# Index

# Practical no.1

# Data Science Introduction & Basics

**Aim 1A):** Explore existing Packages, APIs, Data Sets and Models, Explore Jupyter Notebook, Explore Google Collab for Python/R Software used:

## Explore Existing Python Packages

### 1.Numpy:

NumPy is a very important Python package for working with numbers. It helps to handle large lists of numbers, called arrays, very easily. You can do mathematical calculations like addition, subtraction, multiplication, division, and even more complicated operations. NumPy is very fast compared to normal Python lists, so it is used when we have a lot of data. Scientists, engineers, and students use it for experiments, research, and data analysis. It also works as a base for many other Python packages like Pandas and SciPy. Learning NumPy makes working with data much easier.

### 2.Pandas:

Pandas is used to work with structured data, like tables with rows and columns. It allows you to clean messy data, organize it, and analyze it to find patterns and useful information. You can sort data, filter it, or calculate statistics like averages and sums easily. Pandas is very popular in schools, businesses, and research projects because it makes big data easy to understand. It can read data from Excel files, CSV files, and databases. It also works well with other packages like Matplotlib and Seaborn to visualize data.

### 3. Matplotlib:

Matplotlib is a Python package used to create graphs and charts from data. It can make line graphs, bar charts, scatter plots, histograms, and more. It helps people see patterns and trends in the data visually instead of just looking at numbers. Scientists, students, and businesses use it to explain information clearly. It can be customized to change colors, labels, and styles. Although it is simple, it is very powerful for understanding data.

### 4.Seaborn:

Seaborn is built on Matplotlib and helps make graphs look better and easier to read. It is very useful for showing statistical relationships between data points. With Seaborn, you can make heatmaps, correlation plots, and distribution charts quickly. It is widely used in data analysis and research to make results look professional. Seaborn also helps to compare data between different groups. It makes analysing complex data much easier for students and professionals.

### 5. Scikit-learn:

Scikit-learn is a popular Python package for machine learning. It helps computers learn from data and make predictions. You can use it for tasks like classifying data, predicting numbers, or grouping similar information. For example, it can predict weather, stock prices, or student results. Scikit-learn is easy to use and does not require writing complex algorithms. It is widely used in schools, companies, and research projects. It also works well with Pandas and NumPy.

## 6. TensorFlow:

TensorFlow is a Python package for building artificial intelligence (AI) models. It can help computers recognize images, understand text, and even speak like humans. It is used in self-driving cars, chatbots, voice assistants, and other AI projects. TensorFlow allows both beginners and experts to create powerful AI applications. It can handle large amounts of data and train complex models. It is supported by Google and widely used around the world.

## 7. PyTorch:

PyTorch is another Python package for AI and deep learning. It is easy to learn and allows users to build smart models quickly. Researchers use PyTorch to create neural networks that can understand images, text, or other data. It is very flexible, which makes it ideal for testing new ideas. PyTorch is also widely used in AI research and industry projects. Many AI companies prefer PyTorch because of its speed and simplicity.

## 8. Requests:

Requests is a Python package that allows you to get data from websites or online services. It is very simple and helps to collect information from the internet. For example, you can get weather information, stock prices, or news articles from websites. Requests is important for building programs that need online data. It is easy to use and works with most websites. Beginners use it to practice working with APIs and real-world data.

## 9. OpenCV:

OpenCV is a Python package used for working with images and videos. It can detect faces, recognize objects, and track movement in videos. OpenCV is used in security systems, robotics, AI projects, and video processing. It is very powerful and helps computers “see” and understand images. Beginners can start with simple tasks like reading and displaying images, and later move to advanced tasks like object detection. OpenCV is widely used in schools, universities, and industries.

## 10. Plotly:

Plotly is a Python package used to make interactive and colorful graphs. Unlike normal charts, you can zoom in, hover over points, and explore the data in real-time. It is very helpful for showing data clearly in presentations, reports, or websites. Plotly can create line charts, bar graphs, scatter plots, maps, and even 3D graphs. Students and professionals use it to understand trends and patterns easily. It also works well with Pandas and NumPy to visualize large datasets. With Plotly, data analysis becomes more fun and engaging because the graphs are interactive.

# Application Programming Interfaces(APIs)

## 1. Requests API:

Requests allows Python programs to get or send information from websites. You can collect news, weather, or stock data automatically. It is simple and beginner-friendly. It is widely used in research, school projects, and apps that need live online data.

### 2. Tweepy API:
Requests allows Python programs to get or send information from websites. You can collect news, weather, or stock data automatically. It is simple and beginner-friendly. It is widely used in research, school projects, and apps that need live online data.

### 3. Google Maps API:
Requests allows Python programs to get or send information from websites. You can collect news, weather, or stock data automatically. It is simple and beginner-friendly. It is widely used in research, school projects, and apps that need live online data.

### 4. OpenWeatherMap API:
Requests allows Python programs to get or send information from websites. You can collect news, weather, or stock data automatically. It is simple and beginner-friendly. It is widely used in research, school projects, and apps that need live online data.

### 5. YouTube Data API:
Requests allow Python programs to get or send information from websites. You can collect news, weather, or stock data automatically. It is simple and beginner-friendly. It is widely used in research, school projects, and apps that need live online data.

## Datasets and Models

### 1. Iris Dataset & Decision Tree:
The Iris dataset contains information about different types of iris flowers, like petal and sepal size. Using this data, a Decision Tree model can classify which type of flower a new sample belongs to. The tree works like a flowchart, splitting data step by step to make predictions. This combination is widely used to learn basic classification in machine learning. Students find it simple and easy to understand.

### 2. Titanic Dataset & Logistic Regression:
The Titanic dataset has details about passengers, such as age, gender, and ticket class. Logistic Regression can use this data to predict whether a passenger survived or not. This model is perfect for beginners because it shows how data can be used to make yes/no predictions. It is widely used in learning basic machine learning and classification tasks.

### 3. MNIST Dataset & Neural Networks:
MNIST is a dataset of handwritten digits from 0 to 9. Neural Networks, a type of deep learning model, can be trained to recognize these digits. This combination is popular for learning image recognition. Students use it to understand how computers can “see” and identify patterns in images.

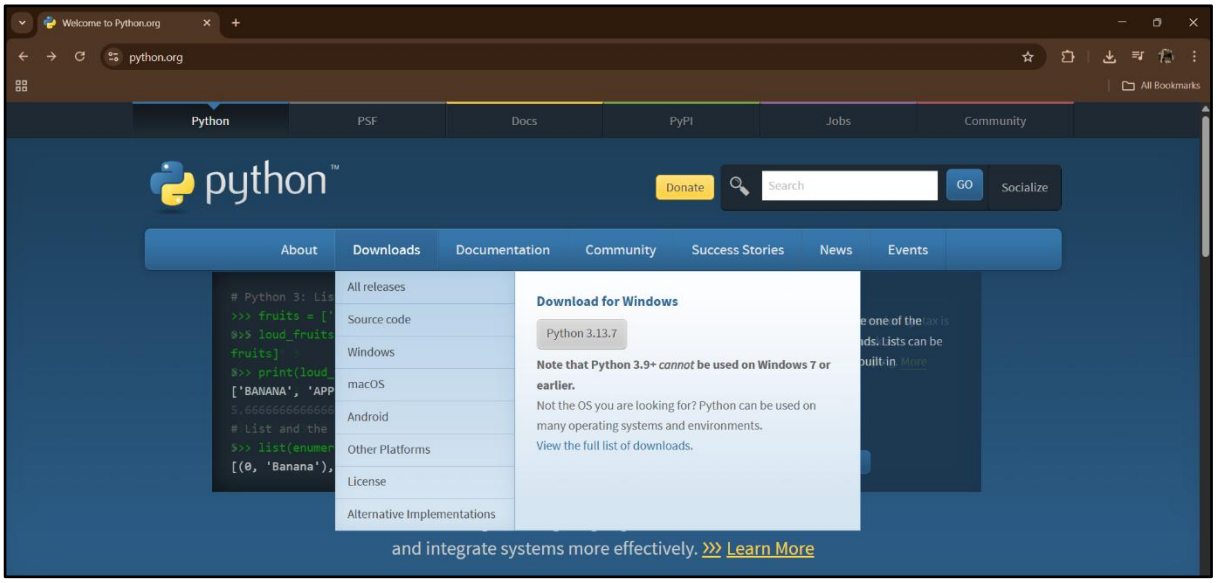
### 4. Boston Housing Dataset & Linear Regression:
The Boston Housing dataset has information about houses, like area, number of rooms, and price. Linear Regression can predict house prices based on these features. It is simple and helps students learn how to predict continuous values from data. This combination is often used in beginner projects for real-world applications.

# Explore Jupyter Notebook and Google Collab for Python Software

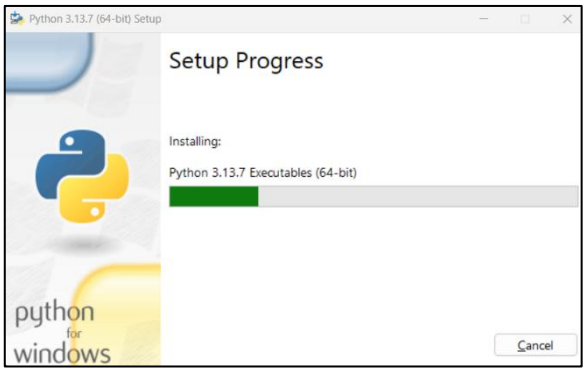
## 1. Installation of Jupyter Notebook

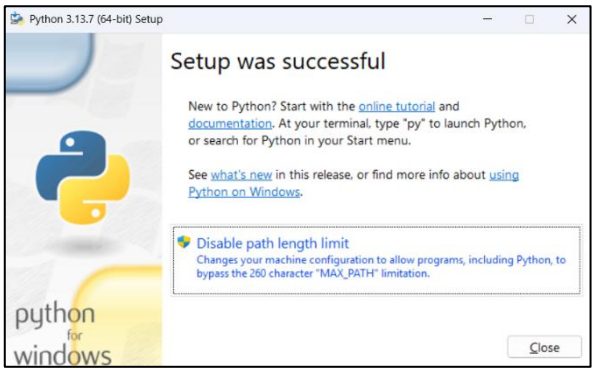
### Step 1: Install Python

- Go to official Python website: https://www.python.org/downloads/
- Click on the “Downloads” menu and choose your operating system (Windows/macOS/Linux).
- Download the latest stable version of Python.
- Run the installer. **Important:** Check the box **“Add Python to PATH”** before installing.
- Click **Install Now** and wait for the installation to complete.





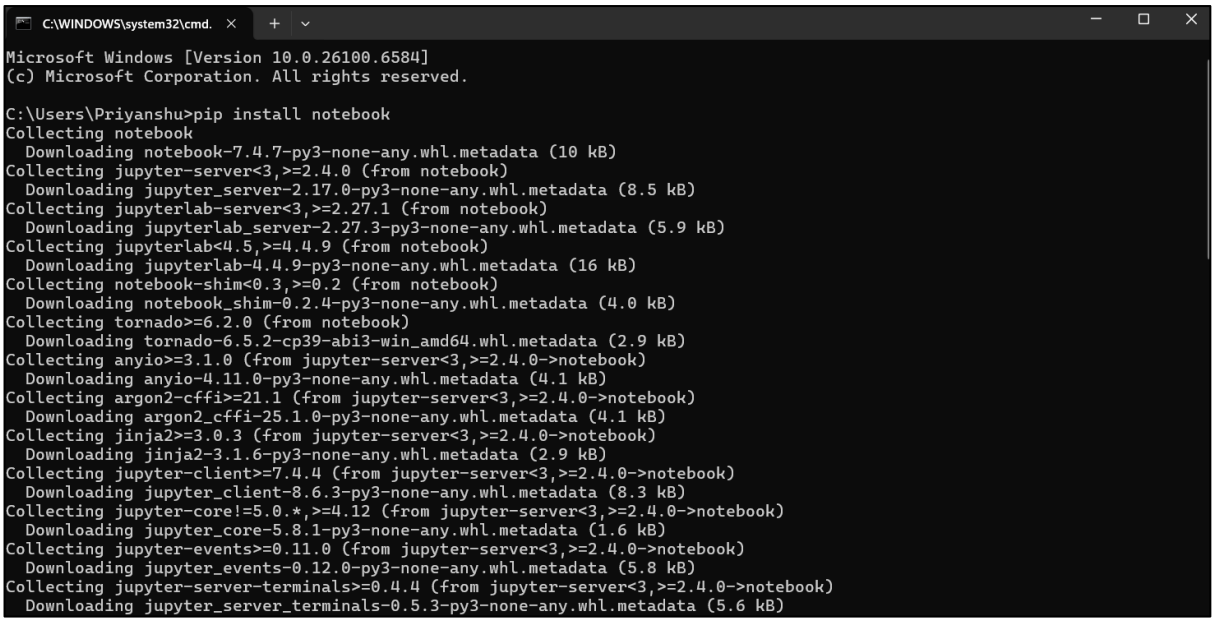


- Open Command Prompt (Windows) or Terminal (macOS/Linux) and Type:

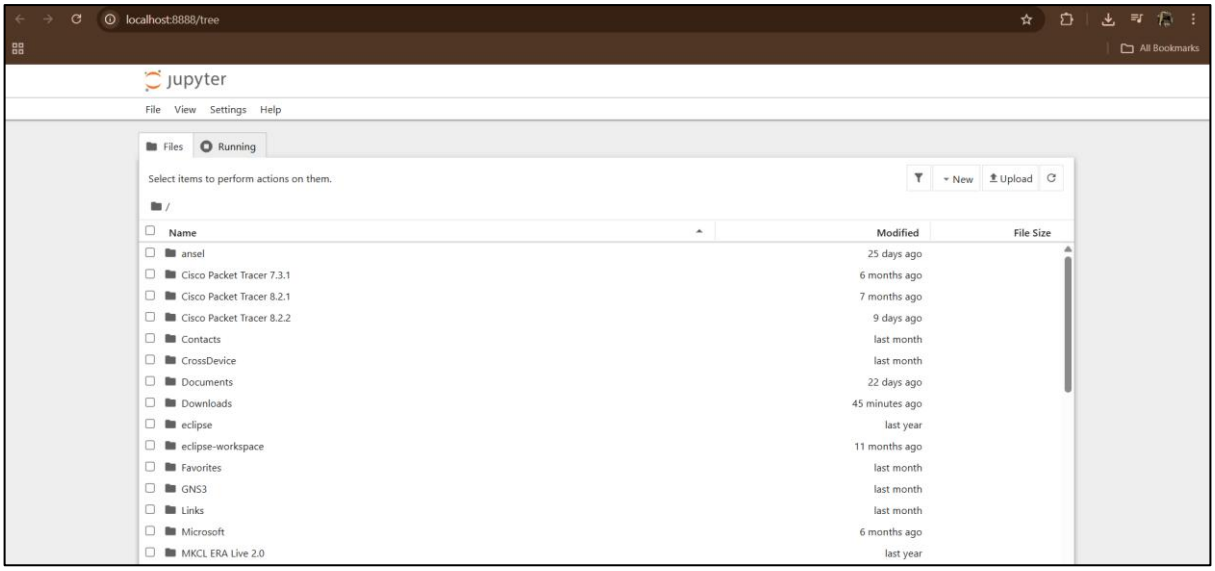

### Step 2: Install Jupyter Notebook

- After the Python Installation. Now Install
- Open Command Prompt (Windows) or Terminal (macOS/Linux)
- Type this Command (pip install notebook)
- Press Enter and wait for the installation to finish. You will see messages indicating the progress.
- Once installed successfully, the terminal will show **"Successfully installed notebook"**.



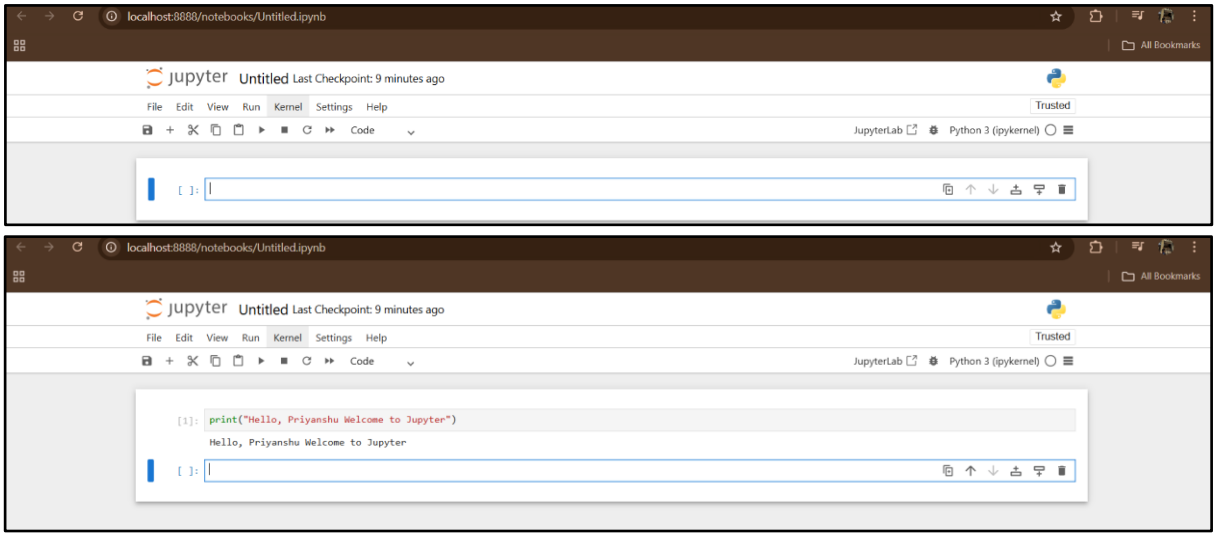

### Step 3: Launch Jupyter Notebook

- In the Command Prompt and Terminal type **"jupyter notebook"**
- Press Enter. This will start a local server, and your default web browser will open the **Jupyter Notebook Dashboard**.
- The dashboard shows your current folders and files. You can open existing notebooks or create a new one.
- Default URL: http://localhost:8888




**Step 4: Create and Run a Notebook**

- Click New → Python 3 to create a blank notebook.
- A notebook interface opens with empty cells. Each cell is a space where you can write Python code.
- Type the following code in the cell: **print("Hello, Priyanshu Welcome to Jupyter")**
- Press **Shift + Enter** to run the cell. The output will appear just below the cell.
- You can add multiple code cells by clicking **Insert → Insert Cell Below** or using the **+ button**.




## Conclusion:

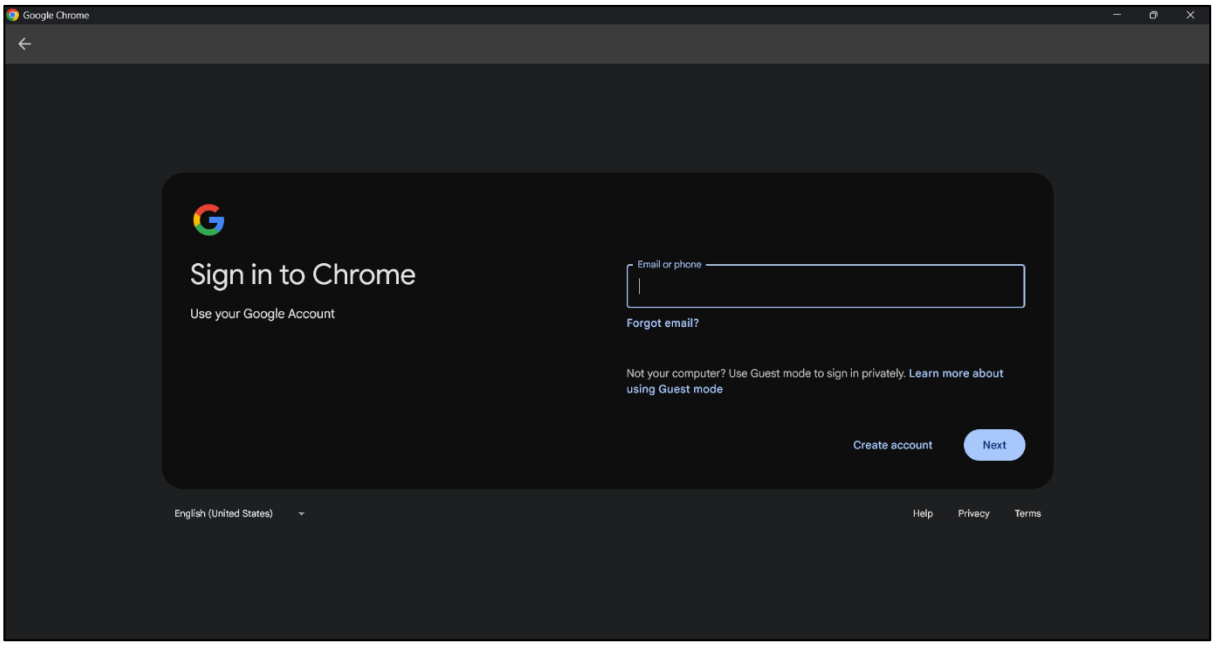
The installation of **Python** and **Jupyter Notebook** is a simple but very important step for anyone starting with programming or data science. While installing Python, it is best to check the option **"Add Python to PATH"** so that Python commands can be used from any folder without extra setup. After Python is ready, Jupyter Notebook can be installed easily using the command **pip install notebook**, which uses Python's package manager to download all the required files. Once the installation is complete, you can start Jupyter by typing **jupyter**

**notebook** in the command prompt or terminal. This opens an interactive workspace in your web browser where you can write, run, and save Python code. Following these steps ensures that your computer is fully prepared for coding, data analysis, and creating projects in a smooth and organized way.

