

**AAVISHKAR**  
ENGINEERING CLASSES  
*Work Smarter, Not Harder.*

**DIPLOMA, DEGREE AND M.E. CLASSES**

**ALL SUBJECTS OF ALL SEMESTERS**




CE BCA BSC - IT  
IT MCA MSC - IT

Come & Join Us For Bright Future

**Programming Language For Beginners**

OUR TRAINING COURSE C++ php python iOS android java

Education Is The Most **POWERFUL** Weapon Which You Can Use To Change The World.

Follow Us :   

## CAUTION: PHOTOCOPYING IS ILLEGAL

### SAVE YOURSELF, DON'T BUY PHOTOCOPIED NOTES

This Notes is protected under Copyright Act 1999 and sold subject to the condition that the Notes and any extract thereof shall be not photocopied and includes the said condition being imposed on any subsequent purchaser.

Any person found selling, stocking or carrying photocopied Notes may be arrested for indulging in criminal offence of copyright piracy and may be imprisoned for 3 years and also fined a sum of 2,00,000/- for first offence.

Sharing of PDF's, any Drives, Links, Storing in Hard Disks, Pen drive and circulating on social media like Instagram, Telegram, Facebook, Snapchat, Google Drive & WhatsApp etc also violates the Copyright Laws and will be reported to Cyber Crime Division.

Publisher has raided many such offenders. Their Machines were Seized. Criminal case has also been registered against them. Civil Suits are filed for recovering damages.

Name of informer will be kept highly confidential and he will be suitably rewarded.

Call/WhatsApp us on these Numbers:

**7600449191, 9898002406**

*Work Smarter, Not Harder.*



Email ID: [aavishkar2406@gmail.com](mailto:aavishkar2406@gmail.com)

Instagram ID: aavishkar2406

📍: 202, I - Address. Panchamrut Bunglows II, Sola Bridge S.G. Highway,  
Opp. Science City Approach BRTS Stop, Science City, Ahmedabad - 380060.

📞: + 91 7600449191, 📧: [aavishkar2406@gmail.com](mailto:aavishkar2406@gmail.com)

## Unit – 6

### Python Libraries for Machine Learning

#### 1. Numpy:

1. Creating Array
2. Accessing Array
3. Stacking & Splitting
4. Maths Functions
5. Statistics Functions

#### 2. Pandas:

1. Series
2. DataFrames
3. Read CSV File
4. Cleaning Empty Cells
5. Cleaning Wrong Data
6. Removing Duplicates

#### 7. Pandas Plotting

#### 3. Matplotlib:

1. Pyplot.plot
2. labels
3. grid
4. bars



5. histogram
  6. subplot
  7. pie chart
4. Sklearn:
1. Key concepts and features
  2. Steps to Build a Model in Sklearn:
    - Loading a Dataset
    - Train\_test\_split



**AAVISHKAR**  
ENGINEERING CLASSES

*Work Smarter, Not Harder.*

## 1. Numpy:

Ans:

- NumPy is a **Python library**.
- NumPy is used for **working with arrays**.
- NumPy is short for "**Numerical Python**".
- NumPy is a Python library used for working with arrays.
- It also has functions for working in domain of **linear algebra, fourier transform, and matrices**.
- NumPy aims to provide an **array object that is up to 50x faster than traditional Python lists**.

### 1. Creating Array:

Ans:

- Create a NumPy ndarray Object
- NumPy is used to work with arrays. The array object in NumPy is called ndarray.
- We can create a NumPy ndarray object by using the array() function.
- **Example:**

```
1 import numpy as np
2
3 arr = np.array([1, 2, 3, 4, 5])
4
5 print(arr)
6
7 print(type(arr))
```

- **Output:**

```
[1 2 3 4 5]
<class 'numpy.ndarray'>
```

- **Dimensions in Arrays**

- A dimension in arrays is one level of **array depth (nested arrays)**.

- **0-D Arrays**

- 0-D arrays, or Scalars, are the elements in an array.
- Each value in an array is a 0-D array.

- **1-D Arrays**

- An array that has 0-D arrays as its elements is called uni-dimensional or 1-D array.
- These are the most common and basic arrays.

- **2-D Arrays**

- An array that has 1-D arrays as its elements is called a 2-D array.
- These are often used to represent matrix or 2nd order tensors.

- **3-D arrays**

- An array that has 2-D arrays (matrices) as its elements is called 3-D array.
- These are often used to represent a 3rd order tensor.

