ASSIGNMENT 1

TITLE: Hadoop Installation on Single Node

OBJECTIVES:

1. To Learn and understand the concepts of Hadoop
2. To learn and understand the Hadoop framework for Big Data
3. To understand and practice installation and configuration of Hadoop.

SOFTWARE REQUIREMENTS:

1 Windows 10/11
2 Virtual Machine
3 Cloudera Virtual Machine

THEORY:

Introduction

Hadoop is an open-source framework that allows to store and process big data in a distributed environment across cluster so computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage.

Due to the advent of new technologies, devices, and communication like social networking sites, the amount of data produced by mankind is growing rapidly every year. The amount of data produced by us from the beginning of time till 2003 was 5 billion gigabytes. The same amount was created in every two days in 2011, and in every ten minutes in2013.This rate is still growing enormously. Though all this information produced is meaningful and can be useful when processed, it is being neglected

BigData

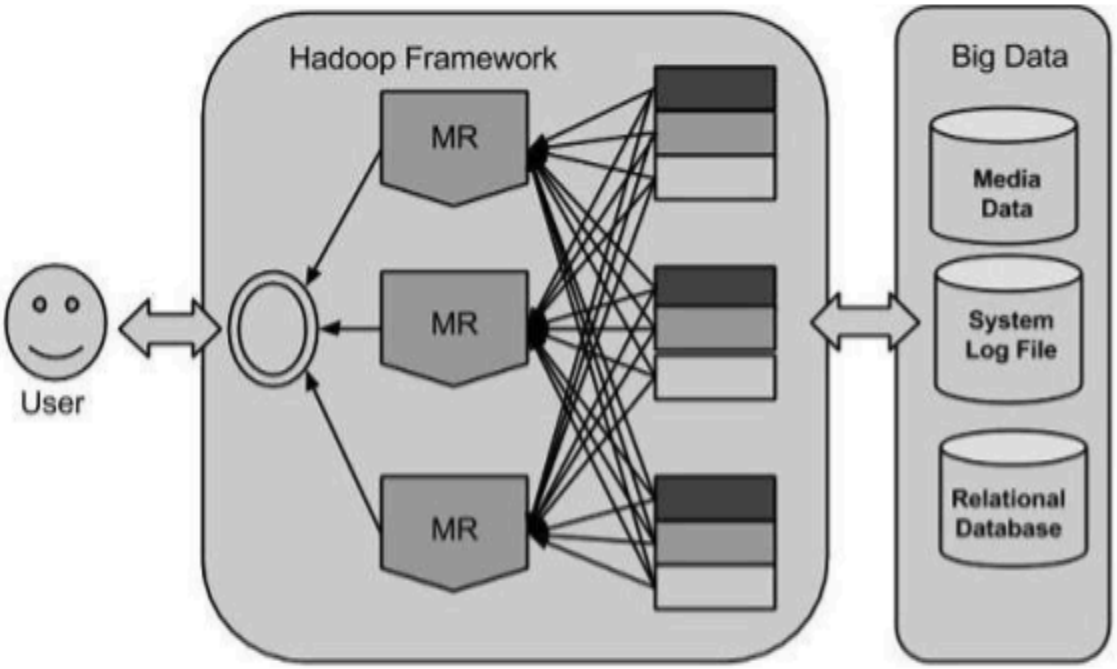
Big data means really a big data, it is a collection of large datasets that cannot be processed using traditional computing techniques. Bigdata is not merely a data, rather it has become a complete subject, which involves various tools, techniques and frameworks. Bigdata involves the data produced by different devices and applications. Given be low are some of the fields that come under the umbrella of BigData.

Hadoop

Doug Cutting, Mike Cafarella and team took the solution provided by Google and start an Open Source Project called HADOOP in 2005 and Doug named it after his son's toy elephant.

Now Apache Hadoop is a registered trademark of the Apache Software Foundation.

Hadoop runs applications using the MapReduce algorithm, where the data is processed in parallel on different CPU nodes. In short, Hadoop framework is capable enough to develop applications capable of running on clusters of computers and they could perform complete statistical analysis for huge amounts of data.



Hadoop is an Apache open source framework written in java that allows distributed processing of large datasets across clusters of computers using simple programming models. A Hadoop frame-worked application works in an environment that provides distributed storage and computation across clusters of computers. Hadoop is designed to scale up from single server to thousands of machines, each offering local computation and storage.

Hadoop Architecture

Hadoop framework includes following four modules:

- **Hadoop Common:** These are Java libraries and utilities required by other Hadoop modules. These libraries provides file system and OS level abstractions and contains the necessary Java files and scripts required to start Hadoop.
- **Hadoop YARN:** This is a framework for job scheduling and cluster resource management.
- **Hadoop Distributed File System (HDFS™):** A distributed file system that provides high-throughput access to application data.

- **Hadoop MapReduce:** This is YARN-based system for parallel processing of large datasets.

Learning Goals

In this activity, you will:

- Download and Install VirtualBox.
- Download and Install Cloudera Virtual Machine (VM) Image.
- Launch the Cloudera VM.

Hardware Requirements: (A) Quad Core Processor (VT-x or AMD-V support recommended), 64-bit;
(B) 8 GB RAM; (C) 20 GB disk free. How to find your hardware information: Open System by clicking the Start button, right-clicking Computer, and then clicking Properties. Most computers with 8 GB RAM purchased in the last 3 years will meet the minimum requirements. You will need a high-speed internet connection because you will be downloading files up to 4 Gb in size.

Instructions

Please use the following instructions to download and install the Cloudera Quickstart VM with VirutalBox before proceeding to the Getting Started with the Cloudera VM Environment video. The screenshots are from a Mac but the instructions should be the same for Windows. Please see the discussion boards if you have any issues.

1. Install VirtualBox. Go to https://www.virtualbox.org/wiki/Downloads to download and install VirtualBox for your computer. The course uses Virtualbox 5.1.X, so we recommend clicking VirtualBox 5.1 builds on that page and downloading the older package for ease of following instructions and screenshots. However, it shouldn't be too different if you choose to use or upgrade to VirtualBox 5.2.X. For Windows, select the link "VirtualBox 5.1.X for Windows hosts x86/amd64" where 'X' is the latest version.
2. For Ubuntu Select "VirtualBox 5.1.X for Windows hosts x86/amd64" where 'X' is the latest version.

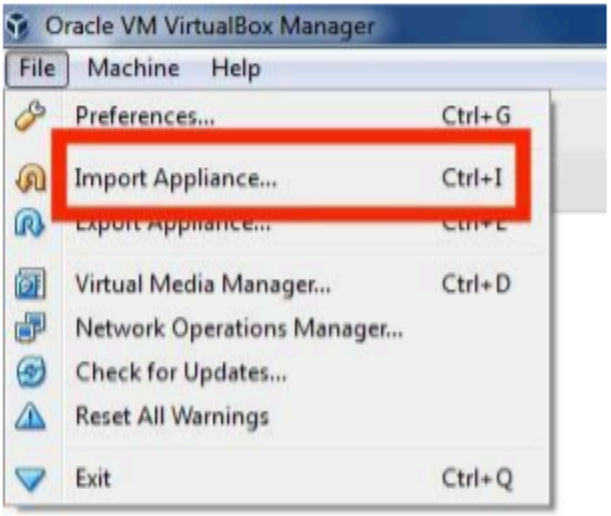
2. **Download the Cloudera VM**. Download the Cloudera VM fromhttps://downloads.cloudera.com/demo_vm/virtualbox/cloudera-quickstart-vm - 5.4.2-0- virtualbox.zip. The VM is over 4GB, so will take some time to download.

3. **Unzip the Cloudera VM**:

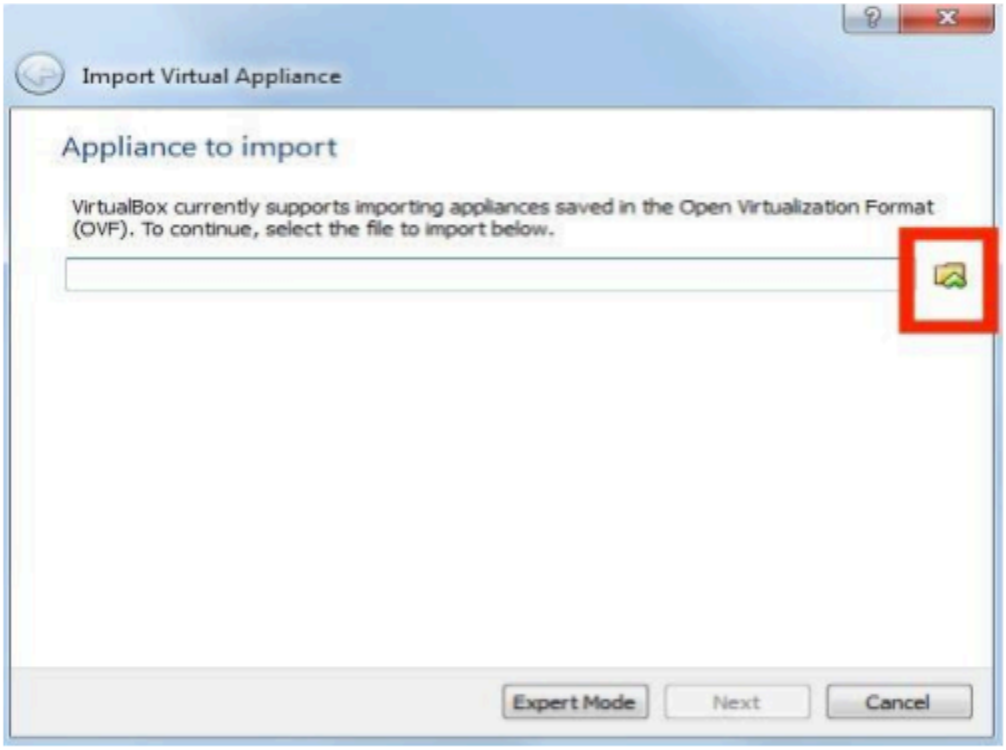
Right-click cloudera-quickstart-vm-5.4.2-0-virtualbox.zip and select “Extract All…”

4. **Start VirtualBox**.

5. **Begin importing**. Import the VM by going to File -> Import Appliance

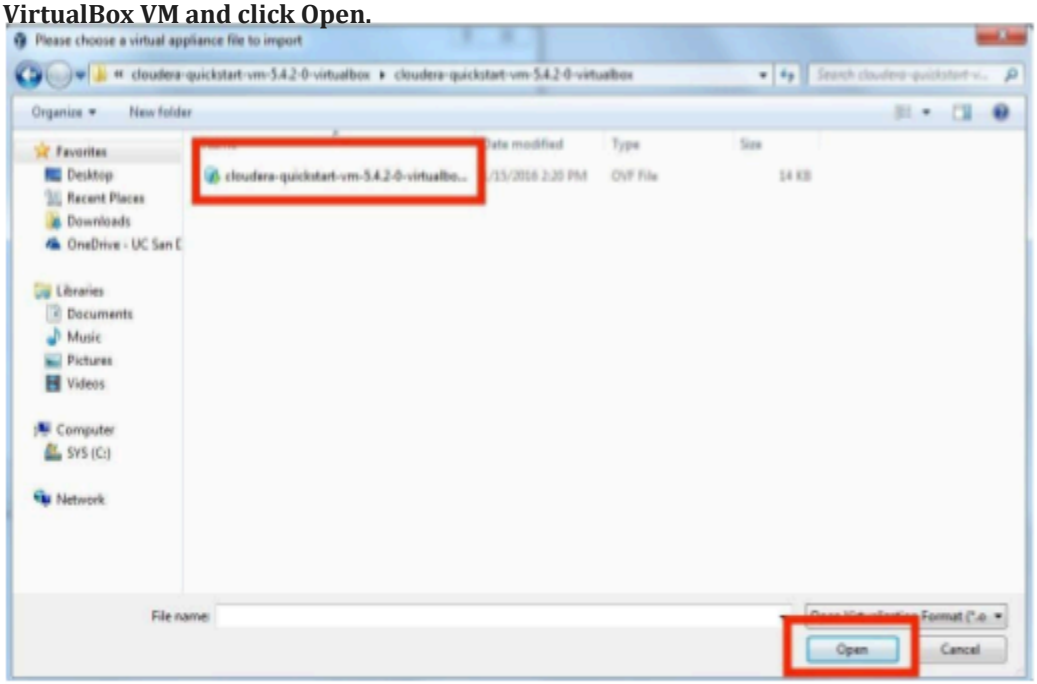


6. Click the Folder icon.

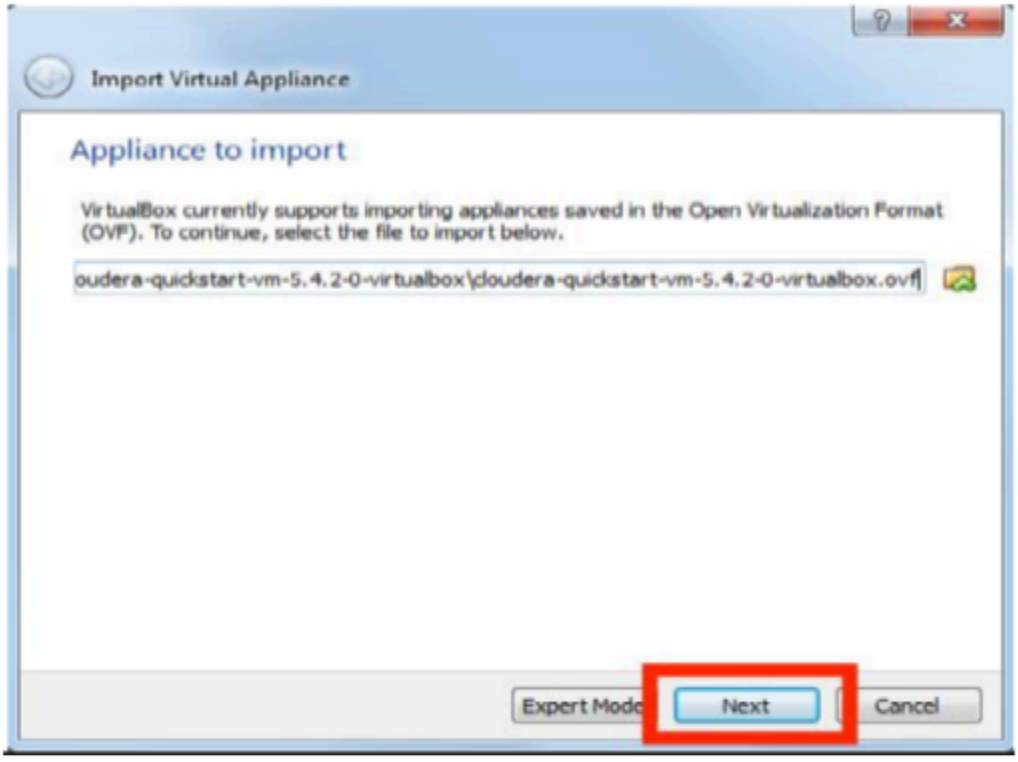


7. Select the cloudera-quickstart-vm-5.4.2-0-virtualbox.ovf from the Folder where you unzipped the

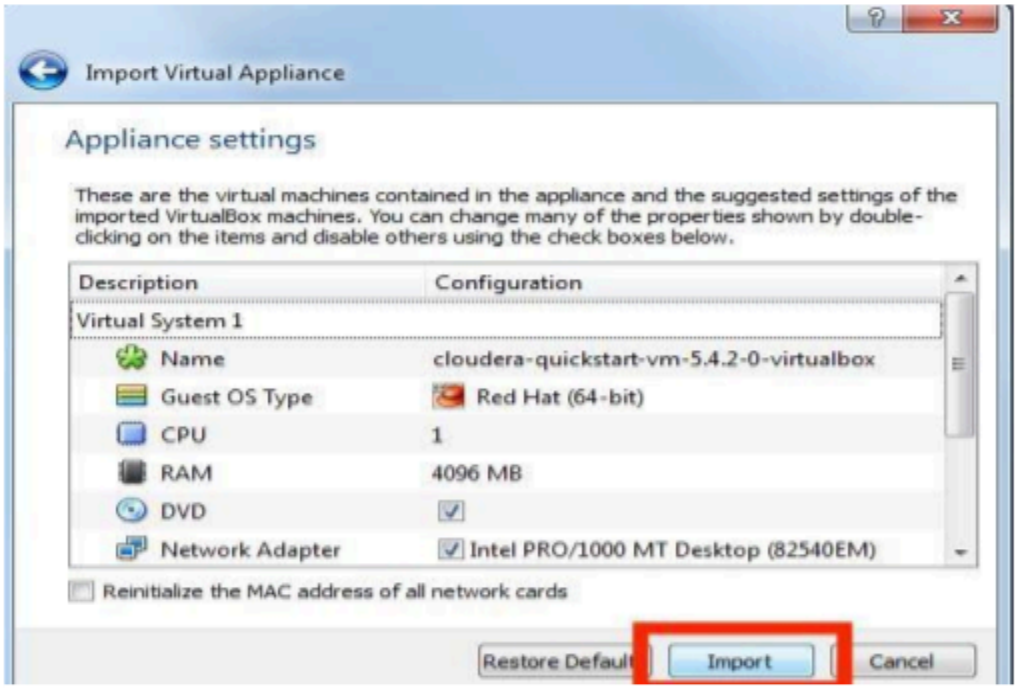
VirtualBox VM and click Open.