## 2. Installation of Google Collab

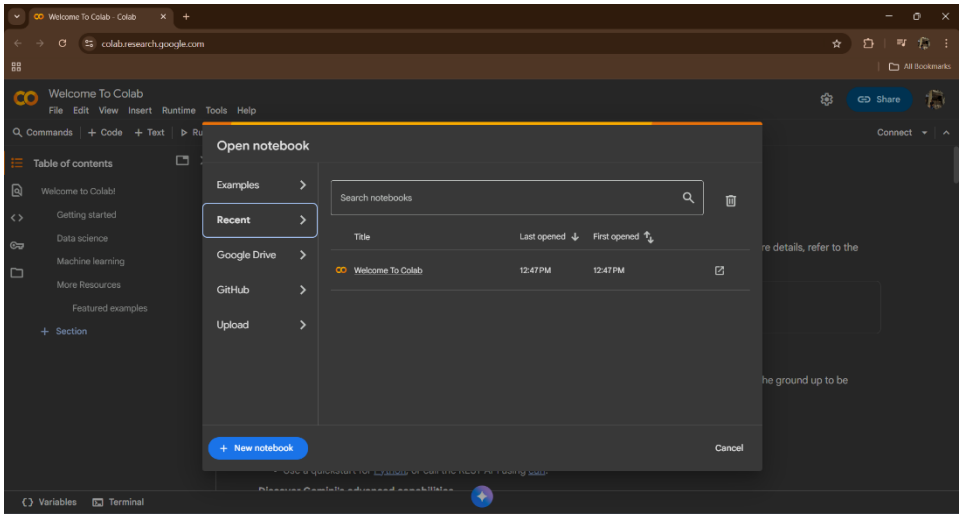
### Step 1: Sign in to Google

- Google Collab requires a Google account.
- Open any browser and go to https://accounts.google.com/.
- Enter your Gmail ID and password to sign in or create a new account if you don't have one.



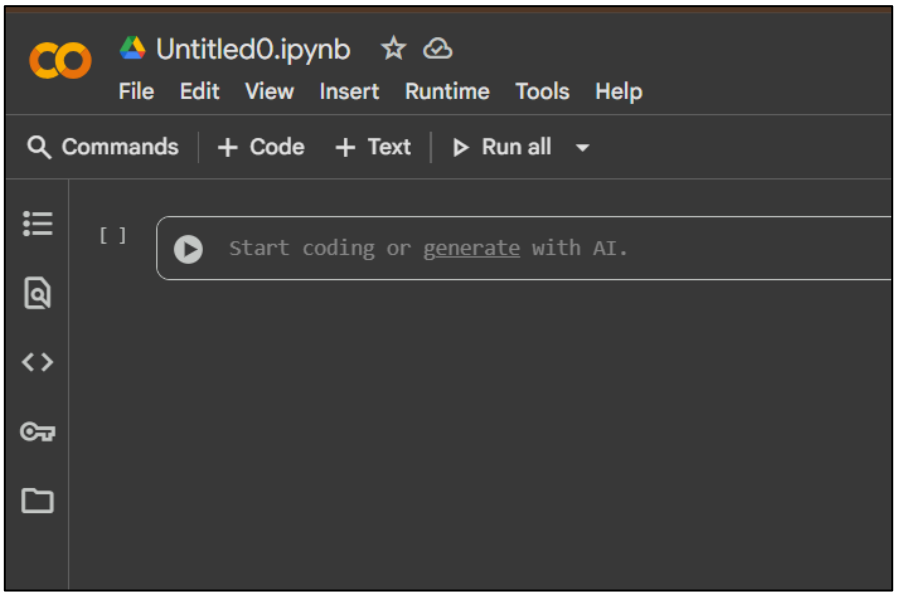

### Step 2: Open Google Collab

- In the browser, type **https://colab.research.google.com.** and press Enter.
- The Collab welcome page will appear with an option to create or open notebooks.
    - Alternative: From **Google Drive → New → More → Google Collaboratory**.



**Step 3: Create a New Notebook**

- Click **File →New Notebook** (bottom-right of the welcome box).
- A blank notebook will open with a default name like Untitled0.ipynb.
- Click the title to rename the file, e.g., MyFirstNotebook.
- When your new notebook opens, you'll see a blank **code cell** with a small in [ ]: label on the left.
- Above it is the **menu bar** (File, Edit, View, etc.) and a toolbar with buttons like **+Code** and **+Text** to add more cells.

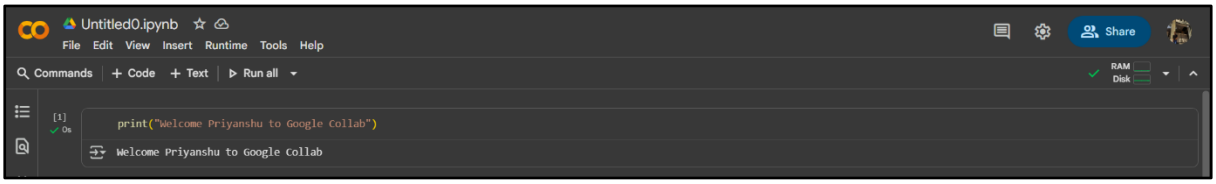



**Step 4: Connect to a Runtime**

- Before running Python, Collab must connect to a Virtual Machine.
- Click **"Connect"** (top-right Corner).
- After a few seconds it will change to **"Connected"**, meaning you can now execute Python code.

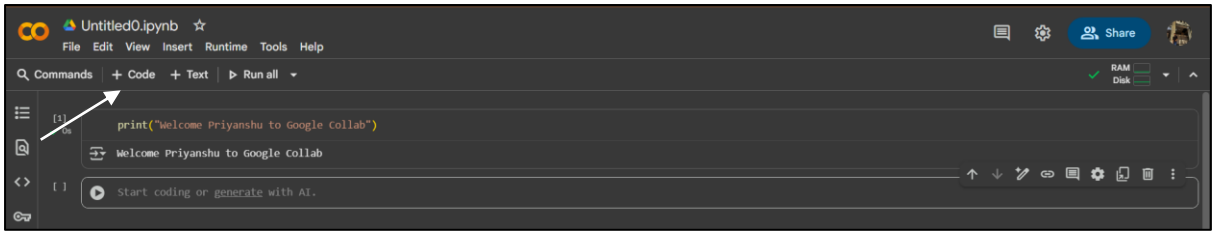
**Step 5: Write and Run Your First Python Code**

- Click inside the code cell and type: **print("Welcome Priyanshu to Google Collab")**
- Run the cell by pressing **Shift + Enter** or by clicking the **Run** button on the left of the cell.
- The output will appear just below the cell.




**Step 6: Add More Code Cell**

- To Keep code organized, click +Code at the top-left to add new cells.
- You can write different pieces of Python code in each cell and run them separately.




## Conclusion:

Google Collab is a powerful and beginner-friendly platform for writing and running Python code directly in a web browser. It removes the need to install heavy software on your computer and provides free access to computing resources like CPUs, GPUs, and TPUs, which are especially helpful for data science and machine learning projects. Because it is connected to Google Drive, all work is saved automatically, making it easy to share notebooks or collaborate with classmates and teammates in real time. With simple steps to create notebooks, run Python code, install packages, and download results, Google Collab offers an easy and flexible environment for both learning and professional projects.

# Practical no.1

# Data Science Introduction & Basics

**Aim 1B):** Study Data Science Methodology-Problem to approach, Requirements to collection, Understanding to Preparation, Modelling to Evaluation, Deployment to Feedback.

## Case Study: Cyberattack on an Organization: Response and Lessons Learned

## 1. Problem to Approach

In early 2025, a mid-sized financial company faced a big problem. Hackers attacked their computers with ransomware, which locked important customer and business files. The IT team and managers got very worried. If they didn't act quickly, they could lose money, leak customer data, and lose the trust of their clients.

The IT and security teams decided to handle the problem carefully. They did not want to just turn the computers back on. Instead, they wanted to understand what happened, find weak points, and make sure it does not happen again. They used both technology and human help, like talking to employees and checking how people used the systems, to find out the cause and the effect of the attack.

The first step was to clearly define the problem:

- Which systems were attacked?
- How did the hackers get in?
- How much damage was done to data and business operations?

## 2. Requirements to Collection

Next, the team needed data to understand the attack. They wanted to know how it happened, which systems were affected, and how serious the damage was. They collected:

- Logs from computers and servers
- Network and firewall data
- Employee login and access records
- Backup files

**Data Collection Process:**

The IT team checked logs from all computers to see how the malware spread. They also studied network activity to find any unusual connections. Employee activities were reviewed to see if someone accidentally clicked on a dangerous link. Backup files were checked to make sure data could be restored.

Collecting this information carefully helped the organization find out everything about the attack and prepare for recovery.

## 3. Understanding to Preparation

After collecting data, the team studied it to understand the attack. They found that it started from a phishing email sent to a senior employee. The employee clicked a link in the email, and malware spread across the network, locking many important files.

**Key Findings:**

- The attack started because of a phishing email — human error played a role.
- Some old systems were weak and easy to attack.
- Important files were affected, but backups were available.

**Preparation for Response:**

The organization created a safe testing area (sandbox) to try removing the malware without causing more damage. Backup systems were ready to restore files. They also made a plan for communicating with employees without causing panic. Roles were assigned so everyone knew what to do.

This phase taught an important lesson: even good technology can fail if people are not careful. Human awareness is very important in cybersecurity.

## 4. Modelling to Evaluation

Next, the team studied the attack patterns and tested recovery strategies:

### Modelling

- They made a timeline to see how the malware moved.
- They created a model to find which systems were most at risk.
- They tested different ways to fix the problem safely.

### Evaluation:

Every action was tested carefully. Systems were restored step by step, and files were checked to make sure they worked. The team also checked whether new security measures would stop similar attacks in the future.

This stage showed the importance of careful planning and data-based decisions. Recovery is not just about acting fast but also about making systems safer in the long term.

## 5. Deployment to Feedback

After testing, the organization restored systems and improved security:

### Deployment

- Locked files were restored from safe backups.
- Critical systems were turned on carefully.
- Security was improved with multi-factor login, updated firewalls, and antivirus protection.
- Employees were trained to spot phishing emails and other tricks.

## Feedback Loop:

After everything was back to normal, the team reviewed the attack. Lessons were written down, and new measures were added:

- Continuous monitoring of network traffic
- Regular phishing tests for employees
- Updating old systems to make them safer

This feedback helped the organization learn from the attack and get better at preventing future attacks.

## Conclusion

The cyberattack was a big lesson. It showed that cybersecurity is not only about technology but also about people and processes. By following a step-by-step approach from problem finding to feedback, the company recovered quickly and became stronger.

Key Points:

- People are as important as technology in cybersecurity.
- Using data helps respond faster and smarter.
- Learning from mistakes is very important to stay safe.

In the end, the company became stronger with better rules, trained employees, and safer systems. Even a big crisis can teach valuable lessons if handled carefully.

# Practical no.2

## Numpy -Working with Numpy arrays

**Aim 2A):** Study Numpy library, Arrays, Dimensions- 2D, 3D, ND, Broadcasting, Indexing, Sciling Numpy Functions: array manipulation, string, arithmetic, statistical, Numpy Functions: arrange, linespace, random number generation, seed, reshape, ravel

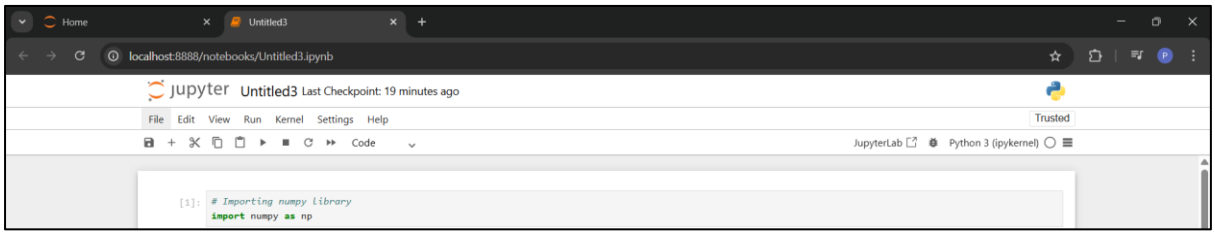
### Theory:- Numpy library

- NumPy (Numerical Python) is a powerful library used for scientific and mathematical computing in Python.
- It provides multi-dimensional arrays, along with tools to perform mathematical, logical, and statistical operations efficiently.

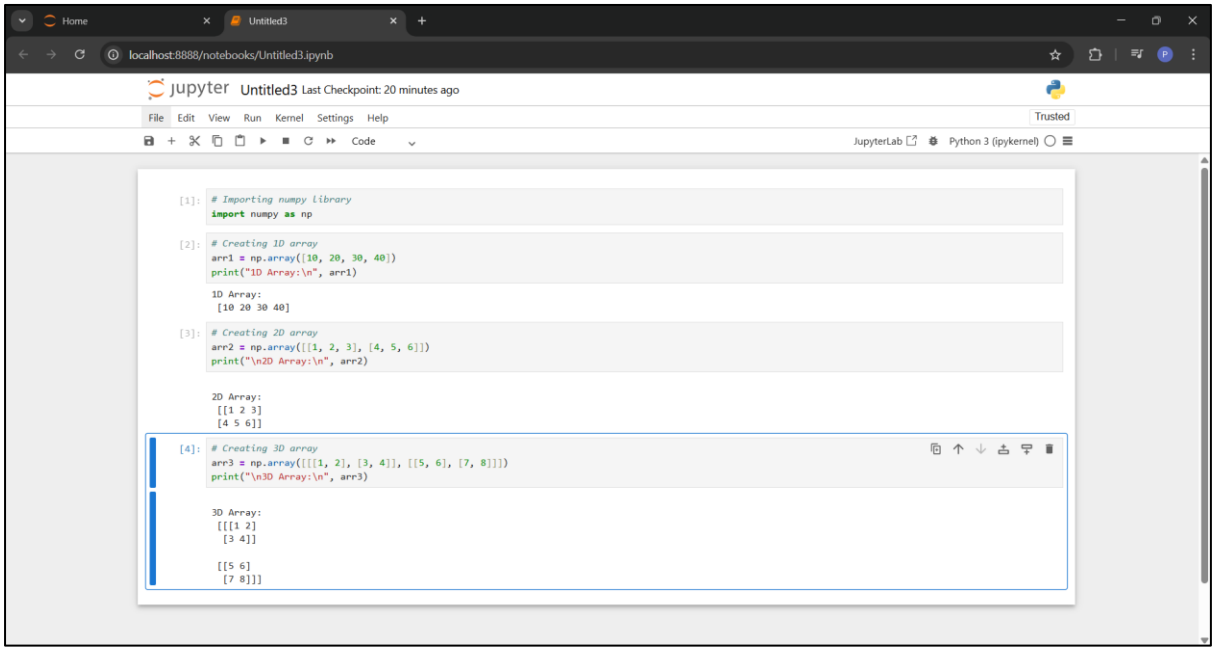
### Key Features

- Provides **N-dimensional arrays (ndarray)**
- Performs **fast numerical computations**
- Supports **broadcasting** for arithmetic between arrays
- Has built-in functions for **statistics, reshaping, random numbers**, etc.

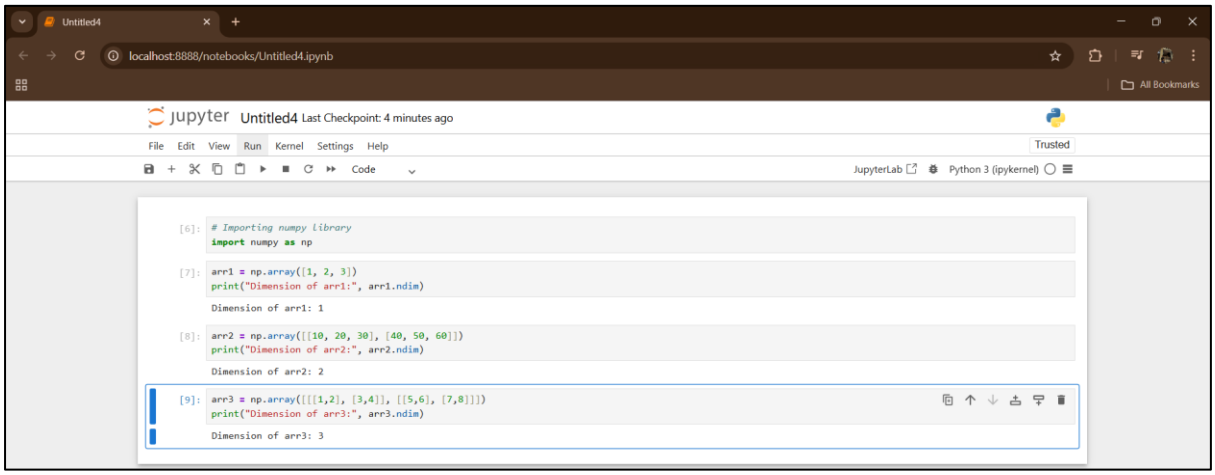
### 1. Importing NumPy Library



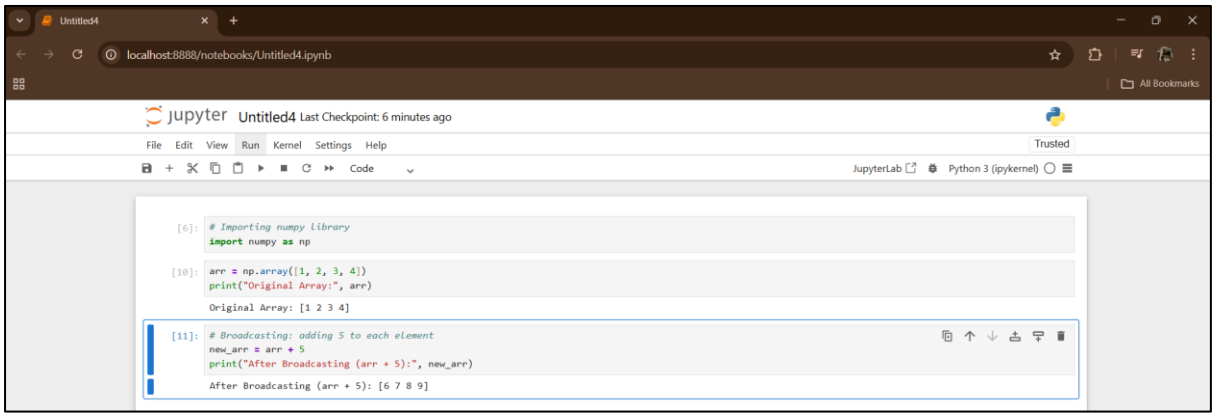

### 2. Creating Arrays



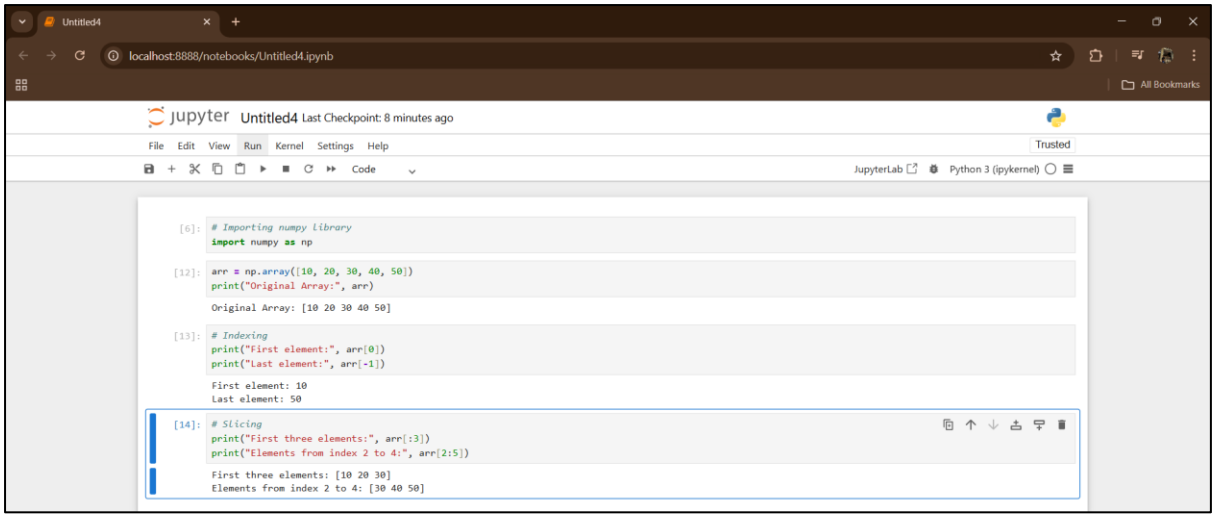
## 3. Checking Array Dimensions




## 4. Broadcasting




## 5. Indexing and Slicing



## 6. Array Manipulation Functions (reshape, ravel)

```
[6]: # Importing numpy library
import numpy as np

[15]: arr = np.arange(1, 10)
print("Original Array:\n", arr)

Original Array:
 [1 2 3 4 5 6 7 8 9]

[16]: # Reshape to 3x3
reshaped = arr.reshape(3, 3)
print("\nReshaped Array (3x3):\n", reshaped)


Reshaped Array (3x3):
 [[1 2 3]
 [4 5 6]
 [7 8 9]]

[17]: # Ravel (flatten) array
flat = reshaped.ravel()
print("\nFlattened Array using ravel():", flat)


Flattened Array using ravel(): [1 2 3 4 5 6 7 8 9]
```

