Python Libraries

NumPy

- NumPy, which stands for **Numerical Python**, is an open-source Python library consisting of **multidimensional and single-dimensional array elements**.
- NumPy is most widely used in almost every domain where **numerical computation** is required, like **science and engineering**;
- hence, the NumPy API functionalities are highly utilized in data science and scientific Python packages, including Pandas, SciPy, Matplotlib, scikit-learn, scikit-image, and many more.
- NumPy is a fundamental package for numerical computation in Python.

NumPy

- NumPy includes a wide range of **mathematical functions** for basic arithmetic, linear algebra, Fourier analysis, and more.
- NumPy performs numerical operations on **large datasets** efficiently.
- NumPy supports **multi-dimensional arrays**, allowing for the representation of complex data structures such as images, sound waves, and tensors in machine learning models.
- It supports the writing of **concise and readable code** for complex mathematical computations.

NumPy

- NumPy **integrates with other libraries** to do scientific computation; these are SciPy (for scientific computing), Pandas (for data manipulation and analysis), and scikit-learn (for machine learning).
- Many scientific and numerical **computing libraries and tools are built on top of NumPy**.
- Its widespread adoption and stability make it a standard choice for numerical computing tasks.

NumPy Applications

- Data Analysis
- Machine Learning and Artificial Intelligence
- Scientific Computing.
- Array manipulation
- Finance and Economics
- Engineering and Robotics
- Image and Signal Processing
- Data Visualisation

NumPy Installation

pip install numpy

NumPy Example

import ***numpy*** *as np*

arr = ***np.array****([1, 2, 3, 4, 5])*

print(arr)

o/p- [1 2 3 4 5]

Creating an ndarray

- An instance of **ndarray class** can be constructed by different array creation routines.
- The basic ndarray is created using the **array()** function in NumPy.

numpy.array(object, dtype = None, copy = True, order = None, subok = False, ndmin = 0)

- **Object** -Any object exposing the array interface method returns an array, or any (nested) sequence.
- **Dtype** - Desired data type of array, optional
- **Copy** - Optional. By default (true), the object is copied
- **Order** - C (row major) or F (column major) or A (any) (default)
- **Subok** - By default, returned array forced to be a base class array. If true, sub-classes passed through
- **Ndmin** - Specifies minimum dimensions of resultant array

Create a One-dimensional Array

```
import numpy as np

a = np.array([1, 2, 3])

print(a)
```

- o/p - [1, 2, 3]

Create a 2-dimensional Array

```
import numpy as np
a = np.array([[1, 2], [3, 4]])
print(a)
```

- o/p - [[1 2]
  [3 4]]

Create a 2-dimensional Array

```
import numpy as np
# Creating array object
arr = np.array( [[ 1, 2, 3],[ 4, 2, 5]] )
# Printing type of arr object
print("Array is of type: ", type(arr))
# Printing array dimensions (axes)
print("No. of dimensions: ", arr.ndim)
# Printing shape of array
print("Shape of array: ", arr.shape)
# Printing size (total number of elements) of array
print("Size of array: ", arr.size)
# Printing type of elements in array
print("Array stores elements of type: ", arr.dtype)
```

O/P -

```
Array is of type:
        <class 'numpy.ndarray'>
No. of dimensions:  2
Shape of array:  (2, 3)
Size of array:  6
Array stores elements of type:  int64
```

Create a 3-dimensional Array

import numpy as np

d = np.array([[[1,2,3],[4,5,6]],[[1,2,3],[4,5,6]]])

print(d)

Print(d.ndim)

- o/p - [[[1 2 3]
  [4 5 6]]

  [[1 2 3]
  [4 5 6]]]

Access 1-D Array Elements

- Array indexing is the same as accessing an array element.
- Can access an array element by referring to its index number.

import numpy as np

arr=np.array([1,2,3,4])

print(arr[0])

print(arr[2])

- o/p – 1

  3

ACCESS 2-D ARRAY ELEMENTS

- 2-D arrays accessed like a table with rows and columns, where the dimension represents the row and the index represents the column.

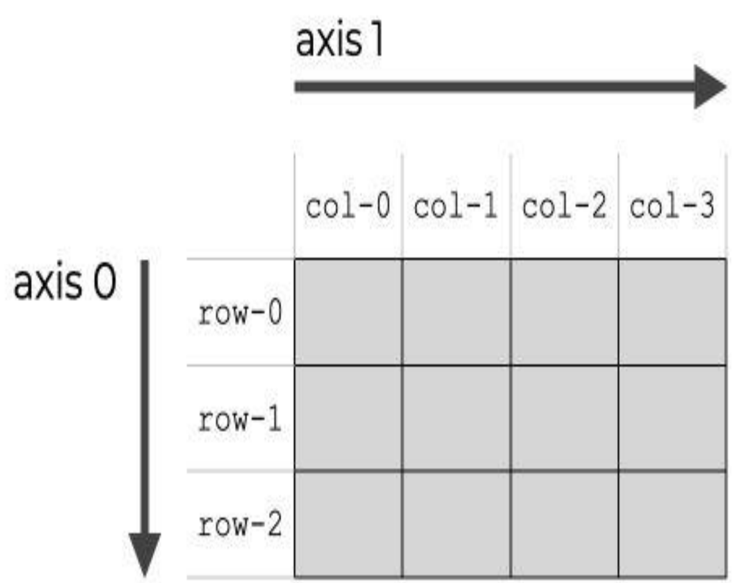
$$\left\{\begin{matrix} 1\ 2\ 3\ 4\ 5 \\ 6\ 7\ 8\ 9\ 10 \end{matrix}\right\}$$

import numpy as np

arr=np.array([[1,2,3,4,5],[6,7,8,9,10]])

print('2nd element on 1st row:',arr[0,1])

print('5th element on 2st row:',arr[1,4])

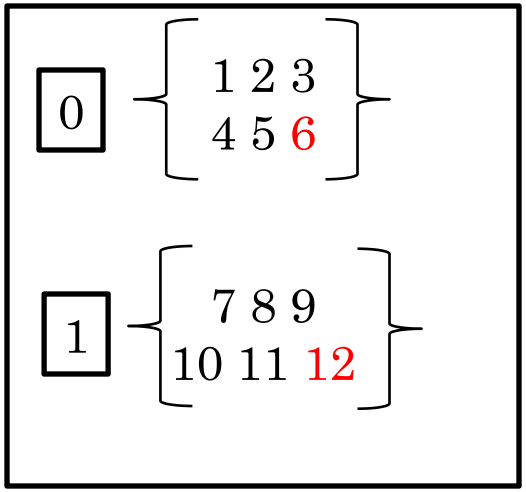


- o/p – 2

  10

ACCESS 3-D ARRAY ELEMENTS

- To access elements from 3-D arrays we can use **comma separated integers** representing the dimensions and the index of the element.



import numpy as np

arr=np.array([[[1,2,3],[4,5,6]],[[7,8,9],[10,11,12]]])

print(arr[0,1,2])

print(arr[1,1,2])

- o/p – 6

  12

Reshaping arrays

- Reshaping means changing the shape of an array.
- The shape of an array is the number of elements in each dimension.
- By reshaping we can add or remove dimensions or change number of elements in each dimension.

```
import numpy as np
arr=np.array([1,2,3,4,5,6,7,8,9,10,11,12])
newarr=arr.reshape(4,3)
print(newarr)
```

```
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]
```

Operations on array

1. **Arithmetic**: Element-wise operations (addition, subtraction, multiplication, division).
2. **Aggregations**: Sum, mean, min, max, standard deviation, etc.
3. **Reshaping**: Reshape, flatten, and transpose arrays.
4. **Matrix operations**: Dot product, matrix multiplication.
5. **Stacking and splitting**: Combine or divide arrays.
6. **Comparison and logical operations**: Perform element-wise comparisons and logical operations.

Arithmetic Operations

```
import numpy as np

arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])

print(arr1 + arr2)  # Output: [5 7 9]
print(arr1 - arr2)  # Output: [-3 -3 -3]
print(arr1 * arr2)  # Output: [4 10 18]
print(arr1 / arr2)  # Output: [0.25 0.4  0.5 ]
print(arr1 ** 2)  # Output: [1 4 9]
```

Matrix Operations - Multiplication

```
import numpy as np
A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6], [7, 8]])
print(np.dot(A, B)) //multiplication
print(A.T)      //transpose



# Output:
   [[19 22]
    [43 50]]
    [[1 3]
    [2 4]]
```

PANDAS

PANDAS

- Pandas is an open-source library that is built on top of NumPy library.
- It is a Python package that offers various **data structures** and **operations** for manipulating numerical data and time series.
- It is mainly popular for importing and analyzing data much easier.
- Pandas is fast and it has high-performance & productivity for users
- Pandas is well-suited for working with tabular data, such as spreadsheets or SQL tables

Benefits of pandas

list of things that we can do using Pandas.

- **Import datasets** - available in the form of spreadsheets, comma-separated values (CSV) files, and more.
- **Data cleansing** - dealing with missing values and representing them as NaN, NA, or NaT.
- **Size mutability** - columns can be added and removed from DataFrame and higher-dimensional objects.
- **Data normalization** – normalize the data into a suitable format for analysis.
- **Data alignment** - objects can be explicitly aligned to a set of labels.

Benefits of pandas

list of things that we can do using Pandas.

- **Intuitive merging and joining data sets** – we can merge and join datasets.
- **Reshaping and pivoting of datasets** – datasets can be reshaped and pivoted as per the need.
- **Efficient manipulation and extraction** - manipulation and extraction of specific parts of extensive datasets using intelligent label-based slicing, indexing, and subsetting techniques.
- **Statistical analysis** - to perform statistical operations on datasets.
- **Data visualization** - Visualize datasets and uncover insights.