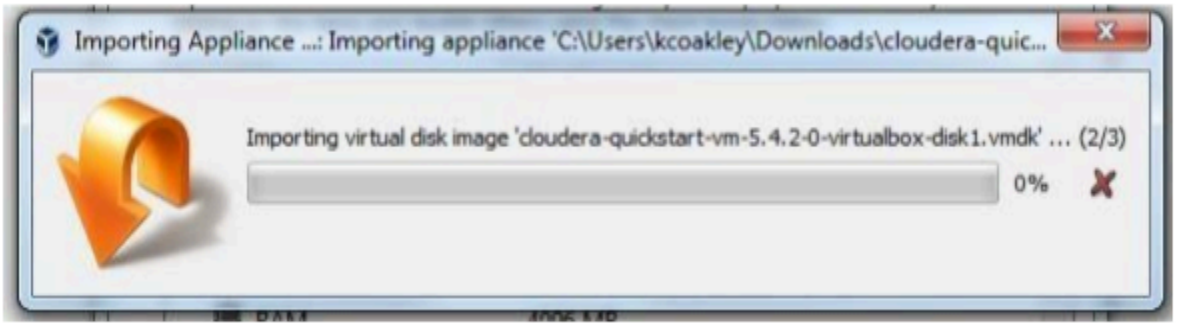
8. **Click Next** to proceed.



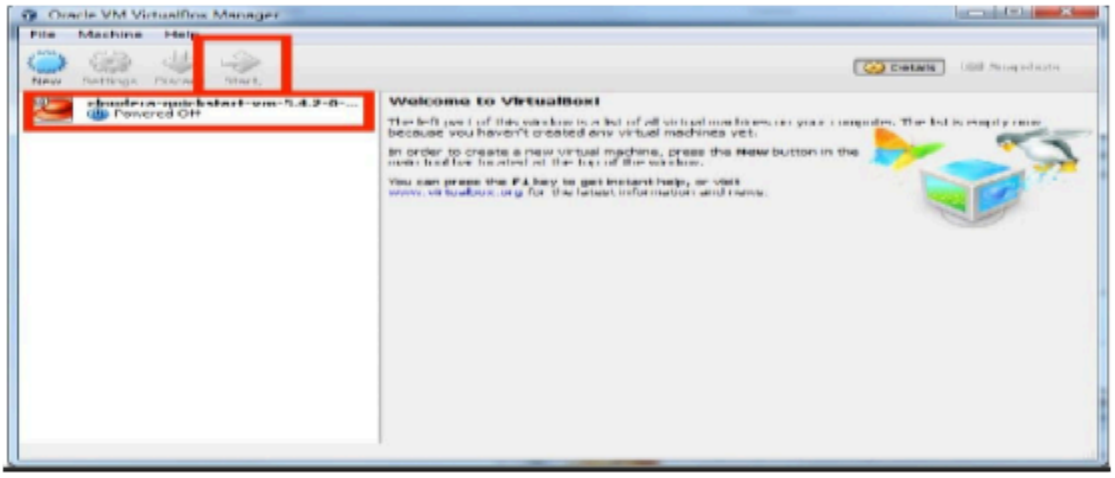
9. **Click Import.**



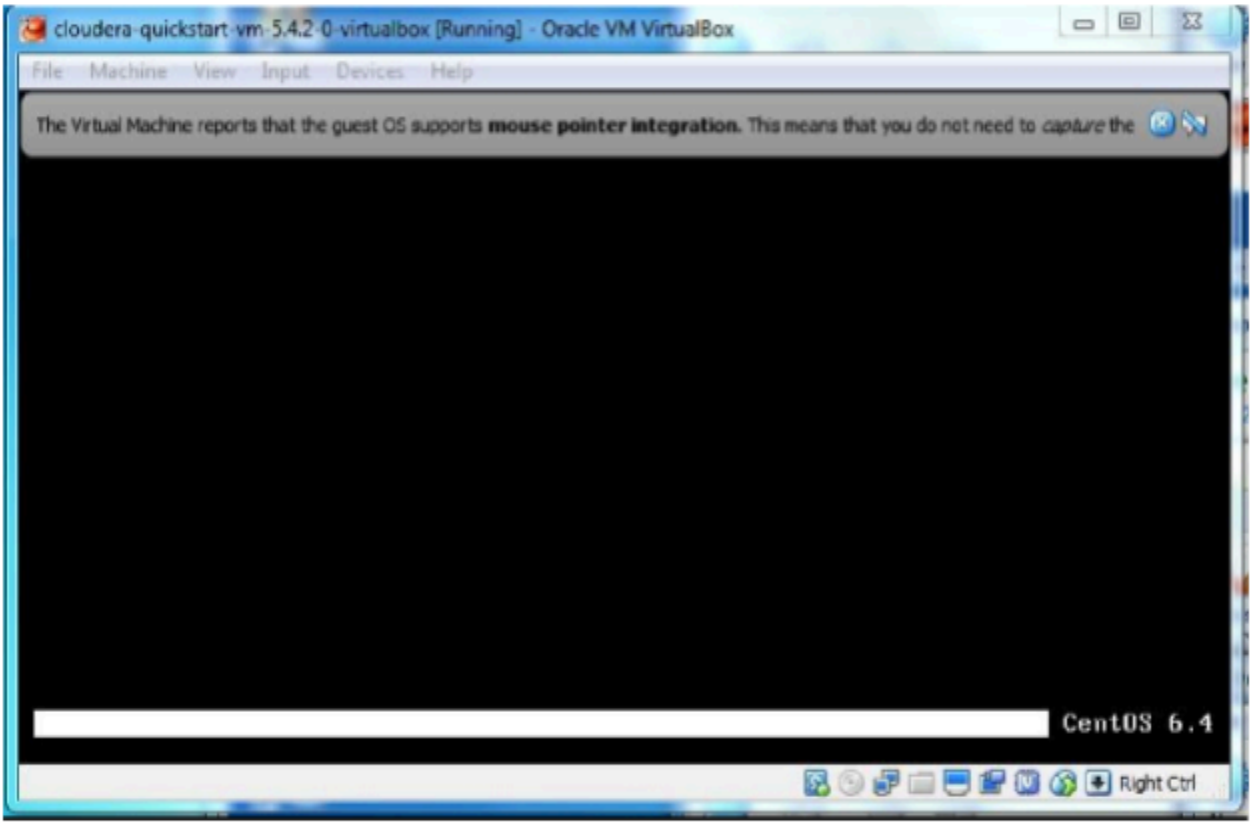
10. **The virtual machine image will be imported.** This can take several minutes.



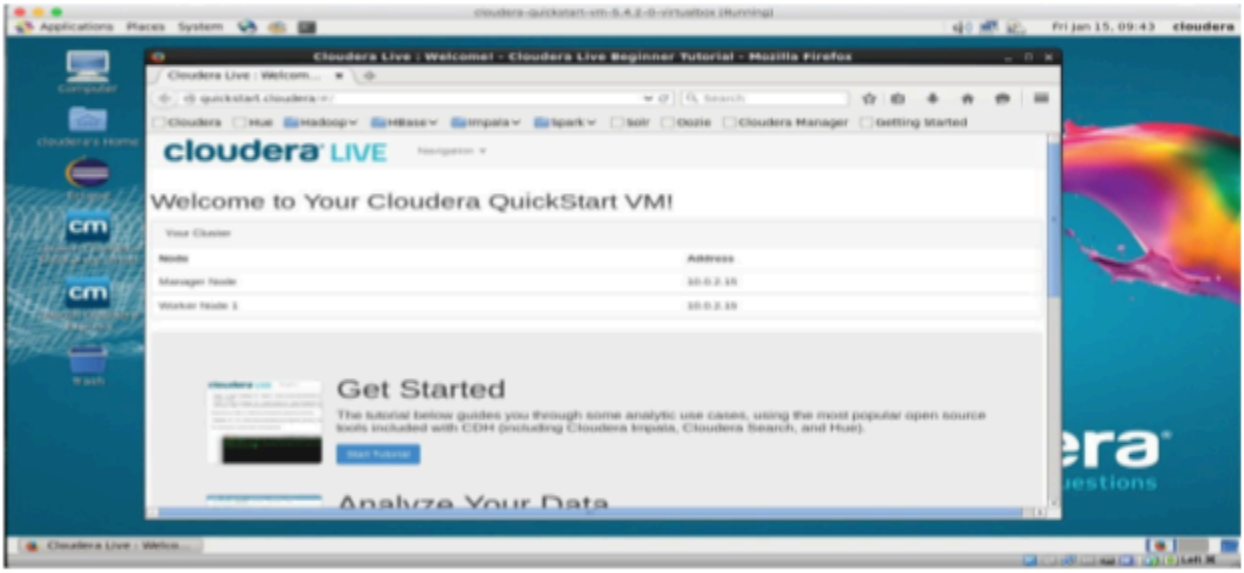
11. **Launch Cloudera VM.** When the importing is finished, the quickstart-vm-5.4.2-0 VM will appearon the left in the VirtualBox window. Select it and click the Start button to launch the VM.



12. **Cloudera VM booting.** It will take several minutes for the Virtual Machine to start. The bootingprocess takes a long time since many Hadoop tools are started.



12. **The Cloudera VM desktop.** Once the booting process is complete, the desktop will appear witha browser.



CONCLUSION: Studied installation of Hadoop and its configuration.

ASSIGNMENT 2

TITLE: Perform operations using Python on the Facebook metrics dataset

OBJECTIVES:

1. To understand and apply the Analytical concept of Big data using Python.
2. To study use of functions in python for data set operations.

SOFTWARE REQUIREMENTS:

1. Windows 10 or above
2. Jupyter Notebook

PROBLEM STATEMENT: Perform the following operations using Python on the Facebook metrics data sets

a. Create data subsets
b. Merge Data
c. Sort Data
d. Transposing Data
e. Shape and reshape Data

THEORY:

Python

Python is a powerful and versatile programming language known for its simplicity and readability. It excels in data visualization through libraries like Matplotlib, Seaborn, and Plotly, offering intuitive tools to create insightful charts, graphs, and interactive plots. Python's syntax facilitates quick prototyping and analysis, making it a preferred choice for data scientists and analysts. With its extensive ecosystem of data manipulation and visualization tools, Python enables users to explore, analyze, and communicate complex data effectively. Whether it's plotting trends, exploring relationships, or presenting findings, Python empowers users to visualize data with ease, driving informed decision-making across various domains.

Python Features

1. General Purpose: Python is a versatile programming language suitable for diverse applications from web development to scientific computing.
2. Readable Syntax: Python emphasizes simplicity and readability, making code writing and maintenance more straightforward and efficient.
3. Interpreted and Interactive: Python's interpreted nature allows for immediate code execution and interactive exploration, fostering rapid development and experimentation.
4. Dynamic and Strong Typing: Python employs dynamic typing for flexible variable assignments while maintaining strong typing to ensure code reliability and error prevention.
5. Extensive Standard Library: Python comes with a vast standard library, offering ready-to-use modules and functions for various tasks, minimizing the need for external dependencies.
6. Cross-platform Compatibility: Python runs seamlessly on different platforms, enabling code portability and deployment across multiple operating systems.
7. Rich Ecosystem of Libraries: Python boasts a vast ecosystem of third-party libraries and frameworks, catering to specialized domains like data analysis, machine learning, and web development, enhancing

productivity and capabilities.
8. Community-driven and Open Source: Python is developed and maintained by a thriving community of contributors worldwide, promoting collaboration, innovation, and accessibility in software development.

Dataset:

	A	B	C	D	E	F	G	H
1	Page total likes	Type	Category	Post Month	Post Weekday	Post Hour	Paid	Lifetime Post Total Reach
2	139441	Photo	2	12	4	3	0	2752
3	139441	Status	2	12	3	10	0	10460
4	139441	Photo	3	12	3	3	0	2413
5	139441	Photo	2	12	2	10	1	50128
6	139441	Photo	2	12	2	3	0	7244
7	139441	Status	2	12	1	9	0	10472
8	139441	Photo	3	12	1	3	1	11692
9	139441	Photo	3	12	7	9	1	13720
10	139441	Status	2	12	7	3	0	11844
11	139441	Photo	3	12	6	10	0	4694
12	139441	Status	2	12	5	10	0	21744
13	139441	Photo	2	12	5	10	0	3112
14	139441	Photo	2	12	5	10	0	2847
15	139441	Photo	2	12	5	3	0	2549
16	138414	Photo	2	12	4	5	1	22784
17	138414	Status	2	12	3	10	0	10060
18	138414	Photo	3	12	3	3	0	1722
19	138414	Photo	1	12	2	12	1	53264
20	138414	Status	3	12	2	3	0	3930
21	138414	Photo	3	12	1	11	0	1591
22	138414	Photo	2	12	1	3	0	2848
23	138414	Photo	1	12	7	10	0	1384
24	138414	Link	1	12	7	10	0	3454
25	138414	Photo	3	12	7	3	0	2723

Merge data sets :
To import a Facebook dataset and perform a merge operation on its subset in Python, you can use libraries like Pandas for data manipulation and merging. Pandas provides powerful tools for working with structured data, including importing datasets, filtering data, and merging datasets based on common keys. The steps are as follows
1. Importing the Facebook Dataset: Begin by importing the Facebook dataset into a Pandas DataFrame. You can use the `read_csv()` function provided by Pandas to read the dataset.
2. Exploring the Dataset: Explore the imported dataset to understand its structure and contents. This step helps identify the columns and keys that can be used for merging.
3. Performing the Merge Operation: Choose a subset of the dataset to merge with another dataset. Identify a common key (column) between the two datasets based on which the merge operation will be performed. Use the `merge()` function provided by Pandas to merge the datasets based on the common key.
Here's a sample code demonstrating these steps:

```
import pandas as pd
# Step 1: Import the Facebook Matrix Dataset
# Assuming the dataset is in a CSV format
facebook_matrix = pd.read_csv('facebook_matrix_dataset.csv')
# Step 2: Create Subsets
# Suppose we want to create subsets based on the values in column 'A' and 'B'
# Creating subset 1 where column 'A' is greater than 50
subset1 = facebook_matrix[facebook_matrix['A'] > 50]
# Creating subset 2 where column 'B' is less than 20
subset2 = facebook_matrix[facebook_matrix['B'] < 20]
# Step 3: Merge the Subsets
# Merging the two subsets on a common key, let's say 'user_id'
merged_subset = pd.merge(subset1, subset2, on='user_id', how='inner')
# Displaying the merged subset
print("Merged Subset:")
print(merged_subset.head())
```

- We use the `merge()` function to perform the merge operation. We specify the common key ('user_id') and the type of merge (inner, outer, left, or right) using the 'how' parameter.

sorting & Transposing the dataset :

```
# Sorting the dataset based on a specific column, let's say 'user_id'
sorted_facebook_matrix = facebook_matrix.sort_values(by='user_id')

# Displaying the sorted dataset
print("Sorted Facebook Matrix Dataset:")
print(sorted_facebook_matrix.head())
# Transposing the dataset to interchange rows and columns
transposed_facebook_matrix = sorted_facebook_matrix.transpose()
# Displaying the transposed dataset
print("\nTransposed Facebook Matrix Dataset:")
print(transposed_facebook_matrix.head())
```

Shape and Reshape

In Python, shaping and reshaping data involves manipulating the structure or dimensions of arrays or matrices. This process is commonly done using libraries like NumPy, which provides functions like `reshape()` to modify the shape of arrays. Here's a basic guide on how to shape and reshape data in Python:

1. Creating Arrays: Begin by creating arrays or matrices using NumPy. You can initialize arrays with desired dimensions and data types.
2. Reshaping Arrays: Use the `reshape()` function to change the shape of the array while preserving its elements. This function takes the desired shape as input.
3. Flattening Arrays: Flatten arrays using `flatten()` or `ravel()` functions to convert multi-dimensional arrays into one-dimensional arrays.
4. Transpose: Transpose arrays using `transpose()` or `T` attribute to interchange rows and columns.
5. Concatenation: Combine arrays along specified axes using functions like `concatenate()`.
6. Splitting: Split arrays into multiple smaller arrays using functions like `split()` or `hsplit()`.

Example:

```
import numpy as np
arr = np.array([[1, 2, 3],
                [4, 5, 6]])     # Create a 2D array
shape = arr.shape       # Reshape the array to a 1D array
print("Shape of the array:", shape)  # Output: (2, 3)
reshaped_arr = arr.reshape(-1)
print("Original Array:")
print(arr)
print("Reshaped Array:")
print(reshaped_arr)
```

This would output:

```
Original Array:
[[1 2 3]
 [4 5 6]]
Reshaped Array:
[1 2 3 4 5 6]
```

In this example, the `reshape(-1)` function call reshapes the 2D array `arr` into a 1D array while maintaining the elements' order.

Conclusion: Studied various operations ; Creating data subsets, Merge Data, Sort Data, Transposing Data, shape of data and reshaping the data and implemented in python.

