Indira Gandhi National Open University
School of Computer and Information
Sciences (SOCIS)

MCS-062
Introduction to Data Science

Block

4

DATA SCIENCE – ALLIED AREAS

Unit 11 Introduction to NOSQL and Bigdata
Unit 12 Mining Data Streams
Unit 13 Tools for Data Science – An Introduction

PROGRAMME DESIGN COMMITTEE

Prof. D. K. Lobiyal, JNU, New Delhi
Prof. Dinesh Kumar Vishwakarma, DTU, New Delhi
Prof. Sandeep Singh Rawat, SOCIS, IGNOU
Prof. P. V. Suresh, SOCIS, IGNOU
Prof. V.V. Subrahmanyam, SOCIS, IGNOU
Dr. Akshay Kumar, Associate Professor, SOCIS, IGNOU
Dr. M.P. Mishra, Associate Professor, SOCIS, IGNOU
Dr. Sudhansh Sharma, Assistant Professor, SOCIS, IGNOU

COURSE DESIGN COMMITTEE

Prof. D. K. Lobiyal, JNU, New Delhi
Prof. Dinesh Kumar Vishwakarma, DTU, New Delhi
Prof. T. V. Vijay Kumar, JNU, New Delhi
Prof. D. P. Vidyarthi, JNU, New Delhi
Prof. Vivek Kumar Singh, University of Delhi, Delhi
Prof. A. K. Mohapatra, IGDTUW, New Delhi
Mr. Kapil Pandey, HCL Technologies Limited, Delhi – NCR
Prof. Manish Trivedi, SOS (Statistics), IGNOU
Dr. Rajesh Kaliraman, Assistant Professor, SOS(Statistics), IGNOU
Dr. Prabhat Kumar Sangal, Assistant Professor, SOS (Statistics), IGNOU
Prof. Sandeep Singh Rawat, SOCIS, IGNOU
Prof. P.V. Suresh, SOCIS, IGNOU
Prof. V.V. Subrahmanyam, SOCIS, IGNOU
Prof. Divakar Yadav, SOCIS, IGNOU
Dr. Akshay Kumar, Associate Professor, SOCIS IGNOU
Dr. M.P. Mishra, Associate Professor, SOCIS IGNOU
Dr. Sudhansh Sharma, Assistant Professor, SOCIS IGNOU

SOCIS FACULTY

Prof. Sandeep Singh Rawat, Director, SOCIS, IGNOU
Prof. V.V. Subrahmanyam, SOCIS, IGNOU
Prof. Akshay Kumar, SOCIS, IGNOU
Dr. Sudhansh Sharma, Associate Professor, SOCIS, IGNOU
Prof. P.V. Suresh, IGNOU, SOCIS, IGNOU
Prof. Divakar Yadav, SOCIS, IGNOU
Prof.M.P. Mishra,SOCIS, IGNOU

COURSE PREPARATION TEAM

Content Writer (Unit-11)
(Adapted from MCS 226: Unit-5 & 8)
Dr. Sudhansh Sharma & Prof. Sandeep Singh Rawat
Assistant Professor Professor
School of Computer and Information Sciences, IGNOU

Content Editor (Unit-11 & 12)
Prof. T.V. Vijay Kumar
Jawahar Lal Nehru University (JNU),
New Delhi

Content Writer (Unit-12)
(Partially Adapted from MCS 226: Unit-10)
Dr. Sudhansh Sharma ***& Dr.*** Rakesh Kumar
SOCIS, IGNOU Central University of Haryana (CUH)

Content Editor (Unit-13)
Prof. Manish Trivedi
School of Sciences, IGNOU

Content Writer (Unit-13)
Dr. Neetu Singh, Assistant Professor,
Department of Computer Applications
Jaypee Institute of Information Technology(JIIT), Noida, U.P.

Language Editor
Prof. Parmod Kumar
School of Humanities, IGNOU

COURSE COORDINATOR

Dr. Sudhansh Sharma,
Associate Professor
School of Computers and Information Sciences, IGNOU.

PRINT PRODUCTION

Sh. Sanjay Aggarwal
Assistant Registrar
MPDD, IGNOU, New Delhi

October, 2025

©Indira Gandhi National Open University, 2025

ISBN-

All rights reserved. No part of this work may be reproduced in any form, by mimeograph or any other means, without permission in writing from the Indira Gandhi National Open University.

Further information on the Indira Gandhi National Open University courses may be obtained from the University's office at Maidan Garhi, New Delhi-110068.

Printed and published on behalf of the Indira Gandhi National Open University, New Delhi by MPDD, IGNOU.

Laser Typesetter: Tamal Basu

BLOCK 4 DATA SCIENCE – ALLIED AREAS

Block-4 of the "Introduction to Data Science" course ventures into advanced and allied areas that are central to the evolving data ecosystem. This block broadens the learner's perspective by introducing scalable data systems, real-time data processing, and essential tools that complement core data science tasks. The units in this block are carefully curated to bridge traditional data processing concepts with contemporary technological developments that define modern data infrastructures.

Unit 11: Introduction to NoSQL and Big Data addresses the limitations of conventional relational databases and introduces learners to the architecture and use-cases of NoSQL databases. This unit also covers the characteristics of Big Data and associated frameworks like HDFS, MapReduce, and Apache Hadoop. In today's data-rich environments, where data is generated at high velocity and volume, understanding these technologies is crucial. Industries such as e-commerce, finance, logistics, and healthcare rely on these systems to store, manage, and analyze petabytes of data efficiently.

Unit 12: Introduction to Data Stream Mining focuses on handling and analyzing continuous, real-time data streams. With the rapid growth of IoT devices, online platforms, and sensor networks, stream data analysis has become critical for applications such as fraud detection, stock market analysis, and network monitoring. This unit introduces data stream models, query processing, and associated challenges, enabling learners to handle time-sensitive and transient data effectively.

Unit 13: Tools for Data Science – An Introduction offers hands-on exposure to widely used tools in the data science industry. The basics of Tableau and Power BI introduce students to powerful data visualization and business intelligence platforms, while introductory modules on Python and R programming lay the groundwork for statistical computing and machine learning. Mastery of these tools is essential for any data science professional, as they are the backbone of analytics workflows across both academia and industry.

Overall, Block-4 prepares learners to engage with large-scale, high-speed data environments and equips them with the technical proficiency to use industry-standard tools. This exposure is especially valuable for students aiming to pursue careers in data engineering, real-time analytics, or advanced data science research. It ensures that learners not only understand foundational principles but are also adept at navigating the tools and platforms shaping the future of data science.

ignou
THE PEOPLE'S
UNIVERSITY

UNIT 11 INTRODUCTION TO NOSQL AND BIGDATA

Structure

11.0 Introduction

11.1 Objectives

11.2 Introduction to NoSQL

- 11.2.1 What is NoSQL
- 11.2.2 Brief History of NoSQL Databases
- 11.2.3 NoSQL Database Features
- 11.2.4 Differentiate between RDBMS and NoSQL

11.3 Types of NoSQL Databases

- 11.3.1 Column Based
- 11.3.2 Graph Based
- 11.3.3 Key-value Pair Based
- 11.3.4 Document Based

11.4 Introduction to Big Data

- 11.4.1 Big Data and Characteristics
- 11.4.2 Big Data Applications
- 11.4.3 Structured vs Semi-structured and Unstructured Data
- 11.4.4 Big Data vs Data Warehouse
- 11.4.5 Distributed File System
- 11.4.6 HDFS and Map Reduce
- 11.4.7 Apache Hadoop 1 and 2 (YARN)

11.5 Summary

11.6 Solutions/Answers

11.7 Further Readings

11.0 INTRODUCTION

NoSQL databases have significantly reshaped the way data is stored and accessed, even though relational databases remain widely utilized. Modern applications often feature advanced functionalities like Google-style search, reflecting a shift in user expectations and data handling needs. The surge in data generation, along with the proliferation of internet usage, mobile technologies, and the rising importance of analytics, has led to new demands on database systems. In response to these evolving requirements, NoSQL databases have emerged and grown rapidly, offering various models such as key-value stores, document databases, column-oriented databases, and graph-based systems.

In the current digital landscape, enormous volumes of data—measured in terabytes—are produced every day from a vast range of sources. These include social media platforms, corporate systems (such as production data, customer records, and financial information), user interactions, sensor data,

and digital devices like smart phones and vehicles. This overwhelming and continuous generation of data is categorized under the umbrella term "Big Data." The concept of Big Data is defined by four key characteristics: volume (the sheer amount of data), velocity (the speed at which data is generated and processed), variety (the different types and formats of data), and veracity (the reliability and accuracy of data).

The discipline that focuses on extracting valuable insights from this vast data landscape is known as **Big Data Analytics**. It requires sophisticated tools and technologies to process and analyze data effectively. The structural design that supports this process is referred to as **Big Data Architecture**. This architecture outlines how data is collected from various sources, processed and stored, analyzed for insights, and ultimately delivered to the end-user or business decision-maker.

At the heart of Big Data Analytics lies the Big Data Architecture, which is typically divided into four logical layers. Each of these layers plays a vital role in managing and analyzing large-scale data:

- **Big Data Source Layer**: This initial layer manages both batch and real-time data input from various origins, including traditional data warehouses, relational database systems, SaaS platforms, and Internet of Things (IoT) devices.
- **Management and Storage Layer**: Data received from the sources is transformed into formats suitable for analytics tools and then stored accordingly in this layer.
- **Analysis Layer**: Here, the stored data is examined to uncover patterns, trends, and business intelligence insights.
- **Consumption Layer**: Finally, this layer ensures that the results of the analysis are made accessible to users or integrated into business intelligence systems for strategic decision-making.

In addition to these layers, Big Data Architecture also supports several essential functions:

- **Connecting to Data Sources**: With the help of connectors and adapters, the system can access a diverse array of storage types, network protocols, and data formats.
- **Data Governance**: This function ensures that privacy and security regulations are applied throughout the data lifecycle—from ingestion to storage and eventual deletion.
- **Systems Management**: As these architectures are generally deployed across large, distributed clusters, they require constant monitoring and control through centralized management tools.
- **Quality of Service Assurance**: This framework helps maintain high standards of data quality, defines compliance policies, and manages the frequency and volume of data intake.

In this Unit, we will delve into the various types of NoSQL databases, particularly those based on key-value pairs, documents, columns, and graphs.

Furthermore, we will explore the concept of Big Data in depth—its key characteristics and the broad spectrum of applications it supports.

11.1 OBJECTIVES

After going through this unit, you should be able to:

- define what is NoSQL;
- differentiate between NoSQL and SQL;
- explain the basic features of Column based NoSQL Database;
- explain the Graph-based NoSQL Database;
- explain the Key-value pair based NoSQL Database and
- explain the Document based NoSQL Database.
- explain Big data and its Characteristics
- compare the characteristics of Structured Semi-Structured, and Unstructured data
- differentiate Big data and Data warehouse
- describe the Distributed file system
- explain HDFS and Map Reduce
- differentiate between Apache Hadoop 1 and 2 (YARN).

11.2 INTRODUCTION TO NoSQL

Databases are a crucial part of many technological and practical systems. The phrase "NoSQL database" is frequently used to describe any non-relational database. NoSQL is sometimes referred to as "non SQL," but it is also referred to as "not only SQL." In either case, the majority of people agree that a NoSQL database is a type of database that stores data in a format that is different from relational tables.

Whenever you want to use the data, it must first be saved in a particular structure and then converted into a usable format. On the other hand, there are some circumstances in which the data are not always presented in a structured style, which means that their schemas are not always rigorous. This unit provides an in-depth look into NoSQL and the features that make it unique.

11.2.1 What is NoSQL?

NoSQL is a way to build databases that can accommodate many different kinds of information, such as key-value pairs, multimedia files, documents, columnar data, graphs, external files, and more. In order to facilitate the development of cutting-edge applications, NoSQL was designed to work with a variety of different data models and schemas.

The great functionality, ease of development, and performance at scale offered by NoSQL have helped make it a popular name. NoSQL is sometimes referred to as a non-relational database due to the numerous data

handling features it offers. Due to the fact that it does not adhere to the guidelines established by Relational Database Management Systems (RDBMS), you cannot query your data using conventional SQL commands. We can think of such well-known examples as Mongo DB, Neo4J, Hyper Graph DB, etc.

11.2.2 Brief history of NoSQL Databases

In the late 2000s, as the price of storage began to plummet, No-SQL databases began to gain popularity. No longer is it necessary to develop a sophisticated, difficult-to-manage data model to prevent data duplication. Because developers' time was quickly surpassing the cost of data storage, NoSQL databases were designed with efficiency in mind.

Table 11.1: History of Databases

Year	Database Solutions	Company / Database Technology
1970-2000	Mainly RDBMS related	Oracle, IBM DB2, SQL Server, MySQL
2000-2005	Dot Com boom – new scale solutions, start of NoSQL dev, whitepapers	Google, Facebook, IBM, amazon
2005-2010	New open source & mainstream databases	Cassandra, Riak, Apache Hbase, neo4j, MongoDB, CouchDB, Redis
2010 onwards	Adoption of Cloud	DBaaS (Database as a Service)

As storage costs reduced significantly, the quantity of data that applications were required to store and query grew. This data came in all forms—structured, semi-structured, and unstructured — and sizes making it practically difficult to define the schema in advance. NoSQL databases give programmers a great deal of freedom by enabling them to store enormous amounts of unstructured data.

In addition, the Agile Manifesto was gaining momentum, and software developers were reconsidering their approach to software development. They were beginning to understand the need of being able to quickly adjust to ever-evolving requirements. They required the flexibility to make rapid iterations and adjustments to all parts of their software stack, including the underlying database. They were able to achieve this flexibility because of NoSQL databases.

The use of the public cloud as a platform for storing and serving up data and applications was another trend that arose, as cloud computing gained popularity. To make their applications more robust, to expand out rather than up, and to strategically position their data across geographies, they needed the option to store data across various servers and locations. These features are offered by some NoSQL databases like MongoDB.

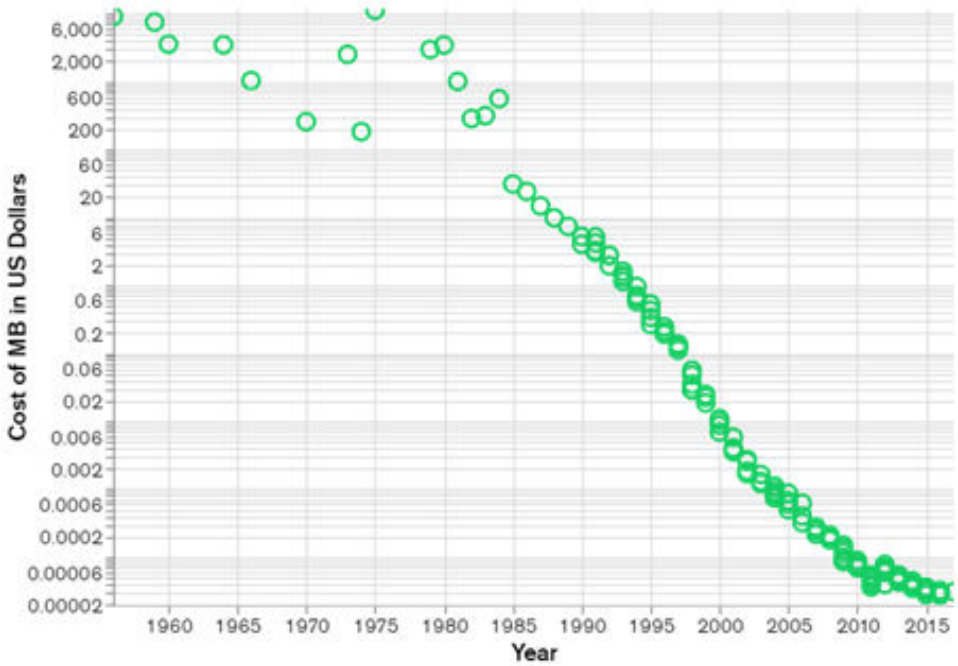


Figure 11.1: Cost per MB of Data over Time (Log Scale)
(Adapted from https://www.mongodb.com)

11.2.3 NoSQL Database Features

Every NoSQL database comes with its own set of one-of-a-kind capabilities. The following are general characteristics shared by several NoSQL databases:

- Schema flexibility
- Horizontal scaling
- Quick responses to queries as a result of the data model
- Ease of use for software developers

11.2.4 Difference between RDBMS and NoSQL

The differences and similarities between the two DBMSs are as follows:

For the most part, NoSQL databases fall under the category of non-relational or distributed databases, while SQL databases are classified as Relational Database Management Systems (RDBMS).

Databases that use the Structured Query Language (SQL) are table-oriented, while NoSQL databases use either document-oriented or key-value pairs or wide-column stores, or graph databases.

Unlike NoSQL databases, which have dynamic or flexible schema to manage unstructured data, SQL databases have a strict or static schema.

Structured data is stored using SQL, whereas both structured and unstructured data can be stored using NoSQL.

SQL databases are thought to be scalable in a vertical direction, whereas NoSQL databases are thought to be scalable in a horizontal direction.

Increasing the computing capability of your hardware is the first step in the

scaling process for SQL databases. In contrast, NoSQL databases scale by distributing the load over multiple servers.

MySQL, Oracle, PostgreSQL, and Microsoft SQL Server are all examples of SQL databases. BigTable, MongoDB, Redis, Cassandra, RavenDb, Hbase, CouchDB, and Neo4j are a few examples of NoSQL databases.

Vertical scalability is required for SQL databases. This means that an excessive amount of load must be able to be managed by increasing the amount of CPU, SSD, RAM, GPU, etc. on your server. When it comes to NoSQL databases, the ability to scale horizontally is one of their defining characteristics. This means that the addition of additional servers will make the task of managing demand more manageable.

☞Check Your Progress 1

1) What is NoSQL?

……

……

……

2) What are the features of NoSQL databases?

……

……

……

3) Differentiate between the NoSQL and SQL.

……

……

……

11.3 TYPES OF NOSQL DATABASES

In this section, we will discuss the many classifications of NoSQL databases. There are typically four types of NoSQL databases:

1) Column-based: Instead of accumulating data in rows, this method organizes it all together into columns, which makes it easier to query large datasets.

2) Graph-based: These are systems that are utilized for the storage of information regarding networks, such as social relationships.

3) Key-value pair based: This is the simplest sort of database, in which each item of your database is saved in the form of an attribute name (also known as a "key") coupled with the value.

4) Document-based: Made up of sets of key-value pairs that are kept in documents.

11.3.1 Column Based

A column store, in contrast to a relational database, is arranged as a set of columns, rather than rows. This allows you to read only the columns you need for analysis, saving memory space that would otherwise be taken up by irrelevant information. Because columns are frequently of the same kind, they are able to take advantage of more efficient compression, which makes data reading even quicker. The value of a specific column can be quickly aggregated using columnar databases.

Although columnar databases are excellent for analytics, because of the way they publish data, it is challenging for them to remain consistent because writes to all the columns need several write events on the disk. However, this problem never arises with relational databases because row data is continuously written to disk.

How Does a Column Database Work?

A columnar database is a type of database management system (DBMS) that allows data to be stored in columns rather than rows. It is accountable for reducing the amount of time needed to return a certain query. Additionally, it is accountable for the significant enhancement of the disk I/O performance. Both data analytics and data warehousing benefit from it. Additionally, the primary goal of a Columnar Database is to read and write data in an efficient manner. Column-store databases include Casandra, CosmoDB, Bigtable, and HBase, to name a few.

Columnar Database Vs Row Database:

When processing Big data analytics and data warehousing, there are a number of different techniques that can be used, including columnar databases and row databases. But they each take a different method.

For instance:

- Row Database: “Customer 1: Name, Address, Location". (The fields for each new record are stored in a long row).
- Columnar Database: “Customer 1: Name, Address, Location”. (Each field has its own set of columns). Refer Table 2 for relational database example.

Table 11.2: Relational database: an example

ID Number	First Name	Last Name	Amount
A01234	Sima	Kaur	4000
B03249	Tapan	Rao	5000
C02345	Srikant	Peter	1000

In a Columnar DBMS, the data will be stored in the following format:

A01234, B03249, C02345; Sima, Tapan, Srikant; Kaur, Rao, Peter; 4000, 5000, 1000.

In a Row-oriented DBMS, the data will be stored in the following format:

A01234, Sima, Kaur, 4000; B03249, Tapan, Rao, 5000; C02345, Srikant, Peter, 1000.

Columnar databases: advantages

The use of columnar databases has various advantages:

- Column stores are highly effective in compression, making them storage efficient. This implies that you can conserve disk space while storing enormous amounts of data in a single column.
- Aggregation queries are fairly quick with column-store databases because the majority of the data is kept in a column, which is beneficial for projects that need to execute a lot of queries quickly.
- Load times are also quite good; a table with a billion rows can be loaded in a matter of seconds. This suggests that you can load and query practically instantly.
- A great deal of versatility because columns do not have to resemble one another. The database would not be affected if you add new or different columns, however, updating all tables is necessary to input whole new record queries.
- Overall, column-store databases are excellent for analytics and reporting due to their quick query response times and capacity to store massive volumes of data without incurring significant costs.

Column databases: Disadvantages

While there are many benefits to adopting column-oriented databases, there are also a few drawbacks to keep in mind.

- It takes a lot of time and effort to create an efficient indexing schema.
- Incremental data loading is undesirable and is to be avoided, if at all possible, even though this might not be a problem for some users.
- This applies to all forms of NoSQL databases, not just those with columns. Web applications frequently have security flaws, and the absence of security features in NoSQL databases does not help. If security is your top goal, you should either consider using relational databases or, if it's possible, use a clearly specified schema.
- Due to the way data is stored, Online Transaction Processing (OLTP) applications are incompatible with columnar databases.

Are column databases always NoSQL?

Before we conclude, we should note that column-store databases are not always NoSQL-only. It is frequently argued that column-store belongs firmly in the NoSQL camp because it differs so much from relational database approaches. The debate between NoSQL and SQL is generally quite nuanced, therefore this is not usually the case. They are essentially the same as SQL techniques when it comes to column-store databases. For instance, keyspaces

function as schema, so schema management is still necessary. A NoSQL data store's keyspace contains all column families. The concept is comparable to relational database management systems' schema. There is typically only one keyspace per program. Another illustration is the fact that the metadata occasionally resembles a conventional relational DBMS perfectly. Ironically, column-store databases frequently adhere to ACID and SQL standards. However, NoSQL databases are often either document-store or key-store, neither of which are column-store. Therefore, it is difficult to claim that column-store is a pure NoSQL system.

11.3.2 Graph Based

The initial hardware hurdles that made it feasible for SQL to handle vast quantities of data are no longer there, despite the fact that SQL is an excellent superb RDBMS and has been used for many years to manage massive amounts of data. As a result, NoSQL has rapidly emerged as the dominant form of contemporary database management and many of the largest websites we rely on today are powered by NoSQL, like Twitter's use of FlockDB and Amazon's DynamoDB.

A database that stores data using graph structures is known as a graph database. It represents and stores data using nodes, edges, and attributes rather than tables or documents. Relationships between the nodes are represented by the edges. This makes data retrieval simpler and, in many circumstances, only requires one action. Additionally, it works fantastically as a database for fast, threaded data structures like those used on Twitter

How does a Graph Database Work?

Graphs, which are not relational databases, rely heavily on the idea of multi-relational data "pathways" for their functionality. However, the structure of graph databases is typically simple. They are largely made up of two elements:

- The Node: This represents the actual data itself. It may be the number of people who watched a video on YouTube; it could be the number of people who read a tweet; or it could even be fundamental information like people's names, addresses, and other such details.
- The Edge: This clarifies the real connection between the two nodes. It is interesting to note that edges can also have their own data, such as the type of connection between two nodes. Similar to edges, mentioned directions may also describe the direction in which the data is flowing.

Graph databases are mostly utilized for studying relationships. For instance, businesses might extract client information from social media using a graph database. For example, some organizations might use a graph database to extract data about relationships between Person, Restaurant, and City, as shown in Figure 11.2.

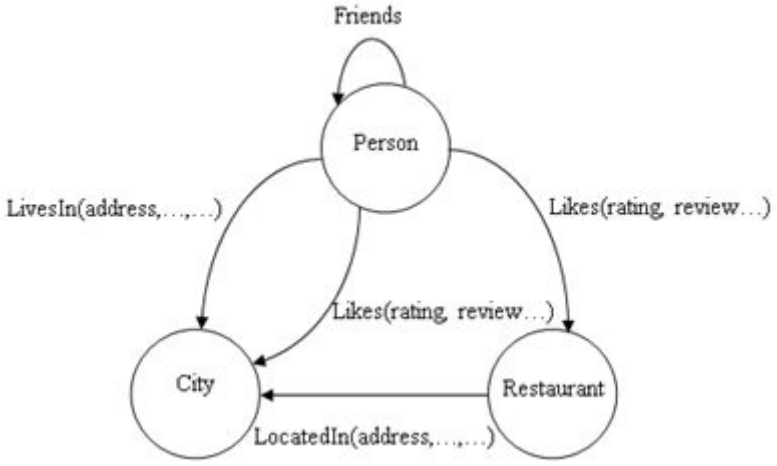


Figure 11.2: Different Nodes and Edges in Graph Database.
(Adapted from https://www.kdnuggets.com/)

When do we need a Graph Database?

1) It resolves issues with many-to-many relationships. For example, many-to-many relationships include friends of friends.
2) When connections among data pieces are more significant. For example, there is a profile with some unique information, but the main selling point is the relationship between these different profiles, which is how you get connected inside a network.
3) Low latency with big amounts of data. The relational database's data sets will grow significantly as you add more relationships, and when you query it, its complexity will increase and it will take longer than usual. However, graph databases are specifically created for this purpose, and one can easily query relationships.

Now, let's look at a more specific illustration to explain a group of people's complicated relationships. For example, five friends share a social network. These friends are Binny, Bhawna, Chaitaya, Manish, and Mohit. Their personal data may be kept in a graph database that resembles this, as shown in Figure 11.3 and Table 11.3:

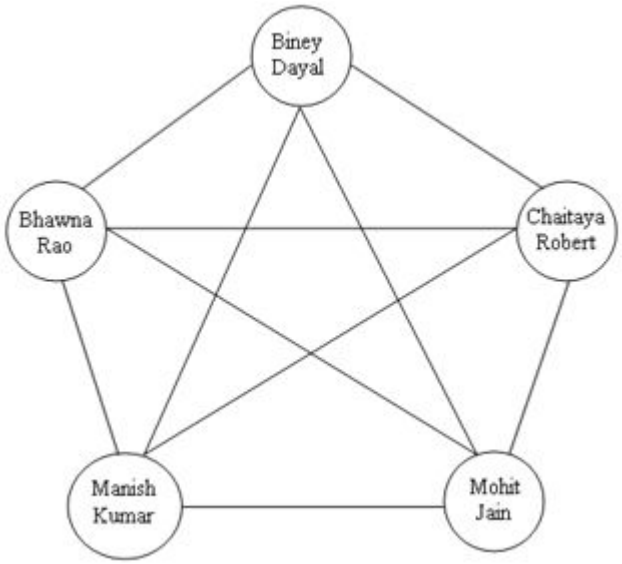


Figure 11.3: Example-Five friends sharing social network.

Table 11.3: Relational database: an example

Id	Firstname	Last Name	Email	Mobile
1001	Biney	Dayal	binnya@example.com	8645212321
1002	Bhawna	Rao	bhawanrao@example.com	9645212323
1003	Chaitaya	Robert	chaitayarob@example.com	7645212356
1004	Manish	Kumar	mkumar@example.com	9955212320
1005	Mohit	Jain	mjain@example.com	9945212329

This means we will need yet another table to keep track of user relationships. Our friendship table (refer Table 11.4) will resemble the following:

Table 11.4: Friendship Table

user_id	friend_id
1001	1002
1001	1003
1001	1004
1001	1005
1002	1001
1002	1003
1002	1004
1002	1005
1003	1001
1003	1002
1003	1004
1003	1005
1004	1001
1004	1002
1004	1003
1004	1005
1005	1001
1005	1002
1005	1003
1005	1004

We won't go too deeply into the theory of the database's main key and foreign key. Instead, presume that the friendship table uses both friends' ids. Let's say that every member on our social network gets access to a feature that lets

them view the personal information of their other users who are friends with them. This means that if Chaitaya were to ask for information, it would be regarding Biney, Bhawna, Manish and Mohit. We shall address this issue in a conventional (relational database) manner. First, we need to locate Chaitaya's user id in the database's Users table (refer Table 11.5).

Table 11.5: Chaitaya's Record

Id	First Name	Last Name	Email	Mobile
1003	Chaitaya	Robert	chaitanyarob@example.net	7645212356

We would now search the friendship table (refer Table 6) for all tuples with the user id of 3. The resulting relationship would look like this:

Table 11.6: Friendship Table for user id 3

user_id	friend_id
1003	1001
1003	1002
1003	1004
1003	1005

Let us now examine the time required for this Relational database strategy. This will be close to log (N) times, where N is the number of tuples in the friendship table. In this case, the database continues to keep the entries in sequential order based on their ids. So, in general, the time complexity for 'M' number of queries is M*log (N). Only, if we had used a graph database strategy the overall time complexity would have been O (N). For the simple reason that once Chaitaya has been located in the database, all the rest of her friends may be found with a single click, as shown in Figure 11.4.

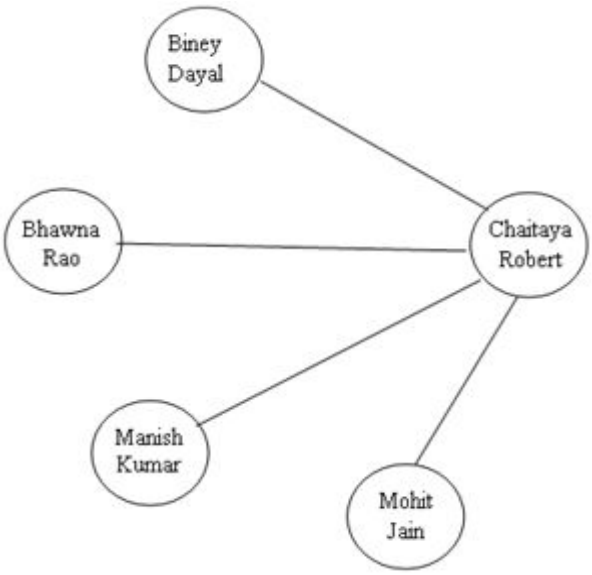


Figure 11.4: Accessing others data with a single click

Graph Database Examples

Although graph databases are not as widely used as other NoSQL databases, there are a handful that have become de facto standards when discussing NoSQL:

Neo4j is both an open-source and an interestingly developed Java graph database. It is considered to be one of the best graph databases. In addition to that, it comes with its own language known as Cypher, which is comparable to the declarative SQL language but is designed to work with graphs. In addition to Java, it supports a number of other popular programming languages, including Python, .NET, JavaScript, and a few others. Neo4j excels in applications such as the administration of data centers and the identification of fraudulent activity.

RedisGraph is a graph module that is integrated into Redis, which is a key-value NoSQL database. RedisGraph was developed to have its data saved in RAM for the same reason that Redis itself is constructed on in-memory data structures. As a result, a graph database with excellent speed and quick searching and indexing is created. RedisGraph also makes use of Cypher, which is ideal if you're a programmer or data scientist looking for greater database flexibility. Applications that require blazing-fast performance are the main uses.

OrientDB It is interesting to note that OrientDB supports graph, document store, key-value store, and object-based data formats. Having stated that, the graph model, which uses direct links between databases, is used to hold all of the relationships. Although it does not use Cypher, OrientDB is open-source and developed in Java, just like Neo4j and the two prior graph databases. OrientDB is designed to be used in situations when many data models are necessary, and as a result, it is optimized for data consistency as well as minimizing data complexity.

11.3.3 Key-value Pair Based

Key-value stores are perhaps the most widely used of the four major NoSQL database formats because of their simplicity and quick performance. Let us examine key-value stores' operation and application in more detail. With some of the most well-known platforms and services depending on them to deliver material to users with lightning speed, NoSQL has grown in significance in our daily lives. Of course, NoSQL includes a range of database types, but key-value store is unquestionably the most used.

Because of its extreme simplicity, this kind of data model is built to execute incredibly quickly when compared to relational databases. Furthermore, because key-value stores adhere to the scalable NoSQL design philosophy, they are flexible and simple to set up.

How Does a Key-Value Work?

In reality, key-value storage is quite simple. A value is saved with a key that specifies its location, and a value can be pretty much any piece of data or information. In reality, this design idea may be found in almost every programming language as an array or map object, refer Figure 11.5. The fact

that it is persistently kept in a database management system makes a difference in this case.

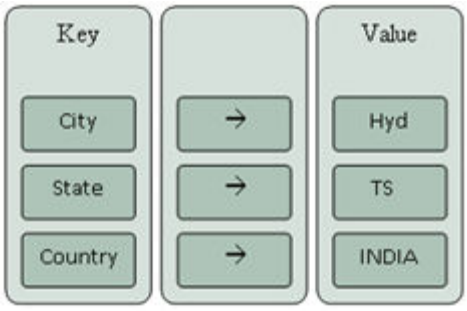


Figure 11.5: Example Key-Value database.

Popularity of key-value stores is due to the fact that information is stored as a single large piece of data instead of as discrete data. As a result, indexing the database is not really necessary to improve its performance. Instead, because of the way it is set up, it operates more quickly on its own. Similar to that, it mostly uses the get, put, and delete commands rather than having a language of its own.

Of course, this has the drawback that the data you receive in response to a request is not screened. Under certain conditions, this lack of data management may be problematic, but generally speaking, the trade-off is worthwhile. Because key-value stores are both quick and reliable, the vast majority of programmers find ways to get around any filtering or control problems that may arise.

Benefits of Key-Value

Key-value data models, one of the more well-liked types of NoSQL data models, provide many advantages when it comes to creating a database:

Scalability: Key-value stores, like NoSQL in general, are infinitely scalable in a horizontal fashion, which is one of its main advantages over relational databases. This can be a huge advantage for sophisticated and larger databases compared to relational databases, where expansion is vertical and finite, as shown in Figure 11.6.

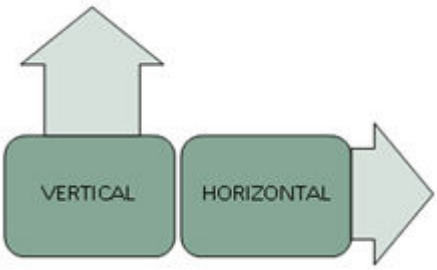


Figure 11.6: Horizontal and Vertical Scalability

More specifically, partitioning and replication are used to manage this. Additionally, by avoiding things like low-overhead server calls, it decreases the ACID guarantees.

No/Simpler Querying: With key-value stores, querying is really not possible except in very particular circumstances when it comes to querying keys, and

even then, it is not always practicable. Because there is just one request to read and one request to write, key-value makes it easier to manage situations like sessions, user profiles, shopping carts, and so on (due to the blob-like nature of how the data is stored). Similar to this, concurrency problems are simpler to manage because only one key needs to be resolved.

Mobility: Because key-value stores lack a query language, it is simple to move them from one system to another without modifying the architecture or the code. Thus, switching operating systems is less disruptive than switching relational databases.

When to Use Key-Value

Key-value stores excel in this area because traditional relational databases are not actually designed to manage a large number of read/write operations. Key-value can readily scale to thousands of users per second due to its scalability. Additionally, it can easily withstand lost storage or data because of the built-in redundancy.

As a result, key-value excels in the following instances:

- Profiles and user preferences
- Large-scale user session management
- Product suggestions (such as in eCommerce platforms)
- Delivery of personalized ads to users based on their data profiles
- Cache data for infrequently updated data

There are numerous other circumstances where key-value works nicely. For instance, because of its scalability, it frequently finds usage in Big data research. Similar to how it works for web applications, key-value is effective for organizing player sessions in MMOG (massively multiplayer online game) and other online games.

Key-Value Database Examples

Some key-value database models, for instance, save information to a solid-state drive (SSD), while others use random-access memory (RAM). We depend on key-value stores on a daily basis in our lives since they are some of the most popular and frequently used databases. The fact is that some of the most popular and commonly used databases are key-value stores.

Amazon DynamoDB is most likely the database that is used the most often for key-value storage. In point of fact, study into Amazon DynamoDB was the impetus for the rise in popularity of NoSQL.

Aerospike is a free and open-source database that was designed specifically for use with in-memory data storage.

Berkeley DB: Another free and open-source database, Berkeley DB is a high-performance framework for storing databases, despite the fact that it has a very simple interface.

Couchbase: Text searches and querying in a SQL-like format are both possible with Couchbase, which is an interesting feature.

Memcached not only saves cached data in RAM, which helps websites load more quickly, but it is also free and open source.

Riak was designed specifically for use in the app development process, and it plays well with other databases and app platforms.

Redis: A database that serves as both a memory cache and a message broker.

11.3.4 Document Based

A non-relational database that stores data as structured documents is known as a document database (also known as a NoSQL document store). Instead of using standard rows and columns, JSON format is a more recent technique to store data. An XML or JSON file, or a PDF, are all examples of documents. NoSQL is everywhere nowadays; just look at Twitter and its use of FlockDB or Amazon and their use of DynamoDB. Figure 11.7 shows the difference between the Relational and Document Store model.

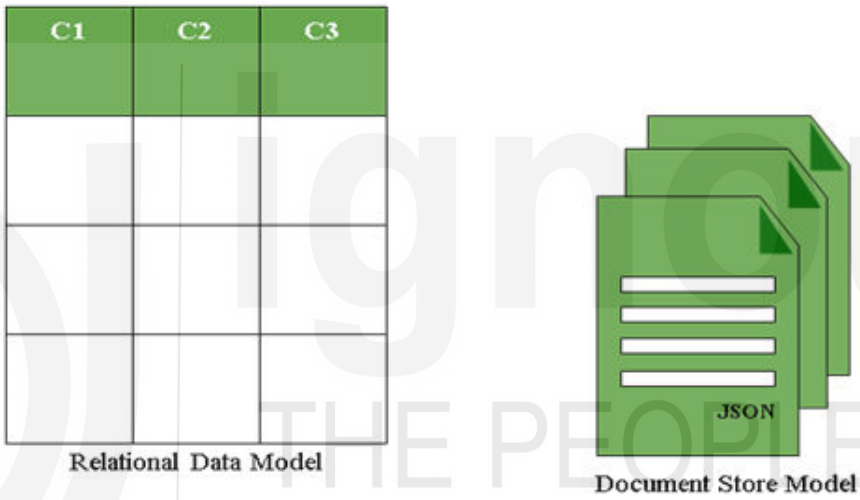


Figure 11.7: Relational Vs Document Store Model

In spite of the fact that there are a great deal of data models, each of which contains hundreds of databases, the one we are going to investigate today is called Document-store. One of the most common database models now in use, document-store functions in a manner that is somewhat similar to that of the key-value model in the sense that documents are saved together with particular keys that access the information. Figure 8 (a) shows the document that holds information about a book. This file is a JSON representation of a book's metadata, which includes the book's BookID, Title, Author, and Year and Figure 11.8 (b) shows the same metadata for Key value database.

A Document

{
“BookID”: “978-1449396091”,
“Title”: “DBMS”,
“Author”: “Raghu Ramakrishnan”,
“Year”: “2022”,
}

(a)

Key	Value
BookID	**978-1449396091**
Title	**DBMS**
Author	**Raghu Ramakrishnan**
Year	**2022**

(b)

Figure 11.8: Example of Document and Key value database

When to use a document database?

- When your application requires data that is not structured in a table format.
- When your application requires a large number of modest continuous reads and writes and all you require is quick in-memory access.
- When your application requires CRUD (Create, Read, Update, Delete) functionality.
- These are often adaptable and perform well when your application has to run across a broad range of access patterns and data kinds.

How does a Document Database Work?

It appears that document databases work under the assumption that any kind of information can be stored in a document. This suggests that you shouldn't have to worry about the database being unable to interpret any combination of data types. Naturally, in practice, most document databases continue to use some sort of schema with a predetermined structure and file format.

Document stores do not have the same foibles and limitations as SQL databases, which are both tubular and relational. This implies that using the information at hand is significantly simpler and running queries may also be much simpler. Ironically, you can execute the same types of operations in a document storage that you can in a SQL database, including removing, adding, and querying.

Each document requires a key of some kind, as was previously mentioned, and this key is given to it through a unique ID. This unique ID processed the document directly instead of being obtained column by column.

Document databases often have a lower level of security than SQL databases. As a result, you really need to think about database security, and utilizing Static Application Security Testing (SAST) is one approach to do so. SAST, examines the source code directly to hunt for flaws. Another option is to use DAST, a dynamic version that can aid in preventing NoSQL injections.

Document database advantages

One major benefit of document-store is that all of the data is stored in a single location, rather than being spread out over many interconnected databases. As a result, if you do not employ relational processes, you perform better than a SQL database.

- **Schema-less:** Because there are no constraints on the format and structure of data storage, they are particularly effective at keeping huge quantities of existing data.
- **Faster creation of document and maintenance:** The creation of a document is a fairly straightforward process, and apart from that, the upkeep requirements are virtually nonexistent.
- **Open formats**: It offers a relatively easy construction process that makes use of XML, JSON, and other formats.

- **Built-in versioning:** Because it contains built-in versioning, it means that when the documents expand in size, there is a possibility that they will also expand in complexity. Versioning makes conflicts less likely.

More precisely, document stores are excellent for the following applications because schema can be changed without any downtime or because you could not know future user needs:

- eCommerce giants (Like Amazon)
- Blogging platforms (such as Blogger, Tumblr)
- CMS (Content management systems) (Like WordPress, windows registry)
- Analytical platforms (such as Tableau, Oracle server)

Document databases' drawbacks

- **Weak Atomicity:** Multi-document ACID transactions are not supported. We will need to perform two different queries, one for each collection, in order to handle a change in the document data model involving two collections. This is where the atomicity criteria are violated.
- **Consistency Check Limitations:** A database performance issue may arise from searching for documents and collections that aren't linked to an author collection.
- **Security:** In today's world, many online apps do not have enough security, which in turn leads to the disclosure of critical data. Thus, web app vulnerabilities become a cause for concern.

Document databases examples

- One of the best NoSQL database engines is **MongoDB**, which is not only well-known but also uses JSON like format. It has its own query language.
- A search engine built on the document-store data architecture is **Elasticsearch**. Database searching and indexing may be accomplished using this straightforward and easy-to-learn tool.
- **CouchDB:** In addition to Ubuntu, it also works with the social networking site Facebook. It utilizes Javascript and is developed in the Erlang programming language.
- **BaseX** is a simple, open-source, XML-based DBM that makes use of Java.

☞Check Your Progress 2

1) How Does a Column Database Work? Discuss.

……

……

……

……

2) What are the different Graph Database Examples?

……

……

……..……..……..……

3) Explain document based NoSQL database.

……

……

……..……..……..……

11.4 INTRODUCTION to BIG DATA

In this modern era of Information and knowledge, terabytes of data are produced by a wide variety of sources every day. These sources include social media; the data of companies, including production data, customer data, financials etc.; the interactions of users; the data created by sensors and data produced by electronic devices such as mobile phones and automobiles, amongst others. This voluminous increase in data relates to the field of Big Data. This concept of "Big Data" applies to the rapidly expanding quantity of data that is being collected, and it is described primarily by having the following characteristics: volume, velocity, veracity, and variety.

The process of deriving useful information and insights from massive amounts of data is referred to as "Big data analytics". It requires the application of technologically advanced tools and procedures. "Big data architecture" refers to the pattern or design that describes how this data is received from many sources, processed for further ingestion, assessed, and eventually made available to the end-user.

The building blocks of Big data analytics are found in the Big data architecture. In most cases, the architecture components of Big data analytics consist of four logical levels or layers, as discussed below:

- **Big Data Source Layer:** A Big data environment may manage both batch processing and real-time processing of Big data sources, like data warehouses, relational database management systems, SaaS applications, and Internet of Things devices. This layer is referred to as the Big data sources layer.
- **The Management and Storage Layer:** It is responsible for receiving data from the source, converting that data into a format that the data analytics tool can understand, and storing the data in accordance with the format in which it was received.
- **Analysis Layer:** This layer is where the business intelligence that was acquired from the Big data storage layer is analysed.
- **The Consumption Layer:** It is responsible for collecting the findings from the Big Data Analysis Layer and delivering them to the appropriate Business Intelligence Layer.