Pandas Applications

- Data Cleaning
- Data Exploration
- Data Preparation
- Data Analysis
- Data Visualisation
- Time Series Analysis
- Data Aggregation and Grouping
- Data Input/Output
- Machine Learning
- Web Scraping
- Financial Analysis
- Text Data Analysis
- Experimental Data Analysis

PANDAS INSTALLATION

pip install pandas

- Once Pandas is installed, import it in your applications by adding the import keyword:

import pandas as pd

- Pandas generally provide **two data structures** for manipulating data. They are:

1. **Series**
2. **DataFrame**

Series

- A Pandas **Series** is a **one-dimensional labeled array** capable of **holding data** of any type (integer, string, float, Python objects, etc.).
- The axis labels are collectively called **indexes**.
- The Pandas Series is nothing but a **column in an Excel sheet**.
- Pandas Series is created by loading the datasets from existing storage (which can be a SQL database, a CSV file, or an Excel file).
- Pandas Series can be created from **lists, dictionaries, scalar values**, etc.
- By default, the index of the series starts **from 0 till the length of series -1**.

Series Example

```
import pandas as pd

a=[10,20,30]

myvar=pd.Series(a)

print(myvar)
```

o/p –

```
0    10
1    20
2    30
dtype: int64
```

Series Example with your own labels

import pandas as pd

a=[10,20,30]

myvar=pd.Series(a, index = ["X", "Y", "Z"])

print(myvar)

o/p –

X   10

Y   20

Z   30

dtype: int64

DataFrames

- Data sets in Pandas are usually **multi-dimensional tables**, called **DataFrames**.
- Series is like a column, a **DataFrame is the whole table**.
- Pandas DataFrame is a **two-dimensional data structure** with labeled axes (rows and columns).
- A Pandas DataFrame will be created by **loading the datasets** from existing storage, storage can be SQL Database, CSV file, and Excel file.
- Pandas DataFrame can be created from the lists, dictionary, and from a list of dictionary etc.

PANDAS EXAMPLE

```
import pandas as pd
a=[10,20,30]
a1 = pd.DataFrame(info)
print(a1)
```

o/p –

```
   0
0  10
1  20
2  30
```

PANDAS EXAMPLE

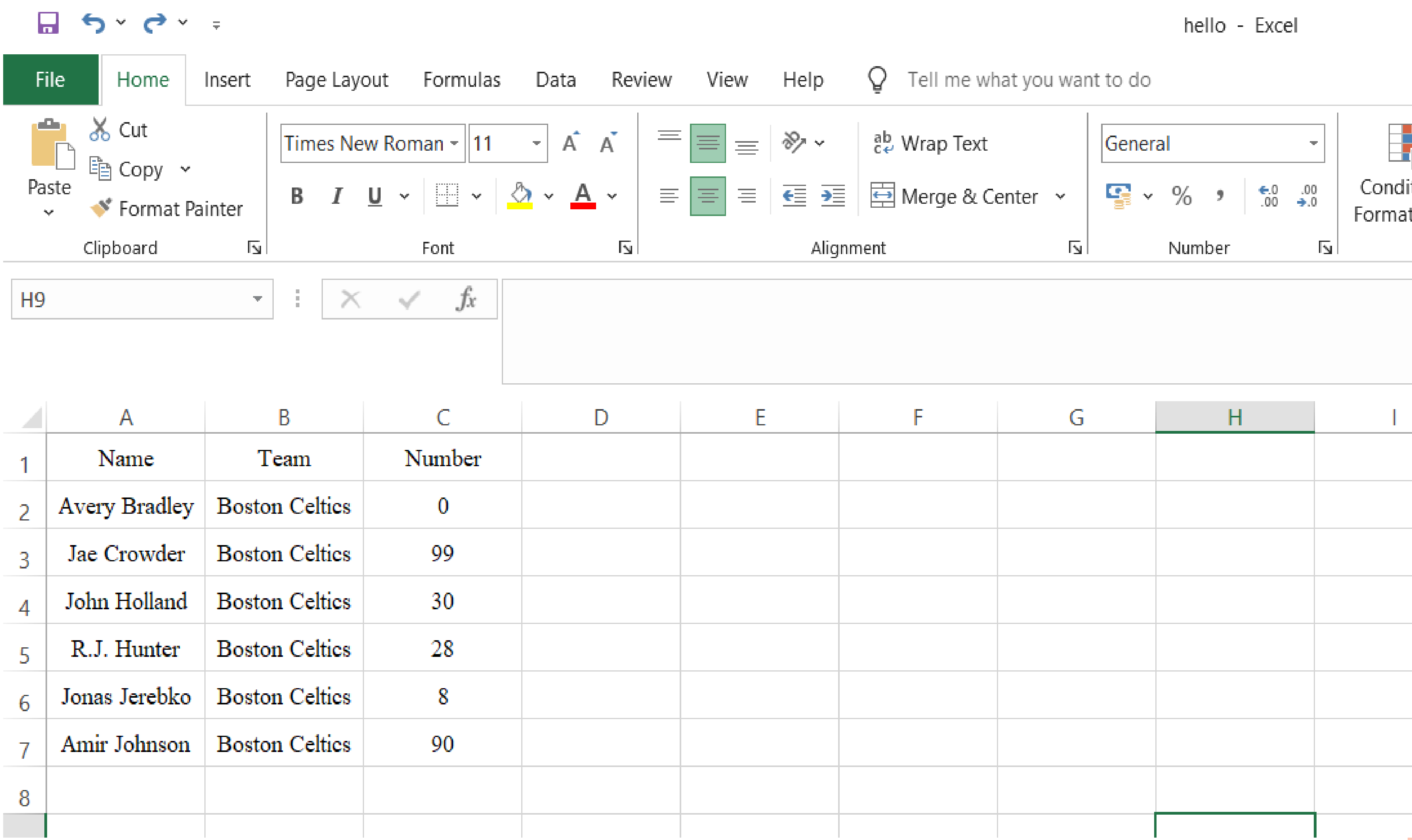
```
import pandas as pd

data = { "calories": [420, 380, 390], “duration": [50, 40, 45]}

myvar = pd.DataFrame(data)

print(myvar)
```

o/p –

	calories	duration
0	420	50
1	380	40
2	390	45

Loading data from .CSV file

- Hello.csv



Name	Team	Number
Avery Bradley	Boston Celtics	0
Jae Crowder	Boston Celtics	99
John Holland	Boston Celtics	30
R.J. Hunter	Boston Celtics	28
Jonas Jerebko	Boston Celtics	8
Amir Johnson	Boston Celtics	90

Loading data from .CSV file

```
import pandas as pd
# making data frame
df = pd.read_csv("d:\\hello.csv")
ser = pd.Series(df['Name'])
print(ser)
ser = pd.DataFrame(df[['Name','Team','Number']])
print(ser)
```

o/p –

```
0    Avery Bradley
1      Jae Crowder
2     John Holland
3      R.J. Hunter
4    Jonas Jerebko
5     Amir Johnson
Name: Name, dtype: object
```

	Name	Team	Number
0	Avery Bradley	Boston Celtics	0
1	Jae Crowder	Boston Celtics	99
2	John Holland	Boston Celtics	30
3	R.J. Hunter	Boston Celtics	28
4	Jonas Jerebko	Boston Celtics	8
5	Amir Johnson	Boston Celtics	90

Loading data from .CSV file

- Hello.csv

	Name	Team	Number	Position	Age	Height	Weight	College	Salary
0	Avery Bradley	Boston Celtics	0.0	PG	25.0	6-2	180.0	Texas	7730337.0
1	Jae Crowder	Boston Celtics	99.0	SF	25.0	6-6	235.0	Marquette	6796117.0
2	John Holland	Boston Celtics	30.0	SG	27.0	6-5	205.0	Boston University	NaN
3	R.J. Hunter	Boston Celtics	28.0	SG	22.0	6-5	185.0	Georgia State	1148640.0
4	Jonas Jerebko	Boston Celtics	8.0	PF	29.0	6-10	231.0	NaN	5000000.0
5	Amir Johnson	Boston Celtics	90.0	PF	29.0	6-9	240.0	NaN	12000000.0
6	Jordan Mickey	Boston Celtics	55.0	PF	21.0	6-8	235.0	LSU	1170960.0
7	Kelly Olynyk	Boston Celtics	41.0	C	25.0	7-0	238.0	Gonzaga	2165160.0
8	Terry Rozier	Boston Celtics	12.0	PG	22.0	6-2	190.0	Louisville	1824360.0
9	Marcus Smart	Boston Celtics	36.0	PG	22.0	6-4	220.0	Oklahoma State	3431040.0
10	Jared Sullinger	Boston Celtics	7.0	C	24.0	6-9	260.0	Ohio State	2569260.0
11	Isaiah Thomas	Boston Celtics	4.0	PG	27.0	5-9	185.0	Washington	6912869.0
12	Evan Turner	Boston Celtics	11.0	SG	27.0	6-7	220.0	Ohio State	3425510.0
13	James Young	Boston Celtics	13.0	SG	20.0	6-6	215.0	Kentucky	1749840.0

HEAD() METHOD IS USED TO RETURN **TOP N (5 BY DEFAULT) ROWS** OF A DATA FRAME OR SERIES.

```
import pandas as pd
# making data frame
df = pd.read_csv("d:\\hello.csv")
data_top = data.head()
print(data_top )
```

o/p –

	Name	Team	Number	Position	Age	Height	Weight	College	Salary
0	Avery Bradley	Boston Celtics	0.0	PG	25.0	6-2	180.0	Texas	7730337.0
1	Jae Crowder	Boston Celtics	99.0	SF	25.0	6-6	235.0	Marquette	6796117.0
2	John Holland	Boston Celtics	30.0	SG	27.0	6-5	205.0	Boston University	NaN
3	R.J. Hunter	Boston Celtics	28.0	SG	22.0	6-5	185.0	Georgia State	1148640.0
4	Jonas Jerebko	Boston Celtics	8.0	PF	29.0	6-10	231.0	NaN	5000000.0

TAIL() METHOD IS USED TO RETURN **BOTTOM N (5 BY DEFAULT) ROWS** OF A DATA FRAME OR SERIES.