ASSIGNMENT 3

TITLE: Web Scrapping

OBJECTIVES:

1. Design and develop Big data analytic application for emerging trends.
2. To study data collection from web resources

SOFTWARE REQUIREMENTS:

1. Windows 10 or above
2. Jupyter notebook

PROBLEM STATEMENT: Create a review scrapper for any ecommerce website to fetch real time comments, reviews, ratings, comment tags, customer name using Python.

THEORY: Web scraping is the process of extracting data from websites. It involves sending HTTP requests to the website's server, retrieving the HTML content, parsing the HTML to extract the desired information, and then saving or processing that information.

Key components of web scraping:

HTTP Requests: Use libraries like requests in Python to send HTTP requests to the website's server.
HTML Parsing: After receiving the HTML content, use libraries like BeautifulSoup to parse the HTML and extract relevant data.
Data Extraction: Identify the HTML elements that contain the data you want to scrape (e.g., review text, ratings, customer names) using CSS selectors or XPath.
Data Processing: Once you've extracted the data, you can process it further as needed, such as saving it to a file or database, performing sentiment analysis, or generating visualizations.

Python Libraries useful for web Scrapping

1. Requests:
   - Explanation: Used to send HTTP requests to web servers and retrieve HTML content.
   - Installation: You can install it via pip: `pip install requests`.

2. Beautiful Soup (bs4):
   - Explanation: A Python library for pulling data out of HTML and XML files. It provides simple methods and Pythonic idioms
     for navigating, searching, and modifying the parse tree.
   - Installation: You can install it via pip: `pip install beautifulsoup4`.

3. Selenium:
   - Explanation: A web automation tool that can be used for web scraping by simulating user interactions with the web page,
     such as clicking buttons and filling out forms. It's particularly useful for scraping JavaScript-rendered web pages.
   - Installation: You can install it via pip: `pip install selenium`.

4. Scrapy:
   - Explanation: An open-source and collaborative web crawling framework for Python. It provides a high-level API for
     extracting structured data from websites, as well as built-in support for handling requests, managing cookies, and following links.
   - Installation: You can install it via pip: `pip install scrapy`.

5. lxml:
   - Explanation: A Python library for processing XML and HTML documents. It provides a fast and efficient way to parse and
     manipulate XML/HTML documents.
   - Installation: You can install it via pip: `pip install lxml`.

6. Pandas:
   - Explanation: While not specifically designed for web scraping, Pandas is a powerful library for data manipulation and
     analysis. It can be useful for cleaning and structuring scraped data into DataFrame objects.
   - Installation: You can install it via pip: `pip install pandas`.

7. PyQuery:
   - Explanation: A jQuery-like library for Python that provides a simple API for parsing HTML documents and selecting
     elements using CSS selectors.
   - Installation: You can install it via pip: `pip install pyquery`.

These libraries offer various levels of abstraction and functionality for different web scraping needs. Depending on the complexity of the task and the specific requirements, you can choose the most suitable library or combination of libraries for your project.

Review Scrapper in python

```
import requests
from bs4 import BeautifulSoup

def scrape_amazon_reviews(url):
    # Send a GET request to the URL
    response = requests.get(url)

    # Check if the request was successful (status code 200)
    if response.status_code == 200:
        # Parse the HTML content
        soup = BeautifulSoup(response.content, 'html.parser')

        # Find all review containers
```

```
    review_containers = soup.find_all('div', class_='a-section review')

    # Iterate over each review container
    for review in review_containers:
        # Extract review text
        review_text = review.find('span', class_='review-text').get_text().strip()

        # Extract rating
        rating = float(review.find('i', class_='review-rating').get_text().split()[0])

        # Extract reviewer name
        reviewer_name = review.find('span', class_='a-profile-name').get_text().strip()

        # Extract comment tags (if available)
        comment_tags = [tag.get_text().strip() for tag in review.find_all('span', class_='a-size-base a-color-
        secondary review-
                        comment-tag-link')]

        # Print the extracted information
        print("Review Text:", review_text)
        print("Rating:", rating)
        print("Reviewer Name:", reviewer_name)
        print("Comment Tags:", comment_tags)
        print("="*50)
    else:
        print("Failed to fetch the page. Status code:", response.status_code)

# URL of the Amazon product page
url = 'https://www.amazon.com/product-url'

# Call the function to scrape reviews
scrape_amazon_reviews(url)
```

Explanation:

1. We start by importing the necessary libraries: `requests` for sending HTTP requests and `BeautifulSoup` for parsing HTML content.

2. We define a function `scrape_amazon_reviews(url)` that takes the URL of the Amazon product page as input.
3. Inside the function, we send a GET request to the URL and check if the request was successful (status code 200).

4. If the request was successful, we parse the HTML content using BeautifulSoup.

5. We find all review containers on the page using the appropriate CSS selector.

6. We iterate over each review container and extract the review text, rating, reviewer name, and comment tags (if available) using BeautifulSoup's methods.

7. Finally, we print the extracted information for each review.

Conclusion: Studied web scraping and implemented a review scraper in python.

ASSIGNMENT 4

TITLE: Write an application using HiveQL for flight information system

OBJECTIVES:

1. To learn Distributed database associated with Hadoop.
2. To study Apache Hive.

SOFTWARE REQUIREMENTS:

1. Cloudera
2. Hadoop
3. HIVE

PROBLEM STATEMENT: -Write an application using HiveQL for flight information system which will include

a. Creating, Dropping, and altering Database tables.
b. Creating an external Hive table.
c. Load table with data, insert new values and field in the table, Join tables with Hive
d. Create index on Flight Information Table
e. Find the average departure delay per day in 2008.

THEORY:

Hive:

Hive is a data warehouse infrastructure tool to process structured data in Hadoop. It resides on top of Hadoop to summarize Big Data, and makes querying and analyzing easy.

- Prerequisites: Core Java,
  Database concepts of SQL,
  Hadoop File system, and any of Linux operating system
- Features:
  - It stores schema in a database and processed data into HDFS.
  - It is designed for OLAP.

- It provides SQL type language for querying called HiveQL or HQL.
- It is familiar, fast, scalable, and extensible.

- Architecture:

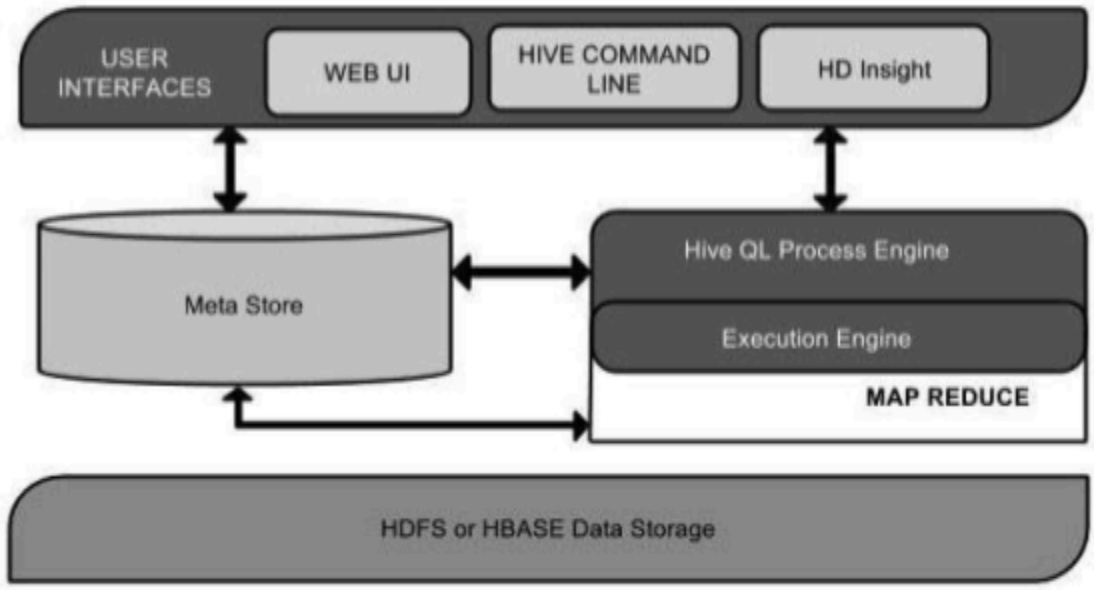


Figure 1: Hive Architecture

This component diagram contains different units. The following table describes each unit:

Unit Name	Operation
User Interface	Hive is a data warehouse infrastructure software that can create interaction between user and HDFS. The user interfaces that Hive supports are Hive Web UI, Hive command line, and Hive HD Insight (In Windows server).
Meta Store	Hive chooses respective database servers to store the schema or Metadata of tables, databases, columns in a table, their data types, and HDFS mapping.
HiveQL Process Engine	HiveQL is similar to SQL for querying on schema info on the Meta store. It is one of the replacements of traditional approach for MapReduce program. Instead of writing MapReduce program in Java, we can write a query for MapReduce job and process it.
Execution Engine	The conjunction part of HiveQL process Engine and MapReduce is Hive Execution Engine. Execution engine processes the query and generates results as same as MapReduce results. It uses the flavor of MapReduce.
HDFS or HBASE	Hadoop distributed file system or HBASE are the data storage techniques to store data into file system.

```
CREATE DATABASE IF NOT EXISTS flights;

 --Use the 'flights' database
 use flights;
```

a) Creating, Dropping, and altering Database tables.
1. Creating a Database Table for Flight Information:

```
-- Create the 'flights_info' internal table to store flight information
CREATE TABLE IF NOT EXISTS flights.flights_info (
    flight_id INT,
    airline_name STRING,
    origin STRING,
    destination STRING,
    departure_time TIMESTAMP,
    arrival_time TIMESTAMP
)
STORED AS ORC;
```

Explanation:
- This query creates a table named `flights_info` within the `flights` database to store flight information.
- Columns such as `flight_id`, `airline_name`, `origin`, `destination`, `departure_time`, and `arrival_time` are defined to hold relevant flight details.
The table is stored as ORC (Optimized Row Columnar) format, which is a highly efficient columnar storage format in Hive.
2. Dropping a Database Table for Flight Information:

```
DROP TABLE IF EXISTS flights.flights_info;
```

Explanation:
- This query drops the table named `flights_info` within the `flights` database if it exists.
- The `IF EXISTS` clause ensures that the table is dropped only if it exists, preventing errors if the table does not exist.

3. ltering a Database Table for Flight Information (Adding a Column):

```
-- Add a column 'delay_minutes' to the 'flights_info' table
ALTER TABLE flights.flights_info
ADD COLUMN delay_minutes INT;
```

Explanation:
- This query alters the `flights_info` table within the `flights` database by adding a new column named `delay_minutes` with the data type `INT`.
- The `ALTER TABLE` statement is used to modify the structure of the existing table, allowing for the addition of new columns.
- This operation is useful for accommodating new requirements or updates to the flight information system.
To create an external table named `nflight` stored as text files in Hive, you can use the following HiveQL query:

b) Creation of and external table

```
CREATE EXTERNAL TABLE IF NOT EXISTS nflight (
    flight_id INT,
    airline_name STRING,
    origin STRING,
```

```
    destination STRING,
    departure_time TIMESTAMP,
    arrival_time TIMESTAMP
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE
LOCATION '/user/hive/warehouse/nflight/';
```

Explanation:
CREATE EXTERNAL TABLE: This command creates an external table named nflight.
IF NOT EXISTS: It ensures that the table is created only if it doesn't already exist.
nflight: This is the name of the external table.
ROW FORMAT DELIMITED: Specifies that the rows in the text files are delimited.
FIELDS TERMINATED BY ',': Specifies that fields in each row are terminated by commas.
STORED AS TEXTFILE: Specifies that the data is stored as text files.
LOCATION '/user/hive/warehouse/nflight/': Specifies the HDFS directory where the external table data is located. Ensure that the Hive user has appropriate permissions to access this location.

c) Load table with data, insert new values and field in the table, Join the external tables with Hive

1)-- Load data into the 'nflight' table from an external location

```
LOAD DATA INPATH '/path/to/external/data' INTO TABLE nflight;
```

2) Insert new Values

```
-- Insert new values and fields into the 'nflight' table
INSERT INTO TABLE nflight
VALUES (1001, 'Airline X', 'City A', 'City B', '2024-04-30 08:00:00', '2024-04-30 10:00:00');
```

3) Join external table

```
-- Join external tables 'nflight' and 'another_table' on a common key
SELECT nflight.*, another_table.field
FROM nflight
JOIN another_table
ON nflight.common_key = another_table.common_key;
```

d)Create index on Flight Information Table

```
CREATE INDEX flight_id_index
ON TABLE nflight (flight_id)
AS 'compact'
WITH DEFERRED REBUILD
IDXPROPERTIES (
  'hive.index.compact.binary.search'='true'
);
```

This query creates an index named flight_id_index on the flight_id column of the nflight table using the 'compact' index handler. The index is initially created but not populated, and it has a property hive.index.compact.binary.search set to true.

e) Find average departure delay in yest 2008

```
SELECT
   departure_date,
   AVG(departure_delay) AS avg_departure_delay
FROM (
   SELECT
      SUBSTR(departure_time, 1, 10) AS departure_date,
      departure_delay
   FROM
      nflight
   WHERE
      SUBSTR(departure_time, 1, 4) = '2008'
) t
GROUP BY
         departure_date;
```

Conclusion : Studied Hive commands and implemented flight information system using Apache Hive.

ASSIGNMENT 5

TITLE: Visualize the data using Python by plotting the graphs for Facebook Metrics , Air quality and heart disease data sets.

OBJECTIVES:

1. To understand and apply the Analytical concept of Big data using Python.
2. To understand different data visualization techniques for Big Data

SOFTWARE REQUIREMENTS:

1. Windows 10 or above
2. Python SDK

THEORY:

Matplotlib library in Python Supports following types of Charts & Graphs:

- Pie chart • Bar chart • Box plots • Histograms • Line graphs • Scatter plots

Pie Chart:

Libraries: We'll use the `matplotlib.pyplot` library (imported as `plt`) for creating visualizations in Python.

Data Representation: Data for pie charts is typically represented as a list of numerical values, where each value represents the size of a slice in the pie. Additionally, a separate list can hold labels for each data point.

Creating the Pie Chart:

- The `plt.pie(data, labels=labels)` function is used to create the pie chart.
  - `data`: The list of numerical values for each slice.
  - `labels` (optional): The list of labels corresponding to each data point.

Customization:

- You can customize the pie chart using various functions from `matplotlib.pyplot`, such as:
  - `plt.title('Title')`: Add a title to the chart.
  - `plt.xlabel('X-axis Label')` and `plt.ylabel('Y-axis Label')`: Add labels for the axes (not applicable for pie charts).
  - `plt.legend(labels)`: Add a legend with slice labels (if provided).
  - Color options: Use arguments like `colors` or a colormap to define slice colors.

- `plt.autopct='%1.1f%%'`: Display percentages on the chart slices, formatted with one decimal place (optional).

Displaying the Chart:

- Use `plt.show()` to display the generated pie chart.

Pie Chart Example:

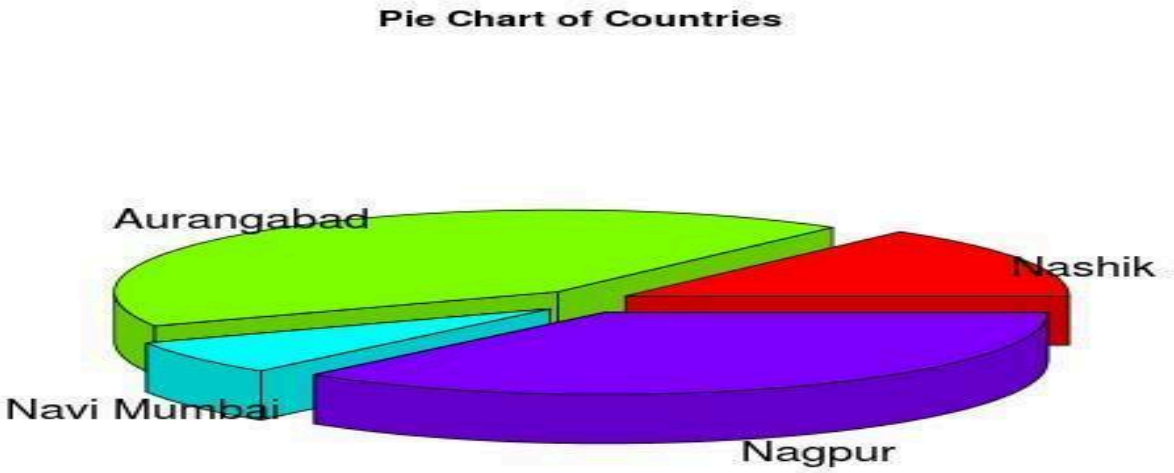
```
import matplotlib.pyplot as plt
# Sample data (replace with your actual data)
data = [30, 40, 20, 10]
labels = ['Category 1', 'Category 2', 'Category 3', 'Category 4']
# Create the pie chart
plt.pie(data, labels=labels, autopct="%1.1f%%")  # Display percentages with one decimal place
# Customize the chart (optional)
plt.title('Pie Chart')
# Display the chart
plt.show()
```

Pie Chart With Colors:

```
import matplotlib.pyplot as plt
# Sample data (replace with your actual data)
data = [21, 62, 10, 53]
labels = ["Pune", "Nashik", "Aurangabad", "Mumbai"]
# Create the pie chart with colors
plt.pie(data, labels=labels, autopct="%1.1f%%", colors=plt.cm.rainbow(len(data)))
# Use rainbow colormap
plt.title("City Pie Chart")
plt.show()
```

3D Pie Chart:

```
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
# Sample data (replace with your actual data)
data = [21, 62, 10, 53]
labels = ["Nashik", "Aurangabad", "Navi Mumbai", "Nagpur"]
# Create the figure and 3D axis
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
# Create the pie chart with slight explosion for 3D effect
explode = (0.1, 0.2, 0.1, 0.0)  # Adjust explosion for each slice (optional)
wedges, texts, autotexts = ax.pie(data, autopct="%1.1f%%", labels=labels, explode=explode)
# Set labels and title
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_zlabel('Z-axis')
plt.title("Pie Chart of Countries")
plt.show()
```



Bar Charts:

- Bar charts represent data using rectangular bars where the bar height (or width for horizontal bars) is proportional to the data value.
- `matplotlib.pyplot` (imported as `plt`) provides functions for creating bar charts.
- You can create both vertical and horizontal bar charts.
- Each bar can be assigned a different color for better visualization.