The architecture components of Big data analytics typically include four logical layers or levels (discussed above), which perform fundamental processes as given below :

- **Connecting to Data Sources:** Connectors and adapters can connect to a wide range of storage systems, protocols, and networks. They can also connect to any type of data format.
- **Data Governance:** includes rules for privacy and security that work from the time data is taken in until it is processed, analysed, stored, and deleted.
- **Systems Management:** Modern Big data architectures are usually built on large-scale, highly scalable clusters that are spread out over a wide area. These architectures must be constantly monitored using central management consoles.
- **Protecting Quality of Service:** The Quality of Service framework helps define data quality, compliance policies, and how often and how much data should be taken in.

In this Unit, we will discuss Big data, its Characteristics and Applications. We will also discuss Big data architecture and related technologies such as Map Reduce, HDFS, Apache Hadoop and Apache YARN.

11.4.1 Big Data and Characteristics

To define the concept of Big data, we first revise the fundamentals of data, and thereafter we will extend our discussion to the advanced concepts of Big data. What exactly does "Data" mean? Data can be numbers or characters or symbols or any other form such as an image or signal, which can be processed by a computer or may be stored in the storage media or sent in the form of electrical impulses.

Now, we define - What is Big data? The term "Big data" refers to a collection of information that is not only extremely large in quantity but also growing at an exponential rate over the course of time. Because of the vast amount and the high level of complexity, none of the standard methods that are used for managing data can effectively store or deal with such data. Simply speaking, "Big data" refers to information that is kept in exceedingly vast amounts.

It is vital to have a thorough understanding of the characteristics of Big data in order to get a handle on how the concept of Big data works and how it can be used. The **characteristics of Big data** are:

1. **Volume:** The amount of information that you currently have access to is referred to as its volume. We measure the quantity of data that we have in our possession using the units of Gigabytes (GB), Terabytes (TB), Petabytes (PB), Exabytes (EB), Zettabytes (ZB), and Yottabytes (YB). As per the present trends, the volume of Big data is increasing exponentially.
2. **Velocity:** The speed at which data is generated and, therefore, should be processed is referred to as the "velocity" of that process. A high velocity is a fundamental requirement for the efficiency of any operation involving vast amounts of data.

3. **Variety:** The term "variety" relates to the numerous forms that Big data might take. When we talk about diversity, we mean the various kinds of Big data that are available. This is a major problem for the Big data sector since it has an impact on the efficiency of analysis. It is critical that you organise your data in order to manage its diversity effectively. You get a wide range of data from a variety of sources.

4. **Veracity:** The veracity of data relates to its accuracy. Lack of veracity can cause significant harm to the precision of your findings; it is one of the Big data characteristics that is considered to be of the utmost importance.

5. **Value:** The benefits that your organisation gains from utilising the data are referred to as "value." Are the results of Big data analysis in line with your company's goals? Is it helping your organisation expand in any way? In Big data, this is a vital component.

6. **Validity:** This characteristic of Big data relates to how valid and pertinent the facts are for the purpose they were designed for.

7. **Volatility:** Volatility relates to the life span of Big data. Some data items may be valid for a very short duration, like the sentiments of people.

8. **Variability:** The field of Big data is continually changing. Sometimes, the information you obtained from one source may not be the same as what you found today. This phenomenon is known as data variability, and it has an impact on the homogeneity of your data.

9. **Visualisation:** You may be able to express the insights that were generated by Big data by using visual representations such as charts and graphs. The insights that Big data specialists have to provide are increasingly being presented to audiences who are not technically oriented.

Some examples of Big data are given below:

- Big data can be illustrated by looking at the Stock Exchange, which creates approximately one terabyte of new trade data every single day.
- Social Media: According to various reports, more than 500 Gigabytes of fresh data are added to the databases of the social media website each and every day. The primary sources of this data are the uploading of photos and videos, sending and receiving of messages, posting of comments, and other activities.
- In just thirty minutes of flying time, a single jet engine is capable of producing more than ten Gigabytes of data. Because there are many thousands of flights each day, the amount of data generated can reach many Peta bytes.

A Big data system is primarily an application of Big data in an organisation for decision support. The following four components are required for a Big data system to function properly:

1. Ingestion (collecting and preparing the data)

2. Storage (storing the data)
3. Analysis (analysing the data)
4. Consumption (presenting and sharing the insights)

You need a component that is responsible for collecting the data and another that is responsible for storing it. In order for your Big data ecosystem to be complete, you require a system that analyses and reports the results of Big data analysis.

Big data may be the future of worldwide governance and businesses. It presents businesses with a great number of opportunities for improvement. The following are some of the more important ones:

1. Improved Decision Making
2. Data-driven Customer Service
3. Efficiency Optimisation
4. Real-time Decision Making

Based on these opportunities, this field of Big data has enormous applications in a variety of fields, and they are discussed in the subsequent section, i.e. section 5.4.

☞**Check Your Progress 3**

1. What does "Big Data" mean?

   ……

   ……

   ……

2. List the components required by a Big data system to function properly

   ……

   ……

   ……

3. Discuss the essential characteristics of Big data.

   ……

   ……

   ……

11.4.2 Big Data Applications

It is generally stated that Big data is one of the most valuable and effective fuel that can power the vast IT companies of the 21st century. Big data has applications in virtually every industry. Big data enables businesses to make better use of the vast quantities of data they generate and collect from a variety of sources. There are many different applications for Big data, which is why it is currently one of the skills that is most in demand. The following are some examples of important applications of Big data:

- One of the industries that makes the most use of Big data technology is the travel and tourism sector. It has made it possible for us to anticipate the need for travel facilities in a variety of locations, thereby improving business through dynamic pricing and several other factors.
- The social media sites produce a significant amount of data. Big data allows marketers to make greater use of the information provided by social media platforms, which results in improved promotional activities. It enables them to construct accurate client profiles, locate their target audience, and comprehend the criteria that are important to them.
- Big data technology is utilised widely within the financial and banking sectors. The use of Big data analytics can assist financial institutions in better comprehending the behaviours of their clients on the basis of the inputs obtained from their investment behaviours, shopping habits, reasons for investing, and personal or financial histories.
- The field of healthcare has already witnessed significant changes brought on by Big data application implementation. Individual patients can now receive individualised medical care tailored to their specific needs. This is possible due to the application of predictive analytics by medical professionals and personnel in the healthcare industry.
- The use of Big data in recommendation systems is another popular application of Big data. Big data allows businesses to recognise patterns of client behaviour in order to provide better and more individualised services to those customers.
- One of the most important industries that produce and use Big data is the telecommunications and multimedia business. Every day, Zettabytes of new data are created and managed.
- In addition, the Government and the military make heavy use of Big data technology. You may think of the quantity of data that the Government generates on its records, and in the military, a standard fighter jet plane needs to handle Petabytes of data while it is in the air.
- Companies are able to do predictive analysis because of the capabilities provided by Big data technologies. It lets businesses make more accurate predictions of the outcomes of processes and events, which in turn helps them reduce risk.
- Companies are able to develop insights that are more accurate because of Big data. They collect big data from a variety of sources, which allows them the capacity to use relevant data to generate insights that can be put into action. If a corporation has more accurate information, it will be able to make decisions that are more profitable and reduce risks.

11.4.3 Structured, Semi-Structured and Unstructured Data

The terms "structured," "unstructured," and "semi-structured" data are frequently brought up whenever we are having a discussion about data or analytics. These are the three varieties of data that are becoming increasingly important for many kinds of commercial applications, i.e. Structured, Semi-

Structured and Unstructured data. Structured data has been around for quite some time, and even today, conventional systems and reporting continue to rely on this type of data. In spite of this, there has been a rapid increase in the production of unstructured and semi-structured data sources over the course of the last several years. As a consequence of this, an increasing number of companies are aiming to include all three kinds of data in their business intelligence and analytics systems in order to take these systems to the next level.

In this section of the Unit, we will discuss data types used for understanding and implementing Big data, i.e. Structured, Semi Structured and Unstructured data.

Structured Data: Structured data is a term that refers to information that has been reformatted and reorganised in accordance with a data model that has been decided in advance. After mapping the raw data into the predesigned fields, the data can be extracted and read using SQL in a straightforward manner. Relational databases, which are characterised by their organisation of data into tables consisting of rows and columns, and the query language supported by them (SQL) provide the most apparent example of structured data.

This relational model of the data format decreases the quantity of duplicated information. On the other side, organised data is more dependent on one another and is less flexible than unstructured data. Both humans and machines are responsible for the generation of this kind of data.

There are several examples of structured data that are generated by machines, such as data from point-of-sale terminals (POS), such as quantity, barcodes, and statistics from blogs. In a similar vein, anybody who works with data has probably used spreadsheets at least once in their life. Spreadsheets are an example of a classic form of structured data that is generated by humans. Because of the way it is organised, structured data is simpler to examine compared to semi-structured data as well as unstructured data. Structured data can be analysed easily because it corresponds to data models that have already been defined. For example, you can organise structured data such as names of customers in alphabetical order, manage telephone numbers in the appropriate format, organise social security numbers in the correct format etc.

Unstructured data: Information displayed in its most unprocessed form is referred to as "unstructured data". It is really difficult to work with this data because it has a minimal structure, and the formatting is also very confusing. Unstructured data management can collect data from a variety of sources, such as postings on social media platforms, conversations, satellite imagery, data from the Internet of Things (IoT) sensor devices, emails, and presentations, and organise it in a storage system in a logical and specified manner.

Semi-Structured Data: There is a third kind of data that falls between structured and unstructured data, called semi-structured data or partially structured data. Your data sets might not always be structured or unstructured. One variety of such data is known as semi-structured data, and

it differs from unstructured data in that it possesses both consistent and definite qualities. It does not restrict itself to a fixed structure like those required for relational databases. Although data organisational features such as metadata or semantic tags are utilised with semi-structured data to make it more manageable, there is still a certain amount of unpredictability and inconsistency in the data.

The use of delimited files is one illustration of a data format that is only semi-structured. It has elements capable of separating the data hierarchies into their own distinct structures. In a similar manner, digital images have certain structural qualities that make them semi-structured, but the image itself does not have a pre-defined structure. If a picture is shot with a Smartphone, it will include certain structured properties like a device ID, a geotag, and a date and time stamp. After they have been saved, pictures can be organised further by affixing tags to them, such as "pet" or "dog," for example.

Due to the presence of one or more characteristics that allow for classification, unstructured data may occasionally be categorised as semi-structured data instead. To summarize, organizations need to analyze all three kinds of data in order to stay ahead of the competition and make the most of the knowledge they generate.

☞**Check Your Progress 4**

1. Give Five applications of Big data.

   ……

   ……

   ……

2. Compare Structured, Semi-Structured and Unstructured Data.

   ……

   ……

   ……

3. Discuss the role of Structured, Semi-Structured and Unstructured Data in decision-making.

   ……

   ……

   ……

11.4.4 Big Data Vs Data Warehouse

What should you buy for implementing an analytics system for your organization - a Big data solution or a data warehouse? A data warehouse and a Big data solution are quite similar in many respects. Both have the capacity to store a significant amount of information. When reporting, either option is acceptable to utilise. But the question that needs to be answered is whether they can truly be utilised as a substitute for each other. In order to understand this, you need to have a conceptual understanding of both, i.e., Big data and Data Warehouse.

The form of Big data that can be found in Hadoop, Cloudera, and other similar Big data platforms is the type that most people understand. The following are the working definitions of Big data solutions (You may find an accurate functioning definition on the websites of Cloudera or Horton Works.):

- A type of technology that is able to store extremely vast volumes of data.
- A technology capable of storing data in devices that are not pricey.
- A type of data storage technology in which the data is kept in an unstructured manner.

The data warehouse has had widespread usage in the past two decades, whereas Big data applications have seen a significant rise over the course of the past decade. Big data is expected to eventually replace traditional data warehousing, which is the first thing that comes to the mind of someone who is not technically very deep into these technologies. The fact that they share similarities is another argument favouring this simplistic way of thinking. Following are some of the similarities between the two:

- Both of these have a significant amount of information.
- Both may be utilised in the reporting process.
- Both are controlled by devices that save information electronically.

But despite this, "Big Data" and "Data Warehouse" are not the same thing at all. Why? In order to understand this, we need to recapitulate the basic concepts of Data Warehouse

A data warehouse is a subject-oriented, non-volatile, integrated, and time-variant collection of data generated to help the management make business decisions. A data warehouse provides a centralised, integrated, granular, and historical point of reference for all of the company's data in one convenient location.

So why do individuals demand a solution that involves Big data? People are looking for a solution for Big data because many organizations have a significant amount of data. In such companies, that data – if it is accessed and processed properly – can include a great deal of important information that can lead to better decisions, which in turn can lead to more revenue, more profitability, and more customers. Thus, a vast majority of businesses want to analyse Big data.

What are the reasons that people require a data warehouse? In order for people to make decisions based on accurate information, a data warehouse is required. You need data that is trustworthy, verifiable, and easily accessible to everyone if you want to have a thorough understanding of what is occurring within your company.

So, in short, we can understand the data Warehouse involves collecting data from a number of sources, processing it, and storing it in a repository where it can be analyzed and used for reporting reasons—the procedure described above results in the establishment of a data warehouse.

Comparison: Big Data Vs Data Warehouse

A Big data solution is an example of technology, whereas data warehousing is an example of an architectural framework. Technology can be loosely defined as any mechanism that is capable of storing and organising huge amounts of data. A facility that stores and organises data with the aim of guaranteeing a company's credibility and integrity is known as a data warehouse. Data warehouses are becoming increasingly common. When data is taken from a data warehouse, the person taking the data is aware that other individuals are using the same data for various purposes.

Table 11.7: Big Data Vs Data Warehouse

S.No.	Big Data	Data Warehouse
1.	Big data refers to a very large amount of data that can be utilised by many technologies.	The term "data warehouse" refers to the accumulation of historical data from a variety of business processes within an organisation.
2.	The term "Big Data" refers to a type of technology that can store and manage extensive amounts of data.	A data warehouse is a structure that is utilised in the process of data organisation.
3.	As input, it will accept data that is either structured, non-structured, or semi-structured.	The only type of data that can be used as input is structured data.
4.	The distributed file system is used for processing Big data.	Processing operations in the data warehouse are not carried out using a distributed file system.
5.	When it comes to retrieving data from Big data stores, Big data may not use SQL queries.	To get data from relational databases, we use structured query language, or SQL, in the data warehouse.
6.	Apache Hadoop is capable of managing massive amounts of data when handled appropriately.	It is difficult for a data warehouse to manage an extremely large volume of unstructured data.
7.	The modifications to the data that occur as a result of adding new data are saved in the form of a file.	Changes in the data that occur as a result of adding new data do not have an immediate and direct effect on the data warehouse.
8.	When opposed to data warehouses, Big data does not require the same level of effective management strategies.	As a result of the fact that the data is compiled from a variety of business divisions, the data warehouse calls for more effective management strategies.

Big data uses a Distributed File System. So, we discuss the Distributed File Systems in section 11.4.5 of this Unit.

☞**Check Your Progress 5**

1. Give similarities between Big Data and Data of a Data Warehouse.

   ……

   ……

   ……

2. Compare Big Data and Data Warehouse.

   ……

   ……

   ……

11.4.5 Distributed File System

A Distributed File System (DFS) is a file system that is distributed across many file servers at several locations (It's possible that these servers are located in various parts of the world). It allows programmers to access or store isolated files the same way as local files, allowing them to access files from any network or computer.

The primary objective of the Distributed File System (DFS) is to facilitate the sharing of users' data and resources between physically separate systems through the utilisation of a Common File System. A setup on DFS is defined as a group of workstations and mainframes that are linked together over a Local Area Network (LAN). Within the framework of the operating system, a DFS is carried out. A namespace is generated by DFS, and the clients are not exposed to the inner workings of this generation process.

The DFS is comprised of two distinct components, given below:

- **Location transparency:** A namespace component is a useful tool for achieving Location transparency.
- **Redundancy:** It is established by the utilisation of a file replication component.

Together, these components make it more likely that data will be accessible even if there is a breakdown or a severe load, as redundant data is stored in multiple locations that are shared under a single folder referred to as the "DFS root."

It is possible to use the namespace component of Distributed File System (DFS) without using the file replication component, and it is perfectly possible to use the file replication component of Distributed File System (DFS) between servers without using the namespace component. It is not essential to make use of both of the DFS components at the same time.

In order to gain more clarity on the Distributed File System (DFS), we need to study the Features of DFS:

1. **Transparency:**

   When discussing distributed systems, the term "transparency" refers to the process of hiding information about the separation of components from both the user and the application programmer. Transparency protects the integrity of the system. It makes the entire system a single system rather than a collection of distinct components working together. It is classified into the following types:

   a) **Structure Transparency –** There is no reason for the client to be aware of the number of file servers and storage devices and their geographical locations. But it is always recommended to provide a number of different file servers to improve performance, adaptability, and dependability.

   b) **Access Transparency –** The access process for locally stored files and remotely stored files should be identical. It should be possible for the file system to automatically locate the file that is being accessed and send that information to the client's side.

   c) **Naming transparency –** There should be no indication in the name of the file as to where the file is located in any way, shape, or form. Once a name has been assigned to the file, it should not have any changes made to it while it is being moved from one node to another.

   d) **Replication transparency –** If a file is copied on multiple nodes, the locations of the copies of the file as well as the copies themselves, should be hidden when moving from one node to another.

2. **User Mobility**: It will automatically transfer the user's home directory to the node where the user logs in.

3. **Performance:** The average amount of time it takes to get what the client wants is used to measure performance. This time includes CPU time, the time it takes to get to secondary storage, and the time it takes to get to the network. It would be best for the Distributed File System to work like both DFS and as well as a centralised file system (CFS), based on requirements.

4. **Simplicity and ease of use:** A file system should have a simple user interface and a small number of commands in the file.

5. **High availability:** If a link fails, a node fails, or a storage drive crashes, a Distributed File System should still be able to keep working. A distributed file system (DFS) that is both reliable and flexible should have different file servers that control different storage devices that are also separate.

6. **Scalability:** Adding new machines to the network or joining two networks is a common way to make the network bigger, so the distributed system will always grow over time. So, a good distributed file system (DFS) should be built so that it can grow quickly as the number

of nodes and users grows. As the number of nodes and users grows, the service should not deteriorate too much.

7. **High reliability:** A good distributed file system (DFS) should make data unlikely to be lost. That is, users should not feel they have to make backup copies of their files because the system is unreliable. Instead, a file system should make copies of important files in case the originals get lost. Stable storage is used by many file systems to make them very reliable.

8. **Data integrity:** A file system is often used by more than one person at a time. The file system must make sure that the data saved in a shared file stays the same. That is, a concurrency control method must be used to keep track of all the different users' requests to access the same file at the same time. Atomic transactions are high-level concurrency management that a file system often offers the users to keep their data safe.

9. **Security:** A distributed file system should be safe so that its users can trust that their data will be kept private. Security measures must be put in place to protect the information in the file system from unwanted and unauthorized access.

10. **Heterogeneity:** Because distributed systems are so big, there is no way to avoid heterogeneity. Users of heterogeneous distributed systems can choose to use different kinds of computers for different tasks.

The working of DFS can be put into practice in one of two different ways:

- **A standalone DFS namespace** is a namespace that only permits DFS roots to be used in it if they are located on the local computer and do not make use of Active Directory. Only the computers on which a Standalone DFS was initially created can be used to acquire the file system. It offers no fault tolerance, and it cannot be connected to any other DFS. Standalone DFS roots are not very common because of their limited advantage.
- **Domain-based DFS namespace** — This creates the DFS namespace root, which can be accessed at domainname>dfsroot> and stores the configuration of DFS in Active Directory.

Advantages of Distributed File System (DFS):

a) DFS permits many active users to access or store data simultaneously.

b) DFS allows the remote sharing of data

c) DFS facilitates file searching; it is faster to get to the data and makes the network run better.

d) DFS improves the ability to change the size of the data and the ability to exchange data.

e) DFS maintains data transparency and replicability; even if a server or disc fails, Distributed File System makes data available.

Disadvantages of Distributed File System (DFS):

a) Because all the nodes and connections in a Distributed File System need to be secured, we can safely claim that security is at risk.

b) During the process of moving from one node to another in the network, there is a chance that some of the messages and data will be lost.

c) While using a distributed file system, Difficulty is encountered when attempting to connect to a database

d) When compared to a system with a single user, a Distributed File System makes database management more difficult.

e) If every node in the network attempts to transfer data at the same time, there is a possibility that the network will become overloaded.

After understanding the Distributed File System (DFS), now it is time to understand Hadoop and HDFS (Hadoop Distributed File System). The discussion for HDFS (Hadoop Distributed File System) and MapReduce is given in the subsequent section, and the details of MapReduce are explicitly available in Section 11.4.6.

☞**Check Your Progress 6**

1. Describe the term "Distributed File System" in the context of Big data.

   ……

   ……

   ……

2. What is the primary objective of the Distributed File System?

   ……

   ……

   ……

3. Explain the components of the Distributed File System.

   ……

   ……

   ……

4. Discuss the features of Distributed File System

   ……

   ……

   ……

5. Discuss the number of ways through which the working of Distributed File System can be put into practice

   ……

   ……

6. Give advantages and disadvantages of Distributed File Systems.

   ……

   ……

   ……

11.4.6 HDFS and Mapreduce

Data is being produced at lightning speed in the modern period by a wide variety of sources, such as corporations, scientific data, emails, blogs, and other online platforms, amongst other places. It is necessary to implement data-intensive applications and storage clusters in order to analyze and manage the massive amount of data that is being generated and to gain information that is of use to the users. This is a prerequisite for being able to analyze and manage the data. This category of application is required to fulfill a number of characteristics, such as being able to tolerate errors, performing parallel processing, distributing data, maintaining load balance, being scalable, and having highly available operation. The MapReduce programming approach was developed by Google specifically for the purpose of resolving problems of this kind. Hadoop is another name for the open-source software project known as Apache Hadoop, which implements MapReduce technology.

Hadoop is a collection of several software services that are all freely accessible to the public and can be used in conjunction with one another. It offers a software framework for storing a huge amount of data in a variety of locations and for working with that data by utilising the MapReduce programming style. Both of these capabilities are accomplished through the use of software. Hadoop's essential components are the Hadoop Distributed File System (HDFS) and the MapReduce programming paradigm. Hadoop is operated using a programming style called MapReduce, and the Hadoop Distributed File System (HDFS) is the file system that is used to store data. The combination of HDFS and MapReduce creates an architecture that, in addition to being scalable and fault-tolerant, conceals all of the complexity associated with the analysis of Big data.

Hadoop Distributed File System abbreviated as HDFS, is a self-healing, distributed file system that offers dependable, scalable, and fault-tolerant data storage on commodity hardware. HDFS was developed by Hadoop. It collaborates closely with MapReduce by distributing storage and processing across huge clusters. This is accomplished by combining storage resources that, depending on the requests and queries being processed, are able to scale up or down. HDFS is not architecture-specific; it may read data in any format, including text, photos, videos, and so on, and it will automatically optimise it for high bandwidth streaming. The ability to tolerate errors is the most significant benefit of HDFS. The possibility of a catastrophic failure can be reduced by ensuring a quick data movement between the nodes and that Hadoop can continue to offer service even if individual nodes fail.

The information presented in this section can be used for understanding the development of large-scale distributed applications that can make use of the

computing capacity of several nodes in order to finish tasks that are data and computation intensive.

Let us discuss the Hadoop Architecture, which includes the following components:

- the MapReduce programming framework and
- the Hadoop distributed file system (HDFS)

An understanding of the Hadoop architecture requires the introduction of various daemons involved in its working. So, we need to understand the various daemons involved in Apache Hadoop, which includes the five daemons, three of which, i.e. NameNode, the DataNode, and the Secondary NameNode, relate to **HDFS** for the purpose of efficiently managing distributed storage, and The JobTracker and TaskTracker components that are utilised by the **MapReduce** engine are responsible for both job tracking and job execution respectively, and each of the mentioned daemons runs on their respective JVM

Firstly, we will discuss **HDFS**, a distributed file system comprising three nodes, i.e., the NameNode, the DataNode, and the Secondary NameNode, respectively.

1) **NameNode:** A single NameNode daemon operates on the master node. NameNode is responsible for storing and managing the metadata that is connected with the file system. This metadata is stored in a file that is known as fsimage. When a client requests to read from or write to a file, the metadata is held in a cache located within the main memory so that the client may access it more rapidly. The I/O tasks are completed by the slave DataNode daemons, which are directed in their actions by the NameNode.

   The NameNode is responsible for managing and directing how files are divided up into blocks, selecting which slave node should store these blocks, and monitoring the overall health and fitness of the distributed file system. In addition, it decides which slave node should store these blocks. Memory and input/output (I/O) are both put to intensive use in the operations that are carried out by the NameNode in the network.

2) **DataNode:** A DataNode daemon is present on each slave node, which is a component of the Hadoop cluster. DataNodes are the primary storage parts of HDFS. They are responsible for storing data blocks and catering to requests to read or write files that are stored on HDFS. These are under the authority of NameNode. Blocks that are kept in DataNodes are replicated in accordance with the configuration in order to guarantee both high availability and reliability. These duplicated blocks are dispersed around the cluster so that computation may take place more quickly.

3) **Secondary NameNode:** A backup for the NameNode is not provided by the Secondary NameNode. The job of the Secondary NameNode is to read the file system at regular intervals, log any changes that have

occurred, and then apply those changes to the fsimage file. This assists in updating NameNode so that it can start up more quickly the next time, as shown in Figure 3.

The HDFS layer is responsible for daemons that store data and information, such as NameNode and DataNode. The MapReduce layer is in charge of JobTracker and TaskTracker, which are seen in Figure 9 and are responsible for keeping track of and actually executing jobs.

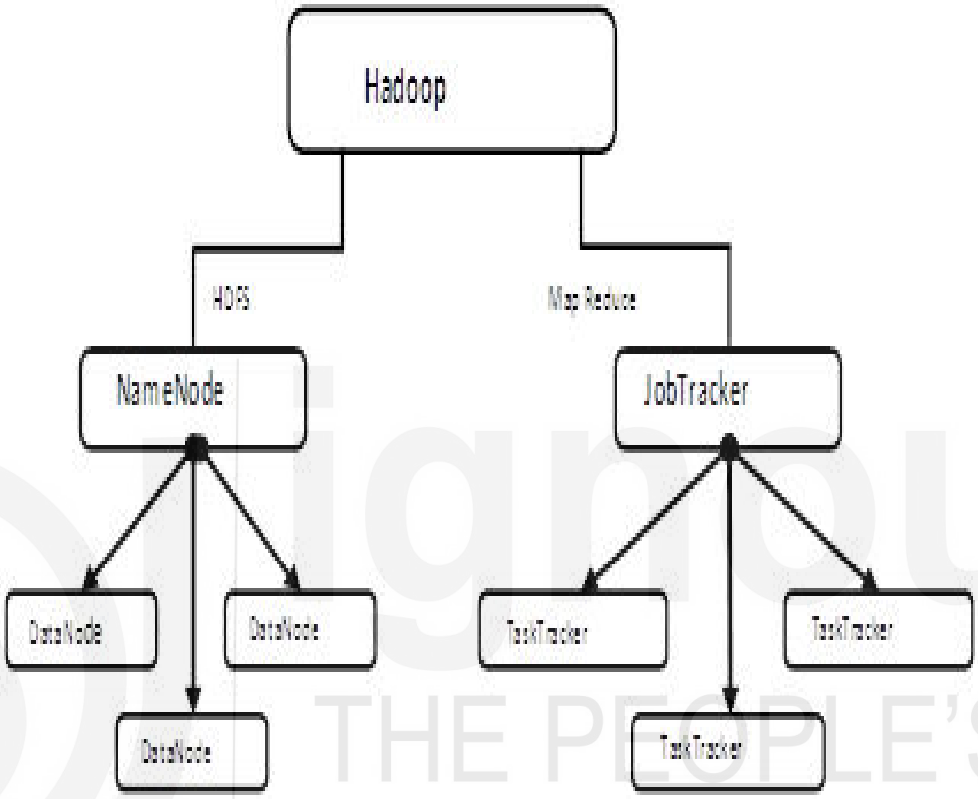


Figure 11.9: Hadoop Daemons

The master/slave architecture is utilised by HDFS, with the NameNode daemon and secondary NameNode both operating on the master node and the DataNode daemon running on each and every slave node, as depicted in Figure 11.10. The HDFS storage layer consists of three different daemons.

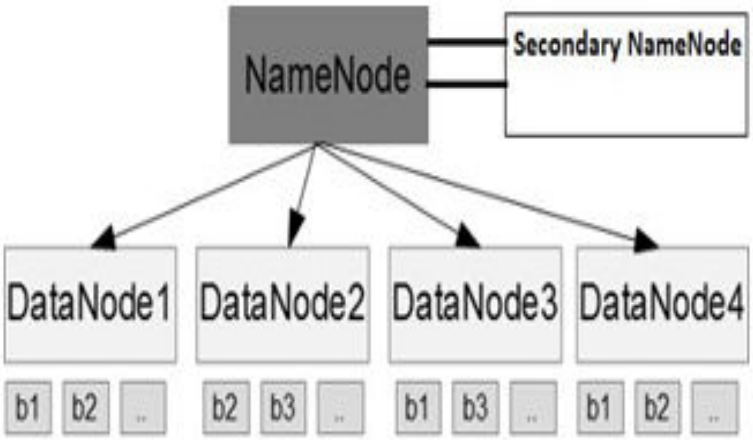


Figure 11.10: HDFS Architecture

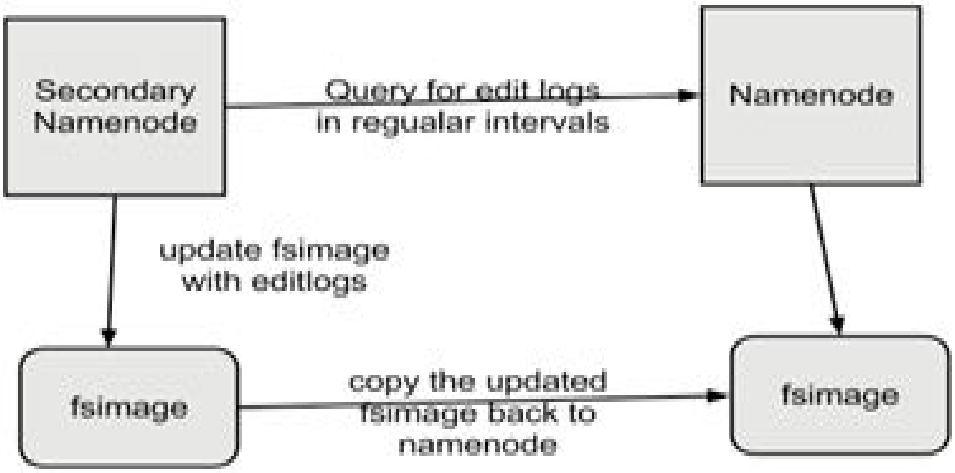


Figure 11.11: Task of Secondary name Node

A Hadoop cluster consists of several slave nodes in addition to a single master node. NameNode is the master daemon for the HDFS storage layer, while JobTracker is the master daemon for the MapReduce processing layer. Both of these daemons are executed by the master node. The remaining machines will be responsible for running the "slave" daemons, which include DataNode for the HDFS layer and TaskTracker for the MapReduce layer.

It is possible for a master node to take on the role of a slave at times, as it possesses the potential to flip roles. As a result, the master node is capable of operating both the master daemons and the slave daemons in addition to running the master daemons. The daemons that are operating on the master node are basically responsible for coordinating and administering the daemons that are running as slaves on all of the other nodes. This responsibility lies with the master node. These slave daemons are responsible for carrying out the tasks essential for the processing and storing of data.

Now we will discuss the MapReduce concept and the daemons involved with MapReduce, i.e. the JobTracker and TaskTracker components that are utilised by the **MapReduce** engine.

MapReduce can refer to either a programming methodology or a software framework. Both are utilised in Apache Hadoop. Hadoop MapReduce is a programming framework that is made available for creating applications that can process and analyse massive data sets in parallel on large multi-node clusters of commodity hardware in a manner that is scalable, reliable, and fault tolerant. The processing and analysis of data consist of two distinct stages known as the Map phase and the Reduce phase.

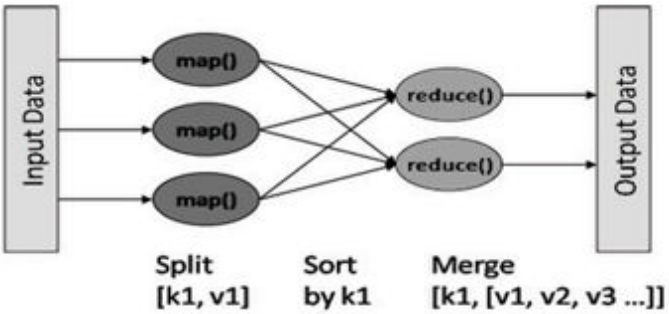


Figure 11.12: Map and Reduce Phase

During a MapReduce task, the input data is often segmented and partitioned into pieces before being broken up and processed in parallel by the "Map phase" and then by the "Reduce phase." The data that is generated by the Map phase is arranged and organised by the Hadoop architecture.

The result of the Map phase is sorted by the Hadoop framework, and this information is then sent as input to the Reduce phase in order to begin parallel reduce jobs (see Figure 12). These input and output files are saved in the system's file directory. The HDFS file system is the source of input datasets for the MapReduce framework by default. It is not required that tasks involving Map and Reduce proceed in a sequential way. This means that the Reduce jobs can begin as soon as any of the Map tasks finishes the work that has been given to them. It is also not required that all Map jobs be finished before any reduction tasks begin their work. MapReduce operates on key-value pairs as its data structure. In theory, a MapReduce job will accept a data set as an input in the form of a key-value pair, and it will only produce output in the form of a key-value pair after processing the data set through MapReduce stages. As can be seen in Figure 13, the output of the Map phase, which is referred to as the intermediate results, is sent on to the Reduce phase as an input.

Figure 11.13: MapReduce key-value pairs

On the same lines as HDFS, MapReduce also makes use of a master/slave architecture. As illustrated in Figure 14, the JobTracker daemon resides on the master node, while the TaskTracker daemon resides on each of the slave nodes.

The MapReduce processing layer consists of two different daemons, i.e. JobTracker and TaskTracker; their roles are discussed below:

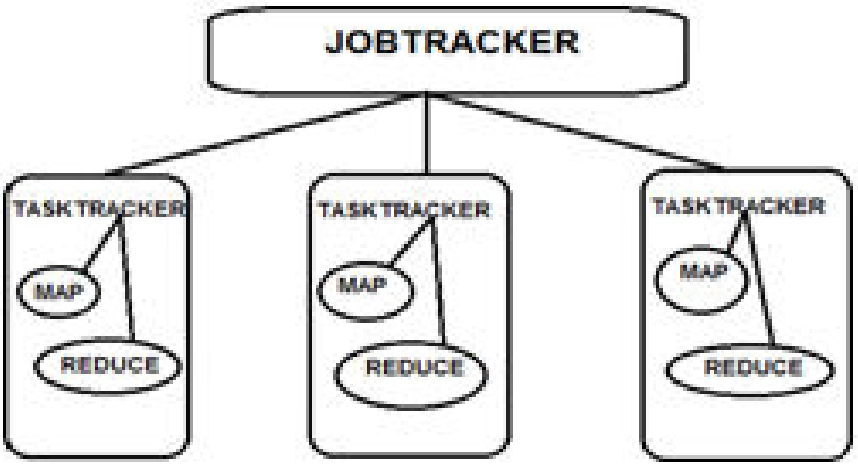


Figure 11.14: JobTracker and TaskTracker

4) **JobTracker:** The JobTracker service is responsible for monitoring MapReduce tasks that are carried out on slave nodes and are hosted on the master node. The job is sent to the JobTracker by the user through their interaction with the Master node. The next thing that happens is that JobTracker queries NameNode to find out the precise location of the data in HDFS that needs to be processed. JobTracker searches for TaskTracker on slave nodes and then sends the jobs to be processed to TaskTracker on those nodes. The TaskTracker will occasionally send a heartbeat message back to the JobTracker to verify that the TaskTracker on a specific slave node is still functioning and working on the task that has been assigned to it. If the heartbeat message is not received within the allotted amount of time, the TaskTracker running on that particular slave node is deemed to be inoperable, and the work that was assigned to it is moved to another TaskTracker to be scheduled. The combination of JobTracker and TaskTracker is referred to as the MapReduce engine. If there is a problem with the JobTracker component of the Hadoop MapReduce service, all active jobs will be halted until the problem is resolved.

5) **TaskTracker:** On each slave node that makes up a cluster, a TaskTracker daemon is executed. It works to complete MapReduce tasks after accepting jobs from the JobTracker. The capabilities of a node determine the total amount of "task slots" that are available in each TaskTracker. Through the use of the heartbeat protocol, JobTracker is able to determine the number of "task slots" that are accessible in TaskTracker on a slave node. It is the responsibility of JobTracker to assign suitable work to the relevant TaskTrackers, and the number of open task slots will determine how many jobs can be assigned. On every slave node, TaskTracker is the master controller of how each MapReduce action is carried out. Even though there is only one TaskTracker for each slave node, each TaskTracker has the ability to start multiple JVMs so that MapReduce tasks can be completed simultaneously. JobTracker, the master node, receives a "heartbeat" message regularly from each slave node's TaskTracker. This message confirms to JobTracker that TaskTracker is still functioning.

In short, it can be said that, when it comes to processing massive and unstructured data volumes, the Hadoop MapReduce computing paradigm and HDFS are becoming increasingly popular choices. While masking the complexity of deploying, configuring, and executing the software components in the public or private cloud, Hadoop makes it possible to interface with the MapReduce programming model. Users are able to establish clusters of commodity servers with the help of Hadoop. MapReduce has been modelled as an independent platform-as-a-service layer that cloud providers can utilise to meet a variety of different requirements. Users are also given the ability to comprehend the data processing and analysis processes.

11.4.7 Apache Hadoop 1 And 2 (YARN)

Apache Hadoop 1.x suffered from a number of architectural flaws, the most notable of which was a decline in the overall performance of the system. Actually, the cause of the problem was the excessive strain that was placed on the MapReduce component of Hadoop-1. MapReduce was responsible for both application management and resource management in Hadoop 1; however, with Hadoop 2, application management is now handled by a new component known as YARN, which takes over responsibility for resource management (yet another resource negotiator). As a result, MapReduce is in charge of managing application management in Hadoop 2, whereas YARN is in charge of managing the resources. With the release of Hadoop 2, YARN has added two additional daemons. These are-

- Resource Manager
- Node Manager

These two new Hadoop 2 daemons have replaced the JobTracker and TaskTracker in Hadoop 1.

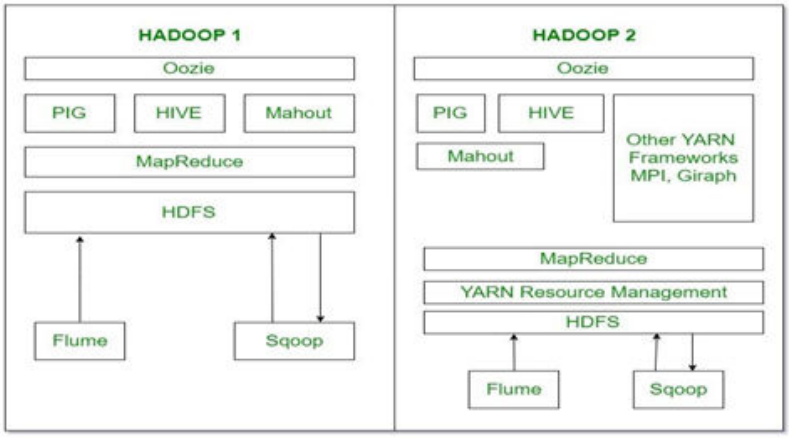
In addition, there is only one NameNode in the Hadoop 1.x cluster, which implies that it serves as the single point of failure for the entire system. Hadoop 2.x's architecture, on the other hand, incorporates both Active and Passive NameNodes in several places. In the event that the active NameNode is unable to complete the tasks assigned to it, the passive NameNode will step in and assume responsibility. This element is directly responsible for Hadoop 2.x's high availability, which is one of the most important characteristics of the new version.

The processing of data is problematic in Hadoop 1.x; however, Hadoop 2. x's YARN provides a centralised resource management that allows sharing of sharable resources. This makes it possible for multiple applications included within Hadoop to execute concurrently while sharing a single resource.

The comparison between Apache Hadoop-1 and Hadoop-2 is consolidated below:

Comparison Parameter	Apache Hadoop 1	Apache Hadoop 2
Components	Has MapReduce	Has YARN(Yet Another Resource Negotiator) and MapReduce version 2.
Daemons	1) NameNode, 2) DataNode, 3) Secondary NameNode, 4) JobTracker, 5) TaskTracker	1) NameNode, 2) DataNode, 3) Secondary NameNode, 4) Resource Manager, 5) Node Manager

Working	Both HDFS and MapReduce are the components of Hadoop 1 HDFS is responsible for data storage, and MapReduce, which sits above HDFS, manages resources and performs data processing; as a result of this, the system performance is heavily impacted.	The Hadoop Distributed File System, or HDFS, is utilised once again for storage in Hadoop 2, while YARN, a Resource Management system, is layered over HDFS to perform its duties. In essence, it distributes the available resources and ensures that everything continues to function normally. YARN actually shares the load of MapReduce. Hence the system performance improves
Limitation	The architecture of Hadoop 1 is known as Master-Slave architecture. One master rules over a large number of slaves in this arrangement. In the event that the master node experienced a catastrophic failure, the cluster would be wiped out regardless of the quality of the slave nodes. Again, in order to create that cluster, you have to copy system files, picture files, and so on onto another machine, which takes an excessive amount of time and is something that enterprises just cannot accept.	Hadoop 2 utilises the same Master-Slave architecture as its predecessor. However, this is made up of a number of different masters (also known as active namenodes and standby namenodes) and a number of different slaves. In the event that this master node suffers a crash, the standby master node will take control of the network. You are able to create a wide variety of different active-standby node combinations. As a result, the issue of having a single point of failure will be resolved with Hadoop 2.

EcoSystem	 Oozie is a simplified version of a workflow scheduler. It is responsible for determining the specific times at which jobs will run based on the dependencies between them. Pig, Hive, and Mahout are examples of data processing tools that sit above Hadoop and execute their job there. Sqoop is an application that can import and export structured data. Using a SQL database, it is possible to immediately import and export data into HDFS. Flume is a tool that is used to import and export streaming data as well as unstructured data.	
Support	No Support for Microsoft Windows is available for Hadoop 1.x	Support of Microsoft Windows available for Hadoop 2.x

☞**Check Your Progress 7**

1. What is Hadoop? Discuss the components of Hadoop.

   ……

   ……

   ……

2. Discuss the role of HDFS and MapReduce in Hadoop Architecture

   ……

   ……

   ……

3. Discuss the relevance of various nodes in HDFS architecture

   ……

   ……

   ……

4. Explain how the master/slave process works in HDFS architecture.

   ……

   ……

……

5. What do you understand by the Map phase and reduce phase in MapReduce architecture?

……

……

……

6. List and explain the various daemons involved in the functioning of MapReduce.

……

……

……

7. Differentiate between Apache Hadoop-1 and Hadoop-2.

……

……

……

11.5 SUMMARY

This Unit provided a foundational understanding of NoSQL databases, detailing the various types such as column-based, graph-based, key-value, and document-oriented databases. It highlighted NoSQL's growing adoption across industries due to its ability to support mission-critical applications, reduce data dispersion, lower ownership costs, and mitigate risks—though selecting the right database platform remains a complex task.

Each NoSQL model has its strengths and limitations. Column-store databases, while powerful, may face consistency issues due to the way data is written to disk. Graph databases excel in delivering intuitive, high-performance content structures, particularly useful in applications like social networks. Key-value stores offer simplicity and speed but may fall short in handling complex operations such as financial transactions. Document databases are widely used for their flexibility and support for diverse data types, making them ideal for analytical applications. Combining relational and non-relational systems can lead to more efficient data processing pipelines.

In addition, the Unit covered the key concepts of Big Data, including its defining characteristics—volume, velocity, variety, and veracity—and its practical applications. It also explained the differences between structured, semi-structured, and unstructured data, and compared Big Data systems with traditional data warehouses. Important infrastructure components such as distributed file systems were discussed, culminating in a comparison of technologies like MapReduce, HDFS, and Apache Hadoop (versions 1 and 2 with YARN).

