration of TF-IDF with term saturation and document length normalization has seen BM25 score performance vastly improved, as such making it stronger and more robust for any modern search engine. This ensures the relevant ones are rated higher, without overemphasizing long or repetitiously redundant content, thus making document retrieval a bit more balanced and effective.

In an example where the query is "machine learning", BM25 is meant to assign higher grades for documents that contained these terms nominally, instead of by shallow counting number of occurrences.

Pros:

- More efficient than TF-IDF for weaving through real searches since 2000.
- Makes much sense for natural language queries.

Cons:

- Needs hyper-parametrical tuning (k1 and b).
- Still does not consider word meaning like deep learning models do.

3. **Vector Space Model:** The Vector Space Model (VSM) is a widely used approach in information retrieval that represents documents and queries as vectors in a multi-dimensional space. This model allows search engines to determine the relevance of documents by computing the similarity between a query and each document in the collection. The ranking is primarily based on cosine similarity, which measures how close or similar two vectors are within the space.

- **Representation of Documents as Vectors:** In VSM, each document is represented as a vector, with each dimension corresponding to a unique term in the entire collection. The value in each dimension is generally determined using TF-IDF or other weighting schemes. The idea is that documents with similar content will have similar vector representations, thus making it easier to compare them mathematically. The document will have three terms, namely (car, engine, fuel). It is represented in a three-dimensional space, and each term has values that correspond to their weights based on how important each term is. A similar example can be observed: in this case, the query is car engine efficiency. This would then also be converted to a vector in the same space so that comparisons between document vectors can be made.
- **Cosine Similarity for Ranking:** Cosine Similarity is used to measure the relevancy of this document concerning the user query. It measures the angle rather than the absolute distance between the two vectors. Thus, the lesser the angle value, the more appropriately the document will align with the query.

The cosine similarity formula is given by:

$$\cos(\theta) = \frac{\sum(A_i \times B_i)}{\sqrt{\sum A_i^2} \times \sqrt{\sum B_i^2}}$$

where:

- **A** represents the document vector.
- **B** represents the query vector.
- **Σ (A_i × B_i)** is the dot product of the two vectors.
- **Σ A_i²** and **Σ B_i²** are the magnitudes of the document and query vectors.

The resulting score ranges from **0 to 1**, where:

- **1** means the document is an exact match to the query.
- **0** means there is no similarity between the document and the query.

Example:

Doc1 Vector: (0.5, 0.3, 0.7)

Doc2 Vector: (0.1, 0.9, 0.6)

Query Vector: (0.4, 0.2, 0.8)

The document having the highest cosine similarity score should be ranked with the highest score.

Strengths:

- Very effective for computing similarity between documents.
- Good for semantic search tasks.

Limitations:

- Computationally too demanding for big datasets.
- Does not consider synonyms or relations on the basis of meaning.

4. **Learning-to-Rank (LTR):** LTR is a structure used to implement learning methods on how documents should be ranked by search engines. Most ranking methods rely on heuristics typically represented in the form of scores: TF-IDF or BM25. LTR, on the other hand, uses learning models to improve ranking based on user interactions, query patterns, and feedback. It evaluates previous user behavior, including click-through rates, dwell time, and query reformulations, to boost relevance in results of search engines.Learning to Rank is very beneficial in modern search engines, recommendation systems, and e-commerce platforms, as the quality of ranking directly affects user satisfaction. Some of the most commonly used learning models for Learning to Rank include RankNet, LambdaMART, and XGBoost.

Types of Learning to Rank (LTR):

- **Pointwise LTR**—regression-based treatment of ranking (predicting scores for every document).
- **Pairwise LTR**—learns from document pairs, where the task is to optimize which one should be ranked higher.
- **Listwise LTR**—takes into account the complete list of results and optimizes rankings globally.

Popular Algorithms of LTR:

Several machine learning algorithms are commonly used to implement **LTR** in search engines:

1. RankNet (Neural Network-based LTR)

- **Developed by Microsoft**, RankNet is a **pairwise LTR model** based on neural networks.
- It **compares pairs of documents** and determines which one should be ranked higher based on user interaction signals.
- Can learn non-linear ranking patterns through deep learning.

2. LambdaMART (Gradient Boosted Decision Trees - GBDT)

- An extension of **LambdaRank**, LambdaMART is a **listwise** ranking model that uses **Gradient Boosted Decision Trees (GBDTs)**.
- It is highly effective for ranking problems and is widely used in search engines like **Yahoo! and Bing**.
- More interpretable and efficient than neural network-based approaches.

3. XGBoost (Extreme Gradient Boosting)

- **XGBoost** is a powerful **gradient boosting algorithm** widely used in **search engines and recommendation systems**.
- It is optimized for **speed and accuracy**, making it a popular choice for ranking tasks.
- Handles large-scale ranking problems efficiently and supports feature importance analysis.

Example:

Through LTR, Google's ranking system uses a combination of imperfect signals, clicks, dwell time, and bounce rate to feed the search results dynamically.

Strengths:

- Matches in-the-field usage of users.
- Outperforms the classical models (BM25, TF-IDF).

Limitations:

- Requires a great deal of labeled training data.
- Computationally expensive to train.

5. **Neural Ranking Models:** Neural Ranking Models are some of the highest-performing advanced, deep-learning-based techniques for search ranking. They understand the semantic relationships between words in a query versus document. These are very different from older keyword-based methods, like TF-IDF and BM25. These neural search algorithms are generally based on neural networks and often use transformer-based architectures like BERT. Unlike older methods that just check for the number of occurrences of a word, these neural networks are able to understand and capture the meaning and context of words, thus being able to output and manage better and smarter search ranks.

   Neural Ranking Models are widely applied in search engines, question-answering systems, and recommendation engines to enhance search relevance, personalization, and user experience.

How Neural Ranking Models Work:

Neural Ranking Models employ context-aware embeddings to process and rank documents based on deepsemantic understanding. The process involves the following steps:

- **Context-Aware Embeddings:** Traditional search engines treat words as isolated terms while neural ranking models analyse the relations between words in a phraseThese models generate word embeddings (vector representations) that capture semantic meaning and improved ranking quality.

  Classic example: In a query like "best budget smartphones", neural models could generates two vectors that understand they both share the common meaning of "budget smartphones" in pointing to cheap pricing, whereas treating them each one on its own would not have the same impact of their interpretation.
- **Understanding Entire Phrases Instead of Isolated Words:** Instead of matching exact words, these models could interpret it that way: a phrase, a synonym, that contextually makes sense

  For example, a neural ranking model recognizes that "cheap flights" and "affordable air tickets" have the same intentThis acts beneficial in terms of enhanced result ranks when there is a mismatch of exact keywords along the content of a document.
- **Processing Complex Search Queries:** Neural models are far superior in solving long/ conversational / vague queries because traditional search methods face difficultiesRegular queries with direct meanings are handled nicely by more of traditional search approachesOther queries where there could be multiple contextual implications, say in case of queries: like one being "Apple store near me" means searching for an Apple retail location; case number two being "Apple storage issues," means referring to the issues with storage concerning their electronics.

Pros:

- Great understanding of natural language.
- Made use of in present-day search engines (Google, Bing...).

Cons:

- High computational-power demands (requires GPUs).
- Difficult to interpret why a certain model ranked a document very high or very low.

Such models perform a significant role within a modern-day search engine or a recommendation system, assuring user experience and querying efficacy.

Popular Neural Ranking Models

Several deep learning models have been developed for **neural search ranking**, with **BERT** being the most widely used:

1. BERT (Bidirectional Encoder Representations from Transformers)

- BERT **understands word relationships in both forward and backward directions**, making it highly effective for ranking search results.
- It considers **entire sentence context**, improving relevance compared to traditional term-matching methods.
- **Example**: If a user searches for *"can you get medicine without a prescription"*, BERT understands that **"without"** negates the meaning, ensuring accurate results.

2. T5 (Text-to-Text Transfer Transformer)

- T5 is a **sequence-to-sequence model** that can be fine-tuned for ranking by rephrasing queries and generating better search responses.
- It is particularly useful in **question-answering systems** and **passage ranking tasks**.

3. GPT-based Models

- Large **language models like GPT** can be used for **semantic search** and ranking, where results are retrieved based on meaning rather than exact word matching.
- **Example**: In a legal search engine, GPT models can retrieve **case laws** based on similar arguments, even if different terminology is used.

8.4 Complete Mechanism of a Search Engine

8.4.1 Overview of Search Engine

A search engine is a complex piece of software that retrieves and ranks such documents as its user queries have in mind. Its components work on interlinked duties, collecting the documents, ranking them, and returning results to users. The prime concern of a search engine is the retrieval of valid information from a vast storehouse in the least possible time.

8.4.2 Major Components of Search Engines

1. Web Crawling:

- A web crawler (spider or bots) traverses the internet with the help of links to collect web pages for indexing.
- Web crawlers follow rules like robots.txt and crawl delay to prevent server overloads on websites.
- Some popular frameworks used include Apache Nutch and Scrapy.

2. Indexing:

- The processed web pages are entered into the index to facilitate speedy future retrieval.

- Includes tokenization, stemming, stopword removal, and metadata extraction.
- These processes use inverted indexes and distributed indexing to better utilize space and permit more efficient retrieval.

3. Query Processing:

- The search engine interprets a user's query by stopping words, expanding such words into synonyms, and correcting spelling errors.
- Some search engines use query understanding models powered by artificial intelligence to interpret user intent.

4. Ranking and Retrieval:

- The search engine retrieves relevant documents from the index, scoring and ranking them using various scoring formulas like TF-IDF, BM25, or PageRank.
- Machine learning models like BERT have gained popularity with time, improving in relevance to ranking.

5. User Interface and Result Presentation:

- The interface provides users with a visual presentation of the ordered results.
- User experience is optimized further with features like search snippets, rich results, and autocomplete suggestions.

8.4.3 Search-Engine Architecture

A conventional search engine follows pipeline architecture:

1. **Crawling Layer:** Collects and updates web content.
2. **Parsing & Processing Layer:** Cleans and tokenizes the collected content.
3. **Indexing Layer:** Organizes content for quick lookup.
4. **Query Execution Layer:** Handles user requests and retrieves relevant information.
5. **Ranking Layer:** Ranks them according to relevance scores.
6. **User Interaction Layer:** Provides the search results to the user in a productive manner.

8.4.4 Challenges in Search Engine Development

- **Scalability:** This involves efficiently dealing with billions of documents.
- **Latency Optimization:** It is very important to improve the response time for real-time search queries.
- **Spam And Malicious Content Detection:** This involves filtering good content from harmful and bad content.
- **Personalization:** Delivering customized search results based on user preferences and history.
- **Multilingual Search:** Supporting queries coming from different languages and scripts.
- **Privacy And Data Protection:** It springs from getting users verified and securing the users' information according to the civil laws.

Modern search engines should augment AI with deep learning, vector searching, or reinforcement learning to provide genuinely accurate and personalized responses to the demands of its users in this era of digital primacy.

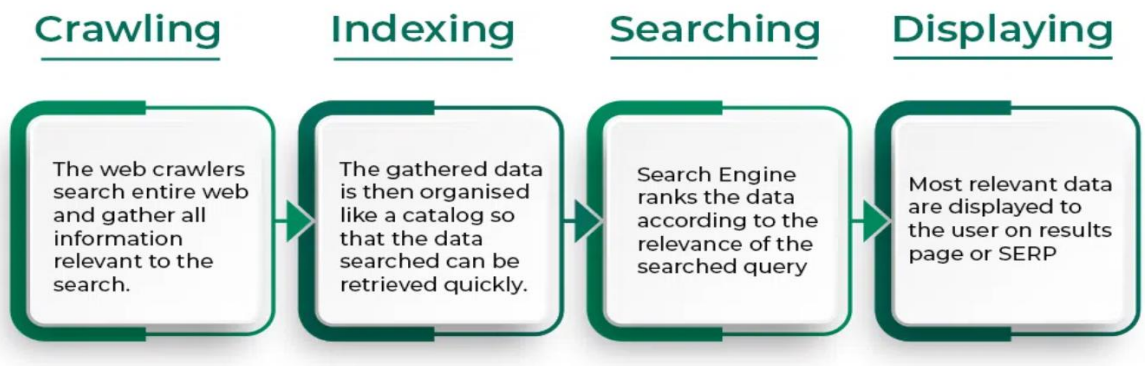


Fig. 8.3: Components of search engine

8.5 Evaluation Methods in Information Retrieval (IR)

Evaluation forms a very important step in Information Retrieval (IR) because it enables one to gauge how well a system meets user needs in terms of relevant and accurate search. Different measures and methodologies are available to evaluate the performance of Information Retrieval systems.

8.5.1 Effectiveness Metrics

a. Precision

Precision is the correctness of the retrieved documents. It is also the value assigned to the ratio to the number of relevant documents retrieved over all documents retrieved by the system.

$$\text{Precision} = \frac{\text{Number of Relevant Retrieved Documents}}{\text{Number of Retrieved Documents}}$$

High precision correlates with the degree of relevance of the documents retrieved to the query.

b. Recall

Recall measures how well a system identifies pertinent documents compared to all such documents available within a specific database.

$$\text{Recall} = \frac{\text{Number of Relevant Retrieved Documents}}{\text{Total Number of Relevant Documents}}$$

High recall indicates that the system retrieves nearly all available relevant documents.

c. F-Score (F1-Score)

F-Score is the measure of precision and recall combined. It provides a balanced measure of precision and recall when both measures are important.

$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

It's of particular value in situations when precision and recall cannot be traded off easily.

d. Mean Average Precision (MAP)

Mean Average Precision (MAP) computes the average precision over all the relevant documents in the collection for each query, finally averaging the resulting scores for all queries.

MAP is very essential in measuring systems in which the order of retrieval matters, such as in search engines, etc.

e. Normalized Discounted Cumulative Gain (NDCG)

NDCG measures the ranking quality of the retrieved documents. Hemispherically, higher weights are assigned to relevant documents at the top of the ranking, indicating the importance of retrieving the most relevant documents first.

$$NDCG_k = \frac{DCG_k}{IDCG_k}$$

where:

- DCG_k (Discounted Cumulative Gain at rank kkk) is the weighted sum of the relevance of documents retrieved.
- IDCG_k is the Ideal Discounted Cumulative Gain, which means the maximum possible DCG.
- NDCG is frequently utilized, for instance, in web search engines and recommender systems.

f. Mean Reciprocal Rank (MRR)

This metric is used to evaluate in what rank the first relevant document is found. Most a grade for evaluating the first document of interest alone pleases either the creator or the sponsor.

$$MRR = \frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{1}{rank_i}$$

Where:

- |Q| is the number of queries.
- $rank_i$ is the rank position of the first relevant document for the i-th query.

8.5.2 Efficiency Metrics

These metrics has an idea regarding the speed of how the IR system works in getting results.

a. Response Time: Refers to the total time taken by the IR system to search and display results after receiving a query. This metric is of utmost importance for real time search engines and systems demanding immediate answers.

b. Throughput: Throughput is the measurement of the number of queries or documents handled by the system during a given time frame. This plays an important role in assessing the scalability of the system.

c. Latency: A relation to delay time that stands up between the moment when the query is issued and when the results come back. For the web search or large database systems, a low latency can be very important for end-user experience.

d. Scalability: The ability of the IR system under examination to handle ever-growing amounts of data, queries, and complexity without any noticeable degradation in performance.

8.5.3 User-Centered Evaluation Methods

Most of the methods center on the user and focus on how effective the system is in meeting the needs and expectations of the user. Each method presents unique aspects of user-centered evaluation.

a. User satisfaction: User satisfaction is assessed through surveys, feedback, or even direct observation of users interacting with the system. Finding an outcome that is highly relevant for a user can be a measure of their satisfaction.

b. **Task Success Rate:** This indicates success in meeting the information retrieval task by the users, such as finding the right documents through the system or passing a problem-solving task with the system.
c. **Click-through rate (CTR):** The click-through rate (CTR) shows the number of times that a search result has clicked on against the number of times it is displayed. Higher values of the CTR can be an indicator that the system is able to provide results that are more relevant to the user.
d. **Time Spent:** Measurement on how long a user finds information can be a guideline on if a user found the results of the retrieved information helpful or if they struggled to find relevant results.

8.5.4 The Evaluation of Result Ranking in Search Results

Real-life searches and the information retrieval systems commonly return a ranked list of documents. Evaluating the effectiveness of these lists in ranked order is very important in determining how well they rank relevant documents.

a. **Ranked Precision:** This rank-order precision evaluates documents' relevance at various positions on the list. High precision at top ranks is often considered more essential than at lower ranks.
b. **Cumulative Gain:** CG is the total of the relevance scores of all items in the result list whereby documents with more relevance contribute higher scores.
c. **Discounted Cumulative Gain:** DCG subsumes the discounts with regard to the respective ranked order of documents in the list. This refers to the decreasing importance of each document at the lower position of ranking.

8.5.5 Evaluation of Information Retrieval Specific Tasks

Tasks of information retrieval have different evaluation methods.

a. **Web Search Evaluation:** Web search engines generally employ NDCG and MAP as ranking metrics to measure the overall effectiveness of their pages, finding in their metrics that they are evaluated based on precision, recall, and user-centric metrics like CTR and user satisfaction.
b. **Document Retrieval:** Document retrieval is more concentrated on recall and precision. Assessing the performance of the system in recovering relevant documents from a highly massive collection is thus crucial.
c. **Multimedia Retrieval:** Multimedia retrieval, image, and video searching are measured via more specific metrics-such metrics include Mean Average Precision for ranked retrieval as well as Precision at k to determine the relevance of top-k retrieved media.
d. **Question Answering Systems:** In systems that produce direct answers, in the sense of a chatbot or a question-and-answer system, performance is assessed with regard to metrics such as exact match, recall, and precision for answers.

8.5.6 Evaluation Framework and Datasets

Evaluation in Information Retrieval links the questions and frameworks that can enable uniformity in evaluating system performance.

a. **TREC (Text Retrieval Conference):** TREC is a universally acknowledged framework and dataset of IR system evaluation. It consists of a collective set of queries with relevance judgments for the accessibility of success in tests to check efficiency in retrieval.
b. **Cranfield Dataset:** The Cranfield dataset is older. It provides queries and relevant documents for evaluating document retrieval systems and often finds use in traditional IR assessment.

c. **Judgment-Based Evaluation:** In judgment-based evaluation, human assessors judge the relevance of documents to the given query. This is quite labor-intensive but provides the least unreliable relevance measure.
d. **User-Centered Evaluation Datasets:** Datasets arising from real user interactions, for example, search logs, can be employed to evaluate the performance of their systems in much more dynamic and practical contexts.

8.5.7 Cost-Based Evaluation

Cases exist where it is crucial to assess retrieval in terms of costs involved.

a. **Cost of Computation:** The term refers to the resources such as processor time, memory, and CPU used by the retrieval system while executing and delivering results.
b. **Cost of Error:** The cost of error (which could be either false positives or false negatives) quantifies the impact of erroneously retrieved results on the user or system.

Hence, Evaluating Information Retrieval (IR) systems takes a plethora of forms ranging from the technical metrics-all measured with precision, recall, MAP, NDCG-to efficiency measures, such as response time, latency, scalability, and last, but by no means least, the user-centered metrics-satisfaction, task success, and CTR. With these evaluation models, developers and researchers may measure, fine-tune, and optimize the performance of IR systems as well as the user experience. Assessment forms the kernel of how well an IR system meets its intended target-or goal-be it conventional document retrieval, web search, or specialized multimedia or question-answering tasks.

8.6 Data Processing Technologies

8.6.1 Batch Processing

Introduction to Batch Processing

Batch processing is a mode of data processing in which large volumes of data are collected, processed, and stored at scheduled time intervals rather than in real-time. Data warehousing, ETL processes, and analytics applications traditionally follow this route of batch processing. This approach is suited especially when processing large amounts of data not needed to be addressed immediately. Unlike real-time or stream processing, batch processing focuses on efficiency by executing tasks collectively, typically during off-hours to optimize the utilization of computing resources.

Batch processing is the conventional yet very effective method used in many finance, healthcare, retail, and telecommunications sectors. This approach enables the organization to execute complex computations at a large scale, derive reports, and conduct data transformations. The batch processing model ensures the effective and cost-efficient utilization of resources, which renders it as the best choice for those applications that do not require instantaneous results.

8.6.1.1 MapReduce

MapReduce is a programming model developed by Google for processing and generating very large datasets in a parallel and distributed manner. MapReduce is structured around two main steps:

1. Map

Phase: The input data is divided into smaller chunks to be processed independently; these smaller chunks are converted into key-value pairs. With these key-value pairs, the data is easily distributed across multiple nodes for running in parallel.

2. Reduce

Phase: The key-value pairs generated on the map phase are grouped up and aggregated to produce the final output. This phase usually includes sorting, filtering, and summarizing the data.

MapReduce plays a fundamental role in the Hadoop ecosystem to process large-scale data efficiently and allows enterprises to manage petabytes of data. MapReduce is known for its fault-tolerant and scalable parallel processing, making it ideal for applications like log analysis, recommendation systems, and search indexing.

8.6.1.2 Data Parallelism

This refers to distributing the data across multiple processors or nodes to perform the computations in parallel. This technique improves efficiency and scalability in big data processing. Non-overlapping computations are able to run in parallel, while execution of other processing continues without waiting for each to finish. The time of execution is reduced, and performance gets boosted.

The key techniques are employed within data parallelism as follows:

- **Partitioning:** partitioning large datasets into smaller pieces and sending those parallel for
- **Load Balancing:** ensures that the work is evenly distributed between nodes to avoid
- **Replication:** This usually involves copying data across multiple nodes to improve fault tolerance and ensure faster processing speed.

Data parallelism is critical for modern big data frameworks such as Apache Spark and Apache Hadoop, allowing efficient processing and analytics of big data.

8.6.2 Stream Processing

Stream processing is the form of data processing which deals with real-time, instantaneous processing, whereby applications can continuously process data streams and take immediate action. Unlike batch processing, which processes batches of scheduled time, stream processing identifies the arrival of data, being capable of rendering insights and responses in virtually real-time.

Stream processing plays a role in applications such as:

- **Fraud Detection:** Real-time detection of fraudulent transactions on banking systems.
- **Real-time Analytics:** Monitoring and analyzing user activity on websites and applications.
- **Monitoring systems:** Processing data from IoT files, log files, and telemetry systems to yield anomaly detection.

Through the use of frameworks like Apache Kafka, Apache Flink, and Apache Storm, businesses can construct resilient stream processing pipelines to deal with high-velocity data handling.

8.6.2.1 Apache Kafka

An event-streaming platform for distributed systems that allows for high-throughput and real-time processing of data. It is largely seen as one of the most prolific message brokers that may also cater to event-driven applications and log aggregation.

Some key characteristics of Apache Kafka:

Scalability: Kafka can handle millions of messages per second across multiple nodes.

Fault Tolerance: Data is replicated across multiple brokers to prevent data loss.

High Throughput: Optimized for handling large volumes of real-time data efficiently.

Publish-Subscribe Model: Kafka follows a publish-subscribe architecture, allowing multiple consumers to process the same data stream.

Kafka finds application in the financial services sector, retail, and in real-time analytics, all outrunning one another with reliable and scalable data streaming.

8.6.2.2 Apache Flink

Apache Flink is a processing framework that accepts streaming input for performing real-time analytics and complex event processing. Unlike batch-oriented frameworks, Flink provides a mechanism for continuous data processing with minimal latency.

Key Features of Apache Flink:

- **Exactly-Once Processing:** One of the key features is to achieve data consistency by ensuring that each event is processed exactly once.
- **Low-Latency Event Handling:** Provides near-real-time response, with insights coming within sub-second after the event happens.
- **Distributed and Scalable:** Flink allows users to run applications on different clusters and offers support for large-scale data applications.
- **Support for Stateful Processing:** State can be maintained across multiple events, allowing for complex event pattern detection.

Flink is widely used for various applications such as predictive maintenance, fraud detection, and dynamic pricing models.

8.6.2.3 Event-Driven Architectures

An event-driven EDA is built for systems that need to listen and respond to events in real time. The loosely coupled, extremely scalable, and flexible application design arises from the conception of such architecture.

Core components of Event Driven Architectures:

Event Producers: These are responsible for generating events based on user actions, system events, or events put in place by IoT devices.

Event Brokers: Platforms such as Kafka or RabbitMQ that service the distribution of events to consumers.

Event Consumers: All applications or services that process and react to incoming events.

Some of the use cases of EDA include:

- **IoT:** Smart home systems reacting to sensor data.
- **Financial Systems:** Real-time stock trading platforms reacting to changes in the markets.
- **Microservices Architectures:** Decoupling the services from one another by communicating through event streams.

DA allows organizations to build resilient and scalable systems, which is the hallmark of modern cloud-native applications.

8.6.3 Batch Processing and Stream Processing

Batch processing and stream processing have different use cases in data engineering. Thus they may be compared with each other.

Both Batch Processing and Stream Processing in Modern Data Engineering, Batch Processing is more suitable for large data transformation and heavy reporting, while Stream Processing provides real-time insight critical to fast decision-making. Organizations should choose the path that is best fitted for the specific use case while laying out their plans by using advanced frameworks like Hadoop, Spark, Kafka, and Flink to make those data pipelines scalable and efficient.

Table 8: Difference between batch processing and stream processing

Feature	Batch Processing	Stream Processing
Data Handling	Process large amount of data at once	Process data in real-time as it arrives
Latency	High (delayed results)	Low (instantaneous results)
Use Cases	ETL, report generation, big data analytics	Real-time analytics, fraud detection, monitoring systems
Complexity	Easier to implement	More complex due to real-time constraints
Resource Usage	Optimized by running at scheduled times	Requires continuous computing resources

8.6.4 In Memory computing

Memory Computing (IMC) saves and processes information directly in RAM, rather than on disks. This elaborates on the speed of which information is accessed and computed. Applications become more efficient. This form of computing also enables real-time analytics. IMC works best with rapid transactions, large data processing and real-time analysis.

Fundamental Concepts of In-Memory Computing:

Memory and Processing Separation:

Conventional computing frameworks distinguish between memory and processing units which increases latencies with data transfers between the storage and processing components.

Such division can be a source of bottlenecks particularly in data rich applications. In-memory computing addresses these challenges efficiently by bringing data close to the CPU for faster access and processing.

Traditional vs. In-Memory Computing:

The computing world suffers from the limitation of using HDDs (or even SSDs) when it comes to accessing stored data, as both are mechanically bound systems whereas RAM is instant. Due to the mechanical nature of hard drives, both traditional and modern computing systems face the challenging hurdle of slow data management. Even SSDs don't come close to the speeds that RAM does.

With the type of special computing known as In-memory computing, volatile memory as a storage medium improves the speed of access, can reduce latencies considerably and even help improve response time. This shift is vital for prospective computing such as, high-frequency trading for services in finance, modern-day e-commerce and intensive data telecommunication.

Memory Wall:

The gap that increases between the CPU processing speed and the time needed to access memory is termed as "memory wall." As CPU processes information more rapidly, the speed at which memory is being accessed becomes a hurdle that ultimately stagnates the processing capacity of the system. In-memory computing tries to minimize the memory wall's effect by reducing the time that data spends traveling between the storage devices and the central processing unit.

Overcoming Data Transfer Bottleneck:

In memory computing eliminates the time lag associated with data transfers by utilizing fast RAM as a storage and processing device. It is helpful in minimizing data transfer lags due to the eradication of constant moving of information and instructions between the CPU and storage disk. This enables faster

calculation and real-time processing. In addition, better compression and parallel processing also improve the overall performance along with optimal memory management.

Fundamental Innovations:

Types of Memory: RAM, NVRAM, And Flash Memory NVRAM OR NON-VOLATILE **RANDOM ACCESS MEMORY:**

Retains stored information even when unplugged or powered down. Makes it feasible to merge traditional storage with advanced RAM technology. Ideal for high-speed applications that also require long-term data storage.

NVRAM: NVRAM functions like a faster and more efficient disk; with features of traditional storage, it conserves stored information even when unplugged or powered down. Makes it feasible to merge conventional storage with advanced RAM technology. Ideal for high-speed applications that also require long-term data storage.

FLASH MEMORY:

A type of non-volatile storage that can be electrically erased and reprogrammed. While faster than traditional hard drives it still does not match the speeds of RAM. Ideal for hybrid in-memory computing systems due to relatively inexpensive nature when compared to other forms.

The Functionality of In-Memory Computing Data Processing in RAM:

In-memory computing refers to data being processed using RAM as opposed to having it saved on disks. While disks allow storage, RAM enables faster information retrieval. RAM helps in fetching information at remarkable velocity which is essential for prompt processing with little to no time lapse in response rate.

The use of RAM in In-Memory Systems aids in the expeditious processing of hefty data and enables rapid computation along with decision making.

Additional Advantages of Storing Data in RAM Comparatively to Disk Storage: Storing information in RAM has numerous advantages when pitted against disk storage. Primarily, RAM simply offers faster speeds and retrieval.

8.6.4.1 REDIS

Redis (Remote Dictionary Server) is an open-source data structure store that stores data in memory and is being extensively used as a database, cache, and message broker. It is architected for high performance, low latency, and scalability and hence is one of the tools that every data engineer considers important in their workflow. Redis has the capability of supporting millions of operations every second, thus it is predominantly used for real-time applications like analytics, caching, and streaming.

Fig. 8.4: Logo of REDIS

Why is Redis Important in Data Engineering?

Data engineering is the act of planning, constructing, and operating data pipelines that convert raw data into valuable insights. With contemporary applications creating voluminous amounts of data, Redis is central to processing and managing the data quickly. Redis is especially useful in the following situations:

i. **Quick Data Access:** Redis retains data in memory, thereby supporting sub-millisecond response times.
ii. **Scalability:** Redis features clustering, sharding, and replication to address large-scale distributed workloads.
iii. **Data Persistence:** It provides snapshotting (RDB) and logging (AOF) for durability.
iv. **Real-time Processing:** Redis is heavily employed in real-time analytics, event-driven architectures, and streaming applications.
v. **Effective Data Storage:** Equipped with sophisticated data structures such as sets, sorted sets, hashes, hyperloglogs, and time series, Redis is able to optimize numerous data engineering operations.

How Redis Fits into Data Engineering Pipelines?

A common data engineering pipeline has several steps, such as data ingestion, transformation, storage, and retrieval. Redis improves all these steps with quick access to data, caching of middle results, and support for real-time event processing.

For instance, in a real-time recommendation system, Redis can do the following**:**

i. Ingest user behaviour data (e.g., clicks, views, purchases) using Redis Streams.
ii. Store and process data in Redis Sorted Sets to rank the most frequently viewed items.
iii. Cache the outcomes to deliver recommendations in real time to users.