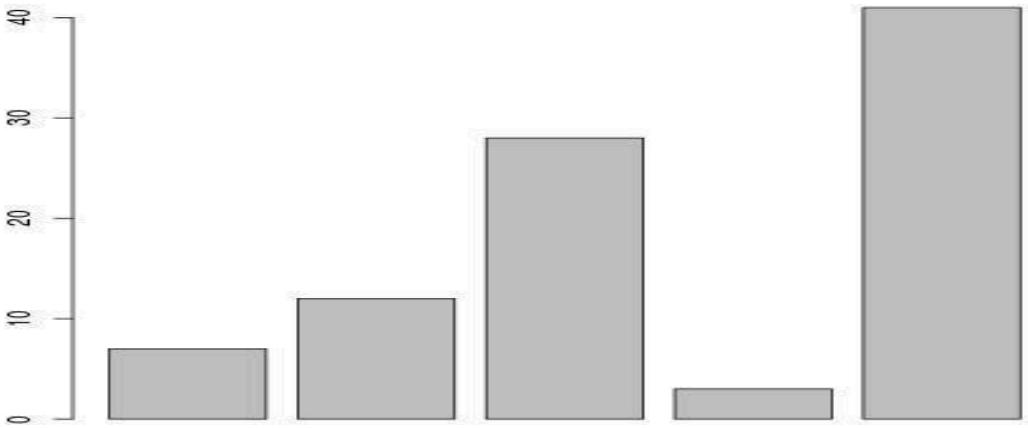
Bar Charts- Syntax:

`plt.bar(x, height, width=..., color=...)`: This is the basic function for creating a bar chart.

- `x`: A list of positions for each bar on the X-axis.
- `height`: A list of heights for each bar.
- `width` (optional): The width of each bar (default is 0.8).
- `color` (optional): A list of colors for each bar, or a single color for all bars.

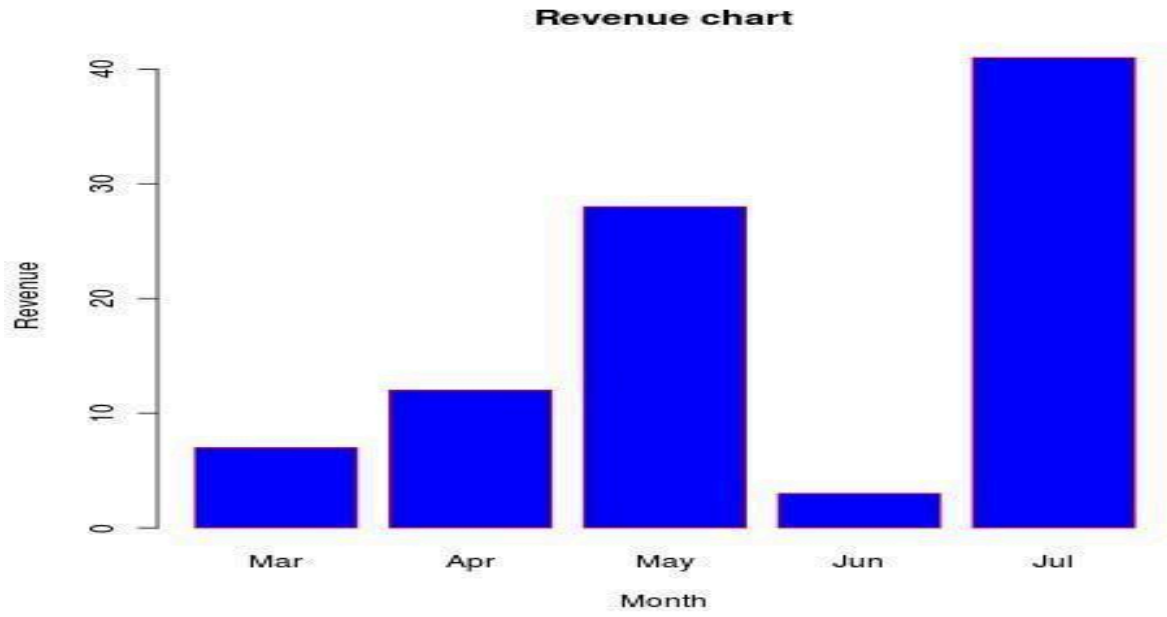
Barchart Example:

```
import matplotlib.pyplot as plt
# Sample data (replace with your actual data)
data = [7, 12, 28, 3, 41]
# Create the bar chart
plt.bar(range(len(data)), data)  # Default color will be used
# Optional: Customize the chart (consider adding labels and title)
plt.xlabel("X-axis Label (Optional)")
plt.ylabel("Y-axis Label (Optional)")
plt.title("Bar Chart Title (Optional)")
# Display the chart
plt.show()
```



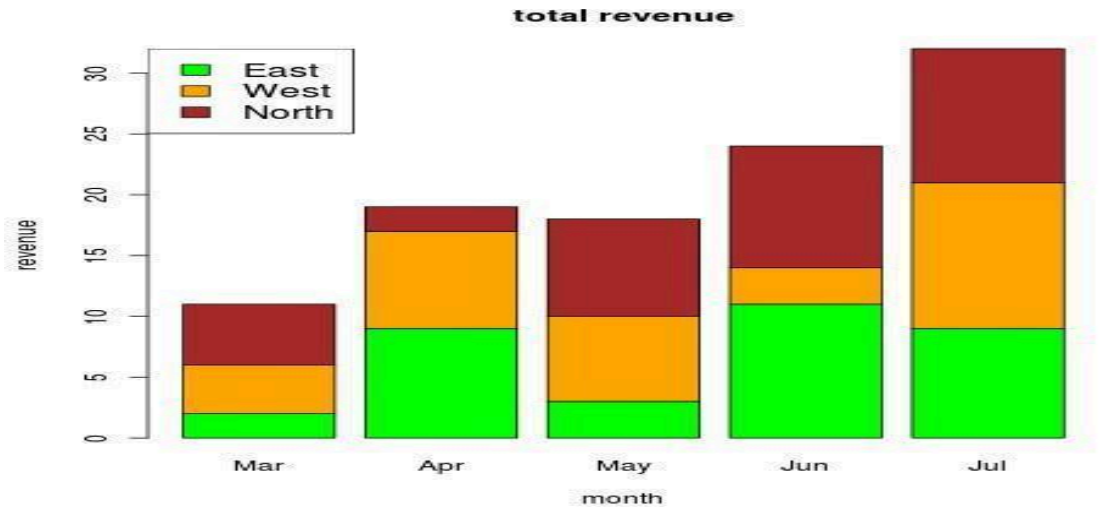
Barchart with Attribute:

```
import matplotlib.pyplot as plt
# Sample data (replace with your actual data)
data = [7, 12, 28, 3, 41]
month_labels = ["Mar", "Apr", "May", "Jun", "Jul"]  # Month labels for the X-axis
# Create the bar chart
plt.bar(month_labels, data, color='blue')  # Set color
# Add labels and title
plt.xlabel("Month")
plt.ylabel("Revenue")
plt.title("Revenue Chart")
# Set bar labels on the X-axis (optional)
plt.xticks(rotation=0)  # Optional: Rotate labels if needed
# Add border color (optional)
plt.gca().bar_label(plt.gca().containers[0])  # Get bars
for bar in plt.gca().containers[0]:
    bar.set_edgecolor('red')  # Set border color to red
# Display the chart
plt.show()
```



Barchart Stacked:

```
import matplotlib.pyplot as plt
# Sample data (replace with your actual data)
colors = ["green", "orange", "brown"]
months = ["Mar", "Apr", "May", "Jun", "Jul"]
regions = ["East", "West", "North"]
values = [[2, 9, 3], [11, 9, 4], [8, 7, 3], [12, 5, 2], [8, 10, 11]]  # Values matrix
# Create the stacked bar chart
plt.stackplot(months, values, labels=regions, colors=colors)
# Add labels and title
plt.xlabel("Month")
plt.ylabel("Revenue")
plt.title("Total Revenue")
# Add legend
plt.legend(loc='upper left', title='Regions', title_fontsize=12, bbox_to_anchor=(1, 1))  # Adjust
legend position
# Display the chart
plt.show()
```



Boxplot:

- Boxplots are a measure of how well distributed is the data in a data set.
- It divides the data set into three quartiles. This graph represents the minimum, maximum, median, first quartile and third quartile in the data set.
- It is also useful in comparing the distribution of data across data sets by drawing boxplots for each of them.

- **Function:** `matplotlib.pyplot.boxplot(data)` (from `matplotlib.pyplot`)
- **Purpose:** Creates a boxplot to show the spread and variability of a dataset.
- **Applications:**

  - Identifying potential outliers
  - Comparing distributions across multiple datasets

Boxplot Syntax:
This will create a boxplot showing the distribution of your data.

```
import matplotlib.pyplot as plt
# Your data (replace with your actual data)
data = [5, 7, 2, 8, 1, 9, 3, 4, 10]
# Basic boxplot
plt.boxplot(data)
plt.show()
```

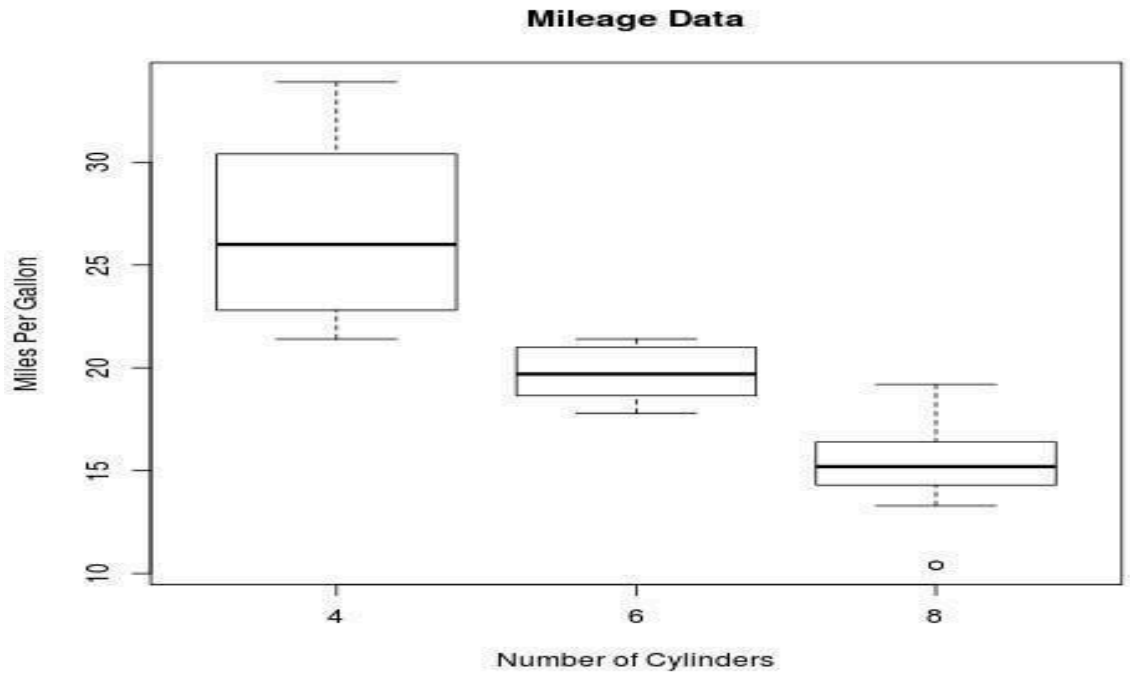
Here are some optional arguments you can use to customize the boxplot:

- `notch` (boolean, True/False): Add a notch to the box indicating variability within medians.
- `vert` (boolean, True/False): Create a vertical (default) or horizontal boxplot.
- `patch_artist` (boolean, True/False): Use patches for box elements, allowing for further customization (colors, patterns).
- `labels` (list): List of labels for multiple boxplots on the same plot.
- `showmeans` (boolean, True/False): Display means as markers (default: False).
- `medianprops` (dictionary): Customize median line properties (e.g., color, linewidth).

- `boxprops` (dictionary): Customize box properties (e.g., color, fill color).
- `whiskerprops` (dictionary): Customize whisker properties (e.g., color, linewidth).
- `capprops` (dictionary): Customize cap properties (e.g., color, linewidth).

Boxplot Example:

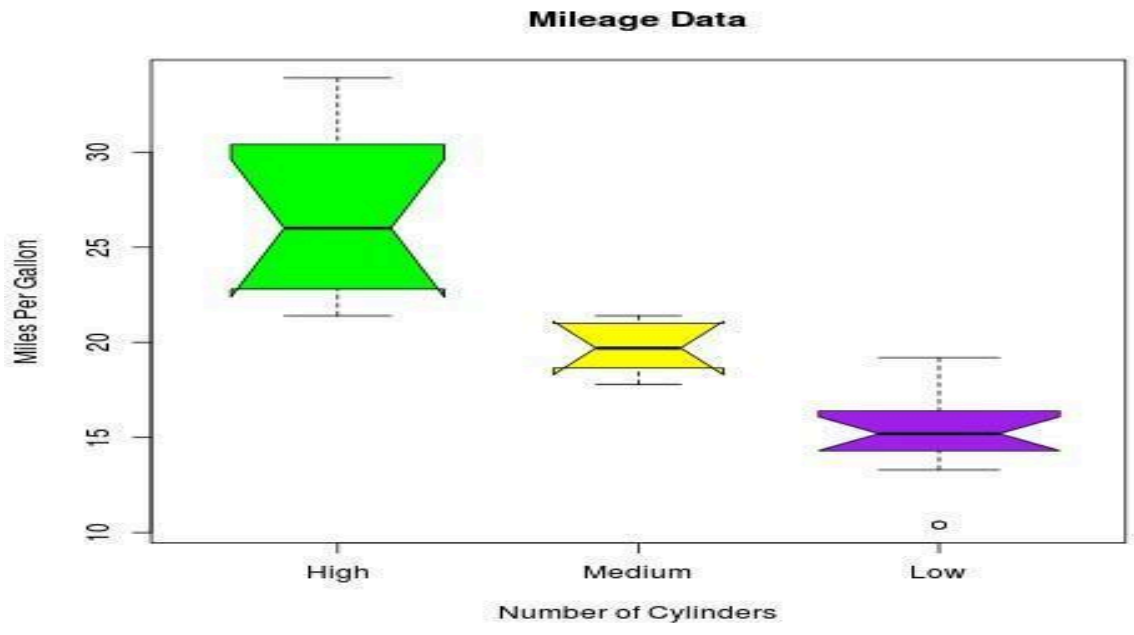
```
import matplotlib.pyplot as plt
import seaborn as sns  # Optional for aesthetics (consider installing seaborn)
# Load the "mtcars" dataset (assuming it's available in your environment)
# If using a different dataset, replace accordingly
data = sns.load_dataset("mtcars")
# Extract the desired columns
mpg = data["mpg"]
cyl = data["cyl"]
# Create the boxplot (using seaborn for a nicer plot, optional)
sns.boxplot(
    x="cyl",
    y="mpg",
    showmeans=True,  # Display means as markers (optional)
    data=data
)
# Customize labels and title (optional)
plt.xlabel("Number of Cylinders")
plt.ylabel("Miles Per Gallon")
plt.title("Mileage Data")
# Save the plot as a PNG image (optional)
# plt.savefig("boxplot.png")  # Uncomment to save
plt.show()
```



Boxplot with Notch:

```
import matplotlib.pyplot as plt
# Load the "mtcars" dataset (assuming it's available in your environment)
# If using a different dataset, replace accordingly
```

```
data = sns.load_dataset("mtcars")
# Extract the desired columns
mpg = data["mpg"]
cyl_groups = data["cyl"].unique()  # Get unique cylinder values
# Define color mapping for cylinder groups (modify as needed)
cyl_colors = {"4": "green", "6": "yellow", "8": "purple"}
# Create boxplot groups based on cylinder values (assuming 3 groups)
# Modify this section if your data has different cylinder categories
boxplot_data = []
for cyl_value in cyl_groups:
    cyl_data = mpg[data["cyl"] = = cyl_value]
    boxplot_data.append(cyl_data)
names = ["High", "Medium", "Low"]  # Adjust names based on your cylinder categories
# Create the boxplot with notch and variable width
plt.boxplot(
    data=boxplot_data,
    notch=True,
    vert=True,  # Ensure vertical boxplot (default)
)
# Customize labels and title
plt.xlabel("Number of Cylinders")
plt.ylabel("Miles Per Gallon")
plt.title("Mileage Data with Notch and Variable Width")
# Set box colors based on cylinder groups
box_container = plt.gca().get_children()[0]  # Access boxplot container
for box, color in zip(box_container.boxes, cyl_colors.values()):
    box.set_facecolor(color)
# Save the plot as a PNG image (optional)
# plt.savefig("boxplot_with_notch.png")  # Uncomment to save
plt.show()
```



Histogram:

- A histogram represents the frequencies of values of a variable bucketed into ranges.
- Histogram is similar to bar chart but the difference is it groups the values into continuous ranges.
- Each bar in histogram represents the height of the number of values present in that range.