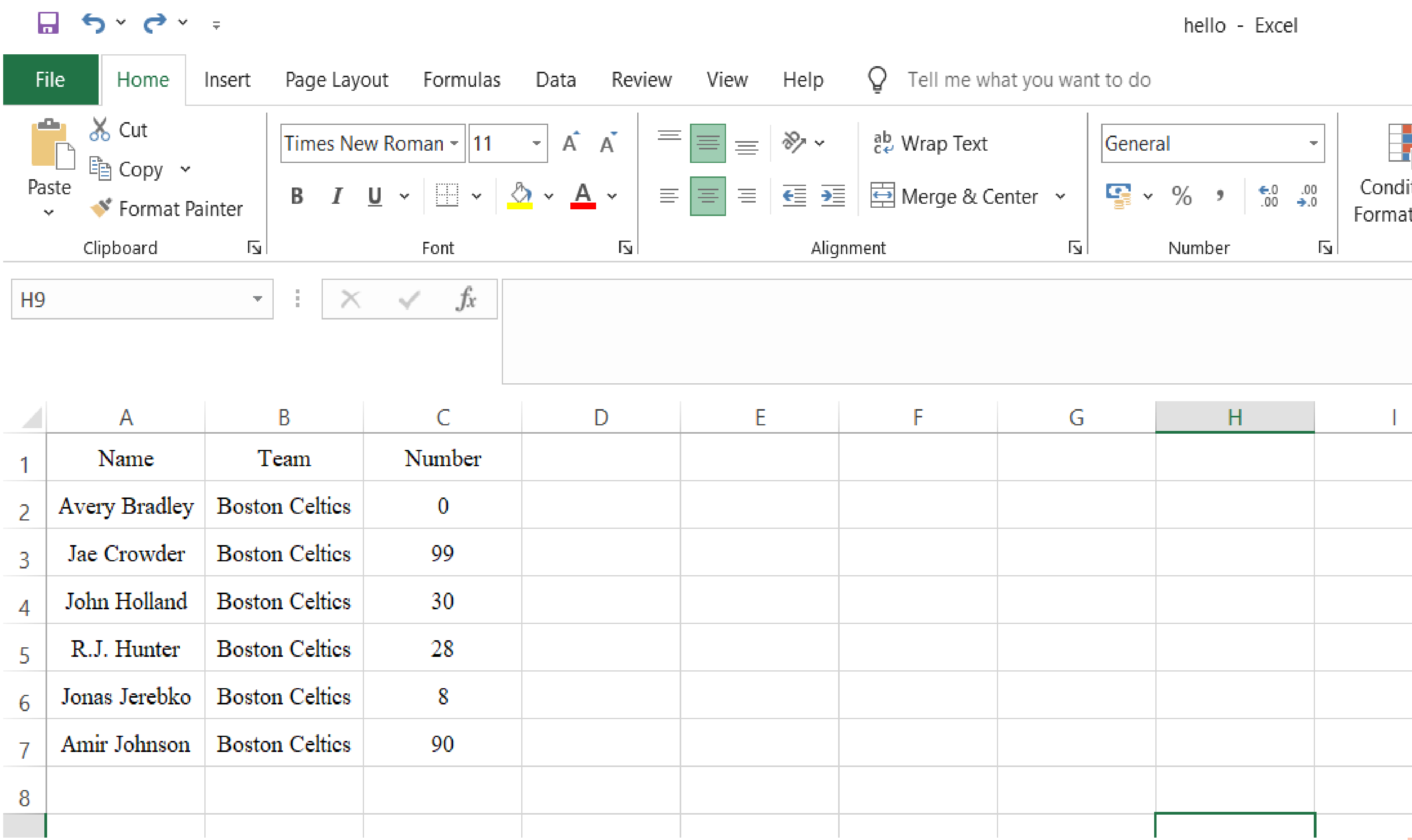
```
import pandas as pd

# making data frame

df = pd.read_csv("d:\\hello.csv")

data_bottom = data.tail()

print(data_bottom )
```

o/p –

	Name	Team	Number	Position	Age	Height	Weight	College	Salary
9	Marcus Smart	Boston Celtics	36.0	PG	22.0	6-4	220.0	Oklahoma State	3431040.0
10	Jared Sullinger	Boston Celtics	7.0	C	24.0	6-9	260.0	Ohio State	2569260.0
11	Isaiah Thomas	Boston Celtics	4.0	PG	27.0	5-9	185.0	Washington	6912869.0
12	Evan Turner	Boston Celtics	11.0	SG	27.0	6-7	220.0	Ohio State	3425510.0
13	James Young	Boston Celtics	13.0	SG	20.0	6-6	215.0	Kentucky	1749840.0

Loading data from .CSV file

- Hello.csv



	A	B	C
1	Name	Team	Number
2	Avery Bradley	Boston Celtics	0
3	Jae Crowder	Boston Celtics	99
4	John Holland	Boston Celtics	30
5	R.J. Hunter	Boston Celtics	28
6	Jonas Jerebko	Boston Celtics	8
7	Amir Johnson	Boston Celtics	90
8			

PANDAS DESCRIBE() IS USED TO VIEW SOME BASIC STATISTICAL DETAILS LIKE PERCENTILE, MEAN, STD, ETC. OF A DATA FRAME OR A SERIES OF NUMERIC VALUES

```
import pandas as pd
df = pd.read_csv("d:\\hello.csv")
ser = pd.DataFrame(df[['Name','Team','Number']])
print(ser)
print(ser.describe())
```

o/p –

	Name	Team	Number
0	Avery Bradley	Boston Celtics	0
1	Jae Crowder	Boston Celtics	99
2	John Holland	Boston Celtics	30
3	R.J. Hunter	Boston Celtics	28
4	Jonas Jerebko	Boston Celtics	8
5	Amir Johnson	Boston Celtics	90

	Number
count	6.000000
mean	42.500000
std	41.979757
min	0.000000
25%	13.000000
50%	29.000000
75%	75.000000
max	99.000000

MATPLOTLIB

MATPLOTLIB

- Matplotlib is a powerful and widely-used plotting library in Python which enables us to **create a variety of static, interactive and publication-quality plots and visualizations**.
- It's extensively used for data visualization tasks and offers a wide range of functionalities to create plots like **line plots, scatter plots, bar charts, histograms, 3D plots** and much more.
- Matplotlib library provides **flexibility** and **customization** options to tailor our plots according to specific needs.
- It is a cross-platform library for making **2D plots** from data in arrays.
- Matplotlib is written in Python and makes use of **NumPy**, the numerical mathematics extension of Python.

MATPLOTLIB

- It provides an **object-oriented API** that helps in embedding plots in applications using Python GUI toolkits such as PyQt, WxPython, Tkinter.
- Matplotlib has a procedural interface named the Pylab which is designed to resemble MATLAB a proprietary programming language developed by MathWorks.
- Matplotlib along with NumPy can be considered as the open source equivalent of MATLAB.
- Matplotlib was originally written by John D. Hunter in 2003. The current stable version is 2.2.0 released in January 2018.