Key Features That Make Redis Ideal for Data Engineering:

i. **In-Memory Storage:** Stores all data within RAM, which is much faster than disk-based databases.
ii. **Advanced Data Structures:** Supports lists, sets, sorted sets, hashes, bitmaps, hyperloglogs, and streams for optimized data modeling.
iii. **Horizontal Scalability:** Redis Clustering enables distributing data across multiple nodes.
iv. **Data Persistence:** Provides both snapshot-based (RDB) and log-based (AOF) persistence for fault tolerance.
v. **Pub/Sub and Streams:** Enabling event-driven real-time processing.
vi. **Low Latency:** Supports millions of operations per second with low latency.
vii. **Flexible Deployment:** Supports on-premises, cloud (AWS, Azure, Google Cloud), or hybrid deployments.

Redis Architecture and Components:

Redis uses a client-server architecture (as shown in fig.8.5), where several clients communicate with a single Redis server to process requests. Redis is built for performance, scalability, and fault tolerance, and hence it is widely used in data engineering, caching, and real-time processing. In the following, we discuss the main components of Redis architecture in detail.

1. Single-Node Redis Architecture:

A single-node Redis configuration is a Redis server, one or more clients, and an optional persistence mechanism. It is the most basic way of deploying Redis and is often utilized for caching and small applications.

How It Works:

i. The client issues a request (e.g., SET key value) to the Redis server.
ii. The server executes the request in memory, modifies its key-value store, and responds.
iii. If persistence is on, Redis saves data to disk periodically.

Pros:

i. **Quick and easy:** Because all operations are done in memory, response times are in the range of microseconds.
ii. **Low setup:** Perfect for small applications and development environments.
iii. **Low maintenance:** No multiple nodes to manage.

Cons:

i. **Not highly available:** If the Redis server crashes, all data is lost unless persistence is on.
ii. **Not scalable:** One instance can support finite traffic and data.

2. Master-Slave Replication:

Redis has asynchronous replication, where the master node processes write operations and forward changes to one or more slave nodes. This configuration enhances performance and availability by spreading read requests.

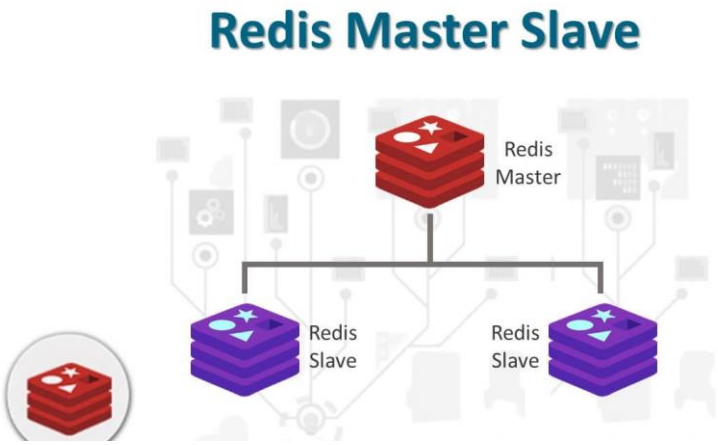


Fig. 8.5: Architecture of REDIS

How It Works:

i. The master node processes all write requests (e.g., SET user:1 "John").
ii. Slave nodes mirror data from the masters in real time.
iii. Clients can read from slave nodes, which decreases the load on the master.
iv. If the master becomes unavailable, the slave can be upgraded to master with Redis Sentinel.

Pros:

i. **High availability:** In case the master goes down, a slave can become the new master.
ii. **Load balancing:** Read queries can be routed across multiple slaves.

iii. **Data redundancy:** Minimizes data loss by keeping multiple copies.

Cons:

i. **Eventual consistency:** It may have some delay in synchronizing master and slave.

ii. **Slaves are read-only:** They don't support write requests, which may be a problem in some cases.

3. Redis Cluster (Sharding for Scalability):

Redis Cluster enables data to be sharded (partitioned) across multiple nodes, enhancing scalability and fault tolerance. It is optimized for large-scale applications with high throughput needs as shown as fig.8.6)

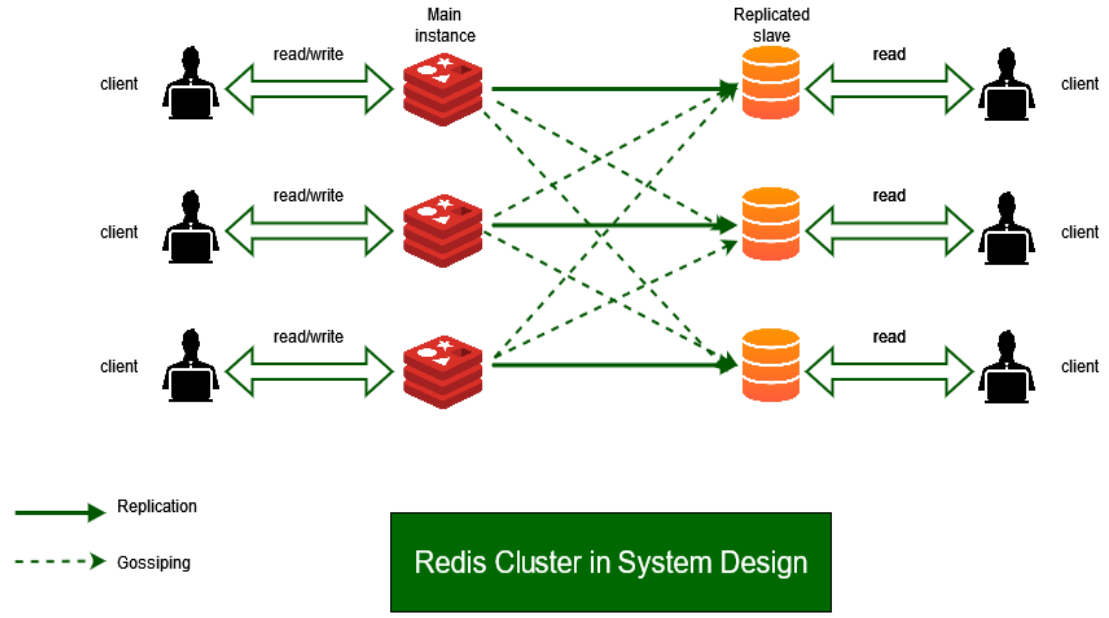


Fig. 8.6: REDIS cluster in system Design

How It Works:

i. Data is split into shards based on a hashing algorithm.

ii. Each Redis node holds a subset of the overall dataset.

iii. Nodes talk to one another and rebalance data automatically when new nodes are added.

iv. If a node fails, another node in the cluster assumes its slot.

Pros:

i. **Horizontal scalability:** Additional nodes can be added as data increases.

ii. **High availability:** In the event of a node failure, others compensate.

iii. **Automatic failover:** Redis Cluster automatically promotes replicas.

Cons:

i. **Complex setup:** Needs proper cluster setup and monitoring.

ii. **Not entirely ACID-compliant:** Multi-shard transactions are not natively supported.

4. Persistence Mechanisms (RDB & AOF):

As Redis is an in-memory database, it has two mechanisms of persistence to store the data on disk permanently as shown in fig.8.7.

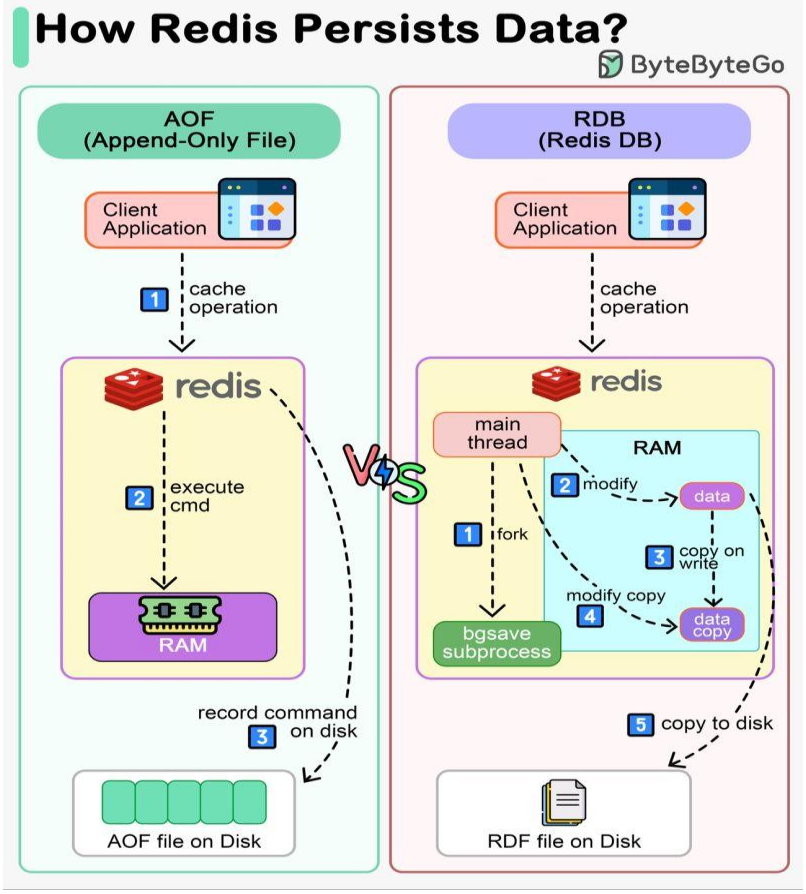


Fig. 8.7: Persistence Mechanism of Redis

4.1 RDB (Redis Database):

i. Snapshot-Based Persistence
ii. Redis takes point-in-time snapshots of the dataset at regular time intervals.
iii. These snapshots are saved in the form of a binary file (dump.rdb) on disk.

Pros:

i. **Space-efficient storage:** Consumes less disk space.
ii. **Speedy restart:** Helpful in case of fast recovery in case of failure.

Cons:

Risk of data loss: Changes are lost if Redis crashes before the next snapshot.

4.2. AOF (Append-Only File):

i. Log-Based Persistence
ii. Redis appends all write operations in a different file.
iii. When Redis is restarted, it replays the log in order to bring back data.

Pros:

i. **More durable:** Reduces data loss risk.
ii. **Fine-grained recovery:** Able to bring back data to the last logged operation.

Cons:

i. **Larger file size:** Takes up more disk space compared to RDB.
ii. **Slower restart time:** Redis must replay all operations from the log.

4.3. Hybrid Approach:

Redis supports using both RDB and AOF for a trade-off between performance and durability.

5. Redis Eviction Policies (Handling Memory Constraints):

Since Redis runs in RAM, it must have an effective mechanism to deal with memory usage when it is at its limit. Redis has several eviction policies to eliminate old data to free up space for new entries.

Eviction Policies available:

i. Noeviction: Fails with an error when the memory is full (default behaviour).
ii. allkeys-lru: Evicts the least recently used keys.
iii. volatile-lru: Evicts the least recently used keys among keys with an expiry.
iv. allkeys-lfu: Deletes the least recently used keys.
v. volatile-lfu: Deletes the least recently used keys from keys with expiration.
vi. volatile-random: Deletes randomly selected keys from keys with expiration.
vii. allkeys-random: Deletes randomly selected keys from all keys.

6. Pub/Sub and Redis Streams (Event-Driven Architecture):

Redis has real-time messaging and event processing with Pub/Sub and Streams as elaborated as fig.8.8.

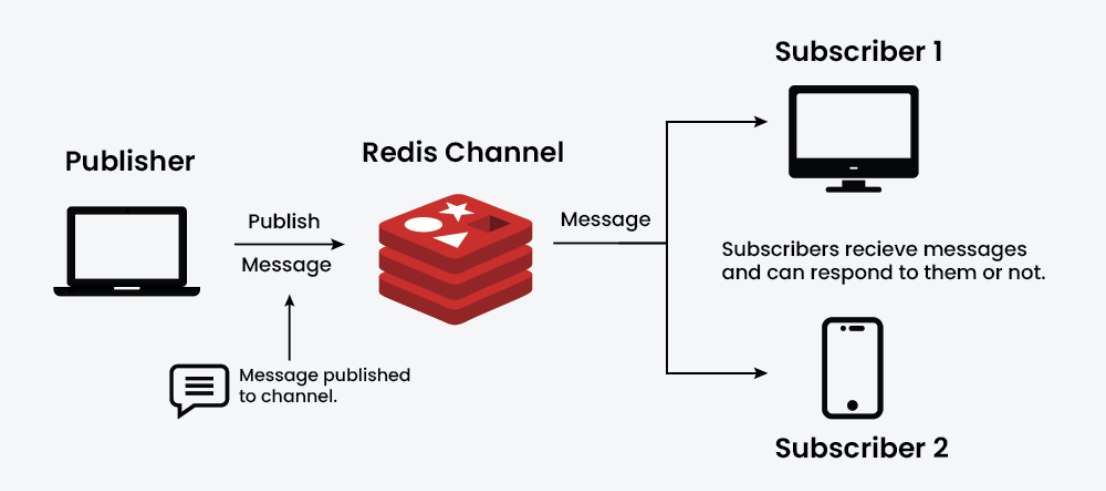


Fig. 8.8: Real-time event processing of Redis

6.1. Pub/Sub (Publish-Subscribe Pattern):

i. Publishers publish messages to a channel.
ii. Subscribers who are listening to the channel receive messages immediately.
iii. Suitable for event-driven applications, chat services, and notifications.

6.2. Redis Streams (Message Queue with Logs):

i. Redis Streams offers a log-based message queue for processing real-time data ingestion.
ii. Allows consumer groups so that multiple services can efficiently process messages.
iii. Utilized IoT event processing, log aggregation, and real-time analytics.

7. Redis Sentinel (High Availability and Failover):

Redis Sentinel is a failover and monitoring system used to provide high availability in a Redis installation as shown in fig.8.9.

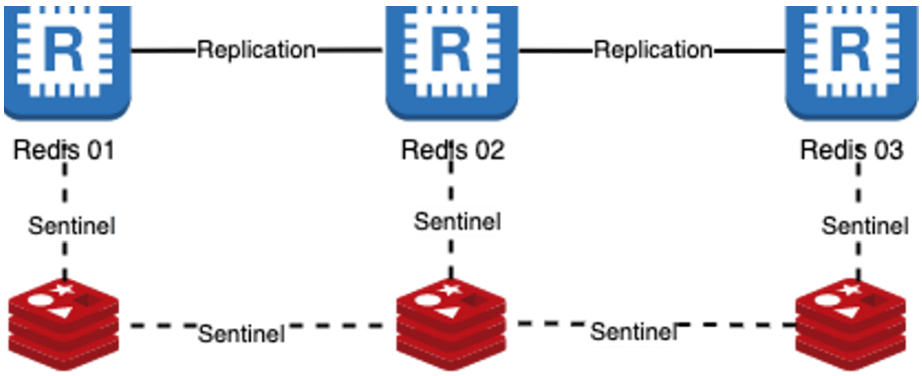


Fig. 8.9: Redis Sentinel

Redis Sentinel Features:

i. **Auto Failover:** If the master node is down, Sentinel will elect a slave as master.
ii. **Monitoring:** Continuously monitors the well-being of the Redis nodes.
iii. **Notification:** Notifies administrators of failures.

Use Case:

It is used in distributed caching to avoid downtime when Redis is down.

How Redis is integrated into Data Engineering Pipelines:

Redis plays a key function in data engineering pipelines through the offering of high-speed caching, real-time processing of data, message queuing, session storage, and management of temporary data. It boosts the efficiency of data pipelines by lowering latency, enhancing fault tolerance, and promoting seamless integration with other technologies such as Apache Kafka, Spark, and Flink.

In this section, we'll discuss how Redis gets integrated into various phases of a data engineering pipeline and how it is beneficial.

1. Caching for Faster Data Access:

One of the most common applications of Redis in data engineering is caching. Caching lightens the load on master databases and accelerates query performance, making Redis a perfect fit for high-traffic applications, API responses, and database query acceleration.

How It Works:

i. Highly accessed data (e.g., API responses, database query results) is cached in Redis.
ii. When a request is initiated, the system checks Redis first for the data.
iii. If the data is available (cache hit), it is immediately returned, lessening the burden of database queries.
iv. If the data is unavailable (cache miss), it is loaded from the database and is kept in Redis for retrieval in the future.

Use Cases:

i. Web applications: Cache user session data, authentication tokens, and page content.
ii. Microservices: Cache often utilized configurations and metadata.
iii. Machine learning: Cache model predictions and intermediate computations for fast access.

Advantages:

i. **Lowers database load:** Less database queries, better performance.
ii. **Faster response times:** In-memory allows sub-millisecond access.

iii. **Scalability:** Efficient handling of high-traffic workloads.

2. Message Queuing with Redis Pub/Sub:

Redis Publish-Subscribe (Pub/Sub) is a feature that is utilized for real-time messaging and asynchronous data processing within data pipelines.

How It Works:

i. Producers send messages to a Redis channel.
ii. Subscribers listen to the channel and get messages in real time.
iii. Can be utilized to stream events, process logs, or coordinate microservices.

Use Cases:

i. **Event-driven architectures:** Real-time alerts, notifications, and activity tracking.
ii. **Data pipelines:** Streaming real-time data or logs into processing engines such as Flink or Apache Spark.
iii. **IoT applications:** Processing and gathering sensor data in real time.

Advantages:

i. **Low-latency messaging:** Supports millions of messages per second.
ii. **Lightweight and scalable**: Can be used with distributed systems.
iii. **Seamless integration:** Integrates easily with data processing tools such as Kafka and Flink.

3. Data Streaming with Redis Streams:

Redis Streams offers a log-based, event-driven data structure, which makes it a great option for real-time data processing and ingestion. It is a high-performance message queue that guarantees ordered and reliable message delivery.

How It Works:

i. Producers insert messages into a Redis stream.
ii. Consumers (data processing applications) consume messages in batches or in real-time.
iii. Consumer groups enable multiple applications to process the stream concurrently.

Use Cases:

i. **Log aggregation:** Centralized logging for monitoring and debugging.
ii. **Event sourcing:** Keeping and replaying business events.
iii. **Real-time analytics:** Handling clickstream data, stock quotes, or sensor readings.

Advantages:

i. **Highly scalable:** Can handle huge amounts of data in real-time.
ii. **Fault-tolerant:** Redis guarantees message persistence and restoration.
iii. **Supports parallel processing:** Multiple consumers can process simultaneously.

4. Session Storage for Real-Time Applications:

Redis is also commonly applied to session management, where it holds session user information temporarily for web applications, authentication mechanisms, and API sessions.

How It Works:

i. When a user logs in, their session information is cached in Redis.
ii. A unique key with an expiration date is assigned to every session.
iii. When the session times out, Redis deletes it automatically, freeing memory efficiently.

Use Cases:

i. **E-commerce applications:** Maintain shopping cart data in real-time.
ii. **Authentication systems:** Keep user tokens for OAuth-based logins.
iii. **Chat and gaming applications:** Store active user connections efficiently.

Advantages:

i. **Low-latency access:** Slows down login authentication times.
ii. **Automatic expiration:** Remains memory-leak-free by removing old sessions.
iii. **Scalable:** Supports millions of concurrent users.

5. Temporary Data Storage for ETL Pipelines:

In ETL (Extract, Transform, Load) workflows, Redis is employed for staging data, holding intermediate calculations, and temporary computation prior to loading data into a data warehouse.

How It Works:

i. Raw data is temporarily cached in Redis from various sources.
ii. Data is transformed (cleansed, enriched, or aggregated) in-memory.
iii. Processed data is written to a permanent data store such as Amazon Redshift or BigQuery.

Use Cases:

i. Pre-processing data before loading into databases.
ii. Managing transient data in streaming analytics pipelines.
iii. Caching processed data for quicker downstream processing.

Advantages:

i. **Increased speed of processing:** In-memory processing accelerates ETL processes.
ii. **Minimizes load on primary databases**: Avoids unnecessary writes.
iii. **Optimal memory management:** Data can automatically expire upon processing.

6. Real-Time Analytics & Monitoring:

Redis is widely utilized in real-time analytics use cases where real-time insights are needed.

How It Works:

i. Raw data is consumed into Redis Streams or Pub/Sub.
ii. Processing engines (e.g., Spark, Flink, or ClickHouse) process the data in real time.
iii. Results are cached in Redis for rapid dashboard retrieval.

Use Cases:

i. **Fraud detection:** Detect anomalies in banking transactions.
ii. **Ad-tech platforms:** Monitor impressions, clicks, and conversions in real-time.
iii. **System monitoring:** Merge server logs and identify failures.

Advantages:

i. **Sub-millisecond query latency:** Allows real-time data retrieval.
ii. **Supports high data rate:** Ideal for Big Data use cases.
iii. **Supports multiple frameworks:** Integrates with Apache Spark, Kafka, etc.

7. Redis as a Secondary Index for Databases:

Databases such as MySQL, PostgreSQL, and MongoDB may utilize Redis as an additional index to enhance query performance.

How It Works:

i. Redis holds the indexes that are used often in queries and lookup tables.
ii. Apps query Redis first for immediate results.
iii. If the data is not located in Redis, the query gets sent to the main database.

Use Cases:

i. Rapid lookups on big data sets.
ii. Metadata indexing for search engines.
iii. Optimizing database queries for high-performance applications.

Benefits:

i. **Enhances query performance:** Minimizes database load.
ii. **Enables full-text search:** Employed for speedy keyword-based lookup.
iii. **Compatible with several databases:** Used with SQL as well as NoSQL databases.

8.6.4.2 Apache Ignite

Apache Ignite is an open-source computing platform, distributed database, and cache designed for high-performance and scalable data processing. It is ideal for real-time applications, data integration, and analytics due to its in-memory and persistent storage capabilities. Apache Ignite's architecture is based on a distributed cluster model, ensuring high fault tolerance, scalability, and high availability.

Role of Apache Ignite in Data Engineering:

Since it enables fast data processing, real-time analysis, and seamless integration with other data platforms, Apache Ignity is critical to data engineering. It is high- speed data layer that enables massive – scale computations, increases data availability, and accelerates query performance

1. **Streamlining ETL Processes**:

   As an intermediary in-memory data store that reduces the latency of traditional batch processing, Apache Ignite facilitates the optimization of Extract, Transform, Load (ETL) processes. Ignite supports rapid data ingestion and transformation without the need for traditional databases or disk-based systems such as Hadoop.

   - Streaming ETL: Ignite processes data streams in real-time before storing them in a data warehouse.
   - Batch ETL Optimization: Ignite accelerates batch loads by storing results of intermediate queries in memory if it is paired with Spark.

2. **Handling Data in Real Time:**

   Handling streaming data from producers such as Kafka, Flink, or Apache Pulsar is routine work in today's data engineering. An in-memory distributed compute grid provided by Ignite supports event-driven data transformations and real-time analysis.

   - Low-Latency Computation: Ignite provides response times of under a millisecond due to its in-memory data storage
   - Discovery of patterns, correlations, and outliers in data streams is supported by complex event processing, or CEP.

3. **Caching in Distribution for High-Performance Queries**:
   Query Optimization is one of the largest issues in data engineering. With Igniteis distributed caching approach, it is made possible to run queries effectively by storing data that is frequently accessed in memory. This significantly enhances performance for high-throughput applications.
   - SQL Query Acceleration: By supporting ANSI SQL-99, Ignite can act as a very fast query layer over PostgreSQL or MySQL databases.
   - Partitioning and Indexing: Advanced Indexing Methods ensure efficient query execution over nodes.
4. **Data Lake Integration:**
   Apache Ignite supports interactive querying of big data using Amazon S3, HDFS, and other data lakes integration.
   - Hybrid Storage Model: Supports efficient querying of disk-resident as well as in-memory data.
   - Transaction and analytical workloads are consolidated into one system by homogeneous data access
5. **Distributed Computing with High Performance**
   An in-Apache Ignite computing grid allows for simultaneous execution of computationally intensive tasks.
   - Map Reduce – Like Execution: Distributed Execution of tasks to minimize computing time.
   - Workloads for AI and machine learning: Supports in-memory data sets for distributed ML training.
6. **Scalability and Fault Tolerance:**

Scalability and fault tolerance are a must for data engineering pipelines. Ignite provides high availability and data durability by:

- Replication & Failover: For protection against data loss, data gets replicated across nodes automatically.
- Dynamic Rebalancing: Ignite dynamically rebalances data when nodes join or leave.

Apache Ignite Architecture: Apache Ignite architecture is designed to enable data storage with in-memory and persistent features alongside high-performance distributed computation as shown in fig.8.10. It consists of multiple components that collaborate to provide fault tolerance, high availability, and seamless scaling.

1. **Distributed Architecture via Clusters**: Apache Ignite Operates as a distributed cluster in which multiple nodes work together in processing and holding data. The nodes can be categorized into two broad categories:
   - Server Node: Data is maintained on server nodes, which further perform calculations and queries.
   - Client Nodes: Make requests to the cluster without maintaining data, thereby facilitating efficient processing of data for light application.

Since Ignite clusters support a peer-to-peer structure, there is no point of failure. Nodes can be dynamically added or removed without incurring any downtime, and all server nodes are

equal. During the addition or removal of nodes, automatic data rebalancing ensures that the data is evenly distributed.

Memory-Centric Storage Architecture:

With Apache Ignite's hybrid storage mechanism, data can be permanently stored on disk and in-memory. This provides an equilibrium among durability and velocity:

In-Memory Storage: RAM dramatically boosts read and write performance by putting frequently used data there.

- Persistent Storage: Upon node failure, data is saved to disk to ensure durability and recoverability.
- n an effort to better manage memory usage, Ignite utilizes a page store architecture instead of storing complete objects in memory.

2. **Data Partitioning and Replication**:

Apache Ignite employs a partitioning and replication scheme to distribute data across multiple nodes in order to enable scalability:

- Partitioned Mode: For horizontal scalability, data is split into partitions and spread across multiple nodes
- Replicated Mode: For high availability and fault tolerance, copies of data are maintained on multiple nodes
- Affinity collocation reduces network overhead to optimize query performance by ensuring that related data is stored together.

3. **SQL Query Engine:**

Using Apache Ignite distributed SQL engine that complies with ANSI SQL-99, users can run queries as they would on a regular relation database. Key features are:

- Full-Text Search & Indexing: Supports hash indexing and B+ trees for rapid query execution.
- Distributed Query Execution: For low latency, queries are executed concurrently by multiple nodes.
- Support for JDBC and ODBC: Enables integration with reporting tools, business intelligence, and external applications.

4. **Distributed Processing Using the Compute Grid:**

Compute-intensive tasks can be run in parallel on cluster nodes due to Apache Ignite's compute grid.

- MapReduce-like execution is based on dividing jobs into smaller ones and performing them in parallel on different nodes.
- Service grid:Even if there is a node failure, the service grid ensures background services persist.
- Collocated processing reduces data transfer overhead by bringing computation to data locations.

5. **Fault Tolerance and High Availability**:

High dependability and availability are ensured by Apache Ignite using:

- Automated task migrations to healthy nodes when a node fails.
- Replication and Backup: Multiple data copies ensure durability and redundancy of the data
- Persistent and recoverable committed transactions are achieved in case of failures due to Write-Ahead Logging (WAL) and Checkpointing.

7. Integration with External Systems: The following data environments can seamlessly be integrated into Apache Ignite:

- Big Data Systems: Use Apache Spark, Apache Hadoop, and Apache Kafka for handling data in real time.
- Big Data Systems: Use Apache Spark, Apache Hadoop, and Apache Kafka for handling data in real time.
- Options for cloud storage are Google Cloud Storage, AWS S3, and Azure Blob Storage.
- Relational databases: Oracle, MySQL, and PostgreSQL through the use of JDBC and ODBC drivers for Ignite.

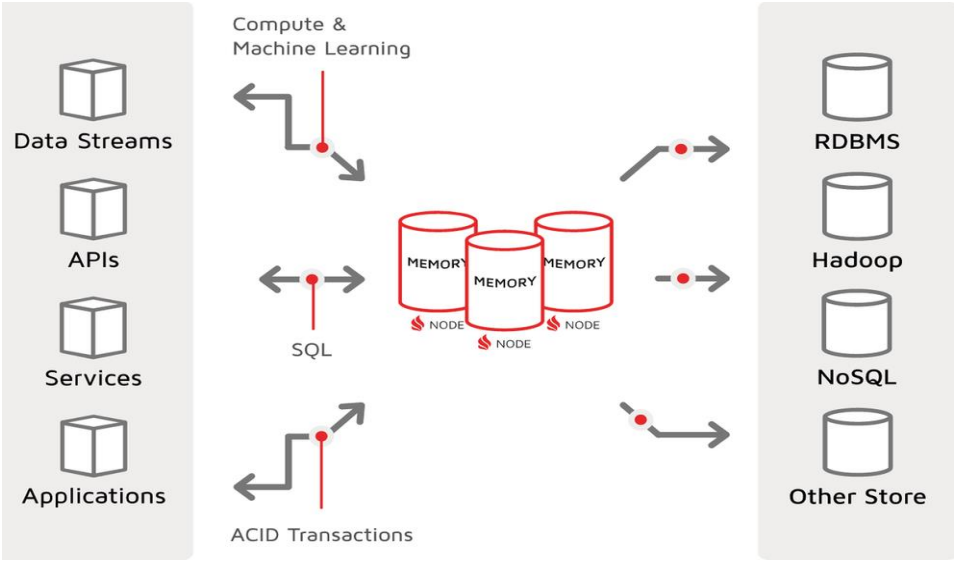


Fig. 8.10: Apache Ignite Architecture

Application of Apache Ignity:

Many different businesses employ Apache Ignite, especially those where distributed computing, real-time analytics, and fast data access are essential.

Distributed Caching

Distributed caching is a high-performance storage layer placed in front of relational databases such as Oracle, MySQL, and PostgreSQL, with a substantial impact on application performance. It minimizes database query latency by storing frequently accessed (hot) data in memory across multiple nodes, thereby improving response times for web applications and APIs.

This methodology not only reduces the load on the master database but also enables scalability, a key requirement in high-traffic environments.

As an example, online stores use distributed caching mechanisms such as Apache Ignite to cache user sessions, product lists, and search results. This ensures faster page loads, a smoother shopping experience for clients, and reduces the burden on the backend database.

Real-Time Analytics and Big Data Processing

Apache Ignite excels in real-time data analysis and large-scale data processing through its use of distributed computing methods. It supports the fast processing of massive data volumes, making it ideal for high-speed data computation applications.

Businesses that rely on real-time event processing—such as IoT, log analysis, and stream data processing—greatly benefit from Ignite's ability to handle continuous data streams efficiently.

One of the most impactful applications of this capability is in banking fraud detection, where Ignite processes millions of transactions per second to identify suspicious patterns and potential fraudulent behavior. By processing vast amounts of data in real time, Apache Ignite helps financial institutions reduce risks and enhance security.

Financial Services & Banking

The financial industry relies heavily on low-latency, high-speed access to data for real-time decision-making, and Apache Ignite plays a crucial role in this domain. It enables real-time risk assessment, trade processing, and credit scoring, allowing financial institutions to respond instantly to market developments.

Its high-throughput transaction handling capability makes Ignite an ideal choice for stock exchange trading platforms, where real-time price movements influence trade decisions. By leveraging Ignite, traders can process transactions in milliseconds, enabling them to capitalize on market opportunities without compromising system stability and performance.