- **Example:**

```
1 import numpy as np
2
3 a = np.array(42)
4 b = np.array([1, 2, 3, 4, 5])
5 c = np.array([[1, 2, 3], [4, 5, 6]])
6 d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])
7
8 print(a.ndim)
9 print(b.ndim)
10 print(c.ndim)
11 print(d.ndim)
```

```
0
1
2
2
3
```

- **Output:**

## 2. Accessing Array (NumPy Array Indexing)

Ans:

- Access Array Elements

- Array indexing is the same as accessing an array element.
- You can access an array element by referring to its index number.
- The indexes in NumPy arrays start with 0, meaning that the first element has index 0, and the second has index 1 etc.

- Example:

```
1 import numpy as np
2
3 arr = np.array([1, 2, 3, 4])
4
5 print(arr[0])
```

- Output:

1

- Access 2-D Arrays

- To access elements from 2-D arrays we can use comma separated integers representing the dimension and the index of the element.
- Think of 2-D arrays like a table with rows and columns, where the dimension represents the row and the index represents the column.

- Example:

```
1 import numpy as np
2
3 arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
4
5 print('2nd element on 1st row: ', arr[0, 1])
```

- Output: 2nd element on 1st row: 2

### 3. Stacking & Splitting

Ans:

- **Stacking NumPy Arrays**

- Stacking is the concept of joining arrays in NumPy. Arrays having the same dimensions can be stacked. The stacking is done along a new axis. Stacking leads to increased customization of arrays.

- **Stacking and Joining in NumPy**

- NumPy implements the function of stacking.
- We can perform stacking along three dimensions:
  - **vstack()**: it performs vertical stacking along the rows.
  - **hstack()**: it performs horizontal stacking along with the columns.
  - **dstack()**: it performs in-depth stacking along a new third axis.
- NumPy stack function takes two input parameters:
  - the arrays for stacking and the axis for the resultant array.

- **Syntax:**

```
import numpy as np  
np.stack(array,axis)
```

- **NumPy vstack() Function**

- We use the vstack() function to sequentially stack arrays in a vertical order i.e. along the rows.
- This is a more specific concept of array stacking. It is very useful for arrays with up to 3 dimensions.
- The vsplit() function splits the array into a list of multiple sub-arrays vertically.



○ **Example:**

```
1 import numpy as np
2
3 arr1 = np.array([1, 2, 3])
4 arr2 = np.array([4, 5, 6])
5
6 arr = np.vstack((arr1, arr2))
7
8 print(arr)
```

○ **Output:**

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

○ **NumPy hstack() Function**

- We use this function to sequentially stack arrays in horizontal order i.e. along with the columns.
- This is a specific concept for column wise concatenation.
- We cannot perform horizontal stacking if the number of rows is not equal in both the arrays.

○ **Example:**

```
1 import numpy as np
2
3 arr1 = np.array([1, 2, 3])
4 arr2 = np.array([4, 5, 6])
5
6 arr = np.hstack((arr1, arr2))
7
8 print(arr)
```

○ **Output:**

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

## ○ NumPy dstack() Function

- This function is for stacking the arrays along depth.
- The resultant array is along a third new dimension.

### ○ Example:

```
1 import numpy as np
2
3 arr1 = np.array([1, 2, 3])
4 arr2 = np.array([4, 5, 6])
5
6 arr = np.dstack((arr1, arr2))
7
8 print(arr)
```

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

### ○ Output:

## ● Splitting NumPy Arrays

- Splitting is reverse operation of Joining.
- Joining merges multiple arrays into one and Splitting breaks one array into multiple.
- We use array\_split() for splitting arrays, we pass it the array we want to split and the number of splits.

### ○ Example:

```
1 import numpy as np
2
3 arr = np.array([1, 2, 3, 4, 5, 6])
4
5 newarr = np.array_split(arr, 3)
6
7 print(newarr)
```

### ○ Output:

```
[array([1, 2]), array([3, 4]), array([5, 6])]
```

## 4. Maths Functions

**Ans:**

- Numpy provides a wide range of mathematical functions that can be performed on arrays.
- Let's explore three different types of math functions in NumPy:
  1. Trigonometric Functions
  2. Arithmetic Functions
  3. Rounding Functions
- **Trigonometric Functions**
  - NumPy provides a set of standard trigonometric functions to calculate the trigonometric ratios (sine, cosine, tangent, etc.)
  - Here's a list of commonly used trigonometric functions in NumPy.