## 7. String Functions

```
[6]: # Importing numpy library
import numpy as np

[18]: str_arr = np.array(['Python', 'NumPy', 'Lab'])
print("Original String Array:", str_arr)

Original String Array: ['Python' 'NumPy' 'Lab']

[19]: # Convert to lowercase
print("Lowercase:", np.char.lower(str_arr))

Lowercase: ['python' 'numpy' 'lab']

[20]: # Add text
print("Add ' Class':", np.char.add(str_arr, ' Class'))

Add ' Class': ['Python Class' 'NumPy Class' 'Lab Class']

[21]: # Replace character
print("Replace 'P' with 'X':", np.char.replace(str_arr, 'P', 'X'))

Replace 'P' with 'X': ['Xython' 'NumXy' 'Lab']
```

## 8. Arithmetic Functions

```
[6]: # Importing numpy library
import numpy as np

[23]: a = np.array([5, 10, 15])
b = np.array([2, 4, 6])

print("Addition:", np.add(a, b))
print("Subtraction:", np.subtract(a, b))
print("Multiplication:", np.multiply(a, b))
print("Division:", np.divide(a, b))

Addition: [ 7 14 21]
Subtraction: [3 6 9]
Multiplication: [10 40 90]
Division: [2.5 2.5 2.5]
```

## 9. Statistical Functions

```
[6]: # Importing numpy library
import numpy as np

[24]: data = np.array([5, 10, 15, 20, 25])

print("Mean:", np.mean(data))
print("Median:", np.median(data))
print("Standard Deviation:", np.std(data))
print("Variance:", np.var(data))
print("Sum:", np.sum(data))

Mean: 15.0
Median: 15.0
Standard Deviation: 7.071067811865475
Variance: 50.0
Sum: 75
```

## 10. arange() and linspace() Functions

```
[6]: # Importing numpy library
     import numpy as np
```

```
[25]: import numpy as np

      # arange(start, stop, step)
      arr1 = np.arange(1, 11, 2)
      print("Using arange():", arr1)

      # linspace(start, stop, num)
      arr2 = np.linspace(0, 1, 5)
      print("Using linspace():", arr2)
```

```
Using arange(): [1 3 5 7 9]
Using linspace(): [0.   0.25 0.5  0.75 1.  ]
```

## 11. Random Number Generation and Seed

```
[6]: # Importing numpy library
     import numpy as np
```

```
[26]: # Random numbers between 0 and 1
      rand_nums = np.random.rand(4)
      print("Random Numbers:", rand_nums)

      # Using seed (same random numbers every time)
      np.random.seed(42)
      print("Random Numbers with Seed=42:", np.random.rand(4))
```

```
Random Numbers: [0.01121711 0.8972144  0.08019634 0.52798045]
Random Numbers with Seed=42: [0.37454012 0.95071431 0.73199394 0.59865848]
```

## Conclusion:

In this practical, we studied the NumPy library, which is one of the most important tools for numerical and scientific computing in Python. We learned how to create and work with 1D, 2D, and 3D arrays, and how to check their dimensions using the .ndim property. NumPy arrays are much faster and more memory-efficient compared to normal Python lists. We performed various operations such as indexing, slicing, reshaping, and flattening arrays using functions like reshape() and ravel(). The broadcasting feature of NumPy allows arithmetic operations between arrays of different shapes very efficiently. We also explored important statistical functions such as mean, median, variance, and standard deviation to analyze data. Functions like arange() and linspace() help in creating number sequences easily for mathematical calculations. NumPy also provides random number generation and seed control for reproducible results in simulations. Its string and arithmetic functions make data manipulation simple and quick. Overall, NumPy serves as the foundation library for Data Science, Machine Learning, and Analytics, helping to perform complex numerical operations easily and effectively.

# Practical no.2

## Pandas- Working with Pandas data frames

**Aim 2B):** Pandas - Series functions: empty, ndim, size, dtype, values, head, tail, DataFrame functions: datatype, transpose, empty, ndim, shape, size, values, head, tail, DateTime

### Software Used:- Jupyter Notebook

### Theory:- Pandas

- Pandas is a popular open-source Python library used for **data manipulation, cleaning, and analysis**. It provides two main data structures: **Series** (one-dimensional) and **DataFrame** (two-dimensional).
- Pandas makes it easy to handle large datasets by offering simple and powerful functions for sorting, filtering, and performing mathematical or statistical operations.
- It is built on top of NumPy and integrates well with other data science libraries like Matplotlib and Scikit-learn. With Pandas, we can read data from multiple file formats like CSV, Excel, and SQL, and analyze it efficiently using just a few lines of code.

### SERIES FUNCTIONS

```
[1]: # Import the pandas library for data manipulation and analysis
     import pandas as pd
```

```
[23]: # Load the CSV file into a DataFrame
      data = pd.read_csv("C:/Users/Priyanshu/Downloads/DATASETS (5)/DATASETS/automobile.csv")
      print("Series Functions:\n")
      print(data)
```

```
Series Functions:

     symboling  normalized-losses         make aspiration num-of-doors  \
0            3                122  alfa-romero        std          two
1            3                122  alfa-romero        std          two
2            1                122  alfa-romero        std          two
3            2                164         audi        std         four
4            2                164         audi        std         four
```

```
[11]: # Check if the Series is empty
      print("Empty:", series_data.empty)

Empty: False

[12]: # Find the number of dimensions (for Series it is always 1)
      print("Dimensions (ndim):", series_data.ndim)

Dimensions (ndim): 1

[13]: # Find the total number of elements in the Series
      print("Size:", series_data.size)

Size: 201

[14]: # Check the data type of elements in the Series
      print("Data Type (dtype):", series_data.dtype)

Data Type (dtype): object

[15]: # Display the first 5 values of the Series
      print("Values:\n", series_data.values[:5])

Values:
 ['alfa-romero' 'alfa-romero' 'alfa-romero' 'audi' 'audi']

[16]: # Display the first 5 rows using head()
      print("Head:\n", series_data.head())

Head:
 0    alfa-romero
1    alfa-romero
2    alfa-romero
3           audi
4           audi
Name: make, dtype: object
```

```
[17]: # Display the last 5 rows using tail()
      print("Tail:\n", series_data.tail())
```

```
Tail:
 196     volvo
197     volvo
198     volvo
199     volvo
200     volvo
Name: make, dtype: object
```

# DATAFRAME FUNCTIONS

```
[1]: # Import the pandas library for data manipulation and analysis
     import pandas as pd
```

```
[25]: # Load the CSV file into a DataFrame
      data = pd.read_csv("C:/Users/Priyanshu/Downloads/DATASETS (5)/DATASETS/automobile.csv")
      print("\nDataFrame Functions:\n")
```

```
DataFrame Functions:
```

```
[26]: # Display the data type of each column in the DataFrame
      print("Data Types of Each Column:\n", data.dtypes)
```

```
Data Types of Each Column:
 symboling             int64
normalized-losses      int64
make                  object
aspiration            object
num-of-doors          object
body-style            object
```

```
[27]: # Transpose the DataFrame (swap rows and columns) and show first 3 rows
      print("Transpose (First 3 Rows):\n", data.transpose().head(3))
```

```
Transpose (First 3 Rows):
                         0            1            2     3     4     5    \
symboling                3            3            1     2     2     2
normalized-losses      122          122          122   164   164   122
make          alfa-romero  alfa-romero  alfa-romero  audi  audi  audi

                    6     7     8    9    ...   191   192   193   194  \
symboling             1     1     1    2  ...    -1    -2    -1    -2
normalized-losses   158   122   158  192  ...    74   103    74   103
make               audi  audi  audi  bmw  ...  volvo  volvo  volvo  volvo

                   195   196   197   198   199   200
symboling           -1    -1    -1    -1    -1    -1
normalized-losses   74    95    95    95    95    95
make             volvo  volvo  volvo  volvo  volvo  volvo

[3 rows x 201 columns]
```

```
[28]: # Check if the DataFrame is empty
      print("Empty:", data.empty)
```

```
Empty: False
```

```
[29]: # Find the number of dimensions (for DataFrame it is 2)
      print("Dimensions (ndim):", data.ndim)
```

```
Dimensions (ndim): 2
```

```
[30]: # Display the shape (rows, columns)
      print("Shape (rows, columns):", data.shape)
```

```
Shape (rows, columns): (201, 29)
```

```
[31]: # Display total number of data elements in the DataFrame
      print("Size (total elements):", data.size)
```

```
Size (total elements): 5829
```

[32]:
```
# Display the first 3 rows of all values as an array
print("Values (First 3 Rows):\n", data.values[:3])
```

```
Values (First 3 Rows):
 [[3 122 'alfa-romero' 'std' 'two' 'convertible' 'rwd' 'front' 88.6
  0.8111484863046613 0.8902777777777777 48.8 2548 'dohc' 'four' 130
  'mpfi' 3.47 2.68 9.0 111.0 5000.0 21 27 13495.0 11.190476190476188
  'Medium' 0 1]
 [3 122 'alfa-romero' 'std' 'two' 'convertible' 'rwd' 'front' 88.6
  0.8111484863046613 0.8902777777777777 48.8 2548 'dohc' 'four' 130
  'mpfi' 3.47 2.68 9.0 111.0 5000.0 21 27 16500.0 11.190476190476188
  'Medium' 0 1]
 [1 122 'alfa-romero' 'std' 'two' 'hatchback' 'rwd' 'front' 94.5
  0.8226814031715521 0.9097222222222222 52.4 2823 'ohcv' 'six' 152 'mpfi'
  2.68 3.47 9.0 154.0 5000.0 19 26 16500.0 12.36842105263158 'Medium' 0 1]]
```

```
# Show first 5 rows of the DataFrame
print("Head:\n", data.head())
```

```
Head:
    symboling  normalized-losses         make aspiration num-of-doors  \
0          3                122  alfa-romero        std          two
1          3                122  alfa-romero        std          two
2          1                122  alfa-romero        std          two
3          2                164         audi        std         four
4          2                164         audi        std         four

    body-style drive-wheels engine-location  wheel-base    length  ...  \
0  convertible          rwd           front        88.6  0.811148  ...
1  convertible          rwd           front        88.6  0.811148  ...
2    hatchback          rwd           front        94.5  0.822681  ...
3        sedan          fwd           front        99.8  0.848630  ...
4        sedan          4wd           front        99.4  0.848630  ...

   compression-ratio  horsepower  peak-rpm city-mpg highway-mpg    price  \
0                9.0       111.0    5000.0       21          27  13495.0
1                9.0       111.0    5000.0       21          27  16500.0
2                9.0       154.0    5000.0       19          26  16500.0
3               10.0       102.0    5500.0       24          30  13950.0
4                8.0       115.0    5500.0       18          22  17450.0

  city-L/100km  horsepower-binned  diesel  gas
0    11.190476             Medium       0    1
1    11.190476             Medium       0    1
2    12.368421             Medium       0    1
3     9.791667             Medium       0    1
4    13.055556             Medium       0    1

[5 rows x 29 columns]
```

```
# Show last 5 rows of the DataFrame
print("Tail:\n", data.tail())
```

```
Tail:
     symboling  normalized-losses   make aspiration num-of-doors body-style  \
196         -1                 95  volvo        std         four      sedan
197         -1                 95  volvo      turbo         four      sedan
198         -1                 95  volvo        std         four      sedan
199         -1                 95  volvo      turbo         four      sedan
200         -1                 95  volvo      turbo         four      sedan

    drive-wheels engine-location  wheel-base    length  ...  \
196          rwd           front       109.1  0.907256  ...
197          rwd           front       109.1  0.907256  ...
198          rwd           front       109.1  0.907256  ...
199          rwd           front       109.1  0.907256  ...
200          rwd           front       109.1  0.907256  ...

     compression-ratio  horsepower  peak-rpm city-mpg highway-mpg    price  \
196                9.5       114.0    5400.0       23          28  16845.0
197                8.7       160.0    5300.0       19          25  19045.0
198                8.8       134.0    5500.0       18          23  21485.0
199               23.0       106.0    4800.0       26          27  22470.0
200                9.5       114.0    5400.0       19          25  22625.0

    city-L/100km  horsepower-binned  diesel  gas
196    10.217391             Medium       0    1
197    12.368421               High       0    1
198    13.055556             Medium       0    1
199     9.038462             Medium       1    0
200    12.368421             Medium       0    1

[5 rows x 29 columns]
```

```
print("\nDateTime Example:\n")

# Create a date range starting from 1st Jan 2025 for 5 days
date_series = pd.date_range(start="2025-01-01", periods=5, freq='D')

# Print the generated DateTime series
print(date_series)
```

```
DateTime Example:

DatetimeIndex(['2025-01-01', '2025-01-02', '2025-01-03', '2025-01-04',
               '2025-01-05'],
              dtype='datetime64[ns]', freq='D')
```

## Conclusion:

In this practical, we explored various **Pandas Series and DataFrame functions** using the automobile.csv dataset. We learned how to analyze dataset properties such as dimensions, size, data types, and structure. The Series functions like empty, ndim, size, dtype, head(), and tail() helped in understanding individual columns, while DataFrame functions such as shape, values, transpose(), and dtypes provided insights into the overall dataset. Additionally, the use of **DateTime** demonstrated Pandas' capability for handling time-based data.
Overall, this experiment helped in understanding how Pandas simplifies **data analysis, exploration, and manipulation** efficiently.

# Practical no.2

## Matplotlib & Seaborn- Basic plots using Matplotlib

**Aim 2C):** Study Matplotlib library, Seaborn, Plyplot, plotting, markers, line, labels, grid, subplot, scatter, bar, histogram, pie charts, countplot

**Software Used:- Jupyter Notebook**

**Theory:-**

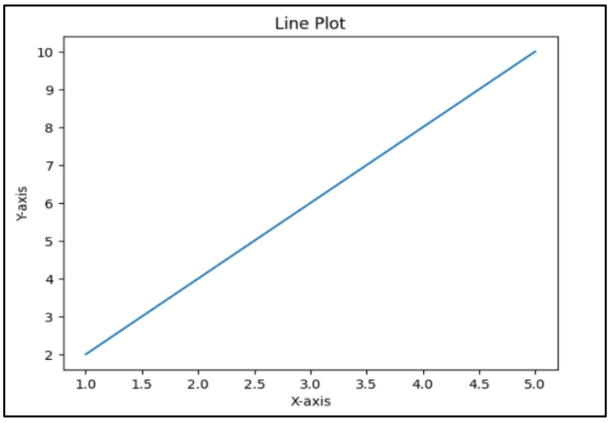
- Matplotlib is a Python library used for creating different types of 2D graphs and visualizations such as line, bar, scatter, histogram, and pie charts. It helps in understanding data through visual representation. The Pyplot module in Matplotlib provides simple functions to plot and customize graphs with labels, titles, grids, and subplots.
- Seaborn is a statistical data visualization library built on top of Matplotlib. It provides more attractive and easy-to-use functions for plotting data, such as countplot, boxplot, and heatmap. Both libraries are widely used in data analysis to make data more meaningful and easier to interpret.

**Plotting with Matplotlib pylot**

### 1. LINE PLOT

- This code uses **Matplotlib** and **Seaborn** to plot a simple line graph showing the relationship between x and y.
- The output displays a **straight line** through points (1,2), (2,4), (3,6), (4,8), and (5,10), showing a **linear increase** of y with x.

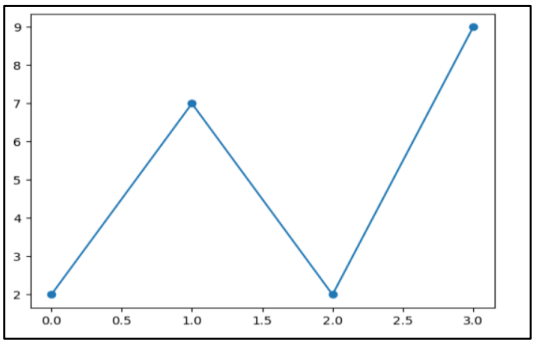
```
# Import libraries
import matplotlib.pyplot as plt
import seaborn as sns

x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]
plt.plot(x, y,)
plt.title("Line Plot")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.show()
```



## 2. Marker

- This code uses Matplotlib and Seaborn to plot data points from the NumPy array ypoints with circular markers.
- The output shows a line graph with points at (0,2), (1,7), (2,2), and (3,9), connected by lines and marked with ‘o’ symbols.

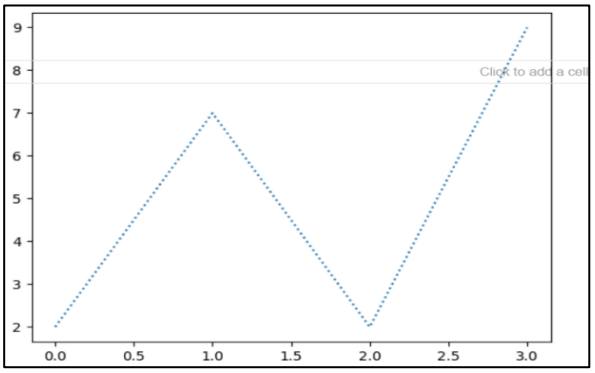
```
# Import libraries
import matplotlib.pyplot as plt
import seaborn as sns

ypoints = np.array([2,7,2,9])
plt.plot(ypoints, marker= 'o')
plt.show
```




## 3. Line type

- This code uses Matplotlib and Seaborn to plot the values from the NumPy array ypoints using a dotted line style.
- The output displays a line graph connecting the points (0,2), (1,7), (2,2), and (3,9) with a dotted line pattern.