IoT and Telecom Industry

The Telecommunications and IoT sectors generate a tremendous volume of real-time sensor data, smart device data, and network element data. Apache Ignite enables the efficient processing and analysis of this data, empowering businesses to streamline network traffic, perform predictive maintenance, and enhance operational performance.

In smart electricity grids, for example, smart meters continuously send consumption data to Ignite. Ignite then dynamically calculates electricity charges based on demand. This real-time analysis ensures a fair distribution of energy, optimizes costs, and helps avoid system overloading.

Telecommunication companies (Telcos) also leverage Ignite to process call detail records and monitor network efficiency, ensuring uninterrupted calls and the effective handling of traffic.

AI/ML Model Caching and Computation

Fast model access for artificial intelligence and machine learning functions is necessary to facilitate real-time prediction and user-specific personalization.

Apache Ignite enhances AI/ML tasks by caching machine learning models into RAM, thereby eliminating latency and enabling faster inference. Additionally, its distributed computing capabilities allow for the efficient training and updating of ML models across multiple nodes.

This is particularly beneficial in recommendation systems used by online platforms like Netflix and Amazon, where pre-computed AI-based recommendations are stored in Ignite for rapid retrieval.

By leveraging this approach, such platforms can deliver personalized content suggestions in real time, significantly boosting user experience and satisfaction.

E-commerce and Content Delivery

E-commerce sites and content delivery networks (CDNs) rely on fast data access to deliver personalized user experiences, enable real-time order processing, and support efficient inventory management. Apache Ignite optimizes these functions by caching frequently accessed data, reducing dependency on backend databases, and ensuring seamless performance—even during high-traffic events like Black Friday and Cyber Monday.

By effectively handling massive traffic and transaction volume spikes, Ignite ensures smooth navigation, instant product recommendations, and fast checkout experiences. This capability is critical to maintaining customer satisfaction and driving maximum revenue during peak shopping periods.

Advantage of Apache Ignity:

1. In-Memory High-Speed Computing

Apache Ignite is engineered to store data directly in RAM rather than relying on traditional disk-based storage, significantly enhancing data access speeds. By leveraging in-memory computing, Ignite eliminates latency and maximizes throughput, making it exceptionally effective for real-time computations and analytics. Unlike conventional databases that depend on slow disk I/O operations, Ignite provides direct memory access, resulting in faster query responses and transaction processing. It supports a wide range of functionalities—including in-memory key-value storage, SQL querying, and distributed computing—enabling applications to handle large volumes of data at high speeds without performance bottlenecks.

2. Persistence & Durability (Optional Disk Storage)

Unlike many other in-memory databases such as Memcached, Apache Ignite offers an optional persistent storage layer. This unique feature allows Ignite to function either as a volatile in-memory cache for ultra-fast data access or as a fully persistent database, ensuring data durability even after a system reboot. Users can configure Ignite according to their specific needs—either prioritizing speed through in-memory mode or ensuring data persistence through disk storage. This hybrid capability merges the best of both worlds: the high performance of in-memory computing with the reliability of disk-based storage, making Ignite a robust solution for applications that demand both speed and durability.

3. Full ANSI SQL Compliance

Apache Ignite provides ANSI SQL support, enabling users to query data using standard SQL syntax—just like in traditional relational databases. This eliminates the need for developers to learn a new query language, thereby simplifying the integration of Ignite into existing systems. The platform efficiently handles complex SQL operations, including aggregations, joins, and indexing over distributed datasets, ensuring high-performance data access and manipulation. By delivering SQL-based querying at in-memory speeds, Ignite effectively bridges the gap between NoSQL scalability and relational database familiarity, offering developers the best of both environments.

4. Horizontal Scalability & High Availability

Apache Ignite is architected for horizontal scalability, allowing new nodes to be added effortlessly to expand system capacity. It supports automatic data partitioning and replication across multiple nodes, ensuring high availability and fault tolerance. In the event of a node failure, data remains accessible from replicas, thereby eliminating single points of failure and enhancing overall system reliability. This distributed architecture makes Ignite an excellent choice for large-scale, mission-critical applications that require continuous uptime and the ability to scale dynamically in response to demand.

5. ACID Transactions

Other in-memory caching products like Redis or Memcached do not support ACID (Atomicity, Consistency, Isolation, Durability) transactions. Apache Ignite supports transactions fully. This ensures data consistency and reliability with multiple operations, making it ideal for

financial and banking applications where data integrity is key. With ACID compliance, Ignite guarantees that transactions are processed safely and reliably, even in distributed systems, avoiding problems like partial updates or data corruption. This makes it a good candidate for applications that need transactional guarantees combined with high-speed processing.

6. Multi-Language Support

Apache Ignite offers robust multi-language support, including Java, .NET, C++, Python, and Node.js, making it a versatile choice for developers operating in heterogeneous environments. This extensive language compatibility ensures that Ignite can be seamlessly integrated into a variety of technology stacks, allowing development teams to utilize their preferred programming frameworks while leveraging the high-performance benefits of in-memory computing. By supporting multiple languages, Ignite effectively caters to a wide range of applications—from enterprise systems to AI and machine learning workloads—enhancing its appeal across diverse industries.

7. Distributed Computing & Machine Learning Support

Apache Ignite goes beyond standard caching and storage by offering an in-memory distributed computing grid, which enables large-scale data processing and real-time analytics. It seamlessly integrates with Apache Spark, Hadoop, and Kafka, allowing organizations to process big data effectively. Ignite's computing power facilitates parallel processing of computationally intensive tasks across nodes, greatly speeding up computations. Secondly, Ignite features integrated machine learning (ML) and artificial intelligence (AI) capabilities so that data scientists can conduct model training, inference, and predictive analytics within the distributed environment. This makes Ignite a very suitable option for large-scale data-driven industries such as financial services, healthcare, and e-commerce.

Drawbacks of Apache Ignite

1. Uncontrolled Memory Consumption

One of the major disadvantages of Apache Ignite is that it has a high memory usage. As Ignite mostly caches data in RAM for quick retrieval, it needs to have high memory resources in order to run smoothly. For big deployments, this can translate into huge infrastructure expenses since large RAM sizes are costly to maintain. Companies must evaluate their hardware needs very carefully before using Ignite to ensure they have enough resources to manage their loads.

2. Difficult Configuration and Administration

Apache Ignite configuration and management is more involved in comparison to the more straightforward caching solutions of Redis or Memcached. Ignite's distributed character demands knowledge of clustering, replication, and query optimization to facilitate seamless operation. Installation and tuning Ignite to deliver optimal performance can prove demanding, particularly for teams with little experience in distributed systems. Furthermore, Ignite's feature richness can add overhead to projects with only basic caching needs, so it is not ideal for smaller applications.

3. Slower Writes in Comparison to NoSQL Databases

Although Apache Ignite is strong in read performance because of its in-memory design, write operations can be slower than with some NoSQL databases such as Redis. The main reason is that Ignite mandates ACID transactions and SQL-based processing, which incur extra overhead during

writing data. Applications that demand very high write throughput can be required to assess if Ignite aligns with their performance requirements or explore other alternatives for write-heavy workloads.

4. Overhead for Small Applications

Apache Ignite is optimized for commercial-grade applications which need high-performance computing and massive data handling. Ignite introduces too much overhead and complexity for trivial caching requirements in tiny applications. For such applications, simple caching like Redis or Memcached can be more pragmatic since they have easy configurations and less memory usage without sacrificing access speed.

5. Needs Cluster Management

Managing an Apache Ignite cluster involves handling multiple nodes, monitoring data consistency, partitioning, and fault tolerance. Unlike Redis, which offers relatively simple master-slave replication, Ignite requires administrators to actively manage cluster health, ensuring that data remains synchronized across nodes. This adds an operational burden, particularly for teams that lack experience with distributed databases. Maintaining an Ignite cluster demands continuous monitoring and tuning to prevent performance bottlenecks and system failures.

Apache Ignite vs Traditional Data Engineering Tools:

Table 9: Difference between apache ignite, Hadoop and spark

Feature	Apache Ignite	Hadoop	Spark
Storage Type	Hybrid(In- Memory + Persistent)	Data- Based	In- Memory
Query Performance	Sub - millisecond	High Latency	Moderated
Real-Time Processing	Yes	No	Yes
Fault Tolerance	Automatic Replication	HDFS Replication	Checkpointing
Integration	SQL, Kafka, Hadoop, Spark	Hive, Pig, Spark	Kafka, Ignite

For data engineers who want to optimize ETL procedures, increase query performance , and enable real – time processing. Apache Ignite is a powerful tool. Ignite is an ideal instrument to develop modern data engineering pipelines requiring speed, scalability, and reliability due to its distributed caching, compute grid, and in- memory SQL capabilities.

8.6.4.3 MEMCAHED

Memcached is a high-performance, distributed memory object caching system that is used to improve dynamic web application performance by minimizing database load. It simply saves often-retrieved data into memory so applications can get data fast without repeatedly querying the backend database.

Memcached has a key-value storage scheme and is thus suitable for storing results of database queries, API responses, session data, and other frequently accessed objects. It is used extensively in high-traffic web applications like Facebook, Twitter, and Wikipedia.

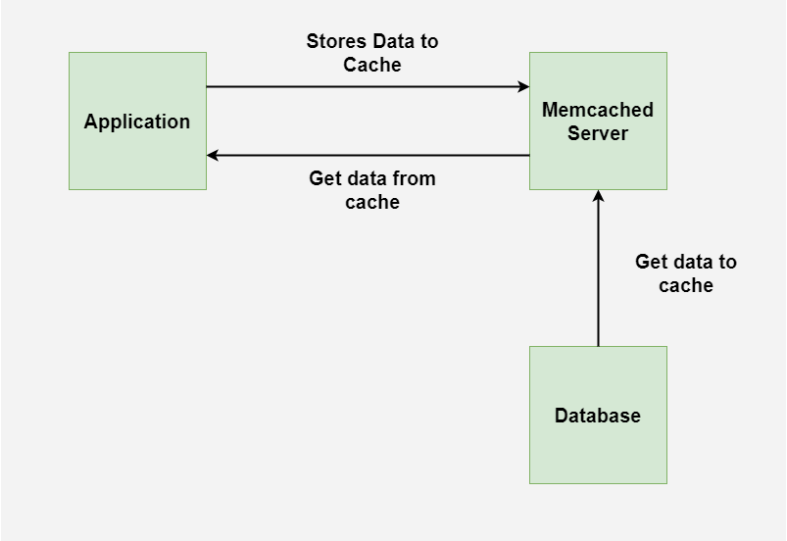


Fig. 8.11: Working of Memcached

Key Characteristics of Memcached:

i. **Lightning-fast:** Hosts data in RAM, with sub-millisecond response times.
ii. **Distributed:** Facilitates horizontal scaling through the addition of additional nodes.
iii. **Simple API:** Offers simple key-value operations.
iv. **Volatile storage:** The data is held in memory and is not saved.
v. **Effective memory handling:** Utilizes Least Recently Used (LRU) eviction for cache management.

Memcached Architecture and Components:

Memcached has a client-server architecture, where several clients communicate with distributed Memcached servers to store and efficiently fetch data. It is optimized for speed, scalability, and simplicity, making it a vital caching system for contemporary applications. The memcached architecture includes clients, servers, hashing algorithms, storage engines, and memory management units.

Here, we describe each component in detail:

1. Client:

Client refers to the program or system interacting with Memcached to store as well as get cached data. It issues SET, GET, and DELETE commands to the Memcached server.

Major Functions of the Client:

i. It stores data in Memcached using the SET command.
ii. Retrieves data with the GET command.
iii. Removes expired or unnecessary data with the DELETE command.
iv. Manages cache miss situations by fetching data from a database as required.

Memcached clients can be developed in any programming language, such as Python, Java, PHP, Node.js, and C++.

2. Server:

The Memcached server is used to store and handle cached data in RAM. It runs as a daemon process, so it will be operating in the background, processing requests from clients all the time.

Some of the main features of the Server are:

i. Stores data in a key-value form for instant lookups.
ii. Handles multiple requests at a time through multi-threading.
iii. Applies memory management schemes to use the data effectively.
iv. Provides high-speed data access with low latency.

A Memcached cluster is made up of several Memcached servers, enabling horizontal scaling and load distribution among various nodes.

3. Hashing Mechanism:

Memcached employs a hashing algorithm to decide which server will hold a specific chunk of data. Consistent hashing is usually employed to distribute data evenly across many servers, providing even load distribution.

How Hashing Works:

i. The client hashes the key using a hash function (e.g., MD5).
ii. The result of the hash function decides which Memcached server holds the data.
iii. When accessing data, the same hash function is used to find the appropriate server.

Advantages of Hashing in Memcached:

i. **Optimized load distribution:** Avoids loading one server excessively.
ii. **Guarantees data consistency:** Always reads data from the right node.
iii. **Enhances scalability:** Allows adding or removing servers dynamically.

4. Storage Engine:

Memcached does not rely on a conventional database system. Instead, it utilizes an in-memory storage engine that efficiently stores data.

Memory Allocation in Memcached:

i. Data is allocated in "slabs," which are fixed-size memory allocations.
ii. A slab has several pages, and each page has items (key-value pairs).
iii. The slab allocation system minimizes memory fragmentation and enhances performance.

How Storage Works in Memcached:

i. When data is stored by a client, Memcached allocates the data into an appropriate slab according to the size of data.
ii. If something is too large, it will be distributed into more than one slab.
iii. On full cache, the least recently used (LRU) are discarded to free memory.

Advantages of Slab-Based Storage:

i. **Improved memory allocation:** RAM is not wasted**.**
ii. **Short access times:** Access to the data is quicker.
iii. **Consistent performance:** Prevents unexpected slowdowns caused by memory fragmentation.

5. Memory Management and LRU Eviction:

Memcached has a Least Recently Used (LRU) eviction strategy that deletes old data when the memory is full.

How LRU Eviction Takes Place:

i. Every cached item has a last access timestamp.
ii. When memory is full, Memcached deletes the least recently accessed items.

iii. The evicted items are replaced by new data, ensuring that memory is utilized effectively.

Other Memory Management Features:

i. Expiration Time: Any cached element can be provided with a TTL (Time-To-Live) when it automatically gets removed after that time.
ii. Flush All Command: Clears every piece of data stored if required.

Advantages of LRU Eviction:

i. **Averts memory overflow:** Maintains continuous caching without crashing.
ii. **Improves utilization of resources**: Provides easily accessible data most often requested.
iii. **Auto-clean:** Manual cache maintenance is not needed.

6. Communication Protocols:

Memcached has a straightforward text-based protocol to talk between clients and servers and also has a binary protocol for more efficiency.

Typical Memcached Commands:

i. SET key value [expiration]: Stores a value.
ii. GET key: Retrieves a value.
iii. DELETE key: Deletes a value.
iv. FLUSH_ALL: Deletes all cached content.

Memcached runs over TCP/IP, so it can be utilized in distributed systems and cloud environments.

7. Distributed Architecture and Scalability:

Memcached is extremely scalable, and it is possible to have several servers cooperating in a distributed caching system.

How Memcached Scales:

i. **Horizontal Scaling:** New servers can be added to process more cache data.
ii. **Client-Side Load Balancing:** The client decides where to store and fetch data.
iii. **Consistent Hashing:** Guarantees uniform data distribution among many nodes.

Benefits of Distributed Memcached:

i. **No single point of failure:** When one server fails, others keep working.
ii. **Supports mass applications:** Supports millions of requests per second.
iii. **Boosts overall performance:** Lowers latency by distributing the load.

How Memcached Works:

Memcached is a distributed memory caching system with high performance that caches frequently accessed data in RAM to minimize database load and accelerate application performance. It uses a basic key-value storage model, where data is stored and retrieved based on unique keys.

Memcached operates in client-server mode, with clients requesting data from Memcached servers and storing and fetching data efficiently. The system adopts a non-blocking, event-driven mechanism to process multiple requests concurrently.

In the following, we outline the step-by-step process of how Memcached operates, including data storage, fetching, expiration, eviction, and distributed cache mechanisms.

1. Step-by-Step Process of Data Storage and Retrieval:

Step 1: Client Fetches Data from Memcached (GET Request)

i. The application checks Memcached first if it needs data and sends a GET request.

ii. If data is in Memcached (cache hit), it is delivered instantly.
iii. If the data is not in Memcached (cache miss), Memcached fetches it from the database.

Step 2: Cache Miss – Retrieving Data from Database

i. If data is not in Memcached, the client requests it from the database.
ii. The data that is retrieved is cached in Memcached to be used in the future.

Step 3: Saving Data to Memcached (SET Request)

i. The client initiates a SET request to save the data into Memcached.
ii. Memcached assigns a special key to the data and saves it in RAM.
iii. A TTL can be set for automatically deleting old data.

Step 4: Subsequent Requests Fetch Cached Data

i. On repeated requests, the information is retrieved from Memcached rather than asking the database.
ii. This minimizes response time and reduces database load.

Example of Data Storage & Retrieval:

1. First request (Cache Miss)

i. Client asks for user:1234 → Memcached does not have it.
ii. Database query is performed → Data is retrieved.
iii. Data is cached in Memcached (SET user:1234 {"name":"John"}).

2. Second request (Cache Hit)

i. The client asks for user:1234 → Memcached gives data directly (GET user:1234).
ii. No requirement to query the database.

2. Key Operations in Memcached:

Memcached supports several key operations for efficient caching:

1. Storing Data (SET Command):

Adds a key-value pair to Memcached.

Example:

bash

SET user:1234 3600 0 9

{"name":"John"}

(Stores data for 3600 seconds.)

2. Retrieving Data (GET Command):

Fetches data using a key.

Example:

bash

GET user:1234

(Returns {"name":"John"} if present in cache.)

3. Deleting Data (DELETE Command):

Removes a key-value pair from Memcached.

Example:

bash

DELETE user:1234

4. Flushing All Data (FLUSH_ALL Command):

Clears all cached data.

Example:

bash

FLUSH_ALL

3. Cache Expiration and Eviction in Memcached:

A. Expiration Time (TTL - Time-To-Live):

When data is stored in Memcached, it can have an expiration time (TTL).

After the TTL expires, the data is automatically removed from cache.

Example:

bash

SET user:1234 1800 0 9

{"name":"John"}

(Stores data for 1800 seconds.)

B. Cache Eviction Policy (LRU - Least Recently Used):

i. If Memcached runs out of memory, it removes the least recently used (LRU) items.
ii. This ensures that frequently used data stays in cache.
iii. LRU eviction happens automatically to free up space for new data.

How LRU Works:

i. Memcached tracks how often each key-value pair is accessed.
ii. When memory is full, the least recently accessed items are removed.
iii. Newly added data replaces the evicted data.

4. Distributed Caching in Memcached:

Memcached scales horizontally by distributing data across multiple servers. This allows applications to cache large volumes of data efficiently.

A. How Data is Distributed?

i. Memcached uses hashing (consistent hashing) to distribute keys across multiple servers.
ii. The client determines which server stores a particular key.

B. Load Balancing in Memcached

i. Clients distribute requests across multiple Memcached servers.
ii. This prevents a single bottleneck, improving performance.
iii. If a server fails, the remaining servers continue handling requests.

Example of Distributed Memcached:

i. Suppose we have three Memcached servers:
   - Server 1: Stores keys starting with A–H.
   - Server 2: Stores keys starting with I–P.
   - Server 3: Stores keys starting with Q–Z.
ii. When a client stores user:1234, the hashing algorithm determines which server should store it.

Difference Between Apache ignite , Redis and Memcached:

Table 10: Difference between apache ignite, redis and memcached

Features	**Memcached**	**Redis**	**Apache Ignite**
Type	Distributed in-memory data grid & database	In-memory key-value store & NoSQL database	Simple in-memory key-value cache
Persistence	Yes (supports disk persistence)	Yes (via RDB, AOF, or snapshot-based persistence	No (RAM-based, volatile cache only)
Data Storage	Stores structured & unstructured data	Key-value store with rich data types	Key-value store with only string-based values
Data Types	Supports SQL tables, key-value, and compute grid	Supports strings, hashes, lists, sets, streams	Supports only key-value pairs (strings)
Query Support	Full ANSI SQL, distributed joins, indexes	Supports Lua scripting and basic queries	Horizontally scalable via client-side hashing
Scalability	Horizontally scalable peer-to-peer cluster	Horizontally scalable with replication & sharding	Horizontally scalable via client-side hashing
Clustering Model	True distributed computing (peer-to-peer)	Master-Slave or Cluster-based	Client-based hashing (no internal clustering)
Data Replication	Supports synchronous & asynchronous replication	Supports replication (Master-Slave, Cluster)	No built-in replication, relies on multiple nodes
Fault Tolerance	High (supports automatic failover, backups)	High (supports replica failover)	Low (data loss if a node fails)
Transactions	Supports ACID transactions	Supports transactions (but not fully ACID)	No transaction support
Performance	Fast but heavier due to distributed computing	Fast with low-latency reads/writes	Extremely fast but limited features
Use Cases	Distributed caching, real-time analytics, SQL queries	Caching, real-time messaging, leaderboard ranking	Simple caching, session storage, web acceleration
Best For	Enterprise-grade caching & computing	General-purpose caching & data storage	Lightweight, fast caching only

In summary, Memcached is best at delivering simple, fast caching solutions, while Redis and Apache Ignite offer more advanced features like data persistence as well as diverse data structures.

CHAPTER 9

REGRESSION

Navdeep Kaur[1], Raviraj[2] and Arshdeep Singh[3]

[1,2,3]GNA University, Phagwara

9.1 Introduction:

A statistical technique used to simulate and analyze the relationship or more independent variables and a dependent variable is regression analysis. Graphing patterns, developing predictions, and understanding the relationships among variables are the goal.

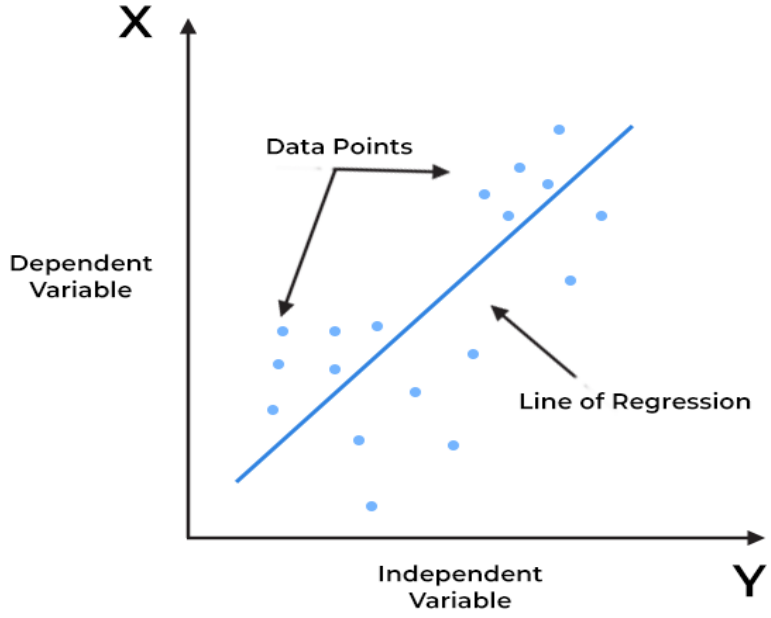
Dependent Variable (Target): The dependent variable is the result that we wish to forecast or explain. Another name for it is the target variable. Eg: stock price.

Independent variable (Predictors): The input that influences the dependent variable is called independent variable. Eg: Interset rate, market trend, company revenue etc.

9.2 Types of regression models: Here are the main types of regression models used in data engineering.

1 Linear Regression
2 Multiple Regression
3 Logistic Regression
4 Polynomial Regression
5 Ridge and Lasso Regression

9.2.1 Linear Regression: One Supervised learning method that is applied in machine learning is known as linear regression. It is the most popular and simplest algorithm. This method is predictive analysis. It predicts outcomes for real- valued and continuous variables. The reason why it is known as linear regression is that it illustrates the linear relationship between dependent and independent variables (as shown in fig.9.1).



.

Fig. 9.1: Linear Regression

It illustrates the connections between the independent and dependent variables by drawing a straight line with a slope.

Equation:

$Y = bo + b1x + \epsilon$

Here

b0= intercept of the line

b1= Linear regression coefficient

Y=Dependent Feature

x= Independent Feature

ϵ = Error Term

Regression line: A straight line represting the relationship between independent and dependent variable is reffered to as regression line.

Best fit line: A straight line that best fits the data on a scatterplot.

Types of Linear Regresssion: Two types of linear regression can be identified:

I. **Simple Linear Regression:** Simple linear regression is the process of estimating the value of a dependent variable using only one independent variable.

Equation: $Y=\beta0+\beta1X+\epsilon$

Y= Dependent Feature

X= Independent Feature

$\beta0$=intercept of the line

ϵ = Error Term

II. **Multi linear regression**:

Multi linear regression is the process of using multiple independent variables to predict the value of a dependent variable. Multiple linear regression is the process of predicting a dependent variable's value using more than one independent variable.

Formula: $Y=\beta0+\beta1X1+\beta2X2+...+\beta nXn+\epsilon$

In this case:

Y= Dependent Feature

$\beta0$=intercept

$\beta1, \beta2, \text{------} \beta n$ = Regression Coefficients

X1,X2,----Xn = Independent Feature

ϵ = Error Term

9.2.2 Logistic Regression:

Logistic regression is a prevalent machine learning method that is based on supervised learning paradigm . Predicting a categorical dependent variable's value from a given set of independent variables is its primary objective**.**

The results must be discrete or categorical since logistic regression estimates the output of a categorical dependent variable. Instead of providing exact values like 0 or 1, the results provide probabilities between 0 and 1. The results can be values like Yes or No, True or False, 0 or 1.

Even though logistic regression and linear regression are analogous, they have different uses. Logistic regression is applied to classification problems, while linear regression is applied to regression problems.

In logistic regression, a "s" shaped logistic function as shown in fig.9.2 which predicts two highest values (0 or 1) is fitted as opposed to a regression line. The logistic function is demonstrated in the image below:

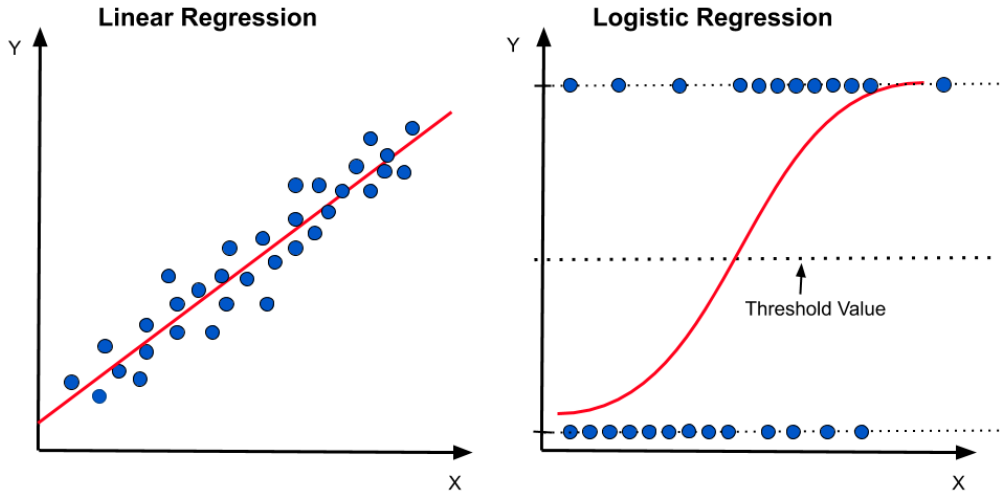


Fig. 9.2: Linear v/s Logistic Regression

Logistic Function (sigmoid function):

A mathematical operator for translating predicted values into probability is the sigmoid function.

It converts any real integer to a value in the range between 0 and 1.

The resulting "S"-shaped curve of logistic regression should remain within the interval.

The sigmoid function or logistic function is the term used to refer to such an S-shaped curve.

Type of Logistic Regression: Logistic Regression can be categorized into three types depending on the nature of the dependent variables:

- **Binomial**: There are just two possible values for the dependant variable in binomial logistic regression.. e.g: Pass or Fail or 0 or 1.
- **Multinomial**: There are three or more unordered categories exist for the dependent variables. e.g: predicting three species of flowers.
- **Ordinal**: There are three or more ordered categories that the dependent variable can have . e.g: “High”, “Medium”, or “Low”.

Logistic Regression Formula:

For a given input X=(X1,X2,...,Xn) the probability that the output Y is 1 is given by:

$$P(Y=\frac{1}{X})=\frac{1}{1+e^{-(\beta 0+\beta 1X1+\beta 2X2+\cdots+\beta nXn)}}$$

In this case:

- P(Y=1|X) = probality that Y belongs to class 1.
- β0 = intercept.
- β1,β2,...,βn = coffecients of the model.
- X1,X2,...,Xn = input features.

9.2.3 Difference between Linear and Logistic Regression:

Table 11: Difference between linear and logistic regression

Feature	Linear Regression	Logistic Regression
Model	Supervised Regression Model	Supervised Classification Model
Purpose	Predicts a **continuous** output (e.g., price, weight).	Predicts a **categorical** outcome (e.g., pass/fail, spam/not spam).
Output Type	Continuous numeric values (e.g., 0.5, 2.3, 150).	Probabilities between 0 and 1, then mapped to classes (e.g., 0 or 1).
Equation	$Y=\beta0+\beta1X+\epsilon$	$P(Y=1)= \frac{1}{1+e^{-(\beta0+\beta1X)}}$
Function Used	Uses a straight-line equation	Uses a sigmoid (logistic) function
Dependent Variable	Continuous (e.g., house prices, salaries).	Continuous (e.g., house prices, salaries).
Interpretability	Directly interpretable coefficients	Coefficients affect probability, not direct changes in output.
Use Cases	Predicting prices, stock trends, etc.	Spam detection, medical diagnosis, etc.
Decision Boundary	Linear (straight line in 2D).	Non-linear in many cases.

9.2.4 Polynomial Regression

Polynomial regression extends the work of linear regression by utilizing a polynomial function to describe the association between the independent feature (X) and the dependent feature (Y).While linear regression is based on a straight line relationship , polynomial regression allows for curves , hence it is suitable for capturing complex patterns in data .

For example: When trying to predict population growth over a period , the trend may not be linear but could show the tendency to curve . polynomial regression is handy in such situations.