- **Function:** `matplotlib.pyplot.hist(data, bins=None, range=None, density=False)` (from `matplotlib.pyplot`)
- **Purpose:** Visualizes the distribution of a dataset by dividing it into bins (ranges) and plotting the frequency of values within each bin.
- **Applications:**
  - Identifying patterns in data distribution (e.g., skewness, normality)
  - Comparing distributions across datasets

Histogram Syntax:

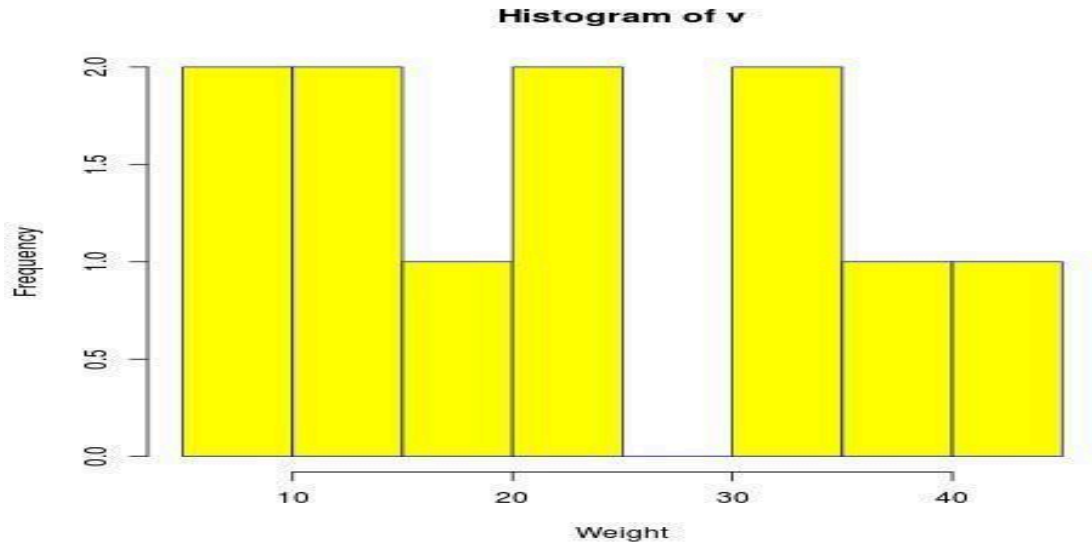
```
matplotlib.pyplot.hist(data, bins=None, range=None, density=False)
```

Explanation:

- `data`: The list or NumPy array containing the data points for the histogram.
- `bins` (optional): Number of bins (intervals) to divide the data into. By default, `matplotlib` chooses a reasonable number.
- `range` (optional): Tuple of minimum and maximum values for the x-axis.
- `density` (optional): Boolean (True/False). If True, the histogram is normalized to represent probability density.

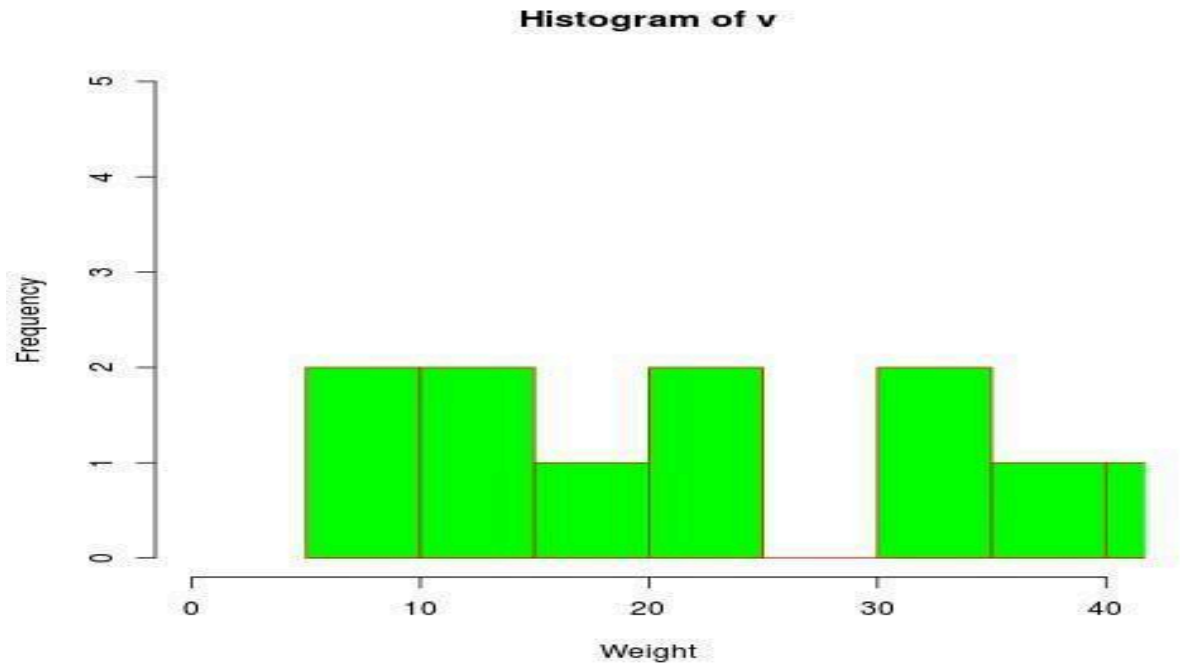
Histogram Example:

```
import matplotlib.pyplot as plt
# Sample data (replace with yours)
data = [9, 13, 21, 8, 36, 22, 12, 41, 31, 33, 19]
# Create the histogram
plt.hist(data, bins=None, color="yellow", edgecolor="blue")  # Adjust bins if needed
# Customize labels and title
plt.xlabel("Weight")
plt.ylabel("Frequency")
plt.title("Histogram of Weight Data")
# Save the plot as a PNG image (optional)
# plt.savefig("histogram.png")  # Uncomment to save
plt.show()
```




```
import matplotlib.pyplot as plt
# Sample data (replace with yours)
data = [9, 13, 21, 8, 36, 22, 12, 41, 31, 33, 19]
```

```
# Create the histogram with limits and breaks
plt.hist(
    data,
    bins=5,  # Set the number of bins based on breaks
    color="green",
    edgecolor="red",
    xlim=(0, 40),  # Set x-axis limits
    ylim=(0, 5),  # Set y-axis limits
)
# Customize labels and title
plt.xlabel("Weight")
plt.ylabel("Frequency")
plt.title("Histogram with Limits and Breaks")
# Save the plot as a PNG image (optional)
# plt.savefig("histogram_lim_breaks.png")  # Uncomment to save
plt.show()
```



Line Graph:

- A line chart is a graph that connects a series of points by drawing line segments between them.
- These points are ordered in one of their coordinate (usually the x-coordinate) value.
- Line charts are usually used in identifying the trends in data.

Function: `matplotlib.pyplot.plot(x, y)` (from `matplotlib.pyplot`)
Purpose: Visualizes the relationship between two variables by plotting a series of connected data points.
Applications:

- Identifying trends and patterns in data over time or across categories
- Comparing multiple datasets on the same plot

Line Graph – Syntax:

```
import matplotlib.pyplot as plt
# Sample data (replace with yours)
```

```
x = [1, 2, 3, 4, 5]
y = [2, 5, 7, 1, 3]
# Create the line graph
plt.plot(x, y)
plt.show()
```

Explanation:

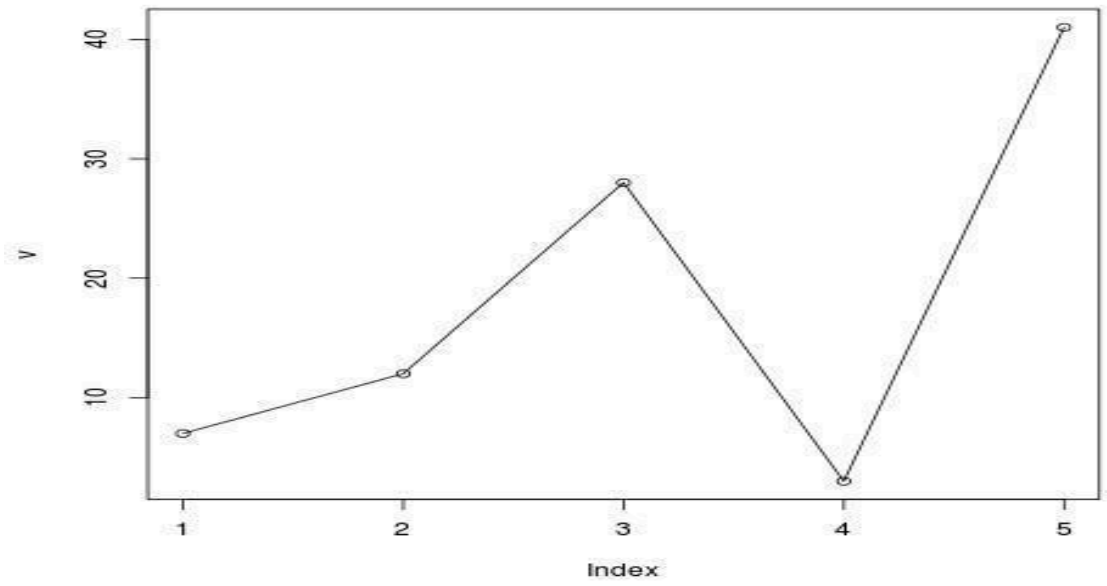
- `plt.plot(x, y)`: This is the core function. It takes two lists or NumPy arrays (`x` and `y`) representing the data points for the x and y axes, respectively. These elements are connected to form a line graph.

Customization Options:

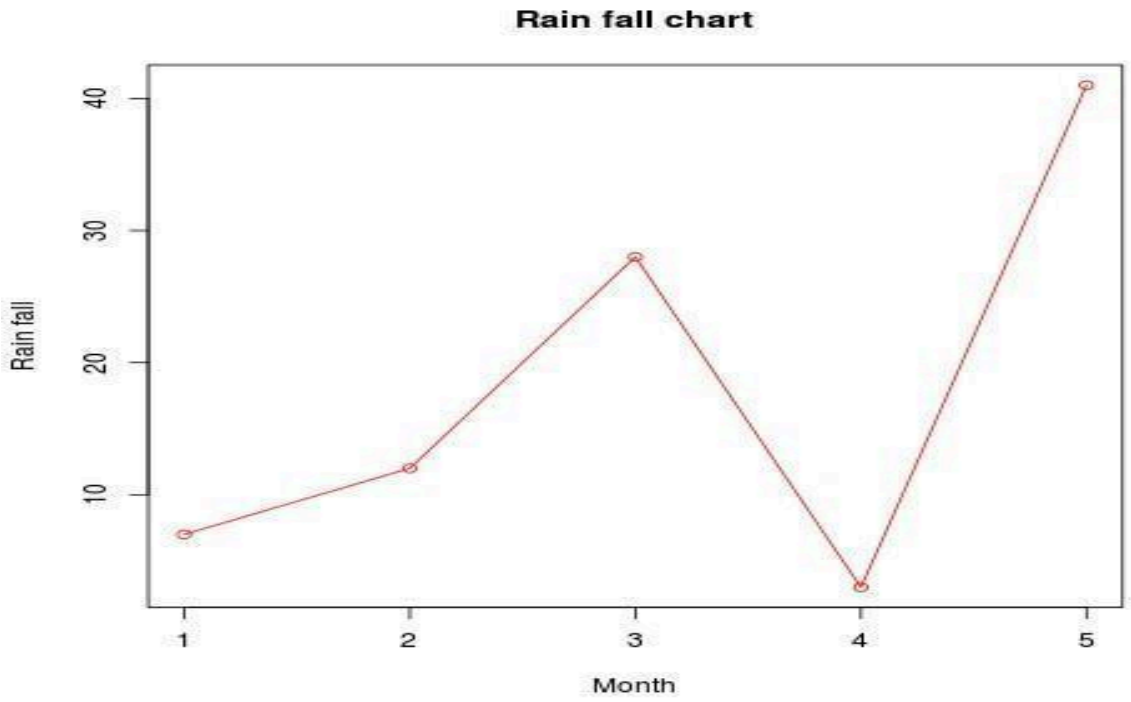
- **Markers:** Add markers to data points using the `marker` argument within `plt.plot()`. Common marker styles include:
  - `'o'`: Circles (default)
  - `'s'`: Squares
  - `'^'`: Triangles
  - `'x'`: X-shaped markers

Line Graph-Example:

```
import matplotlib.pyplot as plt
# Create the data (replace with your actual data)
data = [7, 12, 28, 3, 41]
# Create the line plot
plt.plot(data, marker='o')  # Use 'o' for circles as markers
# Customize the plot (optional)
plt.xlabel('X-axis Label')
plt.ylabel('Y-axis Label')
plt.title('Line Graph Title')
# Save the plot as an image (optional)
plt.savefig('line_chart.png')
# Display the plot
plt.show()
```



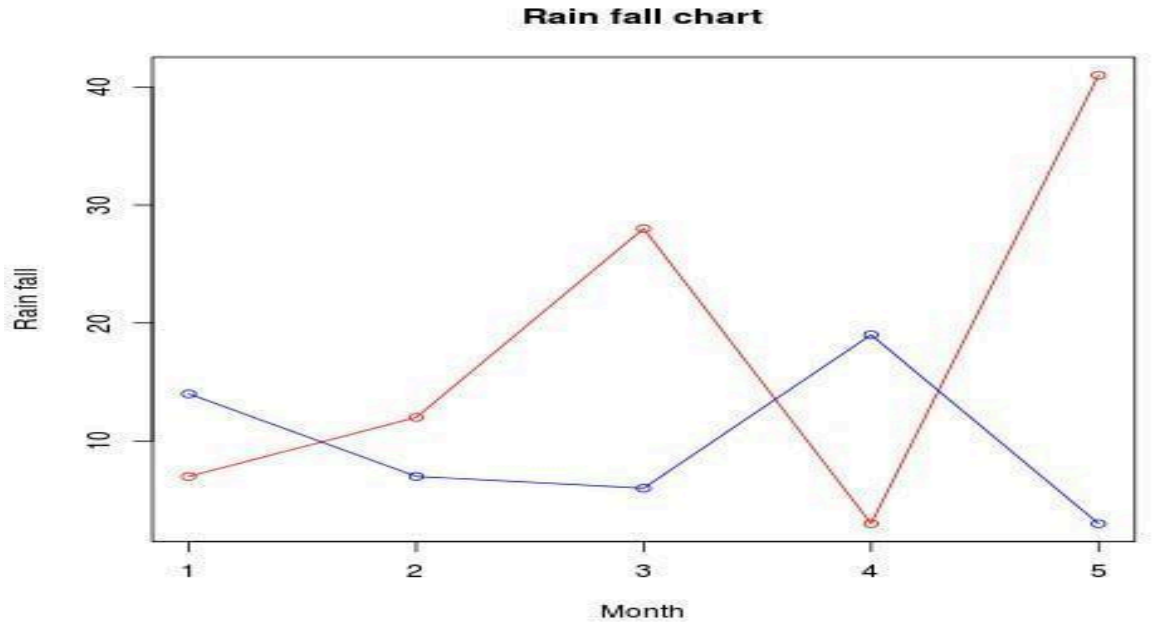
```
import matplotlib.pyplot as plt
# Create the data
data = [7, 12, 28, 3, 41]
# Create the line plot with color and markers
plt.plot(data, marker='o', color='red', label='Rainfall')  # Use 'o' for circles and 'red' for color, add label
# Customize the plot with labels and title
plt.xlabel('Month')
plt.ylabel('Rainfall')
plt.title('Rainfall Chart')
# Add legend (optional)
plt.legend()  # Show the label for the line
# Save the plot as an image (optional)
plt.savefig('line_chart_label_colored.png')
# Display the plot
plt.show()
```



Multiple Lines in Chart-Example:

```
import matplotlib.pyplot as plt
# Create the data
data1 = [7, 12, 28, 3, 41]  # Rainfall data (v)
data2 = [14, 7, 6, 19, 3]   # Other data (t)
# Create the line plot with color and markers
plt.plot(data1, marker='o', color='red', label='Rainfall')
plt.plot(data2, marker='o', color='blue', label='Other Data')  # Add second line with blue color and label
# Customize the plot with labels and title
plt.xlabel('Month')
plt.ylabel('Values')  # Use a generic label as data may not represent rainfall
plt.title('Line Chart with Two Lines')
# Add legend (optional)
plt.legend()  # Show labels for both lines
# Save the plot as an image (optional)
plt.savefig('line_chart_2_lines.png')
# Display the plot
```

```
plt.show()
```