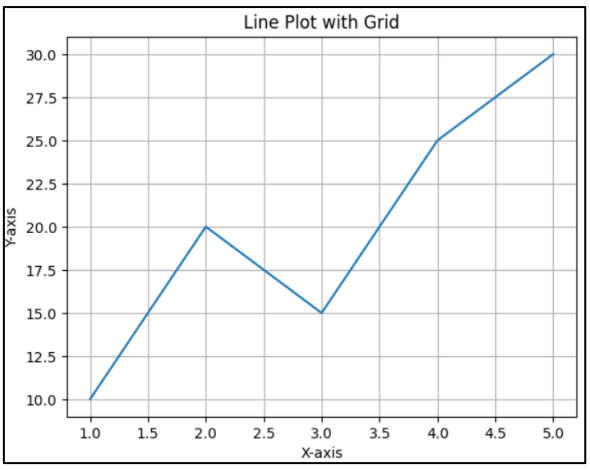
```
# Import libraries
import matplotlib.pyplot as plt
import seaborn as sns

ypoints = np.array([2,7,2,9])
plt.plot(ypoints, linestyle='dotted')
plt.show
```




## 4. Grid type

- This Python code uses Matplotlib to create a simple line plot of x vs y values with grid lines enabled.
- A line graph showing points (1,10), (2,20), (3,15), (4,25), and (5,30) connected by a line, labeled **“X-axis”** and **“Y-axis”**, titled **“Line Plot with Grid”**, with a visible grid in the background.

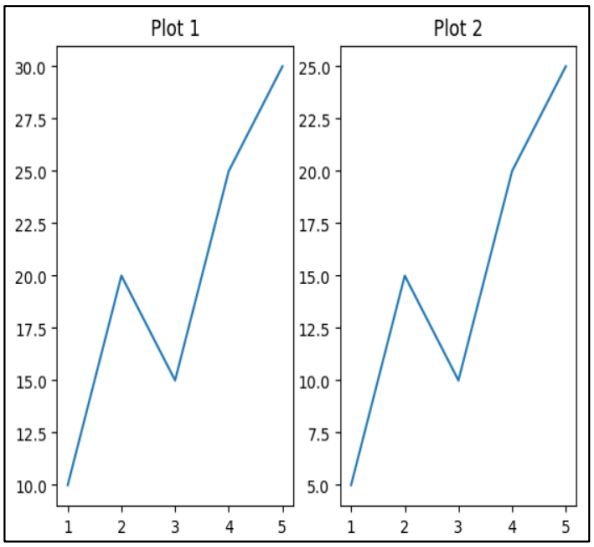
```
# Import libraries
import matplotlib.pyplot as plt
import seaborn as sns

x = [1, 2, 3, 4, 5]
y = [10, 20, 15, 25, 30]

plt.plot(x, y)
plt.title("Line Plot with Grid")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.grid(True)  # Enable grid
plt.show()
```



## 5. Subplot type

- Creates two side-by-side line plots.
- Two plots titled “Plot 1” and “Plot 2” showing different y values.

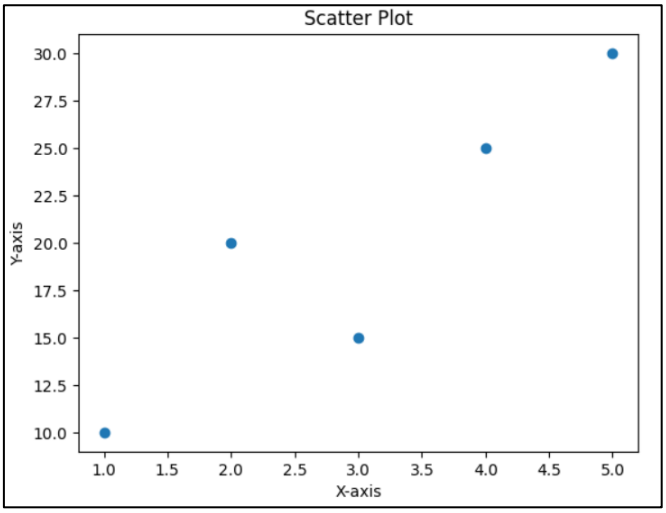
```
# Import libraries
import matplotlib.pyplot as plt
import seaborn as sns

x = [1, 2, 3, 4, 5]
y1 = [10, 20, 15, 25, 30]
y2 = [5, 15, 10, 20, 25]

plt.subplot(1, 2, 1)   # 1 row, 2 columns, plot 1
plt.plot(x, y1)
plt.title("Plot 1")

plt.subplot(1, 2, 2)   # 1 row, 2 columns, plot 2
plt.plot(x, y2)
plt.title("Plot 2")

plt.show()
```




## 6. Scatter Plot

- Creates a scatter plot of x vs y values.
- Dots showing data points like (1,10), (2,20), (3,15), (4,25), (5,30).
- A **scatter plot** displays individual data points to show the **relationship between two variables** (how one changes with the other).

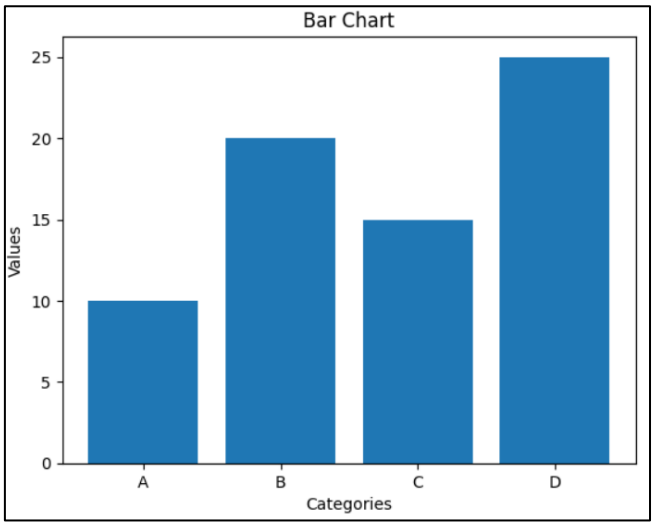
```
# Import libraries
import matplotlib.pyplot as plt
import seaborn as sns

x = [1, 2, 3, 4, 5]
y = [10, 20, 15, 25, 30]

plt.scatter(x, y)
plt.title("Scatter Plot")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.show()
```



## 7. Bar Chart

- This code uses Matplotlib to create and display a bar chart comparing categories A–D with values 10, 20, 15, and 25, where each bar's height represents its value—showing how bar charts visually compare data across categories.
- A **bar chart** is a graphical representation of data using rectangular bars to compare different categories or groups by their values.

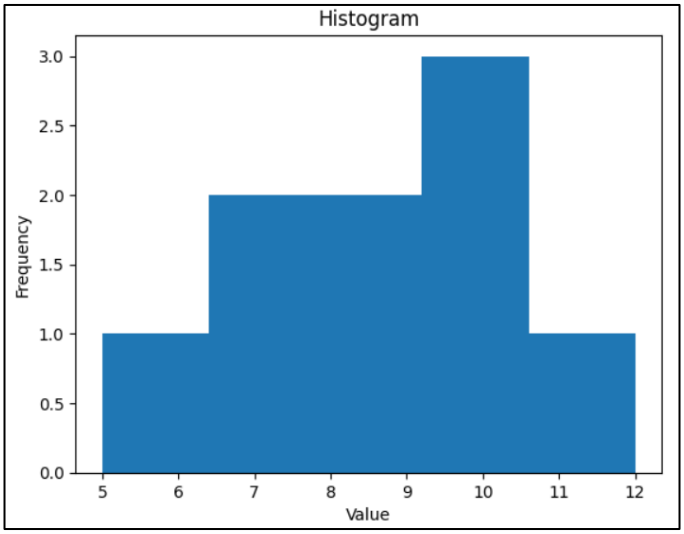
```
# Import libraries
import matplotlib.pyplot as plt
import seaborn as sns

x = ['A', 'B', 'C', 'D']
y = [10, 20, 15, 25]
plt.bar(x, y)
plt.title("Bar Chart")
plt.xlabel("Categories")
plt.ylabel("Values")
plt.show()
```




## 8. Histogram

- This code uses Matplotlib to create and display a histogram dividing the data [5, 7, 7, 8, 9, 10, 10, 10, 12] into 5 bins, showing how often (frequency) values fall within each range—illustrating that a histogram displays the distribution of numerical data.

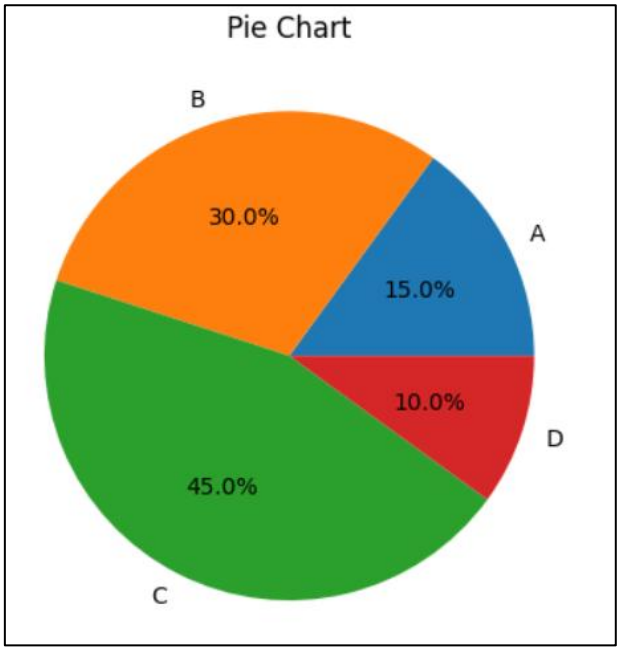
```
# Import libraries
import matplotlib.pyplot as plt
import seaborn as sns

data = [5, 7, 7, 8, 9, 10, 10, 10, 12]

plt.hist(data, bins=5)  # 5 bins
plt.title("Histogram")
plt.xlabel("Value")
plt.ylabel("Frequency")
plt.show()
```



## 9. Pie Chart

- This code uses Matplotlib to create and display a pie chart showing categories A–D with proportions 15%, 30%, 45%, and 10%, where each slice's size represents its share of the total—illustrating that a pie chart shows how parts contribute to a whole.

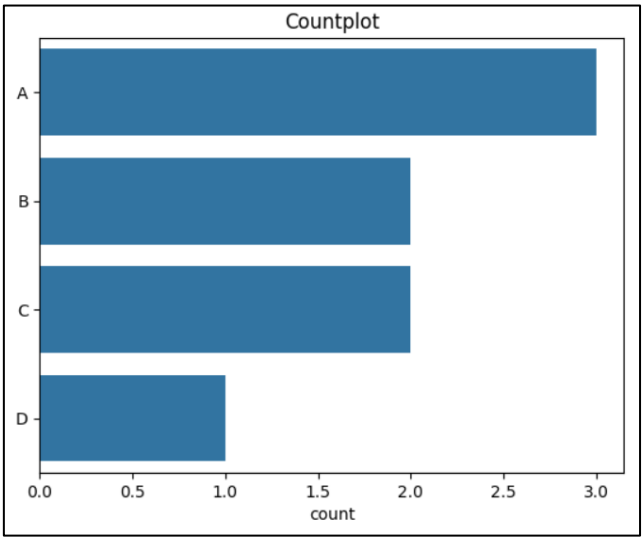
```
# Import libraries
import matplotlib.pyplot as plt
import seaborn as sns

labels = ['A', 'B', 'C', 'D']
sizes = [15, 30, 45, 10]

plt.pie(sizes, labels=labels,
        autopct='%1.1f%%')
plt.title("Pie Chart")
plt.show()
```




## 10. Seaborn Countplot

- This code uses Seaborn to create and display a count plot showing how many times each category (A, B, C, D) appears in the data list—illustrating that a count plot visualizes the frequency of categorical values.

```
# Import libraries
import matplotlib.pyplot as plt
import seaborn as sns

import seaborn as sns

data = ['A', 'B', 'A', 'C',
        'B', 'A', 'D', 'C']
sns.countplot(data=data)
plt.title("Countplot")
plt.show()
```




## Conclusion:

In this practical, we learned how to visualize data using **Matplotlib** and **Seaborn**. Matplotlib helps create various plots like line, bar, scatter, and pie charts with titles, labels, and grids to make them clear and attractive. Seaborn, built on Matplotlib, makes it easier to create stylish statistical plots such as countplots and works well with Pandas data.

# Practical no.3

# Data Analysis

**Aim 3A):** Study Data Analysis, Import Libraries & datasets, load data, check data type, shape, drop columns, merge datasets, sort, concatenate, statistical summary of data, skewness, co-relation.

**Software Used:- Jupyter Notebook**

## Theory:-

Data analysis is all about exploring and understanding data to uncover useful information and insights. In Python, we usually use libraries like **Pandas**, **NumPy**, **Matplotlib**, and **Seaborn** to make this process easier.

The first step is to bring in the necessary libraries and load the dataset into a **Pandas DataFrame**. After that, it's important to look at the **data types** and **shape** of the dataset to get a sense of what kind of information it holds.

Next, we can clean the data by removing unnecessary columns and combining multiple datasets using **merge** or **concatenate** if needed. Sorting the data helps organize it based on specific columns, making it easier to work with.

We also use basic statistical methods to summarize the data, like finding the **mean, median, mode, minimum, and maximum**. Checking **skewness** helps us understand the distribution, while **correlation** shows how different variables relate to each other.

Following these steps helps prepare the data for further analysis or visualization, so we can make better decisions based on what the data tells us.

### 1. Check data type and load data

```
[66]: import  pandas as pd
      import numpy as np

[69]: dataset= pd.read_csv("C:\\Users\\Priyanshu\\Downloads\\DATASETS (5)\\DATASETS\\melb_data.csv")

[72]: df=pd.DataFrame(dataset)
      type(df)

[72]: pandas.core.frame.DataFrame
```

### 2. Shape and Drop columns

```
[75]: #shape and drop columns
      df.shape

[75]: (13580, 21)

[74]: dataset2=df.drop(['Type'],axis=1)
      dataset2.shape

[74]: (13580, 20)
```

## 3. Merge Datasets

```
[79]: dataset= pd.read_csv("C:\\Users\\Priyanshu\\Downloads\\DATASETS (5)\\DATASETS\\melb_data.csv")
      dataset.to_excel("melb_data.xlsm",index=False)

[80]: cd=pd.merge(df,dataset,on="Address")

[81]: cd.shape

[81]: (14002, 41)
```

## 4. Sort

```
[82]: dataset3=cd.sort_values(by=["Address"])
      dataset3.head()
```

[82]:

| | Suburb_x | Address | Rooms_x | Type_x | Price_x | Method_x | SellerG_x | Date_x | Distance_x | Postcode_x | ... | Bathroom_y | Car_y | Landsize_y | Building |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **8049** | Doncaster | 1 Adelle Ct | 3 | h | 1700000.0 | S | Barry | 29/04/2017 | 13.9 | 3108.0 | ... | 2.0 | 2.0 | 820.0 | |
| **13062** | Brighton | 1 Airlie St | 3 | h | 1342000.0 | S | Marshall | 16/09/2017 | 10.5 | 3186.0 | ... | 1.0 | 3.0 | 287.0 | |
| **13614** | Hoppers Crossing | 1 Albion Ct | 3 | h | 650000.0 | S | Purplebricks | 23/09/2017 | 18.4 | 3029.0 | ... | 2.0 | 2.0 | 913.0 | |
| **1241** | Brighton East | 1 Alexander St | 3 | h | 1301000.0 | SP | Buxton | 12/11/2016 | 10.7 | 3187.0 | ... | 2.0 | 2.0 | 401.0 | |
| **12606** | Aberfeldie | 1 Alma St | 4 | h | 1436000.0 | S | Brad | 3/09/2017 | 7.5 | 3040.0 | ... | 3.0 | 3.0 | 511.0 | |

5 rows × 41 columns

## 5. Concatenate

```
[83]: finaldata=pd.concat([cd,dataset3])
      finaldata.shape

[83]: (28004, 41)
```

## 6. Statistical summary of data

```
[85]: #Generate and display statistical summary of finaldata
      summary=finaldata.describe()
      print(summary)
```

```
             Rooms_x       Price_x    Distance_x    Postcode_x    Bedroom2_x  \
count  28004.000000  2.800400e+04  28004.000000  28004.000000  28004.000000
mean       2.939937  1.078112e+06     10.091237   3104.638409      2.915941
std        0.957568  6.387724e+05      5.843775     90.307245      0.967717
min        1.000000  8.500000e+04      0.000000   3000.000000      0.000000
25%        2.000000  6.500000e+05      6.100000   3044.000000      2.000000
50%        3.000000  9.080000e+05      9.200000   3084.000000      3.000000
75%        3.000000  1.335000e+06     13.000000   3147.000000      3.000000
max       10.000000  9.000000e+06     48.100000   3977.000000     20.000000

         Bathroom_x        Car_x     Landsize_x  BuildingArea_x  \
count  28004.000000  27876.000000   28004.000000    14758.000000
mean       1.535566      1.610059     555.259963      152.176195
std        0.691999      0.967329    3930.682116      532.177051
min        0.000000      0.000000       0.000000        0.000000
25%        1.000000      1.000000     177.000000       93.000000
50%        1.000000      2.000000     441.000000      126.000000
75%        2.000000      2.000000     650.000000      174.000000
max        8.000000     10.000000  433014.000000    44515.000000
```

## 7. Skewness

```
[87]: num_feature=finaldata.select_dtypes(include=np.number)
      skewness=num_feature.skew()
      print("Skewness:",skewness)
```

```
Skewness: Rooms_x              0.378707
Price_x                2.210548
Distance_x             1.702058
Postcode_x             4.085918
Bedroom2_x             0.761034
Bathroom_x             1.368175
Car_x                  1.385835
Landsize_x            96.652378
BuildingArea_x        78.855156
YearBuilt_x           -1.485805
Lattitude_x           -0.441373
Longtitude_x          -0.197889
Propertycount_x        1.059873
Rooms_y                0.378707
Price_y                2.210548
Distance_y             1.702058
```

## 8. Co-relation

```
num_feature=finaldata.select_dtypes(include=np.number)
corrs=num_feature.corr()
print("Corelation:",corrs)
```

```
Corelation:                 Rooms_x   Price_x  Distance_x  Postcode_x  Bedroom2_x  \
Rooms_x          1.000000  0.496936    0.289608    0.053805    0.943409
Price_x          0.496936  1.000000   -0.161445    0.109585    0.476725
Distance_x       0.289608 -0.161445    1.000000    0.432205    0.291498
Postcode_x       0.053805  0.109585    0.432205    1.000000    0.059007
Bedroom2_x       0.943409  0.476725    0.291498    0.059007    1.000000
Bathroom_x       0.591450  0.468720    0.123966    0.111840    0.583749
Car_x            0.402827  0.237571    0.259098    0.048269    0.400561
Landsize_x       0.025711  0.037450    0.025575    0.024660    0.025747
BuildingArea_x   0.125327  0.092541    0.098622    0.054879    0.123534
YearBuilt_x     -0.070695 -0.323710    0.247228    0.031281   -0.058910
Lattitude_x      0.014090 -0.213874   -0.133112   -0.409620    0.014436
Longtitude_x     0.098568  0.204237    0.239686    0.449433    0.100150
Propertycount_x -0.080541 -0.037271   -0.056266    0.064210   -0.080619
Rooms_y          0.988083  0.491988    0.287622    0.054459    0.933851
```

## Conclusion:

To conclude, performing data analysis in Python allows us to efficiently manage and explore large datasets. By using libraries, loading data, and applying operations such as removing unnecessary columns, merging datasets, sorting, and generating summaries, we can extract valuable insights. Analyzing skewness and correlation provides a better understanding of patterns and relationships within the data. In summary, data analysis is essential for making well-informed decisions and preparing the data for visualization, reporting, or predictive modeling. It is a fundamental part of any data science or research workflow.

# Practical no.3

# Data Wrangling

**Aim 3B):** Study Data Wrangling, Pre-processing Data - Dealing with Missing values, Correcting Data Format, Data standardization, Data Normalization, Binning, Turning categorical variables into quantitative variables.