Mathematical foundation:

The general formula for polynomial regression:

$$Y=\beta0+\beta1X+\beta2X2+\beta3X3+\cdots+\beta nXn+\epsilon$$

X= Independent Feature

Y= Dependent Feature

B0, β1, β2,----, βn= Regression Coefficients

ϵ= random error

How it differ from Linear Regression:

Linear Regression Fits a straight line:

$$Y=\beta0+\beta1X+\epsilon$$

If the data shows a curve , a straight line may not be sufficient instead, we introduced Polynomial terms:

$$Y=\beta0+\beta1X+\beta2X2+\beta3X3+\cdots+\beta nXn+\epsilon$$

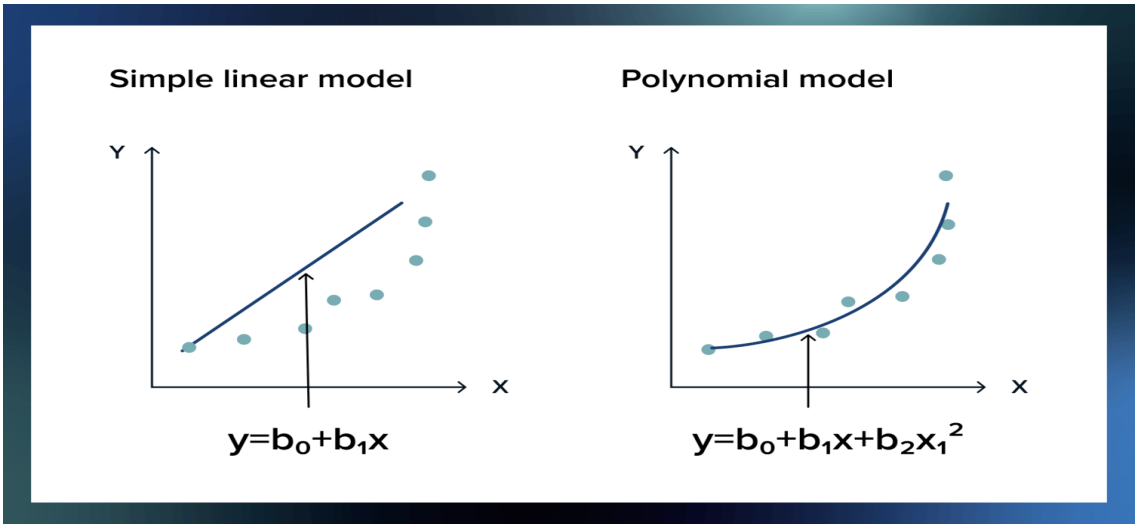


Fig. 9.3: Simple and polynomial model

Application of Polynomial Regression:

1. Forecasting and Trend Analysis:

Polynomial regression is most commonly applied in trend analysis to predict consumer behavior, market trends, or sales trends. It is able to capture peaks, bottoms, and seasonality, as opposed to linear models. Sales data in the retail sector, for example, tend to fluctuate due to promotions, holidays, and economic shifts. Inventory control and strategic planning are enhanced in companies by being able to anticipate future trends more accurately by using a polynomial curve.

1. Medical and Biological Data:

Medical studies often apply polynomial regression in the examination of dose-response associations, wherein a drug's activity can increase, plateau, or decline with various levels of dosage. It also monitors non-linear growth trends like height, weight, or improvement rates over a period of time. Polynomial regression models these tendencies to enable physicians to predict, plan treatment for, and optimize patient care.

2. The Financial Sector:

In finance, polynomial regression can be applied to analyzing investment risks, predicting stock prices, and researching market trends. Polynomial regression assists in the determination of major trends and turning points because financial information often follows non-linear trends. It also assists in risk evaluation by analyzing credit scores, trends in spending, and other fiscal indicators.

3. Environmental Science:

In environmental modeling, where data often have complex patterns, polynomial regression is effective. It is applied, based on a range of environmental factors to predict temperature change, rainfall patterns, and levels of pollution. For example, polynomial regression is applied in air quality models to forecast weather-dependent pollution concentrations, which improves public health planning.

4. Engineering and Physics:

Engineering makes use of polynomial regression to model complex systems whose behavior is not linear. Examples include temperature profiles in thermal systems, vibration signatures in machinery, and stress versus strain in materials, which often are non-linear. Polynomial regression accurately models

these trends, which allow engineers to predict system performance and optimize design parameters for more efficiency and stability.

Advantage of polynomial regression:

1) Captures of non linear relationship:

Recording Non-Linear Connections when data is modeled with complex, non-linear trends that cannot be well represented by linear regression, polynomial regression excels. In forecasting sales, for example, trends could be curved in nature or seasonally spiky that could not be detected by a linear model. These oscillations could be fitted well by using polynomial terms by the model. When data tends to follow naturally curved trends, like temperature changes, population growth, or stock market behavior, polynomial regression is thus very effective.

2) Adaptable Model Structure:

Since the degree of the polynomial can be varied according to data complexity, there is flexibility offered by polynomial regression. Though more complex patterns are captured using a higher-degree polynomial, for simple curves, a lower-degree polynomial may be adequate. For instance, varying the degree of the polynomial helps to model various stress-strain behavior in engineering procedures involving complex mechanical behavior. While choosing higher-degree polynomials, care needs to be taken so that the process does not become overfitting.

3)Improved Accuracy for Curved Data:

Polynomial regression is superior to linear models in cases where data points follow curved patterns, for example, exponential increases, parabolic motion, or cyclical trends. In biology, for example, a patient's growth curve can rise rapidly in youth before moderating, something that polynomial regression is more capable of anticipating. It enhances model accuracy and reduces prediction errors by closely fitting the curve to the data points.

4) Useful for Feature Development:

By incorporating polynomial features, such as squared or cubic terms, polynomial regression enhances linear models. Such modified characteristics help reveal hidden correlations between variables. For instance, squared terms for location distance or property size can enhance prediction accuracy in real estate price models. Through feature engineering, model performance can be improved without necessitating a shift to more complex machine learning models.

Disadvantage of polynomial Regression:

1. Overfitting Risk:

High-degree polynomials tend to make polynomial regression models excessively complex. They may improve the fit of the model to the training data, but often impair the ability of the model to generalize well to new data. This causes poor prediction performance through overfitting, which occurs when the model captures noise or minor variations instead of actual trends.

2. **Increased Complexity of Computation:**
   The computational demands of the model increase with the degree of the polynomial. The model is slower and less efficient when working with higher-degree polynomials because they need more computations and coefficients, especially when working with large datasets.
3. **Sensitive to Outliers:** Models based on polynomial regression are highly vulnerable to outliers. Because of the versatility of the polynomial curve, even a few outlier data points can distort the model, generating erroneous predictions and unstable models.
4. **Interpretation Challenges:**
   The polynomial regression coefficients are harder to interpret as the degree of the polynomial increases, unlike linear regression, whose coefficients have clear interpretations. It is hard to give a simple explanation of the variable relationship due to this complexity.

9.2.5 Ridge and lasso Regression

Two of the most frequently utilized techniques in machine learning are ridge and lasso. Ridge and Lasso regression are techniques aimed to improved the performance of linear regression models, especially in multicollinearity or overfitting scenarios . These are regularization methods that involve adding a penalty term to the standard linear regression equation to constrain the model's complexity.

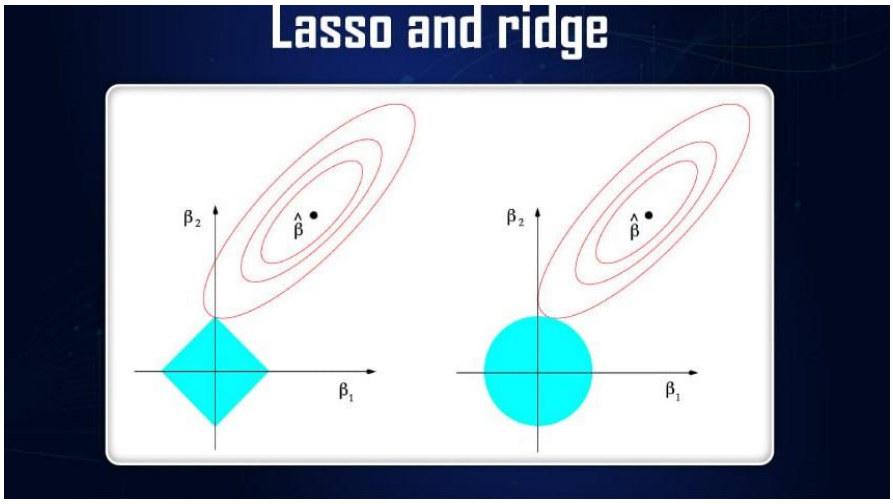


Fig. 9.4: Ridge and lasso regression

Ridge Regression: Ridge Regression is a form of linear regression as shown in fig.9.4 that incorporates an L2 penalty to mitigate overfitting and address multicollinearity. It is particularly beneficial when the dataset contains numerous correlated predictors. Ridge Regression is also called L2 regularization.

Formula of Ridge Regression: Ridge regression introduces a penalty term to the ordinary least squares (OLS) cost function:

$$\sum(Yi-Y\hat{}i)2+\lambda\sum\beta j2$$

In this case:

- Yi = true value
- Y^i = forecasted value.
- λ = Regularization Parameter .
- ∑βj2 = sum of the squared coefficients, which penalizes large coefficient values.

Advantage of Ridge Regression:

1. Reduces of Overfitting: Ridge regression simplifies the model by adding a regularization term (L2 penalty) to the loss function. This improves the model's ability to generalize to new data by keeping it from fitting noise in the original data. Therefore, in the case of very complex datasets, ridge regression performs better than standard linear regression.

2. Deals with Multicollinearity: In highly correlated independent variables datasets, ridge regression is very effective. Shrinking the coefficients makes the models stable and improves predictive power by minimizing multicollinearity's influence.

3. Improved Coefficients' Stability: Ridge regression prevents coefficients from increasing to unusually large numbers using L2 regularization. The model stronger and more reliable through this stabilization, which reduces its sensitivity to small changes in the data.

4. Effective in high dimensional data: Ridge regression effectively minimizes overfitting by penalizing coefficients of large magnitude in data with more features than observations (like gene or text data). Due to this, it is ideal for solving high-dimensional problems.

5. Works well with feature scaling: Standardized variables provide the optimum output from ridge regression. This enhances the efficiency and accuracy of the model by ensuring that the regularization term accounts for each coefficient uniformly.

Disadvantage of Ridge Regression:

1) Limited Interpretability: Ridge regression complicates the identification of the most important factors in the model since it proportionally shrinks all of the coefficients. In cases where feature importance is paramount, this reduced interpretability can be challenging.

2) Less Effective for Strongly Correlated Features: Ridge regression can be challenged by highly correlated attributes, although it can handle multicollinearity. Predictive accuracy could be lost in such scenarios since the model could still assign similar coefficients to highly correlated variables.

3) Parameter Tuning Required: To balance bias and variance, ridge regression relies on selecting the perfect regularization value (α). Full cross-validation is often required to obtain the perfect value, making the model-building.

4) Needs Feature Scaling: Ridge regression does its best when features are standardized (e.g., z-score normalization). Without scaling, higher-value features can overshadow the regularization penalty, resulting in noisy coefficient estimates.

9.2.6 Lasso Regression

Least Absolute Shrinkage and Selection Operator, or Lasso Regression, is a form of linear regression that improves feature selection and stops overfitting by including an L1 regularization penality. Rather than only minimizing the total sum of squared errors, as ordinary least squares (ols) does, Lasso includes a penalty that increases with the absolute coefficients. Through the setting of some of the coefficients to exactly zero, the approach effectively excludes parts from the model that aren't important.

Formula of lasso regression:

Lasso regression adds an L1 pentaly to the conventional linear regression cost function.

$$\sum(Yi-Y\hat{}i)2+\lambda\sum|\beta j|$$

Where:

- Yi = actual target value.
- Y^i = predicted value.
- βj = represents regression coefficients.
- λ = Regularization Parameter.
- The term ∑|βj| ensures that some coefficients shrink to zero, eliminating unnecessary features.

Advantage of Lasso Regression:

1. Automatic Variable Selection (Feature Selection): Lasso regression is automatically variable selection by shrinking some coefficients to zero, keeping only the most important features. It creates a simpler model that is more interpretable, potentially advantageous for high-dimensional datasets where predictor space is greater than sample space, as seen through examples of genomics, text classification, etc.

2. Efficient for High-Dimensional Data: Lasso regression performs well in situations where the number of predictor variables is greater than the number of observations. Lasso takes advantage of its capacity to differentiate and select relevant predictors, managing the high-dimensional data very efficiently.

3. Manages Multicollinearity: In situations when independent variables are highly correlated (i.e. redundancy exists), it is more likely that Lasso will select one variable and shrink the other(s) to zero. This is very important for reducing the redundancy in a model regarding co-correlation, improving model stability when redundancies are present. Therefore, Lasso will most likely work well in the presence of correlated features.

4. Prevents Overfitting: Lasso's L1 regularization will penalize large coefficients in order to reduce complexity over the model itself. Again, this reduces danger for overfitting, while increasing the model's ability for generalization of unseen cases/distribution since overfitting has been managed. Thereby, it is generally a well-performing model with unseen data.

Disadvantage of Lasso Regression:

1. Unstable Inferential Consequences from Selecting Highly Correlated Features: Lasso may fit one of a group of features that are highly correlated while ignoring the others. Thus, Lasso can produce feature selection that is inconsistent across models, especially when the dataset is complex.

2. Limited in Selecting Groups of Variables: In contrast to Ridge regression that shrinks correlated features jointly, Lasso can miss significant variable combinations, possibly compromising model performance in such situations.

3. Computationally Intensive for Large Datasets: While effective for feature selection, Lasso is computationally intensive for very large data or very high-dimensional data.

4. Risk of Underfitting: If the regularization parameter is too large, Lasso can over-regularize the model by reducing too many coefficients to zero, resulting in underfitting and bad prediction.

Difference between Ridge and Lasso Regression:

Table 12: Difference between ridge and lasso regression

Feature	Lasso Regression (L1 Regularization)	Ridge Regression (L2 Regularization)
Regularization Type	Uses L1 norm (absolute values of coefficients).	Uses L2 norm (squared values of coefficients).
Feature Selection	Can shrink some coefficients to exactly zero, effectively selecting features.	Shrinks coefficients but does not set them to zero, keeping all features.
Model Complexity	Produces sparse models by eliminating unimportant features.	Retains all features but reduces their impact.
Overfitting Control	Stronger feature selection helps reduce overfitting	Prevents overfitting but keeps all variables in the model.
When to Use	When feature selection is needed (high-dimensional data).	When all variables are important but need regularization
Computation Complexity	Can be computationally expensive for large datasets.	Computationally less expensive than Lasso
Effect on Coefficients	Some coefficients become exactly zero.	Coefficients are small but never exactly zero.
Handling Multicollinearity	Performs better in eliminating irrelevant correlated features.	Distributes coefficient weights among correlated features
Interpretability	Easier to interpret due to fewer nonzero coefficients	Harder to interpret as all features remain
Example Use Cases	Feature selection in genetics, finance, and text processing	Regression models where all predictors contribute (e.g., econometrics, physics)

9.3 Principle Component Analysis (PCA)

Principle Component Analysis (PCA): Mathematician Karl Pearson first introduced the PCA method in 1901. Data in a space of higher dimension is projected onto data in a space of lower dimension. The data variance should be higher in the lower dimensional space.

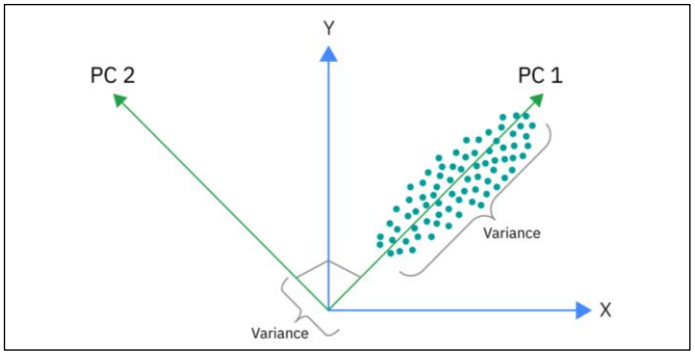


Fig. 9.5: Principle component analysis

Principle component analysis is just one of the unsupervised learning techniques that can be utilized to decrease data dimensionality. The statistical procedure called principle compo-nent analysis applies an orthogonal transformation in order to transform a collection of correlated variables into a collection of uncorrelated variables as visualize in fig.9.5.

Principle component analysis is the most prevalent method of data analysis and machine learning used for predictive models.

9.3.1 Step-By-Step Explanation of PCA:

Step1: Standardization:

First, we must standardize our dataset, making sure that the standard deviation is one and the mean of each variable is zero**..**

$Z=X-\mu / \sigma$

μ = mean of the independent variables

σ= Standard Deviation

Step2: Covariance Matrix Computation:

The degree of joint variability between two or more variables, or how much they differ from one another, is measured by covariance. The following formula can be used te determine the covariance:

$$Cov(x1, x2)= \sum_{i=1}^{n} (x1i-x1-)(x2i-x2-)/n-1$$

value of covariance is Zero, Positive, Negative..

- Positive: x1 rises x2 also rises
- Negative: x1 rises x2 also rises
- Zero: No direct connection

Step3: Calculate Eigen value and Eigen vectors of covariance matrix to determine principle components:

Let A be a square nXn matrix and X be a non- zero vector for which

$$AX= \lambda X$$

For certain scalar values λ, then λ are referred to as the eigen values of matrix A and X are refferred as the eigen vectors of matrix A , corresponding to the eigen values.

It can also be expressed as:

$AX- \lambda X=0$

$(A-\lambda I)X=0$

Where I is the indentify matrix of the same shape as matrixA. Above condition will be satisfied only if

$$|A-\lambda I|=0$$

This equation is reffered to as the characteristic equation.

We can determine the eigenvalues \ λ, and thus corrresponding eigen vectors can be determine using the equation.

$$AX= \lambda X$$

Advantages of PCA:

1. Dimensionality Reduction: One popular method for reducing dimensionality is principle component analysis, where one decreasing the variables of a dataset. With the minimum possible number of variables, PCA reduced the burden on data analysis, increases performance, and improves visualization of the data.

2. Feature Selection: Determining the most significant variables in a collection, or feature selection, is facilitated through the use Principal Component Analysis. This comes in handy with machine learning as there could be numerous factors which are hard to separate, hence the need to extract the most relevant variables.

3. Visualization Simpler: PCA is able to map high-dimensional data onto two or three dimensions so that it can be analyzed more simply by diminishing the variables.

Disadvantages of PCA:

1. Main Component Interpretation: Because the primary components by Principal Component Analysis are linear mixes of the source Variables, it may be tricky to comprehend how they connect with the source variables. This complexity may make it harder to interpret PCA results to other individuals.

2. Scaling of the Data: Size of the data influences Principal Component Analysis. PCA would not work optimally if data is not appropriately scaled. So, standardization of data prior to applying principal Component Analysis is necessary.

3. Computational Complexity: For big datasets, Principal Component Analysis may be computationally intensive. This particularly holds when numerous variables exist within the dataset.

Example of PCA:

Examine the dataset with two variables (X1, X2) that follows:

Sample	X1	X2
A	2.5	2.4
B	0.5	0.7
C	2.2	2.9
D	1.9	2.2
E	3.1	3.0

Step1: Calculate the Mean:

Determine the Mean for every Variable:

$$\mu X1=2.5+0.5+2.2+1.9+3.1/5=2.04$$

$$\mu X2=2.4+0.7+2.9+2.2+3.0/5=2.24$$

Step2: Center the data:

Subtract the mean for each variable:

Sample	X1(Centered)	X2(Centered)
A	0.46	0.16
B	-1.54	-1.54
C	0.16	0.66
D	-0.14	-0.04
E	1.06	0.76

Step 3: Calculate the Covariance Matrix:

$$\text{Cov(X1, X2)}=\frac{1}{n-1}\sum_{i=1}^{n}\text{X1i}-\text{X1}^{-})(\text{X2i}-\text{X2}^{-})$$

Where n=5

$X1^{-}$ and $X2^{-}$ = Mean of X1 and X2

Covariance Matrix form:

$$=\begin{bmatrix}\text{varX1} & \text{covX1, X2}\\ \text{covX1, X2} & \text{varX2}\end{bmatrix}$$

Calculated Variance:

$$\text{Var(X1)}=\frac{(0.46)^2+(-1.54)^2+(0.16)^2+(-0.14)^2+(1.06)^2}{4}$$

$$\frac{0.2116+2.3716+0.0256+0.0196+1.1236}{4}$$

$$=\frac{3.752}{4}=0.938$$

$$\text{Var(X2)}=\frac{(0.16)^2+(-1.54)^2+(0.66)^2+(-0.04)^2+(0.76)^2}{4}$$

$$=\frac{0.0256+2.3716+0.4356+0.0016+0.5776}{4}$$

$$=\frac{3.412}{4}=0.853$$

$$\text{Cov(X1,X2)}=\frac{(0.46)(0.16)+(-1.54)(-1.54)+(0.16)(0.66)+(-0.14)(-0.04)+(1.06)(0.76)}{4}$$

$$=\frac{0.0736+2.3716+0.1056+0.0056+0.8056}{4}=0.8405$$

Final Covariance Matrix:

$$\begin{bmatrix}0.938 & 0.8405\\ 0.8405 & 0.853\end{bmatrix}$$

Step 4: Calculate Eigen values and Eigen Vectors:

Eigen Values are found by solving:

$$\begin{bmatrix}0.938-\lambda & 0.8405\\ 0.8405 & 0.853-\lambda\end{bmatrix}=0$$

$$=(0.938-\lambda)(0.853-\lambda)-(0.8405)^2=0$$

$$=(0.938\times0.853)-(0.938\lambda+0.853\lambda)+(\lambda)^2-(0.8405)^2=0$$

$$=0.800514-1.791\lambda+\lambda^2-0.70644=0$$

$$=\lambda^2-1.791\lambda+0.0941=0$$

Using this Quadratic Equation using Quadratic Formula:

$$\lambda=\frac{-1.791\pm\sqrt{(-1.791)^2-4(0.0941)}}{2}$$

$$=\frac{-1.791\pm\sqrt{3.2078-0.3764}}{2}$$

$$=\frac{-1.791\pm\sqrt{2.8314}}{2}$$

Approximating:

$$\frac{-1.791\pm1.6833}{2}$$

Eigen Values:

$$\lambda 1 = \frac{-1.791+1.6833}{2} = \frac{3.4743}{2} = 1.737$$

$$\lambda 2 = \frac{-1.791-1.6833}{2} = \frac{0.1077}{2} = 0.05385$$

First Eigen Value shows the most Variance.

Second Eigen Value Shows the less Variance.

Find Eigen Vectors:

$$\begin{bmatrix} 0.938-\lambda & 0.8405 \\ 0.8405 & 0.853-\lambda \end{bmatrix} \begin{bmatrix} X \\ Y \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

For $\lambda 1 = 1.737$

$$\begin{bmatrix} 0.938-1.737 & 0.8405 \\ 0.8405 & 0.853-1.737 \end{bmatrix} \begin{bmatrix} X \\ Y \end{bmatrix} = 0$$

$$\begin{bmatrix} -0.7990 & 0.8405 \\ 0.8405 & -0.884 \end{bmatrix} \begin{bmatrix} X \\ Y \end{bmatrix} = 0$$

Solve:

$$-0.799X+0.8405Y=0$$

$$X=\frac{0.8405}{0.7990}Y$$

$$=1.0519Y$$

Thus Eigen Vector:

$$V1=\begin{bmatrix} 1.0519 \\ 1 \end{bmatrix}$$

Normalized:

$$V1=\begin{bmatrix} 0.743 \\ 0.669 \end{bmatrix}$$

For $\lambda 2 = 0.05385$

$$\begin{bmatrix} 0.938-\lambda & 0.8405 \\ 0.8405 & 0.853-\lambda \end{bmatrix} \begin{bmatrix} X \\ Y \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} 0.938-0.05385 & 0.8405 \\ 0.8405 & 0.853-0.05385 \end{bmatrix} \begin{bmatrix} X \\ Y \end{bmatrix} = 0$$

$$\begin{bmatrix} 0.8842 & 0.8405 \\ 0.8405 & 0.7991 \end{bmatrix} \begin{bmatrix} X \\ Y \end{bmatrix} = 0$$

Solve:

$$0.8842x+0.8405y=0$$

$$X=\frac{0.8405}{0.8842}Y$$

$$=-0.95y$$

Eigen Vector:

$$V2=\begin{bmatrix} -0.95 \\ 1 \end{bmatrix}$$

Normalized:

$$V2=\begin{bmatrix} -0.713 \\ 0.706 \end{bmatrix}$$

Final Eigen Value and Eigen Vectors:

$$\text{Eigen Value1} = 1.737, \textbf{Eigenvector 1} = \begin{bmatrix} 0.743 \\ 0.669 \end{bmatrix}$$

Eigen Value2 = 0.05385, **Eigenvector 1** = $\begin{bmatrix}0.713\\0.706\end{bmatrix}$

9.4 Canonical Correlation Analysis:

A statistical technique for assessing if two sets of variables are correlated is referred to as canonical correlation analysis, or CCA. CCA identifies the best linear combinations of two sets of variables that have the maximum correlation, as compared to simple correlation, which works with the correlation between two variables.

Example: Imagine that you are looking at customer data:

Set 1 (X): Demographic attributes (Age, Income, Education)

Set 2 (Y): Purchase behavior (Expenditure, Frequency of purchase, Product categories)

Canonical CorrelationAnalysis(CCA) helps identify the correlation between these two groups, revealing trends that link customer demographics with their purchasing behavior.

Why is CCA Important in Data Engineering:

CCA is useful while working with multi-source data across domains like:

Feature Engineering – Establishing meaningful relations between different data types.

Dimensionality Reduction – Reducing high-dimensional data sets preserving relationships.

Data Integration – Combining information from more than one data set.

Recommendation Systems – Interpreting user choice across platforms.

Time Series Analysis – Finding dependencies between various time series signals.

3.3 How Does CCA Work:

Step 1: Find Two Data Sets (X and Y)

We have two matrices:

X (n × r): Dataset 1 (n observations, r characteristics)

Y (n × s): Dataset 2 (n observations, s characteristics)

Step 2: Find Canonical Variables

CCA finds two sets of weight vectors:

u (r × 1) corresponding to X

v (s × 1) corresponding to Y

In such a manner that the transformed variables:

U=Xu

V=Yv

display the highest correlation.

Step 3: **Compute Canonical Correlation Coefficients**

The correlation between U and V (canonical correlation) is maximized, yielding a canonical correlation coefficient (ρ).

If ρ is close to 1, the two datasets have a high correlation.

9.4.1 Applications of CCA in Data Engineering:

Feature Engineering in Machine Learning

Canonical Correlation Analysis (CCA) is important to feature engineering as it detects useful relations between various sets of features, thus improving model performance. Through detecting correlated feature sets, CCA enables machine learning models to take advantage of other underlying patterns in the data. For example, in online shopping, CCA can be employed to combine text embeddings (X) from product descriptions and customer interactions (Y) for predicting buying behavior. This facilitates a

more holistic representation of data and enhances the accuracy of models by considering several viewpoints of one problem.

Multimodal Data Fusion

In data engineering, it is typical to combine and examine data from multiple sources to gain insights. CCA makes this possible by identifying patterns between multimodal datasets, so that the combined data maintains significant patterns. For instance, a company that examines website user experience can combine website logs (X) with customer support interactions (Y) to have a complete picture of user behavior. By recognizing correlated patterns, companies can enhance customer service strategies and maximize user engagement.

Finance & Time Series Analysis

In financial analysis, CCA is a useful tool for finding associations between different financial metrics and outside economic variables. It assists in examining how different financial metrics affect one another over time. For instance, CCA can reveal relationships between stock prices (X) and economic variables like inflation rates, interest rates, or GDP growth (Y). Through these associations, financial analysts can more accurately forecast market trends and make sound investment choices.

Healthcare Analytics

Healthcare data is usually multidimensional and complicated in nature and hence needs sophisticated methods to reveal concealed relationships. CCA is popularly used to correlate genetic data (X) with results of medical tests (Y) to determine possible health hazards and genetic susceptibility to diseases. For instance, researchers can use CCA to identify genes having significant association with diseases like cancer or diabetes. By knowing the associations, medical scientists can devise targeted treatment plans and individualized medicine strategies.

9.4.2 Advantages of Canonical Correlation Analysis (CCA)

Revealing Latent Relationships

One of the main benefits of CCA is that it can reveal latent relationships between two sets of characteristics, even where variables in each set have high internal correlations. This allows data scientists to learn new things that may not be immediately obvious through standard correlation methods.

Dimensionality Reduction

CCA lowers the dimensionality of information by finding the most influential linear combinations of features in both datasets. It not only decreases computational complexity but also enhances model efficiency by removing redundant information.

Interpretability and Clarity

CCA delivers understandable and interpretable results by creating canonical variables which are the strongest correlated pairs of features between two data sets. This allows researchers and analysts to better comprehend underlying relationships among various variables.

Multivariate Analysis Capability

While individual correlation methods focus on studying relationships between single variables, CCA allows for the study of multiple variables at one time. This renders it useful in exploring intricate interactions in high-dimensional data

Robustness to Non-Normality

CCA demonstrates resilience against deviations from normality assumptions and can be applied even when sample sizes are relatively small. This makes it a flexible and adaptable tool for real-world applications across various domains.

9.4.3 Limitations of Canonical Correlation Analysis (CCA)

Sensitivity to Outliers

One of the key limitations of CCA is its susceptibility to outliers. Outliers in either data set can skew the estimation of the canonical correlations and vectors, resulting in erroneous conclusions. Outlier detection and robust statistical approaches are generally required preprocessing techniques to counter this.

Difficulty in Interpreting Original Variables

While canonical variables are simple to interpret in the context of their correlation, it is sometimes difficult to comprehend the contribution of the original variables towards these canonical variables. This might make it hard to interpret CCA results into useful information.

Assumption of Equal Covariance Matrices

CCA presumes that the two sets of variables share the same covariance structures, which might not always be the case in real-world situations. Violations of this assumption can result in biased estimates and restrict the generalizability of findings.

Large Sample Size Requirement

For accurate outcomes, CCA generally needs a relatively large number of samples. In situations where there is little data, the success of CCA can be compromised, thus it is less preferable for analyses involving few observations.

9.4.4 Problem Statement:

Consider a dataset that provides information about the study habits of students along side their performance in two subjects: English and Hindi.

- Study Habits (Independent Variables):
- X1= Hours Dedicated to Study
- X2= Count of Practice Exams Taken
- Exam Performance (Dependent Variables)
- Y1= Score in English
- Y2= Score in Hindi

We have gathered the following data in five students:

Student	Hours Dedicated to study(X1)	Count of practice Exam Taken(X2)	English Score(Y1)	Hindi Score(Y2)
1	2	1	50	55
2	3	2	60	65
3	5	3	80	85
4	7	5	90	92
5	8	6	95	98

Our objective is to determine the canonical correlation between exam performance (Y) and study habit(X).

Step 1. Calculate the covariance matrices:

For X, Y, and their cross-covariance, we calculate the covariance matrices.

- Determine the Mean for Every Variable

$\bar{x}1 = \frac{2+3+5+7+8}{5} = \frac{25}{5} = 5$

$\bar{x}2 = \frac{1+2+3+5+6}{5} = \frac{17}{5} = 3.4$

$\bar{Y}1 = \frac{50+60+80+90+95}{5} = \frac{375}{5} = 75$

$\bar{Y}2\ \frac{55+65+85+92+98}{5} = \frac{395}{5} = 79$

Calculate the covariances:

Formula of Covariances: $Cov(A,B) = \frac{1}{n-1}\sum(Ai-\bar{A})(Bi-\bar{B})$

Let's manually compute each covariance using this formula.