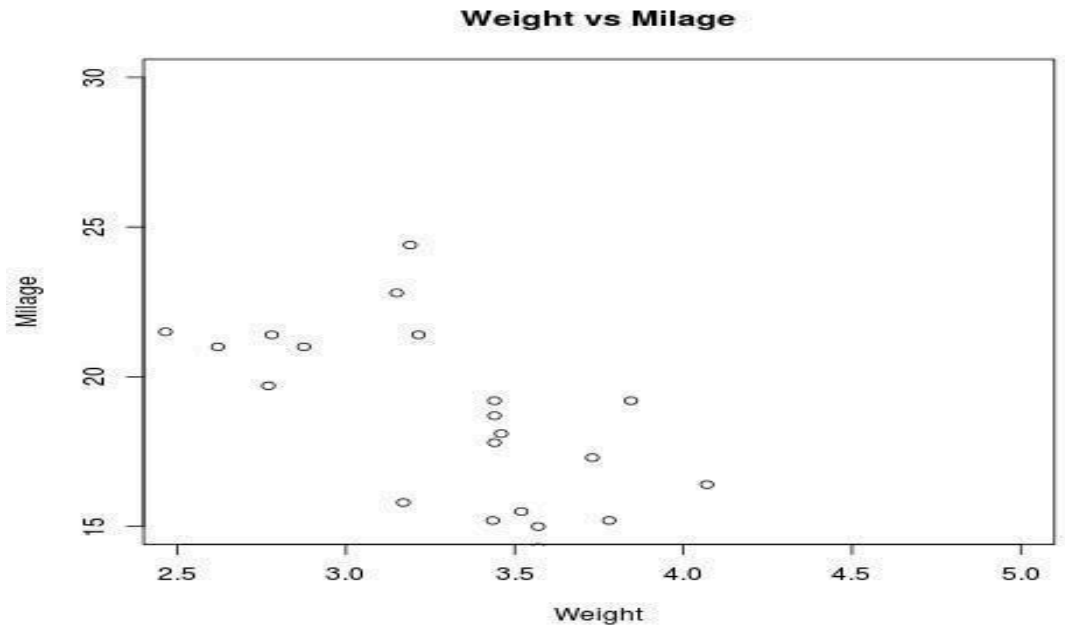


Scatter plot:

- Scatterplots show many points plotted in the Cartesian plane.
- Each point represents the values of two variables.
- One variable is chosen in the horizontal axis and another in the vertical axis.
- The simple scatterplot is created using the plt.scatter() function.

Scatterplot Example:

```
import pandas as pd
import matplotlib.pyplot as plt
# Assuming you have the 'mtcars' dataset available (replace with your data source)
# If using a CSV file, use pd.read_csv('mtcars.csv')
data = pd.read_csv('mtcars.csv')  # Replace with your data loading method
# Filter data for weight between 2.5 and 5 and mileage between 15 and 30
filtered_data = data[(data['wt'] >= 2.5) & (data['wt'] <= 5) & (data['mpg'] >= 15) & (data['mpg'] <= 30)]
# Extract weight and mileage columns
weight = filtered_data['wt']
mileage = filtered_data['mpg']
# Create the scatter plot
plt.scatter(weight, mileage)
# Customize the plot with labels, title, and limits
plt.xlabel('Weight')
plt.ylabel('Mileage')
plt.title('Weight vs. Milage (Filtered)')
plt.xlim(2.5, 5)  # Set x-axis limits
plt.ylim(15, 30)  # Set y-axis limits
# Display the plot
plt.show()
# Save the plot as an image (optional)
# plt.savefig('scatterplot.png')  # Uncomment to save the plot
```



Scatter plot Matrices:

- When we have more than two variables and we want to find the correlation between one variable versus the remaining ones we use scatterplot matrix.
- Use the `pandas.plotting.scatter_matrix` function to create the scatter plot matrix directly from your DataFrame
- Syntax:
  pd.plotting.scatter_matrix(data, alpha=0.8, figsize=(10, 6), diagonal='hist')

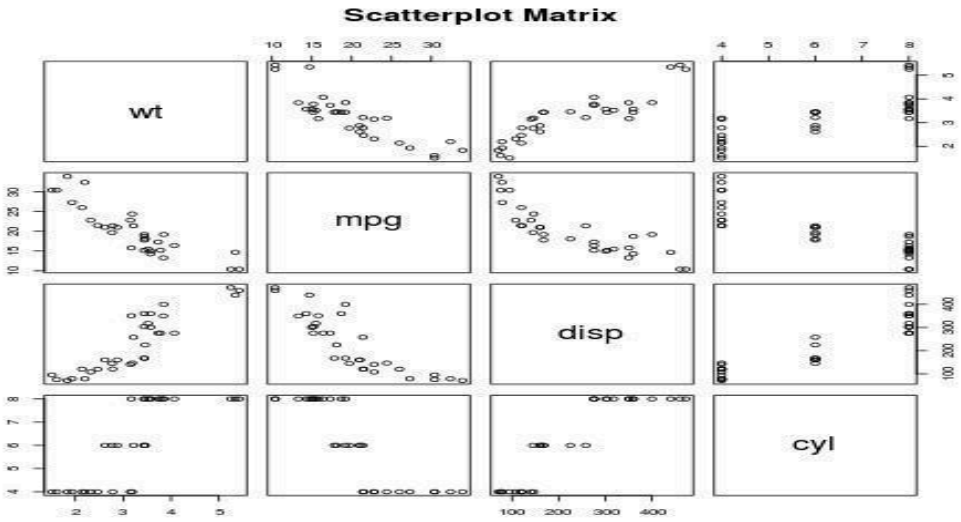
Optional Customizations:

- `alpha`: Adjusts the transparency of the scatter plot points (default is 1, fully opaque).
- `figsize`: Sets the size of the plot figure (width, height).
- `diagonal`: Controls what to display on the diagonal plots. 'hist' creates histograms, 'kde' creates kernel density estimates, None leaves the diagonal blank.

Scatterplot Matrices- Example:

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

# Assuming you have the 'mtcars' dataset available (replace with your data source)
# If using a CSV file, use pd.read_csv('mtcars.csv')
data = pd.read_csv('mtcars.csv')  # Replace with your data loading method
# Select the variables
variables = ['wt', 'mpg', 'disp', 'cyl']
# Create the scatter plot matrix
pd.plotting.scatter_matrix(data[variables], alpha=0.8, figsize=(10, 10), main='Scatterplot Matrix')
plt.show()
```



Conclusion: Learned how to Visualize the data with different types of Charts and implemented using Python.

ASSIGNMENT 6

TITLE: Air quality and Heart Disease data analysis

OBJECTIVES:

1. To understand the different Big data processing techniques.
2. To understand and apply the Analytical concept of Big data using Python.

SOFTWARE REQUIREMENTS:

1. Windows 10 or above
2. Jupyter Notebook

PROBLEM STATEMENT: Perform the following operations using Python on the Air quality and heart diseases data sets.

a. Data cleaning
b. Data integration
c. Data transformation
d. Error correcting
e. Data model building

THEORY:

a) Data Cleaning:

1) Handling missing values
Handling Missing Values: Detect missing values in the dataset.
Drop rows or columns with missing values or fill them with appropriate values such as mean, median, or mod

```
import pandas as pd
# Load dataset
air_quality = pd.read_csv('air_quality.csv')
# Check for missing values
missing_values = air_quality.isnull().sum()
# Drop rows with missing values
air_quality_cleaned = air_quality.dropna()
# Fill missing values with mean
air_quality['PM10'].fillna(air_quality['PM10'].mean(), inplace=True)
```

2) Removing Duplicates:
Identify and remove duplicate records in the dataset.
Ensure each observation is unique, preventing redundancy and potential bias in analysis.

```
# Drop duplicates
air_quality_cleaned = air_quality.drop_duplicates()
```

3) Correcting Data Types:
Convert data types of columns to their appropriate formats, such as converting string representations of dates to datetime objects.

Ensure consistency and accuracy in data representation, facilitating meaningful analysis.

```
# Convert date column to datetime
air_quality['Date'] = pd.to_datetime(air_quality['Date'])
```

4) Handling Outliers:

Detect and address outliers using statistical methods like z-score or interquartile range.
Mitigate the impact of extreme values on analysis, ensuring robustness and reliability of insights.

```
from scipy import stats
# Detect outliers using z-score
z_scores = stats.zscore(air_quality['PM10'])
air_quality_no_outliers = air_quality[(z_scores < 3)]
import pandas as pd
# Load dataset
air_quality = pd.read_csv('air_quality.csv')
```

Interquartile range method:

```
# Calculate the first quartile (Q1) and third quartile (Q3) of the target column
Q1 = air_quality['PM10'].quantile(0.25)
Q3 = air_quality['PM10'].quantile(0.75)
# Calculate the interquartile range (IQR)
IQR = Q3 - Q1
# Define the lower and upper bounds for outlier detection
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
# Identify outliers based on the lower and upper bounds
outliers = air_quality[(air_quality['PM10'] < lower_bound) | (air_quality['PM10'] > upper_bound)]
# Remove outliers from the dataset
air_quality_no_outliers = air_quality[~((air_quality['PM10'] < lower_bound) | (air_quality['PM10'] >
upper_bound))]
```

5) Standardizing data:

Scale numerical features to have a mean of 0 and a standard deviation of 1 using techniques like Standard Scaler.
Ensure comparability and interpretability of data across different scales, enhancing the effectiveness of machine learning models and analysis.

```
from sklearn.preprocessing import StandardScaler
# Standardize numerical columns
scaler = StandardScaler()
air_quality[['NO2', 'PM10']] = scaler.fit_transform(air_quality[['NO2', 'PM10']])
```

b) Data Integration: Data integration is the process of combining data from disparate sources into a unified format. It involves extracting, transforming, and loading data to create a cohesive dataset. By harmonizing data from various sources, organizations can gain a comprehensive view of their operations, enabling better decision-making and insights. Integration methods include ETL (Extract, Transform, Load), ELT (Extract, Load, Transform), and API-based approaches. The goal is to provide a consolidated and accurate representation of data for analysis and reporting purposes.

Example:

```
import pandas as pd
sales_data = pd.read_csv('sales_data.csv')   # Load sales data from CSV
```

```
# Load customer data from CSV
customer_data = pd.read_csv('customer_data.csv')
# Perform data integration based on common field
(e.g., CustomerID)
integrated_data = pd.merge(sales_data, customer_data, on='CustomerID', how='inner')
# Store integrated data into a new CSV file
integrated_data.to_csv('integrated_data.csv', index=False)
 print(integrated_data)  # Display the integrated data
```

c) Data Transformation Techniques

1.Data Smoothing :

Data smoothing is a technique used to remove noise from a dataset by replacing noisy data points with a smoother version of the data.

It helps in identifying trends and patterns in the data by reducing fluctuations and irregularities.

2.Attribution Construction:

Attribution construction involves creating new features or attributes from existing data to improve the performance of machine learning models.

It enhances the predictive power of models by capturing additional information or relationships present in the data.

3.Data Generalization:

Data generalization involves summarizing or simplifying data to a higher level of abstraction while retaining important information.

It helps in reducing complexity and dimensionality of data, making it easier to analyze and interpret.

4.Data Aggregation:

Data aggregation combines multiple data points into a single representative value, typically by applying functions like sum, average, or count.

It helps in summarizing large datasets and extracting meaningful insights at different levels of granularity.

5.Data Discretization:

Data discretization divides continuous numerical data into discrete intervals or categories.

It simplifies complex data and makes it suitable for analysis by converting continuous features into categorical ones.

6.Data Normalization:

Data normalization scales numerical features to a standard range, often between 0 and 1, to ensure that all features have the same scale.

It helps in improving the performance of machine learning algorithms by preventing features with larger magnitudes from dominating the model.

Examples:

Data Transformation using python

Normalization :

```
from sklearn.preprocessing import MinMaxScaler
# Example data
data = [[1, 2], [3, 4], [5, 6]]
# Initialize MinMaxScaler
scaler = MinMaxScaler()
# Fit and transform data
normalized_data = scaler.fit_transform(data)
print(normalized_data)
```

Standardization:

```
from sklearn.preprocessing import StandardScaler
```

```
# Example data
data = [[1, 2], [3, 4], [5, 6]]
# Initialize StandardScaler
scaler = StandardScaler()
# Fit and transform data
standardized_data = scaler.fit_transform(data)
print(standardized_data)
```

One-Hot Coding:

```
import pandas as pd
# Example data
data = pd.DataFrame({'category': ['A', 'B', 'C', 'A']})
# Perform one-hot encoding
encoded_data = pd.get_dummies(data)
print(encoded_data)
```

Data Discretization:

```
 import pandas as pd
from sklearn.preprocessing import KBinsDiscretizer
# Sample data
data = pd.DataFrame({'age': [22, 35, 47, 55, 68, 80]})
# Initialize KBinsDiscretizer
discretizer = KBinsDiscretizer(n_bins=3, encode='ordinal', strategy='uniform')
# Fit and transform data
discretized_data = discretizer.fit_transform(data)
# Convert the transformed data back to a DataFrame
discretized_df = pd.DataFrame(discretized_data, columns=data.columns)
print(discretized_df)
```

In this example we have a sample dataset data containing ages of individuals.
We initialize the KBinsDiscretizer from scikit-learn with parameters n_bins=3 (number of bins), encode='ordinal' (encoding method), and strategy='uniform' (binning strategy).
We then fit and transform the data using the fit_transform() method.
Finally, we convert the transformed data back to a DataFrame and print the result.

Data Smoothing :

```
import pandas as pd
# Sample data
data = pd.Series([5, 10, 15, 20, 25, 30, 35])
# Define window size for moving average
window_size = 3

# Perform data smoothing using moving average
smoothed_data = data.rolling(window=window_size).mean().fillna(data)
print("Original data:")
print(data)
print("\nSmoothed data:")
print(smoothed_data)
```

We have a sample dataset data containing some numerical values.
We define a window size variable, which represents the size of the moving window for the moving average.
We perform data smoothing using the rolling() method in pandas, specifying the window size, and then calculate the mean within each window.

We fill any missing values in the smoothed data with the original data.
Finally, we print both the original and smoothed data for comparison.

Attribute Construction:

```
import pandas as pd
# Sample dataset
data = pd.DataFrame({
   'height': [160, 170, 180, 165, 175],
   'weight': [60, 70, 80, 55, 65]
})
# Construct new feature: Body Mass Index (BMI)
data['bmi'] = data['weight'] / ((data['height'] / 100) ** 2)
print(data)
```

Data Generalization:

```
import pandas as pd
# Sample dataset
data = pd.DataFrame({
   'age': [25, 30, 35, 40, 45, 50],
   'income': [50000, 60000, 70000, 80000, 90000, 100000]
})
# Generalize age into age groups
data['age_group'] = pd.cut(data['age'], bins=[20, 30, 40, 50], labels=['20-30',
'30-40', '40-50'])
print(data)
```

Data Aggregation:

```
import pandas as pd
# Sample dataset
data = pd.DataFrame({
   'city': ['New York', 'New York', 'Los Angeles', 'Los Angeles', 'Chicago',
'Chicago'],
   'temperature': [70, 72, 75, 73, 68, 70],
   'humidity': [50, 55, 60, 58, 45, 48]
})
# Aggregate data by city: calculate mean temperature and humidity
aggregated_data = data.groupby('city').agg({
   'temperature': 'mean',
   'humidity': 'mean'
}).reset_index()
print(aggregated_data)
```

d)Data Correction :

Error correction in datasets involves identifying and rectifying errors or inconsistencies to improve data quality. The methods are

1. Imputation: Missing values are filled in using various techniques such as mean, median, mode, or predictive modeling.
2. Outlier Detection and Removal: Outliers are identified and corrected using statistical methods like z-score, IQR, or machine learning models.
3. Data Validation Rules: Data is checked against predefined rules or constraints to detect and correct errors such as format mismatches, range violations, or referential integrity issues.

4. Data Cleaning Pipelines: Automated pipelines are constructed to clean and preprocess data, incorporating techniques like deduplication, normalization, and standardization.

5. Cross-Referencing and Validation: Data is cross-referenced with external sources or validated against known benchmarks to identify and rectify discrepancies.
6. Machine Learning Models: Advanced techniques such as anomaly detection, clustering, and classification are employed to detect and correct errors in data.

```
import pandas as pd
from sklearn.impute import SimpleImputer
# Sample dataset with missing values
data = pd.DataFrame({
    'age': [25, 30, None, 40, 45, None],
    'income': [50000, None, 70000, 80000, None, 100000]
})
# Initialize SimpleImputer with strategy='mean' to impute missing values with mean
imputer = SimpleImputer(strategy='mean')
# Fit and transform data to impute missing values
imputed_data = pd.DataFrame(imputer.fit_transform(data), columns=data.columns)
print("Original data:")
print(data)
print("\nImputed data:")
print(imputed_data)
```

In this example:
- We have a sample dataset `data` with missing values represented as `None`.
- We use `SimpleImputer` from scikit-learn to impute missing values with the mean of each column.
- The `fit_transform()` method is used to fit the imputer on the data and transform it to impute missing values.
- The resulting `imputed_data` DataFrame contains the original data with missing values replaced by the mean of each column.

2. Outlier Detection and Removal:
   - Example: Using Z-score for outlier detection and removal.