**Software Used:- Jupyter Notebook**

## Theory:-

Data wrangling is the process of cleaning, transforming, and organizing raw data so that it can be easily used for analysis. This step is very important in data science because real-world data is often messy, incomplete, or inconsistent.

Pre-processing the data involves several key steps:

1. **Handling missing values** – filling them with the mean, median, or mode, or removing rows or columns that have missing data.
2. **Correcting data types** – making sure data such as dates, numbers, or text are in the correct format.
3. **Standardizing data** – adjusting numerical data to a common scale with a mean of 0 and a standard deviation of 1.
4. **Normalizing data** – scaling values to fall within a specific range, usually 0 to 1, which is helpful for machine learning models.
5. **Binning** – dividing continuous data into groups or categories to simplify analysis and reduce noise.
6. **Encoding categorical variables** – converting non-numerical data like "Yes/No" or "Red/Blue" into numerical values using methods such as label encoding or one-hot encoding.

### 1. Import Libraries && Load Data

```
[91]: import pandas as pd
      import numpy as np
      from sklearn.preprocessing import StandardScaler
      from sklearn.preprocessing import MinMaxScaler

[97]: dataset= pd.read_csv("C:\\Users\\Priyanshu\\Downloads\\DATASETS (5)\\DATASETS\\melb_data.csv")

[98]: dataset.head()

[98]:
```

| | Suburb | Address | Rooms | Type | Price | Method | SellerG | Date | Distance | Postcode | ... | Bathroom | Car | Landsize | BuildingArea | YearBuilt | CouncilAr |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | Abbotsford | 85 Turner St | 2 | h | 1480000.0 | S | Biggin | 3/12/2016 | 2.5 | 3067.0 | ... | 1.0 | 1.0 | 202.0 | NaN | NaN | Yar |
| 1 | Abbotsford | 25 Bloomburg St | 2 | h | 1035000.0 | S | Biggin | 4/02/2016 | 2.5 | 3067.0 | ... | 1.0 | 0.0 | 156.0 | 79.0 | 1900.0 | Yar |
| 2 | Abbotsford | 5 Charles St | 3 | h | 1465000.0 | SP | Biggin | 4/03/2017 | 2.5 | 3067.0 | ... | 2.0 | 0.0 | 134.0 | 150.0 | 1900.0 | Yar |
| 3 | Abbotsford | 40 Federation La | 3 | h | 850000.0 | PI | Biggin | 4/03/2017 | 2.5 | 3067.0 | ... | 2.0 | 1.0 | 94.0 | NaN | NaN | Yar |
| 4 | Abbotsford | 55a Park St | 4 | h | 1600000.0 | VB | Nelson | 4/06/2016 | 2.5 | 3067.0 | ... | 1.0 | 2.0 | 120.0 | 142.0 | 2014.0 | Yar |

5 rows × 21 columns

## 2. Creating Dataframe && Drop missing Value

```
[102]: #Creating Dataframe
       df=pd.DataFrame(dataset)

[103]: #drop missing values
       data_without_missing_row=df.dropna()
       data_without_missing_columns=df.dropna(axis=1)
       data_without_missing_columns.shape

[103]: (13580, 17)
```

## 3. Correlating Data Format

```
[104]: df['Rooms']=pd.to_numeric(df['Rooms'])
       df.describe()
```

[104]:

| | Rooms | Price | Distance | Postcode | Bedroom2 | Bathroom | Car | Landsize | BuildingArea | YearBuilt | Lattitude |
|---|---|---|---|---|---|---|---|---|---|---|---|
| count | 13580.000000 | 1.358000e+04 | 13580.000000 | 13580.000000 | 13580.000000 | 13580.000000 | 13518.000000 | 13580.000000 | 7130.000000 | 8205.000000 | 13580.000000 |
| mean | 2.937997 | 1.075684e+06 | 10.137776 | 3105.301915 | 2.914728 | 1.534242 | 1.610075 | 558.416127 | 151.967650 | 1964.684217 | -37.809203 |
| std | 0.955748 | 6.393107e+05 | 5.868725 | 90.676964 | 0.965921 | 0.691712 | 0.962634 | 3990.669241 | 541.014538 | 37.273762 | 0.079260 |
| min | 1.000000 | 8.500000e+04 | 0.000000 | 3000.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 1196.000000 | -38.182550 |
| 25% | 2.000000 | 6.500000e+05 | 6.100000 | 3044.000000 | 2.000000 | 1.000000 | 1.000000 | 177.000000 | 93.000000 | 1940.000000 | -37.856822 |
| 50% | 3.000000 | 9.030000e+05 | 9.200000 | 3084.000000 | 3.000000 | 1.000000 | 2.000000 | 440.000000 | 126.000000 | 1970.000000 | -37.802355 |
| 75% | 3.000000 | 1.330000e+06 | 13.000000 | 3148.000000 | 3.000000 | 2.000000 | 2.000000 | 651.000000 | 174.000000 | 1999.000000 | -37.756400 |
| max | 10.000000 | 9.000000e+06 | 48.100000 | 3977.000000 | 20.000000 | 8.000000 | 10.000000 | 433014.000000 | 44515.000000 | 2018.000000 | -37.408530 |

## 4. Data Standardization

```
[105]: scaler=StandardScaler()
       data_Standard=scaler.fit_transform(df[['Rooms']])
       data_Standard

[105]: array([[-0.98146337],
              [-0.98146337],
              [ 0.06487613],
              ...,
              [ 0.06487613],
              [ 1.11121563],
              [ 1.11121563]], shape=(13580, 1))
```

## 5. Data Normalization

```
[106]: MinMaxscaler=MinMaxScaler()
       data_Normalized=MinMaxscaler.fit_transform(df[['Rooms']])
       data_Normalized

[106]: array([[0.11111111],
              [0.11111111],
              [0.22222222],
              ...,
              [0.22222222],
              [0.33333333],
              [0.33333333]], shape=(13580, 1))
```

## 6. Simple Feature Scaling

```
df['Price_scaled']=df['Price'] / df['Price'].max()
print(df['Price_scaled'])
```

```
0        0.164444
1        0.115000
2        0.162778
3        0.094444
4        0.177778
           ...
13575    0.138333
13576    0.114556
13577    0.130000
13578    0.277778
13579    0.142778
Name: Price_scaled, Length: 13580, dtype: float64
```

## 7. Min Max-Scaling in Python

```
[133]: df['Price_scaled']=(df['Price'] - df['Price'].min()) / (df['Price'].max() - df['Price'].min())
       print(df['Price_scaled'])
```

```
0        0.156478
1        0.106562
2        0.154795
3        0.085810
4        0.169938
           ...
13575    0.130118
13576    0.106113
13577    0.121705
13578    0.270892
13579    0.134605
Name: Price_scaled, Length: 13580, dtype: float64
```

## 8. Z – Score in Python

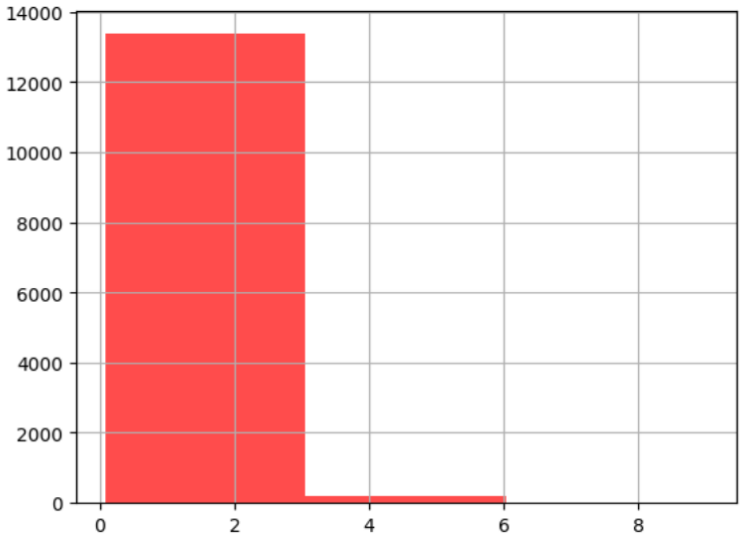
```
df['Price_scaled']=(df['Price'] - df['Price'].mean()) / df['Price'].std()
print(df['Price_scaled'])
```

```
0        0.632425
1       -0.063637
2        0.608962
3       -0.353012
4        0.820127
           ...
13575    0.264841
13576   -0.069894
13577    0.147528
13578    2.227893
13579    0.327409
Name: Price_scaled, Length: 13580, dtype: float64
```

## 9. Binning

```
[135]: Bins= np.linspace(min(df['Price']),max(df['Price']),4)
       labels=['Low','Medium', 'High']
       df['Price_Category'] = pd.cut(df['Price'],bins=Bins,labels=labels,include_lowest=True)
```

```
[136]: import matplotlib.pyplot as plt
       df['Price'].hist(bins=3,color='red',alpha=0.7)
       plt.grid(True)
       plt.show()
```




## 10. Turning Categorical Variable into Quantitative Variable

```
[137]: pd.get_dummies(df['Rooms'])
```

[137]:

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| 0 | False | True | False | False | False | False | False | False | False |
| 1 | False | True | False | False | False | False | False | False | False |
| 2 | False | False | True | False | False | False | False | False | False |
| 3 | False | False | True | False | False | False | False | False | False |
| 4 | False | False | False | True | False | False | False | False | False |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 13575 | False | False | False | True | False | False | False | False | False |
| 13576 | False | False | True | False | False | False | False | False | False |
| 13577 | False | False | True | False | False | False | False | False | False |
| 13578 | False | False | False | True | False | False | False | False | False |
| 13579 | False | False | False | True | False | False | False | False | False |

## Conclusion:-

In summary, data wrangling and pre-processing are vital steps in any data analysis or machine learning task. Taking care of missing values, fixing data formats, standardizing and normalizing numbers, grouping data through binning, and converting categorical variables into numerical form ensures that the dataset is clean and ready for analysis. Proper pre-processing helps minimize errors, enhances the performance of models, and makes it easier to uncover valuable insights from the data. These steps lay the groundwork for making accurate, data-driven decisions.

# Practical no.4

# Exploratory Data Analysis Descriptive Statistics

**Aim 4A):** Study Exploratory Data Analysis, Descriptive Statistics, Categorical Variables- box plots, Value Counts, Scatterplot, Group By, Pivot Table, heat Map.

**Software Used:- Jupyter Notebook**

## Theory:-

### Exploratory Data Analysis (EDA)

Exploratory Data Analysis, or EDA, is the process of examining and understanding a dataset before performing any formal modeling or analysis. The main goal of EDA is to **summarize the data, uncover patterns, detect anomalies, and test assumptions**. It helps in identifying trends, relationships between variables, and potential problems in the dataset. EDA is an essential step in data analysis because it gives a clear understanding of the data and guides the choice of appropriate techniques for further analysis or modeling.
EDA often involves **visualization techniques** like histograms, box plots, scatter plots, and count plots to provide a clear picture of the data distribution and relationships.

### Descriptive Statistics

Descriptive statistics are numerical measures that **summarize and describe the main features of a dataset**. They provide a simple way to understand the data at a glance without going into complex modeling.
Common descriptive statistics include:

- **Mean, median, mode** – measure of central tendency
- **Minimum, maximum, range** – measures of spread
- **Variance, standard deviation** – measure of data dispersion
- **Skewness and kurtosis** – describe the shape and symmetry of the data distribution

By using descriptive statistics, we can quickly understand **how the data is distributed, its variability, and potential outliers**, which is critical before moving on to predictive modeling or deeper analysis.

### 1. Import Library and Load Data

```
[140]: import pandas as pd
       import numpy as np
       import seaborn as sns
       import matplotlib.pyplot as plt

[141]: dataset= pd.read_csv("C:\\Users\\Priyanshu\\Downloads\\DATASETS (5)\\DATASETS\\automobile.csv")

[142]: dataset.head()
```

[142]:

| | symboling | normalized-losses | make | aspiration | num-of-doors | body-style | drive-wheels | engine-location | wheel-base | length | ... | compression-ratio | horsepower | peak-rpm | city-mpg | highway-mpg | price |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 3 | 122 | alfa-romero | std | two | convertible | rwd | front | 88.6 | 0.811148 | ... | 9.0 | 111.0 | 5000.0 | 21 | 27 | 13495.0 |
| 1 | 3 | 122 | alfa-romero | std | two | convertible | rwd | front | 88.6 | 0.811148 | ... | 9.0 | 111.0 | 5000.0 | 21 | 27 | 16500.0 |
| 2 | 1 | 122 | alfa-romero | std | two | hatchback | rwd | front | 94.5 | 0.822681 | ... | 9.0 | 154.0 | 5000.0 | 19 | 26 | 16500.0 |
| 3 | 2 | 164 | audi | std | four | sedan | fwd | front | 99.8 | 0.848630 | ... | 10.0 | 102.0 | 5500.0 | 24 | 30 | 13950.0 |
| 4 | 2 | 164 | audi | std | four | sedan | 4wd | front | 99.4 | 0.848630 | ... | 8.0 | 115.0 | 5500.0 | 18 | 22 | 17450.0 |

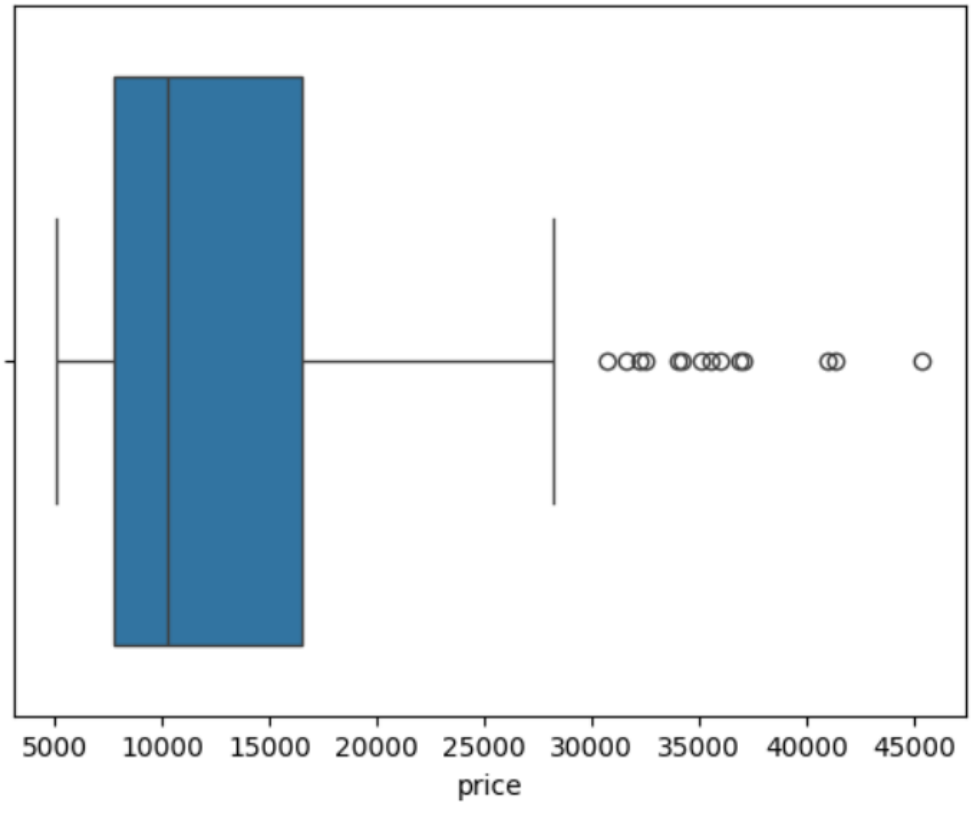
5 rows × 29 columns

## 2. Value Count

```
[144]: dataset['drive-wheels'].value_counts()

[144]: drive-wheels
       fwd    118
       rwd     75
       4wd      8
       Name: count, dtype: int64
```

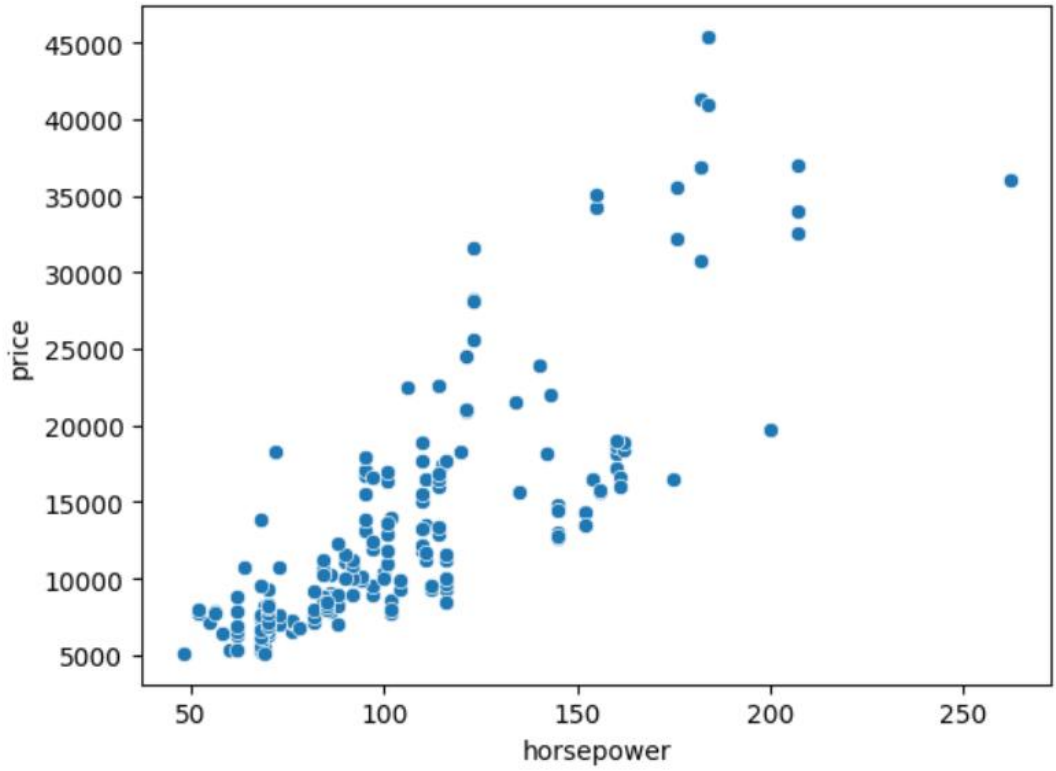
## 3. Box-Plot

```
[146]: sns.boxplot(x=dataset['price'])

[146]: <Axes: xlabel='price'>
```



## 4. Scatter Plot

```
[148]: sns.scatterplot(x=dataset['horsepower'],y=dataset['price']);
```




## 5. groupBy

```
[149]: ds1_test=dataset[['drive-wheels', 'body-style', 'price']]
       ds1_test.head()
```

[149]:

| | drive-wheels | body-style | price |
|---|---|---|---|
| **0** | rwd | convertible | 13495.0 |
| **1** | rwd | convertible | 16500.0 |
| **2** | rwd | hatchback | 16500.0 |
| **3** | fwd | sedan | 13950.0 |
| **4** | 4wd | sedan | 17450.0 |

```
[151]: ds_grp=ds1_test.groupby(['drive-wheels', 'body-style'],as_index=False).mean()
       ds_grp
```

[151]:

| | drive-wheels | body-style | price |
|---|---|---|---|
| 0 | 4wd | hatchback | 7603.000000 |
| 1 | 4wd | sedan | 12647.333333 |
| 2 | 4wd | wagon | 9095.750000 |
| 3 | fwd | convertible | 11595.000000 |
| 4 | fwd | hardtop | 8249.000000 |
| 5 | fwd | hatchback | 8396.387755 |