- Covariance Matrix of X(S_XX)

Calculate cov(X1,X1):

$$Cov(X1,X1) = \frac{1}{4}[(2-5)^2+(3-5)^2+(5-5)^2+(7-5)^2+(8-5)^2]$$

$$= \frac{1}{4}[(3)^2+(-2)^2+(0)^2+(2)^2+(3)^2]$$

$$= \frac{1}{4}[9+4+0+4+9] = \frac{26}{4} = 6.5$$

$$Cov(X2,X2) = \frac{1}{4}[(1-3.4)^2+(2-3.4)^2+(3-3.4)^2+(5-3.4)^2+(6-3.4)^2]$$

$$= \frac{1}{4}[(-2.4)^2+(-1.4)^2+(-0.4)^2+(1.6)^2+(2.6)^2]$$

$$= \frac{1}{4}[5.76+1.96+0.16+2.56+6.76] = \frac{17.2}{4} = 4.3$$

$$Cov(X1,X2) = \frac{1}{4}[(2-5)(1-3.4)+(3-5)(2-3.4)+(5-5)(3-3.4)+(7-5)(5-3.4)+(8-5)(-3.4)]$$

$$= \frac{1}{4}[(-3)(-2.4)+(-2)(-1.4)+(0)(-0.4)+(2)(1.6)+(3)(2.6)]$$

$$= \frac{1}{4}[7.2+2.8+0+3.2+7.8] = \frac{21}{4} = 4.75$$

The S_XX Matrix is

$$Sxx = \begin{bmatrix} 6.5 & 4.75 \\ 4.75 & 4.3 \end{bmatrix}$$

To follow the step to calculate the (S_YY) and (S_XY) matrix:

$$Sxy = \begin{bmatrix} 73.75 & 69.25 \\ 57.75 & 55.9 \end{bmatrix}$$

$$Syy = \begin{bmatrix} 400 & 370 \\ 370 & 390 \end{bmatrix}$$

Step 2: Solve the Canonical Correlation Eauation:

The following formula is used to determine the canonical correlation coefficient:

$$\rho^2 = \text{Eigenvalues}(S^{-1}xxSxyS^{-1}yySyx)$$

First Find Inverse: we first find $S^{-1}xx$ and $S^{-1}yy$

Using the formula for 2*2 matrix inverse:

$$A^{-1} = \frac{1}{\det(A)}\begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

For Sxx:

$$\det(Sxx): (6.5 * 4.3) - (4.75* 4.75) = 5.3875$$

$$S^{-1}xx= \frac{1}{5.3875}\begin{bmatrix} 4.3 & -4.75 \\ -4.75 & 6.5 \end{bmatrix}$$

$$= \begin{bmatrix} 0.798 & -0.881 \\ -0.881 & 1.207 \end{bmatrix}$$

Similarly for $S^{-1}yy$:

$$\begin{bmatrix} 0.0204 & -0.0194 \\ -0.0194 & 0.0209 \end{bmatrix}$$

Compute $M=(S^{-1}xxSxyS^{-1}yySyx)$

After multiply these matrices:

$$M = \begin{bmatrix} 0.994 & 0.002 \\ 0.002 & 0.674 \end{bmatrix}$$

We need to find eigen values λ by solving:

$$\det(M- \lambda I) = 0$$

Where I is the identity matrix:

$$\begin{bmatrix} 0.994 - \lambda & 0.002 \\ 0.002 & 0.674 - \lambda \end{bmatrix}$$

$$= (0.994- \lambda)(0.674 - \lambda) - (0.002 * 0.002) = 0$$

$$= \lambda^2 - 1.668\,\lambda + 0.670352 = 0$$

Using this Quadratic Equation using Quadratic Formula:

$$\lambda = \frac{-(-1.668) \pm \sqrt{(-1.668)^2 - 4(1)(0.670352)}}{2(1)}$$

After solving equation we get:

$$\lambda 1 = 0.993 \text{ and } \lambda 2 = 0.675$$

Compute Cononical Correlation Coefficients:

$$\rho 1 = \sqrt{\lambda 1} = \sqrt{0.993} = 0.996$$

$$\rho 2 = \sqrt{\lambda 2} = \sqrt{0.675} = 0.821$$

After solve this equation we get,

$$\rho 1=0.996 \text{ and } \rho 2=0.821$$

This Means:

- With a very high value of 0.997, the first canonical correlation indicates a strong association.
- Strong but lower is the second canonical correlation, which stands at 0.821.

Step 3. Results:

- Exam performance (English and Hindi scores) is substantially correlated with study habits (hours studied and practice tests), according to the first canonical correlation (0.997).
- Although it is not as strong as the first connection, the second one (0.821) still shows a substantial relationship.

  As a result, students who put in more study time and take more practice exams typically do better in both English and Hindi.

9.4.5 difference between CCA and PCA:

Table 13: Difference between CCA and PCA

Feature	CCA	PCA
Purpose	Finds relationships between two sets of variables	Reduces dimensionality by finding principal components
Input Data	Two or more datasets (X and Y) that may be correlated.	A single dataset with multiple correlated variables.
Output	Canonical variables (pairs of transformed variables that maximize correlation).	Principal components (uncorrelated features that capture maximum variance).
Goal	Maximizes the correlation between linear combinations of two datasets.	Maximizes variance in a single dataset while reducing dimensionality.
Correlation	Measures the relationship between two datasets.	Measures variance within a single dataset.
Application	Used in multimodal data analysis, bioinformatics, and finance.	Used in image processing, compression, machine learning, and exploratory data analysis.
Mathematical Basis	Uses eigenvalues and eigenvectors to maximize correlations	Uses eigenvalues and eigenvectors to maximize variance.
Data Interpretation	Produces pairs of canonical variables for interpretation	Produces principal components ranked by explained variance.
Dependency	Requires at least two datasets for analysis	Works with a single dataset

A helpful way to investigate the relationship between two groups of variables is through canonical correlation analysis. By identifying optimal linear combinations to maximize mutual information, it is an extension of correlation analysis. But to gain useful insights, careful interpretation and substantiation are needed.

9.6 Analysis of Variance

ANOVA, or analysis of variance, is a statistical technique used to evaluate the means of two or more groups to see if they differ significantly. ANOVA helps determine how multiple variables impact a set of data and is often applied to A/B testing, development of features, and performance optimization.

Example:

A data engineer wants to compare the performance of three different database configurations to determine which one performs the best. Instead of making a number of pairwise comparisons (which increases the likelihood of error), ANOVA enables us to test whether at least one configuration performs significantly better or worse than the others.

9.6.1 Why is ANOVA Important in Data Engineering:

Performance Assessment – Evaluate the performance of different databases or queries.

Feature Identification – Identify key features that affect an outcome.

A/B Testing – Evaluate the effects of different system or algorithm tweaks.

Anomaly Identification – Identify abnormalities between system components.

9.6.2 How Does ANOVA Work:

Step 1: Formulate Hypotheses:

1. Null hypothesis (H_0) is that there are no significant differences between the groups means.

2.The alternative hypothesis (H_1) is that the mean of one or more groups in different from the rest.

Step 2: Calculate Variance:

ANOVA separates total variance into two:

1.Between-Group Variance (Explained Variance) – Variances due to different groups.

2. Within-Group Variance (Unexplained Variance) – Variances within each group.

We reject H_0 and conclude that at least one group is different if

the variation between groups is significantly larger than the within-group variance.

Step 3. **Calculate the F-Statistic**:

A measure of between – group variance in relation to within – group variation is the F- statistic.

F= Variance within groups/ Variance among groups

A big F-value suggests large differences between groups.

Step 4: Determine Statistical Significance:

Compare the F-statistic to a critical value (or p-value) to decide on significance:

p-value $< 0.05 \rightarrow$ Reject H_0 (there is at least one group that differs).

p-value $\geq 0.05 \rightarrow$ Fail to reject H_0 (no significant difference exists).

9.6.3 Applications of ANOVA in Data Engineering:

ANOVA is important in data engineering as it gives a statistical model through which one can measure differences among several groups or system components.

Performance measurement:

In which ANOVA facilitates the comparison of the effectiveness of various databases, query plans, or data processing architectures. Comparing measures such as query execution times, data throughput, or system latency for various configurations, data engineers are able to find the best setup for maximum performance.

Feature Identification:

ANOVA is useful for ascertaining which features have a significant impact on a target value. For example, in predictive modeling exercises, ANOVA can examine if specific attributes of the data, such as customer profiles or types of transactions, contribute meaningfully to important measures like sales, churn percentages, or system performance. Data engineers use this information to prioritize high-value features during feature engineering.

A/B testing:

A/B testing is yet another key space where ANOVA is utilized to contrast the effect of varying system modifications, e.g., algorithmic changes, caching methodologies, or hardware upgrades. By examining

differences in performance metrics among various test groups, ANOVA indicates if observed gains are statistically significant, facilitating data-driven decision-making.

Identify Anomalies:

Identify anomalies by sensing unusual deviations among system elements or sources of data. For instance, if there is data obtained from several servers that exhibits inconsistent behavior, ANOVA is able to pin down sources of discrepancies and guarantee data reliability and system stability.

9.6.4 Types of Anova:

Table 14: Types of anova

Type	Use Case
One-Way ANOVA	One-Way ANOVA compares a single factor or categorical variable between different groups (e.g., query performance within different databases).
Two – Way ANOVA	examines the effect of two variables (e.g., the interaction between database type and indexing strategy) on query performance.
Repeated Measures ANOVA	Repeated Measures ANOVA compares changes over time (e.g., API response times before and after optimization).

9.6.5 ANOVA vs. Statistical Tests:

Table 15: Difference between ANOVA and statistical tests

Feature	ANOVA	T-Test	Chi-Square
Purpose	Assess means among three or more groups	Assess means between two groups	Analyze Categorical data
Data Type	Numerical	Numerical	Categorical
Example Use Case	Assess execution times between different databases	Compare API latency before and after update	Measure user preference for new UI designs

9.6.6 Advantage of ANOVA:

1. Allows Comparison of Three or More Groups at One Time:

- Compared to t-tests, which can only analyze two groups in a single analysis, ANOVA can handle three or more groups simultaneously
- This is efficient and reduces the chances of accumulation of errors that result from doing many t-test.

2. Reduces Type I Error (False Positives):

- Running many t-tests increases the risk of Type I error (rejecting a true null hypothesis).
- Anova is able to counteract error rates, leading to more statistical results.

3. Evaluates Overall Differences Between Groups:

- It helps to determine whether one or more groups differ significantly from each other. This is useful in the fields of medical research, manufacturing, and marketing.

4. Allows for Multiple Experimental Designs:

- ANOVA may be applied across various study designs, namely:
  - One-Way ANOVA: Contrasting the mean of a solitary independent variable (e.g., different diets).
  - Two-Way ANOVA: Determining the effect of two independent variables (e.g., diet and physical exercise).
  - Repeated Measures ANOVA: Testing the same sample over a period of time (e.g., the effect of a medication over a couple of months).

5. Displays Interaction Effects (For Two-Way ANOVA):

- It tells us how two independent variables affect and interact to impact the dependent variable.
- For example: Does diet and exercise combined result in more weight loss than either variable alone?

6. More Statistically Effective:

- It increases statistical power so smaller differences between groups can be detected more precisely than running multiple t-tests.

7. Versatile and Often Used:

- It is applicable in areas including business analytics, medical studies, social sciences, manufacturing, and A/B testing.

9.6.7 Limitations of ANOVA:

1. Assumption of Normality: ANOVA assumes the data in all groups are normally distributed. The results can be misleading if the assumption is violated, particularly for small sample sizes. Non-normal data might be needed to transform or use a different non-parametric test.

2. Equal Variance Requirement (Homogeneity of Variance): ANOVA requires equal variance among all groups. If the variances significantly vary between groups (heteroscedasticity), the test can provide misleading results. Methods such as Welch's ANOVA are frequently required in such situations.

3. Sensitive to Outliers: ANOVA is extremely sensitive to outliers. Outliers tend to skew group means and increase the variance, which influences the validity of the results.

4. Limited to Mean Comparisons: ANOVA can detect only differences between group means and does not tell us which particular groups are different. Post-hoc tests (such as Tukey's HSD) must be used to follow up.

5. Assumes Independence of Observations: ANOVA presumes that observations within and between groups are independent. When this assumption is violated (such as with related data points or repeated measures), specialized tests such as repeated-measures ANOVA must be used.

6. Difficulty Handling Complex Data Structures: Although ANOVA is good for straightforward comparisons, it can be challenged by intricate data structures with several interactions and may need extensions such as factorial ANOVA or mixed-effects models.

7. Restricted to Linear Relationships: ANOVA is used to test linear relationships between variables. Non-linear trends might not be effectively captured, restricting its use in some datasets.

Chapter 10

DATA VISUALIZATION

Vikramjit Parmar[1], Manpreet Singh[2], Navdeep Kaur[3] and Jasmeet Kaur[4]

[1,3,4]GNA University, Phagwara

[2]Sant Baba Bhag Singh University, Jalandhar

10.1 Introduction

Data visualization is a crucial part of data analysis that allows us to represent information graphically, making patterns, trends, and insights easier to understand. Various types of charts and plots are available to suit different kinds of data and analytical needs. In this chapter, we will explore several essential visualization techniques, including tables, graphs, histograms, pie charts, area plots, boxplots, scatterplots, bubble plots, waffle charts, and word clouds.

Data visualization make data easier for the human brain to comprehend and draw conclusions from, data visualization involves converting information into a visual environment, like a map or graph. Making it simpler to spot patterns, trends, and outliers in big data sets is the primary objective of data visualization. The terms statistical graphics, information visualization, and information graphics are frequently used interchangeably.

Nearly every professional field benefits from data visualization. Computer scientists use it to investigate developments in artificial intelligence (AI), educators use it to show test results, and CEOs use it to communicate with stakeholders. It is also crucial for big data initiatives. Businesses required a method to swiftly and simply obtain an overview of their data as they amassed enormous amounts of data. In order to deliver valuable information, visualization tools were a logical choice.

Using data visualization to get some insights from the data is undoubtedly quicker than merely looking at a chart. The Tableau screenshot below makes it very simple to determine whether states have experienced a net loss as opposed to a profit. This is because it is clear that states have incurred a loss because, when utilizing a heat map, all the cells with a loss are colored red. Contrast this with a typical table, where a loss would be determined by looking at each cell to see if it has a negative value. In this case, data visualization can save a great deal of time.

10.2 Tables

A spreadsheet or data table is a suitable mechanism to enable comparative data analysis on classified objects. Generally, the row are the category objects while the columns are the things being compared. Then the cell, the location where the row and column intersect, is where the value is shown.

Example: The monthly payments for purchasing or leasing different types of cars (categories) are contrasted in this table. Additional, secondary data is contained in the other columns; the first two are being compared.

	Bank loan monthly payments	Monthly lease payment	Minimum downpayment for lease	Total interest paid over 48 months	Monthly insurance payment
Ford Fusion	552	395	0	2,529	180
Honda Civic	538	424	0	2,466	236
Mazda 3	506	478	1,000	2,318	251
Toyota Yaris	435	490	1,000	1,992	198
VW Golf	596	550	2,500	2,730	244

How to construct a Table-

```
 import pandas as pd
import matplotlib.pyplot as plt
def create_table():
 # Sample data
 data = {
 'Category': ['A', 'B', 'C', 'D'],
 'Value 1': [10, 15, 30, 25],
 'Value 2': [20, 25, 40, 35]
 }
 # Create DataFrame
 df = pd.DataFrame(data)
 # Display table using Matplotlib
 fig, ax = plt.subplots(figsize=(4, 2))
 ax.axis('tight')
 ax.axis('off')
 table = ax.table(cellText=df.values, colLabels=df.columns, cellLoc='center', loc='center')
 # Show the table
 plt.show()
 # Call function to generate table
create_table()
```

output:

Category	Value 1	Value 2
A	10	20
B	15	25
C	30	40
D	25	35

10.2.1 When to use Tables:

1. Focus on how individual, precise values

1. Tables typically provide the best clarity and precision if you are interested in the precise numerical values in your data and wish to correctly reference particular data points.
2. To display individual numbers, a straightforward table works best, particularly when the exact value—rather than a general pattern or trend—is crucial.

Item	Units sold	Unit price (in $)	Status
Sofa	5,421	956.10	In stock
Table	6,407	899.90	In stock
Desk	3,586	651.21	In stock
Chair	2,341	648.30	Sold out
Rug	15,921	234.00	Sold out

---

2. Show ranks

In optimal ways of presenting information, tables are especially useful tools for ranking. Ranking data, allows the viewer to see a simple, ordered list that makes it easy to compare and evaluate. When using a table the viewer can instantly understand how those values rank - from lowest to highest - or vice versa. A table is a great way to present leaderboards or sales ranks or anything else where ranking is helpful information as shown in fig.10.1.

Tables can easily accommodate additional columns and rows to provide context or supplementary data, such as percentage change or categorical groupings. This further enriches the ranking narrative without sacrificing readability.

The top 10 universities in the World University Rankings 2024

All of the top 10 universities are in the United Kingdom GB or United States US

Rank	Name	Location	Overall Score
1	University of Oxford	GB United Kingdom	98.50
2	Stanford University	US United States	98.00
3	Massachusetts Institute of Technology	US United States	97.90
4	Harvard University	US United States	97.80
5	University of Cambridge	GB United Kingdom	97.50
6	Princeton University	US United States	96.90
7	California Institute of Technology	US United States	96.50
8	Imperial College London	GB United Kingdom	95.10
9	University of California, Berkeley	US United States	94.60
10	Yale University	US United States	94.20

Fig. 10.1: Show ranks

3. Make your data set explorable with a search

Make your data easily exploreable by adding a searchable table. With one click you can add a search to your table so your readers can easily find information and narrow down what information they want to see.

Having a search to a table turns a table from linear text (as shown in fig.10.2) and makes it easier to filter results based on queries and actually drill down through different levels of layers of data, which leads to better user engagement and a more customized analysis because you can filter the amount of data and get exactly what you are most interested in.

You can even add emojis such as arrows to make the information easier to digest, color your cells as a cue for if the percentage change direction was positive or negative, or simply add mini bar charts to show a visual trend.

Unemployment rates in the United States in 2022

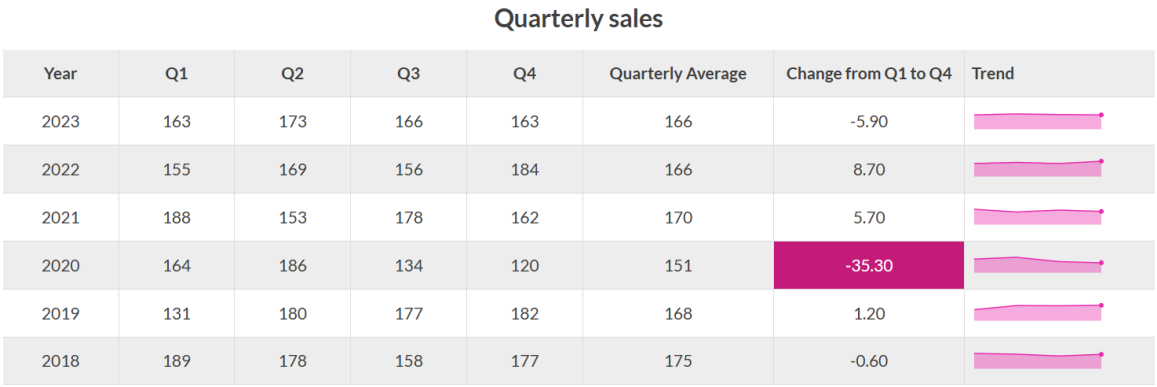
Use the search bar to see how your county fares

Search for a county or state

County Name	Labor Force	Unemployment Rate (%)	Change in pp
Autauga County, AL	26,789	2.3	-0.5
Baldwin County, AL	102,849	2.4	-0.5
Barbour County, AL	8,241	4.1	-1.4
Bibb County, AL	8,726	2.5	-0.9
Blount County, AL	25,796	2.2	-0.2
Bullock County, AL	4,554	2.8	-1.1
Butler County, AL	8,804	3.4	-1.7

Fig.10.2: Searching in show ranks

4. Compare values



Quarterly sales

Year	Q1	Q2	Q3	Q4	Quarterly Average	Change from Q1 to Q4	Trend
2023	163	173	166	163	166	-5.90	
2022	155	169	156	184	166	8.70	
2021	188	153	178	162	170	5.70	
2020	164	186	134	120	151	-35.30	
2019	131	180	177	182	168	1.20	
2018	189	178	158	177	175	-0.60	

Fig. 10.3: Comparing values in show ranks

Tables are great for making comparisons. By presenting each data point side by side, the information is clearly apparent to the user. This makes it easy for them to compare matching values, such as quarterly sales across years. Seeing values side by side makes it much easier to notice both differences and similarities, allowing your reader to quickly analyze upwards and downwards trends, as well as any outliers. In Flourish, you can add mini line charts, or "sparklines," to represent any trend visually. Be sure

to also shade in your cells to distinguish a particular figure, like in the table below, which follows the drop in sales from Q1 to Q2 of 2020 as shown as fig.10.3.

5. Show different metrics

Tables provide a means to visually convey a data set that has multiple values and measurement units without developing more complex visualizations. The linear, organized orientation of a table enhances clarity and minimizes risks of misinterpretation. Using a table enhances the presentation and comparison of even complicated data due to its different and varied measurement units and makes the information readily accessible to your audience. Tables allow you to have separate columns for each unit metric making it simple to compare different types of data and data quality either column to column or row to row whether it is financial data, performance statistics or demographic information all in the same visualization. For example, in the following table(as shown in fig.10.4), we can present and compare several universities by rank, number of full time students, and the share of students who are international students. Moreover, you can add a small stacked bar that illustrates the female-to-male ratio table (in a visual way) and a search bar to explore the data.

World University Rankings 2024

Search for a university or country

Rank	University	Country	FTE students	Students per staff	International students	International Student Share (%)	Female-to-Male Ratio
1	University of Oxford	United Kingdom	21,750	10.9	42	23.00	
2	Stanford University	United States	14,517	6.4	23	22.00	
3	Massachusetts Institute of Technology	United States	11,085	8	33	40.00	
4	Harvard University	United States	20,050	9	25	12.00	
5	University of Cambridge	United Kingdom	20,565	11.5	38	27.00	
6	Princeton University	United States	7,753	7.3	23	43.00	
7	California Institute of Technology	United States	2,240	6.1	33	49.00	

Fig. 10.4: Different metrices in show ranks

Advantages of Data Visualization Using Tables

1. Precise Information

Tables are able to convey precise numerical information, which is why they are extremely well-adapted for contexts where absolute figures are called for. As opposed to graphical presentations, where values can be approximated or scaling applied in a way that distorts one's perception, tables guarantee that all information presented is in its original form. Such precision is vital in cases like financial accounting, scientific investigation, and health data analysis, where even the slightest inaccuracies can result in gross misinterpretations.

2. Easy to Read for Small Data Sets

When there is little data, tables provide an easy and straightforward method of presenting information. In contrast to intricate visualizations that may need interpretation, tables enable users to view the raw numbers directly. This makes them especially handy for situations like brief reports, meeting minutes, or fast reference papers where simplicity and clarity are essential.

3. Supports Comparisons

Perhaps the largest benefit of tables is that they can represent multiple values next to each other, enabling users to make direct comparisons. In examining financial results between intervals of time, for example, or comparing the impact of various marketing efforts, tables enable users to compare differences and similarities quickly. This is what makes them indispensable in business decision-making, research, and applications based on data.

4. Flexibility in Data Presentation

Tables offer several formats for organizing and engaging with information, such as sorting, filtering, and segmenting data. Users can customize rows and columns according to various criteria, thereby facilitating detailed analyses. This versatility is particularly advantageous in database administration and software where organized data handling is paramount.

5. Supports Merging of Data Types

In contrast to graphical formats like charts, which are mainly used for numeric data, tables support the easy merging of both qualitative and quantitative data. For example, a financial report can contain numeric sales figures along with qualitative customer comments, giving a better overall view of business performance. This ability makes tables indispensable in situations that demand a combination of textual and numeric information.

Disadvantages of Using Tables to Visualize Data

1. Lack of Good Representation of Large Data Sets

Although tables are perfect for small data sets, they are cumbersome and time-wasting when handling big data sets. Users might not be able to draw out meaningful insights in a timely manner since navigating through many rows and columns is time-consuming. In such a situation, visual instruments such as charts or dashboards offer a more intuitive way of consuming large volumes of information.

2. Difficult to Identify Trends

While graphs or charts easily point out trends, patterns, and outliers, tables need thorough scrutiny to identify insightful information. For example, a search for seasonal variations in sales through a table involves a lot of manual input, while a line graph would at once indicate rising or falling trends. This is a drawback that reduces the efficacy of tables when it comes to trend analysis or bird's eye view.

3. Less Engaging

Tables are generally seen as dry and uninteresting compared to graphical presentations. While infographics and charts employ colors, shapes, and design to make reading easier and draw attention, tables only depend on numerical or textual information. Therefore, audiences tend to find tabular presentations less engaging, lowering comprehension and interest, particularly in environments such as reports and presentations.

4. May Be Confusing

When a table has too many rows and columns, it is easy to become confused and get lost. It can be challenging for users to locate important data or match information in different parts of the table, which could result in mistakes or misreadings. This problem is quite prevalent in rich datasets, for instance, thorough financial reports or large medical histories, where an abundance of data might make meaningful analysis difficult.

5. Constrained in Presentation

Tables mainly present information in a formal, row-and-column layout, which restricts them from highlighting important findings visually. In contrast to charts that portray most important points through trend-colored cues or proportional displays, tables necessitate users to look for important figures or conclusions. That can be a letdown when trying to present high-level summaries or making data storytelling more compelling.

Applications of Tables

1. Financial Reporting and Analysis

Tables are essential in financial reports like balance sheets, income statements, and budget reports. They give a precise picture of revenues, expenses, profits, and losses, enabling accountants, investors, and analysts to analyze financial performance with accuracy. Because of their precision and organized structure, tables are essential in auditing, forecasting, and financial decision-making.

2. Database Management

Tables in relational databases form the backbone of structuring and storing data. Every row is a single record, while columns are particular attributes, and this allows efficient retrieval and management of data. Databases such as MySQL, PostgreSQL, and Microsoft SQL Server are largely dependent on tabular structures to provide complex queries, relationships, and reporting capabilities.

3. Scientific Research

Scientists employ tables heavily to report experimental data, statistical observations, and comparative information. To report clinical trials, laboratory tests, or survey reports, tables provide a structured mechanism for presenting information. Their capacity to clearly represent numerical values makes them a necessity in peer-reviewed journals and academic papers.

4. Business and Sales Analysis

Organizations utilize tables to monitor key performance indicators (KPIs), sales, and customer demographics. By organizing business-related information in tabular form, companies are able to evaluate market trends, gauge profitability, and make informed decisions. Sales reports, inventory management systems, and supply chain analyses all take advantage of the clarity and precision that tables offer.

5. Healthcare and Medical Records

The health care sector depends on tables in order to keep and show important patient information, such as medical histories, treatment protocols, and medication records. Tables allow doctors, nurses, and health care administrators to find correct patient information speedily, thus enabling effective diagnosis and treatment. Electronic health records (EHRs) employ tabular arrangements to organize patient data logically.

6. Education and Academic Performance Monitoring

Tables are employed by schools, universities, and educational institutions to track student grades, attendance, and performance indicators. Teachers and administrators can quickly compare student progress in several subjects or evaluate overall class performance. Report cards, examination scores, and enrollment figures often employ tabular structures to display educational information in a structured format.

10.3 Graphs

Graph visualizations represent connected data visually or in a way that displays each entity/point with the connections between them. It can be a visual representation of the literature, or any number of topics - either like connections as a graph and/or by using color to represent innovation, type, or topic. Graph visualizations can also be called network visualization or link analysis visualizations. A graph visualization is displayed as a network, in which each point of data is connected to another point of data with a link that represents how they are connected. A network visualization can represent any number of things: how goods move from one place to another in a supply chain, how plugs parts of an IT system together, or the transactions between accounts.

10.3.1 Importance of Graphs in Data Visualization

1. Simplifies Complex Data – Graphs make it easier to comprehend large and complex datasets by summarizing them visually.
2. Enhances Decision Making – Visual representations help stakeholders and analysts make informed decisions based on trends and insights.
3. Identifies Patterns and Trends – Graphs reveal patterns that may not be evident in raw data, such as seasonality in sales or growth trends.
4. Improves Data Communication – Graphs enable effective communication of insights to different audiences, including non-technical users.
5. Supports Predictive Analysis – By visualizing historical data, graphs help predict future trends and behaviors.
6. Highlights Outliers and Anomalies – Unusual patterns or deviations in data can be easily spotted through visual representation.

10.3.2 Common Types of Graphs:

Line graph: The x-axis and y-axis in fig.10.5 show example of how a line graph compares two variables visually. 5. It depicts the data by connecting all the x, y coordinates on a graph with a single line. There are three different types of line graphs.

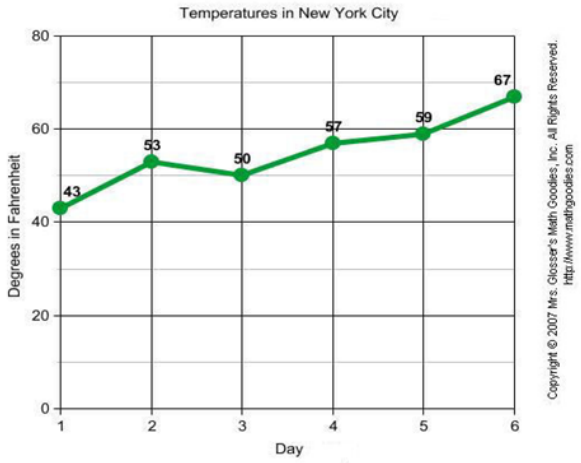


Fig. 10.5: Line Graph

They include:

1. Line graph- a line is presented on the graph.
2. More than one line on the same set of axes: the multiple line graph .This makes it useful for comparing similar properties in the same time frame.