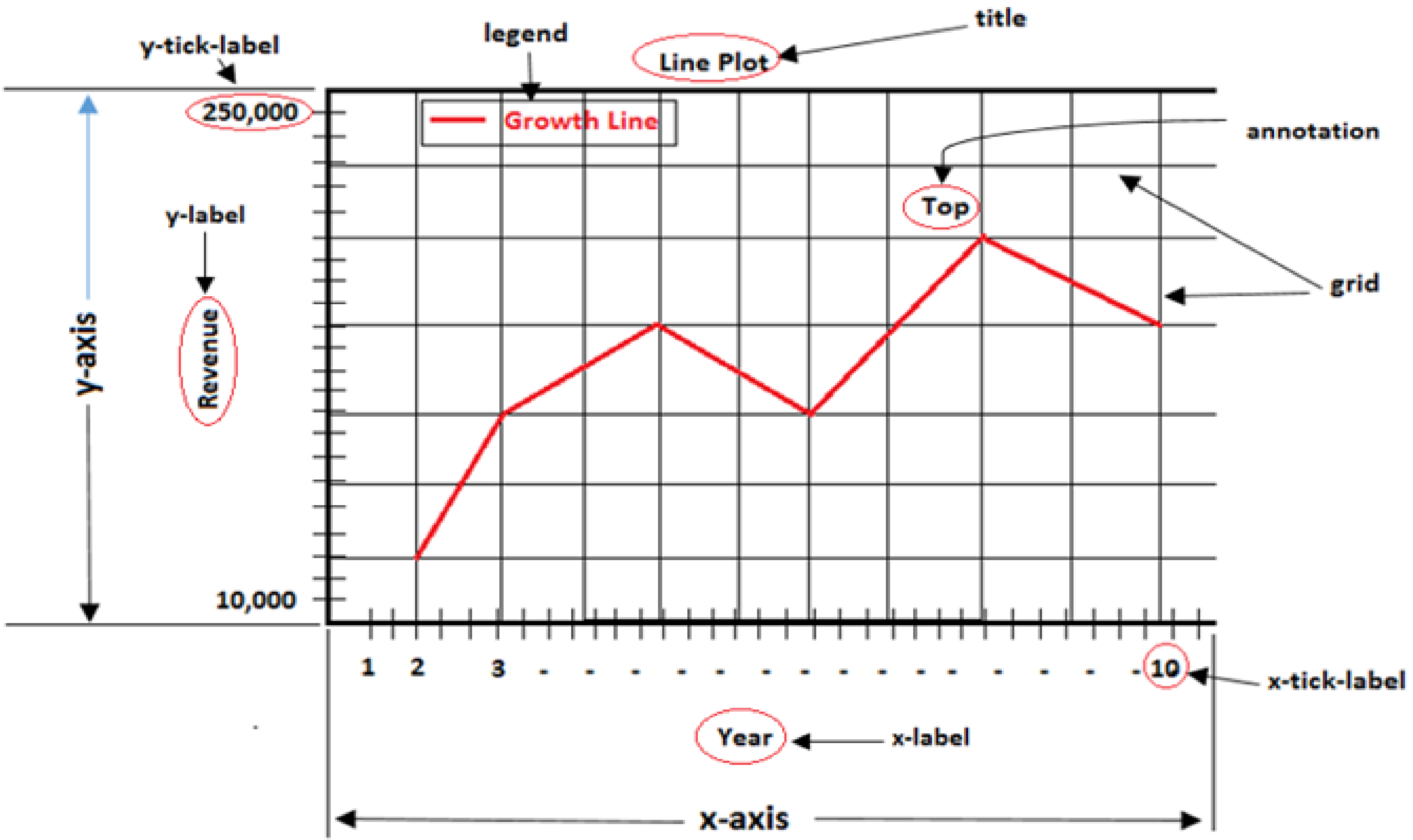
Applications of Matplotlib

- Data Visualization
- Scientific Research
- Engineering
- Education
- Geospatial Analysis
- Finance

MATPLOTLIB INSTALLATION

pip install matplotlib

MATPLOTLIB COMPONENTS



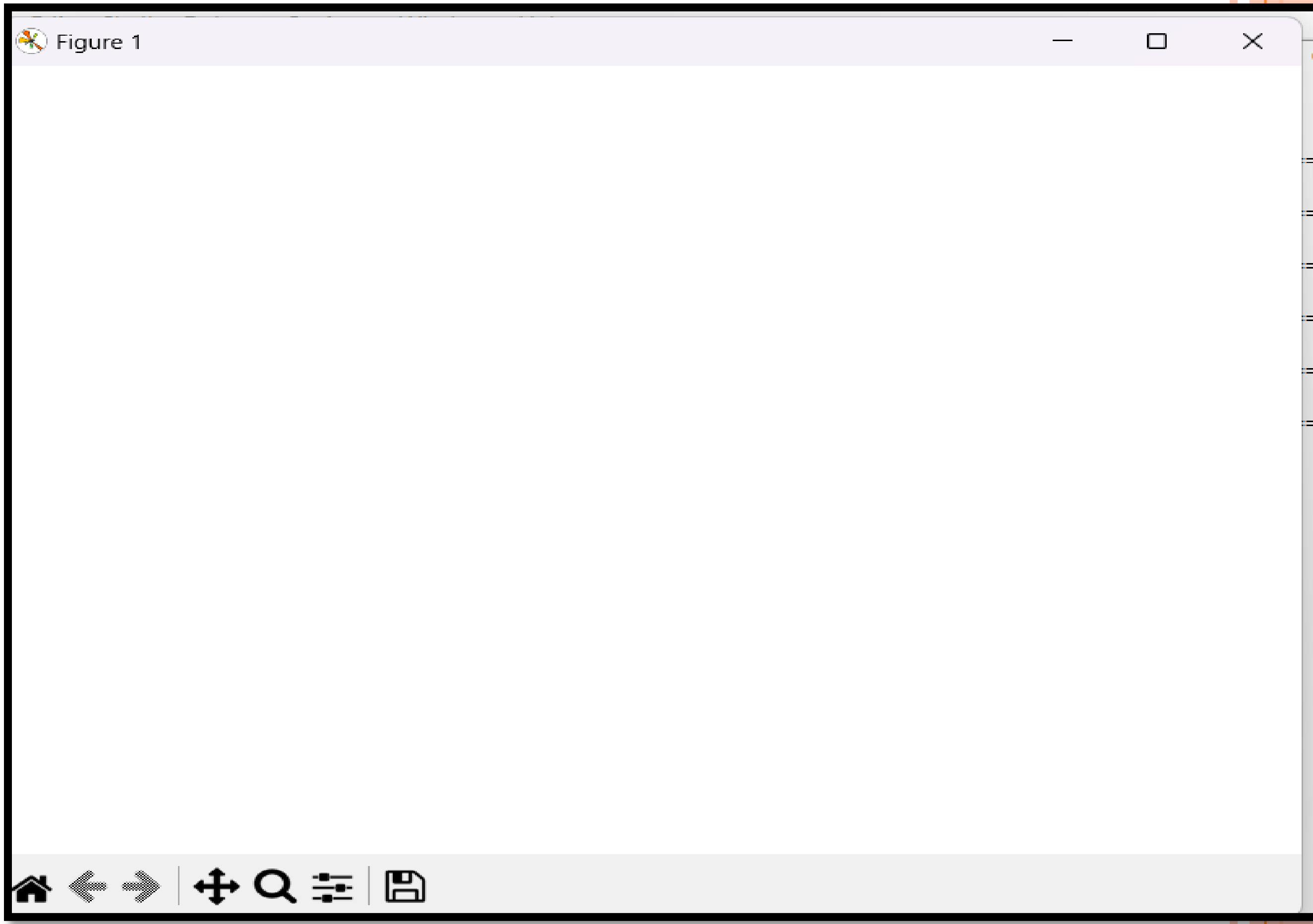
Figure

- A figure is the entire window or page that displays our plot or collection of plots.
- It acts as a container that holds all elements of a graphical representation which includes axes, labels, legends and other components.
- Most of the Matplotlib utilities lies under the **pyplot submodule**, and are usually import as follows –

import matplotlib.pyplot as plt

FIGURE

```
import matplotlib.pyplot as plt

# Create a new figure

fig = plt.figure()

plt.show()
```



PLOT

plt.plot(x, y, color='red', linestyle='--', linewidth=2, marker='o', markersize=5)

plt.savefig('my_plot.png', dpi=300)

- **Plotting:** plt.plot() is used to create the line chart.
- **Legend:** Add a legend using plt.legend(['Label']) if you have multiple lines.
- **Titles and Labels:** plt.title(), plt.xlabel(), and plt.ylabel() add titles and labels to the axes.
- **Display:** plt.show() renders the plot.