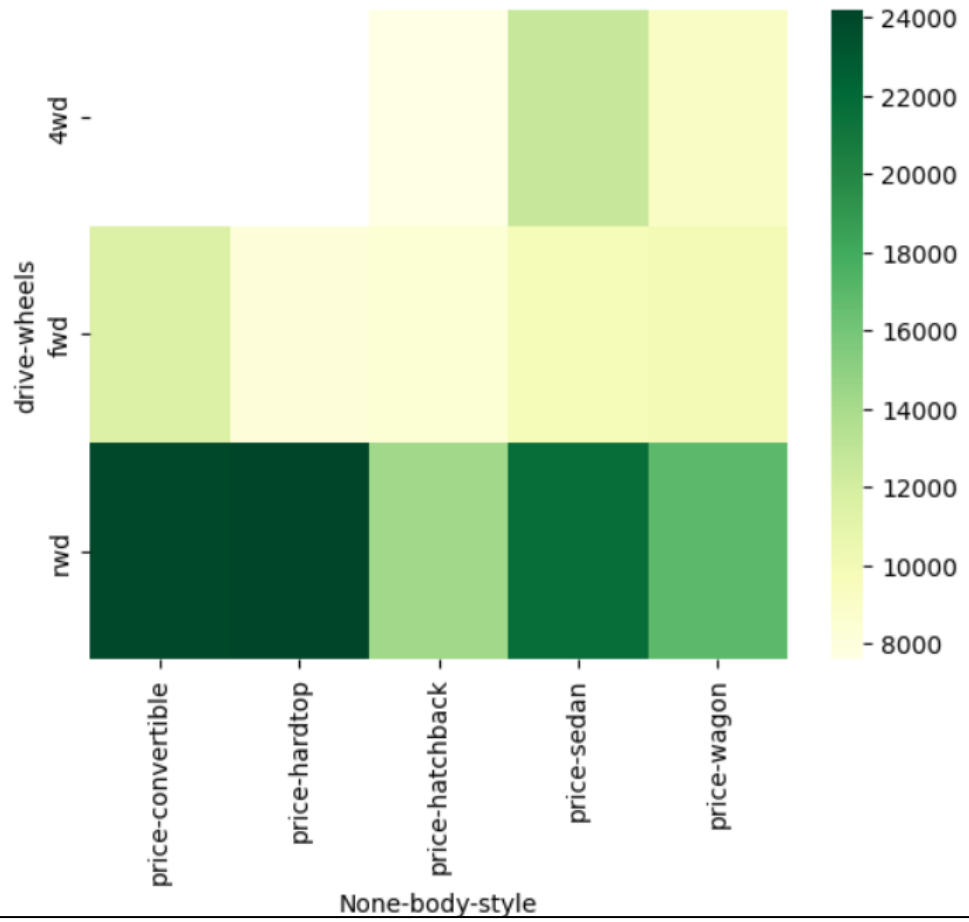
## 6. Pivot Table

```
[153]: ds_pivot=ds_grp.pivot(index='drive-wheels', columns='body-style')
       ds_pivot
```

[153]:

| | price | | | | |
|---|---|---|---|---|---|
| body-style | convertible | hardtop | hatchback | sedan | wagon |
| drive-wheels | | | | | |
| 4wd | NaN | NaN | 7603.000000 | 12647.333333 | 9095.750000 |
| fwd | 11595.0 | 8249.000000 | 8396.387755 | 9811.800000 | 9997.333333 |
| rwd | 23949.6 | 24202.714286 | 14337.777778 | 21711.833333 | 16994.222222 |

```
[158]: sns.heatmap(ds_pivot,cmap="YlGn")

[158]: <Axes: xlabel='None-body-style', ylabel='drive-wheels'>
```




## Conclusion:-

In conclusion, Exploratory Data Analysis (EDA) is an important part of any data study. By using simple tools like averages, charts, box plots, scatter plots, and tables, we can understand how the data is spread, find unusual values, and see how different pieces of data are related. EDA helps us clean the data, choose the right features, and prepare it for building models. It gives a clear picture of the data and makes it easier to make smart, data-based decisions.

# Practical no.5

## Linear Regression

**Aim 5A):** Study Model Selection, Model Evaluation using Visualization, Measures for In Sample Evaluation.

**Software Used:- Jupyter Notebook**

## Theory:-

Model selection is the process of picking the best machine learning model for a particular dataset and problem. The aim is to choose a model that works well on new, unseen data while keeping a balance between accuracy, simplicity, and computing resources. The usual steps include splitting the data into training and testing sets, trying different algorithms, and adjusting settings (hyperparameters) to improve performance.

Model evaluation checks how well a model performs using numbers and visual tools. For regression problems (predicting numbers), metrics like Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and $R^2$ score are used. For classification problems (predicting categories), metrics like accuracy, precision, recall, F1-score, and confusion matrix are common. Visual tools like ROC curves, precision-recall curves, and residual plots help us see how the model is performing and where it can be improved.

### 1. Import Library and Load Data

[1]:
```
import pandas as pd
import seaborn as sns
import numpy as np
from sklearn.linear_model import LinearRegression
```

[2]:
```
ds=pd.read_csv("C:\\Users\\Priyanshu\\Downloads\\DATASETS (5)\\DATASETS\\automobile.csv")
ds.head()
```

[2]:

| | symboling | normalized-losses | make | aspiration | num-of-doors | body-style | drive-wheels | engine-location | wheel-base | length | ... | compression-ratio | horsepower | peak-rpm | city-mpg | highway-mpg | price |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 3 | 122 | alfa-romero | std | two | convertible | rwd | front | 88.6 | 0.811148 | ... | 9.0 | 111.0 | 5000.0 | 21 | 27 | 13495.0 |
| **1** | 3 | 122 | alfa-romero | std | two | convertible | rwd | front | 88.6 | 0.811148 | ... | 9.0 | 111.0 | 5000.0 | 21 | 27 | 16500.0 |
| **2** | 1 | 122 | alfa-romero | std | two | hatchback | rwd | front | 94.5 | 0.822681 | ... | 9.0 | 154.0 | 5000.0 | 19 | 26 | 16500.0 |
| **3** | 2 | 164 | audi | std | four | sedan | fwd | front | 99.8 | 0.848630 | ... | 10.0 | 102.0 | 5500.0 | 24 | 30 | 13950.0 |
| **4** | 2 | 164 | audi | std | four | sedan | 4wd | front | 99.4 | 0.848630 | ... | 8.0 | 115.0 | 5500.0 | 18 | 22 | 17450.0 |

5 rows × 29 columns

## 2. Linear Regression

```
[3]: from sklearn.linear_model import LinearRegression
     LR=LinearRegression()
     x=ds[['highway-mpg']]
     y=ds[['price']]
     LR.fit(x,y)
```

[3]: LinearRegression

▼ Parameters

| | |
|---|---|
| fit_intercept | True |
| copy_X | True |
| tol | 1e-06 |
| n_jobs | None |
| positive | False |

```
[5]: LR2=LinearRegression()
     x1=ds[['highway-mpg']]
     y1=ds[['price']]
     LR2.fit(x1,y1)
```

[5]: LinearRegression

▼ Parameters

| | |
|---|---|
| fit_intercept | True |
| copy_X | True |
| tol | 1e-06 |
| n_jobs | None |
| positive | False |

```
[6]: LR2.predict(x)
```

```
[6]: array([[16236.50464347],
            [16236.50464347],
            [17058.23802179],
            [13771.3045085 ],
            [20345.17153508],
            [17879.97140011],
            [17879.97140011],
```

## 3. Multiple Regression

```
[8]: intercept=LR2.intercept_
     slope =LR2.coef_
     print("intercept:", intercept, "\nslope:",slope)
```

```
intercept: [38423.30585816]
slope: [[-821.73337832]]
```

```
[9]: desiredMPG=int(input("Enter highway-mpg for preiciting price:"))
     y=slope*desiredMPG + intercept
     print(y)
```

```
Enter highway-mpg for preiciting price: 45
[[1445.30383367]]
```

```
[10]: MLR=LinearRegression()
      x=ds[['highway-mpg', 'horsepower']]
      y=ds['price']
      MLR.fit(x,y)
      MLR.predict(x)
```

```
[10]: array([14968.32084179, 14968.32084179, 21435.83465725, 13122.55666208,
            16435.22322519, 15174.68495437, 15174.68495437, 15174.68495437,
            20445.55574556, 13152.58547976, 13152.58547976, 16255.05031911,
            16255.05031911, 16784.05620382, 26237.75719909, 26237.75719909,
```

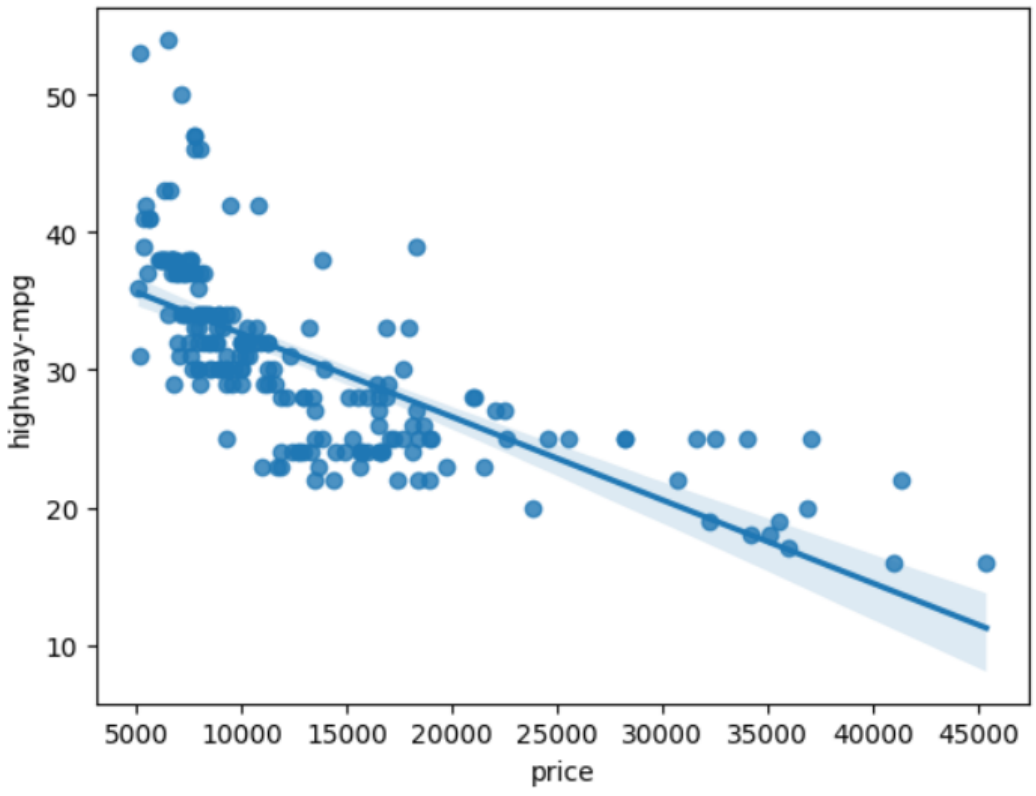
```
[11]: intercept = MLR.intercept_
      slope=MLR.coef_
      print("intercept: ", intercept, "\nslope:", slope)
```

```
intercept:  3489.3548324887633
slope: [-176.3352949   146.30647722]
```

```
[15]: desired_mpg=int(input("Enter a highway-mpg for Predicting Price: "))
      horsepower=int(input("Enter a horsepower for Predicting Price"))
      y=(slope[0]*desired_mpg+intercept)+(slope[1]*desired_mpg+intercept)
      print(y)

      Enter a highway-mpg for Predicting Price:  38
      Enter a horsepower for Predicting Price 50
      5837.614593107461
```

## 4.Regression Plot

```
[16]: import seaborn as sns
      sns.regplot(x='price',y='highway-mpg',data=ds)

[16]: <Axes: xlabel='price', ylabel='highway-mpg'>
```




## Conclusion:-

To conclude, choosing the right model and checking its performance are very important steps in creating effective machine learning systems. Picking the right model ensures it fits the problem well, and carefully evaluating it with metrics and charts shows how well it works on both the training data and new, unseen data. Using both in-sample and out-of-sample evaluation helps avoid overfitting, makes the model more reliable, and allows it to make accurate predictions in real-world situations.

# Practical no.6

# Logistic regression

**Aim 7A):** Implement Regression – Logistic, Linear Regression.

**Software Used:- Jupyter Notebook**

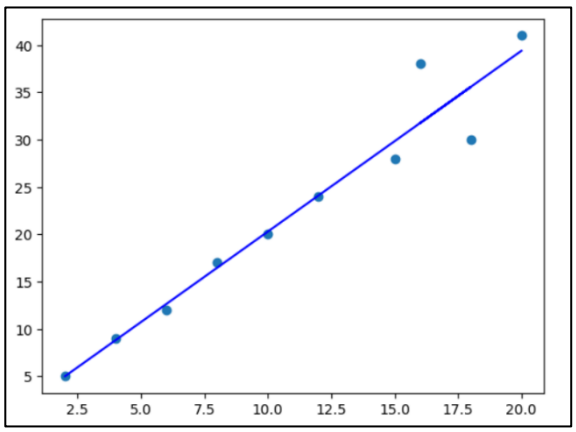
**Theory: -**

**Linear Regression:**
Linear Regression is a statistical method used to predict a continuous value based on one or more input variables. It finds the best-fitting straight line (called the regression line) that shows the relationship between the dependent and independent variables. For example, it can predict house prices based on area or number of rooms.

**Logistic Regression:**
Logistic Regression is used when the output variable is **categorical**, such as "Yes/No" or "Pass/Fail." Instead of fitting a straight line, it uses a **sigmoid curve** to predict the probability of an outcome. It is commonly used in classification problems like spam detection or disease prediction.

## 1. Linear Regression

```
[17]: import matplotlib.pyplot as plt
      from scipy import stats

[20]: x =[2,4,6,8,10,12,15,18,16,20]
      y=[5,9,12,17,20,24,28,30,38,41]
      slope, intercept,r,p,std_err=stats.linregress(x,y)

      #print the result
      print("r-value:", r)
      print("p-value:",p)
      print("Standard error:",std_err)

      def myfunc(x):
          return slope * x + intercept

      mymodel=list(map(myfunc,x))

      #plot data points
      plt.scatter(x,y)

      #plot regression line
      plt.plot(x, mymodel, color='blue')

      r-value: 0.9701244362524559
      p-value: 3.361916417243901e-06
      Standard error: 0.16890693074833715
[20]: [<matplotlib.lines.Line2D at 0x18f1ba9ad50>]
```



## 2. Logistic Regression

```
[4]: import numpy as np
     from sklearn import linear_model

     # Sizes of tumors in centimeters (feature)
     X = np.array([3.78, 2.44, 2.09, 0.14, 1.72, 1.65, 4.92, 4.37, 4.96, 4.52, 3.69, 5.88]).reshape(-1, 1)

     # Binary target variable (0: benign / not cancerous, 1: malignant / yes)
     y = np.array([0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1])

     # Initialize logistic regression model
     logr = linear_model.LogisticRegression()

     # Train the model
     logr.fit(X, y)

     predicted = logr.predict(np.array([3.46]).reshape(-1, 1))
     print(predicted)

     # Display the prediction result
     print(f"The tumor of size 3.46 cm is predicted to be: {'malignant (Yes)' if predicted[0] == 1 else 'benign (No)'}")
```

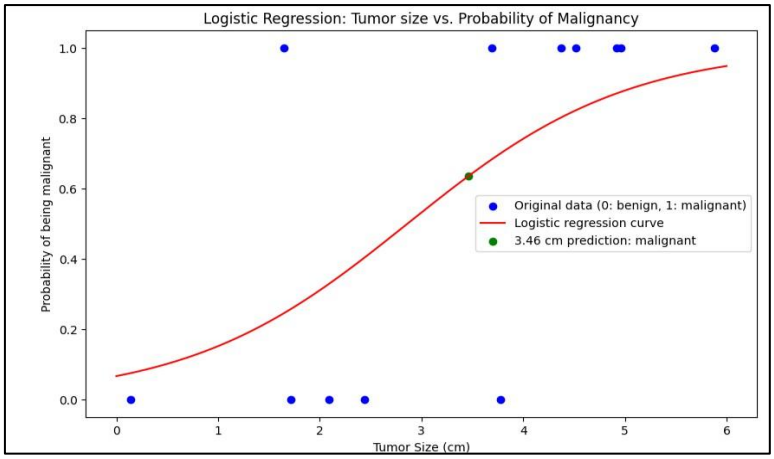
```
[1]
The tumor of size 3.46 cm is predicted to be: malignant (Yes)
```

## 3. Logistic Regression on plot

```
[6]: import numpy as np
     import matplotlib.pyplot as plt
     from sklearn import linear_model

     # Sizes of tumors in centimeters (feature)
     X = np.array([3.78, 2.44, 2.09, 0.14, 1.72, 1.65, 4.92, 4.37, 4.96, 4.52, 3.69, 5.88]).reshape(-1, 1)

     # Binary target variable (0: benign, 1: malignant)
     y = np.array([0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1])

     # Initialize the logistic regression model
     logr = linear_model.LogisticRegression()

     # Train the model
     logr.fit(X, y)

     # Predict probabilities for the input data
     probs = logr.predict_proba(X)[:, 1]

     # Predict for a specific value (3.46 cm)
     new_data = np.array([3.46]).reshape(-1, 1)
     predicted = logr.predict(new_data)
     predicted_prob = logr.predict_proba(new_data)[:, 1]

     print(f"Prediction for tumor size 3.46 cm: {'malignant' if predicted[0] == 1 else 'benign'}")
     print(f"Probability of being malignant: {predicted_prob[0]:.4f}")
```

```
# Plot the data and the logistic regression curve
plt.figure(figsize=(10, 6))

# Scatter plot of the original data
plt.scatter(X, y, color='blue', label='Original data (0: benign, 1: malignant)')

# Plot the logistic regression curve
x_values = np.linspace(0, 6, 300).reshape(-1, 1)
y_values = logr.predict_proba(x_values)[:, 1]
plt.plot(x_values, y_values, color='red', label='Logistic regression curve')

# Highlight the specific prediction point
plt.scatter(new_data, predicted_prob, color='green', marker='o',
            label=f'3.46 cm prediction: {"malignant" if predicted[0] == 1 else "benign"}')

# Labels and title
plt.xlabel('Tumor Size (cm)')
plt.ylabel('Probability of being malignant')
plt.title('Logistic Regression: Tumor size vs. Probability of Malignancy')
plt.legend()

# Show the plot
plt.show()
```

```
Prediction for tumor size 3.46 cm: malignant
Probability of being malignant: 0.6344
```



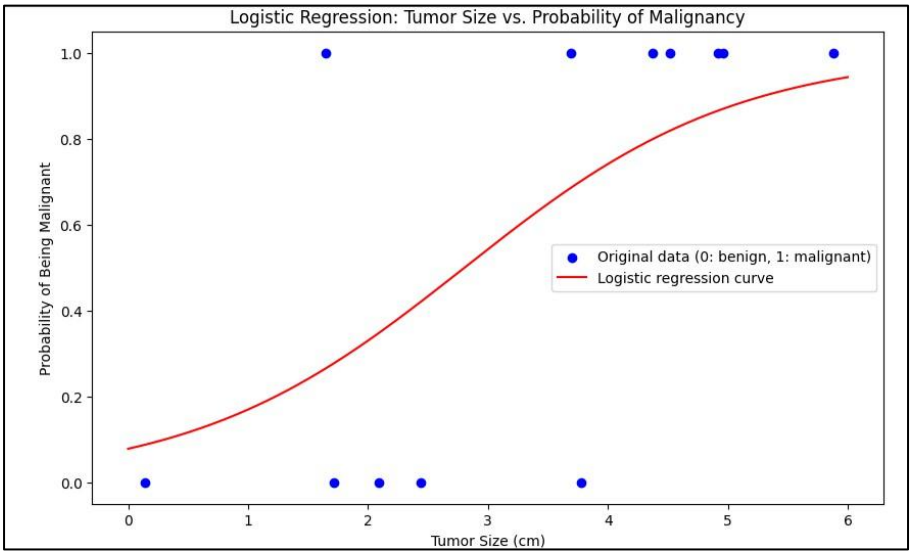

## 4. Logistic Regression-on unseen Data

```
[7]: import numpy as np
import matplotlib.pyplot as plt
from sklearn import linear_model
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report

# Sizes of tumors in centimeters (feature)
x = np.array([3.78, 2.44, 2.09, 0.14, 1.72, 1.65, 4.92, 4.37, 4.96, 4.52, 3.69, 5.88]).reshape(-1, 1)

# Binary target variable (0: benign, 1: malignant)
y = np.array([0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1])

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=42)

# Initialize the logistic regression model
logr = linear_model.LogisticRegression()

# Train the model
logr.fit(X_train, y_train)

# Predict on the testing set
y_pred = logr.predict(X_test)

# Evaluate the model
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```