11.6 SOLUTIONS/ANSWERS

☞Check Your Progress 1

1. What is NoSQL? NoSQL (Not Only SQL) is a type of database designed to store and manage large volumes of unstructured, semi-structured, or structured data. It offers flexible data models and is well-suited for distributed systems, real-time web apps, and big data use cases.
2. What are the features of NoSQL databases?
   - Schema-less or flexible data models
   - Horizontal scalability
   - High performance for read/write operations
   - Support for unstructured and semi-structured data
   - Designed for distributed and cloud-based systems
   - Includes various types: key-value, document, column, and graph databases
3. Differentiate between NoSQL and SQL:

Feature	SQL (Relational)	NoSQL (Non-relational)
Data Model	Table-based with fixed schema	Flexible schema (key-value, document, etc.)
Scalability	Vertical (scale-up)	Horizontal (scale-out)
Query Language	Structured Query Language (SQL)	Varies (e.g., JSON queries, APIs)
Data Integrity	Strong consistency (ACID)	Often eventual consistency (BASE)
Best For	Structured data and complex queries	Large-scale, unstructured, real-time data

☞Check Your Progress 2

1. How Does a Column Database Work? A column database (or column-family database) stores data in columns instead of rows. Each column stores values of a single attribute for multiple records, which allows for faster querying and better compression—especially when dealing with large datasets where only certain attributes are needed. It is ideal for analytical workloads and big data applications. Examples include Apache Cassandra and HBase.
2. What are the Different Graph Database Examples? Graph databases store data in nodes (entities) and edges (relationships), making them suitable for relationship-heavy data. Popular graph database examples include:
   - Neo4j

- Amazon Neptune
- ArangoDB
- OrientDB
- Microsoft Azure Cosmos DB (with Gremlin API)

3. Explain Document-Based NoSQL Database: A document-based NoSQL database stores data as documents, typically in JSON, BSON, or XML format. Each document is self-contained and can have a flexible schema, making it suitable for storing complex, hierarchical data. These databases are ideal for content management systems, catalogs, and real-time analytics. Examples include MongoDB and CouchDB.

☞Check Your Progress 3

1. What does "Big Data" mean? Big Data refers to extremely large and complex data sets generated from various sources at high speed. These datasets are too vast to be processed using traditional data management tools and require advanced methods for storage, processing, and analysis to extract meaningful insights.
2. List the components required by a Big Data system to function properly:
   - Data Sources Layer: Collects data from various inputs (e.g., sensors, web logs, social media, IoT devices).
   - Management and Storage Layer: Stores and manages the incoming data in a suitable format.
   - Analysis Layer: Processes and analyzes the stored data to derive insights.
   - Consumption Layer: Delivers the analysis results to users or systems.
   - Data Ingestion Tools: For collecting and importing data (e.g., Apache Flume, Kafka).
   - Distributed File Systems: For storing large volumes of data (e.g., HDFS).
   - Processing Frameworks: For computation and analytics (e.g., MapReduce, Spark).
   - Data Governance and Security Tools: To ensure data quality, privacy, and compliance.
3. Discuss the essential characteristics of Big Data: Big Data is commonly defined by the 4 Vs:
   - Volume: The vast amount of data generated every second.
   - Velocity: The speed at which data is produced and needs to be processed.
   - Variety: The different types and formats of data (structured, semi-structured, unstructured).
   - Veracity: The accuracy, reliability, and trustworthiness of the data.

☞Check Your Progress 4

1. Five Applications of Big Data:
   - **Healthcare:** Predictive analytics for disease detection and patient care.
   - **Retail:** Personalized recommendations and customer behavior analysis.
   - **Finance:** Fraud detection and algorithmic trading.
   - **Manufacturing:** Predictive maintenance and supply chain optimization.
   - **Social Media:** Sentiment analysis and trend prediction.
2. Comparison of Structured, Semi-Structured, and Unstructured Data:

Feature	**Structured Data**	**Semi-Structured Data**	**Unstructured Data**
Format	Fixed schema (tables, rows, columns)	Flexible schema (tags, keys)	No predefined format
Examples	SQL databases, spreadsheets	XML, JSON, NoSQL databases	Emails, images, videos, social media posts
Storage	Relational databases	NoSQL databases, XML/JSON files	Data lakes, file systems
Ease of Analysis	Easy	Moderate	Difficult

3. Role of Data Types in Decision-Making:
   - **Structured Data:** Provides quantifiable insights for quick, accurate decisions (e.g., sales trends).
   - **Semi-Structured Data:** Adds context and flexibility, enabling better understanding of diverse sources (e.g., customer reviews).
   - **Unstructured Data:** Offers deep insights through content analysis, especially in areas like sentiment, image recognition, and behavior patterns.

☞Check Your Progress 5

1. Similarities between Big Data and Data Warehouse:
   - Data Storage: Both store large volumes of data.
   - Data Analysis: Used for analyzing data to support decision-making.
   - Data Integration: Combine data from multiple sources.
   - Historical Data: Maintain historical records for trend analysis.
   - Business Intelligence: Support reporting and insights generation.

2. Comparison of Big Data and Data Warehouse:

Feature	Big Data	Data Warehouse
Data Type	Structured, semi-structured, unstructured	Mainly structured
Volume & Variety	Very large, diverse formats	Moderate to large, structured format
Processing	Real-time or batch (via tools like Hadoop, Spark)	Batch processing (ETL processes)
Scalability	Highly scalable (distributed systems)	Limited scalability (traditional systems)
Use Case	Advanced analytics, machine learning	Reporting, OLAP, business intelligence

☞Check Your Progress 6

1. Describe the term "Distributed File System" in the context of Big Data: A Distributed File System (DFS) is a storage system that allows data to be stored across multiple machines or nodes, enabling scalable and fault-tolerant access to large datasets, commonly used in Big Data platforms like Hadoop.
2. What is the primary objective of the Distributed File System? The primary objective is to store and manage large volumes of data across multiple systems while ensuring high availability, fault tolerance, and efficient data access.
3. Explain the components of the Distributed File System:
   - NameNode (or Metadata Server): Manages metadata and file directory structure.
   - DataNodes (or Storage Nodes): Store actual data blocks.
   - Client: Accesses and interacts with the DFS for read/write operations.
4. Features of Distributed File System:
   - Scalability: Can handle large data volumes.
   - Fault Tolerance: Automatically recovers from node failures.
   - High Availability: Ensures continuous access to data.
   - Data Replication: Copies data across nodes for safety.
   - Transparency: Hides complexity from users.
5. Ways to Implement Distributed File System in Practice:
   - HDFS (Hadoop Distributed File System)
   - Google File System (GFS)
   - Amazon S3 (for cloud-based DFS-like storage)
   - Ceph and GlusterFS (open-source DFS solutions)

- Microsoft DFS (for enterprise file sharing)

6. Advantages and Disadvantages of Distributed File Systems:

   Advantages:

   - High fault tolerance
   - Scalability
   - Efficient data access
   - Data redundancy
   - Cost-effective for large data

   Disadvantages:

   - Complex setup and maintenance
   - Network dependency
   - Latency issues during access
   - Security concerns across nodes
   - Data consistency challenges

☞Check Your Progress 7

1. What is Hadoop? Discuss the components of Hadoop: Hadoop is an open-source framework for processing and storing large-scale data across distributed systems. Main components:
   - HDFS (Hadoop Distributed File System): For data storage.
   - MapReduce: For data processing.
   - YARN (Yet Another Resource Negotiator): Manages resources and job scheduling.
   - Hadoop Common: Utilities and libraries used by other Hadoop modules.
2. Role of HDFS and MapReduce in Hadoop Architecture:
   - HDFS: Provides scalable and fault-tolerant storage by distributing data across nodes.
   - MapReduce: Processes large data sets in parallel by dividing tasks into map and reduce phases.
3. Relevance of Various Nodes in HDFS Architecture:
   - NameNode (Master): Manages metadata and file directory structure.
   - DataNode (Slave): Stores actual data blocks and handles read/write requests.
   - Secondary NameNode: Periodically backs up NameNode metadata (not a standby node).
4. Master/Slave Process in HDFS Architecture:
   - The NameNode (master) maintains metadata, while multiple DataNodes (slaves) store actual data.
   - Clients request file operations from the NameNode, which directs

them to appropriate DataNodes.

5. Map Phase and Reduce Phase in MapReduce Architecture:
   - Map Phase: Splits input data into key-value pairs and processes them in parallel.
   - Reduce Phase: Aggregates and summarizes the intermediate key-value pairs to produce the final output.
6. Daemons in MapReduce Functioning:
   - JobTracker (Hadoop 1): Manages jobs and resource scheduling.
   - TaskTracker (Hadoop 1): Executes map and reduce tasks on nodes.
   - ResourceManager (Hadoop 2/YARN): Manages cluster resources.
   - NodeManager (Hadoop 2): Manages resources and monitors tasks on individual nodes.
   - ApplicationMaster (Hadoop 2): Manages execution of a single job.
7. Difference Between Hadoop-1 and Hadoop-2:

Feature	**Hadoop-1**	**Hadoop-2**
Resource Management	JobTracker	YARN (Resource Manager + Application Master)
Scalability	Limited	Highly scalable
Fault Tolerance	Single JobTracker (bottleneck)	Better fault tolerance with YARN
Data Processing	Only MapReduce	Supports other models (Spark, Tez, etc.)
Cluster Utilization	Less efficient	Improved resource utilization

11.7 FURTHER READINGS

1) Next Generation Databases: NoSQL and Big Data 1st ed. Edition, G. Harrison, Apress, December 26, 2015.

2) Shashank Tiwari, Professional NoSQL, 1st Edition, Wrox, September 2011.

3) https://www.kdnuggets.com/

4) C. Lam, "Introducing Hadoop", in Hadoop in Action, MANNING, 2011.

5) D. Borthakur, "The Hadoop distributed file system: architecture and design," Hadoop Project Website [online]. Available: http://hadoop.apache.org/ core/docs/current/hdfs design.pdf

UNIT 12 MINING DATA STREAMS

Structure

12.0 Introduction

12.1 Objectives

12.2 Data Streams

12.2.1 Model for Data Stream Processing

12.3 Data Stream Management

12.3.1 Queries of Data Stream

12.3.2 Examples of Data Stream and Queries

12.3.3 Issues and Challenges of Data Stream

12.4 Summary

12.5 Answers

12.6 References/Further Readings

12.0 INTRODUCTION

In today's data-driven world, vast amounts of data are generated continuously from various sources such as sensors, online transactions, social media platforms, and network monitoring systems. This type of continuously flowing data is referred to as a **data stream**. Unlike traditional datasets that are static and stored in databases, data streams are dynamic, unbounded, and arrive at high speed, requiring immediate processing and analysis.

To understand and work with data streams effectively, it's important to explore a **model for data stream processing**, which outlines how incoming data can be captured, processed, and analyzed in real time. Such models typically focus on minimizing memory usage, ensuring quick computation, and providing approximate but useful results.

Managing these data streams is equally important and is addressed under the concept of **Data Stream Management**. In this context, learners should understand how to write **queries for data streams**, which differ from traditional SQL queries because of the continuous and time-sensitive nature of the data. Real-life **examples of data streams and related queries** help illustrate how theoretical concepts are applied in practical scenarios like stock market analysis or network traffic monitoring.

However, working with data streams also brings several **issues and challenges**. These include coping with limited memory, handling out-of-order data, processing at high speeds, and generating timely insights. Understanding these challenges is essential for designing efficient data stream processing systems.

By studying the concepts of data stream models, management techniques, and the practical challenges involved, learners can develop the skills necessary to work with real-time data in various modern applications.

This unit explains the characteristics and models of data stream processing. It also identifies the challenges or stream processing and introduces you to various techniques for processing of data streams. You may refer to further readings of this Unit for more details on data streams and data stream processing.

12.1 OBJECTIVES

After going through this Unit, you would be able to:

- Define the characteristics of data streams
- Discuss models of data stream processing
- Explain the uses of data streams
- Illustrate the example of data stream queries
- Address Issues and Challenges of Data Stream

12.2 DATA STREAMS

In many traditional applications, data is stored in a **database** or a **distributed file system**, allowing users to access the same data multiple times whenever needed. This works well when the data is relatively stable and can be stored for long-term use. However, there are certain real-world scenarios where this approach is not practical. For example, in **real-time applications** like data received from **weather satellites**, the volume and speed of incoming data are so high that storing all of it in a database becomes either **impractical or inefficient**.

In some cases, the data might not be stored at all, or if it is, the **database may be so large** that retrieving information quickly enough to make timely decisions becomes impossible. This is where traditional query methods fall short. To deal with such situations, we need **new techniques** that allow us to process and analyze data on the fly.

To address this, we use **approximation techniques** and **data compression methods** that help us extract useful insights from the data without needing to store or scan the entire dataset. These methods aim to answer the most important queries **quickly and efficiently**, often with a trade-off between accuracy and performance.

This brings us to the concept of **data stream mining**—a process of extracting meaningful knowledge from **continuous, high-speed, and volatile data streams** in **real time**. The term "volatile" here means that the data is constantly changing and evolving, and due to resource constraints (like memory or processing power), we cannot store it all permanently. Instead, we must process it as it arrives, making **real-time analysis** a necessity.

So, how can we analyze such fast and endless streams of data effectively? To explore this question, the next section introduces the **models for data stream processing**, which provide structured approaches for handling, analyzing, and querying data streams under real-world constraints.

12.2.1 Model for Data Stream Processing

Let's Understanding the Model for Data Stream Processing, when working with traditional datasets, we usually have the entire data available beforehand, stored in a database or a file system. We can query it multiple times, sort it, scan it repeatedly, or even join it with other data. But in data stream processing, things are different. The data arrives continuously, often at high speed, and we only get one chance to process it. This is because storing the entire stream is not practical due to limited memory, time, and processing power.

To deal with this challenge, researchers and developers have come up with specific models for processing data streams efficiently. Each model provides a strategy to analyse and summarize the incoming data using minimal resources, often with a trade-off in accuracy. Let's explore some of the commonly used models with easy-to-understand examples.

1. **Sliding Window Model :** In this model, we focus only on the **most recent data** in the stream. Instead of processing the entire stream, we maintain a **"window"** of data (either time-based or count-based), and continuously update our analysis as new data enters and old data exits the window.

   Example:
   Imagine monitoring **temperature readings** from a sensor every second. Using a sliding window of the **last 10 minutes**, we can calculate the **average temperature** in that window. As new readings arrive, old ones are discarded, and the average is updated in real time.

2. **Damped Window Model (Exponential Decay Model):** In this model, **recent data is given more importance** than older data. Rather than maintaining a fixed-size window, we use a mathematical function (usually exponential decay) to reduce the weight of older data gradually.

   Example:
   In **stock market trend analysis**, recent price changes are more relevant than older ones. Using the damped window model, we can calculate a **weighted average price**, where the latest data points have the highest influence on the result.

3. **Sampling Model:** Here, instead of analysing the entire stream, we **randomly select a subset** (sample) of the data and use it to make estimations. This reduces memory and computation requirements significantly. **Example:** If we are monitoring **website traffic** from millions of users per day, we might randomly sample 1% of user activity to estimate **popular pages or average session time**, instead of analysing all traffic.

4. **Sketching Model:** This model uses compact data structures called **sketches** to summarize large data streams. Sketches provide **approximate answers** to queries like frequency count or cardinality (number of unique items), using very little memory. **Example:** A network router might use sketching to **count the number of unique IP**

addresses accessing it in real time, without storing every single IP. One popular sketching technique is the **Count-Min Sketch**.

5. **Aggregation Model:** In this model, the data stream is processed using **aggregate functions** like sum, count, average, min, or max. This model is useful for summarizing the data continuously. **Example:** A smart energy meter may use aggregation to calculate the **total electricity consumption** every hour by continuously adding up the readings from the data stream.

These models help us handle the unique challenges of data stream processing. Depending on the application—whether it's real-time weather monitoring, network traffic analysis, or sensor-based automation—we can choose the most suitable model to process data efficiently. As a learner, understanding these models equips you to design systems that can make fast, resource-efficient decisions from rapidly changing data.

You may use any model for data stream processioning, but how are data streams managed? Next section discusses the data stream management system.

12.3 DATA STREAM MANAGEMENT

Managing data streams is a critical task in today's data-centric world, especially when dealing with real-time applications like traffic monitoring, stock trading, or weather forecasting. Unlike traditional data stored in databases, data streams are continuous, fast, and often infinite. This means we usually don't have the luxury to store all the incoming data or process it multiple times. So, how do we effectively manage such data? The answer lies in using smart models that help us process, analyze, and summarize data streams efficiently, often with limited memory and computing resources.

One useful approach is the **sliding window model**, where only the most recent portion of the data stream is kept in memory. For example, if we are monitoring traffic at a busy intersection, we might only care about vehicle counts in the last 5 minutes, rather than the entire day. As new data arrives, old data is removed from the window, allowing us to continuously analyze the latest situation without overloading the system.

Another important model is the **damped window model**, which helps when we still want to consider older data, but give more importance to recent values. This is especially helpful in applications like fraud detection, where past user behavior provides useful context, but recent actions are more indicative of suspicious activity. In this model, the influence of older data gradually fades, allowing the system to focus on current trends while still acknowledging historical patterns.

In cases where the data stream is too fast or too large to process entirely, we can use the **sampling model**. This approach involves selecting a small, random subset of data to analyze, which still gives us a fairly accurate understanding of the overall trend. For example, a video streaming service may sample a small percentage of user viewing data to identify popular content, rather than analyzing every single view.

For even more compact summaries, the **sketching model** comes in handy. Sketching uses clever data structures to keep track of key statistics—like the number of unique users or the most frequent search terms—without storing the entire stream. This is very useful in network monitoring or cybersecurity, where we might need to detect abnormal behavior in real time using only minimal memory.

Finally, the **aggregation model** is ideal for continuously calculating values like totals, averages, or maximums. For instance, an electricity provider may use this model to calculate the total energy consumed by each household every hour. The aggregation is done on-the-fly, which helps make timely decisions such as load balancing or dynamic billing.

So, we learned that, managing data streams effectively means choosing the right processing model based on the type of analysis required and the limitations of the system. Sliding and damped windows help focus on the most relevant data, sampling reduces processing load, sketching offers lightweight summaries, and aggregation delivers real-time metrics. By understanding and applying these models thoughtfully, we can build systems that turn continuous data into meaningful insights—even in the face of speed, scale, and complexity. In general, such systems are known as Data Stream Management Systems (DSMS).

In today's world, where data is generated continuously from sources like sensors, social media, mobile devices, and financial transactions, managing this constant flow of information becomes a challenge. Traditional database systems are not designed to handle such high-speed, real-time data. This is where a **Data Stream Management System (DSMS)** comes into play.

The Data Stream Management System (DSMS) extracts knowledge from multiple data streams by eliminating undesirable elements, as shown in Figure 1. DSMS is important where the input data rate is controlled externally. For example, Google queries.

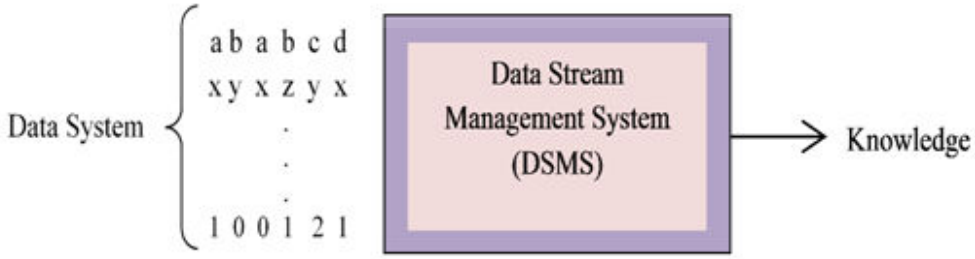


Figure 12.1: A simple outline of Data Stream Management System

The sources of data streams include Internet traffic, online transactions, satellite data, sensors data, live event data, real-time surveillance systems, etc.

So, we can say that a **Data Stream Management System** is a specialized software system designed to **process, manage, and analyze data streams in real time**. Unlike traditional database management systems (DBMS), which deal with static, stored data, a DSMS handles **dynamic and transient data**—data that arrives continuously and may not be stored permanently.

Thus we can think of a DSMS as a **real-time engine** that captures live data, runs queries on it as it flows in, and produces immediate results. It allows us to perform actions like filtering, aggregating, joining, or detecting patterns **on the fly** without storing the entire dataset.

For example, imagine a weather monitoring station that receives temperature, humidity, and wind speed data every few seconds from hundreds of sensors. A traditional system would store all this data first and then analyze it later. But a DSMS processes this data **as it arrives**, allowing meteorologists to instantly detect unusual weather conditions, such as a sudden drop in temperature or wind speed spikes.

A DSMS typically includes several key components:

- **Data Stream Input Interface** – It connects with data sources like sensors, logs, or applications to receive the stream.
- **Query Processor** – It continuously runs user-defined queries (like SQL) over the data stream.
- **Memory Manager** – Since storing all data isn't possible, it uses memory efficiently to store only what's needed, often using techniques like sliding windows or summaries.
- **Scheduler and Optimizer** – These components ensure that queries run quickly and efficiently, even under heavy data loads.
- **Output Interface** – Sends the results to dashboards, alerting systems, or other applications for further action.

Some popular DSMS tools include **Apache Flink**, **Apache Storm**, **Spark Streaming**, and **Google Dataflow**, which are widely used in industries ranging from finance to healthcare to smart cities.

Figure 2 shows the detailed view of a data stream management system. The components of this system are described as follows:

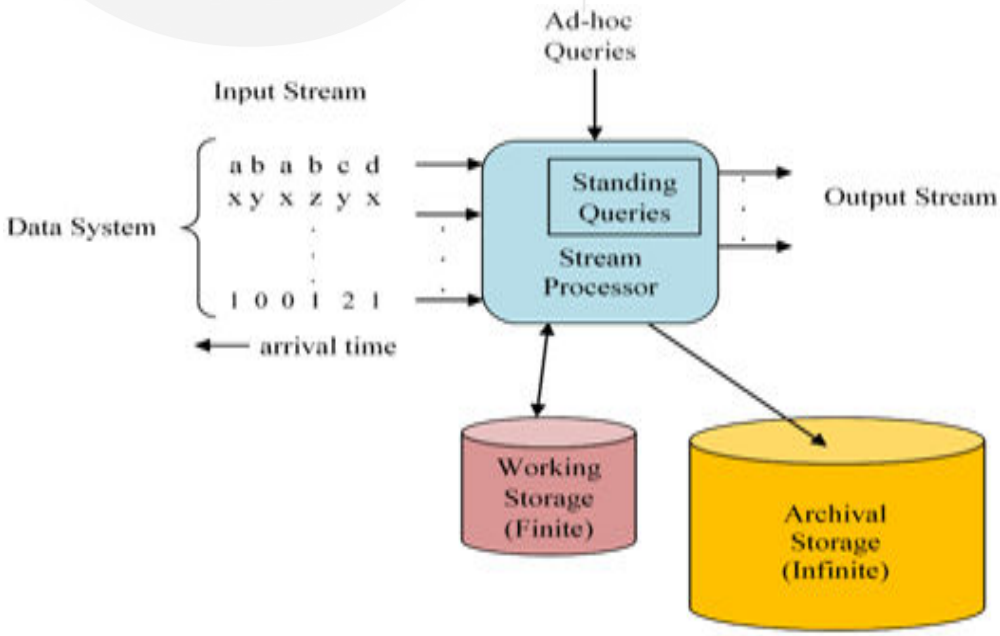


Figure 12.2: A detailed view of Data Stream Management System

- **Processor:** The processor is a software that executes the queries on the data stream. There can be multiple processors working together. The processor may store some standing queries and also allows ad-hoc queries (refer to section 10.3.1) to be issued by the users.
- **Streams Entering:** There can be several streams entering in the system. Conventionally, we will assume that the elements at the right end of the stream have arrived more recently. The time goes backward to the left i.e. the further left the earlier the element entered the system.
- **Output:** The system makes output in response to the standing queries and the ad-hoc queries (refer to section 10.3.1).
- **Archival Storage:** There is a massive archival storage and we cannot assume the archival storage is architected like a database system. Further, we can use appropriate indices or other tools to efficiently answer the queries from that data. We only know that if we had to reconstruct the history of the streams it could take a long time.

Limited Working Storage: It might be a main memory, or a flash storage, or even a magnetic disk. But we assume that it holds important parts of the input streams in a way that supports fast execution of query.

In short, learning about DSMS is essential if you want to build systems that work with real-time data. It enables **faster decision-making**, supports **live analytics**, and helps us keep up with the **fast-paced digital environment** we live in. As a learner, try exploring how DSMS tools work in real-world applications like fraud detection, traffic monitoring, and online recommendation systems to understand their power and importance.

There is a **fundamental difference between Data Base Management System (DBMS) and Data Stream Management System (DSMS)** is who controls how data enters into the system.

(a) DBMS:

In a DBMS, the staff associated with the management of the database usually insert data into the system using a bulk loader or even explicit SQL INSERT commands. The staff can decide how much data to be loaded into the system. The staff can also decide when and how fast to load the data into the system. *For example,* the records of students stored in schools or colleges or universities.

(b) DSMS:

In a DSMS, the management cannot control the rate of input. *For example,* the search queries that arrive at Google search engine are generated by random people around the globe, who search for information at their respective pace. Google staff literally has no control over the rate of arrival of queries. They have to design and architect their system in such a way that it can easily deal with the varying data rate.

Next we need to understand, how to make queries of data streams, which is discussed next.

12.3.1 Queries Of Data Stream

When working with traditional databases, we write queries that run on stored data and return results after the computation is complete. But in the world of data streams, where data is constantly flowing in real time, we need a different kind of approach to querying. Here, the data is not static—it's live, fast, and potentially endless. This is where understanding how queries work in data streams becomes really important.

In data stream systems, we mainly deal with two types of queries: adhoc queries and standing (or continuous) queries.

Adhoc Queries: Adhoc queries are similar to traditional database queries. They are executed once on a snapshot or current state of the data (usually within a defined window). These queries are useful when we want to check something quickly or perform a one-time analysis. Example: Imagine we want to know how many users visited a website in the last 10 minutes. We can run an adhoc query over that time window to get the result once.

Standing (Continuous) Queries: Standing queries, also called continuous queries, are much more powerful and common in data stream systems. These queries are always active—they run continuously on the incoming stream and keep updating the result in real time as new data arrives. Example: If we want to monitor the average CPU usage of a server in real time and raise an alert if it goes above 80%, we'd use a standing query that constantly watches the stream of CPU data.

Common Types of Data Stream Queries: Whether adhoc or continuous, data stream queries are used to perform different operations such as:

- Filtering: Selecting data items that match a condition. *E.g., "Select all tweets containing the word 'earthquake'."*
- Aggregation: Calculating summaries like count, sum, average, max, or min, usually within a time window. *E.g., "Calculate the number of transactions every 30 seconds."*
- Joins: Combining two or more data streams based on a common key. *E.g., "Join GPS and fuel sensor streams to analyze fuel usage per route."*
- Pattern Detection: Identifying specific sequences or events. *E.g., "Detect three consecutive login failures from the same account."*

Important: One of the biggest differences between stream and traditional queries is that stream queries don't wait for the data to stop coming—they work on it as it flows. That's why standing queries are so important. They allow us to set up logic that keeps running in the background, continuously updating results or triggering actions. Meanwhile, adhoc queries give us the flexibility to ask one-time questions over a specific period or sample from the stream.

12.3.2 Examples of Data Stream

To understand what data streams really are, it helps to look at real-world situations where data isn't stored first, but instead flows in continuously. A data stream is a live, ongoing flow of information that must be processed in real time. This kind of data appears in many everyday applications, and recognizing these can help you better appreciate how streaming systems work.

1. **Sensor Data Streams**

   One of the most common sources of data streams is sensors. These are used in everything from weather stations to smart homes and industrial machines. For example, weather sensors stream temperature, humidity, and wind speed data every few seconds. Similarly, smart home devices like thermostats or motion detectors constantly send activity updates. In industries, IoT-enabled machines send data about vibration, pressure, or speed to help monitor equipment health and prevent breakdowns.

2. **Social Media Feeds**

   Social media platforms are full of streaming data. Every time someone posts a tweet, uploads a video, or reacts to a story, that action becomes part of a data stream. For example, Twitter generates a live stream of tweets that can be analyzed in real time to detect trends, sentiments, or viral content. Live comment sections during events or trending hashtags are other examples of this constantly flowing data.

3. **Stock Market Data**

   Financial markets depend heavily on real-time data streams. Stock prices update thousands of times per second, and trading systems process buy/sell orders instantly. This stream of data helps investors make quick decisions and allows financial firms to detect patterns or anomalies, such as unusual trading activity that might indicate market manipulation or fraud.

4. **Clickstream Data**

   When you browse a website or use an app, every action you take—like clicking a link, scrolling a page, or hovering over a product—is recorded. This flow of user behavior is called clickstream data. Companies like Amazon or Netflix analyze this data in real time to personalize your experience by recommending products or shows based on what you're doing right now, not just past behavior.

5. **Telecommunication Streams**

   Telecom companies handle a huge amount of streaming data. Each phone call, message, or internet request is recorded and processed as it happens. For instance, call detail records (CDRs) are streamed to billing systems and fraud detection tools. Similarly, network performance data streams help ensure smooth service delivery and detect problems like dropped calls or slow internet speeds.

6. **Video and Audio Streams**

   Any time you watch a live video or listen to music online, you're using a data stream. Platforms like YouTube Live, Zoom, or Spotify stream data in small chunks to your device continuously. These streams are processed in real time for quality control, buffering, and personalization, like adjusting the video resolution based on your network speed.

7. **Traffic Monitoring Systems**

   Modern cities rely on traffic monitoring systems that collect real-time data from GPS devices, cameras, and road sensors. For example, GPS data from thousands of vehicles is streamed to control centers to detect traffic jams, plan signal timings, or suggest alternate routes. This helps improve traffic flow and reduce congestion on the roads.

8. **E-commerce Transactions**

   Large online retailers generate massive data streams from user actions on their websites. Each product view, search, purchase, or payment is captured instantly. These streams are used to update inventory in real time, provide personalized recommendations, or flag suspicious transactions for fraud analysis.

To better understand data streams, look around you. Your fitness app counting steps, Google Maps updating traffic, or even your smart electric meter—all are working with data streams. As you begin to recognize these examples in everyday life, you'll start to see how important real-time data is in making systems smart, responsive, and efficient.

12.3.3 Issues and Challenges of Data Stream

As we explore the world of data stream processing, you'll quickly realize that it's very different from working with traditional static data. In streaming, data flows continuously, and we must process it in real time. While this offers exciting possibilities—like instant decision-making and live monitoring—it also introduces a number of technical and conceptual challenges. Let's break these down into key categories so that you can better understand the landscape.

1. **Continuous and Unbounded Data**: The most obvious challenge is that data streams are infinite—they never stop. Unlike static datasets, which have a definite beginning and end, stream data keeps coming. This makes it difficult to store all the data or reprocess it later. As a learner, ask yourself: how would you handle a firehose of data that never ends? The solution lies in using sliding or tumbling windows, sampling, or approximation techniques.

2. **Limited Memory and Storage**: Because stream processing needs to happen in real time and memory is limited, you can't store all incoming data. Systems must work with bounded resources—only keeping what's essential and discarding the rest. This means designing algorithms that can process and analyze data on the fly, often using summaries, counters, or sketches that give quick, approximate answers.

3. **Real-Time Query Processing**: Stream queries must run continuously and update results dynamically as new data arrives. Unlike traditional queries that run once and return a result, here the queries live alongside the data. This adds complexity: you need efficient ways to update the answers without reprocessing the entire stream. Try to understand how standing (continuous) queries and windowing functions help solve this challenge.

4. **Data Heterogeneity and Integration**: In many cases, data streams come from multiple, diverse sources. For example, a smart city application might receive streams from sensors, traffic cameras, and weather stations—all in different formats. Managing and integrating this heterogeneous data in real time requires strong data transformation and preprocessing techniques. As a learner, think about how you'd normalize or clean such data before it flows into your analysis pipeline.

5. **Noisy, Incomplete, or Inaccurate Data**: Stream data often arrives with errors or missing values, and there's no time for complex cleaning processes. You must learn to work with imperfect data and still extract meaningful insights. Systems often use filtering, approximation, or confidence thresholds to deal with such challenges. Recognizing this helps you build resilient systems that don't break under messy inputs.

6. **Out-of-Order Data and Latency**: One of the trickier issues in data streaming is when data doesn't arrive in the same order it was generated. For example, a sensor update might be delayed due to a network glitch. This causes out-of-order events, which can affect accuracy. Handling this requires event time processing, watermarks, and buffering strategies. As you go deeper into stream processing tools, look at how they manage time and order.

7. **Security and Privacy**: Many data streams include sensitive or personal information—think of financial transactions, GPS locations, or health data. Ensuring that this data remains secure, and is handled in compliance with privacy laws (like GDPR), is critical. As a learner, consider how encryption, access control, and anonymization might be applied in real-time systems.

   Understanding these challenges is the first step to designing robust and scalable data stream systems. Each challenge teaches you something new about real-time computing. Try exploring open-source platforms like **Apache Kafka**, **Apache Flink**, or **Apache Spark Streaming** to see how these issues are solved in practice. With hands-on practice, you'll begin to appreciate the balance between **speed**, **accuracy**, and **efficiency** in stream processing.

Check Your Progress 1:

1. Define the data stream processing. Which model of data stream processing is useful in finding stock market trends? Justify.

   .....

   .....

2. Differentiate between DBMS and DSMS. Why is all the data of data streams not stored?

   .....

   .....

   .....

3. How are Standing queries different to ad-hoc queries?

   .....

   .....

   .....

4. What is the use of mining click streams?

   .....

   .....

   .....

12.4 SUMMARY

This unit introduces the concept of mining data streams. Data streams can be processed using different models. Three such models (landmark, sliding windows, and damped model) for data stream processing are introduced in this unit. Further, the data stream management system (DSMS) is explained. In addition, different types of queries of data streams namely, ad-hoc and standing queries are discussed followed by examples of these queries. The issues and challenges of Data streams and data sampling in data streams with examples of representation sample and sampling Key-Value pairs are discussed in this unit.

12.5 ANSWERS

Check Your Progress 1:

1. The data stream is a process to extract knowledge in real time from a large amount of volatile data, which comes in an infinite stream of data. The data is volatile because it is continuously changing and evolving over time. The system does not store the data in the database due to limited resources. The landmark model of data stream processing is useful in finding stock market trends because it finds the frequently used items in the entire data stream from a specific time till present and all time-points are treated equally after the starting time.

2. In a DBMS, the staff associated with the management of the database usually insert data into the system. In a DSMS, the management cannot control the rate of input. All data from data streams is not stored due to a limited amount of resources.

3. In standing queries, the users expect the system to report the answer available at all times, perhaps outputting a new value each time the answer changes. In ad-hoc queries, the users make a query once and

expect an answer to the query based on the current state of the system, which is available only at current time.

4. The use of mining click streams is to know which URLs are getting clicked-on, a lot more, in the past one hour than normal.

12.6 REFERENCES/FURTHER READINGS

[1] Mansalis, Stratos, et al. "An evaluation of data stream clustering algorithms." Statistical Analysis and Data Mining: The ASA Data Science Journal 11.4 (2018): 167-187

[2] Albert C. "Introduction to Stream Mining." https://towardsdatascience.com/introduction-to-stream-mining-8b79dd64e460

[3] "Mining Data Streams." http://infolab.stanford.edu/~ullman/mmds/ch4.pdf

ignou
THE PEOPLE'S
UNIVERSITY

UNIT 13 TOOLS FOR DATA SCIENCE – AN INTRODUCTION

13.1 Introduction
13.2 Objectives
13.3 Tools for Data Science
13.4 Datasets and File Formats
13.5 Basics of Tableau
13.6 Basics of Power BI
13.7 Basics of Python
13.8 Basics of R-STUDIO
13.9 Summary
13.10 Solutions/Answers
13.11 Further Readings

13.1 INTRODUCTION

In today's digital era, data is often considered as the new oil—an invaluable resource powering decision-making across industries like healthcare, finance, retail, education, and manufacturing. Yet, the true power of data lies not in its raw form, but in "how effectively we can interpret and apply it?". That's where data science steps in: a multidisciplinary domain blending mathematics, statistics, computer science, and domain expertise to transform chaotic data into meaningful knowledge.

At the heart of this transformation from data to useful information, there exists a powerful suite of tools—technologies that enable data professionals to collect, process, analyze, visualize, and communicate insights with efficiency and precision. These tools form the foundation of the data science workflow, streamlining everything from basic data handling to complex predictive modelling and real-time reporting.

This Unit, serves as your foundational guide to four of the most widely adopted tools in the data science pipeline:

- **Tableau** – An industry-leading platform for visual storytelling and dashboard creation.
- **Power BI** – Power BI – A robust business intelligence tool from Microsoft that lets you create interactive reports and easily connects with other Microsoft products for smooth data analysis.
- **Python** – A versatile and easy-to-learn programming language, Python is widely favoured by beginners and professionals alike for data tasks—thanks to powerful libraries like Pandas, NumPy, Matplotlib, and Scikit-learn that simplify data handling and machine learning.

- **R** – A statistical powerhouse designed for analytical rigor, favoured in academia and research-heavy environments.

The sections covered in this unit, introduces not just the technical skills needed to work with these tools, but also offers practical exposure to their user interfaces, dataset handling methods, and data visualization techniques. The goal is to equip students, aspiring data scientists, and professionals with both the confidence and competence to begin their journey in data science.

Whether you're analysing trends, forecasting outcomes, or simply making smarter decisions, mastering these tools is your first step into a data-driven future. Let's begin.

13.2 OBJECTIVES

Upon completing this unit, learners will be able to:

1. To understand the core functionalities of Tableau, Power BI, Python, and R.
2. Identify and work with various dataset formats such as CSV, Excel, and JSON.
3. Upload, preprocess, and clean datasets for analysis.
4. Design and interpret meaningful data visualizations.

These objectives are designed to empower learners to not only master the basics of data tools, but also apply their skills across academic, research, or industry settings with clarity and confidence.

13.3 TOOLS FOR DATA SCIENCE

In the Unit 9 and 10 we had already explored MS-Excel as a tool for Data Science, but today in the rapidly evolving world of data, many other tools are available, these tools are essential for converting raw information into actionable insights. Apart from MS Excel, the following tools for Data science assist in collecting, cleaning, analyzing, and visualizing data across various domains.

Below are four foundational tools widely used in the industry and academia:

- **Tableau**: A powerful and easy-to-use visualization tool that lets you build interactive reports and dynamic dashboards—no coding required. It's perfect for turning data into clear stories and sharing insights with people who may not have a technical background.
- **Power BI**: Created by Microsoft, Power BI is a strong business intelligence tool that works smoothly with Excel and allows users to build real-time dashboards—making it ideal for enterprise-level data analysis and reporting.
- **Python (Anaconda/Google Colab)**: A general-purpose programming language with extensive libraries like Pandas, NumPy, and Matplotlib

for data manipulation, analysis, and visualization. Anaconda simplifies local setup, while Colab provides a free cloud-based environment.

- **R (RStudio)**: R is a programming language tailored for statistical analysis and data visualization. It's widely used in academic research and data-centric projects for its strong capabilities in handling complex data and producing high-quality graphics. RStudio makes coding, visualizing, and managing data efficient and user-friendly.

These tools are essential to today's data science process, and mastering them equips students and professionals with the skills to tackle real-world challenges efficiently and confidently

13.4 DATASET AND FILE FORMATS

A dataset is an organized set of data used for analysis and building models. It's usually laid out like a table, where rows show individual entries and columns represent different features or variables. Depending on the type and use of the data, datasets can be stored in different formats—one of the most common being CSV (Comma-Separated Values),which is widely used for storing tabular data; Excel files (.xls, .xlsx), which allow multiple sheets and more formatting options; JSON and XML, which are suitable for storing nested or hierarchical data; SQL databases, used for managing structured data; Parquet and ORC, which are efficient columnar storage formats often used in big data systems; and TXT files, which store plain text with custom delimiters.

Format	Description / Strengths / Use Cases
CSV (Comma-Separated Values)	A simple, plain-text format where each row is a new line and columns separated by commas. Very widely supported by spreadsheets (Excel, LibreOffice), data-analysis tools, databases. Great for tabular data, easy to read and edit.
Excel (.xls / .xlsx)	Spreadsheet format allowing multiple sheets, richer metadata / formatting than CSV. Handy if you want human-friendly editing, multiple tables in one file, or need features like formulas or embedded metadata. (Often exported to CSV / other formats for ML workflows.)
JSON (JavaScript Object Notation)	A text-based format supporting nested / hierarchical data (objects, arrays). Useful for semi-structured or hierarchical data (e.g., configuration data, nested records, API outputs). Readable by humans and easily parsed by most programming languages.
XML (eXtensible Markup Language)	Another text-based, hierarchical format. Sometimes used when data has complex nested structure or needs to follow a schema. Less common in modern ML tasks than JSON, but still used in legacy systems or data exchange.
SQL Databases / Relational	For structured datasets that need efficient querying, relationships, updates — more dynamic than flat files. Useful

Format	Description / Strengths / Use Cases
Databases	when data is large, relational, or needs concurrent access/updates.
Parquet, ORC, and other columnar / big-data formats	Designed for large-scale data (big data / analytics), these formats store data in a columnar fashion — which can lead to better compression, faster read/write especially when only some columns are needed. Well-suited for large datasets, distributed systems, data warehouses, and big-data frameworks.
Plain text (.txt) / delimiter-separated (custom delimiters)	Sometimes used for very simple or custom data, logs, semi-structured text. Flexible but requires agreed upon structure or parsing logic; often not ideal for complex datasets.

Datasets can be sourced from various platforms depending on the need and application. Popular sources include data-sharing platforms like **Kaggle** and the **UCI Machine Learning Repository**, which host numerous datasets for academic and practical use. **Government portals** such as **Data.gov** provide open-access datasets for public use. Additionally, **APIs** from platforms like Twitter or Google Maps offer real-time and structured data. Other methods include **web scraping**, where data is extracted from websites, and data collected through **IoT devices or sensors**, which continuously generate data in various domains. Some of the usefil links to gather the datasets are given below:

- Kaggle — https://www.kaggle.com/datasets
- UCI Machine Learning Repository — https://archive.ics.uci.edu/datasets
- Google Dataset Search — (search via https://datasetsearch.research.google.com/)
- Open Government Data Platform India (data.gov.in) — https://data.gov.in/
- Data.gov (US) — https://data.gov
- Community/curated lists like "Awesome Public Datasets" on GitHub or data.world

We will discuss, the usage of datasets along with the utilization of various tools mentioned above. Let's begin with Tableau first, in the next section.

13.5 TABLEAU

Tableau is a user-friendly and a powerful tool for creating interactive data visualizations. It connects easily to a wide range of data sources and lets users explore and present their data through clear, engaging visuals with its simple drag-and-drop interface, even beginners can build charts, graphs, and dashboards—no coding needed. Widely used by data analysts and scientists, Tableau is popular in industries like healthcare, tech, and e-commerce to support smart, data-driven decisions. Once installed, users can get started by activating their license or signing in with their Tableau account.

Tableau Installation: Tableau is a free, easy-to-use tool that lets users build engaging and interactive visualizations—all without writing any code. It helps transform raw data into a more readable and insightful format. Tableau has become a widely used tool in the business analytics space.

Users can connect with others across the globe, share dashboards and data visualizations, and even publish their work on websites, blogs, or social media platforms.

Here in this section, we will try to cover most of the features of Tableau in brief. To begin with the Prerequisites of Using Tableau.

As a Prerequisites for Using Tableau, it only requires Basic computer skills, such as running programs and navigating software interfaces, along with Familiarity with spreadsheet applications, lastly your Willingness to explore and learn new tools.

Firstly, Let's learn how to Install Tableau, in order to **Get Started with Installation**, simply go to the official website at https://www.tableau.com and follow the step-by-step instructions provided there.