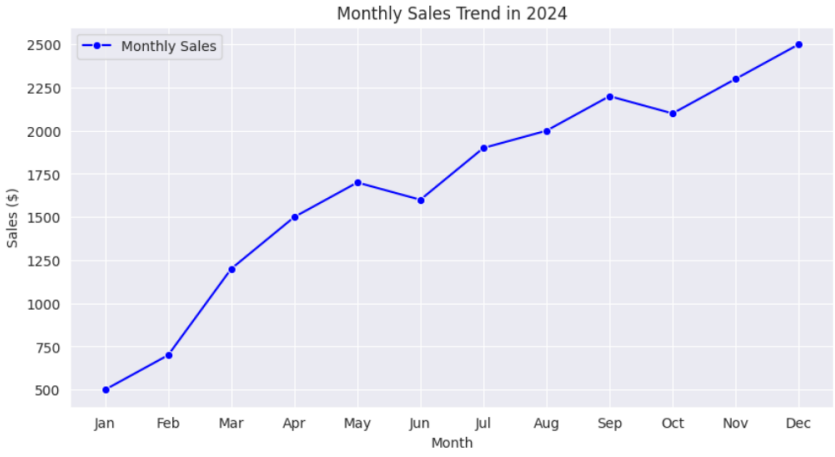
3. Another type is the compound line graph: if we can split the data into two types or more.

The important point is that a compound line graph has lines drawn from the total to show what the component part of the total is. The line above is the total and the line below is part of that total. The space between two lines shows how big each piece is.

How to construct a line graph:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# Sample dataset: Monthly Sales Data
data = {
 "Month": ["Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"],
 "Sales": [500, 700, 1200, 1500, 1700, 1600, 1900, 2000, 2200, 2100, 2300, 2500]
}
# Convert to DataFrame
df = pd.DataFrame(data)
# Set Seaborn style
sns.set_style("darkgrid")
# Create a line graph
plt.figure(figsize=(10,5))
sns.lineplot(x="Month", y="Sales", data=df, marker='o', color='b', label="Monthly Sales")
# Add labels and title
plt.xlabel("Month")
plt.ylabel("Sales ($)")
plt.title("Monthly Sales Trend in 2024")
plt.legend()
plt.show()
```

Output:



Bar Graph: Bar graphs compare different categories with each other. It may be made up of two or more vertical (or horizontal) bars (rectangles) parallel to each other. And will be used to compare 2 or more values in a smalL set of results Bar Graph Types. Enumerating Categories Using Bar Graph. As shown in

fig.10, it can have two or more vertical (or horizontal) bars (rectangles) parallel to one another. 6 A bar chart will help us to correlate two or more value with a small result set. Bar graphs are of two types:

a. A vertical Bar Graph

b. A horizontal Bar Graph

The vertical bar depends on the vertical axis according to the choice of the baseline while the horizontal bar is set on the horizontal axis. Below given bar graph to compare a student's marks to indicate the information required.

Using bar graphs makes it possible to compare students' scores across subjects and also to display the score of one student in each subject

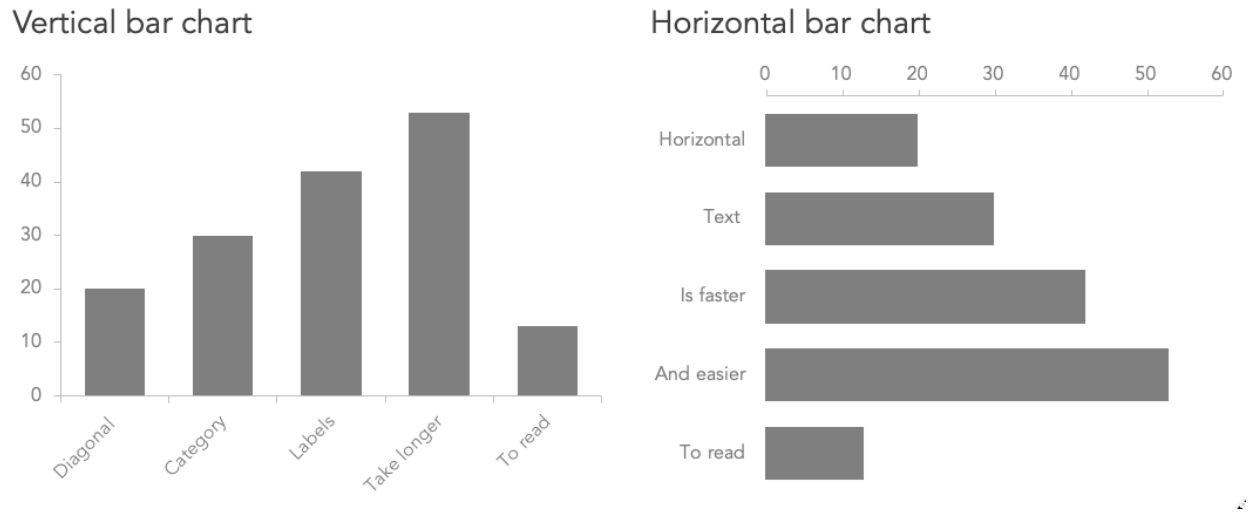


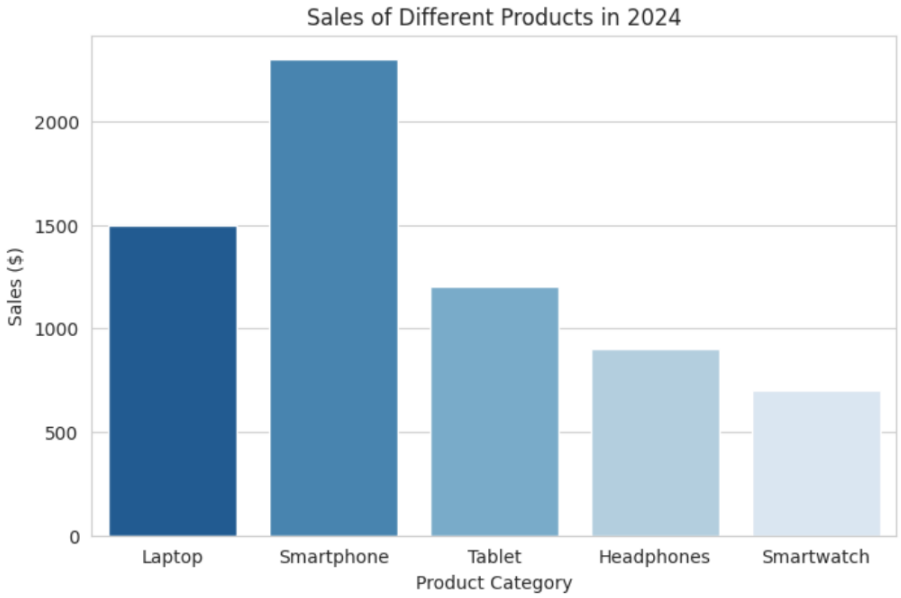
Fig.10.6: Bar charts

How to construct a Bar graph:

```
# Import necessary libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# Sample dataset: Sales of different products
data = {
 "Product": ["Laptop", "Smartphone", "Tablet", "Headphones", "Smartwatch"],
 "Sales": [1500, 2300, 1200, 900, 700]
}
# Convert to DataFrame
df = pd.DataFrame(data)
# Set Seaborn style
sns.set_style("whitegrid")
# Create a bar graph
plt.figure(figsize=(8,5))
sns.barplot(x="Product", y="Sales", data=df, palette="Blues_r")
# Add labels and title
plt.xlabel("Product Category")
```

```
plt.ylabel("Sales ($)")
plt.title("Sales of Different Products in 2024")
# Show the plot
plt.show()
```

Output:



Benefits of Applying Graphs in Data Visualization

1. Simple Interpretation

Graphs take complex information and make it simpler to determine relations, comparisons, and trends. Through visualizing data, they eliminate the necessity to study large quantities of numbers, and users are able to quickly comprehend insights. For example, a line graph is able to immediately present the fluctuations in stock prices and indicate how prices shift with time.

2. Pattern Identification

One of the most significant strengths of graphs is their capacity to point out patterns, correlations, and anomalies in data sets. Trends that may be hard to detect in tables immediately stand out in a graphical presentation. For instance, a scatter plot can identify whether two variables are positively or negatively correlated, helping researchers and analysts determine key relationships.

3. Comparative Analysis

Graphs enable direct comparison among various categories, data points, or time frames. In contrast to tables where one has to go through them carefully in order to compare values, graphs depict these relationships visually making it simple to notice differences and similarities. Bar charts, for instance, are very useful in comparing sales quantities across a number of regions, products, or time periods.

4. Summarization of Big Data

Graphs summarize large sets of data into usable visualizations, allowing the user to focus on major findings without drowning in raw data. This is especially useful for analytics and business intelligence, where managers require instant clear views of performance metrics without the need to read through large spreadsheets.

5. Decision-Making Support

Companies, scientists, and policy-makers use graphs to offer visual insights that drive key decisions. From projecting future sales, studying economic patterns, or monitoring the success of a marketing campaign, graphical data presentation ensures decision-making is supported by concise, evidence-based facts.

Drawbacks of Utilizing Graphs in Data Visualization

1. Misinterpretation Risk

Imprecisely labeled, improperly scaled, or deceptive graphs are likely to give rise to improper conclusions. An instance is the graph that maximizes differences using an inconsistent scale, thereby skewing the perception of trends or data relationships. Clarity and accuracy in designing graphs are fundamental to prevent misinterpretation.

2. Loss of Detail

While graphs are excellent for summarizing data, they sometimes oversimplify information, omitting crucial details. A pie chart, for instance, may show the proportion of sales from different regions but may not provide deeper insights into the reasons behind those sales variations. This limitation can be a drawback in contexts requiring detailed analysis.

3. Data Overload

When a graph contains too many data points or categories, it may become crowded and hard to read. Overloaded graphs tend to confuse the audience instead of explaining the data, and it becomes hard to derive meaningful insights. This is prevalent in line graphs with too many trend lines or bar charts with numerous bars, making them less effective.

4. Needs Correct Choice

An incorrect choice of graph for a dataset can cause misinterpretation of the data. For instance, applying a pie chart to data that is to be trended is ineffective, since pie charts are only capable of representing proportions and not trends over time. The right visualization technique must be chosen in order to correctly convey the intended message.

5. Tool Dependence

Making quality and good-looking graphs usually involves specialized software or programming expertise. Although simple graphs can be produced with spreadsheet software, more sophisticated visualizations could necessitate some level of expertise in data visualization tools like Tableau, Power BI, or even programming languages like Python and R. This tool dependence could be a hindrance for users who are not technically equipped.

Applications of Graphs

1. Business and Economics

Graphs are extensively applied in business and economic analysis to monitor financial trends, sales performance, and market fluctuations. For instance, firms utilize line graphs to observe revenue growth over time and bar charts to compare yearly profits among various branches. Economic analysts use trend graphs to predict inflation rates, unemployment levels, and stock market movements.

2. Science and Engineering

In science and engineering, graphs are used to visualize experimental data, measurements, and statistics. Engineers use graphs to analyze stress-strain behavior in materials, and scientists graph

temperature fluctuations, chemical reaction rates, and other important results. Graphs are also used to model complicated systems so researchers can display their data in a readily understandable manner.

3. Healthcare and Medicine

Graphs are very important in healthcare as they help in monitoring patients, epidemiology research, and medical studies. For instance, line graphs monitor a patient's heart rate or blood pressure over time, assisting physicians in making informed decisions. Epidemiologists apply graphs to analyze disease transmission patterns, while pharmaceutical researchers graphically represent the efficacy of new medications.

4. Education and Research

Within educational environments, graphs are utilized to display performance measures, survey data, and research results. Educators use bar graphs to compare test scores between students and courses, while researchers use visualizations to report experimental results. Graphs are also used within educational dashboards to monitor student progress and areas of improvement.

5. Social Sciences

Graphs assist in interpreting population data, trends in social behavior, and responses to surveys. Demographers make use of population pyramids to study age structures, whereas sociologists employ graphs to monitor trends in behaviors, such as voting or employment patterns. Pie charts and bar graphs are frequently used in surveys to display responses in a readily comprehensible format.

6. Technology and AI

In technology and artificial intelligence, graphs play a crucial role in visualizing machine learning model data, AI predictions, and algorithm performance analysis. Confusion matrices and precision-recall curves help measure model accuracy, while developers use various graphing techniques to visualize network traffic, cybersecurity incidents, and system performance. AI research heavily relies on graphs to understand model behavior and make data-driven decisions.

10.4 Histograms

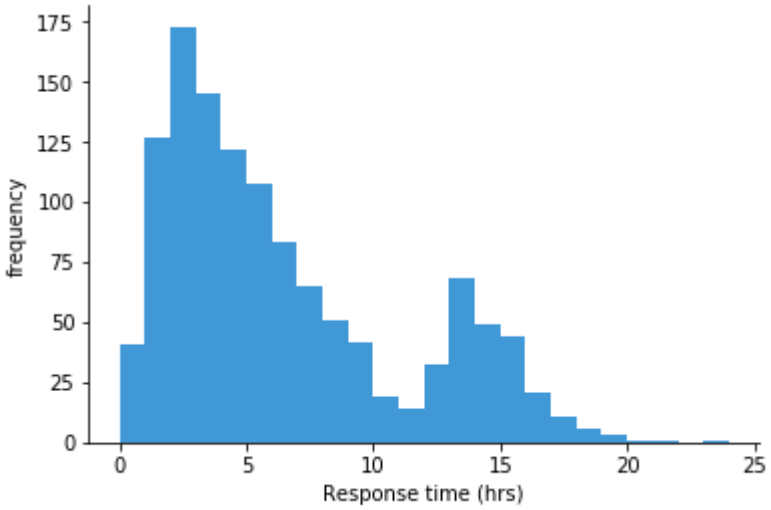


Fig. 10.7: Histogram

The distribution of a dataset can be shown by a histogram when the data is organized into bins. This is particularly useful to visually discern the frequency distribution of a set of numerical values. A

histogram shares some similarities with a bar graph, although it is considered more suitable for representing the distribution of continuous data.

In a histogram:

- Data is represented in intervals or bins
- Each bar's height corresponds to the frequency (or count) of data points within that interval

This is visualized in Figure 10.7.

How to construct a Histogram

```
import numpy as np
import matplotlib.pyplot as plt
data = np.random.randn(1000)
plt.hist(data, bins=30, color='blue', alpha=0.7)
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.title('Histogram Example')
plt.show()
```

Output:

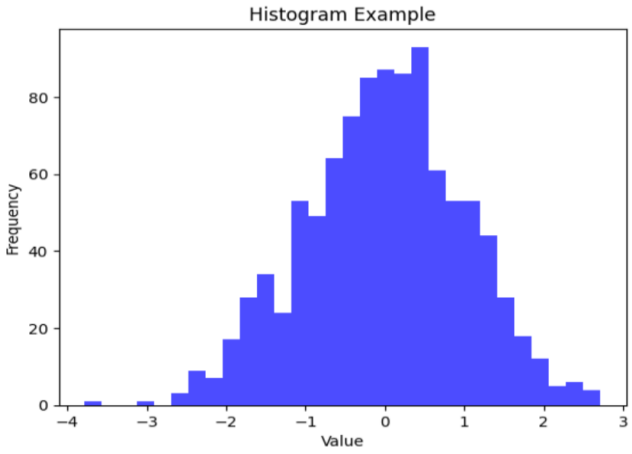


Fig. 10.8: Example Histogram

10.4.1 When you should use a histogram

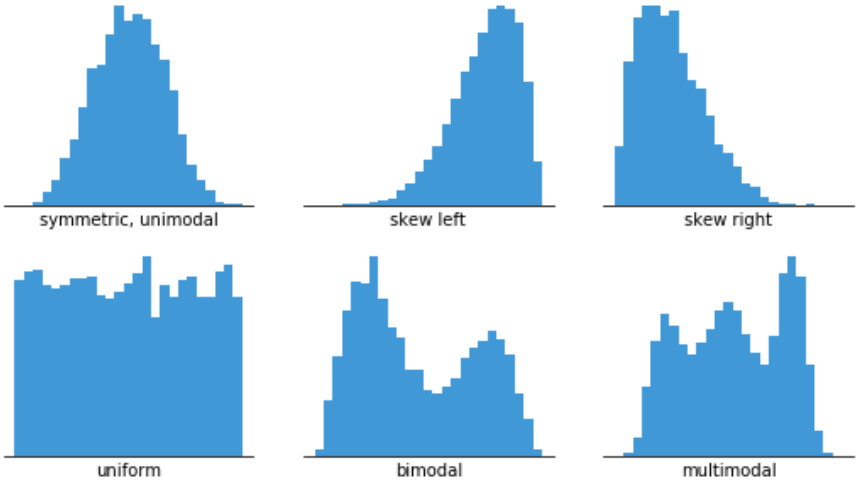


Fig. 10.9: Types of Histogram

Histograms are a good method for examining overall distributional properties of variables in a dataset. You can identify roughly the locations of the peaks in the distribution, if the distribution is skewed or symmetric, and whether there are any outliers.

There are some advantages in using the histogram compared to other representations. To employ a histogram it is sufficient to have a single variable that can take continuous numeric values. This indicates that a difference between values is equal regardless of its absolute value, meaning that the difference between a score of 65 and 60 is the same size of a difference as the difference between a score of 95 and 90. For example, a score may take on the integer values of 0-100 if measuring a test score, therefore we have the convenience of counting to observe comparison levels as being of equal distance either at the bottom or the top. There is no inherent information in the data itself about the number of bins created to job up the data points as being called bin and the inherent boundaries for that bin. Assigning boundaries (bins) is an independent decision for building up the histogram. Clearly how we create a bin will matter a lot in how the histogram can be visualized and identified as we begin to work with them.It is stated that in order to build up a histogram, when there exists a mode of the observation on a boundary of a bin the user is free to locate the boundary on the left or right (or it could go in to or out of the end bins). This use depends of course on the visualization tool; some do not allow you as the user to override their assumption. This source will assume.

10.4.2 Best practices for using a histogram

b. Use a zero-valued baseline

One of the key characteristics of the histogram is that the baseline from which the bars are drawn has to be zero as shown in fig.10.10. The height of each bar indicates the frequency of each bin of data, so a change in baseline, or a discontinuity (a gap in the scale) will misrepresent the distribution of data.

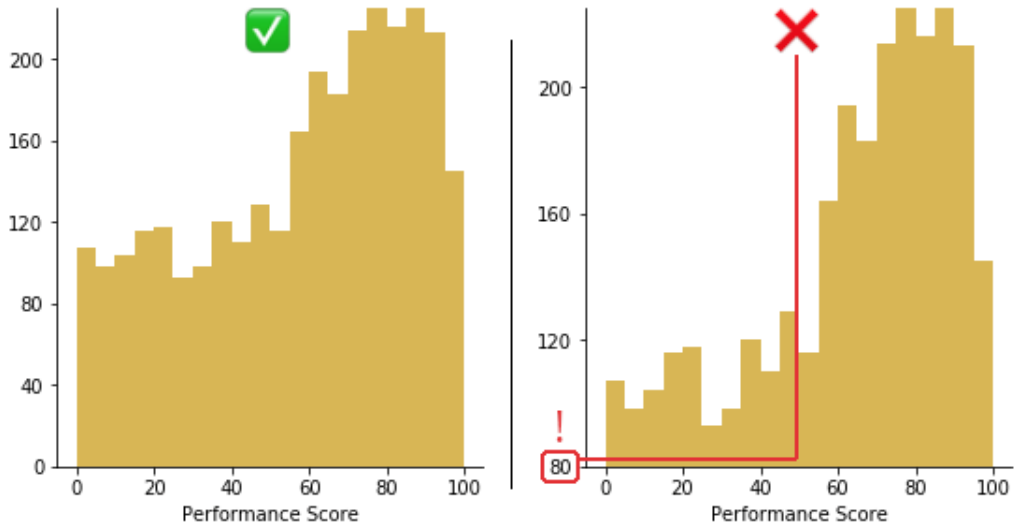


Fig. 10.10: Zero-Valued baseline

c. Number of bins

Although tools that generate histograms, which are often really helpful, have some built-in algorithms in order to select bin boundaries, you will likely want to mess around with the parameters to choose something that is more representative of your data. Wikipedia has a large section that discusses rules

of thumb to choose an appropriate number of bins, and sizes of those bins, but you are best off using knowledge of your domain, along with some messing around using different options. Bin size has an inverse relationship with the number of bins. The larger the bin size, the fewer bins there will be, and the smaller the bins, the more bins there will be. It is a worthy use of some time to test bin sizes, look at the distribution you see in each bin, and choose a final representation that makes sense. You can have too many bins, and the data will look rough, the signal will be hard to find in the noise, and that should be avoided. You can have too few bins, and then the histogram will not have fine enough detail to be very useful (as shown in fig.10.11).

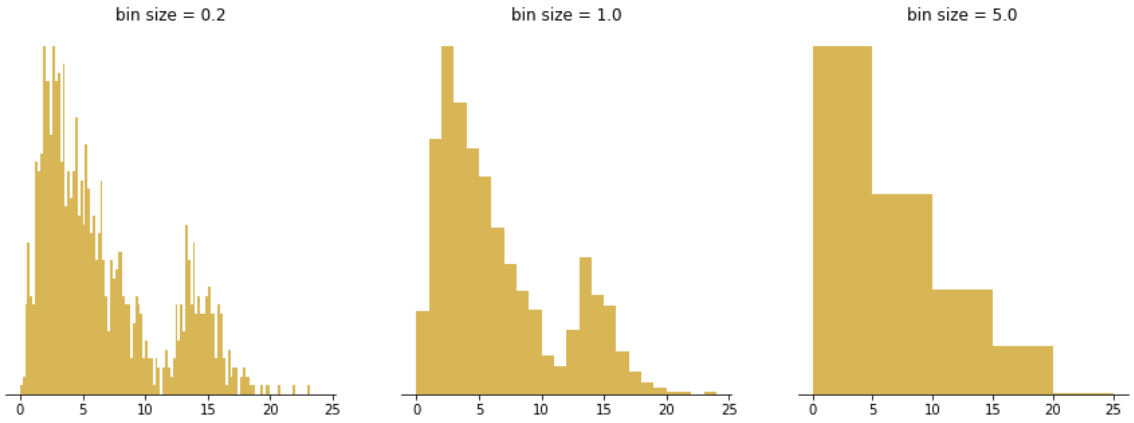


Fig. 10.11: Bin Sizing

d. Choose interpretable bin boundaries

The tick marks and labels usually should fall on the bin boundaries to more clearly signify where each bar ends. While labels are not necessary for each bar, having labels include every few bars will help keep the reader aware of the value being represented. Additionally, it is best to use labels with only a few significant digits so they are easy to read and understand. This indicates that bins with a size of 1, 2, 2.5, 4, or 5 (which nicely divide into 5, 10, and 20) or their powers of ten are good size bins to use as a general guideline. Furthermore, it also would indicate that bins with a size of 3, 7, or 9 are going to be more difficult to read and should not be used unless there is a specific reason for them(as shown in fig.10.12).

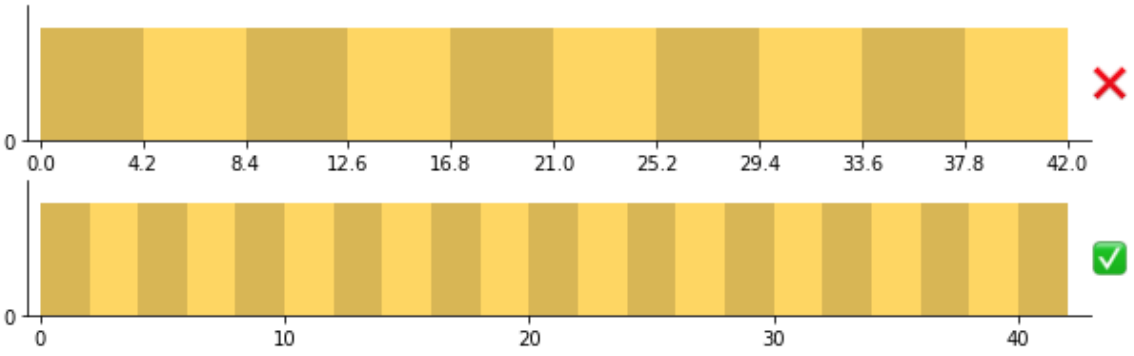


Fig. 10.12: Selection of Bin Boudaries

Benefits of using Histogram

• **Simple to comprehend:** Using a histogram gives a visual representation of the distribution of data so that the impression of it can be captured.

• **Unearth Patterns:** A histogram displays the data in a way that patterns and trends can potentially recognized, such as skewness, peaks, or gaps.

• **Compare Data Sets:** Through the use of a histogram, two different data sets can be compared to highlight similarities or differences in data set distributions.

Disadvantages of using Histogram

• **Not for small data sets:** A histogram may not be appropriate for very small data sets, as it requires a large enough sample size and amount of data for there to be a visual representation of a true distribution.

• **Details that are lacking:** Because histograms are built from the nature of being summarized, we can see the distribution of the data but may not see individual data points including their values and outliers.

Applications of Histograms

Histograms are widely used in various domains to analyze frequency distributions and identify patterns in data. Common applications include:

- **Statistical Analysis**: Used to understand the distribution of data, detect skewness, and identify outliers.
- **Quality Control and Manufacturing**: Helps in monitoring product consistency and detecting defects in production lines.
- **Finance and Investment**: Used for analyzing stock market trends, returns distribution, and risk assessment.
- **Healthcare and Epidemiology**: Helps in studying patient data, such as age distribution or disease prevalence.
- **Education and Testing**: Used for analyzing student performance and exam score distributions.
- **Marketing and Customer Analytics**: Helps in segmenting customers based on purchasing behavior or demographics.

10.5 Pie Charts

A pie chart is a circle that is segmented into parts displaying proportions of the total data. Each segment is representative of a percentage of the whole. Pie charts are circular graphs that are divided into sectors or segments, allowing for the visual representation of the whole as shown in fig.10.13. Slices of pie charts are representative of some proportion of the data, as each sector correlates with some percentage or proportion of the total data. Pie charts are useful for illustrating the composition of the whole and for comparing different categories as parts of a whole.

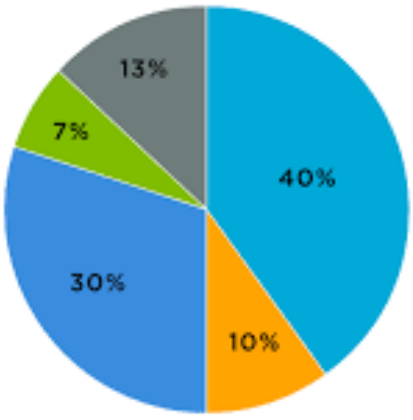


Fig. 10.13: Pie Chart

Constructing a Pie chart