```
# Plot the data and the logistic regression curve
plt.figure(figsize=(10, 6))

plt.scatter(X, y, color='blue', label='Original data (0: benign, 1: malignant)')

x_values = np.linspace(0, 6, 300).reshape(-1, 1)
y_values = logr.predict_proba(x_values)[:, 1]
plt.plot(x_values, y_values, color='red', label='Logistic regression curve')

# Add labels and title
plt.xlabel('Tumor Size (cm)')
plt.ylabel('Probability of Being Malignant')
plt.title('Logistic Regression: Tumor Size vs. Probability of Malignancy')
plt.legend()

# Show the plot
plt.show()

# Predict on new unseen data
unseen_data = np.array([3.46, 4.0, 2.5]).reshape(-1, 1)
unseen_predictions = logr.predict(unseen_data)
unseen_probabilities = logr.predict_proba(unseen_data)[:, 1]

print(f"Predictions for unseen data: {unseen_predictions}")
print(f"Probabilities for unseen data: {unseen_probabilities}")
```

```
Accuracy: 0.75
Classification Report:
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         1
           1       0.75      1.00      0.86         3

    accuracy                           0.75         4
   macro avg       0.38      0.50      0.43         4
weighted avg       0.56      0.75      0.64         4
```




## 5. Logistic Regression-function to predict based on user input

```
Predictions for unseen data: [1 1 0]
Probabilities for unseen data: [0.64021474 0.74110291 0.4331817 ]
```

[8]:
```python
import numpy as np
from sklearn import linear_model

# Sizes of tumors in centimeters (feature)
X = np.array([3.78, 2.44, 2.09, 0.14, 1.72, 1.65, 4.92, 4.37, 4.96, 4.52, 3.69, 5.88]).reshape(-1, 1)

# Binary target variable (0: benign, 1: malignant)
y = np.array([0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1])

# Initialize the logistic regression model
logr = linear_model.LogisticRegression()

# Train the model
logr.fit(X, y)

# Function to predict based on user input
def predict_tumor_size():
    # Get user input
    tumor_size = float(input("Enter tumor size in cm: "))

    # Reshape and predict
    input_data = np.array([[tumor_size]])
    prediction = logr.predict(input_data)
    probability = logr.predict_proba(input_data)[:, 1]
```

```python
    # Print the result
    result = 'malignant' if prediction[0] == 1 else 'benign'
    print(f"The tumor of size {tumor_size} cm is predicted to be: {result}")
    print(f"Probability of being malignant: {probability[0]}")

# Call the function
predict_tumor_size()
```

```
Enter tumor size in cm:  3
The tumor of size 3.0 cm is predicted to be: malignant
Probability of being malignant: 0.5312606157091658
```

## Conclusion:-

- The **Linear Regression** model clearly showed how we can predict a continuous value (such as car price or speed) using a straight-line relationship. The regression line helped visualize the connection between the input (X) and output (Y), highlighting a strong correlation between them.
- The **Logistic Regression** model successfully categorized tumor data as either benign or malignant based on size. It provided both class predictions and probability scores, showing how the likelihood of malignancy increases with tumor size.

# Practical no.7

# Data Visualization using Tableau

**Aim 6A):** Implement Data Visualization using Tableau.

## Software Used:- Tableau Public Desktop

## Theory:-

**Data Visualization** is the process of showing data using **graphs, charts, maps, or dashboards** so that it becomes easier to understand and analyze. It helps in quickly spotting **patterns, trends, and insights** from raw data.

**Tableau** is a widely used and powerful tool for data visualization in analytics and business intelligence. It lets users create **interactive and attractive visualizations** without needing any programming knowledge. Tableau can connect to many types of data sources such as **Excel, CSV files, databases, and cloud services**, and it also provides options to clean, filter, and analyze data.

With Tableau, users can simply drag and drop data fields to create visuals like:

1. **Bar Charts**
2. **Pie Charts**
3. **Line Graphs**
4. **Scatter Plots**
5. **Maps**
6. **Dashboards**

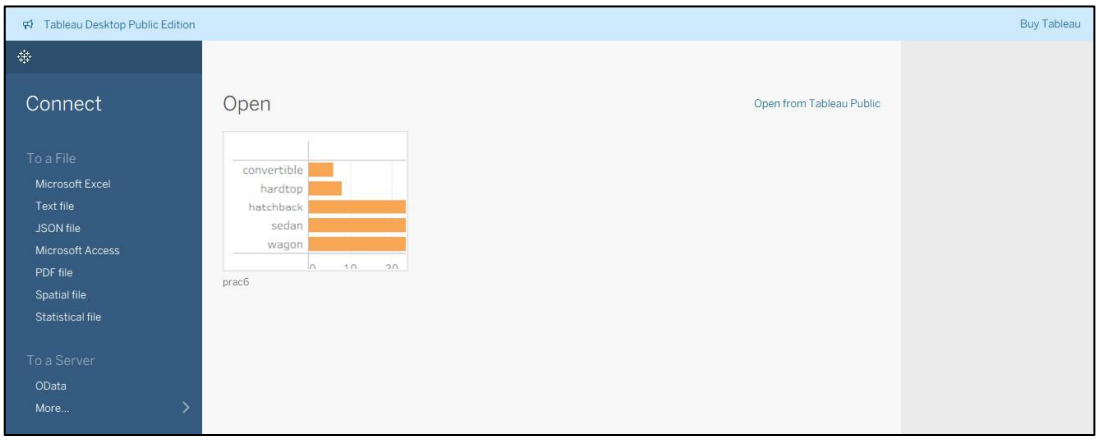
The main parts of Tableau are:

1. **Tableau Desktop** – To create reports and dashboards.
2. **Tableau Public** – A free version to create and share visualizations online.
3. **Tableau Server** – For sharing and managing reports within organizations.
4. **Tableau Reader** – To view reports offline created by others.

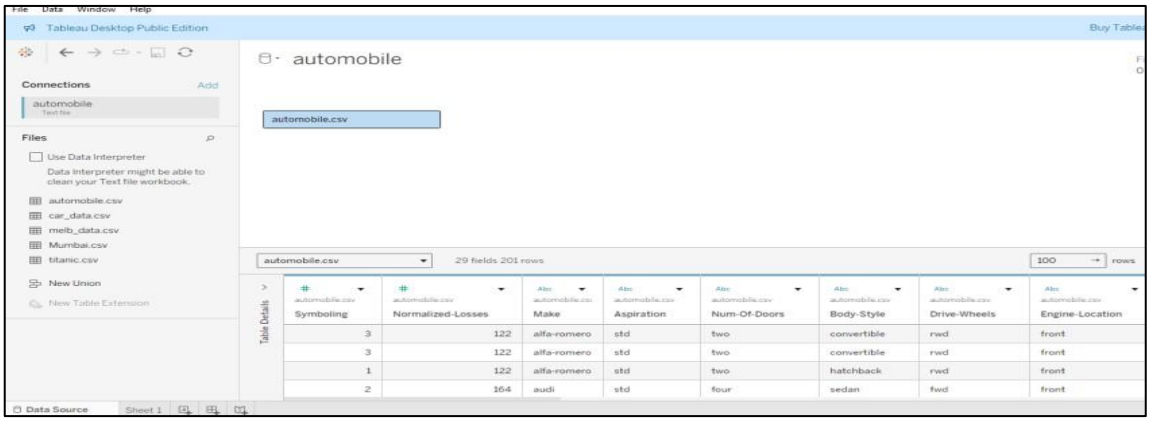
## Installation Process:

**Step 1:** Go to the Tableau Public website.
**Step 2:** Click on **"Create on the Web"** and then select **Download Tableau Public Edition**.
**Step 3:** Open the downloaded .exe file.
**Step 4:** Review the installation settings and then click **Install** to complete the setup.

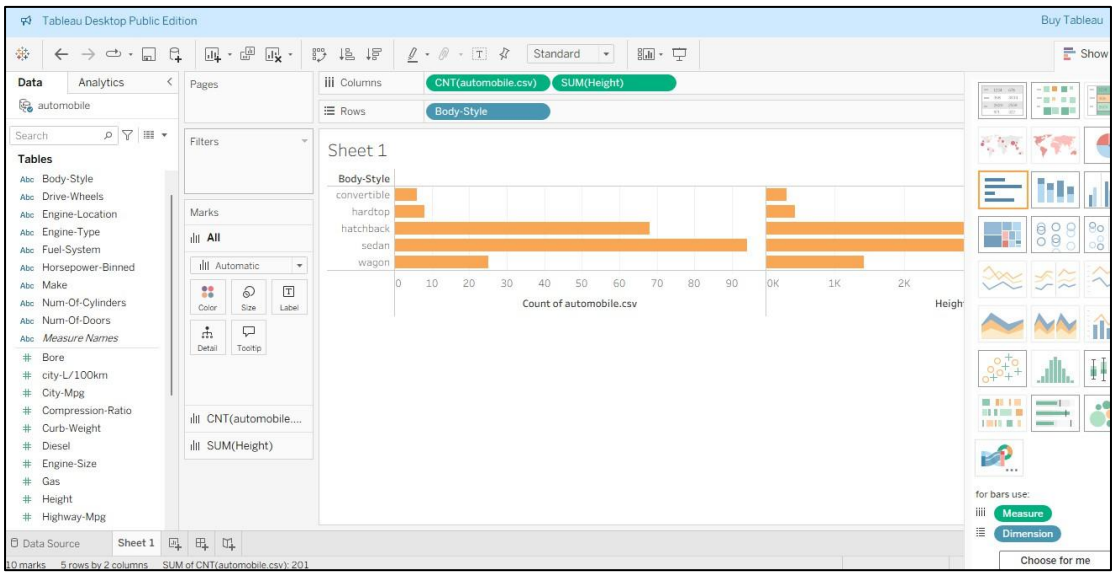
**Import Excel Data File:** We will use a sample Excel data file that is available on the Tableau Public website.



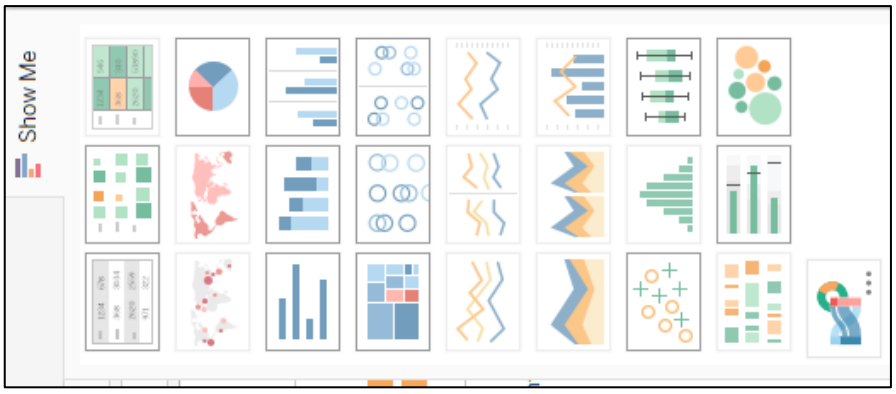

Select any type of data file and import it into Tableau. Once the file is imported, all the data will be displayed, as shown in the figure below.




Click below to **create a new sheet**. Then, add rows and columns by **dragging and dropping the desired fields** into the sheet.

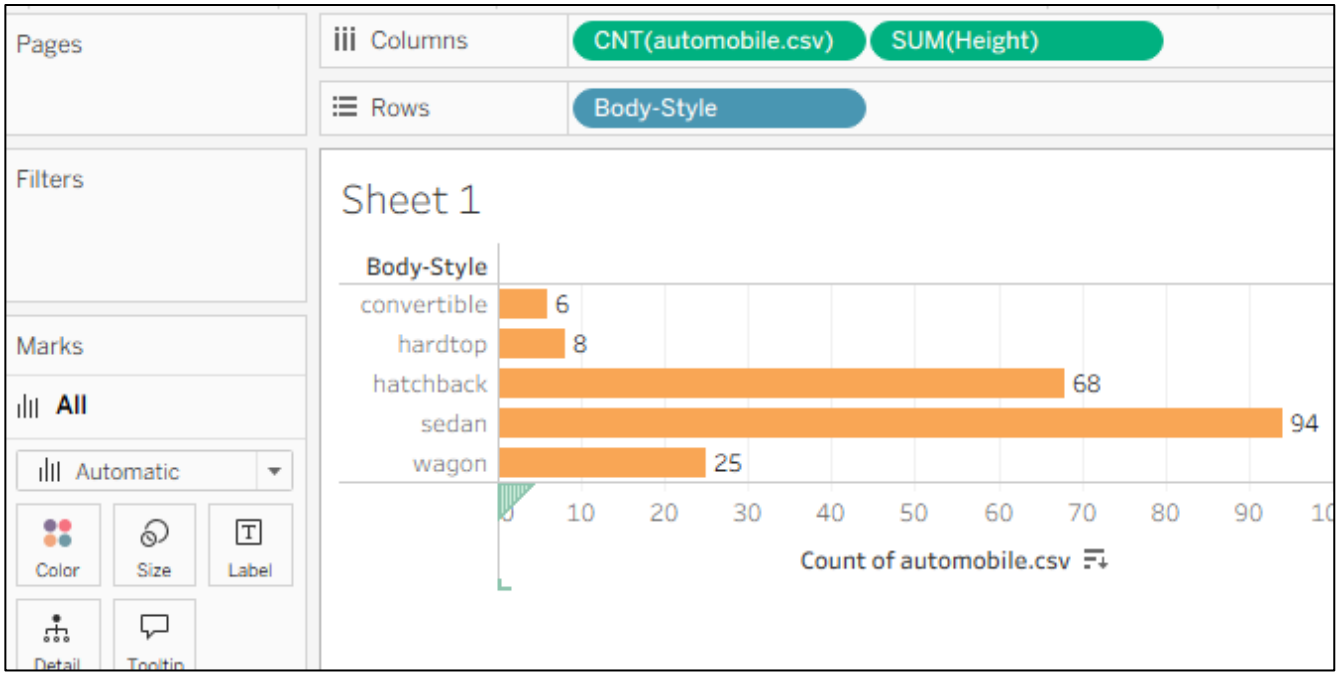


You can **change the chart type** according to your preference by clicking on the **"Show Me"** button

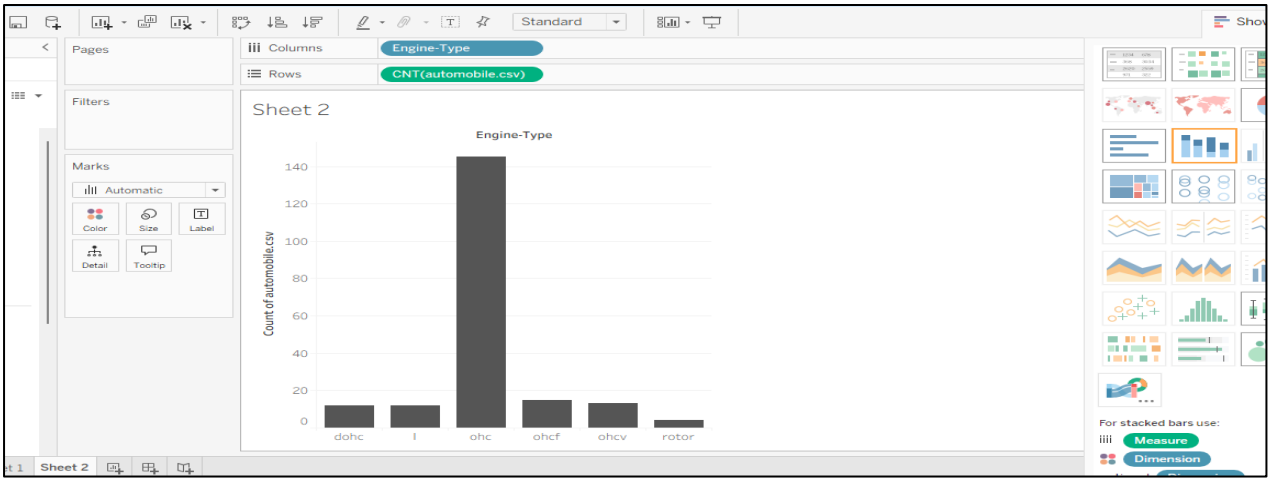



**Add Labels:**

Go to the **Marks** section, click on **All**, and then click the **Labels** button. This will add labels to the selected rows and columns in your chart.

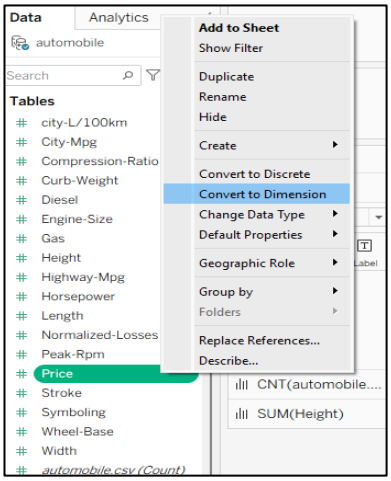


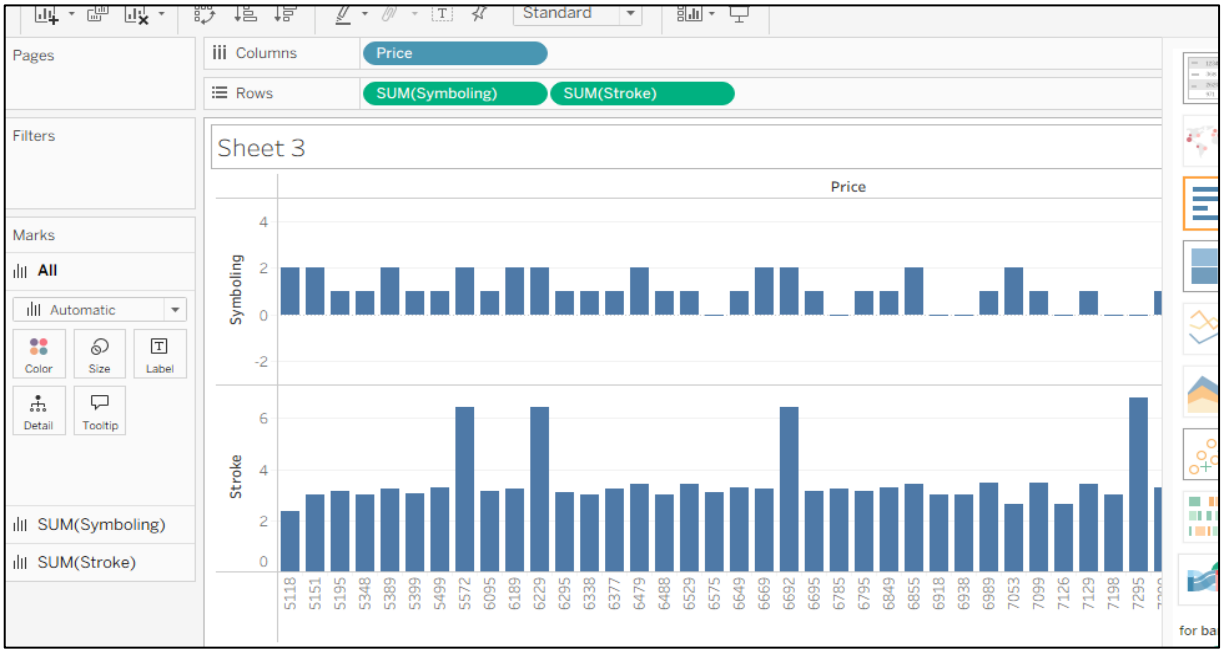

Replace the existing variable by removing the **Age** variable and adding a different one in its place.



**Add Price to Columns:**

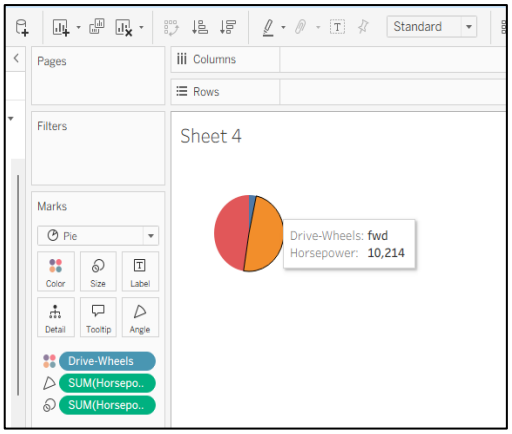
Hold the **Ctrl key** and **left-click** on the **Price** field, then drag it to the **Columns** section and expand it to view the details clearly.

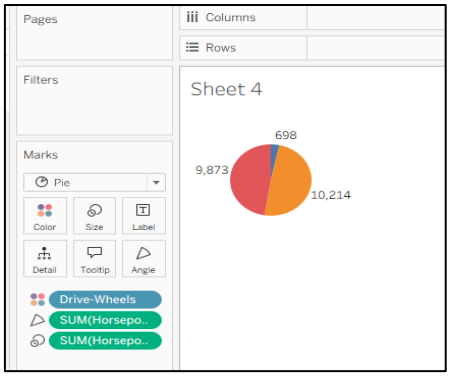



## Pie chart

Go to the “Show Me” panel and select the Pie Chart option to display your data in a pie chart format. Add the labels to the visuals using drag and drop



## Conclusion: -

Tableau makes data visualization simple, quick, and powerful. It helps users analyze large amounts of data easily by creating different types of interactive charts, graphs, and dashboards. Through Tableau, we can identify patterns, compare values, and understand data behavior visually rather than relying only on numbers. It also supports connections to various data sources like Excel, databases, and cloud platforms, making analysis flexible and efficient.

By using Tableau, decision-making becomes faster and more accurate because it turns complex data into clear and meaningful visuals. It is an essential tool for students, analysts, and organizations to explore, explain, and present data effectively. Overall, Tableau plays a key role in transforming raw data into valuable insights that support real-world business and research goals.

# Practical no.8

# Classification Techniques

**Aim 8A):** Implement Classification, Decision Trees, Random Forest.

## Software Used:- Jupyter Notebook

## Theory: -

**Classification:**
Classification is a type of **supervised machine learning** used to categorize data into predefined classes or groups. It learns patterns from labeled data and predicts the class of new, unseen data. Common examples include email spam detection, disease diagnosis, and sentiment analysis.

**Decision Tree:**
A **Decision Tree** is a model that splits data into branches based on certain conditions or features. Each internal node represents a decision based on a feature, each branch represents an outcome, and each leaf node represents a final class label. Decision Trees are easy to understand and interpret but may overfit if not pruned properly.

**Random Forest:**
A **Random Forest** is an advanced ensemble method that builds **multiple decision trees** and combines their results for better accuracy and stability. It reduces overfitting and improves performance by averaging the predictions of several trees. It is widely used in classification and regression tasks due to its reliability and efficiency.

1. We will use the breast cancer dataset from scikit-learn