The stepwise process is also given here, the Fig.1: Tableau Installation shows the interface available at official website at https://www.tableau.com

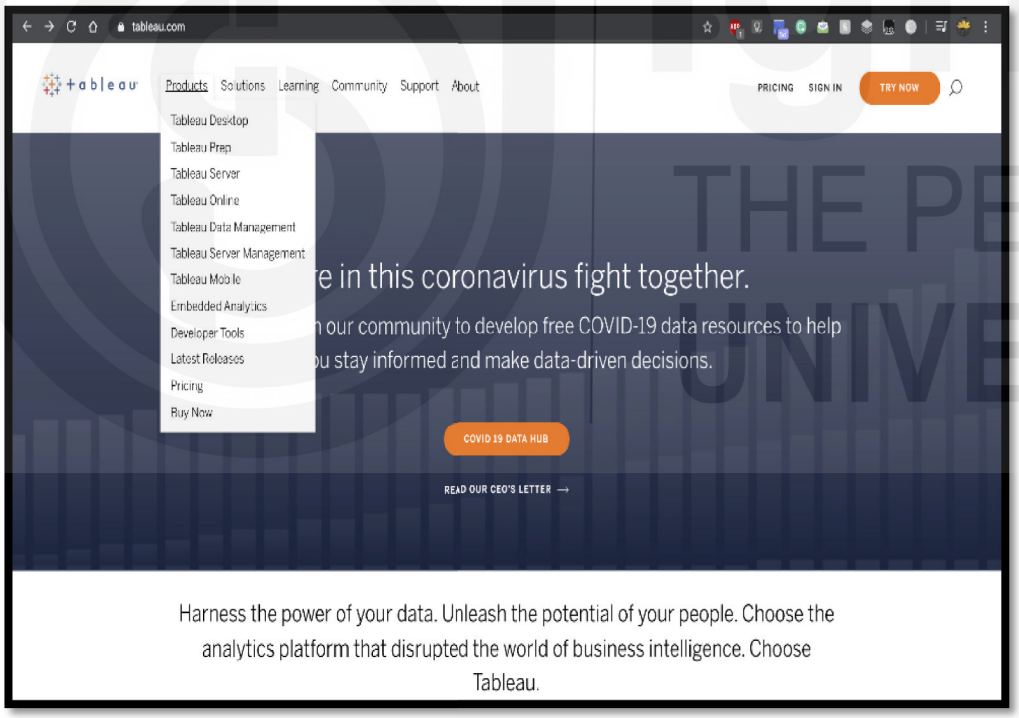


Fig.1: Tableau Installation

At official website at https://www.tableau.com, Navigate to the **Products** section, where you'll find four main options as shown in Fig.2: Tableau Desktop Products:

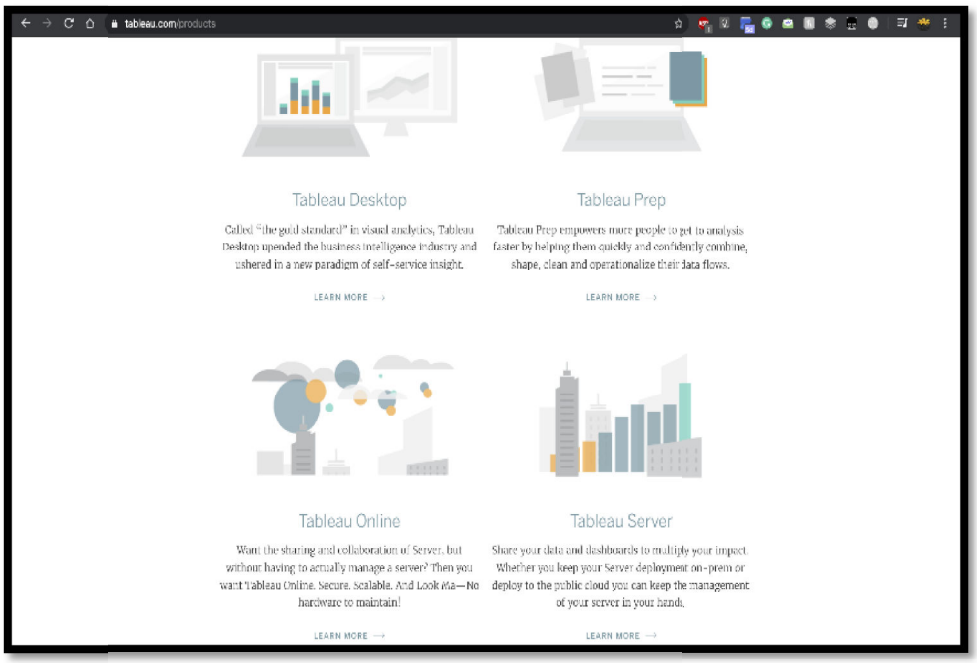


Fig.2: Tableau Desktop Products

Refer to Fig.2: Tableau Desktop Products, it covers **Tableau Desktop and Tableau Prep, Tableau Server** and **Tableau Online** as its major components, a brief description of these components is as follows:

- **Tableau Desktop**: A standalone desktop application designed for individual users to create and analyze data visualizations.
- **Tableau Prep**: This includes two components—**Tableau Prep Builder**, is used to design and build data preparation workflows, while **Tableau Prep Conductor** allows organizations to schedule, manage, and track these workflows efficiently across teams.

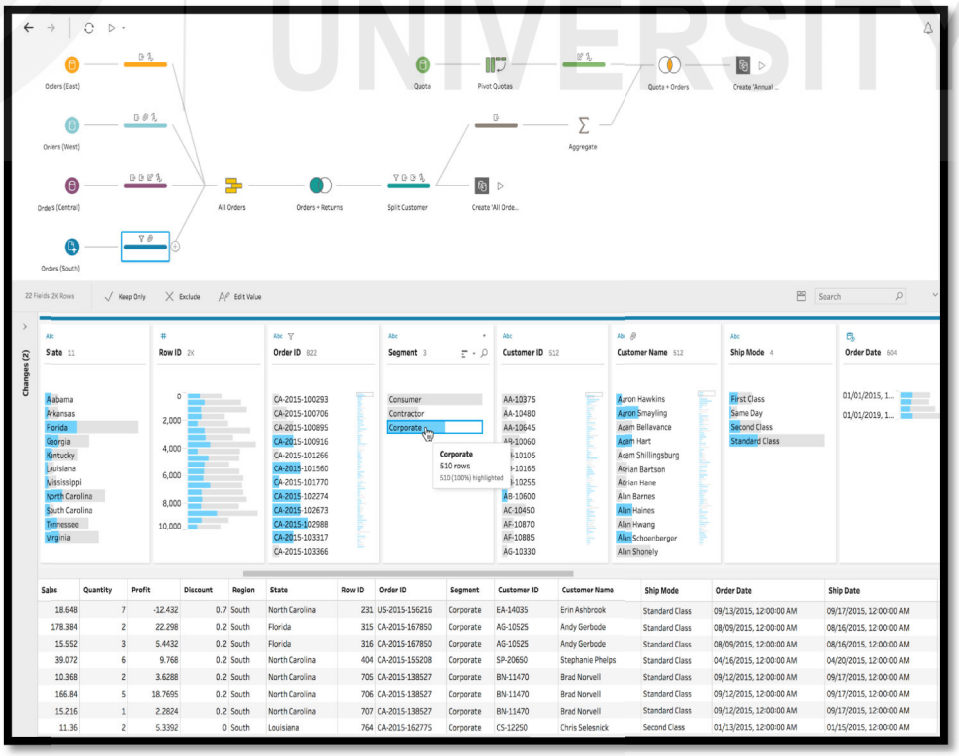


Fig.3: Tableau Prep

- **Tableau Server**: This is primarily designed for enterprise-level use, enabling organizations to securely share, manage, and collaborate on data visualizations across teams.

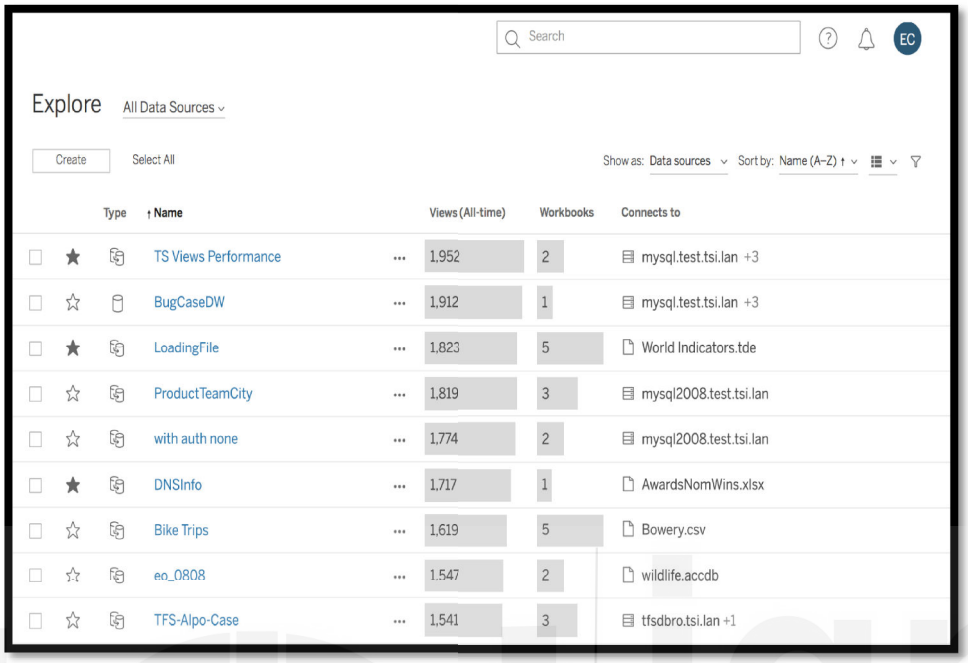


Fig.4: Tableau Server

- **Tableau Online** is a public platform, so any content you upload can be accessed by other users of the software. As a result, your work is no longer private or confidential.

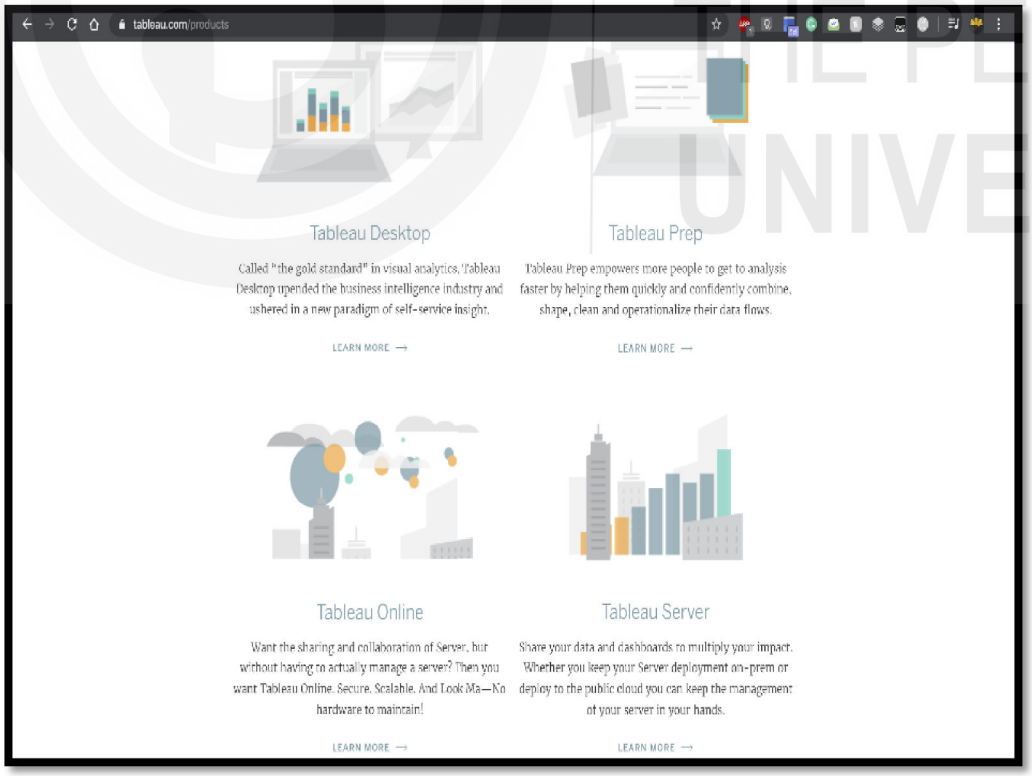


Fig.5: Tableau Online

You can Choose Tableau Desktop and start your free trial, The interface of Tableau Desktop is shown below in Fig.6.

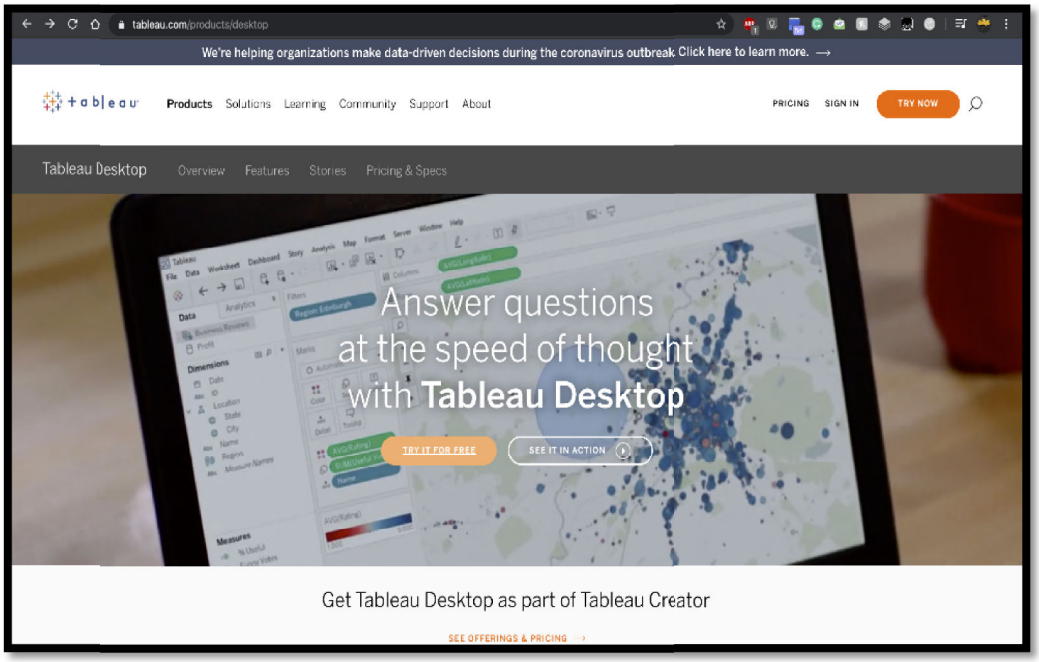


Fig.6: Tableau Desktop

Choose Tableau Desktop and to start your free trial, perform the following steps:

- Choose the software version that matches your system requirements and download the free trial.
- If you prefer not to use the trial, you can download Tableau Public instead. Once done, Tableau will be installed on your system.

Once Tableau is installed you will see the user interface as shown in Fig. 7, below

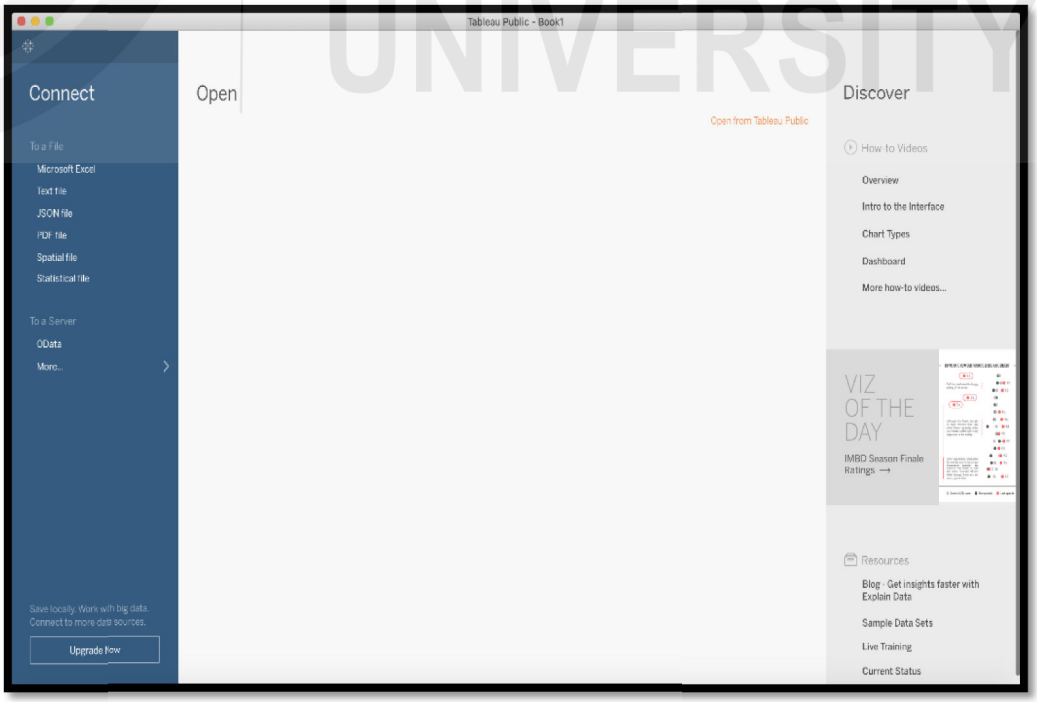


Fig.7: Tableau Installed

13.5.1 Interface of Tableau - Tableau's main interface window

The Tableau workspace consists of several components, including menus, a toolbar, the Data pane, shelves, cards, and one or more sheets. These sheets may be worksheets, dashboards, or stories as shown in Fig.8 below, but for detailed guidance on creating dashboards or working with stories, you can explore the sections dedicated to dashboards and the Story workspace.

If you're working with Tableau in a web browser, check out "Creators: Get Started with Web Authoring" and "Tour Your Tableau Site" for guidance.

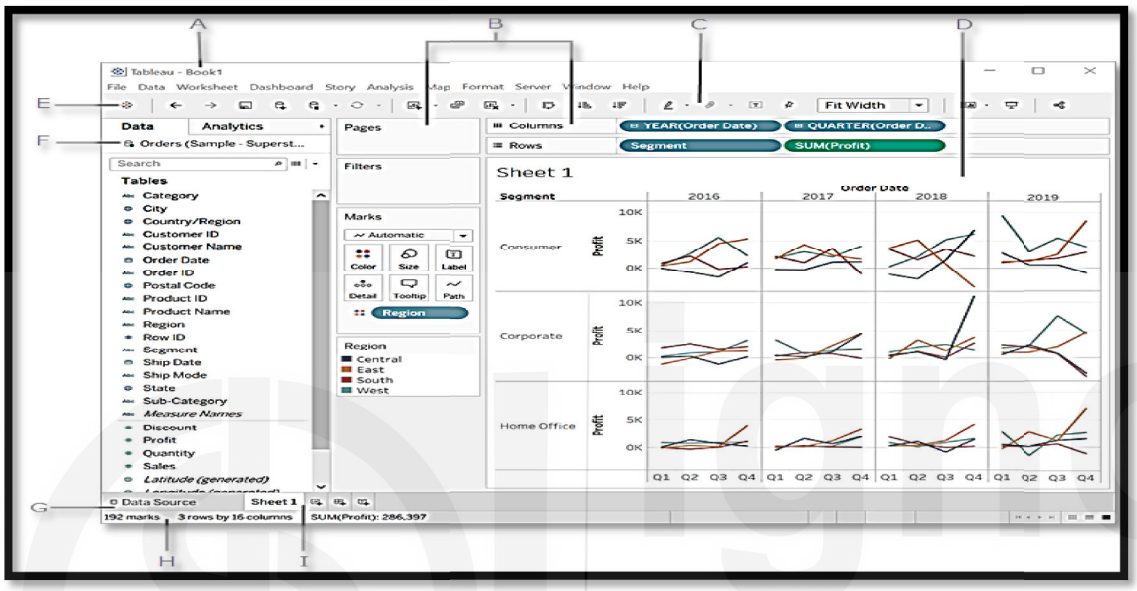


Fig.8: Interface of Tableau

The components of the interface of Tableau are shown in Fig. 8 numbered from A to I, their respective elaboration and utility is given below:

A. **Workbook Title** –A workbook contains multiple sheets, which may include worksheets, dashboards, or stories. For more information, see the section on *Workbooks and Sheets*.

B. **Cards and Shelves** – These are areas where you can drag data fields to create and personalize your visualizations.

C. **Toolbar** – The toolbar offers quick access to various commands, as well as tools for analysis and navigation.

D. **View Area** – This is the primary workspace where your visualization, or "viz," is displayed.

E. **Home Icon** – Click this button to go back to the Start page, where you can connect to new data sources. For more information, refer to the Start Page section [1].

F. **Side Bar** –Located within worksheets, this section contains both the Data pane and the Analytics pane.

G. **Data Source Tab** – Click here to navigate to the Data Source page, where you can view, explore, and manage your data. For more details,

refer to the Data Source Page section.

H. **Status Bar** – Shows details and status of the current visualization or view.

I. **Sheet Tabs** – Each tab represents a different sheet in the workbook, such as a worksheet, dashboard, or story. More info is available in *Workbooks and Sheets*.

13.5.2 Overview of Tableau Toolbar Buttons

While building or modifying a view, the toolbar located at the top provides quick access to frequently used tools and commands. In Tableau Desktop, you can choose to show or hide the toolbar by going to **Window > Show Toolbar**.

The table (Table 1: Tableau Toolbar) shown below describes what each button on the toolbar does. Note that certain buttons may not appear in every version of Tableau. You can also refer to *Visual Cues and Icons in Tableau Desktop* for additional details.

Toolbar Button	Description
Tableau Icon	Opens the Start Page (Desktop only).
Undo	Reverses the last action. Multiple undos are allowed.
Redo	Restores the last undone action.
Save	Saves the workbook (File > Save in Server/Cloud).
New Data Source	Connects to a new or existing data source.
Pause Auto Updates	Stops automatic updates when changes are made.
Run Update	Manually refreshes data (Desktop only).
New Worksheet	Adds a new worksheet, dashboard, or story.
Duplicate	Creates a copy of the current sheet.
Clear	Removes filters, formatting, or entire view elements.
Swap	Switches Rows and Columns fields.
Sort Ascending	Sorts data in increasing order.
Sort Descending	Sorts data in decreasing order.
Totals	Adds or removes grand totals or subtotals.
Highlight	Enables value highlighting in the sheet.
Group Members	Combines selected data into a group.
Show Mark Labels	Toggles visibility of data labels.
Fix Axes	Locks or unlocks axis range (Desktop only).
Format Workbook	Adjust workbook-wide formatting settings.
Fit	Controls how the view fits in the window.
Show/Hide Cards	Shows or hides worksheet cards.
Presentation Mode	Hides all except the view (Desktop only).
Download	Exports view/data in image, PDF, or CSV (Server/Cloud only).
Share	Publishes workbook to Server or Cloud (Desktop only).
Show Me	Suggests best visualizations for selected data.

Table 1: Tableau Toolbar

After understanding the user interface of Tableau, lets understand, how to begin the work of data analysis, wherein the first step is to understand "How to upload any dataset?", next section relates to this only.

13.5.3 How to upload any dataset ?

The stepwise process of uploading any dataset, in Tableau is as follows:

Step – 1: Start Tableau on your system, you will see an interface as shown in Fig 9 below

Fig 9: Tableau Front Page Interface

Step – 2: "Navigate to the 'Connect' section, choose 'To a File', then click on file format you want to upload for an instance 'Microsoft Excel'.", as shown in Fig.10 below.

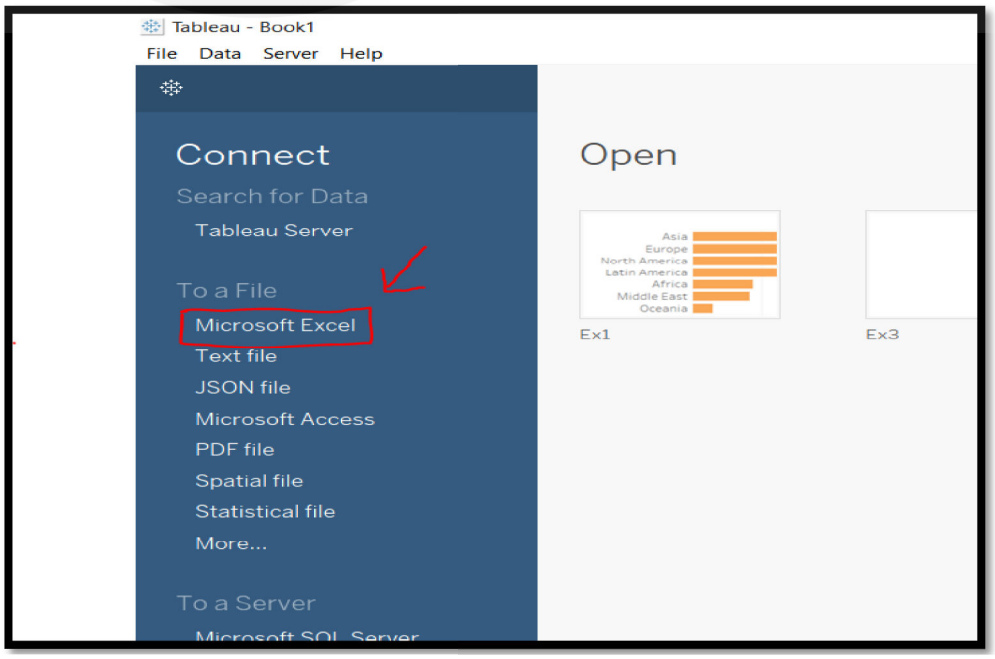


Fig 10: File Upload Interface in Tableau

Step – 3: Clicking this opens a file browser window, allowing you to navigate through folders and choose the file you wish to open, as shown in Fig. 11.

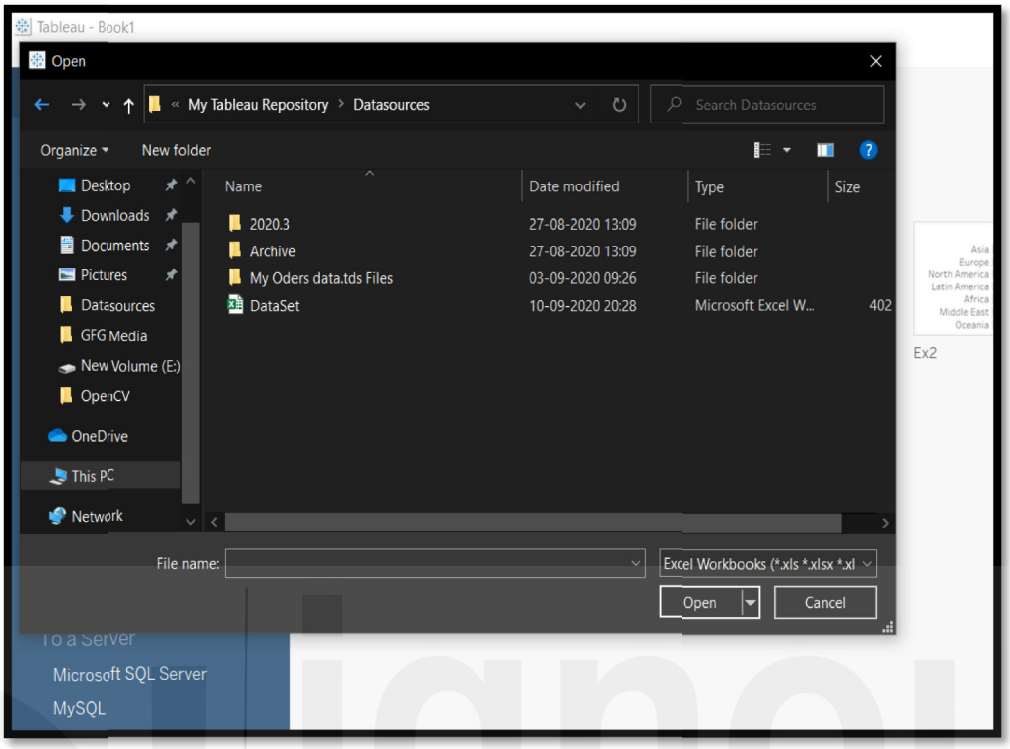


Fig 11: File Browser Interface in Tableau

Now, the sub-steps to be followed for the upload of dataset are as follows:

Step – 3(a): Select an appropriate file and open it, as shown in Fig. 12

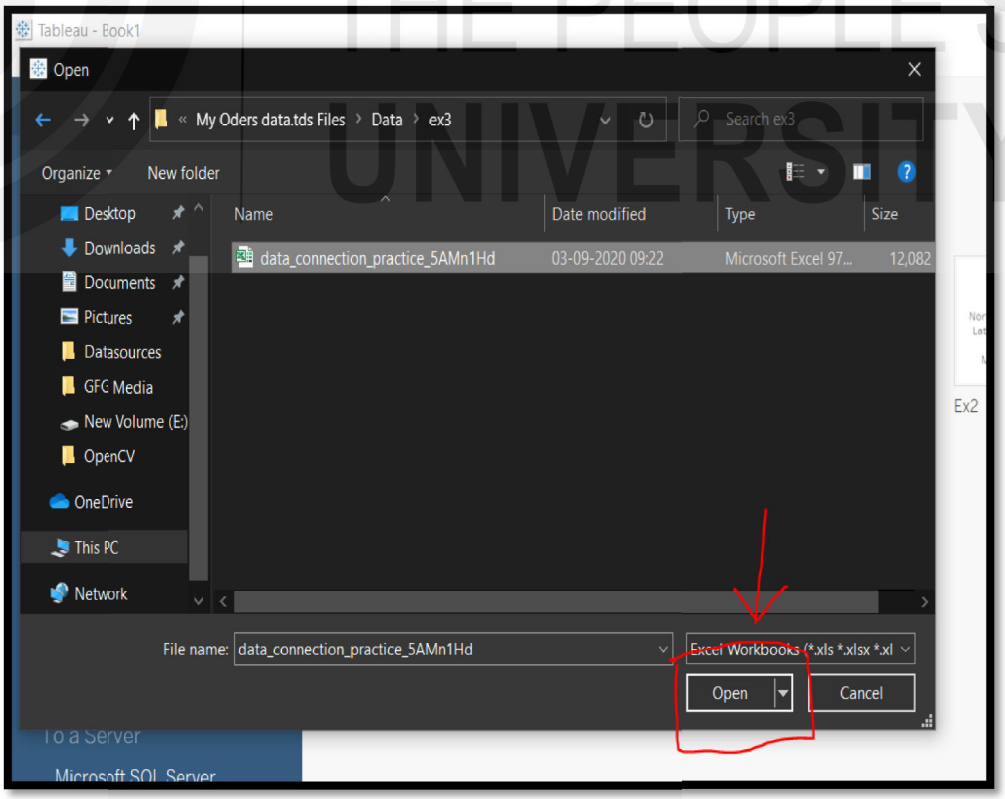


Fig 12: File upload in Tableau

Step – 3(b): Once the file is opened in Tableau, you'll see a screen displaying the connected data, as shown in Fig 13 below

Fig 13: Data Connection in Tableau

Finally, Tableau displays the contents of your Excel file along with all its available sheets as shown in Fig 14,

Fig 14: Data Connection sheets in Tableau

By dragging and dropping a sheet, you'll be able to view its data and proceed with further tasks, like generating descriptive statistics and many more. The subsequent section relates to the generation of descriptive statistics.

13.5.4 Generating Descriptive Statistics - Statistical Analysis and Tableau

Statistical analysis is essential in business intelligence, as it enables organizations to explore data, identify trends, and gain meaningful insights. One key area—**descriptive statistics**—centers on summarizing data through metrics such as measures of central tendency (like mean and median) and variability (such as standard deviation).

Descriptive statistics form the starting point for understanding any dataset. It summarizes the main characteristics of numerical values by providing measures such as averages, ranges, and levels of variation. Tableau includes several tools that allow users to produce these statistics quickly and interpret them either numerically or visually. The following steps explain how descriptive statistics can be generated within Tableau in a practical and intuitive way, by using the Summary card.

A Summary Card in Tableau is a feature used to display quick statistical information about the data in a worksheet or about a selected field. It automatically shows key measures such as minimum, maximum, average, total (sum), and count of values, helping users quickly understand the overall distribution and range of the data. Summary cards can be applied to the entire view or to specific dimensions and measures, making them useful for quick data validation, comparison, and insight generation without performing manual calculations.

Let's learn how to use the Summary Card for generating descriptive statistics, Following are the steps to generate descriptive statistics in Tableau:

1) Load your dataset into Tableau and open a new worksheet.
2) Choose the numerical field you want to analyze and drag it to either the Rows or Columns shelf to create a basic view.
3) On the menu bar, select **Worksheet → Show Summary.**
4) After opening your worksheet, navigate to the *Worksheet* menu in the top-left corner and select **Show Summary, as shown in Fig. 15 below.**

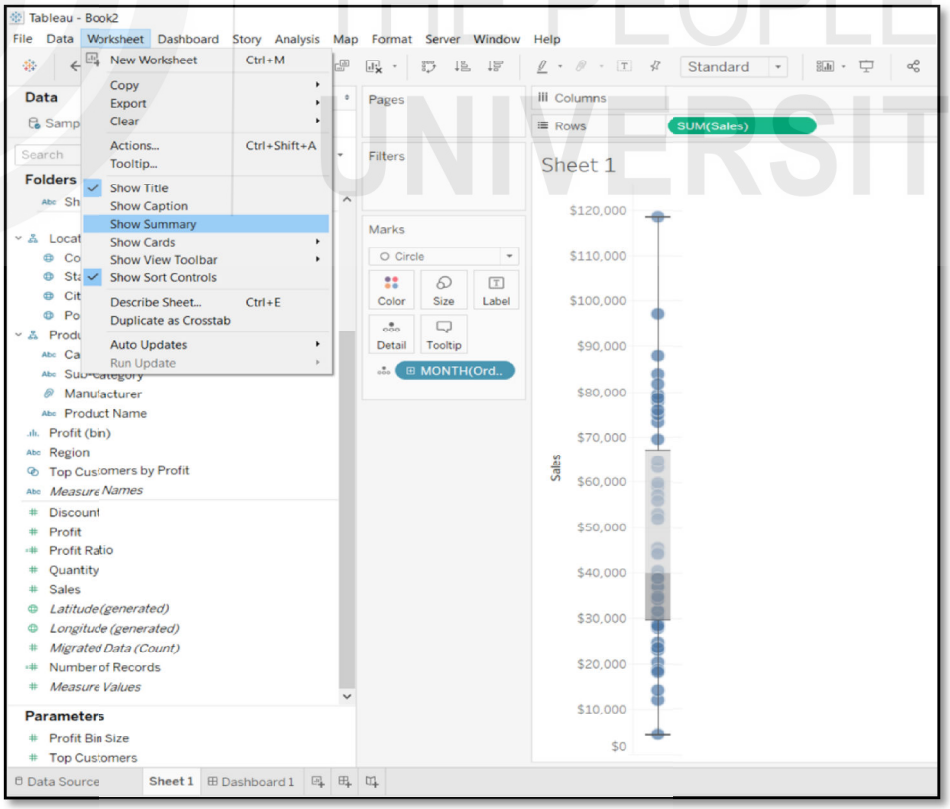


Fig 15: Descriptive Statistics in Tableau

Note: It is to be noted that, once enabled, the Summary Card automatically appears on the right side of the visualization, as shown in Fig. 16 below.

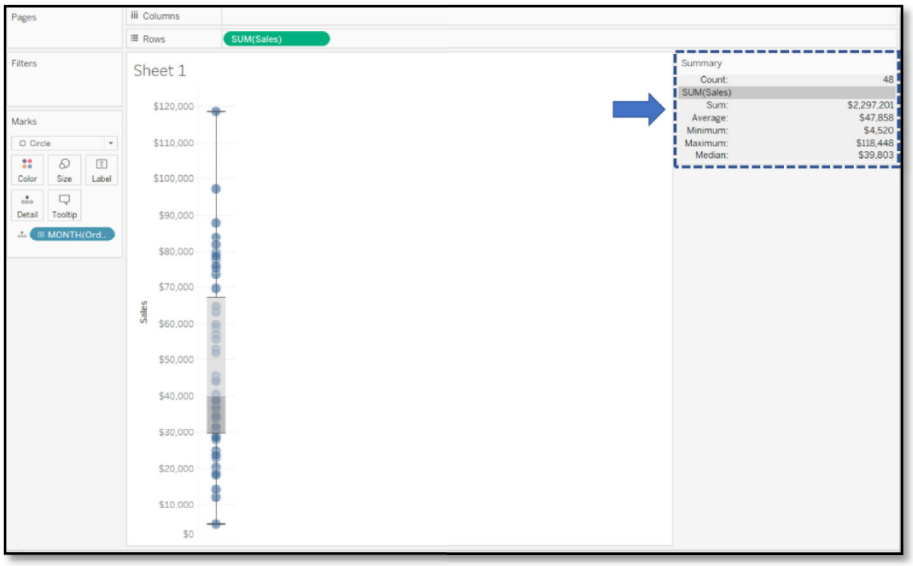


Fig 16: Summary of Descriptive Statistics in Tableau

Note: The Summary Card can be repositioned anywhere on the worksheet—whether to the right, left, top, or bottom—simply by dragging and dropping it as needed, as shown in fig. 17 below.

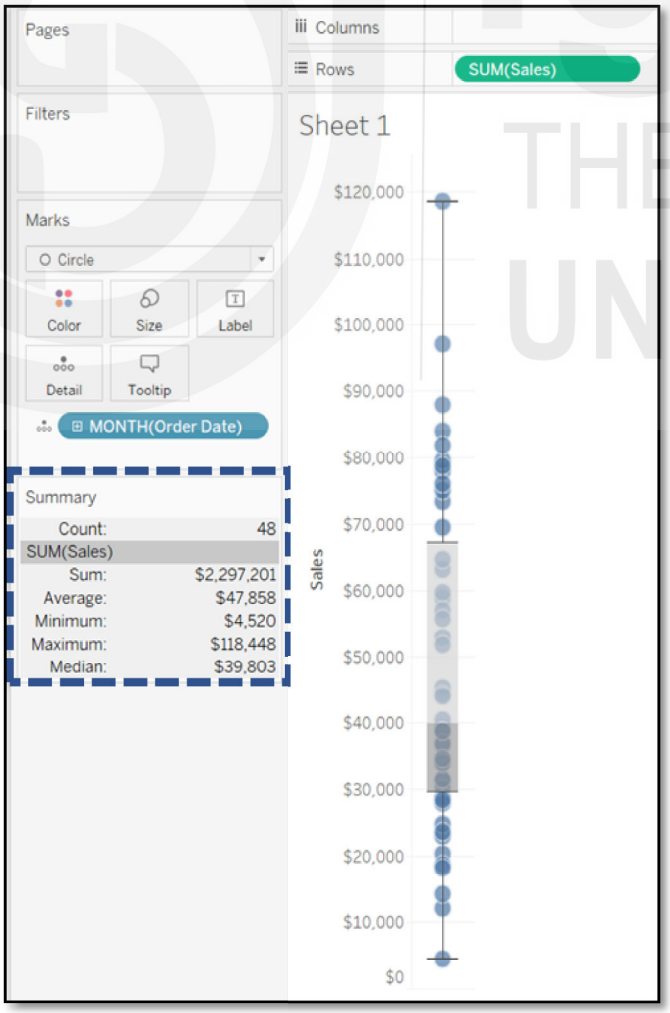


Fig. 17: Summary in Tableau

In short, Tableau provides an efficient way to generate descriptive statistics using the Summary Card. It offers clear, quick insights into key measures such as count, sum, average, minimum, and maximum values. By using the Summary Card, users can easily understand data behavior and obtain essential statistical information. This makes Tableau not only a visualization tool but also a practical platform for basic descriptive analysis.

Till the time we learned, how to upload the data set? and how to generate descriptive statistics? of the available dataset. Now, let's extend our discussion to the plotting of graphs and its analysis. The same is performed in next section.

13.5.5 Graph plotting and its analysis

Once the data has been uploaded, it can be used for visualization, which is the primary goal of using Tableau. This section focuses on demonstrating how to build a series of visualizations in Tableau. To perform visualization we need to **Connect to the database** in Tableau, in this case the referred data source contains **three separate sheets—Product**, **OrderDetails**, and **PropertyInfo**—all included within a single Excel file.

Below are the Steps to follow for visualization:

A) Opening Tableau and Connecting to Dataset

1. Launch **Tableau Desktop**.
2. On the **Start Page**, click on **Text file / Excel** depending on the file format.
3. Browse and select the **Product dataset file**.

Below is the structure of product table (Table-2):

Table 2: Product Table

ProductID	ProductName	ProductCategory	Price
1	Large Towel	Housekeeping	9
2	Hand Towel	Housekeeping	5
3	Washcloth	Housekeeping	3
4	Shampoo	Housekeeping	40
5	Moisturizer	Housekeeping	40
6	Conditioner	Housekeeping	40
7	Hand Soap	Housekeeping	35
8	Bath Soap	Housekeeping	35
9	Tissues	Housekeeping	14
10	Toilet Paper	Housekeeping	19
11	Shower Cap	Housekeeping	10
12	Bed Sheet (King)	Housekeeping	29
13	Bed Sheet (Double)	Housekeeping	20

4. Click **Open** → Tableau loads the data into the **Data Source window**.

5. Verify all column names and data types.

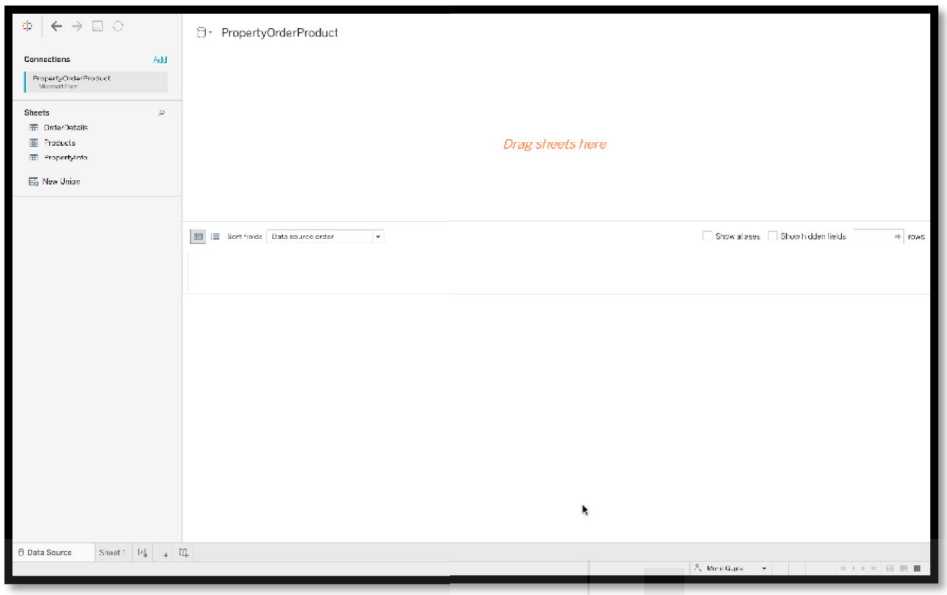


Fig. 18: Dataset Connection with Tableau

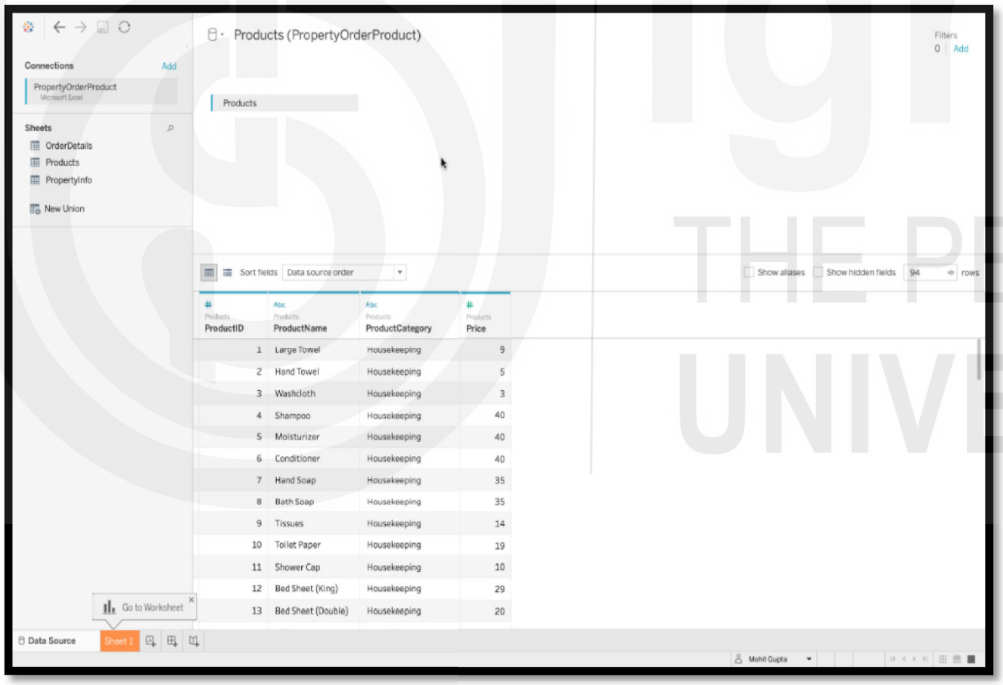


Fig. 19: Open Dataset Connection with Tableau

6. Click on **Sheet 1** to begin visualization.

B) Creating a Graph (Bar Chart: Category vs Price)

Step-by-Step Graph Plotting:

1. Drag **Product_Category** to the **Columns shelf**.
2. Drag **Price** to the **Rows shelf**.
3. From the **Marks card**, select **Bar**.
4. Tableau automatically plots a **bar graph showing the total price for each category**.