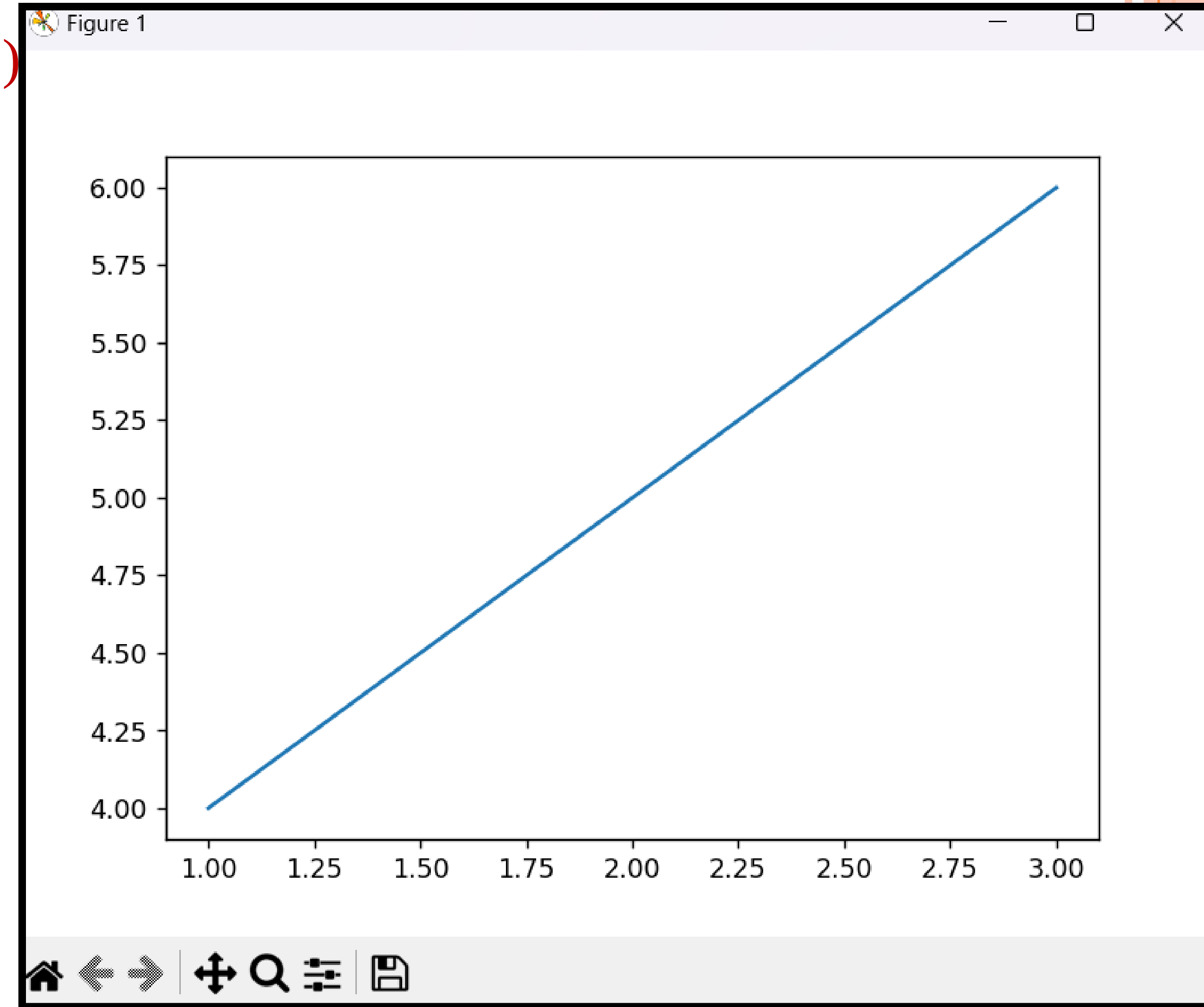
PLOT

import matplotlib.pyplot as plt

Add a plot or subplot to the figure

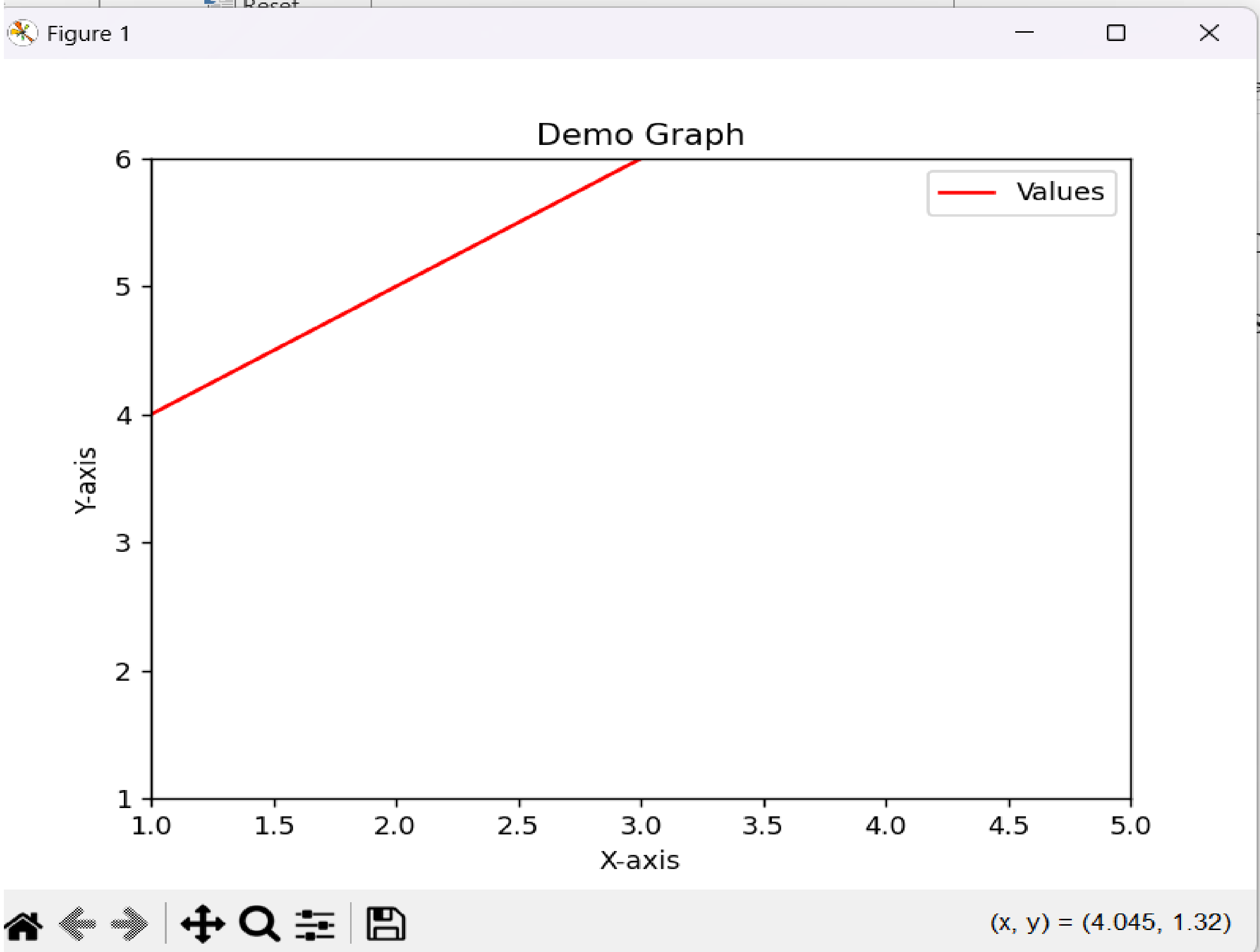
plt.plot([1, 2, 3], [4, 5, 6])

plt.show()

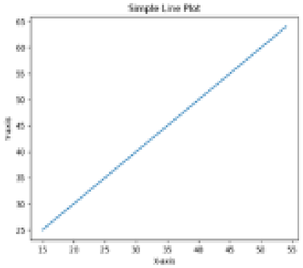
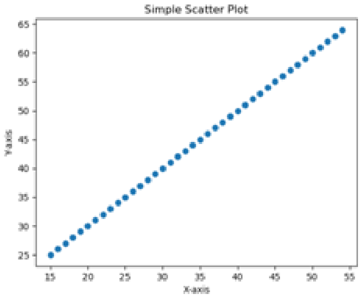


PLOT FUNCTIONS

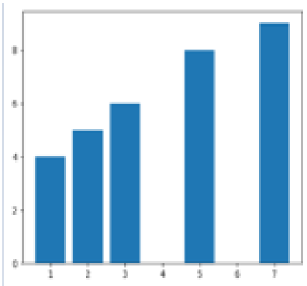
```
import matplotlib.pyplot as plt

# Add a plot or subplot to the figure

#The plot function marks the x-coordinates and y-coordinates

plt.plot([1, 2, 3], [4, 5, 6], color="red")

plt.xlim(1, 5)

plt.ylim(1, 6)

plt.xlabel('X-axis')

plt.ylabel('Y-axis')

plt.title('Demo Graph’)

plt.legend(['Values'])

plt.show()
```



Types of Plots

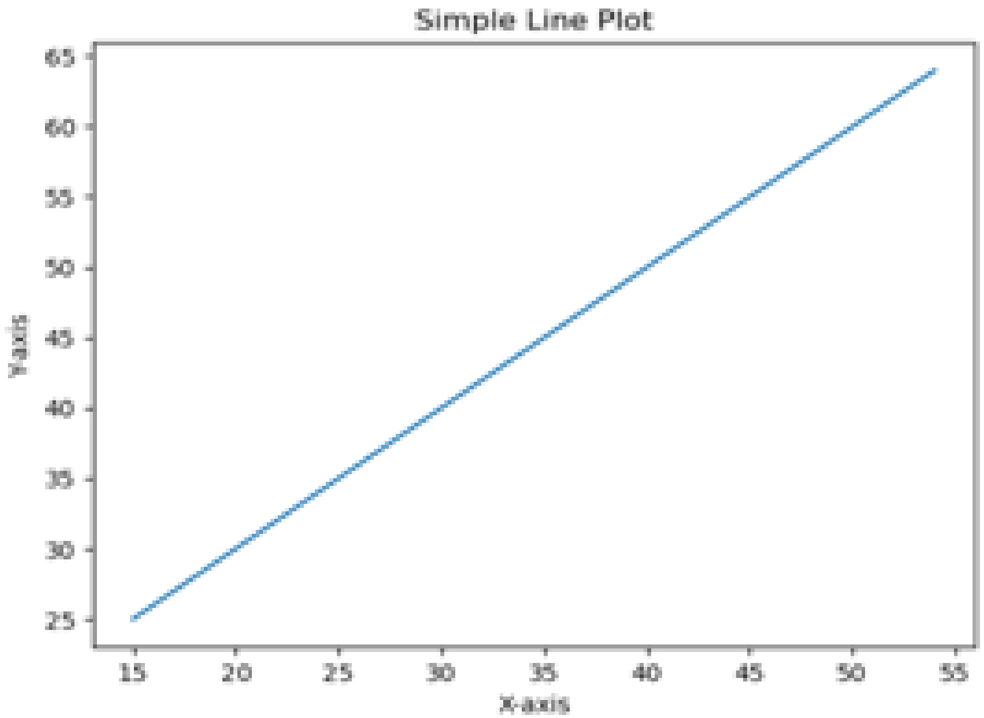
Name of the plot	Definition	Image
Line plot	A line plot is a type of graph that displays data points connected by straight line segments. The plt.plot() function of the matplotlib library is used to create the line plot.	
Scatter plot	A scatter plot is a type of graph that represents individual data points by displaying them as markers on a two-dimensional plane. The plt.scatter() function is used to plot the scatter plot.	

Types of Plots

Bar plot	A bar plot or bar chart is a visual representation of categorical data using rectangular bars. The plt.bar() function is used to plot the bar plot.	
Pie plot	A pie plot is also known as a pie chart. It is a circular statistical graphic used to illustrate numerical proportions. It divides a circle into sectors or slices to represent the relative sizes or percentages of categories within a dataset. The plt.pie() function is used to plot the pie chart.	