Trigonometric Function	Computes (in radians)
<b>sin()</b>	○ The sine of an angle
<b>cos()</b>	○ Cosine of an angle
<b>tan()</b>	○ Tangent of an angle
<b>arcsin()</b>	○ The inverse sine
<b>arccos()</b>	○ The inverse cosine
<b>arctan()</b>	○ The inverse tangent
<b>degrees()</b>	○ Converts an angle in radians to degrees
<b>radians()</b>	○ Converts an angle in degrees to radians

○ **Example:**

```
1 import numpy as np
2
3     # array of angles in radians
4 angles = np.array([1, 2])
5 print("Angles:", angles)
6
7     # compute the sine of the angles
8 sine_values = np.sin(angles)
9 print("Sine values:", sine_values)
```

○ **Output:**

```
Angles: [1 2]
Sine values: [0.84147098 0.90929743]
```

**AAVISHKAR**  
ENGINEERING CLASSES

*Work Smarter, Not Harder.*



## 5. Statistics Functions

**Ans:**

- **Statistics Functions:**

- Statistics involves gathering data, analyzing it, and drawing conclusions based on the information collected.
- NumPy provides us with various statistical functions that can perform statistical data analysis.

- **Common NumPy Statistical Functions**

Functions	Descriptions
median()	○ Return the median of an array
mean()	○ Return the mean of an array
std()	○ Return the standard deviation of an array
percentile()	○ Return the nth percentile of elements in an array
min()	○ Return the minimum element of an array
max()	○ Return the maximum element of an array

- **Example:** Compute Median for Odd Number of Elements

```

1 import numpy as np
2
3 # create a 1D array with 5 elements
4 array1 = np.array([1, 2, 3, 4, 5])
5
6 # calculate the median
7 median = np.median(array1)
8
9 print(median) # Output: 3.0

```

- **Output:**

**3.0**

## 2. Pandas:

**Ans:**

- Pandas is a Python library.
- Pandas is used to analyze data.
- Pandas is a Python library used for working with data sets.
- It has functions for analyzing, cleaning, exploring, and manipulating data.

### 1. Series

**Ans:**

- A Pandas Series is a one-dimensional labeled array-like object that can hold data of any type.
- A Pandas Series can be thought of as a column in a spreadsheet or a single column of a DataFrame.
- It consists of two main components:
  - Labels
  - Data
- **Example:** Create a simple Pandas Series from a list.

```
1 import pandas as pd
2
3 # create a list
4 data = [10, 20, 30, 40, 50]
5
6 # create a series from the list
7 my_series = pd.Series(data)
8
9 print(my_series)
```

- **Output:**

```
1    20
2    30
3    40
4    50
dtype: int64
```

- In this example,

- We created a Python list called data containing **five integer values**.
- We then passed this list to the **Series() function**, which converted it into a Pandas Series called **my\_series**.
- Here, **dtype: int64** denotes that the series stores the values of int64 types.

**AAVISHKAR**  
**ENGINEERING CLASSES**

*Work Smarter, Not Harder.*

## 2. DataFrames

**Ans:**

- A DataFrame is like a table where the data is organized in rows and columns.
- It is a two-dimensional data structure like a two-dimensional array.

○ For example,

	Country	Capital	Population
0	Canada	Ottawa	37742154
1	Australia	Canberra	25499884
2	UK	London	67886011
3	Brazil	Brasília	212559417

○ Here,

- Country, Capital and Population are the column names.
- Each row represents a record, with the index value on the left.
- The index values are auto-assigned starting from 0.
- Each column contains data of the same type.
- For instance, Country and Capital contain strings, and Population contains integers.

### • Create a Pandas DataFrame

○ We can create a Pandas DataFrame in the following ways:

1. Using Python Dictionary
2. Using Python List
3. From a File
4. Creating an Empty DataFrame



- **Pandas DataFrame Using Python Dictionary**

- We can create a dataframe using a dictionary by passing it to the DataFrame() function.
- For example,

```
1 import pandas as pd
2
3 # create a dictionary
4 data = {'Name': ['John', 'Alice', 'Bob'],
5         'Age': [25, 30, 35],
6         'City': ['New York', 'London', 'Paris']}
7
8 # create a dataframe from the dictionary
9 df = pd.DataFrame(data)
10
11 print(df)
```

- **Output:**

```
0   John  ...  New York
1  Alice  ...   London
2   Bob   ...   Paris

[3 rows x 3 columns]
```

○ In this example,

- we created a dictionary called data that contains the **column names** (Name, Age, City) as keys, and **lists of values** as their respective values.
- We then used the **pd.DataFrame() function** to convert the dictionary into a **DataFrame** called df.