```
[11]: #import librarys (Kru)
      import matplotlib.pyplot as plt
      from sklearn.datasets import load_breast_cancer
      from sklearn.model_selection import train_test_split

[12]: dataset=load_breast_cancer(as_frame=True)
      dataset['data'].head()
```

[12]:

| | mean radius | mean texture | mean perimeter | mean area | mean smoothness | mean compactness | mean concavity | mean concave points | mean symmetry | mean fractal dimension | ... | worst radius | worst texture | worst perimeter | worst area | worst smoothness | comp |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 17.99 | 10.38 | 122.80 | 1001.0 | 0.11840 | 0.27760 | 0.3001 | 0.14710 | 0.2419 | 0.07871 | ... | 25.38 | 17.33 | 184.60 | 2019.0 | 0.1622 | |
| 1 | 20.57 | 17.77 | 132.90 | 1326.0 | 0.08474 | 0.07864 | 0.0869 | 0.07017 | 0.1812 | 0.05667 | ... | 24.99 | 23.41 | 158.80 | 1956.0 | 0.1238 | |
| 2 | 19.69 | 21.25 | 130.00 | 1203.0 | 0.10960 | 0.15990 | 0.1974 | 0.12790 | 0.2069 | 0.05999 | ... | 23.57 | 25.53 | 152.50 | 1709.0 | 0.1444 | |
| 3 | 11.42 | 20.38 | 77.58 | 386.1 | 0.14250 | 0.28390 | 0.2414 | 0.10520 | 0.2597 | 0.09744 | ... | 14.91 | 26.50 | 98.87 | 567.7 | 0.2098 | |
| 4 | 20.29 | 14.34 | 135.10 | 1297.0 | 0.10030 | 0.13280 | 0.1980 | 0.10430 | 0.1809 | 0.05883 | ... | 22.54 | 16.67 | 152.20 | 1575.0 | 0.1374 | |

5 rows × 30 columns

2.Split the dataset into training and testing sets

```
[13]: dataset['data'].shape

[13]: (569, 30)

[14]: dataset['target'].head()

[14]: 0    0
      1    0
      2    0
      3    0
      4    0
      Name: target, dtype: int64

[15]: dataset['target'].value_counts()

[15]: target
      1    357
      0    212
      Name: count, dtype: int64

[24]: x=dataset['data']
      y=dataset['target']

[25]: # Split the dataset
      x_train, y_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=1)
```

3. Normalize the data for numerical stability

```
[26]: from sklearn.preprocessing import StandardScaler

[28]: ss_train = StandardScaler()
      x_train = ss_train.fit_transform(x_train)
      ss_test = StandardScaler()
      X_test = ss_test.fit_transform(X_test)
```

4. Fit a logistic regression model to the training data

```
[29]: from sklearn.linear_model import LogisticRegression
      logistic_classifier = LogisticRegression()
      logistic_classifier.fit(X_train,y_train)
```

[29]: LogisticRegression

▼ Parameters

| | |
|---|---|
| penalty | 'l2' |
| dual | False |
| tol | 0.0001 |
| C | 1.0 |
| fit_intercept | True |
| intercept_scaling | 1 |
| class_weight | None |
| random_state | None |
| solver | 'lbfgs' |
| max_iter | 100 |
| multi_class | 'deprecated' |
| verbose | 0 |

**5.Make prediction on the testing data**

```
[30]: y_pred=logistic_classifier.predict(X_test)

[32]: print(y_pred[0:5])
      print(y_test[0:5])

      [1 0 1 0 0]
      421    1
      47     0
      292    1
      186    0
      414    0
      Name: target, dtype: int64
```

6.Calculate the accuracy score by comparing the actual value and predicted values.

```
[33]: from sklearn.metrics import confusion_matrix

[34]: cm = confusion_matrix(y_test, y_pred)

[36]: TN, FP, FN, TP = confusion_matrix(y_test, y_pred).ravel()

[37]: print('True Positive(TP) = ', TP)
      print('True Positive(FP) = ', FP)
      print('True Positive(TN) = ', TN)
      print('True Positive(FN) = ', FN)

      True Positive(TP) =  103
      True Positive(FP) =  3
      True Positive(TN) =  60
      True Positive(FN) =  5
```

Initializing each binary classification to quickly train each module in loop, we'll initialize each mode and store it by name in a dictionary.

```
[38]: from sklearn import metrics
      import matplotlib.pyplot as plt
      print("Logistic Regression's Accuracy: ", metrics.accuracy_score(y_test,y_pred))

      Logistic Regression's Accuracy:  0.9532163742690059

[40]: #Another way to find accuracy
      accuracy= (TP+TN)/(TP+FP+TN+FN)
      print("Logistic Regression Accuracy: = {:0.3f}".format(accuracy))

      Logistic Regression Accuracy: = 0.953

[68]: models={}
      #Logistic Regression
      from sklearn.linear_model import LogisticRegression
      models['Logistic Regression']= LogisticRegression()

      #Support Vector Machines
      from sklearn.svm import LinearSVC
      models['Support Vector Machines']= LinearSVC()

      #Decision Tree
      from sklearn.tree import DecisionTreeClassifier
      models['Decision Tree']= DecisionTreeClassifier()

      #Random Forest
      from sklearn.ensemble import RandomForestClassifier
      models['Random Forest']= RandomForestClassifier()
```

```
#Naive Bayes
from sklearn.naive_bayes import GaussianNB
models['Naive Bayes']= GaussianNB()

#K-Nearest Neighbors
from sklearn.neighbors import KNeighborsClassifier
models['K-Nearest Neighbors']= KNeighborsClassifier()
```

```
[72]: from sklearn.metrics import accuracy_score, precision_score, recall_score
accuracy, precision, recall = {}, {}, {}
for key in models.keys():
    models[key].fit(X_train, y_train)
    predictions= models[key].predict(X_test)
    accuracy[key]=accuracy_score(predictions, y_test)
    precision[key]=precision_score(predictions, y_test)
    recall[key]=recall_score(predictions, y_test)
```

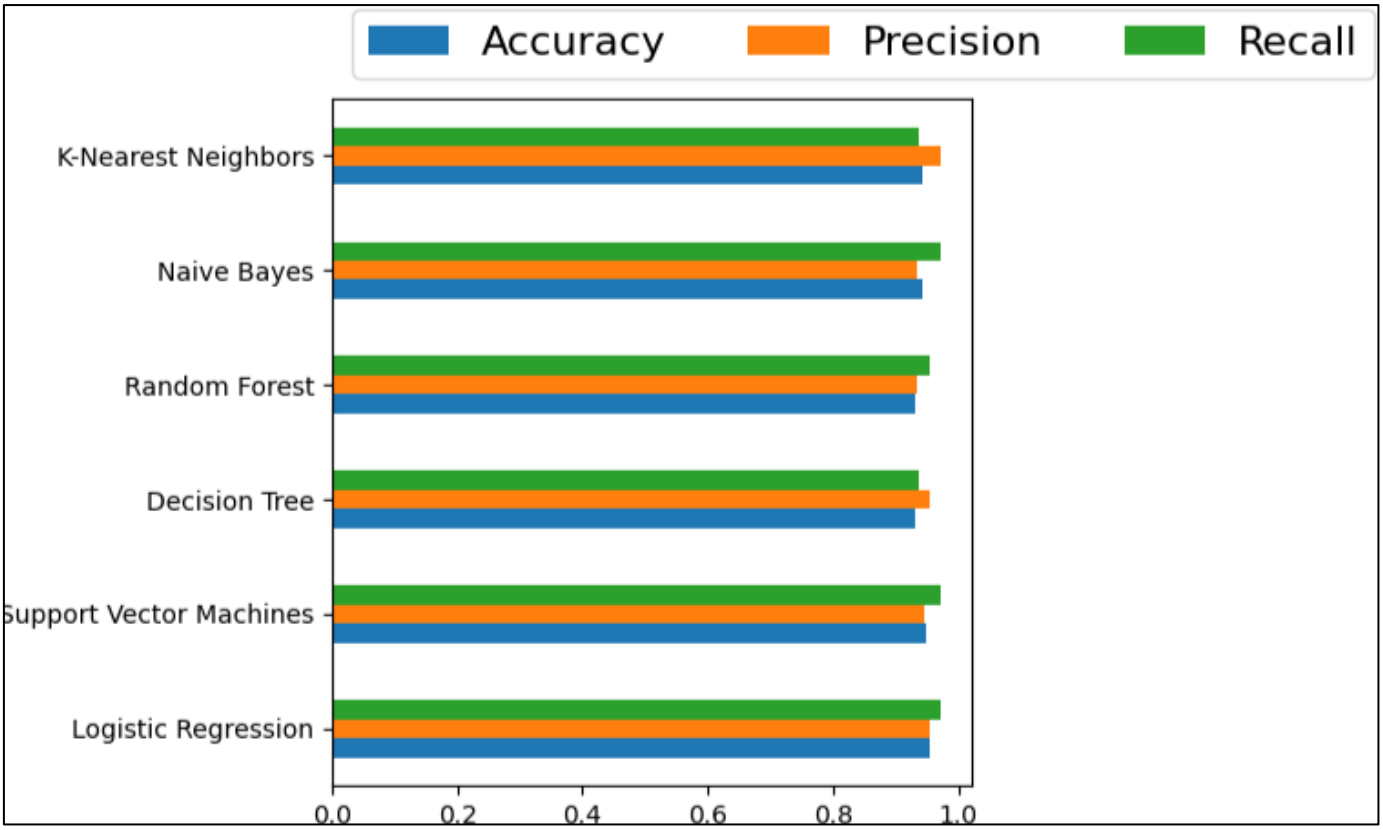
With all metrics stored, we can use pandas to view the data as a table:

```
[74]: import pandas as pd
df_model=pd.DataFrame(index=models.keys(), columns=['Accuracy','Precision','Recall'])
df_model['Accuracy']=accuracy.values()
df_model['Precision']=precision.values()
df_model['Recall']=recall.values()
df_model
```

[74]:

| | Accuracy | Precision | Recall |
|---|---|---|---|
| **Logistic Regression** | 0.953216 | 0.953704 | 0.971698 |
| **Support Vector Machines** | 0.947368 | 0.944444 | 0.971429 |
| **Decision Tree** | 0.929825 | 0.953704 | 0.936364 |
| **Random Forest** | 0.929825 | 0.935185 | 0.952830 |
| **Naive Bayes** | 0.941520 | 0.935185 | 0.971154 |
| **K-Nearest Neighbors** | 0.941520 | 0.972222 | 0.937500 |

```
[79]: ax = df_model.plot.barh()
ax.legend(
    ncol=len(models.keys()),
    bbox_to_anchor=(0, 1),
    loc='lower left',
    prop={'size': 16}
)

plt.tight_layout()
```




## Conclusion:

The practical on **Classification using Decision Tree and Random Forest** was successfully completed. Through this exercise, we learned how to **train, test, and evaluate** classification models using key performance measures such as **accuracy, confusion matrix, and classification report**. The **Decision Tree** algorithm helps visualize and understand the step-by-step process of how decisions are made based on input features. On the other hand, the **Random Forest** technique improves prediction accuracy and reliability by combining multiple decision trees to make a final decision. This practical enhanced our understanding of supervised learning techniques and their importance in solving real-world problems like prediction, pattern recognition, and data classification.

# Practical no.9

# Clustering

**Aim 9A):** Apply K-Means and Hierarchical Clustering with visualization

## Software Used:- Jupyter Notebook

## Theory:-

### Clustering

Clustering is an **unsupervised machine learning** technique used to group similar data points together based on their features, without using any predefined labels. The goal is to discover hidden patterns, structures, or relationships within the dataset. It is widely used in **customer segmentation, image analysis, market research, and pattern discovery**.

One of the most popular clustering methods is **K-Means**, which divides data into **K clusters** by minimizing the distance between points and the cluster center (centroid). Other clustering methods include **Hierarchical Clustering** and **DBSCAN**, each with different approaches to forming clusters.

Clustering helps in **analyzing large datasets**, identifying trends, and making data-driven decisions by organizing data into meaningful groups.

1. **Kmeans**

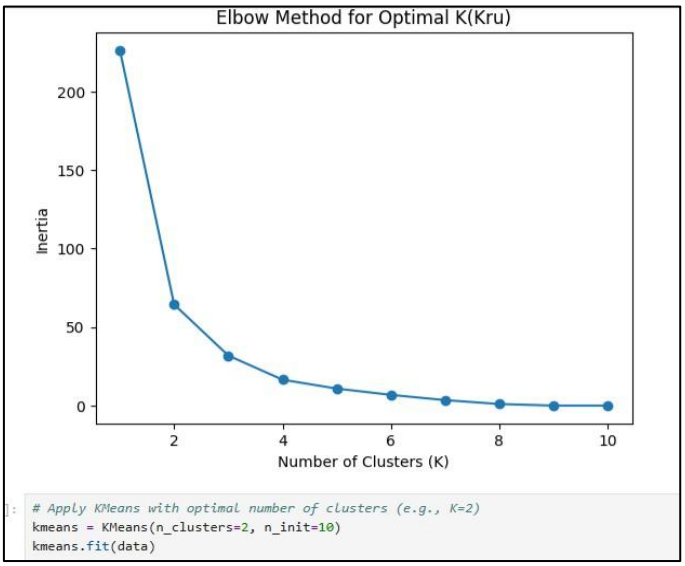
In this code, the Matplotlib and Kmeans libraries are imported, then some sample data points (x and y) are defined and combined into coordinate pairs (like (x, y)). These pairs will be used later for clustering or plotting.

```
[11]: #import library
      import matplotlib.pyplot as plt
      from sklearn.cluster import KMeans

[6]:  # Sample data points
      x = [5, 8, 10, 4, 2, 11, 13, 6, 10, 14]
      y = [21, 20, 24, 18, 16, 26, 24, 22, 24, 22]

[7]:  # Combine into coordinate pairs
      data = list(zip(x, y))
      print("Data points:", data)

      Data points: [(5, 21), (8, 20), (10, 24), (4, 18), (2, 16), (11, 26), (13, 24), (6, 22), (10, 24), (14, 22)]
```

This code uses the **Elbow Method** to find the best number of clusters for K-Means by plotting the number of clusters (K) against inertia values. The point where the curve bends, called the **elbow point**, indicates the optimal number of clusters for the dataset.

```
[8]: # Find optimal number of clusters using Elbow Method
inertias = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, n_init=10)
    kmeans.fit(data)
    inertias.append(kmeans.inertia_)

# Plot the Elbow graph (Krushna)
plt.plot(range(1, 11), inertias, marker='o')
plt.title('Elbow Method for Optimal K(Kru)')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('Inertia')
plt.show()
```

The output shows an **Elbow Curve** where inertia decreases as the number of clusters increases. The point where the curve bends (the "elbow") represents the **optimal number of clusters**, suggesting the best K value for accurate clustering.



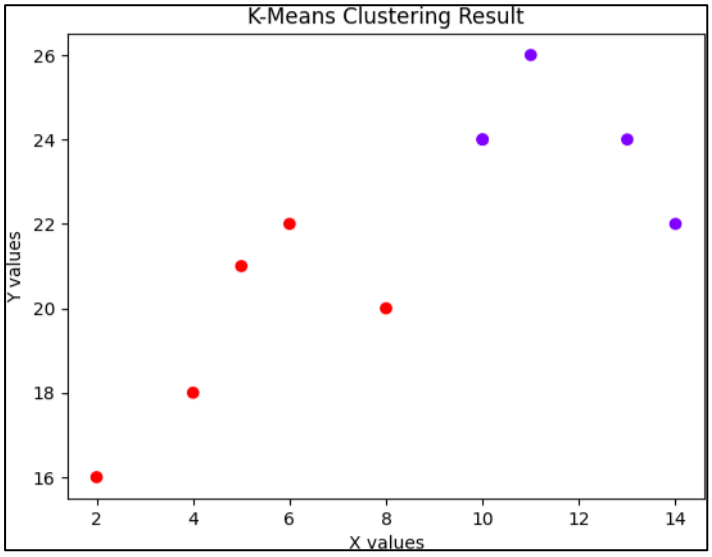

```
: # Apply KMeans with optimal number of clusters (e.g., K=2)
kmeans = KMeans(n_clusters=2, n_init=10)
kmeans.fit(data)
```

```
[9]: KMeans
▼ Parameters
```

| | |
|---|---|
| n_clusters | 2 |
| init | 'k-means++' |
| n_init | 10 |
| max_iter | 300 |
| tol | 0.0001 |
| verbose | 0 |
| random_state | None |
| copy_x | True |
| algorithm | 'lloyd' |

```
[10]: # Plot the clustered data
plt.scatter(x, y, c=kmeans.labels_, cmap='rainbow')
plt.title('K-Means Clustering Result')
plt.xlabel('X values')
plt.ylabel('Y values')
plt.show()
```

This graph shows the **K-Means Clustering Result**, where the data points are divided into two clusters shown in **different colors (red and purple)**. Each color represents a group of points that are close to each other in terms of distance, meaning they share similar characteristics. The clear separation between colors indicates that the algorithm successfully grouped the data into meaningful clusters.




## Conclusion:-

Clustering is a useful technique for uncovering hidden patterns or groups within data without relying on predefined labels. It plays an important role in areas such as data analysis, customer segmentation, image processing, and pattern recognition. Among the different clustering methods, **K-Means** is one of the most popular because it is simple to use, fast to compute, and produces effective results. However, the quality of clustering depends on how the data is spread out and the number of clusters (**K**) selected. In summary, clustering helps group similar data points together, making it easier to understand and analyze large datasets efficiently.