5. Drag **ProductName** to the **Color** option in the Marks card to differentiate products.
6. Click on **Show Mark Labels** to display price values on the bars.
7. Add a title such as: *"Category-wise Product Price Distribution"*

13.5.6 Dashboard Creation in Tableau is an important and interesting task, and to create a dashboard, at least two charts are required.

Here, we create the worksheets for the creation of Dashboard:

Worksheet 1: Category-wise Total Price (Bar Chart)

1. Drag Product_Category → Columns shelf.
2. Drag Price → Rows shelf.
3. From the Marks card, select Bar.
4. Click Show Mark Labels.
5. Rename the sheet as: *"Category-wise Price"*

Worksheet 2: Product-wise Price Comparison (Bar Chart)

1. Drag ProductName → Columns shelf.
2. Drag Price → Rows shelf.
3. Select Bar from Marks.
4. Click on Sort (Descending) by price.
5. Enable Show Mark Labels.
6. Rename the sheet as: *"Product Price Comparison"*

Creating the Dashboard

1. Click on the **New Dashboard** icon at the bottom of Tableau.
2. A blank dashboard page opens.
3. From the **Dashboard Pane (left side)**:
   - Drag *"Category-wise Price"* sheet onto the dashboard.
   - Drag *"Product Price Comparison"* sheet below or beside the first chart.
4. Adjust the size and alignment of both charts.
5. Set **Dashboard Size** to **Automatic** for responsive display.
6. Add a **Dashboard Title**, for example: *"Product Pricing Analysis Dashboard"*

Further, as an alternative you can create from a Dashboard in Tableau. On using following these steps.

- Open Tableau → Connect to Excel / CSV files
- Import the three tables into Data Source Product, OrderDetails, and PropertyInfo—all included within a single Excel file.
- Create manual joins using ProductID

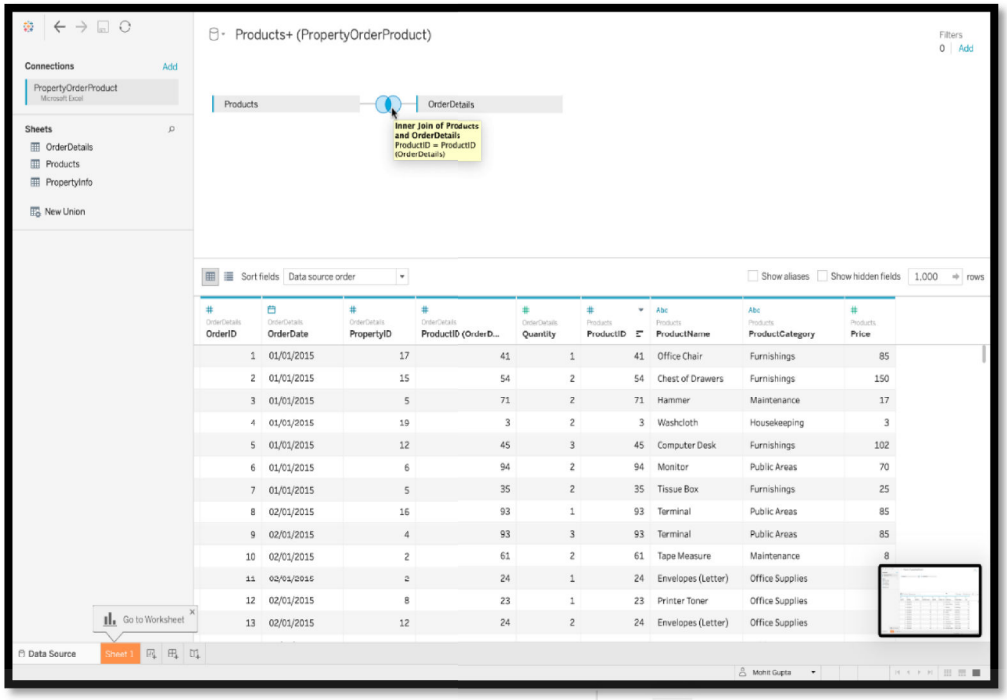


Fig. 20: Two Dataset Connection with Tableau

If you hover over the join, you'll see that an inner join has been created using a common field, namely *Product ID*. An inner join means that the two tables share a common column, allowing their data to be combined seamlessly. Next, add the *PropertyInfo* sheet, and you'll notice that it also gets joined automatically.

Note:

Table au automatically detects relationships between tables based on matching field names. Since fields like **Product ID** *share the same name across the connected sheets, Tableau recognizes them as common keys and creates the join without requiring manual input. This automatic behavior occurs only when field names are consistent across tables.*

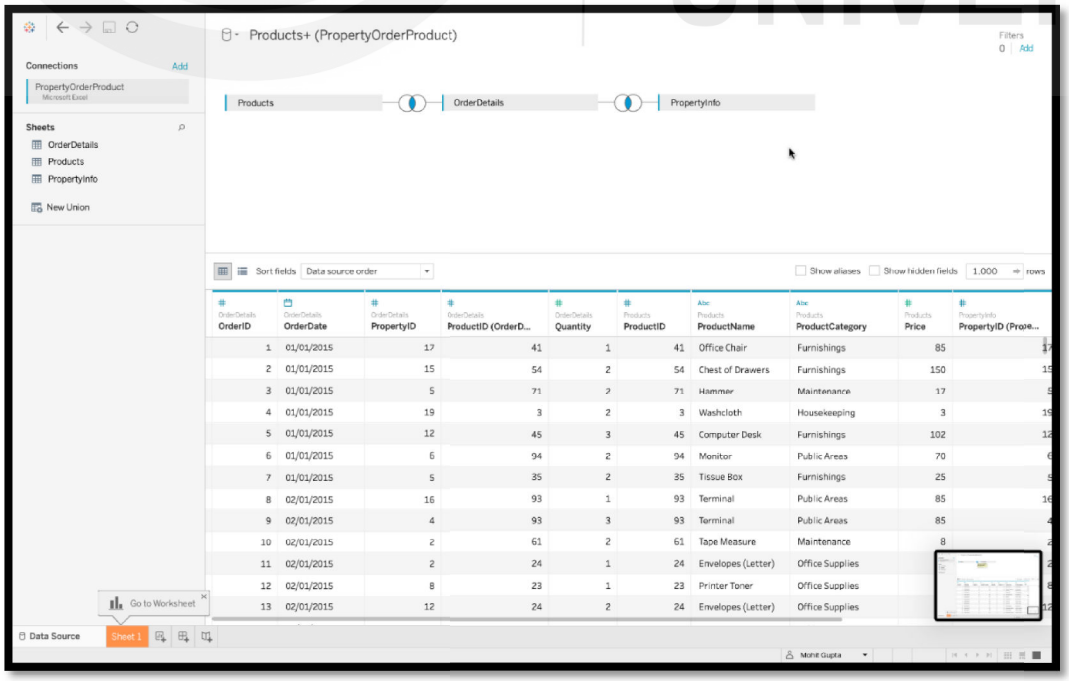


Fig. 21: ExcelSheet Dataset Connection with Tableau

Hovering over the join reveals that an inner join has been established between *OrderDetails* and *PropertyInfo* using *PropertyID* as the common key

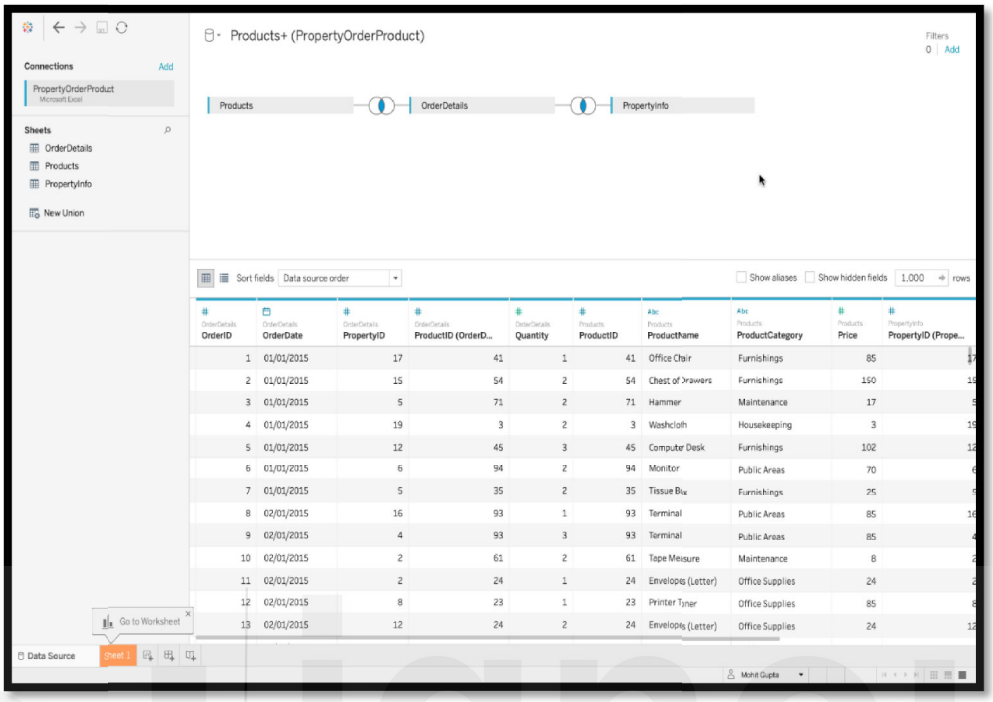


Fig. 22: ExcelSheet Dataset Connection with Tableau

Now, that data is completely ready for data visualization through dashboard

➢ Go to **Sheet 1** → Create charts:

Histogram / Bar chart for Sales Distribution

Box Plot via **Analytics Pane**

- Create additional visuals if required (e.g., Region-wise sales)
- Select **Dashboard → New Dashboard**
- Drag visualizations into the layout and adjust formatting
- Add filters or legends for interactivity

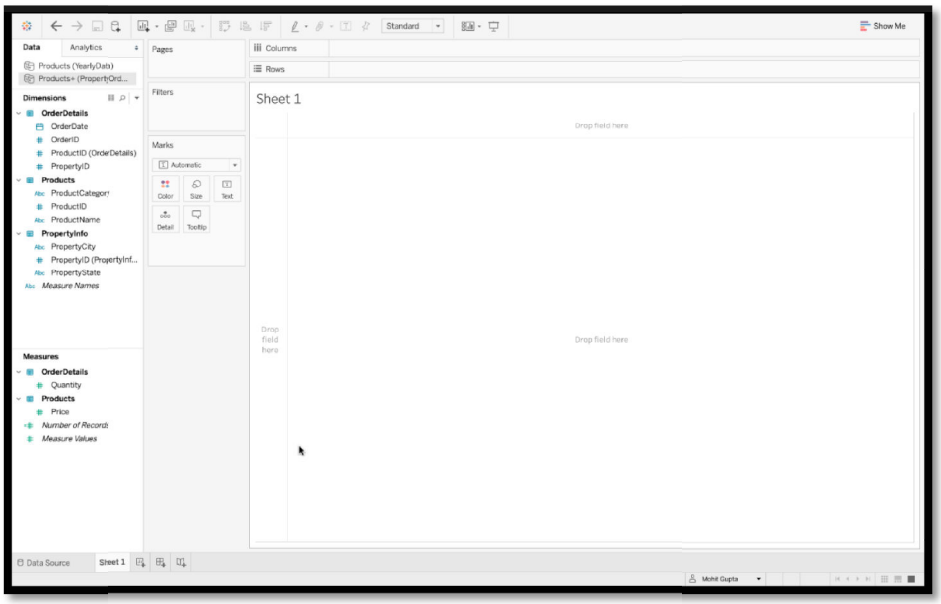


Fig. 23: Excel Sheet Dataset Connection with Tableau

Choose **Quantity**, then navigate to the top-right corner of the toolbar and click the **Show Me** button to generate a visualization.

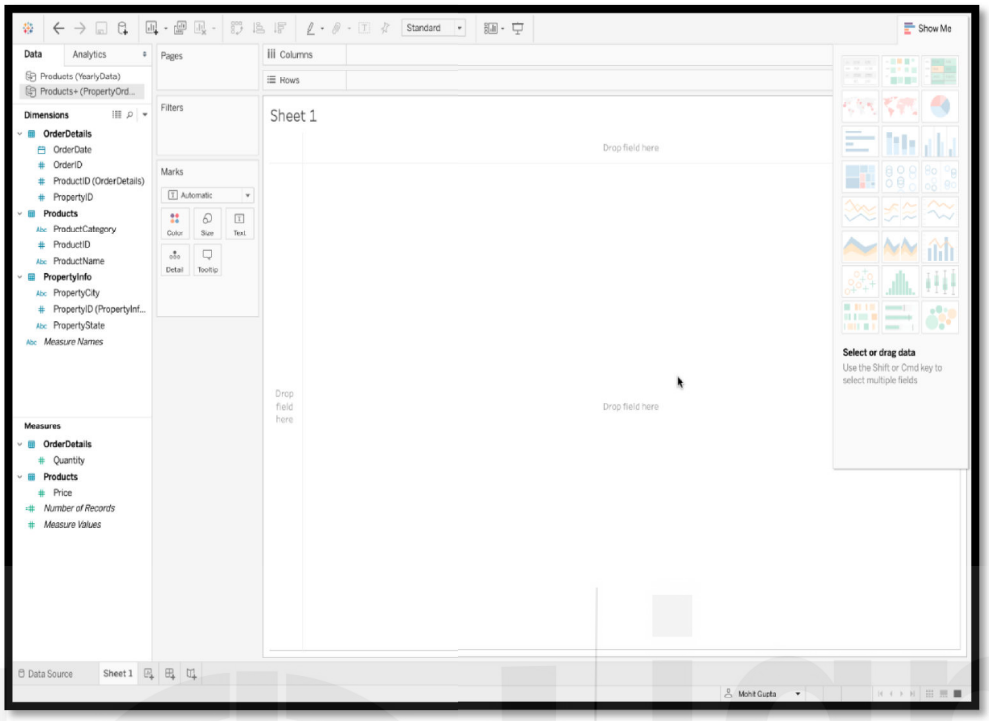


Fig. 24: Excel Sheet Quantity Dataset Connection with Tableau

Go ahead and click on **Text Table** to display the data in a tabular format.

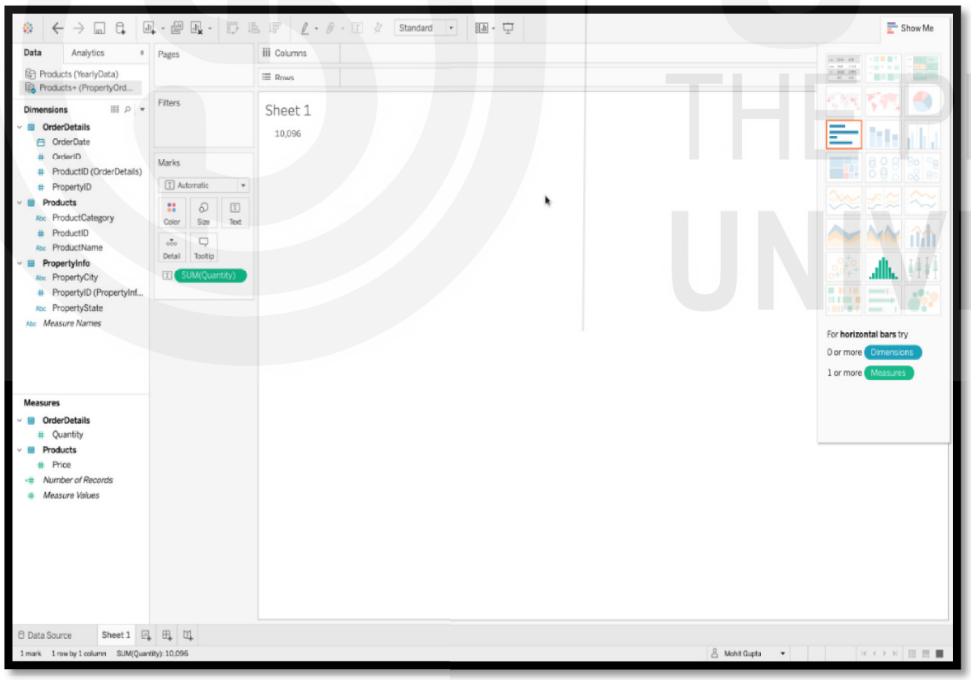


Fig. 25: Text table Dataset Connection with Tableau

The current view displays only one measure (the total quantity), which adds up to 10,096. With no dimensions or categories to break it down, the visualization isn't very insightful. Let's enhance it by adding additional fields to generate a more detailed and interesting chart.

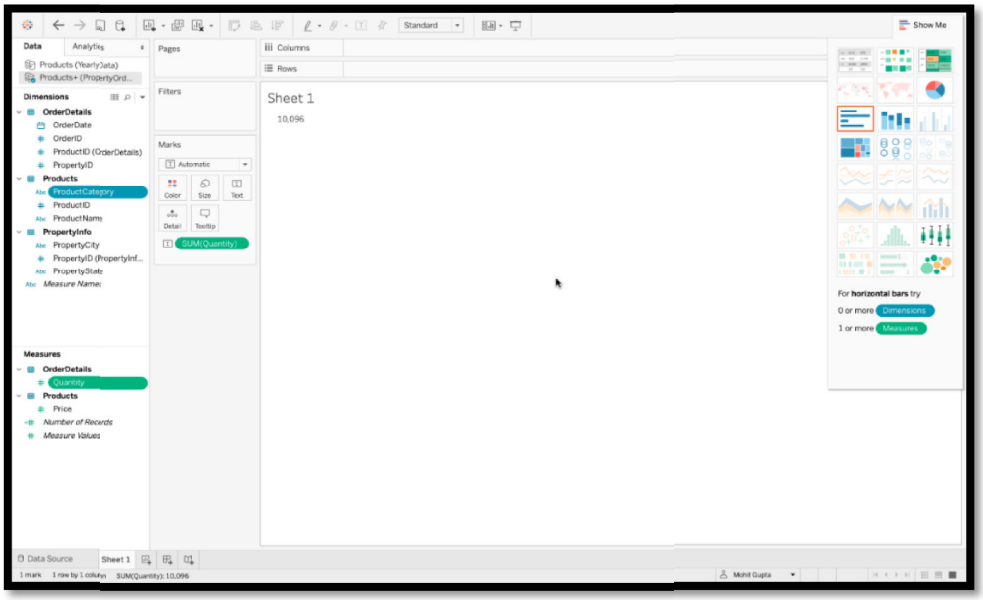


Fig. 26: Text table Dataset Connection with Tableau

To select multiple non-adjacent fields, press and hold the Ctrl key (or Command key on Mac), then click on each one individually. For example, select both *Quantity* and *Product Category*. With these fields selected, Tableau will now suggest a broader range of visualization options.

Now that both Product Category and Quantity are in the view, head to the Show Me panel and select the Tree Map option. This chart uses both color and size to visually represent values, making trends easy to identify.

The Tree Map clearly shows that Furnishings has the highest number of items ordered. Public Areas and Housekeeping follow closely, with Maintenance next, and Office Supplies having the lowest quantity.

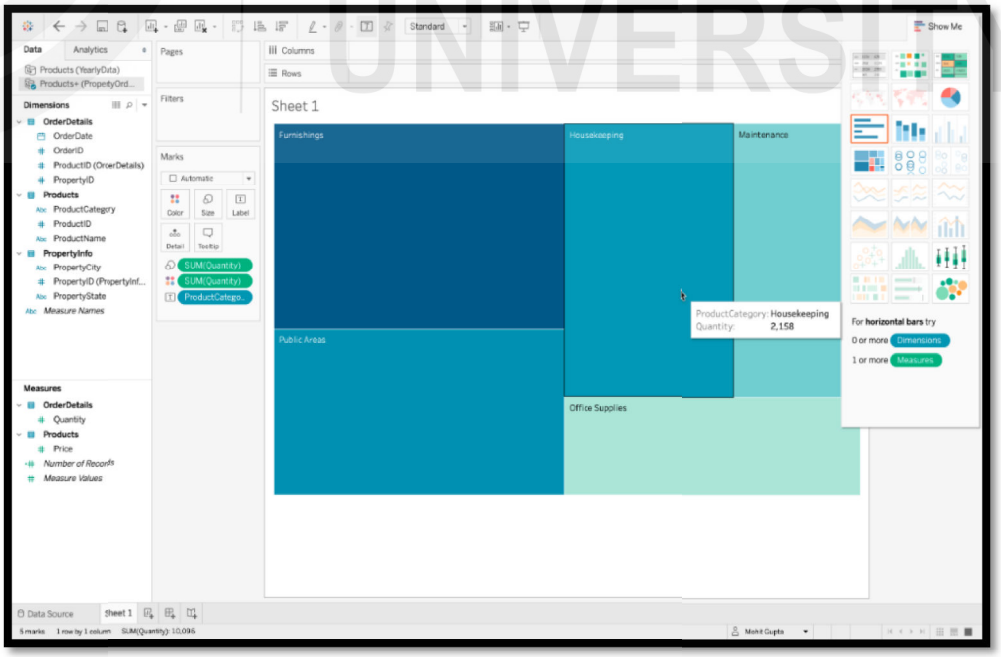


Fig. 27: Tree Map Dataset Connection with Tableau

At first glance, ***Maintenance*** and ***Office Supplies*** appear to be nearly the same size in the Tree Map. However, when you revisit the ***Show Me*** tab and

explore the visual more closely, it becomes clear that ***Maintenance*** has a **slightly higher quantity compared to Office Supplies.**

This illustrates the fundamental steps for quickly creating visualizations in Tableau using the options provided in the **Show Me** panel.

13.5.7SUM UP: Tableau - Data handling and User Interface

1. ***Introduction to Tableau Interface***

- **Workspace**: Consists of Data Pane, Shelves (Rows, Columns), and Worksheets.
- **Marks Card**: Customize visuals (color, size, label).
- **Dashboard Tab**: Combine visuals.
- **Connect Pane**: Connect to CSV, Excel, SQL, etc.

2. ***Uploading the Dataset***

- Launch Tableau Desktop.
- Under "Connect", select CSV/Excel/Database.
- Browse and upload your dataset.
- Drag tables into the workspace if needed.

3. ***Handling Data***

- Use **Data Interpreter** for auto-cleaning.
- Create **Calculated Fields** for new variables.
- Use **Filters**, **Groups**, and **Bins** to organize data.

4. ***Plotting Graphs***

- Drag fields to Rows and Columns.
- Use Show Me panel to choose chart types (bar, line, scatter, pie, maps).
- Build dashboards combining multiple visuals.

5. **Summary : Generating Descriptive Statistics**

Operation	Tableau Feature Used
Median, Quartiles & Outliers	Box Plot from Analytics Pane
Summary Statistics	Show Summary (Worksheet tab)
Distribution Pattern	Histogram + Distribution Bands
Quick statistical insights	Summary Card (can be dragged anywhere on screen)

Table 3: Brief of generative statistics

These components help quickly identify:

- Data spread and center

- Skewness in sales
- Outlier products with extreme performance

6. Final Output (Expected)

A **dashboard** visually showing:

- Sales distribution chart
- Box plot with median & quartiles
- Optional supplier/region filter
- Summary card displaying statistics like:
  ✓ Count
  ✓ Mean
  ✓ Minimum
  ✓ Maximum

☞ Check Your Progress - 1

1. What are the key components of the Tableau workspace?

   ……
   ……
   ……
   ……

2. How do you connect to a Microsoft Excel file in Tableau?

   ……
   ……
   ……
   ……

3. What is the purpose of the 'Show Me' panel in Tableau?

   ……
   ……
   ……
   ……

4. How can you calculate the median in Tableau?

   ……
   ……
   ……
   ……

5. How are outliers identified using Tableau?

   ……
   ……
   ……

13.6 POWER BI

Power BI is a Microsoft application built to convert raw data into valuable insights.

It enables users to create interactive dashboards, reports, and charts to simplify data analysis and understanding. Whether you're in business, research, or any data-driven field, Power BI helps you identify patterns, track trends, and make informed decisions more efficiently.

The process involves three main steps:

1. **Power Query Editor** – Prepare and clean your data.
2. **Data Modeling and Relationships** – Establish connections between data tables and organize your data.
3. **Visualization** –Build interactive graphs and charts to explore and showcase your data efficiently.

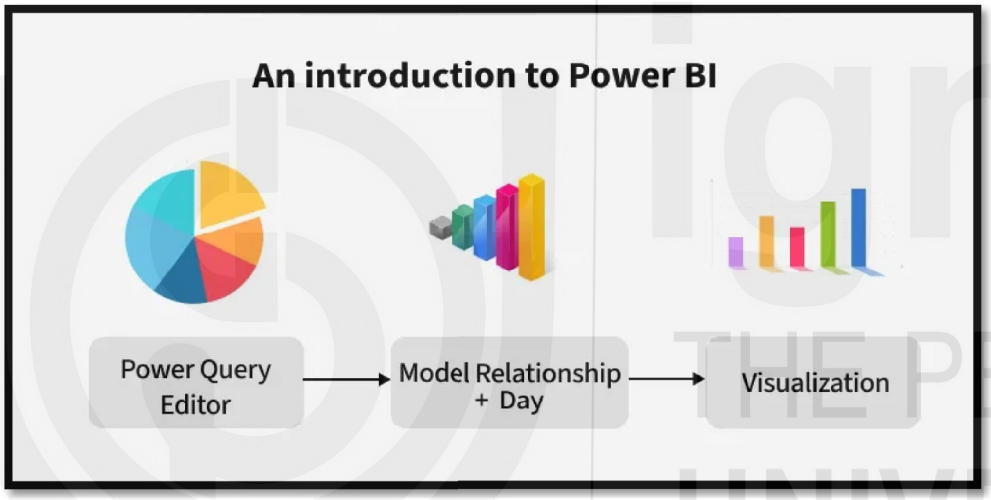


Fig.28: Getting Started with Power BI

Working with Power BI is straightforward, it follows the easy steps, shown in Fig. 29 below

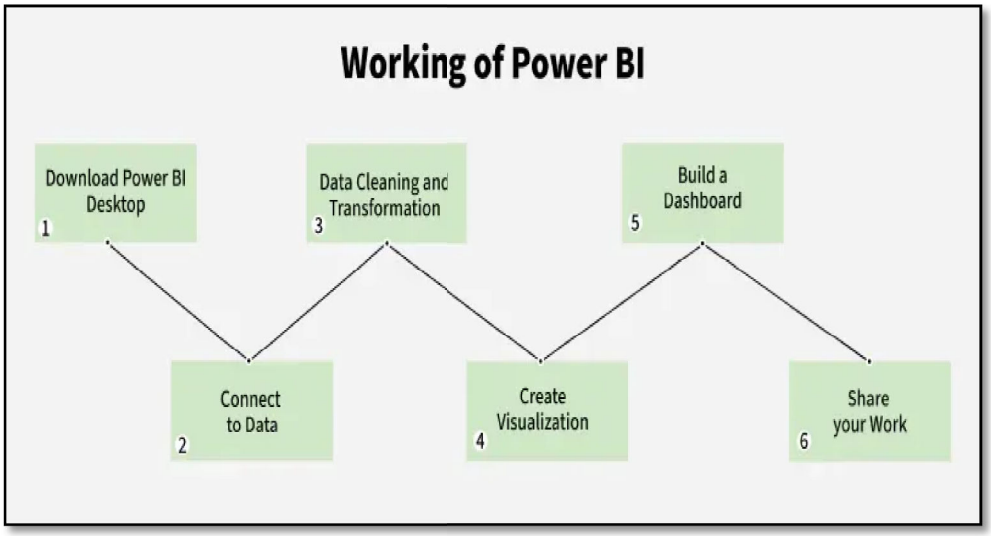


Fig.29: Working of Power BI

Step-by-Step procedure to work by Using Power BI is as follows :

Step 1: Download and Install Power BI Desktop: Go to Microsoft's official website and download Power BI Desktop for free. Install it on your Windows system. This version enables you to create dashboards and reports. After installation, you'll get a clean interface to start working with your data.

Step 2: Connect to Your Data: Open Power BI and click on **"Get Data."** You can connect to a variety of data sources such as Excel files, SQL databases, or cloud services like Google Analytics and Salesforce. Select your desired source and load the data. You can also combine data from multiple sources into one report.

Step 3: Clean and Prepare Your Data: Data often needs cleaning before analysis. Use **Power Query** to remove unnecessary rows, correct data formats, and handle missing values. This ensures your dataset is accurate and ready for analysis.

Step 4: Create Visualizations: Drag data fields into the report canvas to start building visualizations. Power BI will automatically generate charts, tables, or graphs based on the data. Choose visual types that best represent your insights. Visuals help in identifying patterns and key metrics quickly.

Step 5: Build a Dashboard: Arrange multiple visuals into a single-page dashboard. Dashboards allow users to explore different data perspectives by interacting with the visuals. You can customize layouts to highlight important information and enhance the user experience.

Step 6: Share Your Work: After completing your report or dashboard, publish it to the **Power BI Service** (online platform). This allows you to share your insights with others, enabling collaboration and easy access from anywhere others can access and view it at any time. You can also set up automatic data refreshes to keep your reports up to date, ensuring they always display the most recent information.

Using Power BI - Power BI is mainly used to create interactive reports and dashboards that provide deeper insights into your data. After publishing your report to the Power BI service, both you and other users can explore and interact with it while tracking its performance. This guide highlights the essential steps for building and modifying reports and dashboards.

Essential Steps:

1. **Extract the Data** – Begin by importing your data from various sources.
2. **Transform the Data** – Use Power Query to clean, modify, and prepare your data, as well as to manage and adjust relationships between tables as needed.
3. **Apply Calculations with DAX** – Use Data Analysis Expressions (DAX) to perform calculations on your dataset.
4. **Start Visualizing** – Once the data is prepared, proceed to build your visualizations.

5. **Create and Customize Visuals** – Add elements like charts, graphs, and cards, and modify them for clarity and effectiveness.
6. **Publish to the Cloud** – Upload your dashboard to the Power BI service so your team can access and interact with it.

Loading Data - To start handling data in Power BI, select the **'Get Data'** option under the **Home** tab. This allows you to connect to a wide range of data sources, including:

- **Files** – Import data from formats like Excel or CSV.
- **Databases** – Establish connections to databases like SQL Server, Oracle, or MySQL.
- **Direct Query** – Access data directly from the source without importing it.
- **Online Services** – Connect with platforms like Google Analytics, Salesforce, or SharePoint.
- **Live Connection** – Establish a real-time link to data sources for always up-to-date information.

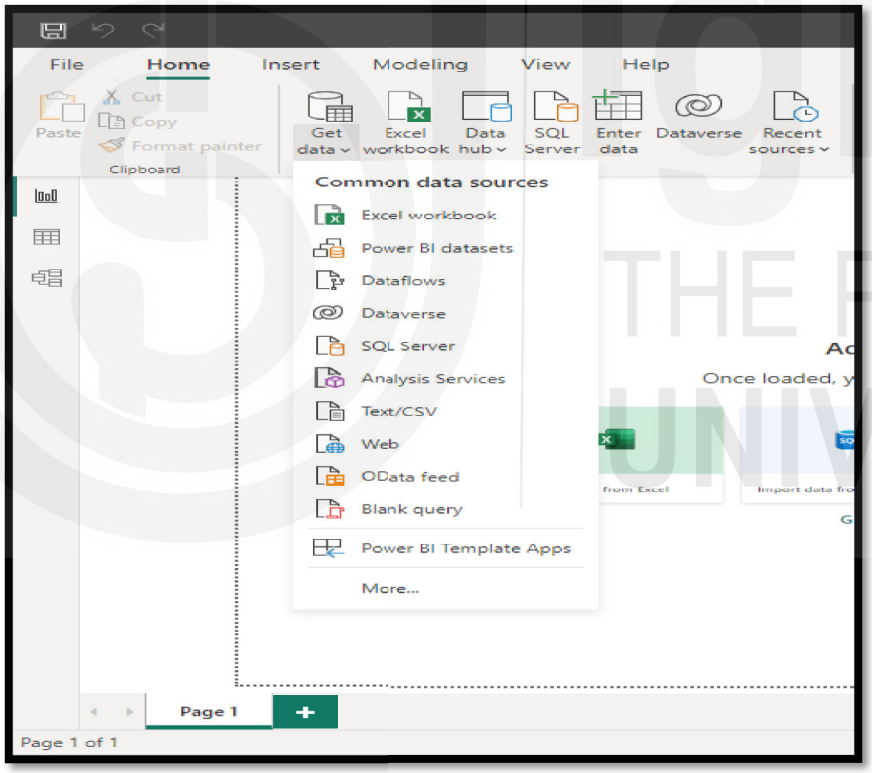


Fig.30: Using Power BI

After importing the data into the application, you can begin transforming and modeling it. You can create new columns with custom configurations and define your own calculated fields.

Before building meaningful dashboards and analytical reports, it is essential to prepare and structure the imported data. Raw data often contains missing fields, formatting inconsistencies, or lacks the calculated values required for analysis. Power BI provides powerful tools for transforming, enriching, and organizing data so that it aligns with analytical and business requirements.

Power BI enables users to enhance the existing dataset by creating **calculated columns** and **measures**. These user-defined fields help implement business logic, perform advanced calculations, and support accurate data insights that may not be directly available from the source. This process forms the core of **data modeling**, which ensures that the data is structured correctly for visualizations and interactions. The steps to create **calculated columns** and **measures are listed below:**

Step 1: Begin by clicking on the **"Data"** icon from the three options available in the left-hand panel.

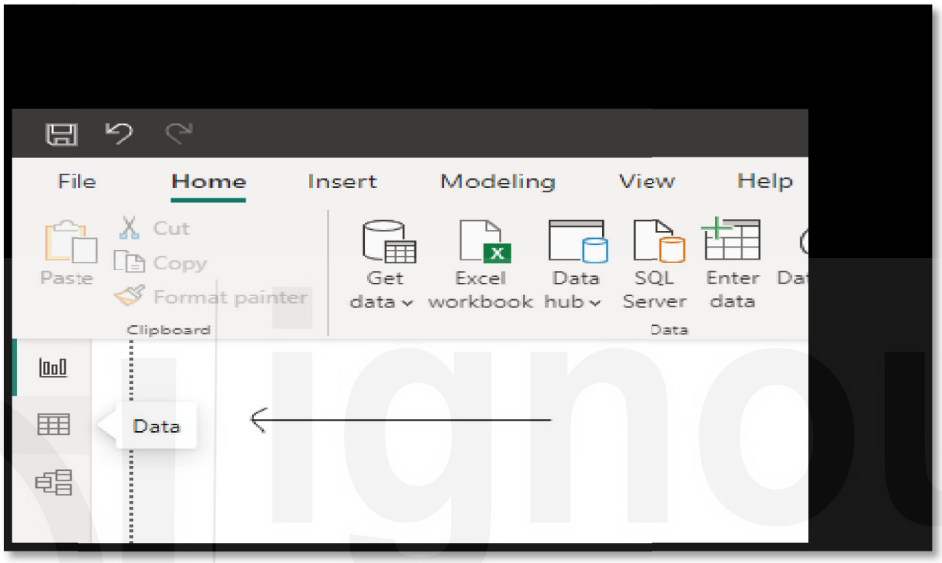


Fig.31: Data Tab Power BI

Step 2: In the **"Table Tools"** tab, go to the **"Calculation"** group and choose **"New Measure"** to create a custom calculation.

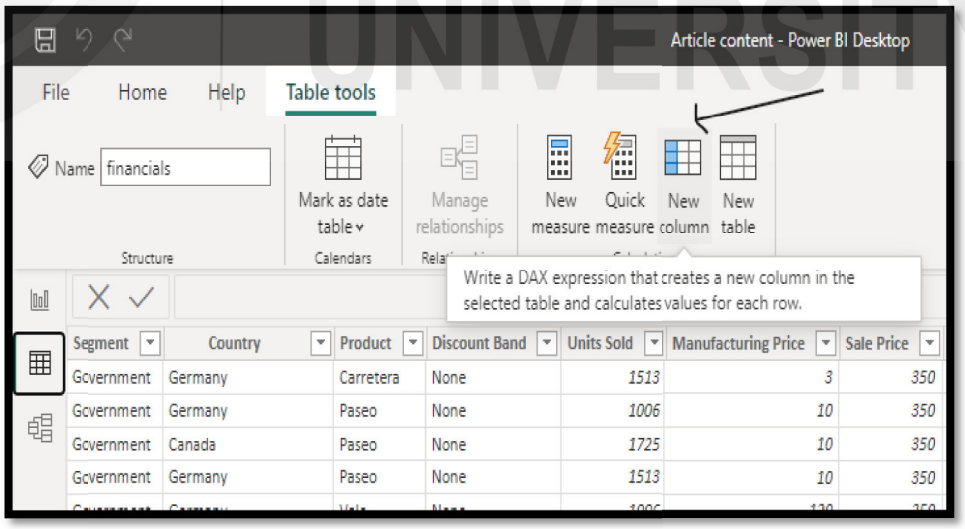


Fig.32: Table Tools Power BI

Step 3: Next, use the desired functions along with the table field names to define your logic and generate a new column.

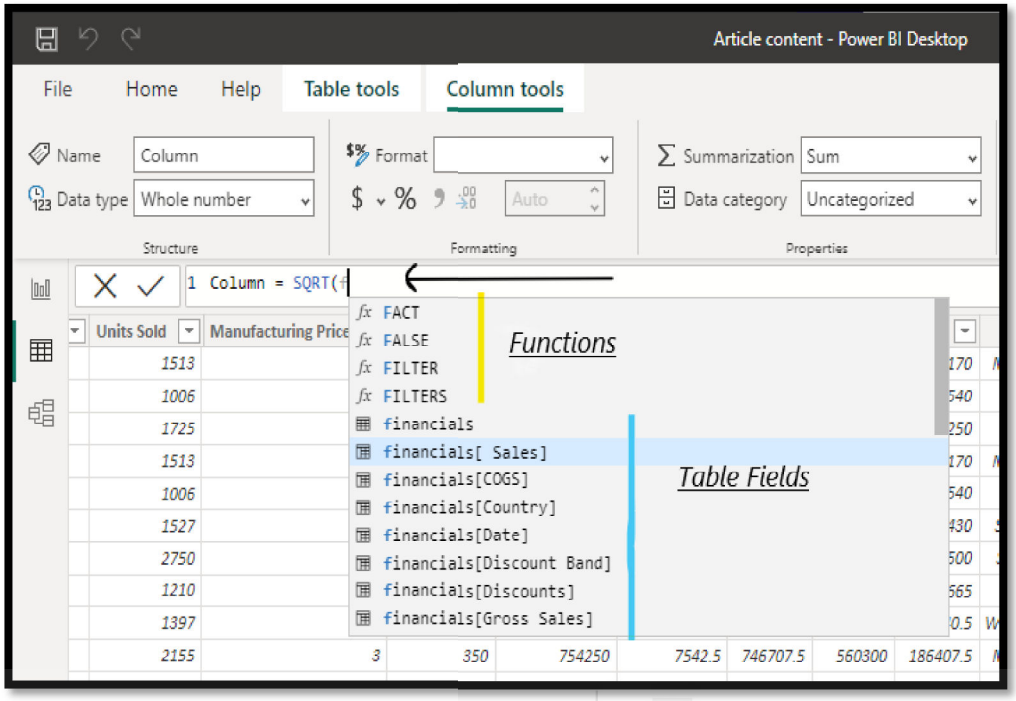


Fig.33: Column Tools Power BI

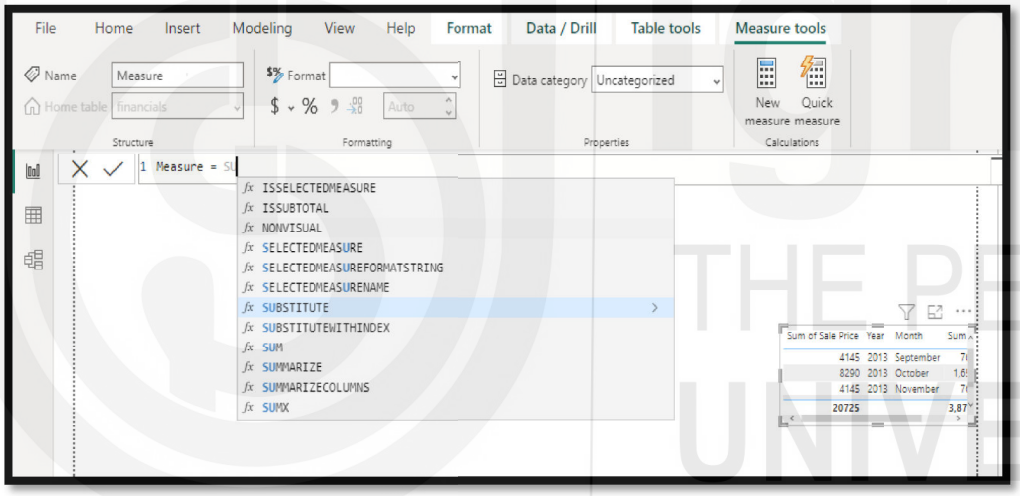


Fig.34: Measure Tools Power BI

Step 4: Once this step is finished, your required columns or measures will be set up. Use the **"New Column"** option when you need the column to remain part of the table and update automatically during data refreshes, as it becomes a permanent part of the data model. On the other hand, **measures** are calculated dynamically and function more like real-time filters—making them more efficient since they aren't stored in memory.

Power BI also offers a built-in **Power Query Editor** that supports advanced data cleaning and transformation, ensuring your data is well-prepared for analysis and visualization .

Power Query - Power Query is a robust tool that allows you to shape and transform your data through operations such as filtering, grouping, and pivoting. It also supports the creation of calculated columns and measures using the formula bar and a wide range of functions. After the data is cleaned and prepared, you can move on to creating visualizations. Following are the steps to be performed to prepare the data for its visualization.

Step 1: Click on the **"Home"** tab from the top navigation menu.

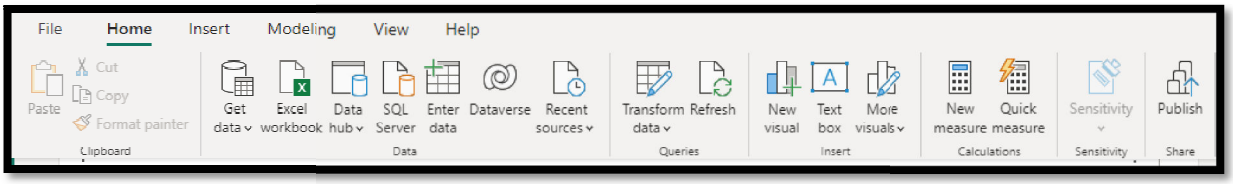


Fig.35: Home Tab Power BI

Step 2: In the **Queries** section, click on **"Transform Data."** (This action will launch the Power Query Editor in a new window. If no data source is connected yet, a blank screen will be displayed.)

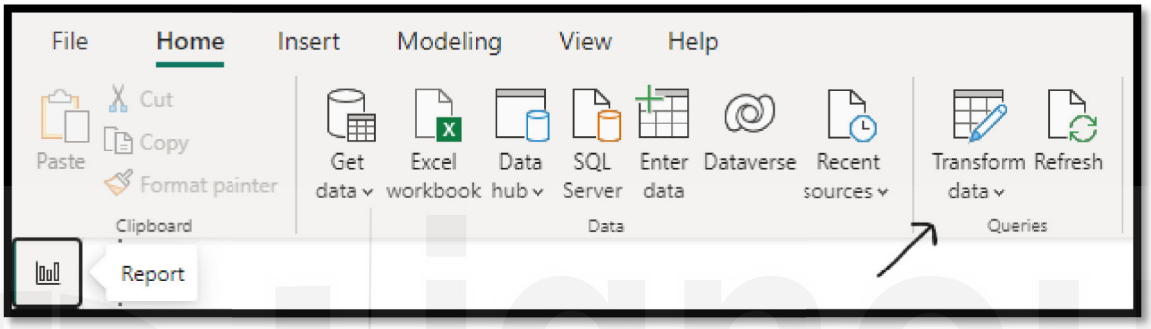


Fig.36: Transform Data Tab Power BI

Step 3: On this screen, you can view, edit, transform, and clean your data as required.

The **Home** tab provides multiple tools for managing your data sources.