### 3. Read CSV File

**Ans:**

- Pandas provides functions for both reading from and writing to CSV files.
- CSV stands for **Comma-Separated Values**.
- It is a popular file format used for storing tabular data, where each row represents a record, and columns are separated by a delimiter (generally a comma).
- **For example, contents of a CSV file may look like,**

```
Employee ID,First Name,Last Name,Department,Position,Salary
101,John,Doe,Marketing,Manager,50000
102,Jane,Smith,Sales,Associate,35000
103,Michael,Johnson,Finance,Analyst,45000
104,Emily,Williams,HR,Coordinator,40000
```

- **Read CSV Files**
  - In Pandas, the `read_csv()` function allows us to read data from a CSV file into a DataFrame.
  - It automatically detects commas and parses the data into appropriate columns.
  - **Here's an example of reading a CSV file using Pandas:**

```
1 import pandas as pd
2
3 # read csv file
4 df = pd.read_csv('data.csv', header = 0)
5
6 print(df)
```

- **Output:**

	Employee ID	First Name	Last Name	Department	Position	Salary
0	101	John	Doe	Marketing	Manager	50000
1	102	Jane	Smith	Sales	Associate	35000
2	103	Michael	Johnson	Finance	Analyst	45000
3	104	Emily	Williams	HR	Coordinator	40000

- The above code reads the contents of the **data.csv** file and creates a **DataFrame** named **df** containing the data from the **CSV file**.
- Here, **header = 0** sets the first row as the header of the dataframe.
- The contents of the **data.csv** file are the same as the contents of the CSV file provided in the introduction section.
- **Write to CSV Files**
  - We used **read\_csv()** to read data from a CSV file into a DataFrame.
  - Pandas also provides the **to\_csv()** function to write data from a DataFrame into a CSV file.
  - Let's see an example.

```
1 import pandas as pd
2
3 # create a dictionary
4 data = {'Name': ['John', 'Alice', 'Bob'],
5         'Age': [25, 30, 35],
6         'City': ['New York', 'London', 'Paris']}
7
8 # create a dataframe from the dictionary
9 df = pd.DataFrame(data)
10
11 # write dataframe to csv file
12 df.to_csv('output.csv', index=False)
```

○ **Output:**

```
1 Name, Age, City
2 John, 25, New York
3 Alice, 30, London
4 Bob, 35, Paris
```

- Here, the above code writes the DataFrame **df** to the **output.csv** file.
- The **index=False** parameter is used to exclude the index labels from the CSV file.

## 4. Cleaning Empty Cells

Ans:

- **Empty Cells**

- Empty cells can potentially give you a wrong result when you analyze data.

- **Remove Rows**

- One way to deal with empty cells is to remove rows that contain empty cells.
- This is usually OK, since data sets can be very big, and removing a few rows will not have a big impact on the result.
- **Note:** By default, the `dropna()` method returns a new DataFrame, and will not change the original.
- If you want to change the original DataFrame, use the **`inplace = True`** argument:
- **Example:** Remove all rows with NULL values:

```
1 import pandas as pd
2
3 df = pd.read_csv('data.csv')
4
5 df.dropna(inplace = True)
6
7 print(df.to_string())
```

- **Replace Empty Values**

- Another way of dealing with empty cells is to insert a new value instead.
- This way you do not have to delete entire rows just because of some empty cells.
- The `fillna()` method allows us to replace empty cells with a value:



○ **Example:**

```
1 import pandas as pd
2
3 df = pd.read_csv('data.csv')
4
5 df.fillna(130, inplace = True)
```

• **Replace Only For Specified Columns**

- The example above replaces all empty cells in the whole Data Frame.
- To only replace empty values for one column, specify the column name for the DataFrame:

```
1 import pandas as pd
2
3 df = pd.read_csv('data.csv')
4
5 df["Calories"].fillna(130, inplace = True)
```

**AAVISHKAR**  
**ENGINEERING CLASSES**

*Work Smarter, Not Harder.*

## 5. Cleaning Wrong Data

**Ans:**

- "Wrong data" does not have to be "empty cells" or "wrong format", it can just be wrong, like if someone registered "199" instead of "1.99".
- Sometimes you can spot wrong data by looking at the data set, because you have an expectation of what it should be.
- If you take a look at our data set, you can see that in row 7, the duration is 450, but for all the other rows the duration is between 30 and 60.
- It doesn't have to be wrong, but taking in consideration that this is the data set of someone's workout sessions, we conclude with the fact that this person did not work out in 450 minutes.