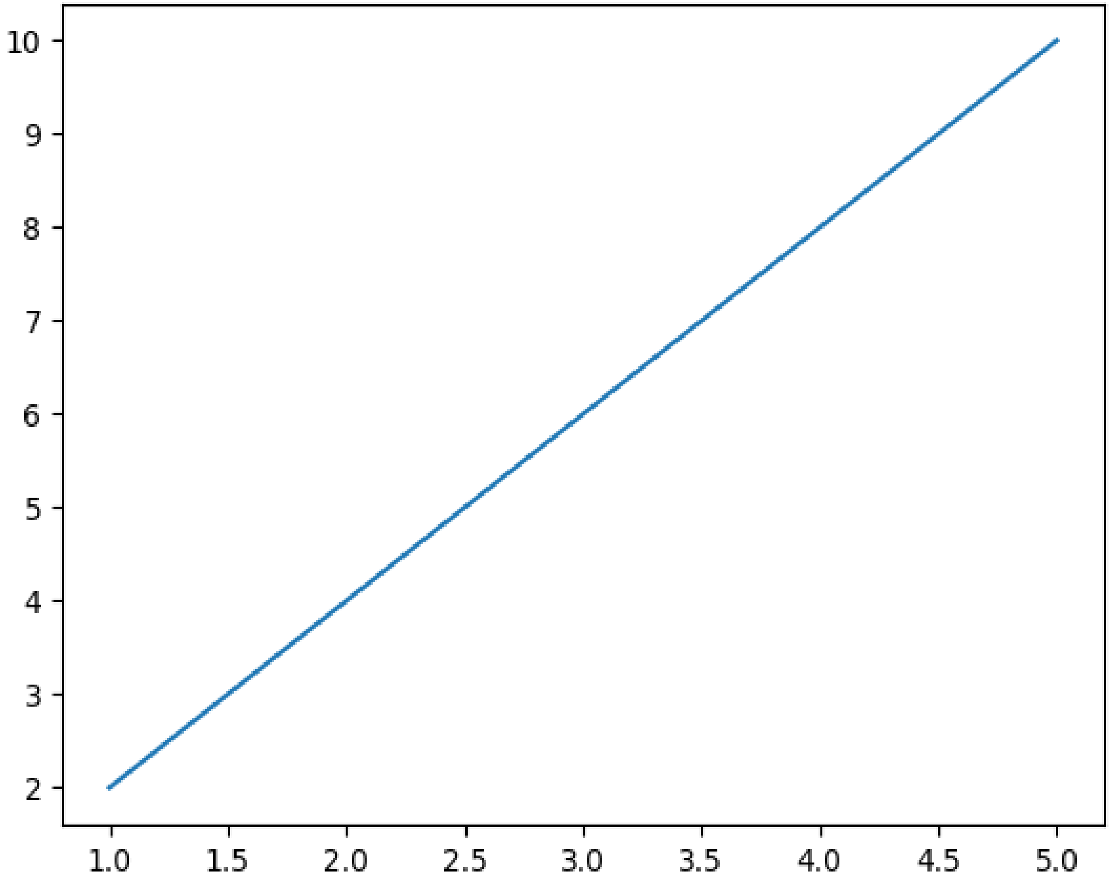
Line plot

- Line plot are used to represent the relation between two data X and Y on a different axis.



Line plot

```
import matplotlib.pyplot as plt

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

y = [2, 4, 6, 8, 10]

plt.plot(x,y)

plt.show()
```

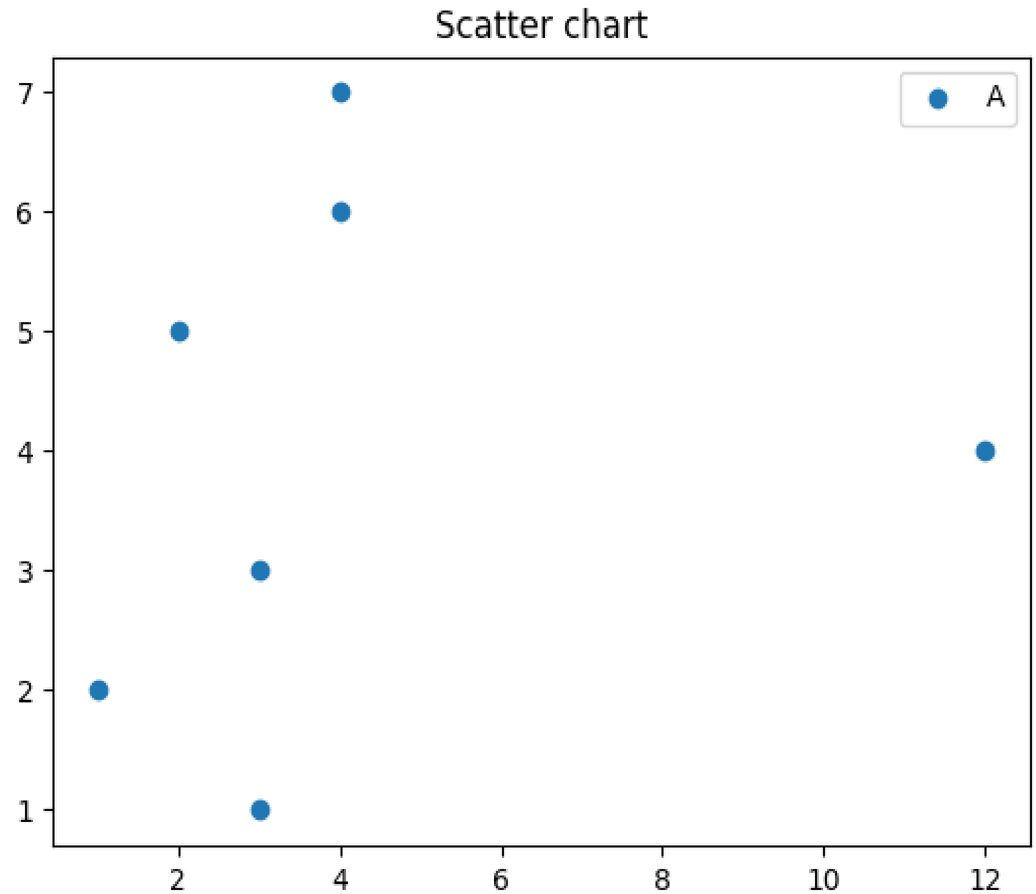


Scatter plot

- The Matplotlib.pyplot.scatter() in Python extends to creating diverse plots such as scatter plots, bar charts, pie charts, line plots, histograms, 3-D plots, and more.
- The matplotlib library provides the scatter() method, specifically designed for creating scatter plots.

Scatter plot

```
import matplotlib.pyplot as plt

# data to display on plots

x = [3, 1, 3, 12, 2, 4, 4]

y = [3, 2, 1, 4, 5, 6, 7]

# To plot simple scatter

plt.scatter(x, y)

# Title to the plot

plt.title("Scatter chart")

plt.show()
```

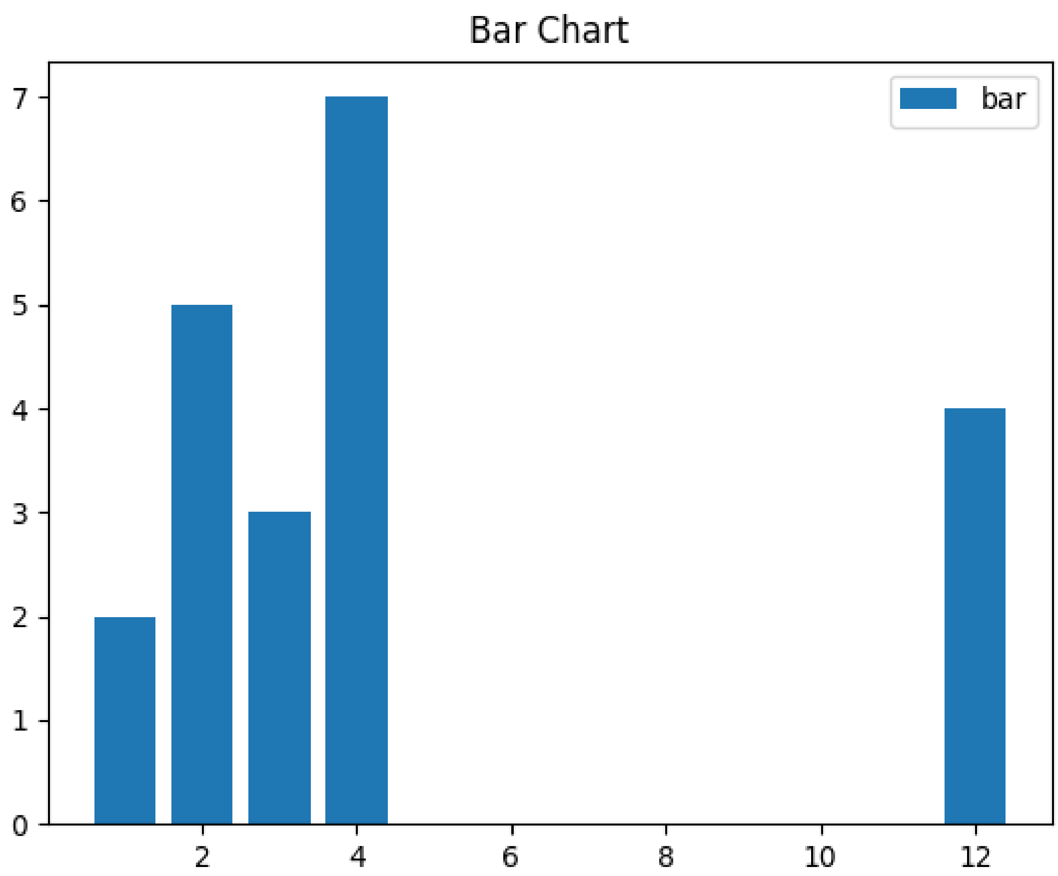


BAR PLOT

- A bar plot or bar chart is a graph that represents the **category of data with rectangular bars with lengths and heights** that is proportional to the values which they represent.
- The bar plots can be plotted **horizontally or vertically**.
- A bar chart describes the comparisons between the discrete categories.
- It can be created using the **bar()** method.