```
from scipy import stats
# Sample dataset with outliers
data = pd.DataFrame({'value': [10, 15, 20, 100, 25, 30]})
# Calculate z-scores
z_scores = stats.zscore(data['value'])
# Remove outliers based on z-score threshold (e.g., 3)
threshold = 3
data_no_outliers = data[(z_scores < threshold)]
print("Original data:")
print(data)
print("\nData without outliers:")
print(data_no_outliers)
```

3. Data Validation Rules:
   - Example: Checking for range violations in a numerical column.

```
 # Sample dataset
 data = pd.DataFrame({'age': [25, 30, 150, 40, 45]})
# Check for values outside the valid range (e.g., age between 0 and 120)
 invalid_data = data[(data['age'] < 0) | (data['age'] > 120)]

print("Invalid data:")
print(invalid_data)
```

4. Data Cleaning Pipelines:
   - Example: Constructing a data cleaning pipeline using scikit-learn's `Pipeline`.

```
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
# Sample dataset with missing values and numerical features
data = pd.DataFrame({'age': [25, 30, None, 40, 45], 'income': [50000, None, 70000, 80000, 100000]})
# Construct a data cleaning pipeline
pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='mean')),
    ('scaler', StandardScaler())
])
# Fit and transform data using the pipeline
cleaned_data = pipeline.fit_transform(data)
print("Cleaned data:")
print(cleaned_data)
```

5. Cross-Referencing and Validation:
   - Example: Cross-referencing data with an external source to detect discrepancies.

```
# Sample dataset
data = pd.DataFrame({'name': ['Alice', 'Bob', 'Charlie', 'David'],'age': [25, 30, 35, 40]})
# Cross-reference data with a list of known names
known_names = ['Alice', 'Bob', 'Eve']
unknown_names = data[~data['name'].isin(known_names)]
print("Unknown names:")
print(unknown_names)
```

6. Machine Learning Models:
   - Example: Using clustering for anomaly detection.

```
from sklearn.cluster import DBSCAN
# Sample dataset with numerical features
data = pd.DataFrame({'value': [10, 15, 20, 100, 25, 30]})
# Fit DBSCAN clustering model
dbscan = DBSCAN(eps=10, min_samples=2)
dbscan.fit(data)
# Identify outliers/anomalies based on cluster labels
outliers = data[dbscan.labels_ == -1]
print("Outliers:")
print(outliers)
```

Data Model Building : Heart diseases data set

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
# Load the dataset
heart_disease_data = pd.read_csv('heart_disease_dataset.csv')
# Select features (X) and target variable (y)
X = heart_disease_data.drop('target', axis=1)
y = heart_disease_data['target']
```

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Initialize the linear regression model
model = LinearRegression()
# Train the model on the training data
model.fit(X_train, y_train)
# Make predictions on the testing data
y_pred = model.predict(X_test)
# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)
```

Explanation:

- The data set is loaded then
- we separate the features (X) from the target variable (y).
- Then, we split the dataset into training and testing sets using `train_test_split()` from scikit-learn.
- We initialize a linear regression model using `LinearRegression()` from scikit-learn.
- The model is trained on the training data using the `fit()` method.
- After training, we use the trained model to make predictions on the testing data.
- Finally, we evaluate the model's performance using the mean squared error (MSE) metric, which quantifies the average squared difference between the predicted and actual target values.

This example demonstrates how to build a simple linear regression model to predict heart disease based on features available in the dataset. You can further enhance the model by performing feature engineering, hyperparameter tuning, and cross-validation for better performance.

Sample code

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
# Assuming 'air' is a DataFrame loaded from a CSV file
# Replace 'air.csv' with the actual file name
air = pd.read_csv('air.csv')
# Display first few rows of the DataFrame
print(air.head())
# Count non-null values in each column
print(air.count())
# Check for missing values in each column
print(air.isnull().sum())
# Display summary statistics of the DataFrame
print(air.describe())
# Display information about the DataFrame
print(air.info())
# Drop rows with missing values
A = air.dropna()
```

```
print(A.shape)
# Fill missing values with 0
A = air.fillna(0)
print(A.shape)
print(A.head())
# Fill missing values with previous value (pad)
A = air.fillna(method='pad')
print(A.head())
# Fill missing values with next value (backfill)
A = air.fillna(method='backfill')
print(A.head())
# Replace missing values in 'Ozone' column with mean
A['Ozone'] = air['Ozone'].replace(np.NaN, air['Ozone'].mean())
print(A.head())
# Replace missing values in 'Ozone' column with median
A['Ozone'] = air['Ozone'].replace(np.NaN, air['Ozone'].median())
print(A.head())
# Replace missing values in 'Ozone' column with mode
A['Ozone'] = air['Ozone'].replace(np.NaN, air['Ozone'].mode()[0])
print(A.head())
# Impute missing values using SimpleImputer
imp = SimpleImputer(missing_values=np.nan, strategy='mean')
A = imp.fit_transform(air)
A = pd.DataFrame(A, columns=air.columns)
print(A.head())
# Split data into training and testing sets
train, test = train_test_split(A, test_size=0.20)
print(len(train))
print(len(test))
# Train a linear regression model
X = A['Ozone'].values.reshape(-1, 1)
Y = A['Temp']
model = LinearRegression()
model.fit(X, Y)
# Evaluate the model
print(model.score(X, Y) * 100)
print(model.predict([[128]]))
# Plot data and regression line
plt.scatter(X, Y)
plt.plot(X, model.predict(X), color='red')
plt.show()
```

CONCLUSION: The operations studied and implemented using Python include data cleaning, integration, transformation, error correction, and model building using Air quality and heart disease data sets.

ASSIGNMENT 7

TITLE: Design a distributed application using MapReduce

OBJECTIVES:

1. To explore different Big data processing techniques with use cases.
2. To study detailed concept of Map-Reduce.

SOFTWARE REQUIREMENTS:

1 Windows 10/11
2 Java
3 Virtual Machine
4 Cloudera Virtual Machine

PROBLEM STATEMENT: - Design a distributed application using MapReduce which processes a log file of a system. List out the users who have logged for maximum period on the system. Use simple log file from the Internet and process it using a pseudo distribution mode on Hadoop platform.

THEORY:

Introduction

MapReduce is a framework using which we can write applications to process huge amounts of data, in parallel, on large clusters of commodity hardware in a reliable manner. MapReduce is a processing technique and a program model for distributed computing based on java.

The MapReduce algorithm contains two important tasks, namely Map and Reduce.Map takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs).

Secondly, reduce task, which takes the output from a map as an input and combines those data tuples into a smaller set of tuples. As the sequence of the name MapReduce implies, the reduce task is always performed after the map job.

The major advantage of MapReduce is that it is easy to scale data processing over multiple computing nodes.

Under the MapReduce model, the data processing primitives are called mappers and reducers. Decomposing a data processing application into mappers and reducers is sometimes nontrivial. But, once we write an application in the MapReduce form, scaling the application to run over hundreds, thousands, or even tens of thousands of machines in a cluster is merely a configuration change. This simple scalability is what has attracted many programmers to use the MapReduce model.

MapReduce

MapReduce program executes in three stages, namely map stage, shuffle stage, and reduce stage.
Mapstage: The map or mapper's job is to process the input data. Generally, the input data is in the form of file or directory and is stored in the Hadoop file system (HDFS). The input file is passed to the mapper function line by line. The mapper processes the data and creates several small chunks of data.
Reduce stage: This stage is the combination of the Shuffle stage and the Reduce stage. The Reducer's job is to process the data that comes from the mapper. After processing, it produces a new set of output, which will be stored in the HDFS.

Inserting Data into HDFS:

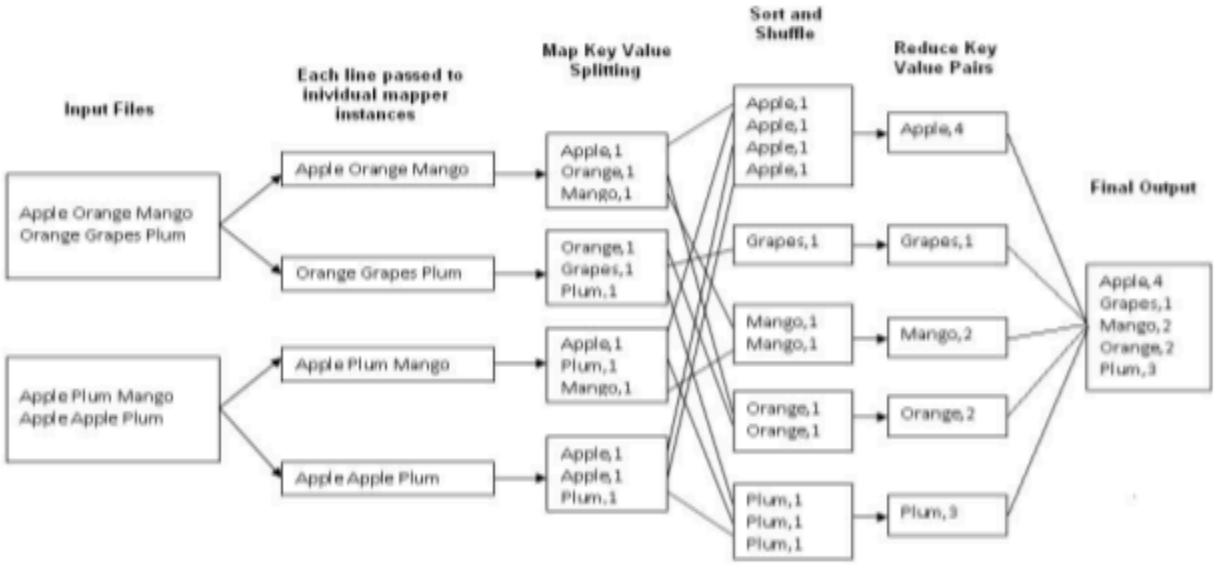


Fig.1 : An Example Program to Understand working of MapReduce Program.

- The MapReduce framework operates on <key, value> pairs, that is, the framework views the input to the job as a set of <key, value> pairs and produces a set of <key, value> pairs as the output of the job, conceivably of different types.
- The key and the value classes should be in serialized manner by the framework and hence, need to implement the Writable interface. Additionally, the key classes have to implement the Writable- Comparable interface to facilitate sorting by the framework.
- Input and Output types of a MapReduce job: (Input) <k1,v1> -> map -><k2, v2>-> reduce -><k3, v3> (Output).

Steps for Compilation & Execution of Program:

Open Eclipse IDE
Create Java Project
Create Java Package
Create Java Class file
Write source code
Add some Libraries to java project
Right click on Java Project > Click Build Path > Add External Archives
    Select path usr/lib/hadoop/hadooop_comman.jar
    Select path usr/lib/hadoop-0.20/hadoop-core-2.6.0-mr1-cdh5.4.2.jar
Right click on Java Project >Click Export Project> Java>JAR file
    Browse and select file name of jar file
    Click next
    Click finish
Create input.txt file on desktop
Copy from desktop to present working directory
    $ hadoop fs –put source path destination path
        for e.g $ hadoop fs –put Desktop/input.txt input.txt
Run a jar file
    $hadoop jar JarFileName.jar Packagename.ClassName inputfile.txt OutputDirectory For e.g $ hadoop jar WordCount.jar WordCount.WordCount input.txt dir
Check output
    $ hadoop dfs –cat Output Directory/part-r-00000

CONCLUSION: Studied how to design a distributed application using MapReduce and process a log file of a system and implemented the same.

ASSIGNMENT 8

TITLE: Perform the following data visualization operations using Tableau on Adult and Iris datasets

OBJECTIVES:

1. To understand and apply the Analytical concept of Big data using Tableau..
2. To Visualize data using Tableau

SOFTWARE REQUIREMENTS:

2. Windows 10 or above
3. Tableau

PROBLEM STATEMENT: Perform the following data visualization operations using Tableau on Adult and Iris datasets

1) 1D (Linear) Data visualization
2) 2D (Planar) Data Visualization
3) 3D (Volumetric) Data Visualization
4) Temporal Data Visualization
5) Multidimensional Data Visualization
6) Tree/ Hierarchical Data visualization
7) Network Data visualization

THEORY:

Introduction

Data visualization or data visualization is viewed by many disciplines as a modern equivalent of visual communication. It involves the creation and study of the visual representation of data, meaning "information that has been abstracted in some schematic form, including attributes or variables for the units of information".

Data visualization refers to the techniques used to communicate data or information by encoding it as visual objects (e.g., points, lines or bars) contained in graphics. The goal is to communicate information clearly and efficiently to users. It is one of the steps in data analysis or data science

1D/Linear

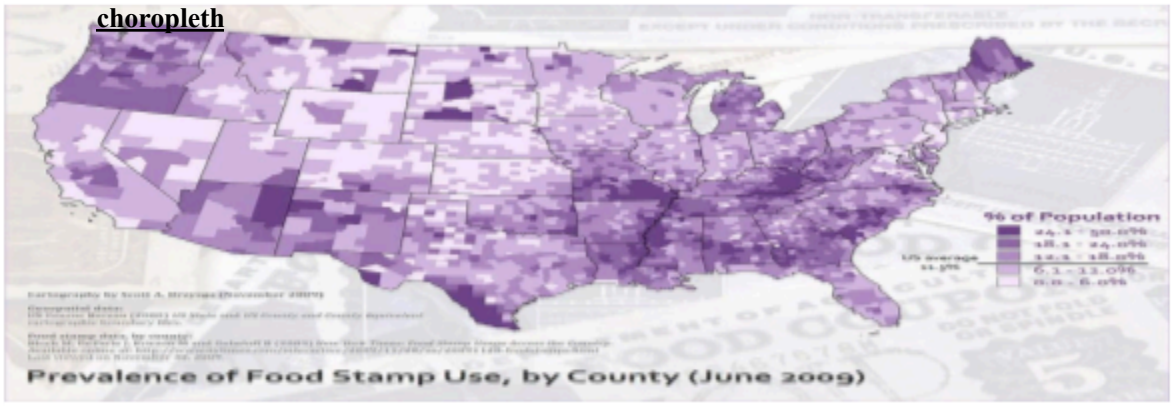
Examples:

lists of data items, organized by a single feature (e.g., alphabetical order) (not commonly visualized)

2D/Planar (especially geospatial)

Examples (geospatial):

choropleth



3D/ Volumetric

3D/Volumetric

Broadly, examples of scientific visualization:

3D computer models

In 3D computer graphics, **3D modeling** (or **three-dimensional modeling**) is the process of developing a mathematical representation of any *surface* of an object (either inanimate or living) in three dimensions via specialized software. The product is called a **3D model**. Someone who works with 3D models may be referred to as a **3D artist**. It can be displayed as a two-dimensional image through a process called *3D rendering* or used in a computer simulation of physical phenomena. The model can also be physically created using 3D printing devices.

Surface and volume rendering

Rendering is the process of generating an image from a model, by means of computer programs. The model is a description of three-dimensional objects in a strictly defined language or data structure. It would contain geometry, viewpoint, texture, lighting, and shading information. The image is a digital image or raster graphics image. The term may be by analogy with an "artist's

rendering" of a scene. 'Rendering' is also used to describe the process of calculating effects in a video editing file to produce final video output.

Volume rendering is a technique used to display a 2D projection of a 3D discretely sampled data set. A typical 3D data set is a group of 2D slice images acquired by a CT or MRI scanner. Usually these are acquired in a regular pattern (e.g., one slice every millimeter) and usually have a regular number of image pixels in a regular pattern. This is an example of a regular volumetric grid, with each volume element, or voxel represented by a single value that is obtained by sampling the immediate area surrounding the voxel.

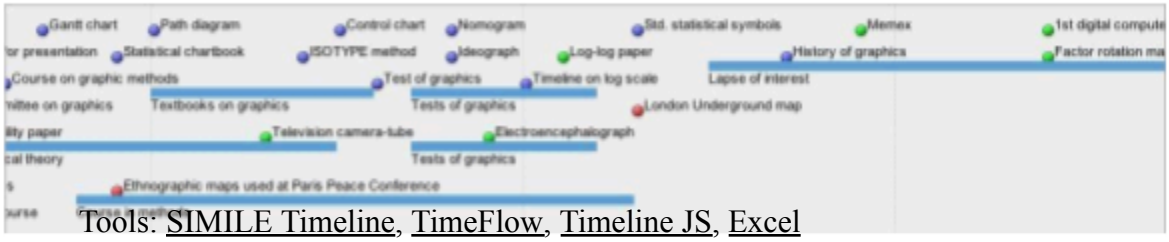
computer simulations

Computer simulation is a computer program, or network of computers, that attempts to simulate an abstract model of a particular system. Computer simulations have become a useful part of mathematical modeling of many natural systems in physics, and computational physics, chemistry and biology; human systems in economics, psychology, and social science; and in the process of engineering and new technology, to gain insight into the operation of those systems, or to observe their behavior.[6] The simultaneous visualization and simulation of a system is called visulation.