	Duration	Date	Pulse	Maxpulse	Calories
0	60	'2020/12/01'	110	130	409.1
1	60	'2020/12/02'	117	145	479.0
2	60	'2020/12/03'	103	135	340.0
3	45	'2020/12/04'	109	175	282.4
4	45	'2020/12/05'	117	148	406.0
5	60	'2020/12/06'	102	127	300.0
6	60	'2020/12/07'	110	136	374.0
7	450	'2020/12/08'	104	134	253.3
8	30	'2020/12/09'	109	133	195.1
9	60	'2020/12/10'	98	124	269.0
10	60	'2020/12/11'	103	147	329.3

- **Replacing Values**

- One way to fix wrong values is to replace them with something else.

- In our example, it is most likely a typo, and the value should be "45" instead of "450", and we could just insert "45" in row 7:
- **Example:**

```
1 import pandas as pd
2
3 df = pd.read_csv('data.csv')
4
5 df.loc[7, 'Duration'] = 45
6
7 print(df.to_string())
```

- **Output:**

	Duration	Date	Pulse	Maxpulse	Calories
0	60	'2020/12/01'	110	130	409.1
1	60	'2020/12/02'	117	145	479.0
2	60	'2020/12/03'	103	135	340.0
3	45	'2020/12/04'	109	175	282.4
4	45	'2020/12/05'	117	148	406.0
5	60	'2020/12/06'	102	127	300.0
6	60	'2020/12/07'	110	136	374.0
7	45	'2020/12/08'	104	134	253.3
8	30	'2020/12/09'	109	133	195.1
9	60	'2020/12/10'	98	124	269.0
10	60	'2020/12/11'	103	147	329.3

*Work Smarter, Not Harder.*

## 6. Removing Duplicates

Ans:

- **Discovering Duplicates**

- Duplicate rows are **rows that have been registered more than one time.**
- By taking a look at our test data set, we can assume that **row 11 and 12 are duplicates.**

	Duration	Date	Pulse	Maxpulse	Calories
0	60	'2020/12/01'	110	130	409.1
1	60	'2020/12/02'	117	145	479.0
2	60	'2020/12/03'	103	135	340.0
3	45	'2020/12/04'	109	175	282.4
4	45	'2020/12/05'	117	148	406.0
5	60	'2020/12/06'	102	127	300.0
6	60	'2020/12/07'	110	136	374.0
7	450	'2020/12/08'	104	134	253.3
8	30	'2020/12/09'	109	133	195.1
9	60	'2020/12/10'	98	124	269.0
10	60	'2020/12/11'	103	147	329.3
11	60	'2020/12/12'	100	120	250.7
12	60	'2020/12/12'	100	120	250.7

- To discover duplicates, we can use the **df.duplicated()** method.
- The **df.duplicated()** method returns a Boolean values for each row:
- **Example:**

```
1 import pandas as pd
2
3 df = pd.read_csv('data.csv')
4
5 print(df.duplicated())
```



```
0 False
1 False
2 False
3 False
4 False
5 False
6 False
7 False
8 False
9 False
10 False
11 False
12 True
```

○ **Output:**

- **Removing Duplicates**

- To remove duplicates, use the `drop_duplicates()` method.
- **Example:** Remove all duplicates:

```
1 import pandas as pd
2
3 df = pd.read_csv('data.csv')
4
5 df.drop_duplicates(inplace = True)
6
7 print(df.to_string())
```

○ **Output:**

# Notice that row 12 has been removed from the result

	Duration	Date	Pulse	Maxpulse	Calories
0	60	'2020/12/01'	110	130	409.1
1	60	'2020/12/02'	117	145	479.0
2	60	'2020/12/03'	103	135	340.0
3	45	'2020/12/04'	109	175	282.4
4	45	'2020/12/05'	117	148	406.0
5	60	'2020/12/06'	102	127	300.0
6	60	'2020/12/07'	110	136	374.0
7	450	'2020/12/08'	104	134	253.3
8	30	'2020/12/09'	109	133	195.1
9	60	'2020/12/10'	98	124	269.0
10	60	'2020/12/11'	103	147	329.3
11	60	'2020/12/12'	100	120	250.7
13	60	'2020/12/13'	106	128	345.3