Bar plot

```
import matplotlib.pyplot as plt
# data to display on plots
x = [3, 1, 3, 12, 2, 4, 4]
y = [3, 2, 1, 4, 5, 6, 7]
plt.bar(x, y)
# Title to the plot
plt.title("Bar Chart")
# Adding the legends
plt.legend(["bar"])
plt.show()
```

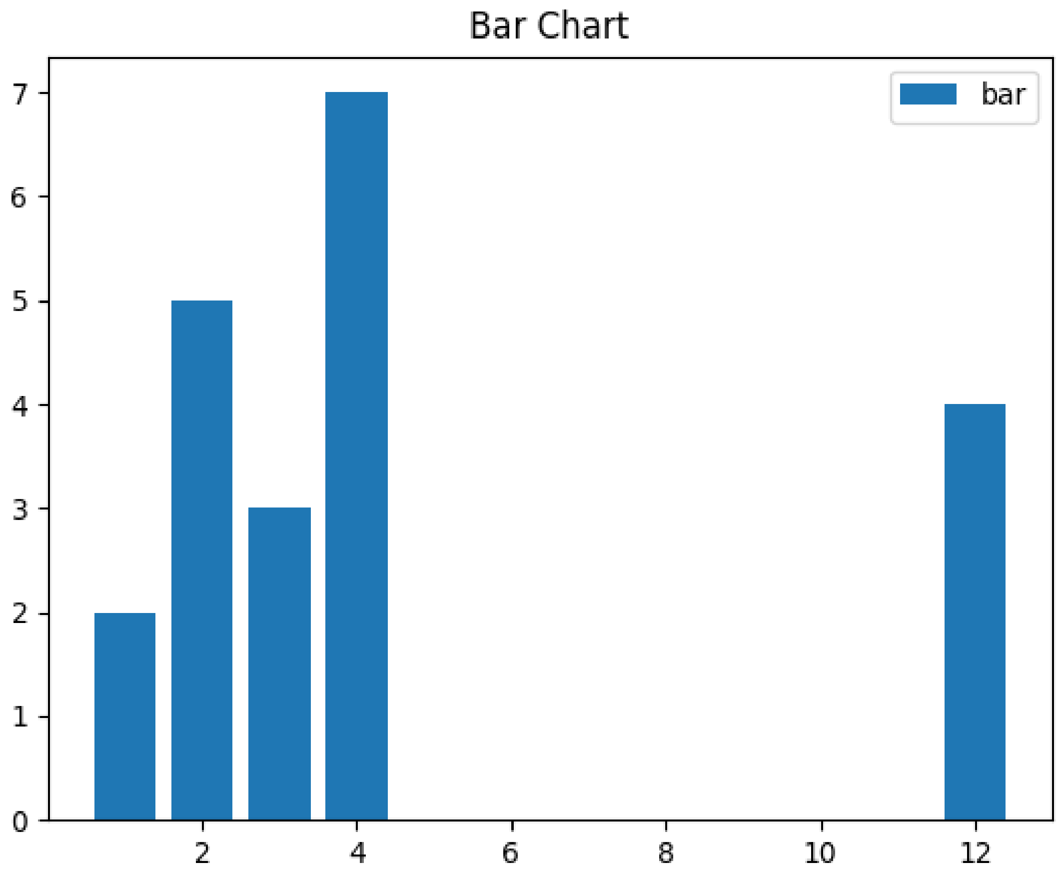


BAR PLOT

- A bar plot or bar chart is a graph that represents the **category of data with rectangular bars with lengths and heights** that is proportional to the values which they represent.
- The bar plots can be plotted **horizontally or vertically**.
- A bar chart describes the comparisons between the discrete categories.
- It can be created using the **bar()** method.

Bar plot

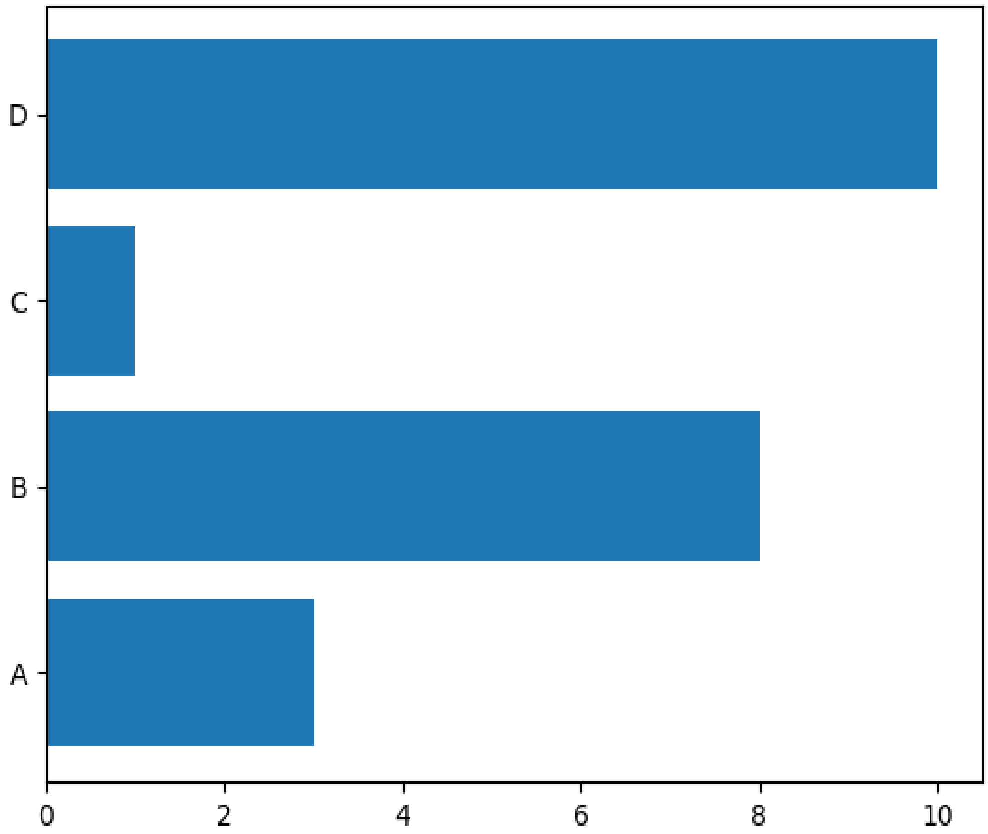
```
import matplotlib.pyplot as plt
# data to display on plots
x = [3, 1, 3, 12, 2, 4, 4]
y = [3, 2, 1, 4, 5, 6, 7]
plt.bar(x, y)
# Title to the plot
plt.title("Bar Chart")
# Adding the legends
plt.legend(["bar"])
plt.show()
```



HORIZONTAL BARS

- Use plt.barh() for horizontal bar graphs.

```
import matplotlib.pyplot as plt

import numpy as np

x = np.array(["A", "B", "C", "D"])

y = np.array([3, 8, 1, 10])

plt.barh(x, y)

plt.show()
```

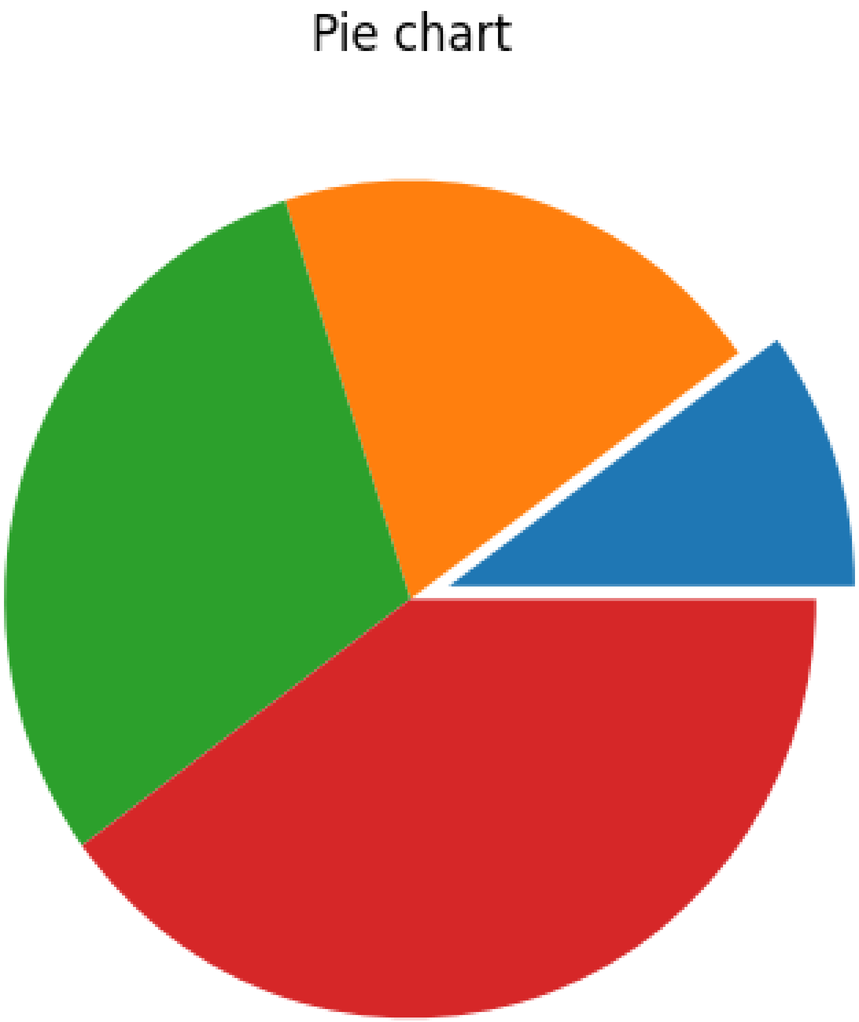


PIE PLOT

- A Pie Chart is a **circular statistical plot** that can display only one series of data.
- The area of the chart is the **total percentage of the given data**.
- The **area of slices of the pie** represents the percentage of the parts of the data.
- The **slices of pie are called wedges**.
- The area of the wedge is determined by **the length of the arc of the wedge.**