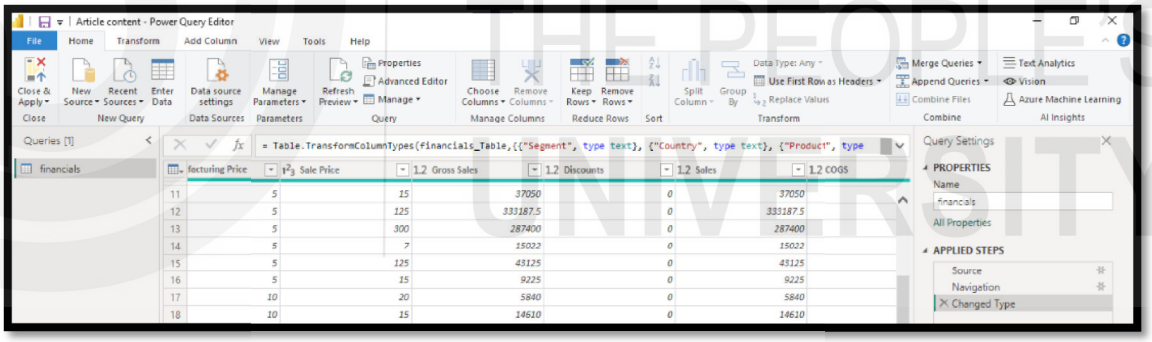


Fig.37: Power Query Editor in Power BI

The "Transform" tab contains functions for adding, deleting, splitting, and changing data types.

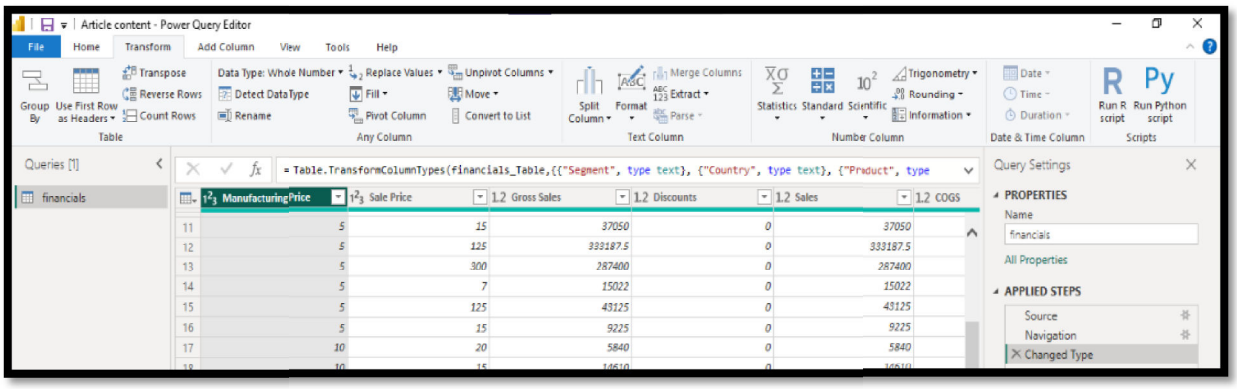


Fig.38: Power Query Editor Transform Tab

"The 'Add Column' tab allows you to add new columns and format them accordingly."

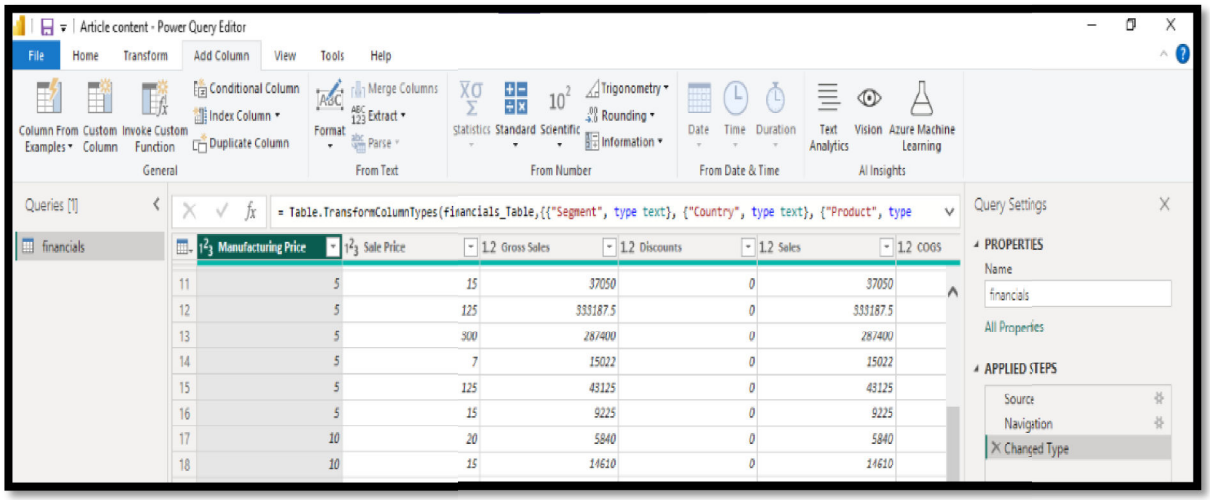


Fig.39: Power Query Editor Adding Column

The **'View'** tab manages the way the page is displayed.

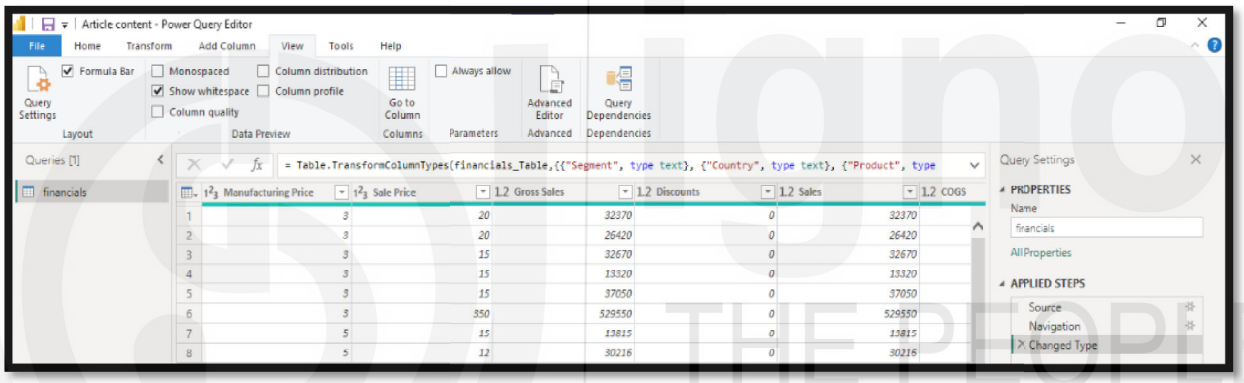


Fig.40: Power Query Editor View Tab

Step 4: By right-clicking on your column, you can apply various modifications.

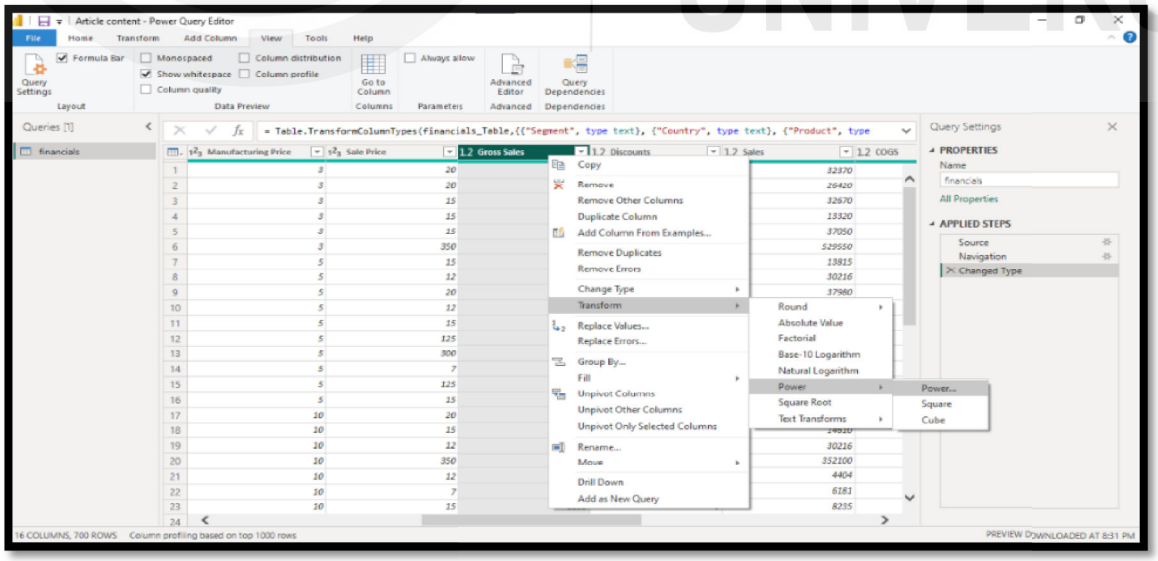


Fig.41: Power Query Editor View Tab

Step 5: After finishing the necessary data transformations, navigate to the **'File'** tab and choose **'Close & Apply'** to save your changes and return to Power BI Desktop.

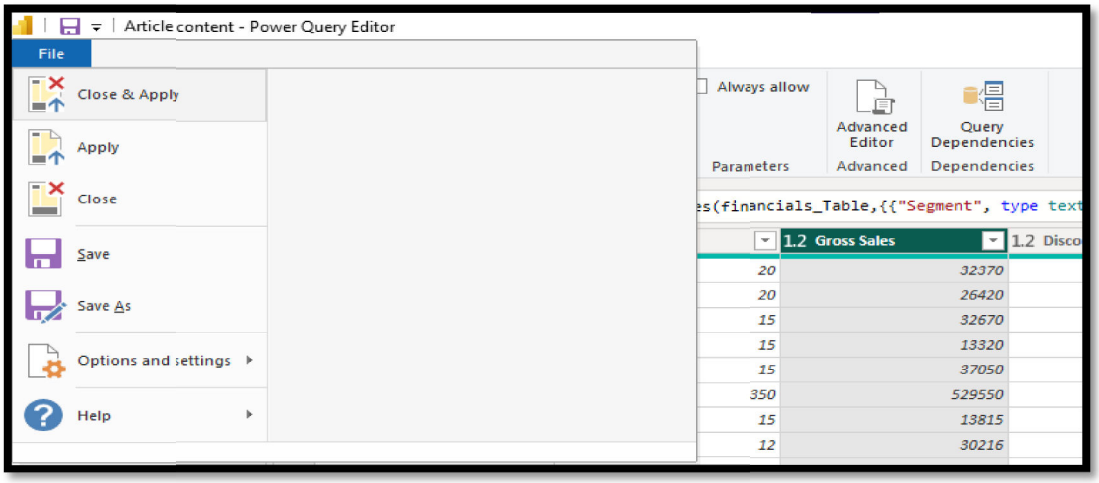


Fig.42: Power Query Editor View Tab

Your data is now ready to be used in creating visualizations and generating reports.

13.6.2 Visualization in Power BI

In Power BI, the Format and Fields panes play a vital role in customizing and enhancing the visual appearance of reports and dashboards. The Fields pane is used to manage and organize the dataset by selecting, adding, or removing fields for different visualizations, while the Format pane allows users to control the visual properties of charts and reports such as colors, titles, labels, backgrounds, data bars, and tooltips. To adjust the overall layout and appearance of a report, users can utilize the View tab, where options such as Themes, Page View, Gridlines, and Snap to Grid are available. The Themes option provides multiple predefined styling formats that help maintain visual consistency and improve report aesthetics according to organizational or presentation requirements. Together, the Format, Fields, and View options in Power BI enable users to design professional, interactive, and visually appealing dashboards and reports.

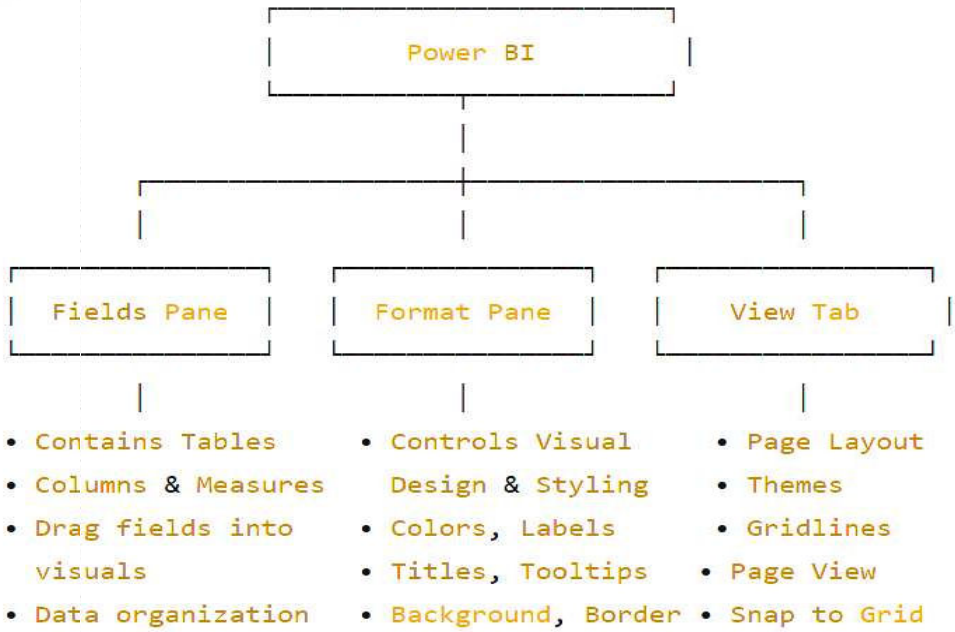


Fig.43: Power BI - Fields pane, Format Pane & View Tab

The **Format** and **Fields** pane is where you customize and improve the appearance of your dashboards and reports. To adjust the layout of your report, use the **"Layout"** option under the **View** tab, which provides various theme options to match your report's design.

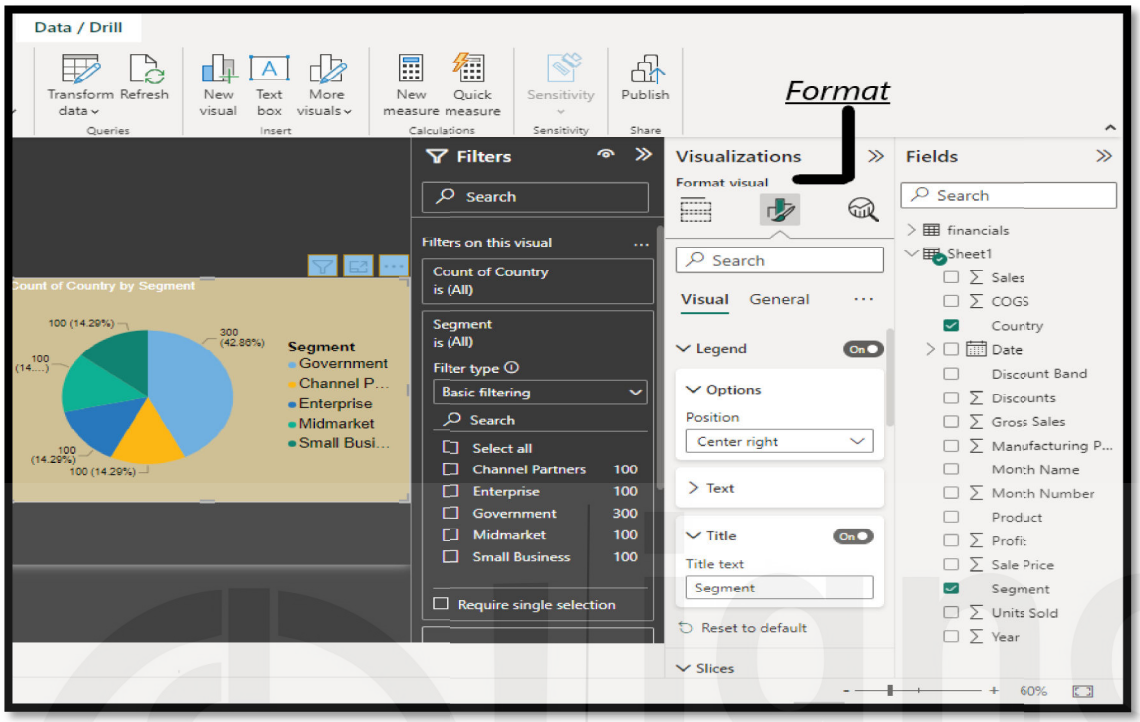


Fig.44: Visualization Tab

To modify a visualization already added to the report, select it and use the "Format" option to customize its appearance.

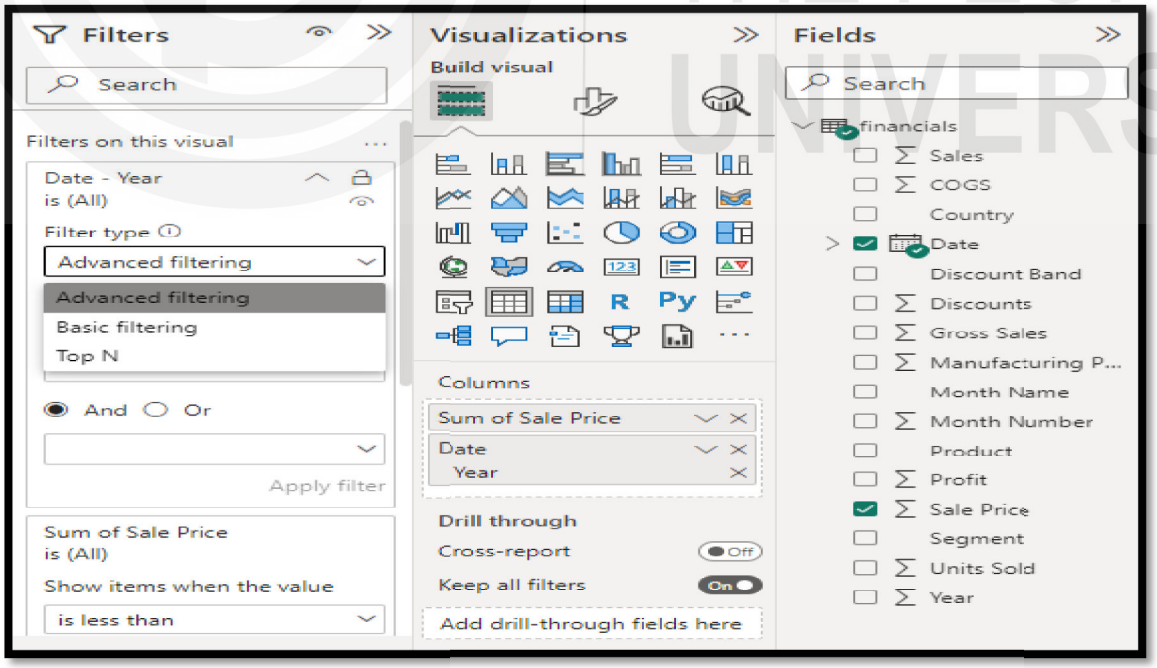


Fig.45: Visualization Tab

For more advanced customization of your visualizations, you can use **DAX formulas** to create calculated columns and measures as shown below in this section.

Data Analysis Expressions (DAX) is a powerful formula language used in **Power BI** (as well as in Excel Power Pivot and SQL Server Analysis Services) to perform **custom calculations on data** after it has been imported into the model. DAX helps users go beyond basic summaries by enabling the creation of **calculated columns, measures, and calculated tables**, which add new analytical meaning to existing data.

A **calculated column** works row by row and is mainly used to add new derived data to a table, such as profit calculated from sales and cost. A **measure**, on the other hand, is used to perform dynamic calculations such as totals, averages, percentages, or ratios that automatically change based on filters and slicers applied in the report. This dynamic behavior is what makes DAX especially valuable for interactive dashboards.

For example, a simple DAX measure like:

Total Sales = SUM(Sales[Amount])

calculates the total revenue by adding all sales values from the Sales table. This measure updates instantly when a user applies filters such as product category, date, or region. In this way, DAX allows reports to become **interactive and responsive**, which is essential for business analysis.

DAX also supports more **advanced analytical functions**, such as **time-intelligence calculations**. These functions make it possible to compare performance across time periods, such as month-to-month growth, year-over-year growth, or running totals. For example, a **Year-over-Year (YoY) Sales Growth** measure compares current year sales with previous year sales and expresses the change as a percentage. Such calculations are extremely useful for trend analysis and performance evaluation.

Basic Steps of Working with DAX in Power BI

Step 1: Load Data into Power BI

1. Open **Power BI Desktop**.
2. Click on **Home → Get Data**.
3. Select **Excel / CSV / Text file**.
4. Browse and load your dataset.
5. Click **Load** to import the data into Power BI.

After loading the dataset:

- Use **Data View** to see the table.
- Use **Report View** to create visuals and apply DAX measures.

6. Create a Basic DAX Measure

1. Go to **Report View**.
2. In the **Fields Pane**, right-click on the table name (e.g., *Sales*).
3. Click on **New Measure**.
4. In the formula bar, type the DAX formula:

Example, Total Sales = SUM (Sales [Price])

5. Press Enter

This measure calculates the **total sales amount dynamically**.

In short, **DAX acts as the analytical brain of Power BI**. It allows analysts to create meaningful metrics, perform complex calculations, compare performance over time, and build interactive dashboards that support informed decision-making. Even at an introductory level, understanding basic DAX concepts like measures, filters, and aggregation functions provides a strong foundation for advanced business intelligence and data analysis tasks.

This measure dynamically evaluates sales growth relative to the previous year based on the current filter context. Thus, the use of DAX formulas significantly enhances analytical flexibility and supports the development of meaningful business insights within Power BI.

13.6.3 Generation of Descriptive Statistics in Power BI

Descriptive statistics are used to summarize and describe the basic features of a dataset in a simple and meaningful way. In Power BI, descriptive statistics are generated using a combination of built-in visual tools and DAX (Data Analysis Expressions) measures, allowing users to quickly understand patterns, trends, and overall performance of data.

Power BI provides built-in analytical capabilities to summarize data and derive descriptive statistics such as totals, averages, minimum and maximum values, counts, percentages, and distributions. Once a dataset is imported into Power BI Desktop, visual elements like Cards, Tables, Matrix, and Clustered Bar/Column Charts can be used to present summary statistics. Additionally, DAX measures enhance the analytical scope by enabling dynamic calculations based on user interactions. Common descriptive statistical measures in Power BI can be generated with simple DAX formulas

To create more flexible and dynamic statistics, Power BI allows users to define custom DAX measures. Simple DAX formulas such as SUM, AVERAGE, DISTINCTCOUNT, MIN, and MAX are commonly used to compute descriptive measures. For instance, Total Quantity = SUM(OrderDetails[Quantity]) calculates the total number of items sold, Average Price = AVERAGE(Product[Price]) gives the typical price of products, and Product Count = DISTINCTCOUNT(Product[ProductID]) returns the number of unique products. Similarly, MIN(Product[Price]) and MAX(Product[Price]) identify the lowest and highest product prices in the dataset. These metrics update automatically based on report filters or slicers, helping users gain real-time insights into product performance, customer preferences, and revenue trends.

One of the major advantages of using Power BI for descriptive statistics is that these values are dynamic. This means the results automatically change whenever a user applies filters or slicers, such as selecting a particular product category, time period, or region. As a result, users can instantly observe how key metrics vary across different conditions.

We learned that, Power BI simplifies the generation of descriptive statistics by combining easy-to-use visuals with powerful DAX calculations. This enables beginners to summarize large datasets, uncover basic patterns, and gain quick insights into product performance, customer behavior, and overall business trends in an interactive and user-friendly manner.

A) Creation of Dashboard Steps in Power BI

The dashboard development process in Power BI involves connecting to data, transforming it as needed, and building interactive visuals. The essential steps are:

Step 1: Importing Data

To begin working in Power BI, the first task is to load the required dataset into the application. Power BI enables data import from multiple sources, including Excel, CSV, databases, and online services. Click on the "Get Data" option located on the left side of the Home screen to browse and select your data file. Once the desired dataset is chosen, it can be easily loaded into the workspace for further processing and analysis.

- Open Power BI Desktop → Click Get Data
- Select the data source (Excel/CSV/Database/etc.)
- Load the dataset into the Data Model

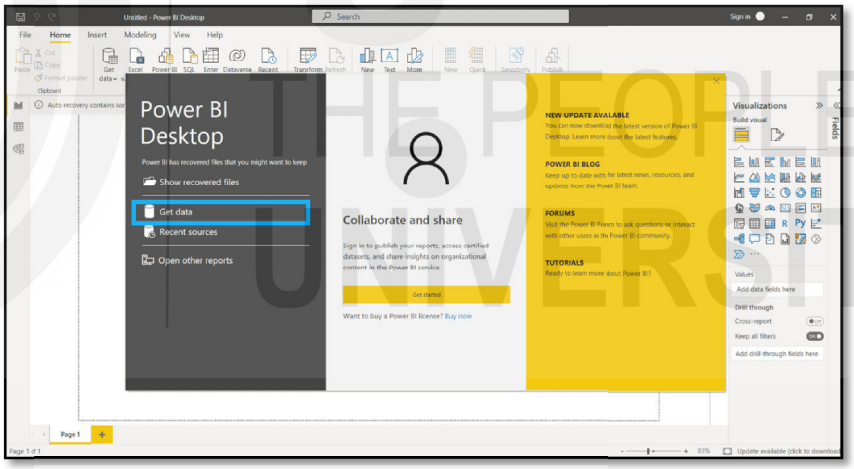


Fig.46: PowerBI Dashboard

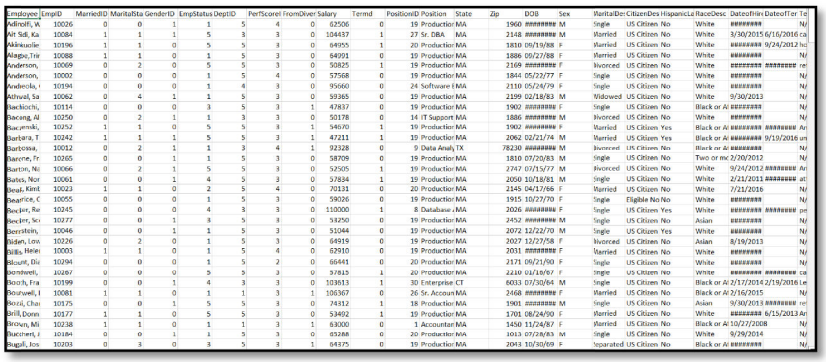


Fig.47: Sample Dataset

The Navigation pane provides an option labeled **Files**, which allows users to

access data stored locally. Select **Files** and browse to the folder where the Excel workbook or other supported data format is saved. After selecting the appropriate file, click **Connect** to establish the data import into Power BI.

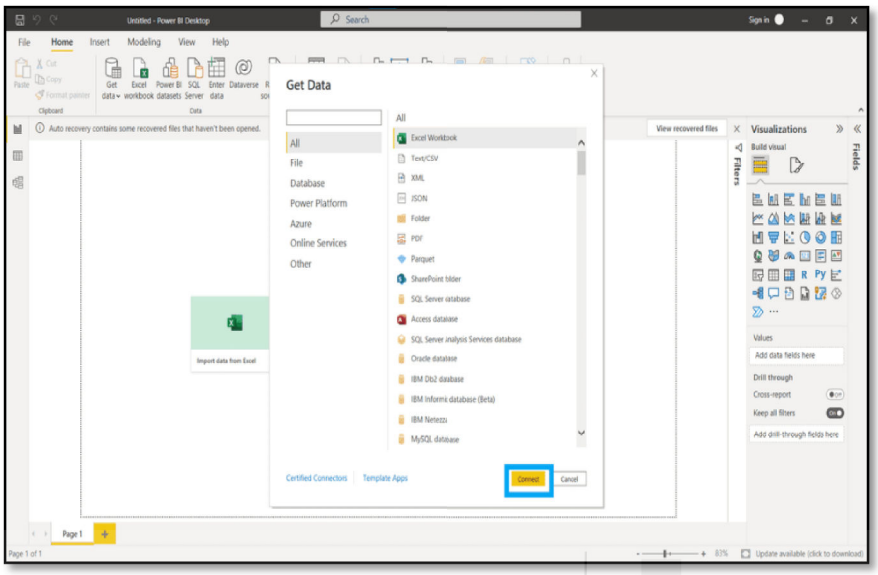


Fig.48: PowerBI Data Upload

Depending on the dataset size, Power BI may require a short processing time to extract the data. Once the preview is displayed, verify that the data has been imported correctly and then click **Load** to bring the dataset into the Power BI workspace.

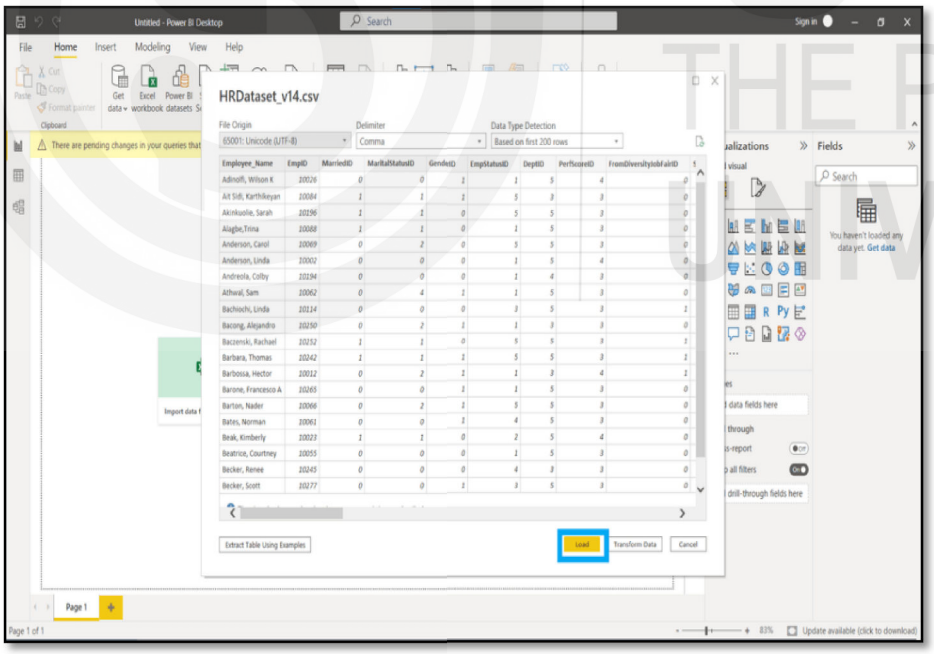


Fig.49: HR Dataset Upload

Step 2: Exploring the Data

After the data is loaded, navigate to the **Data View** to examine the dataset in a structured tabular format. The **Fields Pane** on the right side of the interface displays all available tables along with their respective columns, allowing you to review and understand the attributes present in the imported data.

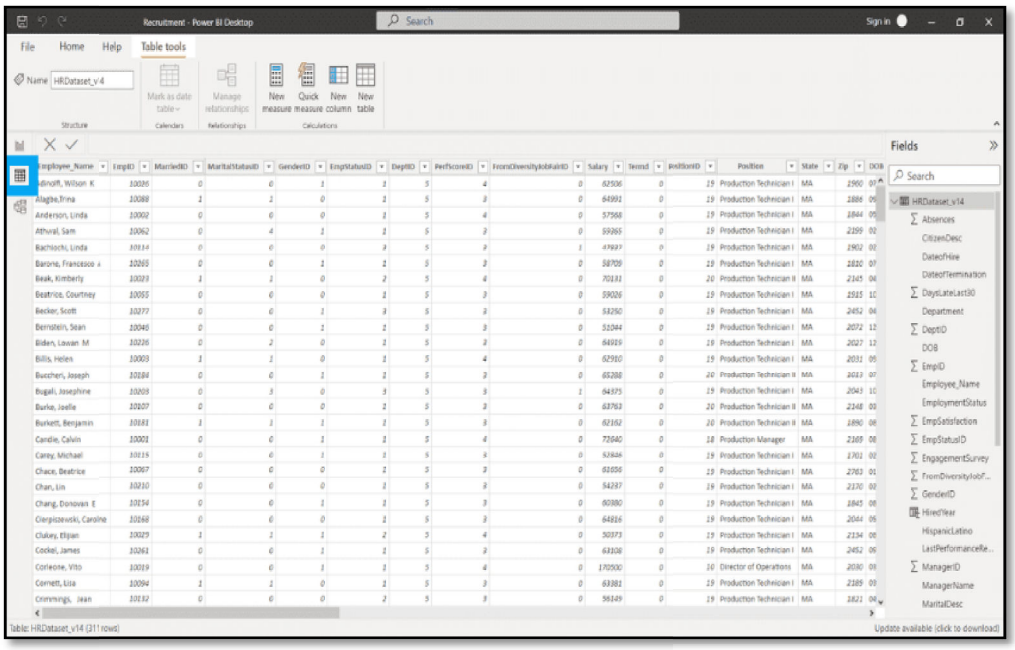


Fig.50: Data View

You can click on any table or individual field to apply formatting options as required. For attributes representing values such as dates, time, geographical locations (city, state), percentages, or currency, the appropriate **data type and formatting** can be assigned using the tools available under the **Modeling** tab.

Step 3: Selecting an Appropriate Visualization

For developing the dashboard, we focus on five key fields: **HiredYear**, **RecruitmentSource**, **Position**, **EmployerID**, and **Gender (Male/Female)**. The first visual to be added is a **Card** visualization. To create it, simply select the **Card** icon from the **Visualizations** pane and place the relevant field onto the canvas.

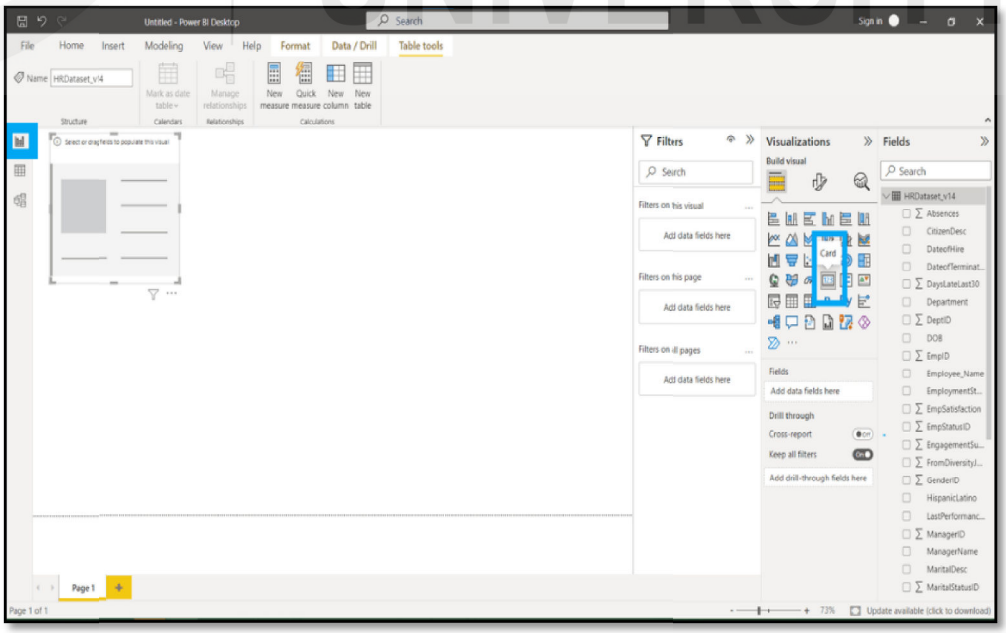


Fig.51: Table Tools

Next, choose the fields that you want to include in the visualization from the **Fields Pane**. You may either click to add them or simply **drag and drop** the selected fields into the corresponding placeholders within the visualization, as indicated in the interface.

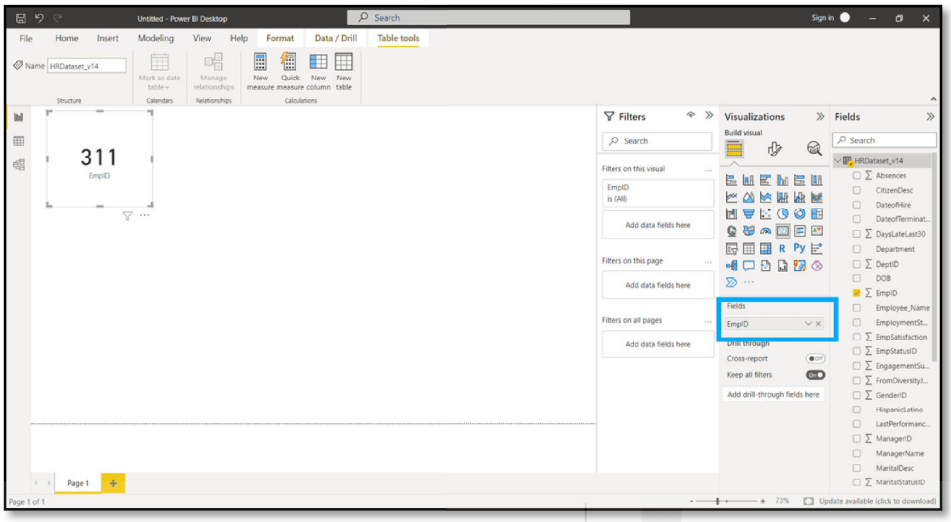


Fig.52: Field Pane

You can customize the visualization further by selecting fields, applying filters, and adjusting appearance settings using the **Format** options. The first Card visualization created displays the **total number of Employers**, providing a quick summary metric for the dashboard.

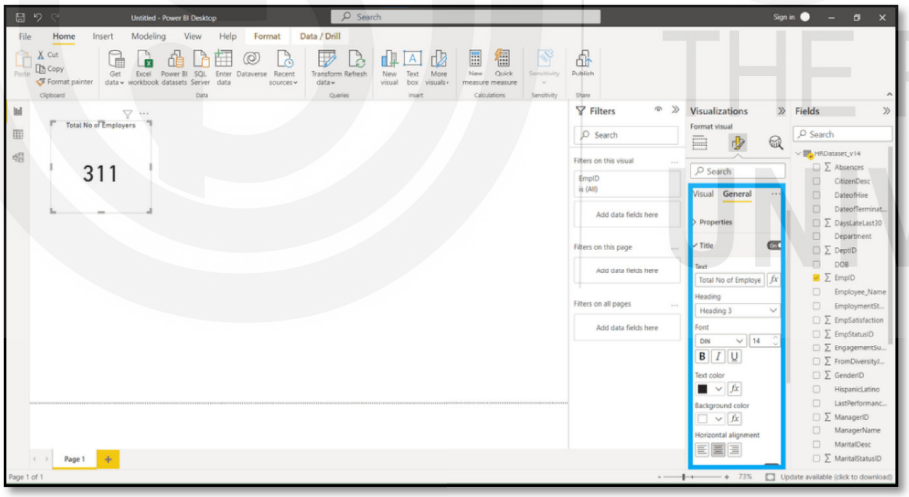


Fig.53: Visualization Pane

The same procedure was repeated to create the remaining Card visuals. These additional cards represent key summary indicators, including the **total number of Recruitment Sources**, **total number of Positions**, and the **Maximum Salary** value present in the dataset. Together, these visuals provide an at-a-glance statistical overview that supports further analysis within the dashboard.

Next, a **Pie Chart** and a **Donut Chart** are created to represent the **distribution of job positions** and the **gender-wise employment ratio**, respectively. These charts can be inserted by selecting the corresponding

icons from the **Visualizations** pane and assigning the required fields to the chart areas.

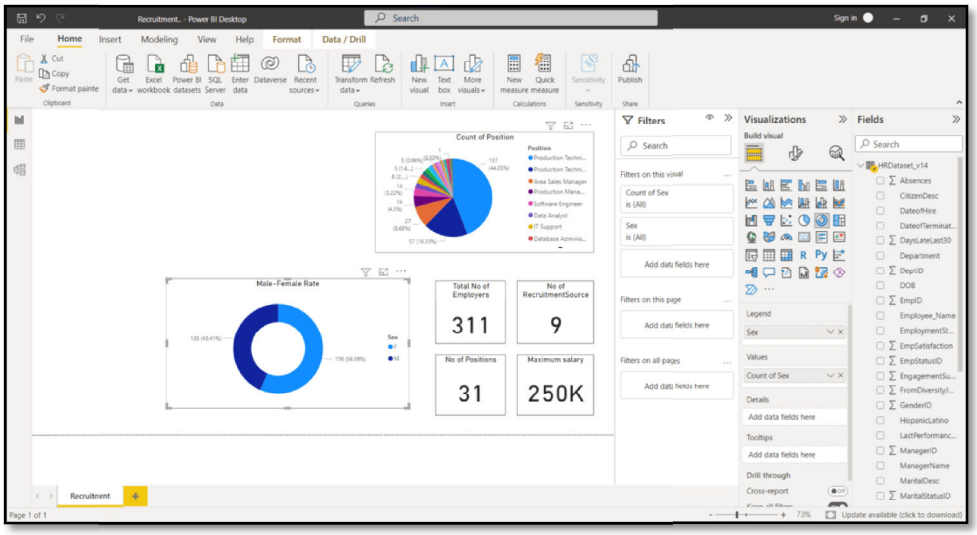


Fig.54: Types of Charts

Finally, a **Funnel Chart** and a **Stacked Bar Chart** are added to visualize the **year-wise hiring counts** and the **proportional contribution of different recruitment sources**, respectively. To enhance clarity and presentation quality, formatting options such as **titles, data labels, axis properties, legends, plot area settings, and color schemes** are applied. These adjustments ensure the dashboard communicates insights effectively, as illustrated in the figure below.

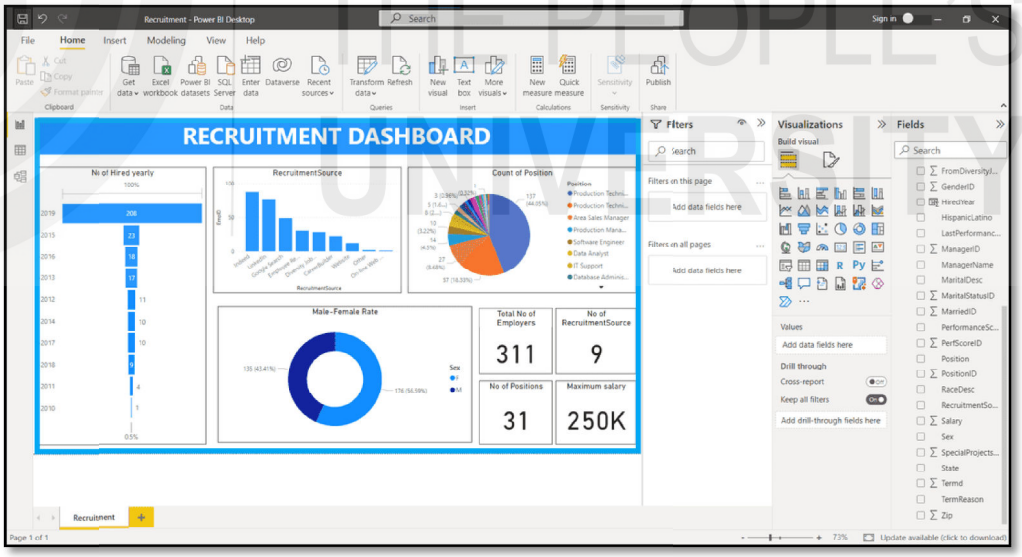


Fig.55: Visual demonstration of Recruitment Dashboard

Step 4: Publish and Share

- Publish report to Power BI Service
- Create dashboards by pinning visuals
- Share insights through secure cloud access

13.6.4 SUM UP: Power BI - User Interface and Data Handling

1. ***Navigating Power BI***

- **Home Pane**: Get Data, Transform Data.
- **Fields Pane**: Lists all columns and tables.
- **Visualizations Pane**: Select and customize charts.
- **Report View**: Main canvas for building reports.

2. ***Uploading the Dataset***

- Launch Power BI Desktop.
- Select **"Get Data"** and choose your source (Excel, CSV, or Database).
- Use the **Power Query Editor** to load or clean and shape your data as needed.

3. ***Handling Data***

- Use Power Query to clean your data by removing rows and adjusting data types.
- Generate new columns and perform calculations with DAX (Data Analysis Expressions).
- Establish **Relationships** between tables.

4. ***Plotting Graphs***

- Use drag-and-drop to build visuals on canvas.
- Available charts: bar, line, scatter, map, waterfall, KPI, donut.
- Interactivity via slicers and filters.

5. ***Example of a DAX Formula***

Power BI allows users to create custom calculations using DAX (Data Analysis Expressions).