```
labels = ['A', 'B', 'C', 'D']
sizes = [30, 20, 25, 25]
colors = ['blue', 'green', 'red', 'purple']
plt.pie(sizes, labels=labels, colors=colors, autopct='%1.1f%%')
plt.title('Pie Chart Example')
plt.show()
```

Output

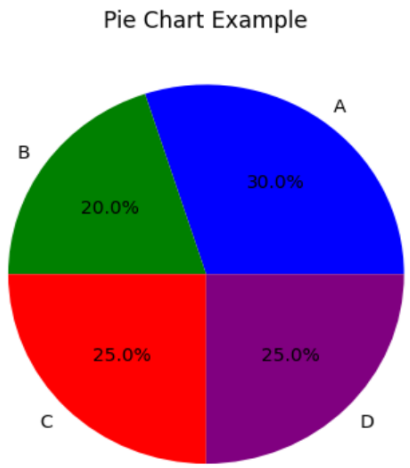


Fig. 10.14: Pie Chart Example

10.5.1 Usage of pie chart

The pie chart can only be used in very limited situations, which is precisely what the definition captures in a pie chart. To use a pie chart, there must be some total of a whole that can then be split into a number of distinct groups.

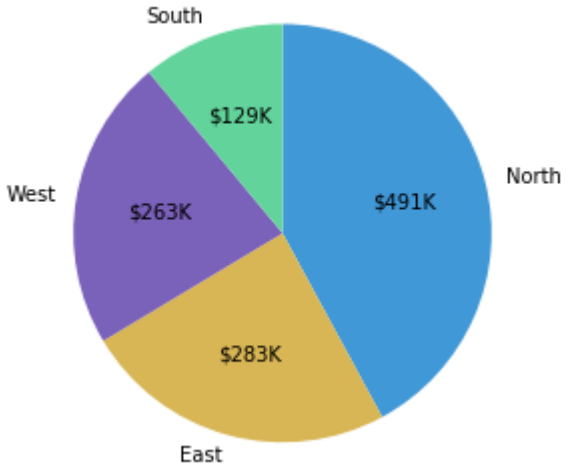


Fig. 10.15: Use of Pie Chart

The main thing your pie chart should accomplish is tell the user how much of the whole each group represents, as against to comparing different groups with each other. If the two statements above are not met, then there is no need to use a pie chart at all and should select a different plot type. The values that comprise a whole and the groups that subdivide the whole can generally be divided into two types of

groups. The first is when the whole has a total count. Examples here might be the votes in an election split by candidate, or number of transactions split by user type (e.g. guest, new user, existing user). The second type of whole, is when the whole has a total that is summed over an actual data variable. For example, we might be interested in not the number of transactions, but the monetary total (value) from all transactions etc. Again, dividing this by some attribute might reveal interesting insights as to where the business is most successful.

10.5.2 Best practices for using a pie charta. Include annotations

Determining precise proportions from a pie chart is indeed challenging and typically limited to small fractions such as 1/2 (50%), 1/3 (33%), or 1/4 (25%). Furthermore, if the slice values are intended to reflect amounts, the tick marks are often lacking to make a value from the size of slice. This is why annotations of some kind are included in pie charts.

b. Consider the order of slices

When it comes to ordering the slices, having a good sequence can really help a reader follow along with the meaning of the plot. The common ordering of the slices is from largest to smallest slice, which is particularly important for cases where slices are similar in value. If the levels of the categories include meaningful ordering, then the best practice is to plot the slices in that order. In terms of where to start, it is a good practice to plot the slices from what might be considered a cardinal orientation. Visualization tools will typically start to from the upper left or from the top right. Cardinal orientation to the right has even been suggested because when working in distance, there is a mathematical reason to start that direction as it is a misconception of conventional angel view. However, using an upper approach to plot of the slices is more in line with the human instinct of reading from the top to the bottom and from left to right and is similar to thinking about time of the progression of time without a clock or watch(as shown in fig.10.16).

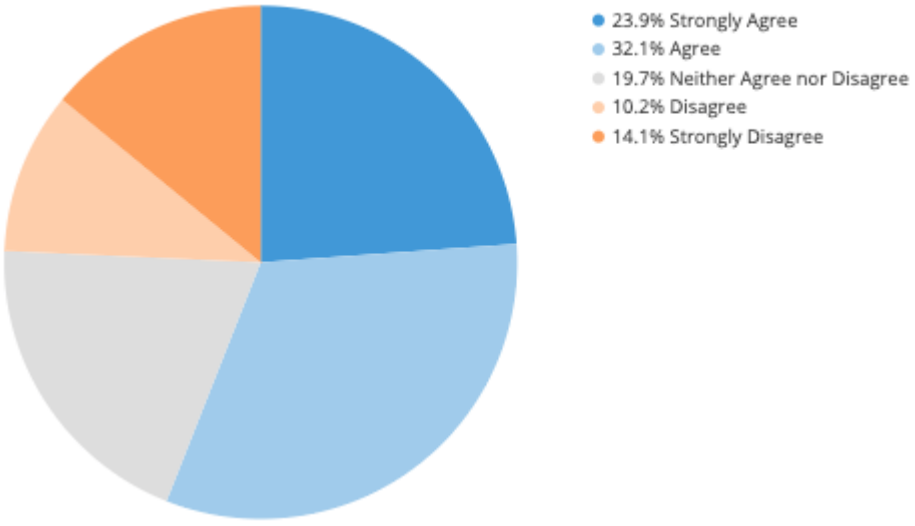


Fig. 10.16: Order of Slices

We do not sort by size here since the labels are meaningful.

c. Limiting the pie slices

It can be hard to read pie charts when the chart has lots of slices. It is difficult to see and analysis small size slices, and it may be difficult to have enough distinguishable colors for each slice. There are varying recommendations, but if there are more than about five categories, you probably would

want to consider a different chart type as shown in fig.10.17. Alternatively, you might consider putting the smallest slices into an 'other' slice colored in a neutral gray.

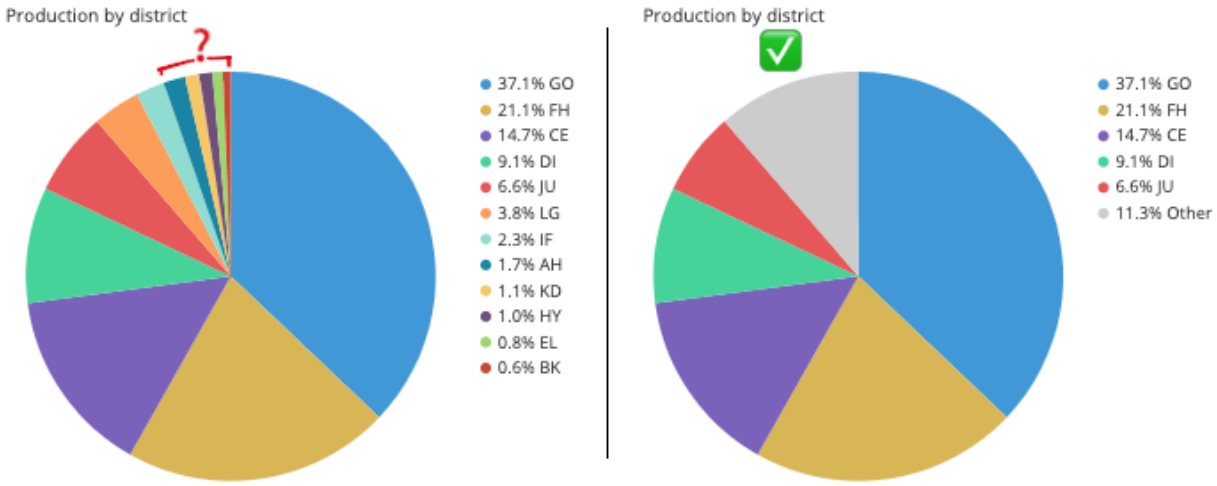


Fig. 10.17: Variation of pie chart according to number of slices

a. Distorting effects in pie chart

To read a pie chart correctly, the areas, arc lengths, and angles of the slices should all represent the data accurately. Generally speaking, it's a good idea to avoid using a 3-d effect for any plot, but it's particularly important with pie charts. If the circle is squashed or stretched or if any depth is added that is not necessary, it can distort the reader's interpretation of how large each slice is compared to the total as shown in fig.10.18. Another sort of distortion is the exploded pie chart, where the slices look like they are simply pulled out from the center to emphasize them. While this does have a visual emphasis, there is probably a cost to this approach, where the gaps are unclear and create a more difficult part-to-whole comparison.

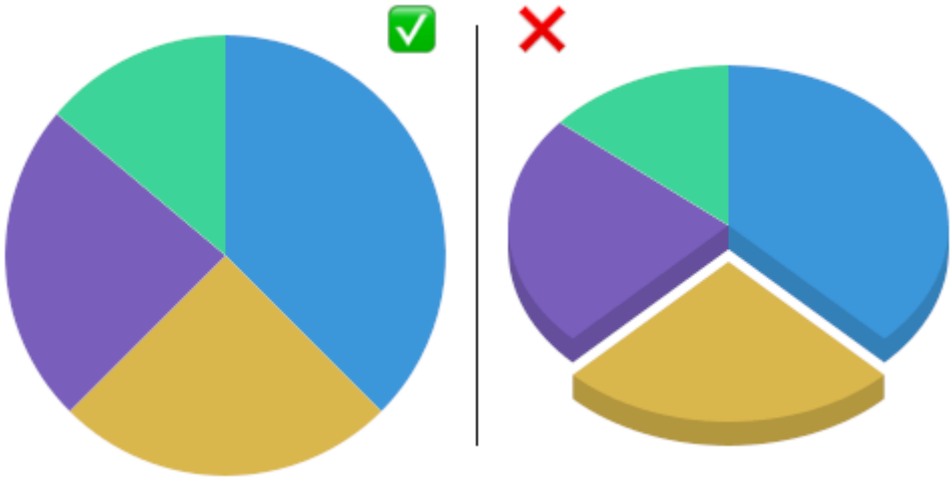


Fig. 10.18: distorting effect in pie chart

Benefits of Pie Graphs

• **Quick to create:** Pie graphs or charts are easy to produce using many different types of software or even by hand; thus, they are accessible to everyone for visualizing data and require no special expertise or knowledge.

• **Visually pleasing:** The circle and color are visually appealing, draw viewers' eyes, and make the data more inviting to read.

• **Simple and easy to read:** Of all the data charts, pie graphs provide the most information in a straightforward way, allowing the viewer to read and quickly gauge relative proportions**.**

Downsides of Using a Pie Chart

• **No trends:** Pie charts do not show data trends or data changes because the chart only shows a snapshot of data at that point in time.

• **Too many data slices**: As more categories are entered into a pie chart, the less effective they become, as the audience may not be able to distinguish or accurately read smaller slices. Pie graphs are best when there are only a few categories of data to represent with meaningful differences in proportions.

Applications of Pie Charts

Pie charts are widely used across various domains to represent proportional data. Some key applications include:

- **Business and Marketing**: Used to display market share distribution, sales composition, and revenue breakdowns.
- **Financial Analysis**: Helps in visualizing budget allocation, expenditure categories, and investment portfolio distribution.
- **Demographics and Statistics**: Commonly used to represent population distributions by age, gender, or ethnicity.
- **Survey Results**: Helps in summarizing responses to categorical survey questions, such as customer satisfaction levels.
- **Education and Research**: Used to showcase student enrollment distribution across different courses or departments.
- **Healthcare and Epidemiology**: Helps in analyzing the proportion of disease cases, medication usage, or hospital admissions by category.

10.6 Area Plots

Area charts are kind of line graphs but with the space below the line shaded in color. Area charts are used to illustrate cumulative totals or stacked data over time. They are great at illustrating changes in composition over time and contrasting the contributions of various categories to the total.

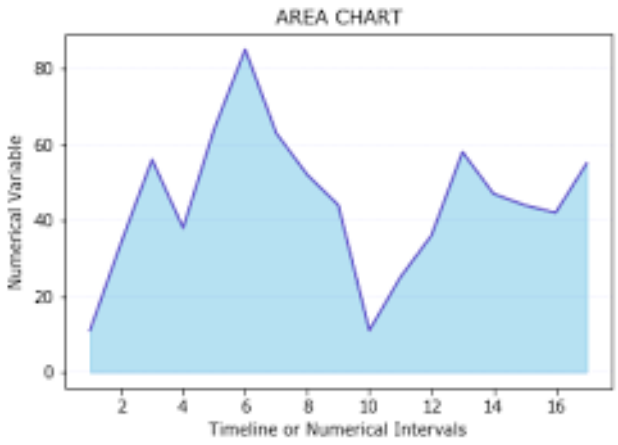


Fig. 10.19: Area Chart

10.6.1 How to construct an Area chart

```
import matplotlib.pyplot as plt
import numpy as np
years = np.arange(2010, 2021, 1)
data1 = np.array([5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55])
data2 = np.array([3, 7, 12, 18, 22, 27, 30, 36, 41, 47, 52])
plt.fill_between(years, data1, color='skyblue', alpha=0.5, label='Category A')
plt.fill_between(years, data2, color='orange', alpha=0.5, label='Category B')
plt.legend()
plt.title('Area Plot Example')
plt.xlabel('Year')
plt.ylabel('Value')
plt.show()
```

Output

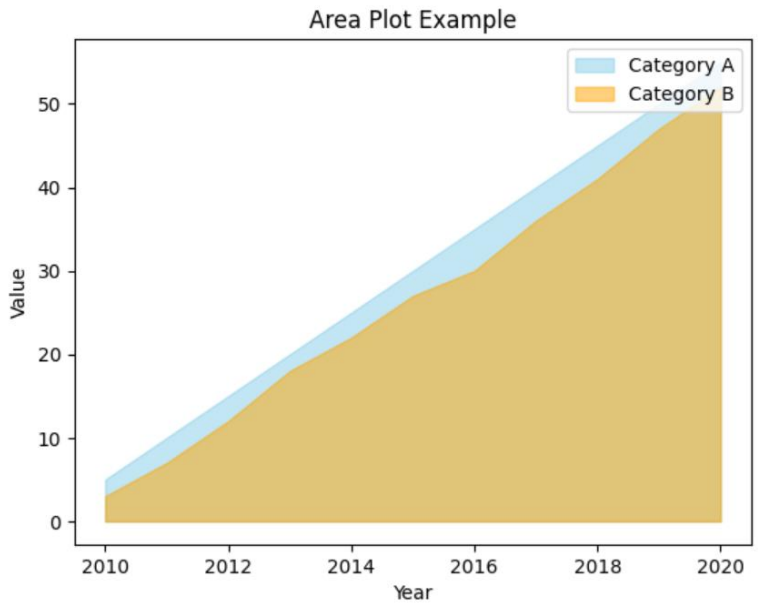


Fig. 10.20: Area plot example

10.6.2 Usage of area chart

While the above example just plots one line with filled area, an area chart is often used with two or more lines to compare a group (otherwise known as a series) against each other, or to compare how a sum is made up of constituent parts. This means there are two different types of area chart, one for the former and one for the latter.

1. Overlapping area chart

If we want to compare the values across groups, we can use an area chart with overlapping areas. In an overlapping area chart, we start off from a simple line Chart. For each group, one point is plotted for every horizontal value, with height indicating the value of that group on the vertical axis variable; a line connects all points of a single group from left to right. Up a notch, the area chart fills between each line and a zero baseline with shading.Since the shading for groups will usually overlap to some extent, some transparency is included in the shading so that all groups' lines can be easily seen. The shading helps to

emphasize which group has the largest value based on which group's pure color is visible(as shown in fig.10.21).

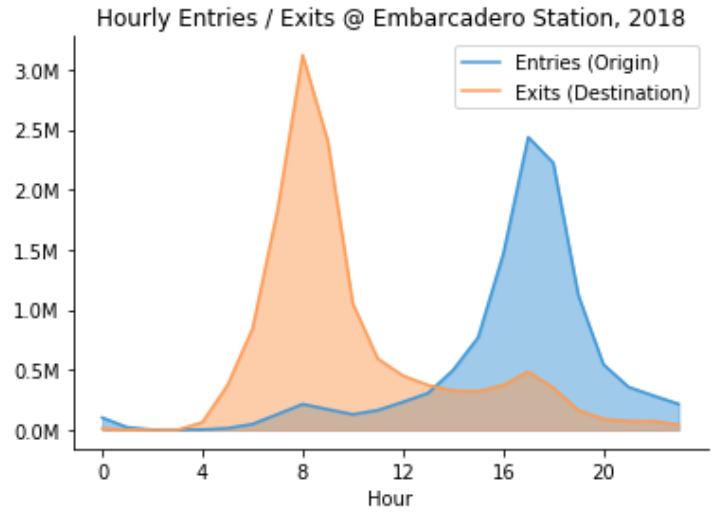


Fig. 10.21: overlapping area chart

Be careful that one series is not always higher than the other, or the plot may become confused with the other type of area chart: the stacked area chart. In those cases, just keeping to the standard line chart will be a better choice.

2. Stacked area chart

Generally, when referring to an area chart, the stacked area chart is implied. In the area chart that overlapped, each line was filled in from its vertical value to a single baseline. The stacked area chart plots each group on top of the previous one, with the height of the previously plotted group serving as a moving baseline, as illustrated in fig.10. 22. So, the total-stacked height of the top line, will be equal to the total when summed across all groups.

Stacked area charts are used when you need to monitor not just the cumulative value, but also need to know the composition of that total by the groups. By comparing the heights of each segment of the curve in area chart , we are able to make a general sense of how each subgroup relates to the other in contributing to the overall total.

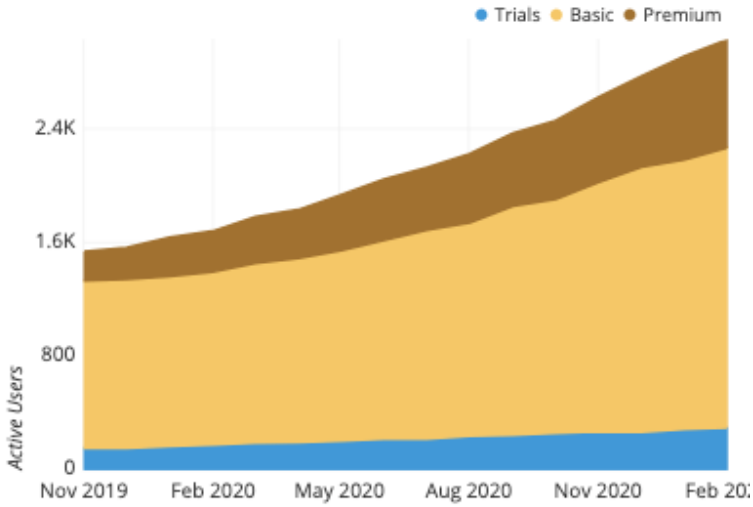


Fig. 10.22: Stacked area chart

10.6.3 Ideal practices for using an area chart

1. Include a zero-baseline

A zero-baseline refers to having a shading means that the values of the colors will be used to contrast how big each group's values are. Therefore, similar to a bar chart, it is necessary for there to be a zero-baseline against each input which the shading must be done. Cutting the axis would cause the real ratio in group values not to coincide with what is suggested by the created plot.

A deviation from this rule is possible when comparing two series in an overlapping area chart with a modification to the shading rule. If we restrict the shading to between lines, instead of from both lines to some common baseline, then we are able to zoom the vertical axis limits into the effect we are interested in without a baseline. Shading is now interpreted differently, with the shade representing which group as a larger value, and the degree of color the size of the difference.

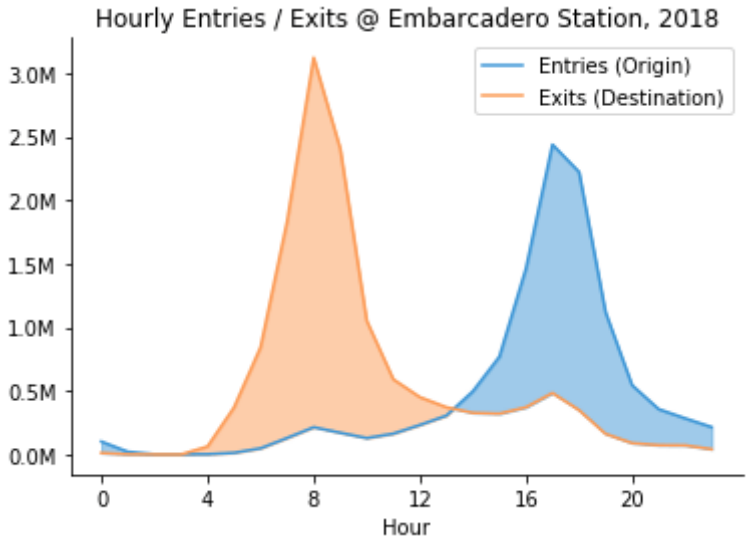


Fig. 10.23: Zero Baseline

2. Limiting overlapping in area chart

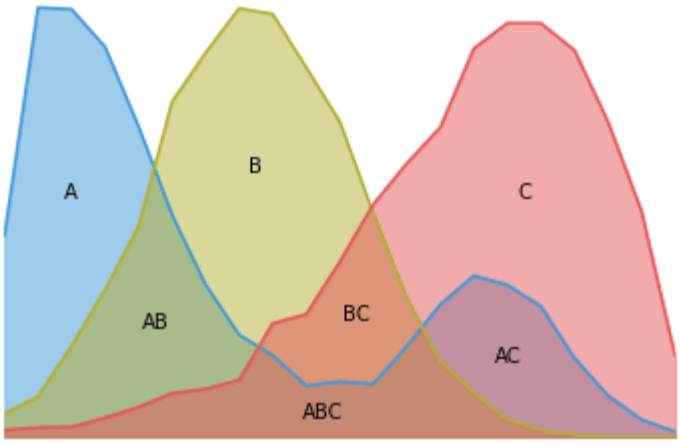


Fig. 10.24: limiting overlapping in area chart

The more series that are represented on an overlapping area chart, the greater the number of color combinations that can arise from their intersections. This phenomenon can often lead to visual complexity, making it challenging to interpret the relationships between the groups. For instance, even with just three series, the situation can become overwhelming. Each series is assigned a distinct color,

yielding three separate hues. The overlaps between each pair of series introduce an additional three colors, one for each pairwise combination. Furthermore, the area where all three series intersect adds yet another color to the palette. When calculated together, these elements result in a total of seven unique colors. This complexity can obscure the relationships between the data points, indicating that certain colors might not be readily associated with any single group. Therefore, careful consideration must be given to the design and interpretation of overlapping area charts to enhance clarity and comprehension (as shown in fig.10.24).

Comparing two series is usually safe, though if one series is always larger than the other, the plot can be easily mistaken for a stacked area plot instead. Readers can also be confused by interpreting the overlapping colors, which will not be present in the general legend. If you're thinking about using an overlapping area chart, limit yourself to two series and think about if using a line chart will show the comparison between groups more clearly.

3. Considering order of lines in stacked area chart

Although the overall shape of the plot will remain constant regardless of the order in which groups' lines are displayed, a thoughtful selection of line order can significantly enhance the readability of the visualization. A prudent guideline is to position the largest or most stable groups at the bottom of the plot, while placing the smaller or more variable groups above them. This arrangement takes into account that, as will be discussed in the following section, readers often find it challenging to extract values for any group other than the bottommost one. Therefore, prioritizing the placement of the most stable group at the top of the list is a logical choice, facilitating easier interpretation and analysis of the data presented.

Benefits of Using Area Charts

• Aesthetically Pleasing:

Area charts are attractive to the eye and can easily grab the attention of the audience because they are filled-in and colorful.

• Ideal for Trends:

They are ideal for displaying trends over time since the filled area below the line highlights the extent of change, making it simple to spot patterns and fluctuations.

• Eases Comparison:

Area charts enable comparison across various categories or datasets to be done conveniently, particularly if various areas are charted together on one graph. This aspect of comparison facilitates showing relative change and proportions.

Downsides to Utilizing Area Charts

• Limited Data Sets:

Area charts cannot effectively handle big or complex datasets since filled regions overlap and mask details, hence it's not easy to understand the data precisely.

• Not For Exact Values:

Area charts aren't as suited for expressing accurate numerical values because the focus lies on trends and ratios and not on exact quantities. This proves to be a drawback when having accurate data correctness is of top priority for examination or decision-making.

Applications of Area Plots

Area plots are widely used in various fields to visualize cumulative data trends over time. Some common applications include:

- **Financial Analysis**: Used to represent revenue, profit, and expenditure trends over time.
- **Population Studies**: Helps in analyzing demographic changes, such as population growth across different age groups.
- **Environmental Data Analysis**: Used for tracking changes in temperature, pollution levels, and climate patterns over the years.
- **Marketing and Sales**: Helps in visualizing market trends, customer acquisition rates, and product sales comparisons.
- **Healthcare and Epidemiology**: Used for studying trends in disease outbreaks, vaccination rates, and patient recovery statistics.
- **Technology and Internet Trends**: Helps in tracking website traffic, app downloads, and user engagement over time.

10.7 Boxplots

Boxplots (or whisker plots) condense the distribution of a dataset in terms of quartiles. They emphasize the median, outliers, and variability of the data. A box plot employs boxes and lines to represent the distributions of one or more groups of numeric data as visualize in fig.10.25. Box limits show the range of the central 50% of the data, with a central line indicating the median value. Lines are drawn from each box to show the extent of the rest of the data, with dots positioned beyond the line boundaries to mark outliers.

Example:

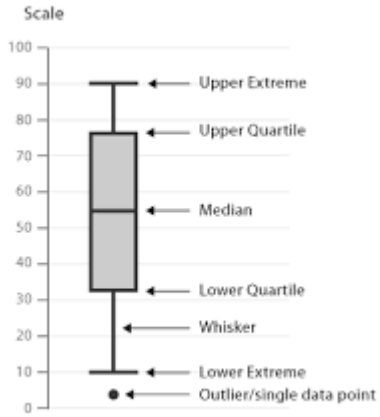


Fig. 10.25: Boxplots

10.7.1 How to construct a Box plot:

```
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
data = np.random.randn(100)
sns.boxplot(y=data)
plt.title('Boxplot Example')
```

plt.show()

Output:

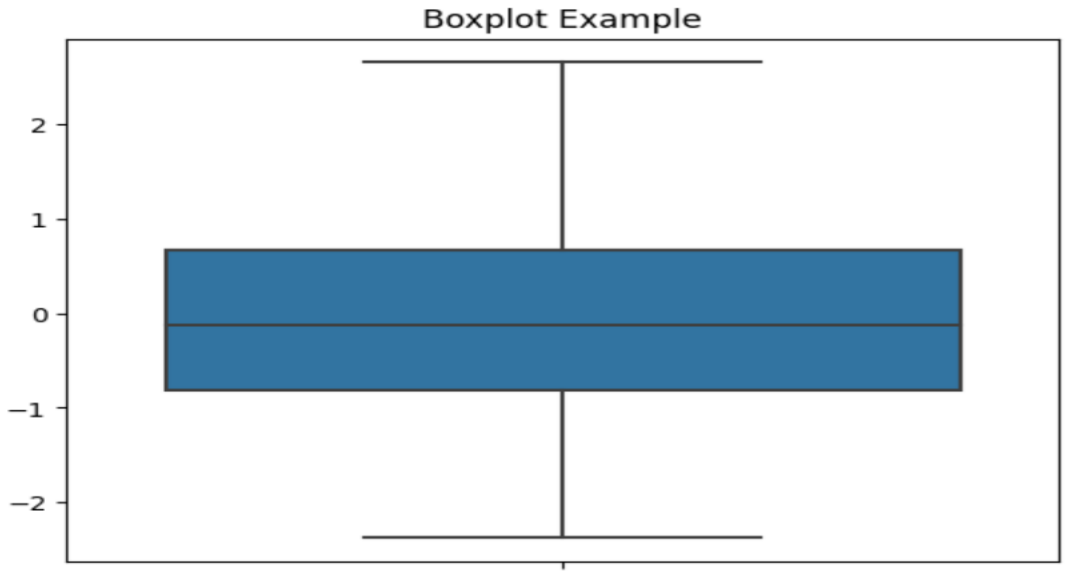


Fig. 10.26: Boxplot example

10.7.2 Usage of box plot

Box plots are a valuable tool for visualizing the distribution of numeric data values, particularly useful when comparing multiple groups. They provide a concise and clear representation of key statistical features, illustrating aspects such as symmetry, skewness, variance, and the presence of outliers within the data set. With box plots, one can quickly identify where the central bulk of the data lies, enabling meaningful comparisons across different groups.

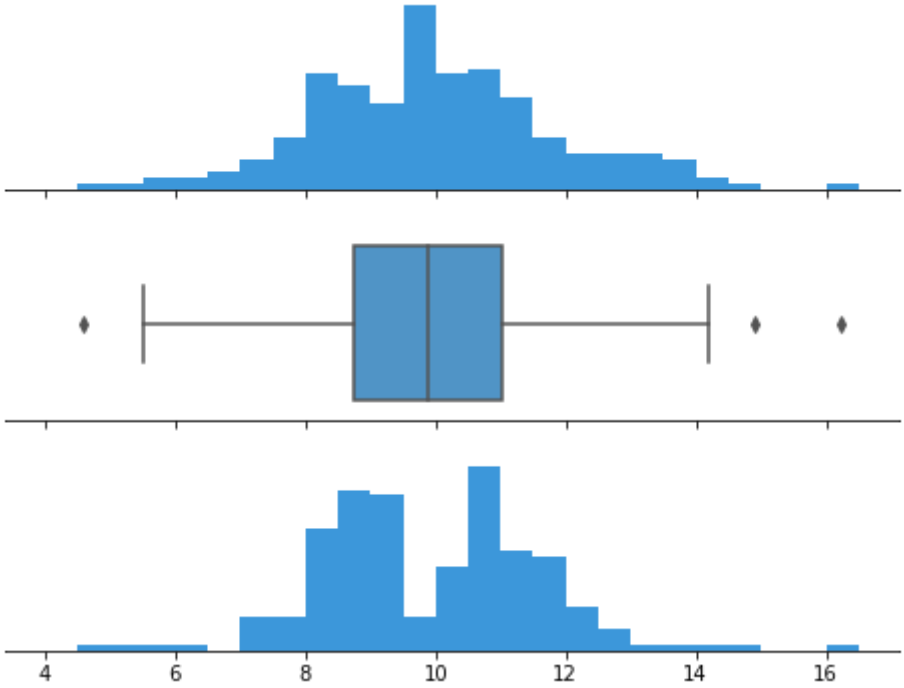


Fig. 10.27: Use of box plot

One notable feature of box plots is their ability to summarize complex data into easily interpretable visuals. They typically display the median, quartiles, and potential outliers, offering a high-level

overview that is accessible to a broad audience. This simplicity, however, comes with some limitations. While box plots effectively convey general distribution characteristics, they do not provide detailed information about the data's shape. Specifically, they obscurities details like the modality of the distribution, which refers to the number of peaks (or "humps") present in the data.

Consequently, while box plots serve as a powerful method for displaying distribution comparisons, users should be aware of the limitations in density representation. For datasets with significant complexity in shape or modality, additional techniques may be necessary to capture the finer nuances of the distribution, ensuring comprehensive insights into the underlying data characteristics (as shown in fig.10.27).

The data sets underlying both histograms produce the same box plot in the middle panel.

Understanding a box and whiskers

Box plot construction is centered on a dataset's quartiles, or the values that split a dataset into equal fourths. The first quartile (Q1) is higher than 25% of the data and lower than the remaining 75%. The second quartile (Q2) is in the middle, splitting the data in half. Q2 is another name for the median. The third quartile (Q3) is greater than 75% of the data and less than the rest of the 25%. In a box and whiskers diagram, the box ends and the middle line identify the positions of these three quartiles as shown in fig.10.28.

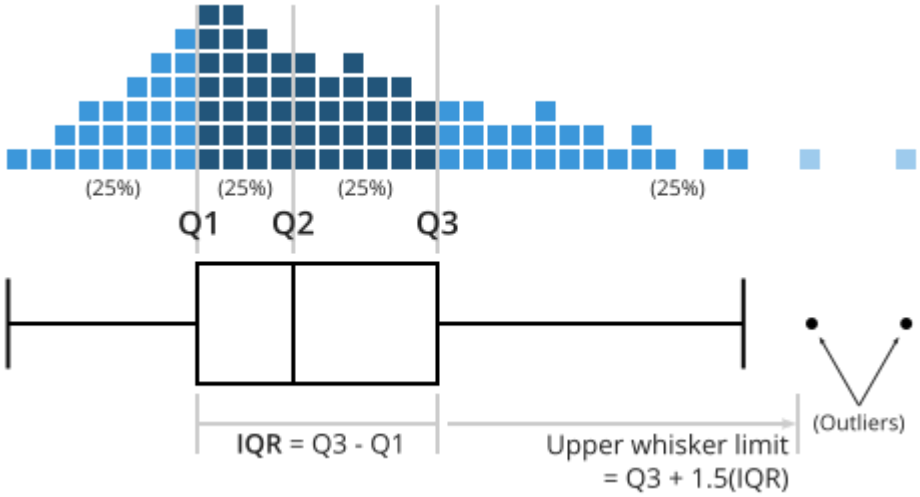


Fig. 10.28: Box and Whiskers

The space between Q3 and Q1 is referred to as the interquartile range (IQR) and is responsible for a large portion of the length of the whiskers extending from the box. Each whisker goes out to the most distant data point in each wing that is less than or equal to 1.5 times the IQR. Any data point beyond that distance is labelled as an outlier, and is indicated by a dot. There are alternative definitions of the whisker lengths, as explained below.

When a distribution of data is symmetrical, the median should sit exactly in the middle of the box; the distance between Q1 and Q2 should equal that between Q2 and Q3. Outliers are expected to be uniformly distributed on both sides of the box. If a distribution is skewed, the median will not lie in the middle of the box, but will lean to one side or another. Or you have this distribution with imbalanced whiskers, i.e., one side is short, having no outliers but the other is long with a lot of outliers.

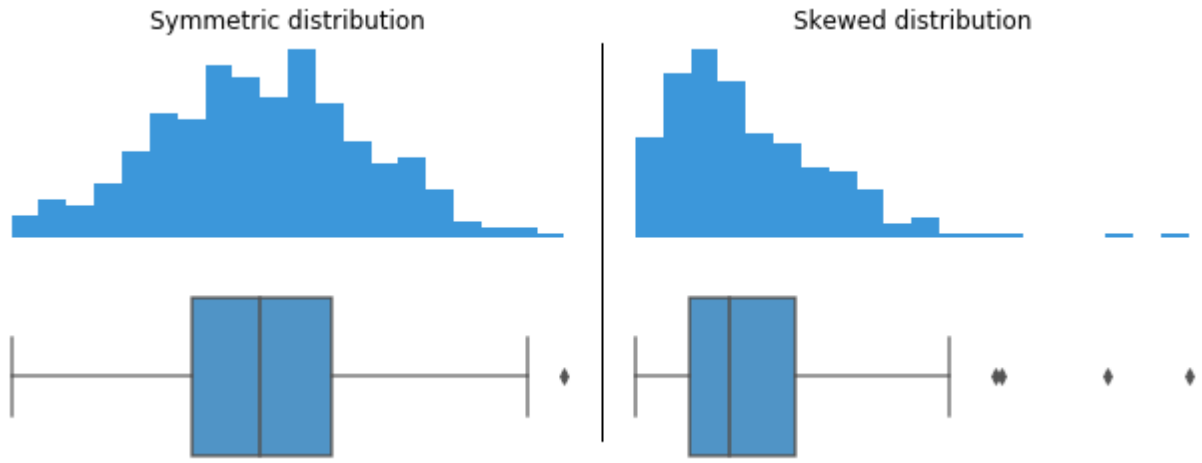


Fig. 10.29: Symmetric and Skewed Distribution

Applications of Boxplots

Boxplots are widely used in various fields to analyze data distributions and detect anomalies. Common applications include:

- **Statistical Analysis**: Used to summarize data distributions, compare multiple datasets, and detect skewness.
- **Quality Control**: Helps in identifying outliers and inconsistencies in manufacturing and production processes.
- **Finance**: Used for analyzing stock price fluctuations, comparing asset performances, and detecting extreme values.
- **Medical and Healthcare Research**: Helps in comparing patient data, such as blood pressure readings across different groups.
- **Machine Learning and Data Science**: Used for preprocessing data, identifying outliers, and understanding feature distributions before model training.
- **Educational Analysis**: Helps in comparing student test scores and academic performance among different institutions.

10.8 Scatter Plot

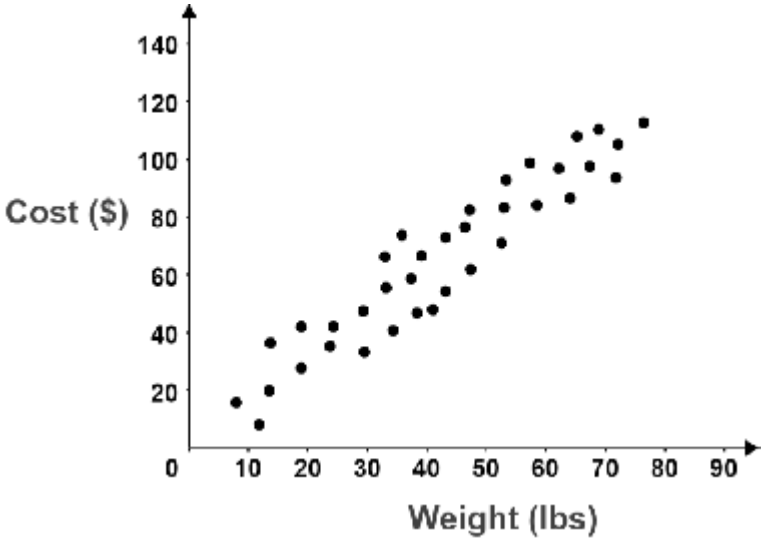


Fig. 10.30: Scatter Chart Example

A scatterplot is a graph that shows individual data points plotted along two axes. It is ideal for identifying correlations and patterns between two variables.

Scatter plots are used to visualize the relationship between two variables. Each data point in scatter plot is a value for both variables, and the location of the point on the graph is the values of the variables as shown in fig.10.30. Scatter plots are helpful in determining patterns, trends and relationships between variables.

10.8.1 How to construct a Scatter plot