PIE PLOT

```
import matplotlib.pyplot as plt
# data to display on plots
x = [1, 2, 3, 4]
# this will explode the 1st wedge
# i.e. will separate the 1st wedge
# from the chart
e  =(0.1, 0, 0, 0)
# This will plot a simple pie chart
plt.pie(x, explode = e)
# Title to the plot
plt.title("Pie chart")
plt.show()
```

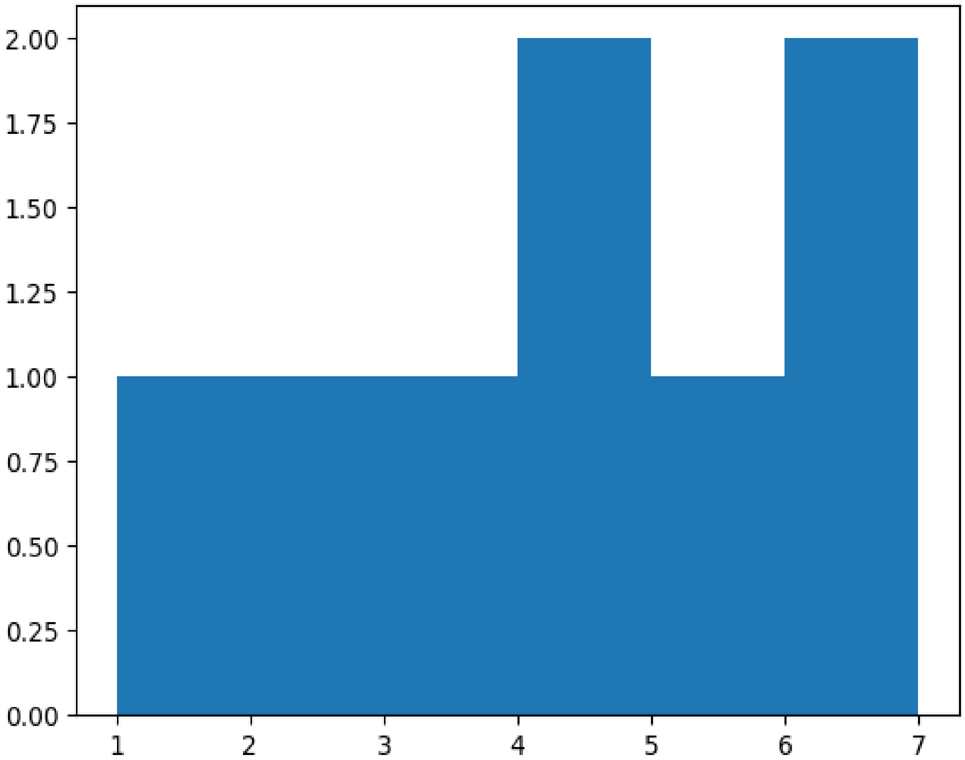


HISTOGRAM

- A histogram is basically used to represent data in the form of some groups.
- It is a type of bar plot where the X-axis represents the bin ranges while the Y-axis gives information about frequency.
- To create a histogram the first step is to create a bin of the ranges, then distribute the whole range of the **values into a series of intervals**, **and count the values which fall into each of the intervals**.
- Bins are clearly identified as consecutive, non-overlapping intervals of variables.

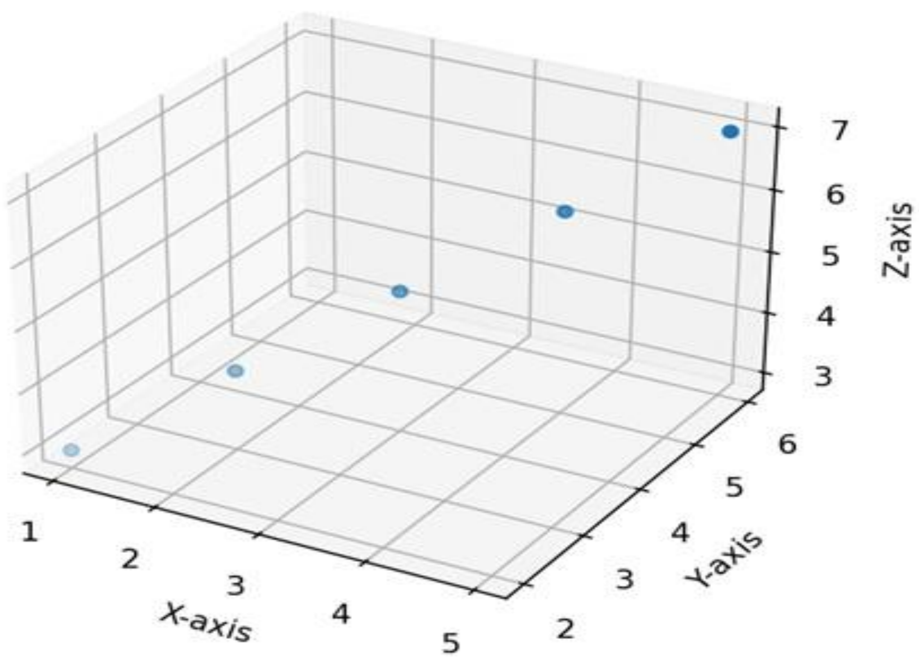
HISTOGRAMS

```
import matplotlib.pyplot as plt

# data to display on plots

x = [1, 2, 3, 4, 5, 6, 7, 4]

# This will plot a simple histogram

plt.hist(x, bins = [1, 2, 3, 4, 5, 6, 7])

plt.show()
```



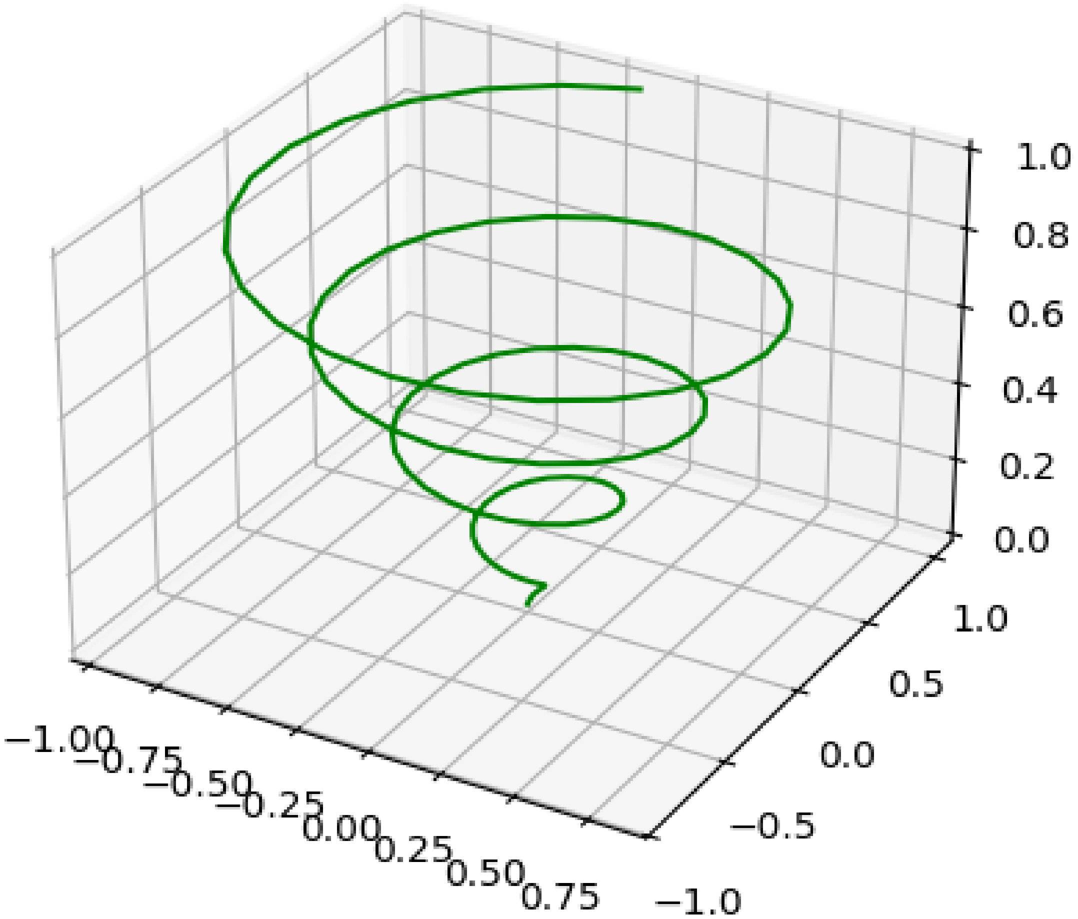
3D Plot

- Sometimes, data visualization requires a **three-dimensional perspective.**
- creating **3D plots** to visualize complex relationships and structures within multidimensional datasets.



3D Plot

```
import matplotlib.pyplot as plt

import numpy as np

fig = plt.figure()

# keeping the projection = 3d

# creates the 3d plot

ax = plt.axes(projection = '3d')

z = np.linspace(0, 1, 100)

x = z * np.sin(25 * z)

y = z * np.cos(25 * z)

# plotting

ax.plot3D(x, y, z, 'green')

plt.show()
```



SUBPLOT

subplot(rows, columns, position)

Subplot

```
import matplotlib.pyplot as plt
import numpy as np
#plot 1:
x =np.array([0,1,2,3])
y =np.array([3,8,1,10])
plt.subplot(1,2,1)
plt.plot(x,y)
plt.title("SALES")
#plot 2:
x =np.array([0,1,2,3])
y =np.array([10,20,30,40])
plt.subplot(1,2,2)
plt.plot(x,y)
plt.title("INCOME")
plt.suptitle("MY SHOP")
plt.show()
```