Below is an example formula that calculates the **Total Sales** based on Price × Quantity:

$$Total\ Sales\ =\ SUMX(OrderDetails, OrderDetails[Quantity]\ *\ OrderDetails[Price])$$

This calculated measure dynamically updates based on filters used in the report and helps in analyzing revenue trends across different dimensions such as products, categories, or properties.

☞ Check Your Progress-2

1. What is Power Query used for?

……

……

……

……

……

2. How can you create a new measure in Power BI?

   …….

   …….

   …….

3. What are calculated columns in Power BI?

   …….

   …….

   …….

4. How do you upload a dataset in Power BI?

   …….

   …….

   …….

5. How do you publish a report in Power BI?

   …….

   …….

   …….

13.7 PYTHON

This section is an Introduction to Python Interface (Anaconda & Google Colab) - **Python IDLE, Jupyter Notebook, and Google Colab** all support Python development but differ in interface and capabilities, a brief introduction to each is given below:

- **Python IDLE**: A simple, built-in IDE with a Python Shell for interactive code and an Editor for scripts. It's lightweight and good for beginners and small projects.
- **Jupyter Notebook**: A web-based tool with a cell-based layout supporting live code, text (Markdown), and visualizations. Ideal for data science, research, and documentation.
- **Google Colab**: A cloud-hosted version of Jupyter with added benefits like free GPU/TPU access, Google Drive integration, and real-time collaboration. Best suited for machine learning tasks.

Before we start diving into Python coding, let's learn how to install Python IDLE**.**

Installing Python and Accessing IDLE : To work with Python locally (instead of Google Colab), we use **Python IDLE**, which comes bundled with the official Python installation. The steps to be followed for download and installation of Python IDLE are as follows:

Steps to Download & Install Python

1. Go to the official Python website: https://www.python.org
2. Click on **Downloads** and choose the version appropriate for your operating system (Windows/Mac/Linux).
3. Run the installer.
4. **Important:** Check the box **"Add Python to PATH"** before clicking Install.
5. Complete the installation. IDLE will be installed automatically.

Opening Python IDLE : Once Python is installed, you can start IDLE by following the steps below:

- On **Windows**: Search for **IDLE (Python GUI)** in the Start Menu.
- On **Mac/Linux**: Run idle3 from the terminal or locate it in Applications.

Now we can start writing Python code locally using the IDLE editor and the steps to be followed are as follows:

1. Open Python IDLE
- Windows: Search for "IDLE" in the Start Menu and open it.
- macOS: Use Spotlight to search for IDLE or open from Applications > Python.
- Linux: Run idle or idle3 in the terminal.
2. Use the Python Shell
- A window opens called the Python Shell, as shown below :

Top of Form

You can enter Python code right here and view the output immediately.

- Example:

3. Open a New File (Script Mode)
- Go to **File > New File** or press **Ctrl+N**.
- A new editor window opens where you can write multiple lines of code.
4. Write Your Program
- Type your Python program in the new window.

  Example:

```python
print("Hello, world!")
```

5. Save Your File

- Navigate to **File > Save** or press **Ctrl+S**.
- Save the file using a **.py** extension, such as **hello.py**.

6. Run Your Program

- In the editor window, go to **Run > Run Module** or press **F5**.
- The output appears in the Python Shell window.

7. Use Basic Features

- IDLE also includes:

✓ **Syntax highlighting** in the editor

✓ **Auto-indentation**

✓ **Basic debugger** (under Debug > Debugger)

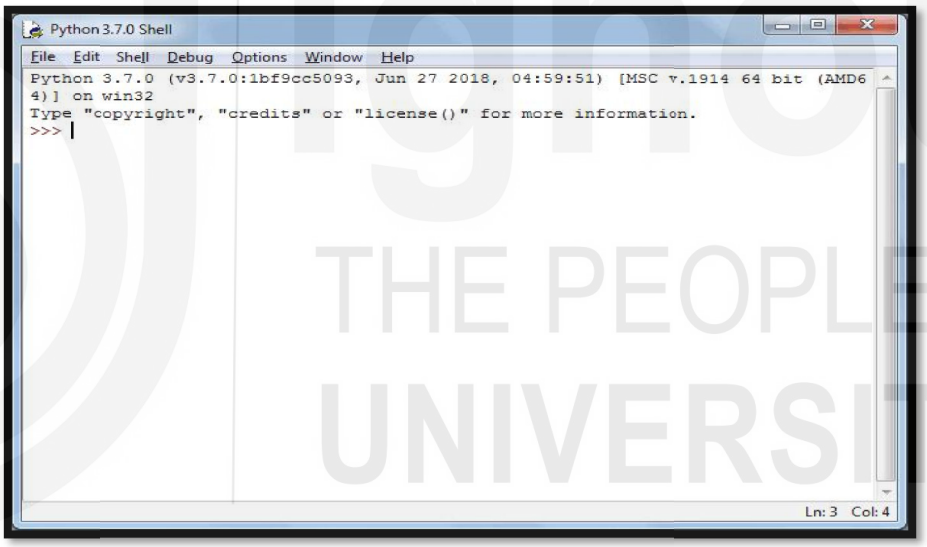


Fig .63: Python IDLE Interface

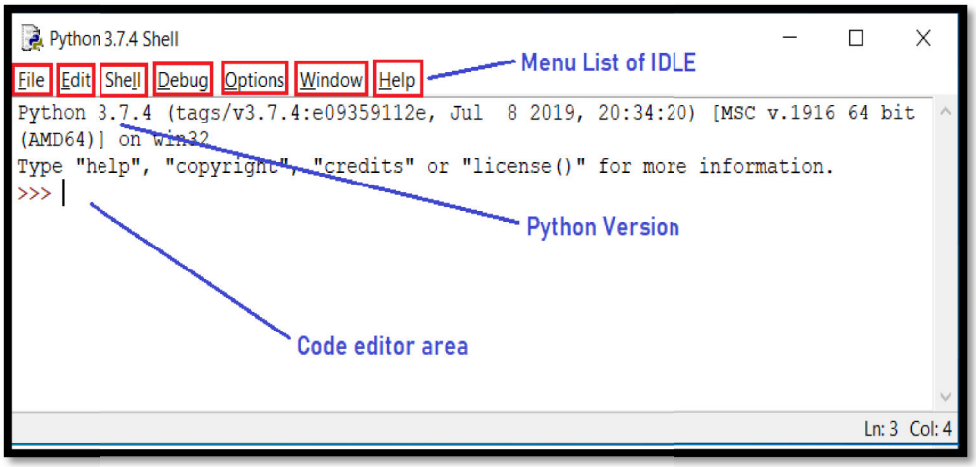


Fig .64: Python IDLE shell Interface

13.7.1 Steps to Use Jupyter Notebook

1. Install Jupyter Notebook (if it isn't already installed)

- You can install it using pip:

```
pip install notebook
```

Or install via Anaconda (which includes Jupyter by default).

2. Start Jupyter Notebook

- Open a terminal or command prompt and enter the following command:

```
jupyter notebook
```

- A web browser will open showing the Jupyter dashboard (usually at http://localhost:8888).

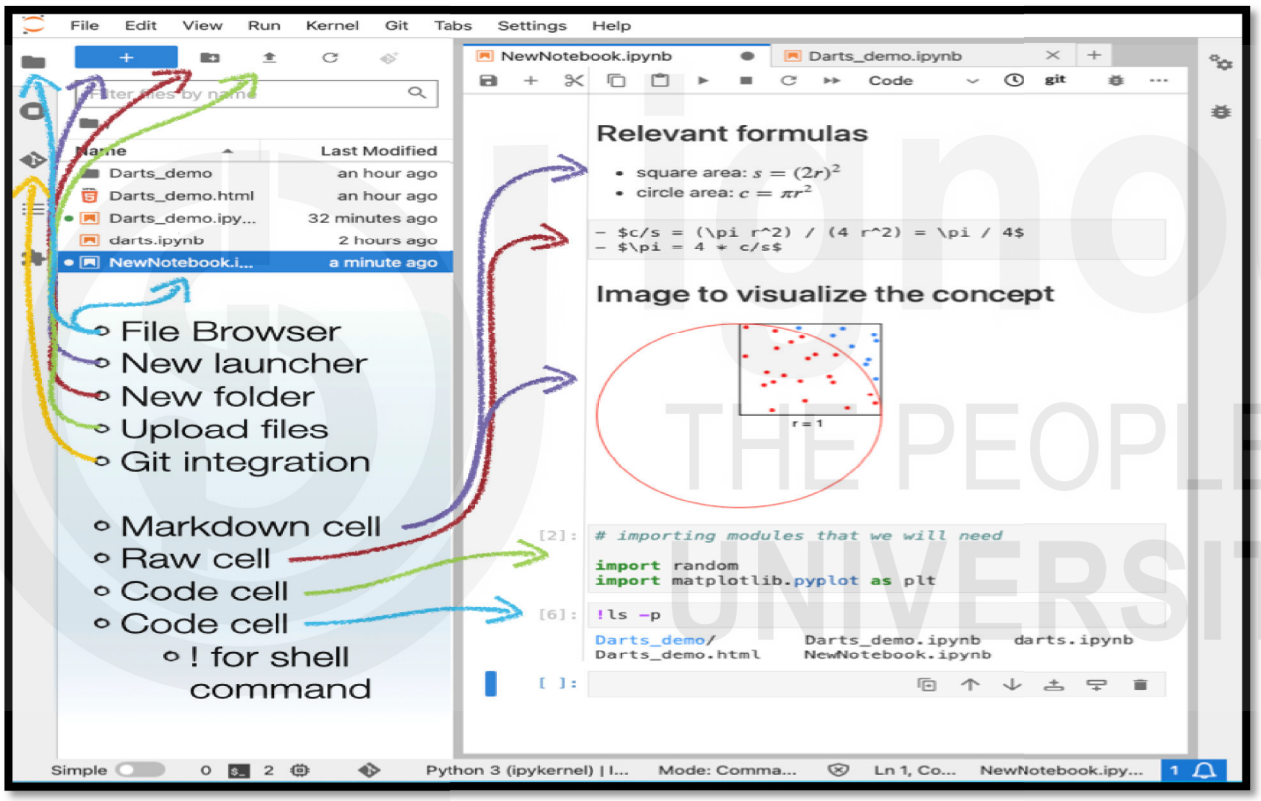


Fig .65: Python JupyterLab Interface

The JupyterLab interface features a toolbar on the left that includes a file browser and a Git tab. The file browser displays the directory from which JupyterLab was launched. On the right side, you can view and work with files you've opened. To enable a split view, simply drag a file tab from the top into another area of the workspace.

7. Create a New Notebook

- Click on **"New"** and select **"Python 3"** to launch a fresh notebook.
- A new tab opens with a blank notebook .

8. Write and Run Code

- Write your code in a **cell**.

Example:

```python
print("Hello, Jupyter!")
```

- Execute the cell by clicking the **Run** button or pressing **Shift + Enter [5]**.

9. Use Markdown for Notes

- Change a cell to **Markdown** (from the dropdown menu).
- Write formatted text using Markdown syntax.

10. Save Your Notebook

- Go to **File > Save and Checkpoint**, or use the shortcut **Ctrl+S**.
- It saves as a. ipynb file.

11. Close the Notebook

- Close the browser tab and shut down the server by pressing **Ctrl+C** in the terminal window.

13.7.2 Steps to Use Google Colab

1. Open Google Colab

- Go to: https://colab.research.google.com

2. **Sign In with Google Account**

- You need a Google account to use Colab.

3. **Create or Open a Notebook**

- Click **"File > New Notebook"** to start a new one.
- Or open an existing one from:
  - Google Drive
  - GitHub
  - Upload from your device

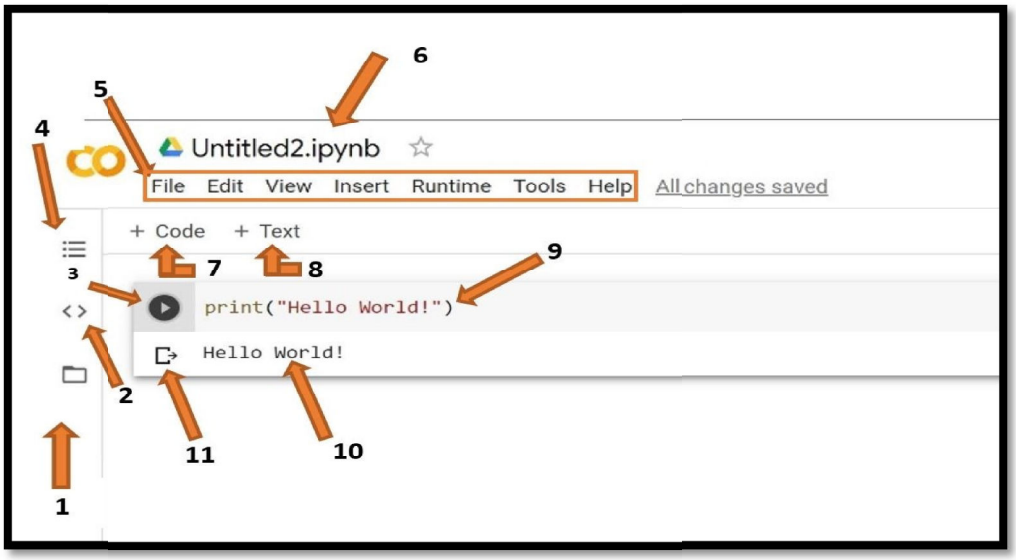


Fig.66: Python Google Colab Notebook Interface

4. **Write and Run Code**

- Enter Python code into a cell and press **Shift + Enter** to execute it.

5. **Add Text Cells**

- Click **+ Text** to add a Markdown cell for formatted notes or headings.

6. **Save the Notebook**

- It autosaves to **Google Drive**.
- To download, click **File > Download > .ipynb** or **.py**.

7. **Use Extra Features**

- You can:
  - Upload files using the sidebar or files.upload()
  - Mount Google Drive using:

```python
from google.colab import drive
drive.mount('/content/drive')
```

  - Use GPUs (under **Runtime > Change runtime type > GPU**)

Till now we learned how to work with Google Colab, Jupyter Notebook, IDLE etc., now lets understand how to work with python using datasets. To understand the working we used MovieLens Small Dataset, you can browse it on internet and download it from the link : https://grouplens.org/datasets/movielens/ ; We'll use the file: movies.csv

13.7.3 Steps to Upload Dataset in Python IDLE

IDLE is a basic IDE and doesn't have a built-in way to upload files — but you can **manually place the file in your script folder**.

Steps to be followed are:

1. **Download movies.csv** from MovieLens and save it in the same folder as your .py file.
2. Open IDLE > File > New File.
3. Write and run the code:

```python
import pandas as pd

# Load the dataset
data = pd.read_csv("movies.csv")

# Display first 5 rows
print(data.head())
```

4. Save and run the script (F5).

Steps to Upload Dataset in Jupyter Notebook are:

1. Open Jupyter Notebook.
2. Place movies.csv in the **same folder** where your notebook is saved.
3. Use this code to load the file:

```
import pandas as pd

# Load CSV
df = pd.read_csv("movies.csv")

# Show first 5 rows
df.head()
```

4. If you want to **browse and upload** via Jupyter:

- Click **Upload** in the Jupyter dashboard > Choose movies.csv > Click **Upload** button again.

13.7.4 Steps to Upload Dataset in Google Colab

Method 1: Upload from Local System

- Open a new Colab notebook.
- Use this code:

```
from google.colab import files
uploaded = files.upload()
```

- Select and upload movies.csv.
- Then use:

```
import pandas as pd
import io

df = pd.read_csv(io.BytesIO(uploaded['movies.csv']))
df.head()
```

Method 2: Load from Google Drive

1. Mount Google Drive:

```
from google.colab import drive
drive.mount('/content/drive')
```

2. Place movies.csv in your Drive, then access it like:

```
df = pd.read_csv('/content/drive/MyDrive/movies.csv')
df.head()
```

Till this point we learned "how to interact with the python tools like Google Colab, Jupyter Notebook etc?", and also we learned "how to upload the dataset?". Now, we will learn to generate the descriptive statistics in the subsequent section.

13.7.5 Descriptive Statistics in Python

Descriptive statistics summarize and describe the main characteristics of a dataset in a meaningful way. They help us understand the central tendency, spread, and overall distribution of the data. Common descriptive measures include the mean, which represents the average value, and the median, which identifies the middle value when the data is sorted. The mode indicates the most frequently occurring value. Measures such as variance and standard deviation describe how much the data varies from the mean, while minimum, maximum, and range provide information about the spread of the dataset. These statistics allow readers to quickly understand the nature of the data before performing advanced analysis.

So, we learned that Descriptive Statistics, helps us to understand the dataset before modeling, and it helps to detect patterns, outliers, or missing data, further it makes us to summarize large data into key insights quickly.

Let's Learn the process of Generating Descriptive Statistics in Python : To produce descriptive statistics in Python, the panda's library is used extensively. The following steps show how to compute them:

Step 1: Import pandas

```
import pandas as pd
```

Step 2: Create or Load a Dataset

```
data = {
'Marks': [65, 70, 85, 90, 75, 88, 92]
}
df = pd.DataFrame(data)
```

Step 3: Generate Descriptive Statistics

```
stats = df.describe()
print(stats)
```

Output will include:

- Count
- Mean
- Standard deviation
- Minimum value
- Maximum value
- 25th, 50th (Median), and 75th percentiles

Now, lets apply the understanding and try to explore, How to Perform Descriptive Statistics in Python?, Let's use the MovieLens ratings.csv (has user ratings of movies) for demonstration.

Step 1: Upload Dataset

(Use the same file upload method as discussed earlier in IDLE, Jupyter, or Colab.)

```
import pandas as pd

# Load ratings.csv (assumed to be uploaded in your current directory)
df = pd.read_csv("ratings.csv")

# Show first 5 rows
print(df.head())
```

Step 2: Apply Descriptive Statistics

```
# Summary of numerical columns
print(df.describe())
```

➢ **Output Includes:**

- **count** – total number of entries
- **mean** – average rating
- **std** – standard deviation (spread)
- **min**, **max** – lowest and highest values
- **25%**, **50%**, **75%** – percentiles (quartiles)

➢ **Additional Analysis**

```
# Count of unique users and movies
print("Unique users:", df['userId'].nunique())
print("Unique movies:", df['movieId'].nunique())

# Most common rating
print("Most frequent rating:", df['rating'].mode()[0])

# Rating distribution
print(df['rating'].value_counts())
```

➢ **Explanation with Example:**

Assume these are some ratings:

```
4.0, 5.0, 3.0, 4.5, 3.5
```

- Mean = average rating = (4 + 5 + 3 + 4.5 + 3.5) / 5 = 4.0

- Median = 4.0 (middle value)
- Mode = most frequent (if any)
- Standard **deviation** = how much ratings deviate from mean

➢ Sample table example:

Term	Meaning	Example
Mean	Average	4.0
Median	Middle value	4.0
Mode	Most common value	4.0
Std. Dev	Spread of data	0.7
Count	Number of ratings	5

After generating descriptive statistics, plotting of graphs is an important step, and we will learn this process of lotting graphs in python in subsequent section.

13.7.6 Plotting Graphs in Python (Using Matplotlib)

To visualize data in Python, the most widely used library is Matplotlib, however many other libraries are available.

Follow the steps below to plot a simple graph:

Step 1: Install Matplotlib (if not already installed)

pip install matplotlib

Step 2: Import Required Libraries

import matplotlib.pyplot as plt

Step 3: Prepare Data

x = [1, 2, 3, 4, 5]

y = [10, 20, 15, 30, 25]

Step 4: Plot the Graph

plt.plot(x, y)

Step 5: Add Labels and Title

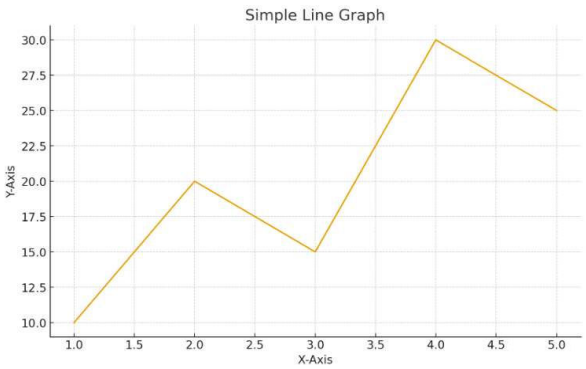
plt.xlabel("X-Axis")

plt.ylabel("Y-Axis")

plt.title("Simple Line Graph")

Step 6: Display the Plot

plt.show()



Let's understand the complete process by executing a Hands-on Example using mtcars dataset: we learned that Descriptive statistics help

highlight key characteristics of a dataset, often using just one value. Creating these statistics is typically the first step after data has been cleaned and prepared for analysis. We've previously encountered examples like the mean and median. In this section, we'll revisit those and introduce some additional statistical tools.

Firstly we begin with the finding of Central tendency of data i.e. Measures of Center, which are statistical methods used to identify the "middle" or most typical value in a numerical dataset. They provide insight into what a standard or expected value might be. The most common measures of central tendency include the mean, median, and mode. We can compute the mean for each column in a DataFrame using:

```
In [1]:
%matplotlib inline

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
In [2]:
mtcars = pd.read_csv("../input/mtcars/mtcars.csv")
mtcars =  mtcars.rename(columns={'Unnamed: 0': 'model'})
mtcars.index = mtcars.model
del mtcars["model"]

mtcars.head()
```

Out[2]:

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
model											
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportaboul	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2

```
In [3]:
mtcars.mean()                    # Get the mean of each column
```

```
Out[3]:
mpg       20.090625
cyl        6.187500
disp     230.721875
hp       146.687500
drat       3.596563
wt         3.217250
qsec      17.848750
vs         0.437500
am         0.406250
gear       3.687500
carb       2.812500
dtype: float64
```

We can also get the means of each row by supplying an axis argument:

```
In [4]:
mtcars.mean(axis=1)          # Get the mean of each row
```

```
Out[4]:
model
Mazda RX4              29.907273
Mazda RX4 Wag          29.981364
Datsun 710             23.598182
Hornet 4 Drive         38.739545
Hornet Sportabout      53.664545
Valiant                35.049091
Duster 360             59.720000
Merc 240D              24.634545
Merc 230               27.233636
Merc 280               31.860000
Merc 280C              31.787273
Merc 450SE             46.430909
Merc 450SL             46.500000
Merc 450SLC            46.350000
Cadillac Fleetwood     66.232727
Lincoln Continental    66.058545
Chrysler Imperial      65.972273
Fiat 128               19.440909
Honda Civic            17.742273
```

The median of a dataset is the value that splits the data into two equal parts—half the values are lower, and half are higher. It represents the center point when the data is sorted. For this reason, it's also known as the 50th percentile. As shown previously, you can compute the median for each column in a DataFrame using:

```
In [5]:
mtcars.median()              # Get the median of each column

Out[5]:
mpg       19.200
cyl        6.000
disp     196.300
hp       123.000
drat       3.695
wt         3.325
qsec      17.710
vs         0.000
am         0.000
gear       4.000
carb       2.000
dtype: float64
```

We can also calculate the median across each row by passing the argument axis=1.Although both the mean and median indicate the central tendency of a dataset, they measure it differently and are not always the same. The median reliably splits the data into two equal halves, while the mean represents the arithmetic average, which can be significantly affected by outliers or extreme values. In a

symmetrical distribution, the mean and median are generally equal. This concept can be better understood by examining a density plot.

```
In [6]:
norm_data = pd.DataFrame(np.random.normal(size=100000))

norm_data.plot(kind="density",
               figsize=(10,10));


plt.vlines(norm_data.mean(),     # Plot black line at mean
           ymin=0,
           ymax=0.4,
           linewidth=5.0);

plt.vlines(norm_data.median(),   # Plot red line at median
           ymin=0,
           ymax=0.4,
           linewidth=2.0,
           color="red");
```

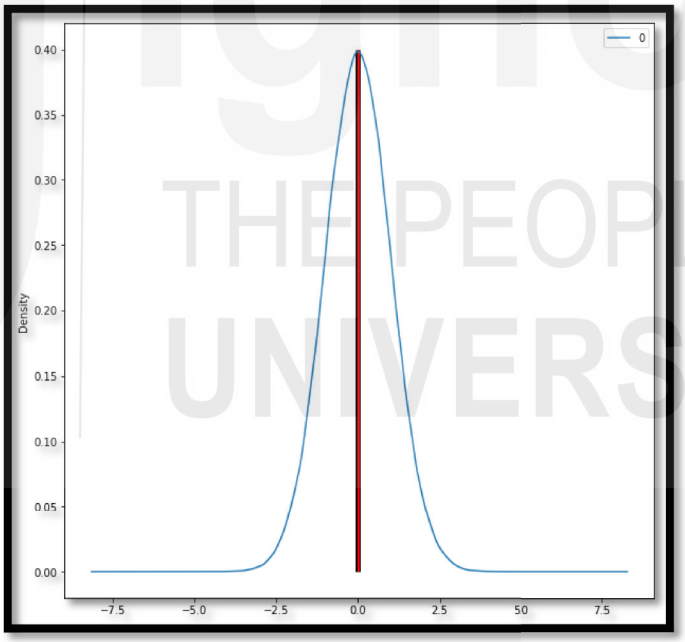


Fig.67: Output to check Normalization in dataset

In the plot shown above, the **mean** and **median** are nearly identical—both are close to zero—so the red line representing the median overlaps the thicker black line that marks the mean.

In skewed distributions, the mean is more affected by the skew and is drawn in the direction of the skewness, while the median is less influenced and tends to stay near the central point of the data.

```
In [7]:
skewed_data = pd.DataFrame(np.random.exponential(size=100000))

skewed_data.plot(kind="density",
                 figsize=(10,10),
                 xlim=(-1,5));


plt.vlines(skewed_data.mean(),     # Plot black line at mean
           ymin=0,
           ymax=0.8,
           linewidth=5.0);

plt.vlines(skewed_data.median(),   # Plot red line at median
           ymin=0,
           ymax=0.8,
           linewidth=2.0,
           color="red");
```

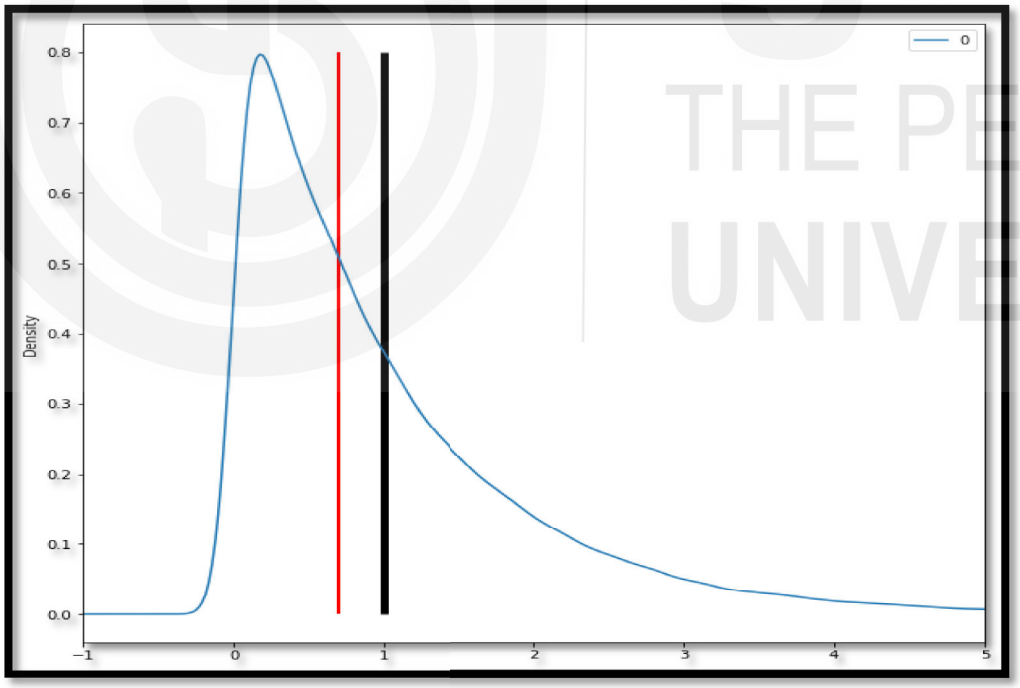


Fig: 68: Mean & Median Output

The mean is greatly influenced by outliers because it includes all values, even the extreme ones, in its calculation. On the other hand, the median is more resistant to outliers and stays fairly consistent, making it a more reliable measure of central tendency when outliers are present.

```
In [8]:
norm_data = np.random.normal(size=50)
outliers = np.random.normal(15, size=3)
combined_data = pd.DataFrame(np.concatenate((norm_data, outliers), axis=0))

combined_data.plot(kind="density",
              figsize=(10,10),
              xlim=(-5,20));


plt.vlines(combined_data.mean(),     # Plot black line at mean
           ymin=0,
           ymax=0.2,
           linewidth=5.0);

plt.vlines(combined_data.median(),   # Plot red line at median
           ymin=0,
           ymax=0.2,
           linewidth=2.0,
           color="red");
```

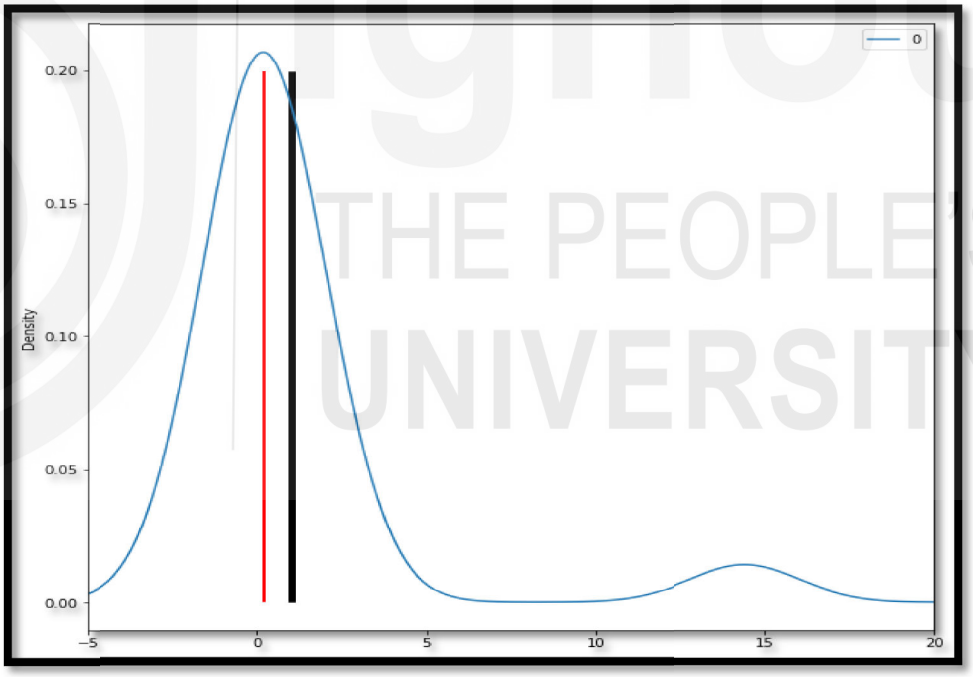


Fig.69: Outliers in dataset

Since the median is less influenced by skewed data and outliers, it is regarded as a "robust" measure. In cases where distributions are noticeably skewed or contain extreme values, the median often gives a more reliable representation of a typical value.

The mode is the value that appears most often in a dataset. Unlike the mean and median, the mode can also be used with categorical data. Additionally, a dataset may have more than one mode if several values share the highest frequency. To identify the mode, you can use:

```
In [9]: mtcars.mode()
```

Out[9]:

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	10.4	8.0	275.8	110.0	3.07	3.44	17.02	0.0	0.0	3.0	2.0
1	15.2	NaN	NaN	175.0	3.92	NaN	18.90	NaN	NaN	NaN	4.0
2	19.2	NaN	NaN	180.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	21.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	21.4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5	22.8	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
6	30.4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

If a column has **multiple modes**—meaning several values occur with the same highest frequency—then all those values are returned as modes. On the other hand, if a column has **no repeating values**, and thus **no mode**, the result will be **NaN**.

Finding the Measures of Spread is also important in data analysis, lets understand how to determine Measures of spread, also known as dispersion, indicate how much the values in a dataset differ from each other. While measures of center highlight a typical or average value, measures of spread show how widely the data points are scattered around that center.

A simple method for assessing spread is the **range**, calculated by subtracting the minimum value from the maximum value in the dataset.

```
In [10]: max(mtcars["mpg"]) - min(mtcars["mpg"])

Out[10]: 23.5
```

As noted earlier, the median represents the 50th percentile of a dataset. To gain a clearer picture of a variable's spread, it's helpful to examine several percentiles. These include the minimum (0th percentile), first quartile (25th percentile), median (50th percentile), third quartile (75th percentile), and maximum (100th percentile). You can extract these values using the quantile () function in pandas.

```
In [11]:
five_num = [mtcars["mpg"].quantile(0),
            mtcars["mpg"].quantile(0.25),
            mtcars["mpg"].quantile(0.50),
            mtcars["mpg"].quantile(0.75),
            mtcars["mpg"].quantile(1)]

five_num

Out[11]:
[10.4, 15.425, 19.2, 22.8, 33.9]
```

Because these percentile values are frequently used to summarize a dataset, they are collectively known as the **five-number summary"** consists of the minimum, 25th percentile (Q1), median (Q2), 75th percentile (Q3), and maximum. These key statistics offer a quick overview of the data's distribution and are also generated by the df.describe() function in pandas.

```
In [12]:
mtcars["mpg"].describe()

Out[12]:
count    32.000000
mean     20.090625
std       6.026948
min      10.400000
25%      15.425000
50%      19.200000
75%      22.800000
max      33.900000
Name: mpg, dtype: float64
```

The **Interquartile Range (IQR)** is a commonly used measure of variability. It shows the spread of the middle 50% of the data and is determined by subtracting the first quartile (Q1) from the third quartile (Q3):

IQR = Q3 - Q1.

```
In [13]:
mtcars["mpg"].quantile(0.75) - mtcars["mpg"].quantile(0.25)

Out[13]:
7.375
```

The boxplots shown in fig 57 representing the summary and the interquartile range (IQR) offers a clear view of the data's distribution, central tendency, spread, and possible outliers.

```
In [14]: mtcars.boxplot(column="mpg",
                        return_type='axes',
                        figsize=(8,8))

         plt.text(x=0.74, y=22.25, s="3rd Quartile")
         plt.text(x=0.8, y=18.75, s="Median")
         plt.text(x=0.75, y=15.5, s="1st Quartile")
         plt.text(x=0.9, y=10, s="Min")
         plt.text(x=0.9, y=33.5, s="Max")
         plt.text(x=0.7, y=19.5, s="IQR", rotation=90, size=25);
```

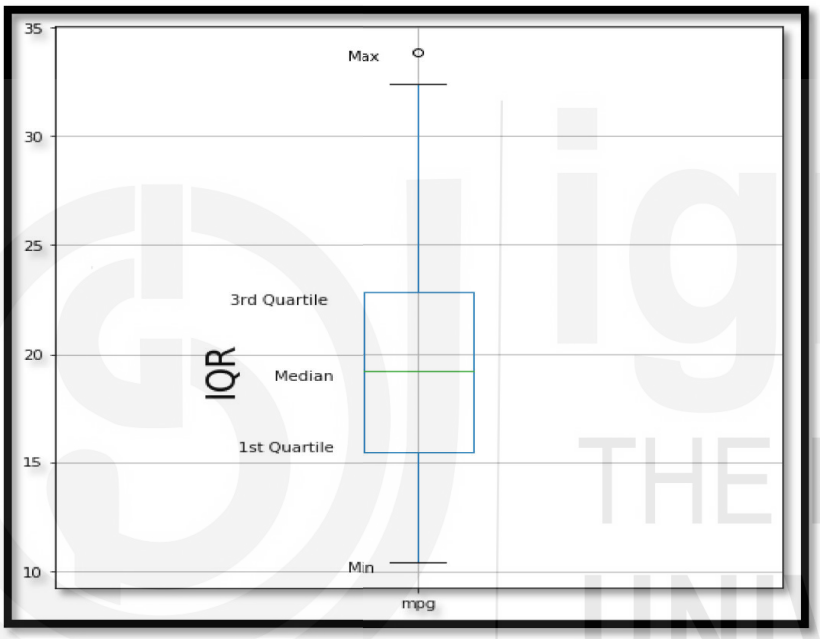


Fig.70: Boxplot of IQR

Two other commonly used measures of spread are **variance** and **standard deviation**.

- Variance is the average of the squared differences between each data point and the mean. It provides insight into how much the values in a dataset deviate from the average.

You can calculate the variance of each column in a DataFrame using:

```
In [15]: mtcars["mpg"].var()

Out[15]: 36.32410282258065
```

- **Standard deviation** is the square root of the variance. It's generally more intuitive to interpret since it's in the same units as the original data, unlike variance, which is expressed in squared units. To calculate the standard deviation for each column in a DataFrame, use:

```
In [16]: mtcars["mpg"].std()

Out[16]: 6.026948052089105
```

Since variance and standard deviation rely on the mean, they can be heavily influenced by skewed data and outliers. A more robust alternative is the Median Absolute Deviation (MAD). MAD is computed as the median of the absolute differences from the median, making it a dependable measure of spread for datasets with non-normal distributions or extreme values.

```
In [17]: abs_median_devs = abs(mtcars["mpg"] - mtcars["mpg"].median())

         abs_median_devs.median() * 1.4826

Out[17]: 5.411490000000001
```

Note: The **Median Absolute Deviation (MAD)** is frequently multiplied by a **scaling factor of 1.4826**. This adjustment makes the MAD comparable to the **standard deviation** when the data follows a normal distribution, allowing for better consistency between the two measures.

Skewness and Kurtosis : In addition to measures of center and spread, descriptive statistics also include metrics that describe the **shape** of a distribution i.e. skewness and kurtosis. A brief introduction to both is given below:

- **Skewness** indicates the **asymmetry** of the distribution—whether the data is skewed to the left or right.
- **Kurtosis** reflects how much of the data lies in the **tails** versus the **center** of the distribution.
-

While we won't dive into the exact formulas, these metrics build on the concept of variance:

- **Skewness** uses **cubed deviations** from the mean.
- **Kurtosis** uses **fourth power deviations** from the mean.

```
In [18]: mtcars["mpg"].skew()  # Check skewness

Out[18]: 0.6723771376290805

In [19]: mtcars["mpg"].kurt()  # Check kurtosis

Out[19]: -0.0220062914240855
```

To better understand **skewness** and **kurtosis**, let's generate some **sample (dummy) data** and examine its distribution using these two measures:

```
In [20]: norm_data = np.random.normal(size=100000)
         skewed_data = np.concatenate((np.random.normal(size=35000)+2,
                                       np.random.exponential(size=65000)),
                                       axis=0)
         uniform_data = np.random.uniform(0,2, size=100000)
         peaked_data = np.concatenate((np.random.exponential(size=50000),
                                       np.random.exponential(size=50000)*(-1)),
                                       axis=0)

         data_df = pd.DataFrame({"norm":norm_data,
                                 "skewed":skewed_data,
                                 "uniform":uniform_data,
                                 "peaked":peaked_data})
```

```
In [21]: data_df.plot(kind="density",
                      figsize=(10,10),
                      xlim=(-5,5));
```

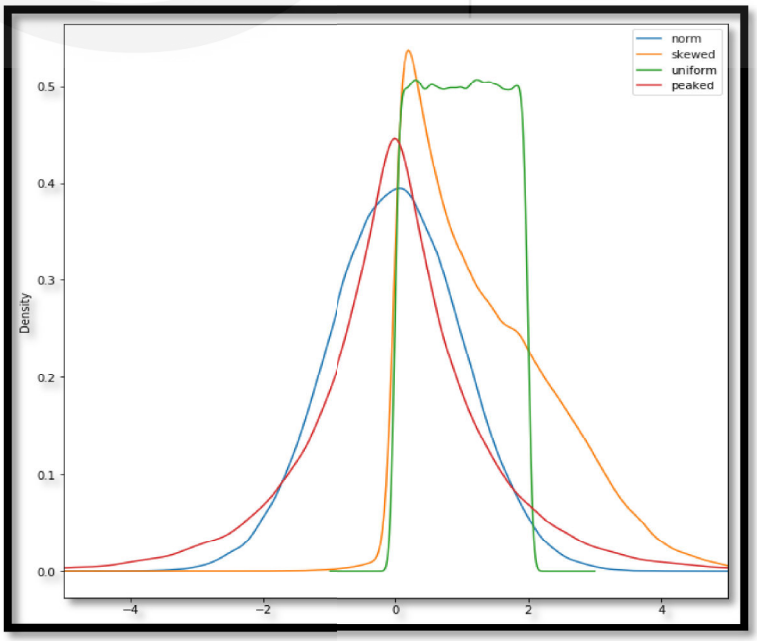


Fig.71: Skewness and Kurtosis

Next, let's evaluate the **skewness** of each dataset. Since skewness reflects the **asymmetry** of a distribution, we expect it to be **low** for most of the distributions that are relatively **symmetric**, and **higher** only for the one that is intentionally **skewed**.

```
In [22]: data_df.skew()

Out[22]:
norm      -0.003337
skewed     1.004511
uniform   -0.001172
peaked     0.001861
dtype: float64
```

Now let's check kurtosis.

```
In [23]: data_df.kurt()

Out[23]:
norm      -0.009062
skewed     1.297040
uniform   -1.200426
peaked     2.952304
dtype: float64
```

As observed from the results, the **normal distribution** shows **kurtosis close to zero**, indicating a typical bell-shaped curve. The **uniform (flat) distribution** has **negative kurtosis**, meaning it has **lighter tails** and a flatter peak. In contrast, the other two distributions—with more data in the **tails than in the center**—exhibit **higher kurtosis**, reflecting heavier tails and more extreme values.

So we learned that Descriptive statistics provide numerical summaries that help you understand key aspects of your data, such as its **center, spread, and shape**. These statistics not only guide the direction of your analysis but also make it easier to communicate findings clearly and efficiently. Moreover, metrics like **mean** and **variance** are foundational in many statistical tests and predictive models.

SUM UP - User Interface and Data Handling in Python (Anaconda/Google Colab)

1. Introduction to Anaconda & Colab

- **Anaconda Navigator**: Launch Jupyter Notebook, Spyder.

- **Jupyter Notebook Interface**: Cell-based execution, Markdown support.
- **Google Colab**: Cloud-based, supports GPU, file uploads.

2. ***Uploading the Dataset***

- In Jupyter:

```
pd.read_csv("file.csv")
```

- In Google Colab, import and use the file upload function:

```
from google.colab import files
uploaded = files.upload()
```

- Then, read the file using Pandas:

```
pd.read_csv("file.csv")
```

3. ***Handling Data***

- Use Pandas: df.head(), df.info(), df.describe()
- Clean data: drop missing, rename columns, filter rows.
- Transformation: df['new'] = df['col1'] + df['col2']

5. ***Creating Visualizations Use libraries like Matplotlib and Seaborn to plot different types of graphs:***

```
import matplotlib.pyplot as plt
import seaborn as sns

plt.bar(x, y)                          # Bar chart
sns.histplot(df['column'])             # Histogram
sns.scatterplot(x='x', y='y', data=df)  # Scatter plot
```

- Use Matplotlib and Seaborn for various graphs.