Temporal

Examples:

timeline

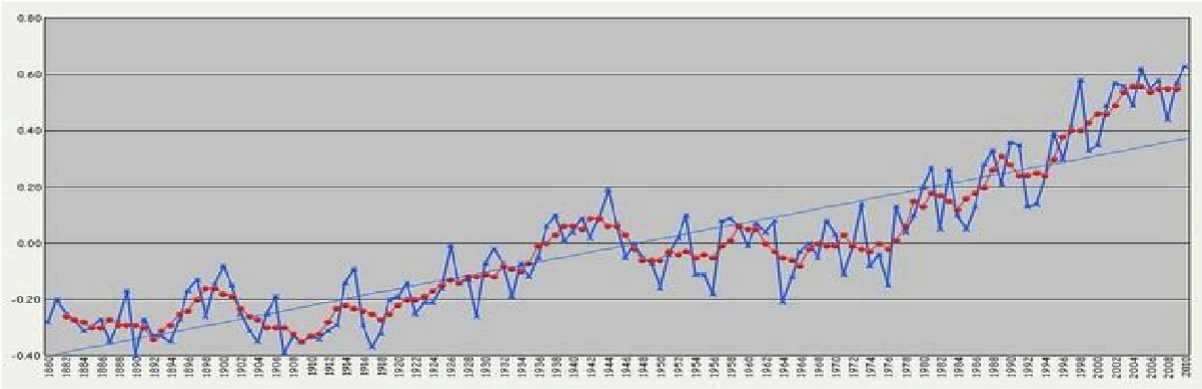


Tools: SIMILE Timeline, TimeFlow, Timeline JS, Excel

Image:
Friendly, M. & Denis, D. J. (2001). Milestones in the history of thematic cartography,

statistical graphics, and data visualization.Web document, http://www.datavis.ca/milestones/.Accessed: August 30, 2012.

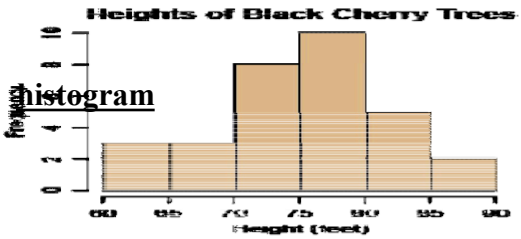
time series



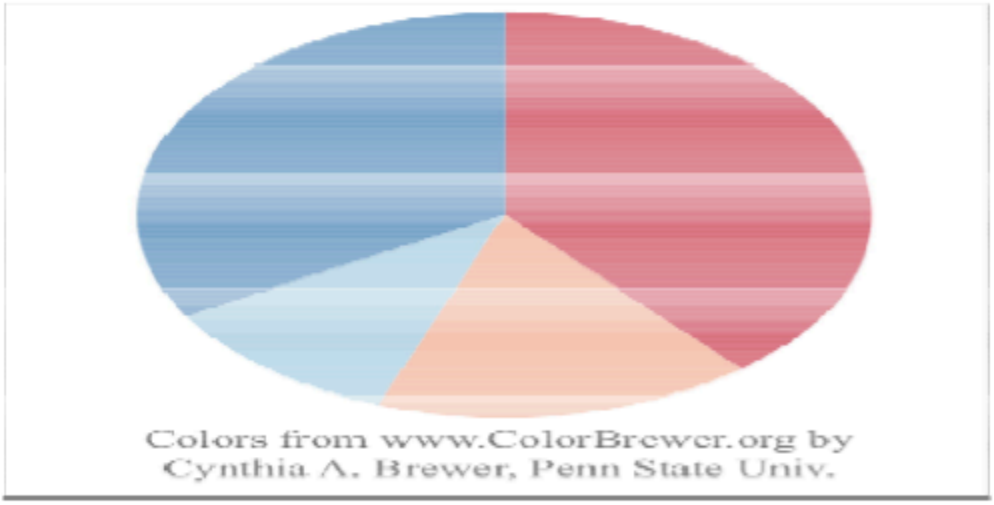
nD/ Multidimensional

Examples (category proportions, counts):

histogram



pie chart

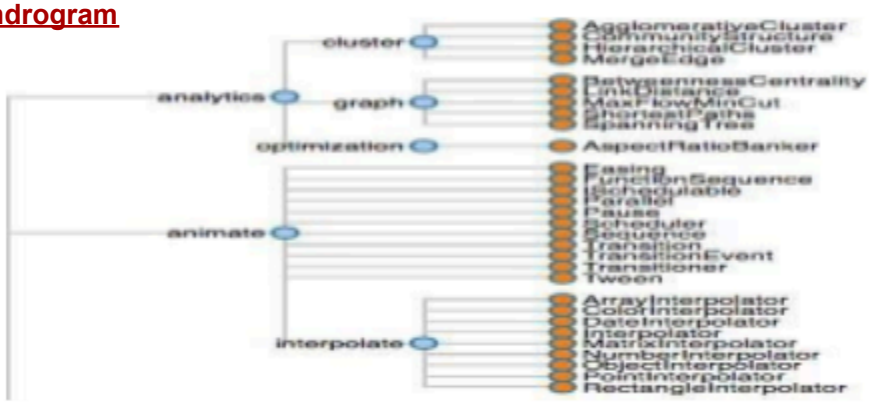


Tree/Hierarchical

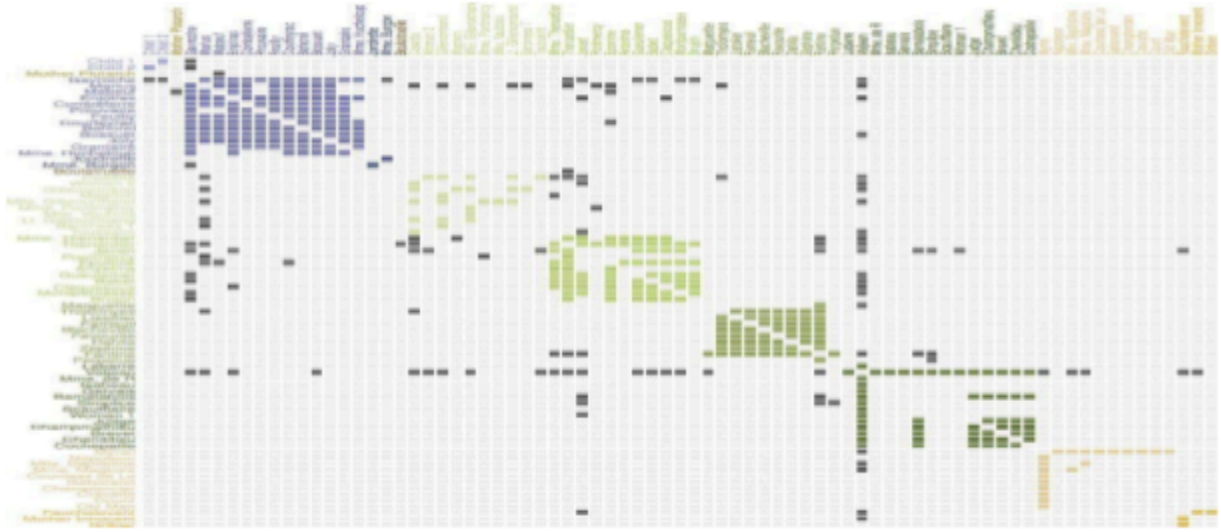
Examples:

general tree visualization

dendrogram



metrix



Network

- **node-link diagram (link-based layout algorithm)**

Tableau:

Tableau is a Business Intelligence tool for visually analyzing the data. Users can create and distribute an interactive and shareable dashboard, which depict the trends, variations, and density of the data in the form of graphs and charts. Tableau can connect to files, relational and Big Data sources to acquire and process data. The software allows data blending and real-time collaboration, which makes it very unique. It is used by businesses, academic researchers, and many government

organizations for visual data analysis. It is also positioned as a leader Business Intelligence and Analytics Platform in Gartner Magic Quadrant.

Tableau Features:

Tableau provides solutions for all kinds of industries, departments, and data environments. Following are some unique features which enable Tableau to handle diverse scenarios.

- **Speed of Analysis** – As it does not require high level of programming expertise, any user with access to data can start using it to derive value from the data.
- **Self-Reliant** – Tableau does not need a complex software setup. The desktop version which is used by most users is easily installed and contains all the features needed to start and complete data analysis.
- **Visual Discovery** – The user explores and analyzes the data by using visual tools like colors, trend lines, charts, and graphs. There is very little script to be written as nearly everything is done by drag and drop.
- **Blend Diverse Data Sets** – Tableau allows you to blend different relational, semi structured and raw data sources in real time, without expensive up-front integration costs. The users don't need to know the details of how data is stored.
- **Architecture Agnostic** – Tableau works in all kinds of devices where data flows. Hence, the user need not worry about specific hardware or software requirements to use Tableau.
- **Real-Time Collaboration** – Tableau can filter, sort, and discuss data on the fly and embed a live dashboard in portals like SharePoint site or Salesforce. You can save your view of data and allow colleagues to subscribe to your interactive dashboards so they see the very latest data just by refreshing their web browser.
- **Centralized Data** – Tableau server provides a centralized location to manage all of the organization's published data sources. You can delete, change permissions, add tags, and manage schedules in one convenient location. It's easy to schedule extract refreshes and manage them in the data server. Administrators can centrally define a schedule for extracts on the server for both incremental and full refreshes.

There are three basic steps involved in creating any Tableau data analysis report.

These three steps are –

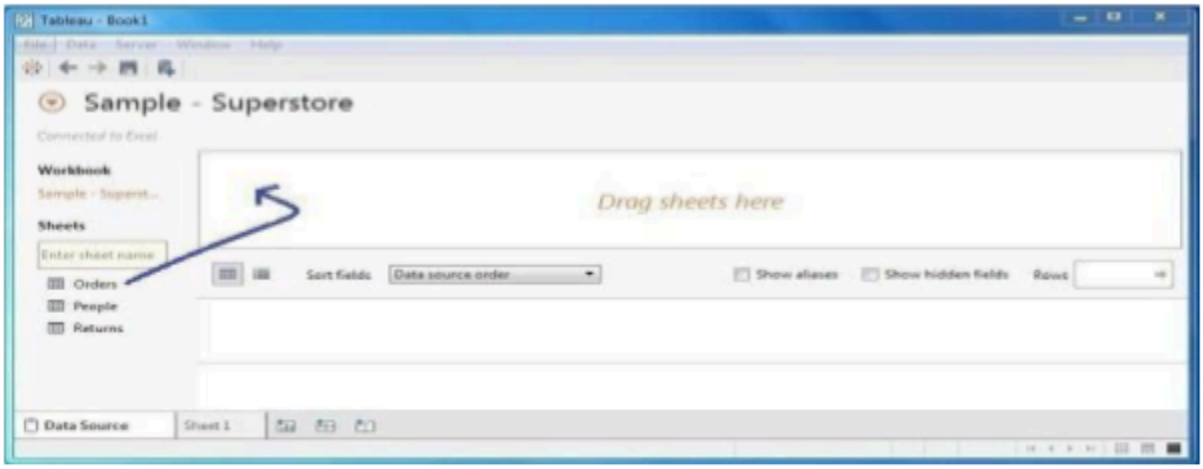
- **Connect to a data source** – It involves locating the data and using an appropriate type of connection to read the data.

- **Choose dimensions and measures** – This involves selecting the required columns from the source data for analysis.
- **Apply visualization technique** – This involves applying required visualization methods, such as a specific chart or graph type to the data being analyzed.

For convenience, let's use the sample data set that comes with Tableau installation named sample – superstore.xls. Locate the installation folder of Tableau and go to **My Tableau Repository**. Under it, you will find the above file at **Datasources\9.2\en_US-US**.

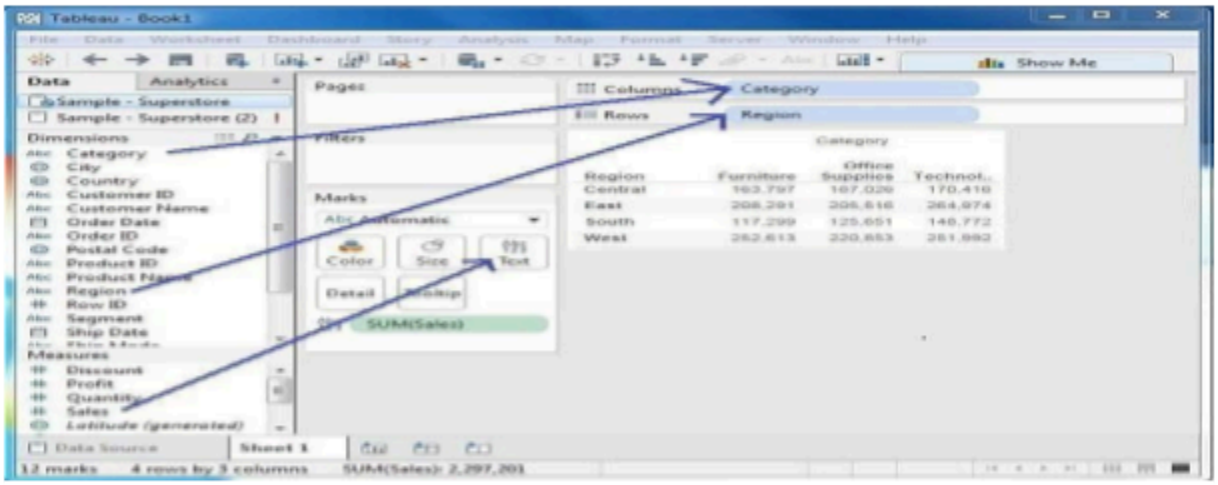
Connect to a Data Sour ce

On opening Tableau, you will get the start page showing various data sources. Under the header **"Connect"**, you have options to choose a file or server or saved data source. Under Files, choose excel. Then navigate to the file **"Sample – Superstore.xls"** as mentioned above. The excel file has three sheets named Orders, People and Returns. Choose **Orders**.



Choose the Dimensions and Measures

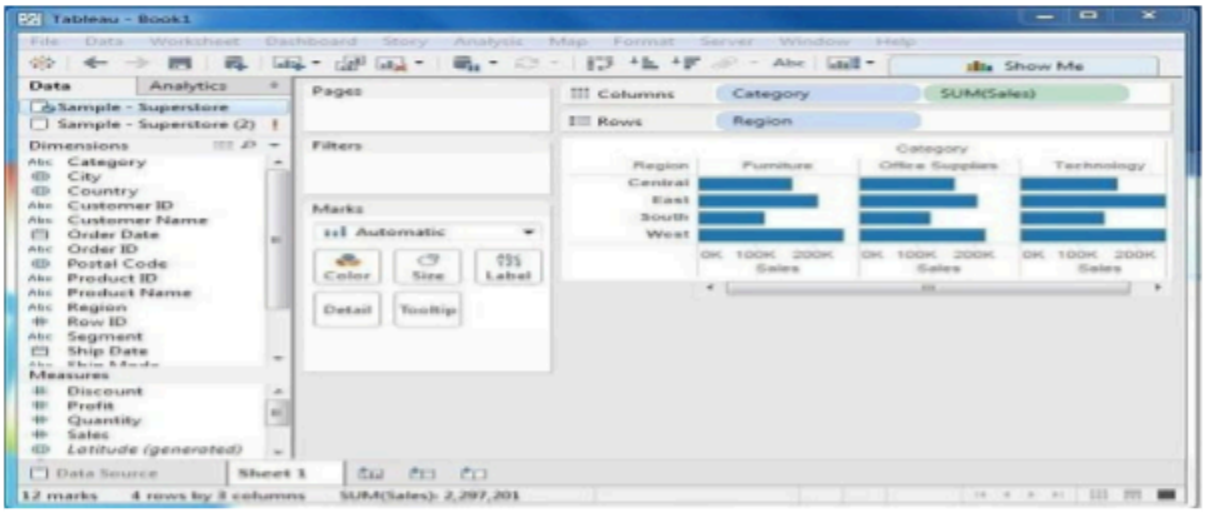
Next, choose the data to be analyzed by deciding on the dimensions and measures. Dimensions are the descriptive data while measures are numeric data. When put together, they help visualize the performance of the dimensional data with respect to the data which are measures. Choose **Category** and **Region** as the dimensions and **Sales** as the measure. Drag and drop them as shown in the following screenshot. The result shows the total sales in each category for each region.



Apply Visualization Technique

In the previous step, you can see that the data is available only as numbers. You have to read and calculate each of the values to judge the performance. However, you can see them as graphs or charts with different colors to make a quicker judgment.

We drag and drop the sum (sales) column from the Marks tab to the Columns shelf. The table showing the numeric values of sales now turns into a bar chart automatically.



You can apply a technique of adding another dimension to the existing data. This will add more colors to the existing bar chart as shown in the following screenshot.

Conclusion: Learned how to Visualize the data using different types of visualization tools (1 1D (Linear) Data visualization,2D (Planar) Data Visualization, 3D (Volumetric) Data Visualization, Temporal Data Visualization, Multidimensional Data Visualization, Tree/Hierarchical Data visualization, Network Data visualization) by using Tableau Software.