```
import matplotlib.pyplot as plt
import numpy as np
x = np.random.rand(50)
y = np.random.rand(50)
plt.scatter(x, y, color='blue', alpha=0.5)
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Scatterplot Example')
plt.show()
```

Output:

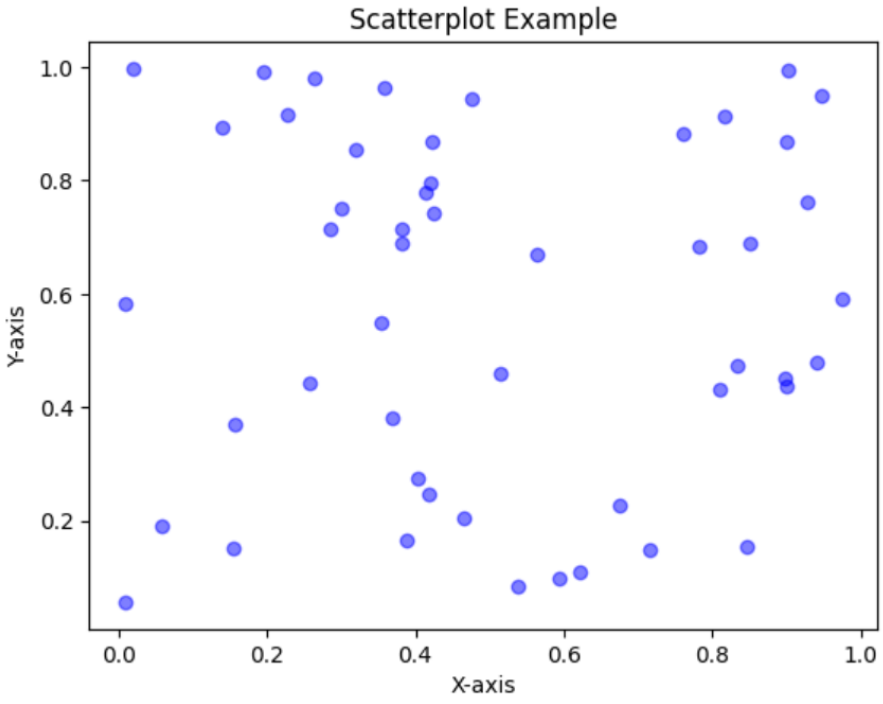


Fig. 10.31: Construction of scatter example

10.8.2 Usage of Scatter Plot

Main applications of scatter plots are to observe and display relations between two numerical variables. Scatter plot also indicate patterns if the data are considered as an aggregate. Correlational relationships are difficult to identify with scatter plots. In those situations, we would like to know, if we were given with the specific horizontal value, how a good prediction would be for the vertical value. You'll frequently see the variable on the horizontal axis labeled an independent variable, and the variable on the vertical axis labeled the dependent variable. Relationships between variables can

be characterized in numerous ways: positive or negative, strong or weak, linear or non-linear as shown in fig.10.32.

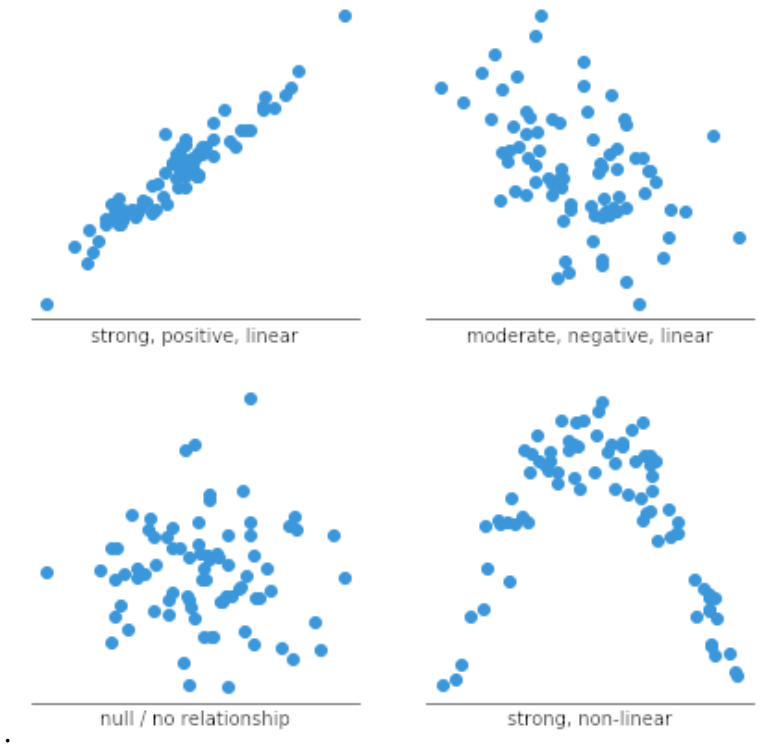


Fig. 10.32: Relationship of variables in scatter plot

A scatter plot can also be helpful in the identification of other data patterns as shown in fig.10.33. We can classify data points in categories depending on how tightly collections of points are bundled together. They can also indicate whether there are any surprising voids and outliers in the data. This can be helpful if we need to break the data into various segments, such as in user persona development.

Fig. 10.33: Identification of patterns

10.8.3 Common issues when using scatter plots

1. Overplotting

When the dataset have many data points to graph, this can fall into the problem of overplotting. Overplotting is when the data points overlap to the point where we cannot easily see relationships between points and variables. It is hard to determine how closely-packed data points are when many of them are concentrated in a small region as shown in fig.10.34. There are a couple of standard methods

to mitigate this problem. One solution is to plot only a subset of the data points: a random set of points will still provide the overall impression of the patterns in the complete data.We can also modify the shape of the dots, introducing transparency so that overlaps can be seen, or decreasing point size so that fewer overlaps happen. As a third alternative, we could even use a different chart type such as the heatmap where color is used to represent the number of points in each bin. Heatmaps in this application are also referred to as 2-d histograms.

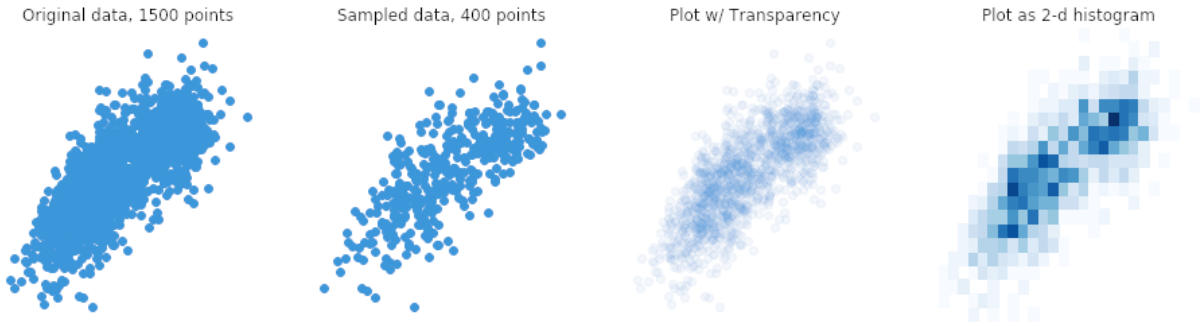


Fig. 10.34: Overplotting in scatter plot

Interpreting correlation as causation

This is less a problem with generating a scatter plot than it is a problem with interpreting it. Just because we see a relationship between two variables in a scatter plot, it does not necessarily mean that variations in one variable are causing variations in the other. This leads to the well-known saying in statistics that correlation does not imply causation. It is likely that the noted correlation is caused by some third factor influencing both of the variables graphed, that the cause and effect connection is reversed, or that the trend is mere coincidence.

For instance, it would be incorrect to examine city figures for how much green area they possess and crimes committed and come to the conclusion that one results in the other. This may overlook the truth that more populated cities will both have more of each and be correlated through the former and various other reasons.

If a causal relationship must be demonstrated, then additional analysis to account for or control other possible variables' effects must be conducted, in order to eliminate other potential explanations.

Benefits of Using Scatter Plots

• **Unveiling Trends and Associations:** Scatter plots are ideal to visually recognize patterns, trends, and associations among two variables. They facilitate an investigation of correlation and dependency within the data.

• **Simple to Interpret:** Scatter plots offer a simple visual picture of data points, making them simple for audience members to perceive and comprehend without needing intricate statistical expertise.

• **Identify Outliers:** Scatter plots allow one to easily spot outliers or unusual data points that are far away from the general trend. This can be very important in identifying unusual behaviour or data errors in the dataset.

Cons of Using Scatter Plot Charts

• **Limited to Two Variables:** Scatter plots are restricted to depicting relations between two variables. Although the simplicity may be beneficial for concentrated analysis, it also implies they can't depict interactions among more than two variables at a time

•**Not Suitable for Exact Comparisons**: Although scatter plots are great for determining trends and correlations, they might not be the most suitable for conducting exact comparisons between data points. Other charts, like bar charts or box plots, can perhaps best be used in comparing certain values or distributions across the data.

Applications of Scatterplots

Scatterplots are widely used in various fields for analyzing relationships between numerical variables. Common applications include:

- **Statistical Analysis**: Helps in identifying correlations, dependencies, and patterns between two continuous variables.
- **Machine Learning**: Used for visualizing feature relationships before applying regression or classification models.
- **Finance**: Helps in identifying trends between market variables like stock prices vs. trading volume.
- **Healthcare and Biology**: Used for analyzing medical data, such as the correlation between body mass index (BMI) and cholesterol levels.
- **Economics and Business**: Helps in market trend analysis, such as the relationship between advertising expenditure and revenue growth.
- **Social Sciences**: Used to examine relationships between variables, such as education levels and income rates.

10.9 Bubble Plots

A bubble chart or bubble plot is a variant of the scatter plot employed to examine relationships among three numeric variables. A dot on a bubble chart represents one point of data, and each of the variables' values for each point are specified by horizontal location, vertical location, and the size of dots as visualized in fig.10.35.

Example of Bubble plot:

Fig. 10.35: Bubble plot

How to construct a Bubble plot:

```
import numpy as np
import matplotlib.pyplot as plt
x = np.random.rand(30)
y = np.random.rand(30)
sizes = np.random.rand(30) * 1000
```

```
plt.scatter(x, y, s=sizes, alpha=0.5, color='green')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Bubble Plot Example')
plt.show()
```

Output:

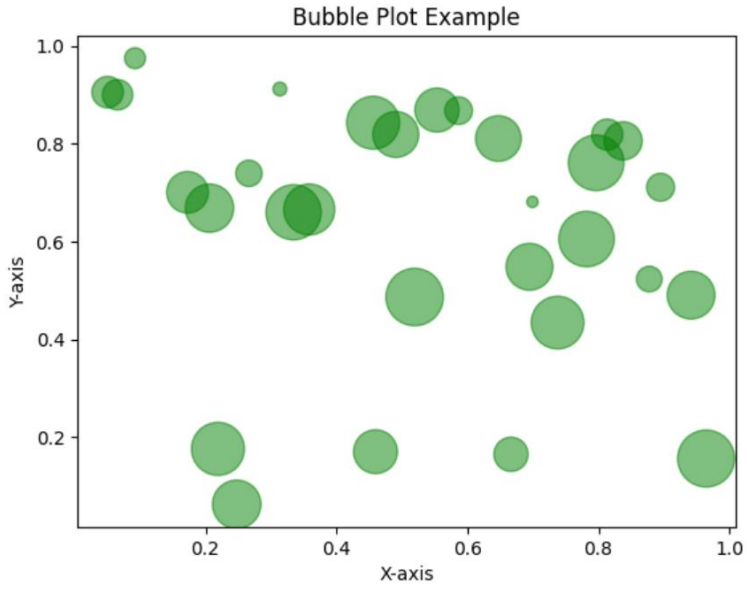


Fig. 10.36: Example of bubble plot

10.9.1 Usage of bubble chart

Similar to the scatter plot, a bubble chart is mostly employed to illustrate and demonstrate relationships between numeric variables. Yet, with the introduction of marker size as an added dimension, three variables can be compared instead of two. Within one bubble chart, we can have three various pairwise comparisons (X vs. Y, Y vs. Z, X vs. Z), as well as a general three-way comparison. It would take several two-variable scatter plots to achieve the same number of insights; even then, deducing a three-way relationship among data points won't be as straightforward as it is in a bubble chart.

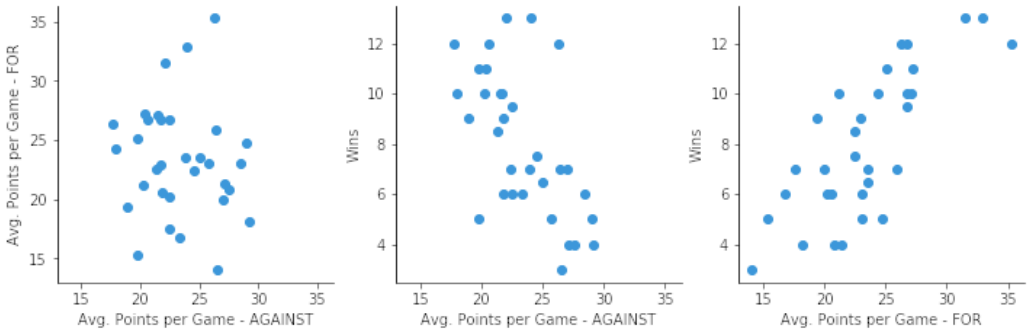


Fig. 10.37: Relationship of variables in bubble chart

The above three scatter plots (shown in fig.10.37) present the same data as the initial example bubble chart. Although it is more convenient to obtain the actual win counts for each team from this sequence of plots, the interaction between all three variables is not as directly expressed as in the bubble chart.

10.9.2 Best practices for using a bubble chart

1. Scale bubble area by value

An easy error to commit is to size the diameters or radii of the points to the values of the third variable. If this type of scaling is done, a point with twice the value of another point will have four times the area and thus its value will appear considerably larger than is truly justified. Instead, ensure that the areas of bubbles match the values of the third variable. In the above scenario, a point with double the value of another should have 2 * 1.41 times the radius or diameter so that its area is twice that of the smaller point.

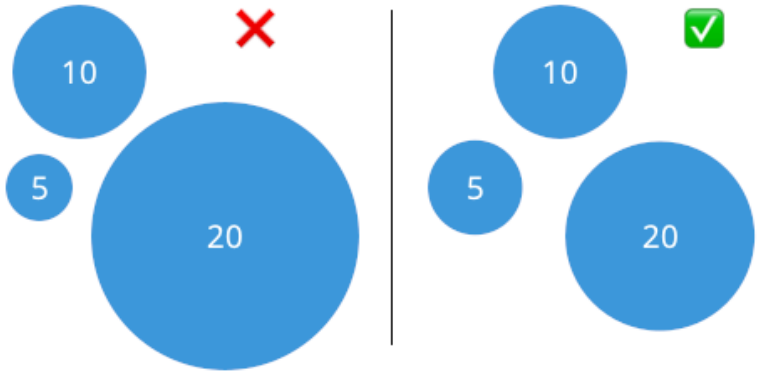
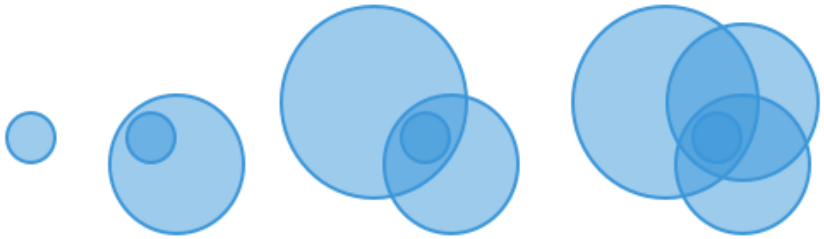


Fig. 10.38: scaling of bubble area by value

Depending on making of your bubble chart, you might have to increase your data to account for how data values get mapped to point size. Most visualization tools will automatically equate value with area, but watch out for those instances where value gets equated with diameter or radius instead.

2. Limit number of points to plot

Bubble charts often get drawn using transparency on points because overlaps are much more common a phenomenon than when all points are tiny. This overlap also implies that there's a limit to how many data points can be visualize while maintaining a readable plot.



The smaller point would not be discernible over the larger ones. There are no strict rules for whether a dataset is suitable for a bubble chart or not, but it's something to]keep in mind when designing a bubble chart. If there seems to be too much overplotting, then it may be worth considering a method of summarizing the data or selecting a different type of chart to display your data. Decreasing bubble size will be able to offer some physical separation between points, but this will also increase the difficulty of reading values from bubble sizes.

3. Include a legend

As a second piece of advice, it is advisable to add a legend or other key to your graph in order to indicate how varying bubble sizes relate to values of your third variable. It

is relatively simple to compare and assess values in terms of horizontal or vertical lengths and locations due to the tick marks on axes. A key to bubble sizes does the same job as those tick marks do for the third variable.

If you are visualizing using interactive software, it might be a good idea to enable the feature so that values can be seen when a point is hovered or clicked on. For publication, it is a good idea to annotate important points to enhance a bubble chart's communicative capabilities.

Applications of Boxplots

Boxplots are widely used in various fields to analyze data distributions and detect anomalies. Common applications include:

- **Statistical Analysis**: Used to summarize data distributions, compare multiple datasets, and detect skewness.
- **Quality Control**: Helps in identifying outliers and inconsistencies in manufacturing and production processes.
- **Finance**: Used for analyzing stock price fluctuations, comparing asset performances, and detecting extreme values.
- **Medical and Healthcare Research**: Helps in comparing patient data, such as blood pressure readings across different groups.
- **Machine Learning and Data Science**: Used for preprocessing data, identifying outliers, and understanding feature distributions before model training.
- **Educational Analysis**: Helps in comparing student test scores and academic performance among different institutions.

10.10 Waffle Charts

A waffle chart is a square grid-based visualization used to display parts-to-whole relationships (similar to pie charts). Each square in the grid represents a proportion of the total dataset, making it an effective tool for displaying percentages and comparisons in a visually appealing way as visualize in fig.10.39.

Key Features of Waffle Charts:

- Uses small squares (tiles) to represent data
- Easy to interpret and visually engaging
- Useful for showing percentages and proportions
- More space-efficient than pie charts

Example of a Waffle Chart

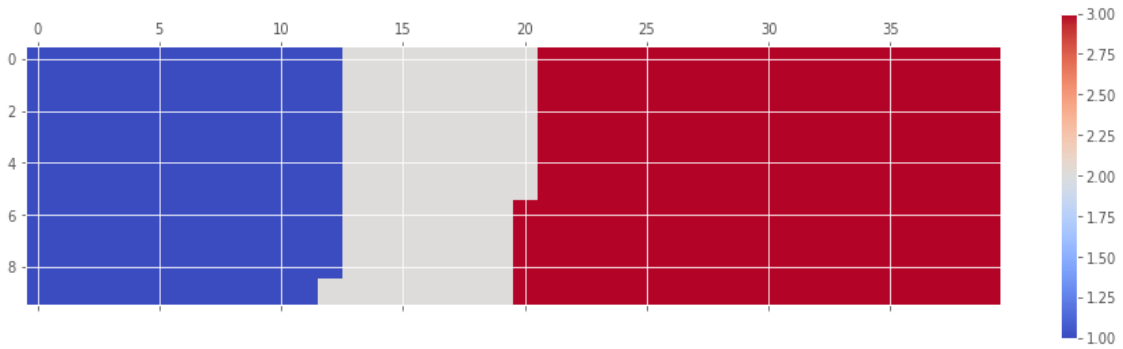


Fig. 10.39: Waffle chart

10.10.1 How to construct a Waffle Chart

```
# importing all necessary requirements
import pandas as pd
import matplotlib.pyplot as plt
from pywaffle import Waffle
# creation of a dataframe
data ={'phone': ['Xiaomi', 'Samsung',
                                    'Apple', 'Nokia', 'Realme'],
        'stock': [44, 12, 8, 5, 3]
        }
df = pd.DataFrame(data)
# To plot the waffle Chart
fig = plt.figure(
        FigureClass = Waffle,
        rows = 5,
        values = df.stock,
        labels = list(df.phone)
)
```

Output Image

The code generates a waffle chart representing the proportion of different stocks. Below is an example output labelled as fig.10.40.

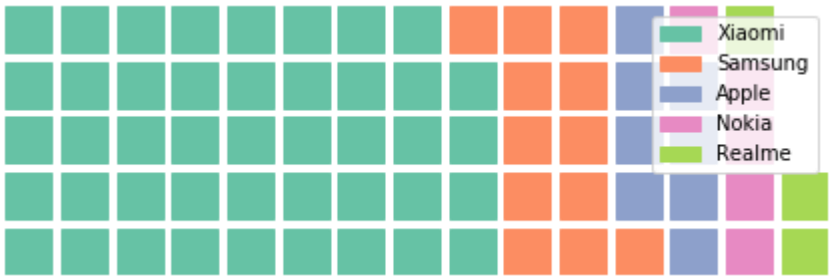


Fig. 10.40: Example waffle chart

10.10.2 When to Use Waffle Charts

Waffle charts are particularly useful when you need to:

- Represent proportions in a visually appealing and grid-like format.
- Provide an alternative to pie charts that is easier to compare across multiple categories.
- Show percentage-based data in an intuitive way, especially when exact numerical precision is not required.
- Make visualizations that are simple, engaging, and easy to understand for non-technical audiences.

Advantages of Waffle Charts

1. Easy to Interpret – Provides a clear and simple visual representation of data.
2. Great for Proportional Data – Best suited for percentage-based comparisons.
3. More Space-Efficient than Pie Charts – Uses a compact grid layout.

4. Visually Appealing – Uses colors and grids to make data engaging.
5. Better for Exact Comparisons – Unlike pie charts, where angles can be misleading, waffle charts provide a clearer numerical perspective.

Disadvantages of Waffle Charts

1. Limited Precision – Works best with rounded percentages (e.g., 10%, 20%, 30%).
2. Not Ideal for Large Categories – Becomes cluttered if there are too many data points.
3. Difficult to Read for Uneven Distributions – Small differences may not be easily noticeable.
4. Less Flexible than Bar Charts – Cannot represent negative values or non-percentage-based data effectively.
5. Hard to Compare Multiple Waffles – Unlike bar charts, comparing multiple waffle charts side by side is not intuitive.

Applications of Waffle Charts

1. Business & Marketing

Sales Distribution – Understanding how different products contribute to revenue. Market Share Analysis – Comparing a company's market position against competitors. Customer Segmentation – Visualizing demographic groups in survey data.

2. Healthcare & Medical Research

Patient Diagnosis Breakdown – Showing the proportion of diseases in a hospital. Vaccination Rates – Representing the percentage of vaccinated individuals. Hospital Resource Allocation – Visualizing the distribution of hospital beds, equipment, and staff.

3. Government & Policy Making

Budget Allocation – Displaying how government funds are distributed among different sectors. Election Results – Representing the percentage of votes per political party. Census Data – Showing the population distribution across regions.

4. Education & Research

Student Performance Analysis – Understanding subject-wise performance in schools. Survey Analysis – Representing responses in research studies. Scientific Data Representation – Displaying proportions in experimental results.

10.11 Word Clouds

A word cloud (also called a tag cloud) is a visual representation of text data where the size of each word reflects its frequency or importance. Words that appear more often in the text are displayed in larger and bolder fonts, while less frequent words appear smaller as shown in fig.10.41.

Word clouds are commonly used for:

- Text analysis and visualization
- Identifying key themes in large texts
- Summarizing customer feedback or survey responses
- Highlighting important topics in research papers, articles, or books

Example of a Word Cloud:

Fig. 10.41: Word cloud

10.11.1 How to Construct a Word Cloud

We'll use the Word Cloud library in Python to generate a word cloud from a sample text.

Installation

Before running the code, install the required libraries using:

bash

```
pip install wordcloud matplotlib nltk
```

Python Code

python

```
import numpy as np
import matplotlib.pyplot as plt
from wordcloud import WordCloud
from nltk.corpus import stopwords
import nltk
# Download stopwords
nltk.download("stopwords")
# Sample text data
text = """Data Science is an interdisciplinary field that uses scientific methods,
 algorithms, and systems to extract insights from structured and unstructured data.
 It applies machine learning, statistics, and artificial intelligence to analyze and interpret complex data."""
# Define stopwords
stop_words = set(stopwords.words("english"))
# Generate word cloud
wordcloud = WordCloud(width=800, height=400, background_color="white",
 stopwords=stop_words, colormap="viridis").generate(text)
# Display the word cloud
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.show()
```

Output Image: The code generates a word cloud where larger words appear more frequently in the input text. Here is an example output:

Fig. 10.42: Word cloud example

Advantages of Word Clouds

1. Easy to Understand – Provides a quick visual summary of large text data.
2. Engaging & Visually Appealing – Helps capture attention and highlight key themes.
3. Quick Insights – Allows businesses, researchers, and analysts to identify trends instantly.
4. Customizable – Colors, font sizes, and layouts can be adjusted for better readability.
5. Useful for Exploratory Data Analysis (EDA) – Helps in the initial stages of text analysis.

Disadvantages of Word Clouds

1. Lack of Context – Words appear individually without any relationship or sentence structure.
2. Bias in Word Frequency – Common words may overshadow important insights.
3. Not Suitable for Detailed Analysis – Does not provide deep insights into sentiment or meaning.
4. Stopword Removal Challenges – Some important words may be mistakenly removed or retained**.**
5. Misinterpretation – Large words may not always represent significance beyond frequency.

Applications of Word Clouds

1. Business & Marketing

- Customer Feedback Analysis – Identifying frequently mentioned words in reviews.
- Brand Sentiment Analysis – Understanding public perception from social media comments.
- SEO & Keyword Research – Finding commonly used words in search queries.

2. Education & Research

- Summarizing Academic Papers – Extracting key themes from research articles.
- Analyzing Books & Literature – Finding important words and themes in novels or historical texts.
- Student Assignments & Presentations – Making visually appealing text summaries.

3. Social Media & News Analysis

- Trending Topics Identification – Understanding popular topics on Twitter, Facebook, or news sites.
- Political Analysis – Examining speeches and debates to highlight key issues.

4. Healthcare & Medical Research

- Patient Feedback Analysis – Identifying common concerns in surveys and reports.
- Medical Record Analysis – Extracting keywords from patient history for faster diagnosis.

**** Notes ****

Fundamentals of Data Engineering Concepts

ISBN: 978-93-48620-19-4

About Editors

Dr. Sumit Chopra is currently serving as an Associate Professor at GNA University, Phagwara, with over 18 years of teaching experience in the field of Computer Science and Engineering. He has made significant contributions to academic research, having published numerous papers in reputed international conferences and peer-reviewed journals. Dr. Chopra is also an active reviewer for various research journals and has delivered expert talks at several esteemed institutions. He played a key role as the NBA Coordinator for the CSE Department at GNA University, successfully leading the B.Tech. (CSE) program to a 3-year accreditation. His primary areas of interest include Data Science and Artificial Intelligence, where he continues to engage in scholarly research and technical writing. Passionate about fostering academic excellence, Dr. Chopra combines his deep subject knowledge with practical insights, contributing meaningfully to the growth of the academic and professional community.

Ramandeep Kaur is currently pursuing her Master of Technology (M.Tech) in Computer Science and Engineering at GNA University. She holds a strong academic interest in Data Science and Artificial Intelligence, and is actively involved in research and technical writing within these domains. Demonstrating a passion for knowledge sharing and academic excellence, she has contributed as an editor for this book, where she played a key role in reviewing and refining content to enhance its clarity, accuracy, and academic value. Her efforts are aimed at supporting fellow students, educators, and researchers by ensuring the content is both informative and accessible. Ramandeep is dedicated to continuous learning and contributing meaningfully to the evolving field of computer science.

Er. Gagandeep Singh Bains holds a Bachelor of Technology (B.Tech.) degree in Computer Science and Engineering from CT Institute of Technology, Jalandhar, completed in 2010, and a Master of Technology (M.Tech.) degree in Computer Science and Engineering from Punjab Institute of Technology, Kapurthala, earned in 2014. With a strong academic background and dedication to the field of engineering education, he is currently serving as an Assistant Professor at GNA University, in the School of Engineering Design and Automation. Er. Bains is committed to fostering innovation and critical thinking among students, and his teaching is complemented by a solid foundation in both theoretical and practical aspects of computer science. He actively contributes to curriculum development and strives to create a dynamic learning environment. His professional interests include emerging technologies, coding, and student mentorship, making him a valuable member of the academic community at GNA University.

Bhumi Publishing

ISBN 978-93-48620-19-4
9 789348 620194