☞ **Check Your Progress - 3**

1. What function in pandas shows basic statistics of a dataset?

   ……

   ……

2. How can you read a CSV file in pandas?

   ……

   ……

   ……

3. What library in Python is used for plotting graphs?

   ……

   ……

4. How do you calculate mean and median in pandas?

   ……

   ……

5. How can you identify outliers using visualization?

   ……

   ……

13.8 R-STUDIO

RStudio is a powerful, user-friendly IDE for R programming. It's the preferred environment for most R users. It offers Clean layout for data science tasks, and quite easy access to documentation and plots with integrated version control (Git). However, Other Interfaces (Optional) are **Jupyter Notebook with IRKernel** – Interactive R notebooks, **VS Code with R extension, StatET Plugin for Eclipse, and Jamovi also.**

Here, we will work on R-Studio only, the user interface of the same is shown below:

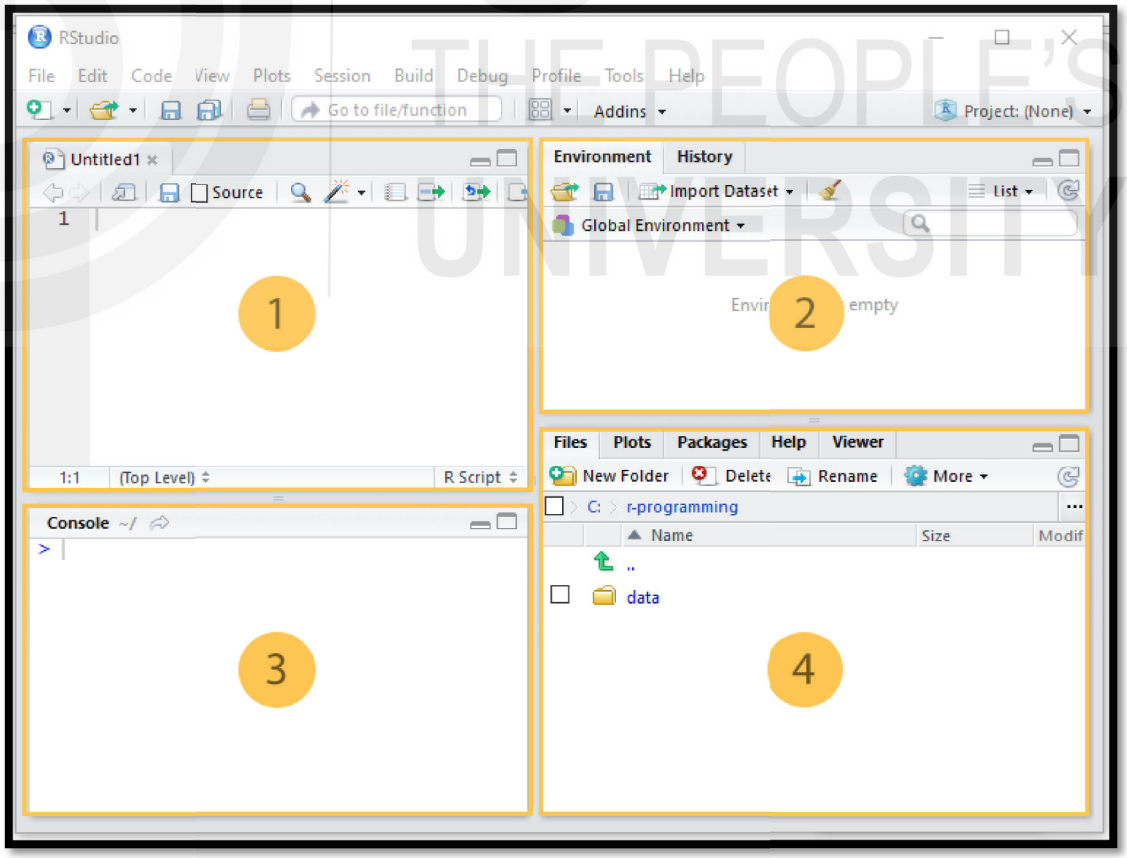


Fig.72: R-Studio: Interface Layout (4 Panes):

The description of the 4-panes shown in the above figure, is as follows:

1. **Source Editor (Top-Left)**

- Write and edit R scripts (.R), RMarkdown (.Rmd), and notebooks.
- Supports syntax highlighting, code folding, and auto-completion.

Source Editor This is where you **write and edit scripts**, RMarkdown files, and notebooks. You can run code from here directly into the Console (Pane 3).Useful for saving and reusing code.

2. **Environment / History Pane**

- **Environment Tab**: Shows all the **objects, variables, and datasets** you've created.
- **History Tab**: Keeps a **record of previously executed commands**.
- Helps manage workspace variables and track command history.

3. Console **(Bottom-Left)**

- Run R commands interactively, and it Displays output, errors, and messages.

4. **Files/Plots/Packages/Help/Viewer (Bottom-Right)**

- **Files**: Navigate your project directory.
- **Plots**: Display visualizations created in R.
- **Packages**: Manage installed libraries.
- **Help**: Access documentation for functions.

Now, after understanding the utility of all 4-panes of the user interface of R-Studio, let's extend our discussion to learn How to use RStudio's GUI for Importing the Dataset. The steps to be performed are as follows:

Steps:

1. Open **RStudio**.
2. In the **Environment pane (top-right)**, click **"Import Dataset"**.
3. Choose from:
   - **From Text (readr)** or **From Excel**, or **From CSV**
4. Browse and select your file (e.g., data.csv).
5. A preview appears—check the options like separator, header, etc.
6. Click **"Import"**.
7. The dataset is loaded into your environment (usually as a data frame).

Steps for Alternate Method : Using Code (R Console or Script)

1. To Import a CSV File:

```
data <- read.csv("path/to/your/file.csv")
```

2. Example (Windows path):

```
data <- read.csv("C:/Users/YourName/Documents/data.csv")
```

3. To Import Excel File:

```
install.packages("readxl")
library(readxl)
data <- read_excel("path/to/your/file.xlsx")
```

Common Import Functions

Function	Use	Package
read.csv()	Load CSV files	Base R
read.table()	Load general delimited text files	Base R
read_excel()	Load Excel .xlsx files	readxl
read_csv()	Faster CSV loading	readr
fread()	Fast and efficient CSV loader	data.table

Table 4: Functions of R

Note : Use getwd() to check the current working directory**, and setwd("path") to change it:**

```
getwd()
setwd("C:/Users/YourName/Documents")
```

After understanding the basic working structure of R-Studio, let's extend our discussion a next level, i.e. how to perform Descriptive analysis in R? the same is addressed in next section.

13.8.2 Descriptive Analysis in R Programming

Descriptive statistics in R involve summarizing and organizing data using various tools such as **charts, graphs, tables, and spreadsheets**. This process helps in presenting the data in a **clear and understandable format**, allowing for meaningful insights. Descriptive analysis aims to **summarize the main features** of a dataset. It is typically performed on **smaller datasets**, providing a foundational understanding that can also guide **future trend predictions** based on existing data patterns.

Descriptive statistics often rely on two major types of measures i.e. Measures of Central Tendency and Measures of Variability (Dispersion), the two are discussed below:

1. Measures of Central Tendency: Indicates the center or typical value of the dataset and include:

- **Mean (Average):** The arithmetic average of all observations. (R-Syntax:

mean(data vector))

- **Median:** The middle value when the data is arranged in ascending order. (R-syntax: median(data vector))
- **Mode:** The most frequently occurring value. R does not have a built-in mode function, so it can be computed as: (R-Syntax: mode value <- names(sort(table(data vector), decreasing = TRUE))

2. ***Measures of Variability (Dispersion):*** Reflects spread out of the data and it includes:

- **Range :** Difference between the highest and lowest values.
- *R-Syntax : range(data vector)*
- *diff(range(data vector))*
- **Interquartile Range (IQR) :** Measures the spread of the middle 50% of the data.
- *R-Syntax : IQR(data vector)*
- **Variance:** Average of the squared deviations from the mean.
- *R-Syntax : var(data vector)*
- **Standard Deviation :** Square root of the variance; shows spread in the same units as the data.
- *R-Syntax : sd(data vector)*

Example:

```
# Sample dataset
data vector <- c(12, 15, 14, 19, 20, 15, 18, 22, 15)
# Central Tendency
mean(data vector)
median(data vector)
mode value <- names(sort(table(data vector), decreasing = TRUE))
# Variability
range(data vector)
IQR(data vector)
var(data vector)
sd(data vector)
```

This code produces output that clearly shows the descriptive statistics,

Given: data vector <- c(12, 15, 14, 19, 20, 15, 18, 22, 15)

Measure	Output
Mean	16.67
Median	15
Mode	15
Range	12 to 22
Range (Difference)	10
IQR	4

Variance	11.25
Standard Deviation	3.354102

Now, lets revisit the Steps for Performing Descriptive Statistics in R

1. **Load your dataset** using functions like read.csv(), read_excel(), or data() for built-in sets.
2. **Summarize the data**:
   - Use summary(data) for a quick overview
   - Use mean(), median(), sd(), var(), range() for individual measures
3. **Visualize the data**:
   - Create graphs such as histograms (hist()), boxplots (boxplot()), and bar charts (barplot())
4. **Interpret results** to understand trends, central points, and variability in your data.

To upload a dataset in R, you can use several methods depending on your file type (e.g., CSV, Excel) and your preference (command line, GUI, etc.). The commonly used method are as follows:

1. ***Uploading a CSV File*** *of dataset can be done by using the read.csv() function:*

```
# Basic CSV file loading
data <- read.csv("path/to/your/file.csv")
```

- Replace "path/to/your/file.csv" with your actual file path.
- If using RStudio, you can drag and drop the file into the Files pane, then use the path shown.

2. ***Uploading an Excel File :*** Use the readxl or openxlsx package:

```
# First, install the package if not already done
install.packages("readxl")

# Load the package
library(readxl)

# Read Excel file (first sheet by default)
data <- read_excel("path/to/your/file.xlsx")
```

3. **Uploading dataset by Using RStudio GUI (Import Dataset Button)**

1. Click on **Environment tab** > **Import Dataset**.
2. Choose:
   - **From Text (readr)** – for CSV/TSV

- **From Excel** – for Excel files

3. Browse and select your file.
4. RStudio auto-generates the code and previews the data.

4. Uploading from the Web

```
data <- read.csv("https://example.com/yourfile.csv")
```

Till the moment we learned how to handle a dataset, now its time to extend our discussion for data visualization and data interpretation. To understand these features we are going to refer to the renowned dataset of mtcars, a built in dataset in R. you can explore the dataset by applying commands, shown in the respective screenshots, given below:

```
In [2]: df <- mtcars #built-in dataset in R

In [3]: head(df) #check the first 6 values
```

A data.frame: 6 × 11

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
Mazda RX4	21.0	6	160	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160	110	3.90	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108	93	3.85	2.320	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360	175	3.15	3.440	17.02	0	0	3	2
Valiant	18.1	6	225	105	2.76	3.460	20.22	1	0	3	1

```
In [4]: tail(df) #check the last 6 values
```

A data.frame: 6 × 11

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
Porsche 914-2	26.0	4	120.3	91	4.43	2.140	16.7	0	1	5	2
Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.9	1	1	5	2
Ford Pantera L	15.8	8	351.0	264	4.22	3.170	14.5	0	1	5	4
Ferrari Dino	19.7	6	145.0	175	3.62	2.770	15.5	0	1	5	6
Maserati Bora	15.0	8	301.0	335	3.54	3.570	14.6	0	1	5	8
Volvo 142E	21.4	4	121.0	109	4.11	2.780	18.6	1	1	4	2

```
In [5]: dim(df) #check the dimensions of the data, 32 rows and 11 variables

32 · 11
```

```
In [6]: str(df) #check the structure of the data

'data.frame':   32 obs. of  11 variables:
 $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
 $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
 $ disp: num  160 160 108 258 360 ...
 $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
 $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
 $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
 $ qsec: num  16.5 17 18.6 19.4 17 ...
 $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
 $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
 $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
 $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

```
In [7]: #The Details of the dataset can be found by using:
        ?mtcars
```

For the purpose of Data Visualization and interpretation we are going to use the data visualization library ggplot() over the built-in dataset "mtcars".

```
In [8]: library(ggplot2)
```

Here's how you can systematically *perform data visualization and data interpretation, using R*

Task-1: *What is the Relation between MPG (Mileage) and Number of Cylinders ?*

We know that MPG, or *miles per gallon*, indicates how far a car can travel using one gallon of fuel. It's a standard measure of **fuel efficiency**—higher MPG means better fuel economy and less fuel consumption per mile. So, by plotting the graph between *MPG (Mileage) and Number of Cylinders, using command shown below:*

```
In [9]: ggplot(df, aes(x= factor(cyl), y=mpg)) + geom_point()
```

We got following output as graph or visualization, showing the relationship between MPG (Mileage) and Number of Cylinders

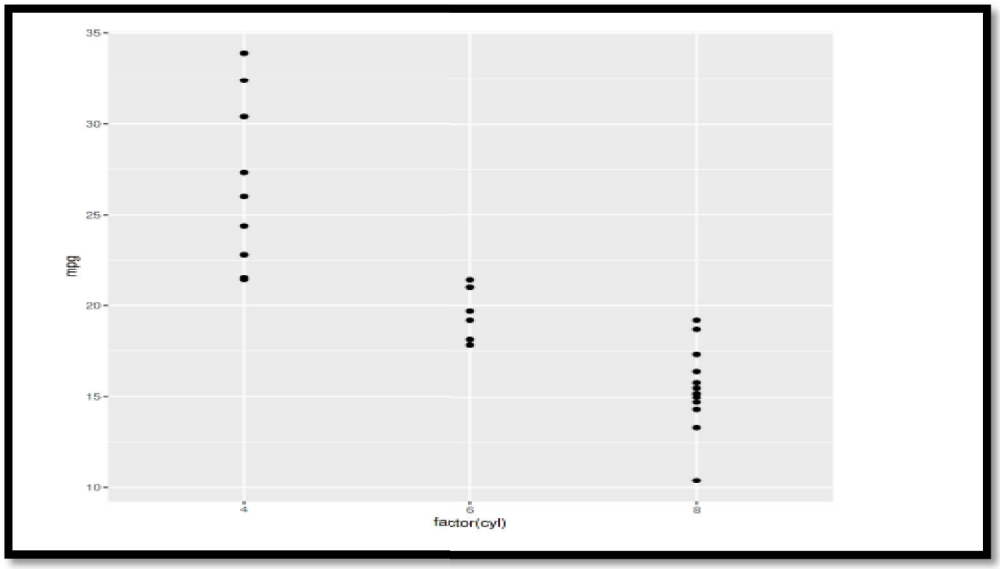


Fig.73: Data Interpretation

Interpretation: The scatter plot shows that as the number of cylinders increases, the mileage (mpg) decreases.

Task-2: *Explore the relationship between mileage (mpg) and engine displacement*: Again, a graph is plotted between the two i.e. between mileage (mpg) and engine displacement, and the same is shown below:

```
In [10]: ggplot(df, aes(x=mpg, y= disp)) +geom_point()
```

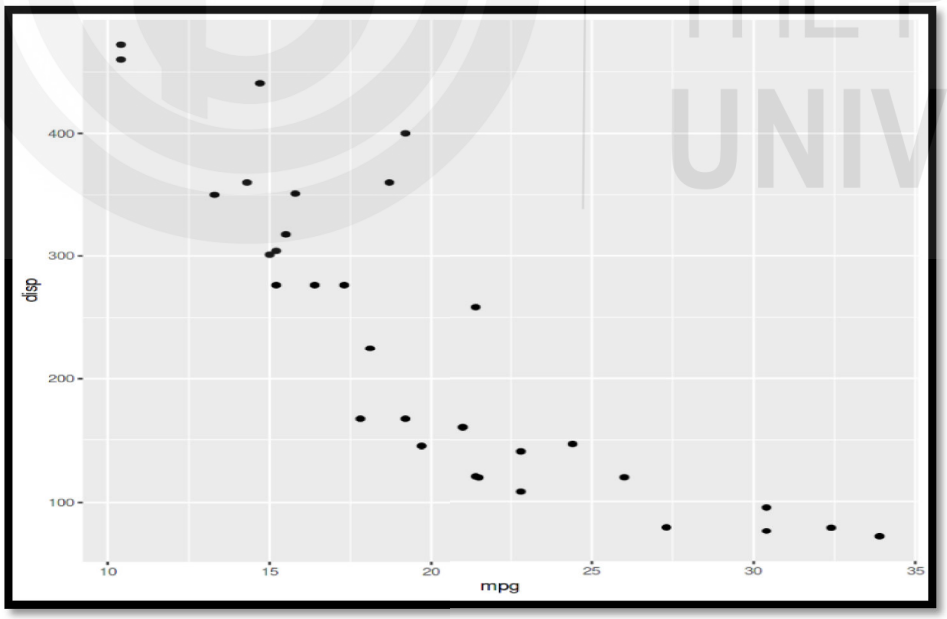


Fig.74: Scatter plot

Interpretation - Greater engine displacement generally results in lower fuel efficiency.

NOTE – (Understanding Displacement): Engine displacement refers to the total volume that all cylinders in an engine can move. A larger displacement allows the

engine to intake more air and fuel, which increases its potential to generate power. However, the actual power output also depends on other internal components and the engine's design. Displacement is usually measured in liters. For example, many modern vehicles have a 2.0-liter four-cylinder engine, meaning each cylinder has a capacity of about 0.5 liters or 500 cc.

Task-3: *How does cylinder effects mpg and displacement?* Again, a graph is plotted between the variables and the same is shown below:

```
In [11]: ggplot(df, aes(x=mpg, y= disp, color =factor(cyl))) + geom_point()
```

```
In [12]: ggplot(df, aes(x=mpg, y= disp, color =factor(cyl))) + geom_point(aes(shape= factor(cyl)))
```

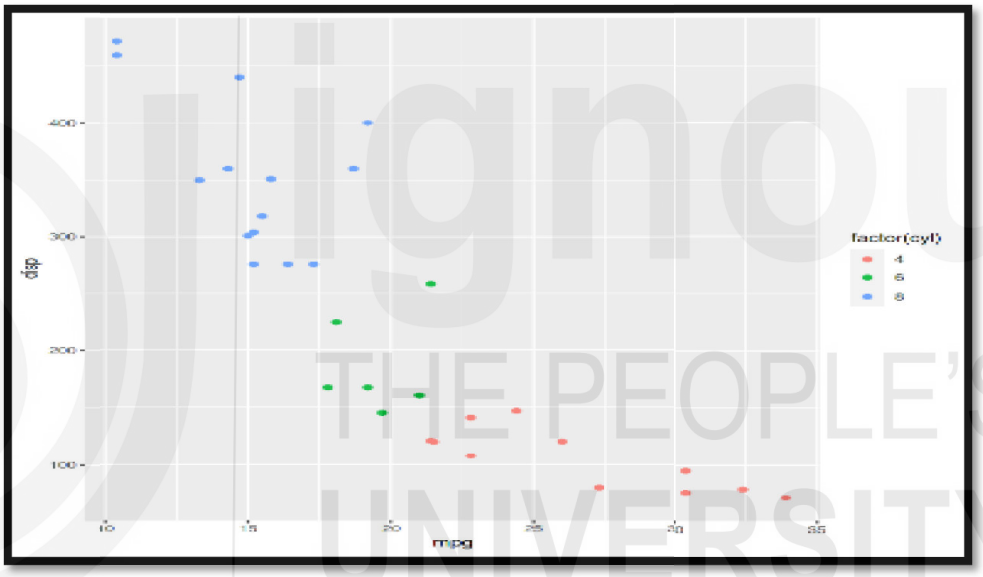


Fig.75: Colored scatter plot

Interpretation - Cars with fewer cylinders tend to have higher mileage and better fuel efficiency.

On the other hand, vehicles with more cylinders usually have larger engine displacement, meaning the engine consumes more air and fuel. As a result, they are generally less fuel-efficient.

Task-4: *Understanding the Relation between mpg, disp and hp, to interpret this a graph is plotted between the variables using the command, shown below:*

```
In [13]: ggplot(df, aes(x=mpg, y= disp, size=hp )) + geom_point()
```

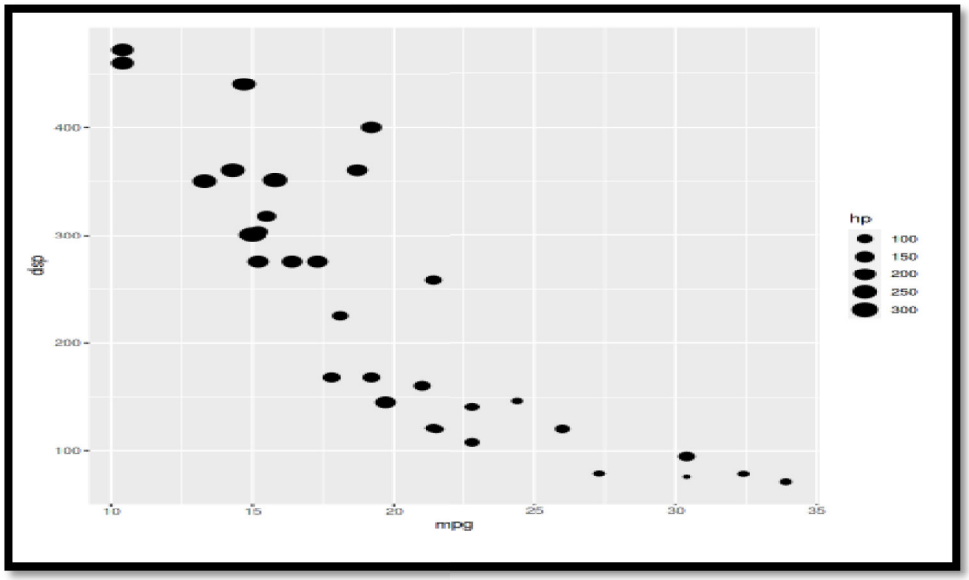


Fig.76: Interpretation of horse power

Interpretation - As horsepower increases, engine displacement also tends to increase. This indicates a more powerful engine, which typically results in lower fuel efficiency — meaning the vehicle delivers less mileage.

Task-5: *Understanding the Relation between hp, disp, mpg, and number of cylinders, to interpret this a graph is plotted between the variables using the command, shown below:*

```
In [14]: ggplot(df, aes(x=mpg, y= disp, size=hp, color=factor(cyl)) ) + geom_point()
```

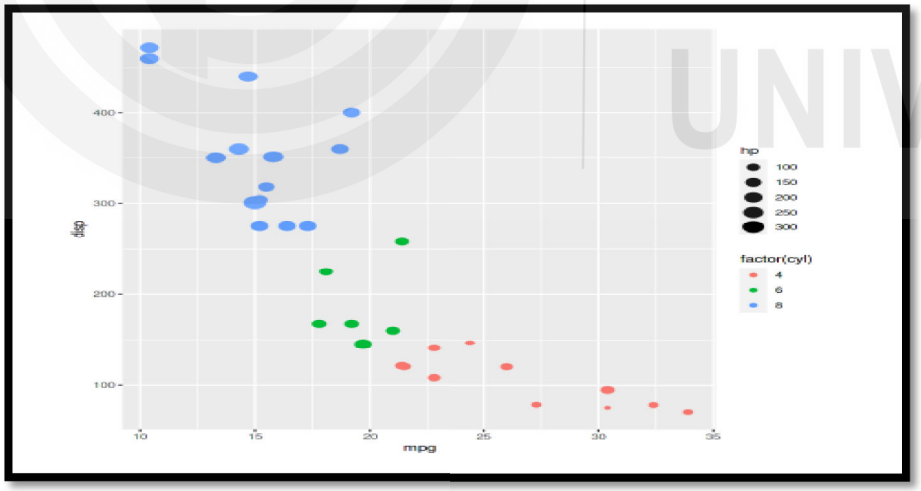


Fig.77: Interpretation of horse power

Interpretation - As horsepower (hp) increases, there is a corresponding increase in engine displacement and usually the number of cylinders as well. This combination points to a more powerful engine. However, such engines tend to be less fuel-efficient, which means cars with higher hp, larger displacement, and more cylinders generally offer lower mileage (mpg)**.**

Addition of shape and color makes it easier to interpret the plot and perform better analysis. This helps in understanding how various features like **displacement**,

horsepower, and **mileage** interact based on engine configuration.

The **Scatter plot** can be generated by using the following syntax in R

```
In [15]: ggplot(df, aes(x=mpg, y= disp, size=hp, color=factor(cyl)) )+ geom_point(aes(shape=factor(cyl)))+
labs(title='Motor Cars Scatter Plot', subtitle='showing Mpg,hp,disp,& cyl relation',
     caption='In-Built Dataset', tag='Chart1')
```

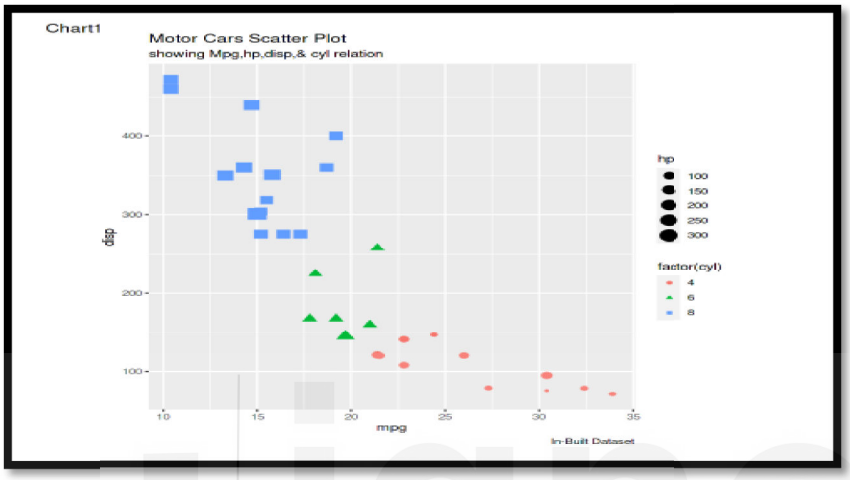


Fig.78: Interpretation of horse power

<u>Generating Bar Charts</u>: Bar charts are useful for summarizing and comparing **categorical variables** or **aggregated values** (like average mileage per cylinder type).

```
In [16]: #simple Bar Chart showing count of cylinders
         ggplot(df, aes(x=factor(cyl)))+geom_bar()
```

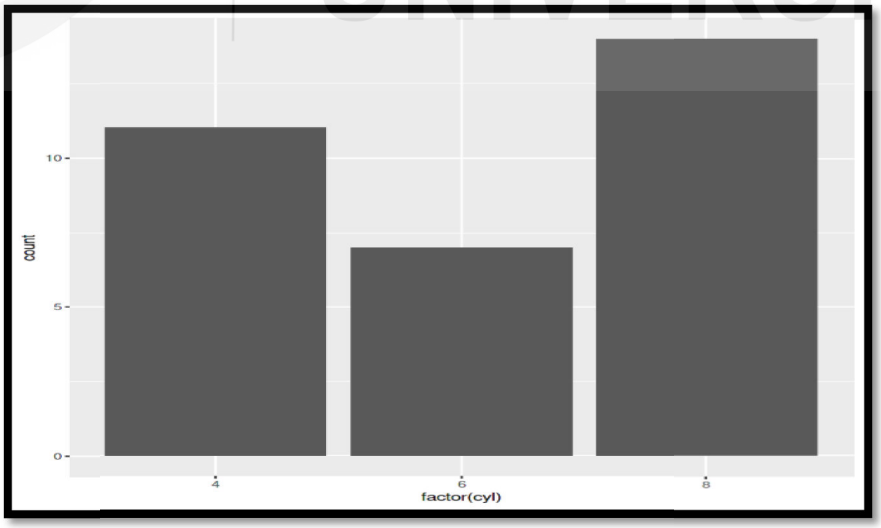


Fig.79: Bar graph

In [17]:
```
#using color parameter
ggplot(df, aes(x=factor(cyl), fill=factor(cyl)))+geom_bar()
```

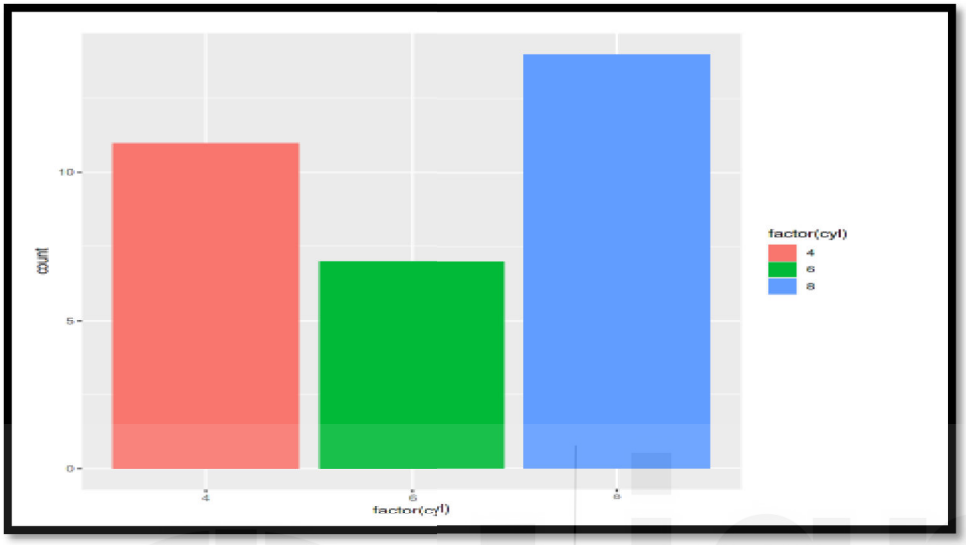


Fig.80: Colored Bar graph

In [18]:
```
#add title and subtitle
ggplot(df, aes(x=factor(cyl), fill=factor(cyl)))+geom_bar()+
labs(title = 'Motor Cars Cyl & Count', subtitle='Bar Plot in R')
```

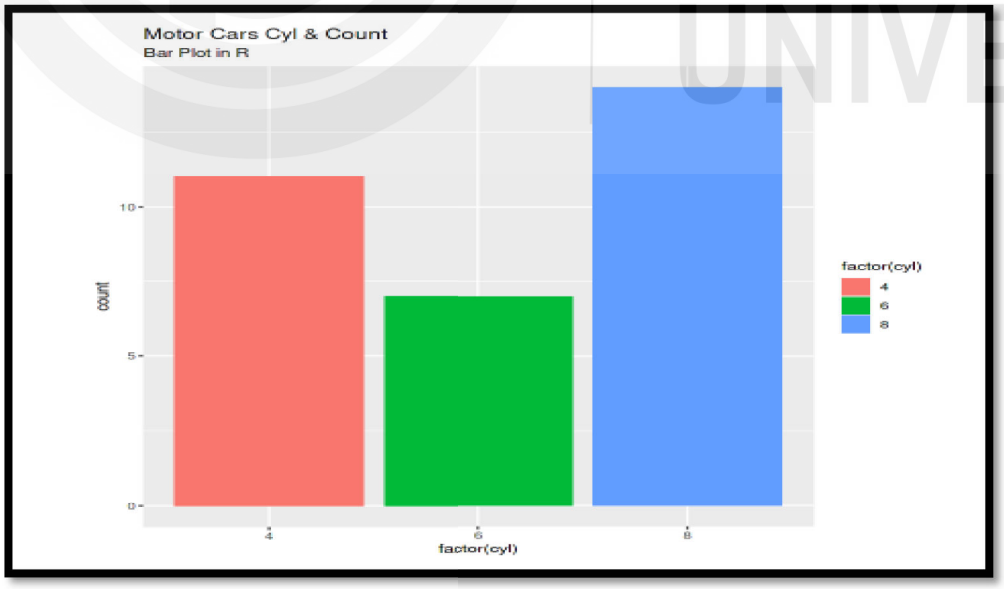


Fig.81: Colored Bar graph (Motor Cars Cyl & Count)

In [19]:
```
#using 2 variables
ggplot(df, aes(x=factor(cyl)))+ geom_bar(aes(fill=factor(vs)))
```

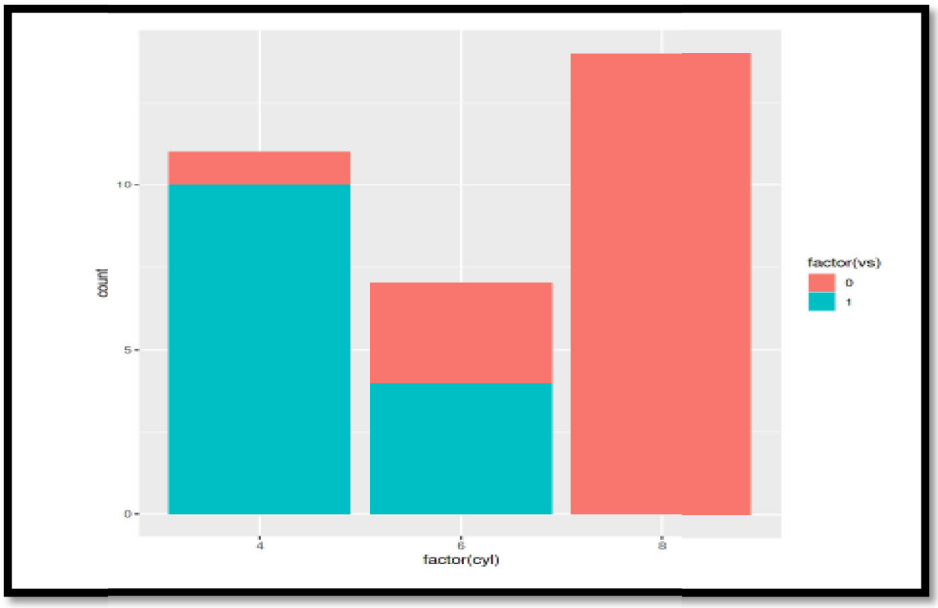


Fig.82 : Factor vs. Count Bar graph

- **From the Analysis of the Bar charts produced above, following can be interpreted:**

1. Most 8-cylinder engines are V-shaped.
2. 4-cylinder engines are mainly straight (inline).
3. 6-cylinder engines have a nearly equal mix of V and straight configurations.

```
In [20]: ggplot(df, aes(x=factor(cyl), fill=factor(carb)))+geom_bar()
```

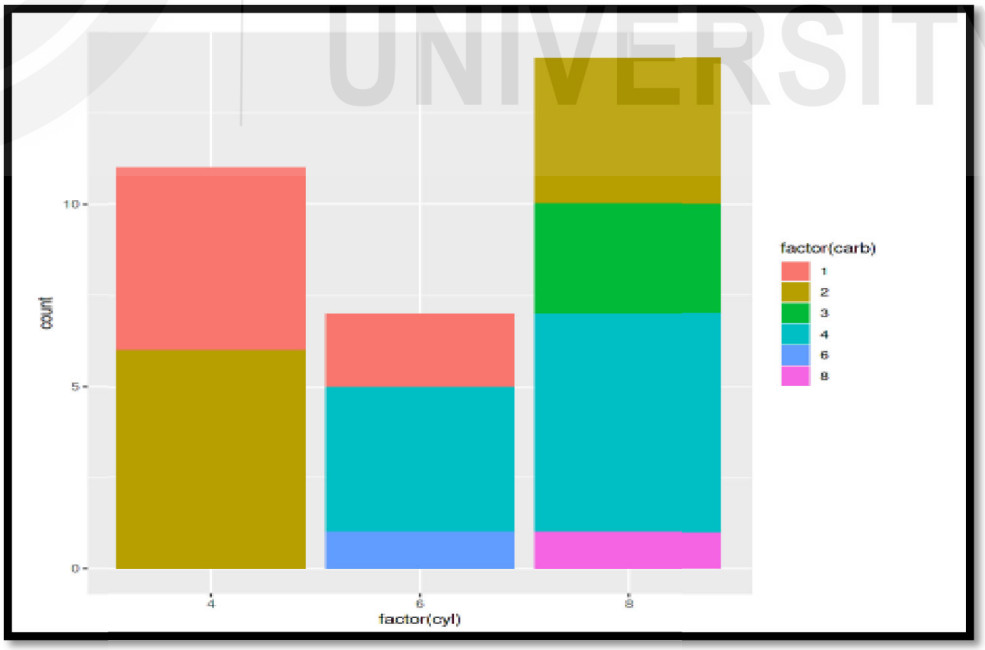


Fig.83: Factor vs. Count Bar graph

- **From the Analysis of the Bar charts produced above, following can be interpreted**:

1) No of Cylinders are 4; carb used are 1 and 2
2) No of Cylinders are 6; carb used are 1,4, and 6
3) No of Cylinders are 8; carb used are 2,3,4, and 8

➢ ***SUM UP - User Interface and Data Handling in R (RStudio)***

1. *Introduction to RStudio Interface*

- **Source Pane**: Script writing.
- **Console Pane**: Code output.
- **Environment/History Pane**: Variables and past commands.
- **Files/Plots/Packages Pane**: File management, visual output.

2. *Uploading the Dataset : df <- read.csv("filename.csv")*

- Use "Import Dataset" GUI in RStudio.

3. *Handling Data*

- Use functions like summary(df), str(df), head(df)
- Clean using na.omit(), subset(), mutate() (from dplyr).

4. *Use Base R plotting functions:*

```
plot(df$var1, df$var2)     # Creates a scatter plot
hist(df$var1)              # Draws a histogram of var1
```

For advanced and customizable visuals, use the **ggplot2** package:

```
library(ggplot2)
ggplot(df, aes(x = var1, y = var2)) + geom_point()   # Scatter plot using ggplot2
```

- Use base R or ggplot2 for advanced visuals.

☞ **Check Your Progress - 4**

1. How do you import a CSV file in R?

   ……

   ……

   ……

2. What is the function of summary() in R?

   ……

   ……

   ……

3. Which R function is used to generate a histogram?

   ……

   ……

   ……

4. How can you check the structure of a dataset in R?

   …….

   …….

   …….

5. How does cylinder count affect MPG in mtcars dataset?

   …….

   …….

   …….

This structured approach introduces students to the basic interfaces and functionalities of Tableau, Power BI, Python (Anaconda/Colab), and R (RStudio) while ensuring they gain hands-on exposure to uploading datasets, preparing data, and generating useful visual insights.

13.9 SUMMARY

This unit-13 titled *"Tools for Data Science – An Introduction"* emphasizes that in today's digital age, data is a critical resource, but its value depends on how effectively it is processed and interpreted. Data science, combining mathematics, statistics, computing, and domain expertise, transforms raw data into actionable knowledge. To do this, professionals rely on powerful tools that enable collection, cleaning, analysis, and visualization of data. This unit introduces four essential tools—**Tableau, Power BI, Python, and R**—which together form the foundation of modern data science practice.

Tableau is presented as an intuitive visualization platform requiring little or no coding. It allows learners to connect datasets from various sources, create interactive dashboards, and generate descriptive statistics. Through drag-and-drop features and built-in analytics, Tableau helps communicate insights to both technical and non-technical audiences. Power BI, developed by Microsoft, is introduced as a robust business intelligence tool, closely integrated with Excel and cloud services. Learners are guided through steps of connecting to data, cleaning it using Power Query, building relationships, applying calculations with DAX, and creating dashboards that can be shared via the Power BI service.

The unit then shifts to **Python**, a versatile programming language widely used in data science for its rich libraries such as Pandas, NumPy, Matplotlib, and Seaborn. Learners are introduced to different interfaces like IDLE, Jupyter Notebook, and Google Colab, with hands-on steps for uploading datasets, performing descriptive statistics (mean, median, mode, standard deviation, skewness, kurtosis), and visualizing data through plots. Practical emphasis is laid on using Python for data preparation and exploratory analysis.

Finally, **R (via RStudio)** is introduced as a statistical powerhouse favored in research environments. The unit explains the RStudio interface and dataset import methods, followed by performing descriptive analysis using summary statistics, measures of central tendency, dispersion, and visualization techniques such as histograms, scatter plots, and boxplots. Learners are encouraged to explore

relationships between variables using examples like the *mtcars* dataset.

Overall, the unit equips learners with a comparative understanding of four foundational tools in data science. Tableau and Power BI provide user-friendly, no-code solutions for visualization and business intelligence, while Python and R offer greater flexibility and analytical depth through coding. By the end of the unit, learners gain the ability to import, clean, analyze, and visualize datasets, thereby building confidence to apply these tools in academic, research, or professional contexts.

13.10 SOLUTIONS/ANSWERS

☞ Check Your Progress - 1

1. What are the key components of the Tableau workspace?

   Answer: The key components include Data Pane, Shelves (Rows, Columns), Cards, Toolbar, Worksheet Tabs, and the View Area.

2. How do you connect to a Microsoft Excel file in Tableau?

   Answer: Go to the 'Connect' section, select 'To a File', and then choose 'Microsoft Excel' to upload the dataset.

3. What is the purpose of the 'Show Me' panel in Tableau?

   Answer: It suggests the most suitable visualizations based on selected data fields.

4. How can you calculate the median in Tableau?

   Answer: You can use the median aggregation function or drag the Median option from the Analytics pane.

5. How are outliers identified using Tableau?

   Answer: Outliers are identified using distribution bands in visualizations like box plots or line charts.

☞ Check Your Progress-2

1. What is Power Query used for?

   Answer: Power Query is used to clean, transform, and shape data before analysis.

2. How can you create a new measure in Power BI?

   Answer: Use the 'New Measure' option under the 'Table Tools' tab and define logic using DAX.

3. What are calculated columns in Power BI?

   Answer: Calculated columns are fields created with DAX that become part of the data model and update with refreshes.

4. How do you upload a dataset in Power BI?

   Answer: Click on 'Get Data' under Home tab, select the source (e.g., Excel), and load the file.

5. How do you publish a report in Power BI?

Answer: Use the 'Publish' button to upload your report to the Power BI Service for sharing.

☞ Check Your Progress - 3

1. What function in pandas shows basic statistics of a dataset?

   Answer: `df.describe()` provides count, mean, std deviation, min, max, and quartiles.

2. How can you read a CSV file in pandas?

   Answer: Use `pd.read_csv('filename.csv')` to read the file into a DataFrame.

3. What library in Python is used for plotting graphs?

   Answer: Matplotlib and Seaborn are widely used for data visualization.

4. How do you calculate mean and median in pandas?

   Answer: Use `df.mean()` for mean and `df.median()` for median.

5. How can you identify outliers using visualization?

   Answer: Use boxplots from Seaborn or matplotlib to highlight outliers.

☞ Check Your Progress - 4

1. How do you import a CSV file in R?

   Answer: Use `read.csv('filename.csv')` or use the Import Dataset GUI in RStudio.

2. What is the function of summary() in R?

   Answer: `summary(data)` gives min, max, mean, median, and quartile information.

3. Which R function is used to generate a histogram?

   Answer: `hist()` is used to create histograms in R.

4. How can you check the structure of a dataset in R?

   Answer: Use `str(data)` to see the data types and structure.

5. How does cylinder count affect MPG in mtcars dataset?

   Answer: More cylinders typically result in lower MPG due to higher fuel consumption.

13.11 FURTHER READINGS

- **Official Tableau Learning Resources** https://www.tableau.com/learn/training
- **Tableau Your Data!** – Daniel G. Murray A comprehensive book covering Tableau techniques and real-world applications.
- **Visual Analytics with Tableau** – Alexander Loth Focuses on building dashboards and interpreting data effectively.
- **Microsoft Power BI Documentation** https://learn.microsoft.com/en-us/power-bi/
- **The Definitive Guide to DAX** – Marco Russo & Alberto Ferrari, Excellent for learning data modeling and DAX functions in Power BI.
- **Power BI for the Excel Analyst** – Wyn Hopkins, Helpful for Excel users transitioning to Power BI.

- **Python Official Documentation,** https://docs.python.org/3/
- **Automate the Boring Stuff with Python** – Al Sweigart (free to read online), Great beginner book with hands-on examples.
- **Python for Data Analysis** – Wes McKinney, Focuses on NumPy, pandas, and data wrangling in Python.
- **Google's Python Class,** https://developers.google.com/edu/python
- **R for Data Science** – Hadley Wickham & Garrett Grolemund, https://r4ds.had.co.nz/ A great introductory book for learning R in the context of data analysis.
- **The Art of R Programming** – Norman Matloff, Ideal for students who want to understand R at a deeper programming level.
- **Hands-On Programming with R** – Garrett Grolemund, Interactive and beginner-friendly for learning R syntax and logic.
- **CRAN R Manuals and Tutorials,** https://cran.r-project.org/manuals.html
- Tableau Software. (2024). *Tableau Training and Tutorials*. Retrieved from https://www.tableau.com/learn/training
- Loth, A. (2019). *Visual Analytics with Tableau*. Wiley.
- Murray, D. G. (2016). *Tableau Your Data! Fast and Easy Visual Analysis with Tableau Software*. Wiley**.**
- Microsoft Corporation. (2024). *Power BI Documentation*. Retrieved from https://learn.microsoft.com/en-us/power-bi/
- Russo, M., & Ferrari, A. (2020). *The Definitive Guide to DAX: Business Intelligence for Microsoft Power BI, SQL Server Analysis Services, and Excel*. Microsoft Press.
- Hopkins, W. (2021). *Power BI for the Excel Analyst: A Step-by-Step Guide to Effective Data Analysis*. Apress.
- Python Software Foundation. (2024). *Python 3 Documentation*. Retrieved from https://docs.python.org/3/
- Sweigart, A. (2019). *Automate the Boring Stuff with Python*. No Starch Press.
- McKinney, W. (2022). *Python for Data Analysis: Data Wrangling with pandas, NumPy, and Jupyter*. O'Reilly Media.
- Google Developers. (2024). *Google's Python Class*. Retrieved from https://developers.google.com/edu/python
- Wickham, H., & Grolemund, G. (2016). *R for Data Science*. O'Reilly Media. Retrieved from https://r4ds.had.co.nz/
- Matloff, N. (2011). *The Art of R Programming*. No Starch Press.
- Grolemund, G. (2014). *Hands-On Programming with R*. O'Reilly Media.
- R Core Team. (2024). *R: The R Project for Statistical Computing*. Retrieved from https://cran.r-project.org/