## 7. Pandas Plotting

Ans:

- Pandas uses the **plot()** method to create diagrams.
- We can use Pyplot, a submodule of the **Matplotlib** library to visualize the diagram on the screen.
- **Scatter Plot**
  - Specify that you want a scatter plot with the kind argument:  
**kind = 'scatter'**
  - A scatter plot needs an x- and a y-axis.
  - In the example below we will use "Duration" for the x-axis and "Calories" for the y-axis.
  - Include the x and y arguments like this:

**x = 'Duration', y = 'Calories'**

- **Example:**

```
#Three lines to make our compiler able to draw:
import sys
import matplotlib
matplotlib.use('Agg')

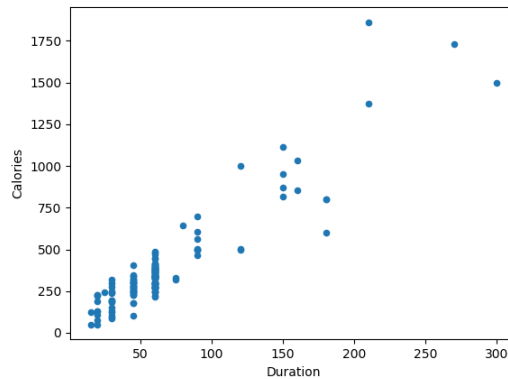
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv('data.csv')

df.plot(kind = 'scatter', x = 'Duration', y = 'Calories')

plt.show()

#Two lines to make our compiler able to draw:
plt.savefig(sys.stdout.buffer)
sys.stdout.flush()
```



○ **Output:**

● **Histogram**

- Use the kind argument to specify that you want a histogram:

**kind = 'hist'**

- A histogram needs only one column.
- A histogram shows us the frequency of each interval, e.g. how many workouts lasted between 50 and 60 minutes?
- In the example below we will use the "Duration" column to create the histogram:

- **Example:**

```
#Three lines to make our compiler able to draw:
import sys
import matplotlib
matplotlib.use('Agg')

import pandas as pd
import matplotlib.pyplot as plt

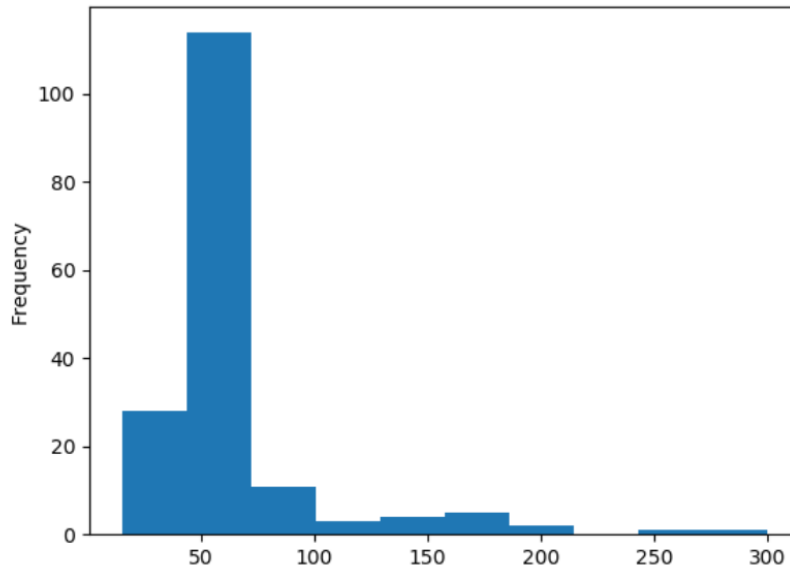
df = pd.read_csv('data.csv')

df["Duration"].plot(kind = 'hist')

plt.show()

#Two lines to make our compiler able to draw:
plt.savefig(sys.stdout.buffer)
sys.stdout.flush()
```

○ **Output:**



**AAVISHKAR**  
ENGINEERING CLASSES

*Work Smarter, Not Harder.*



## 3. Matplotlib:

Ans:

- Matplotlib is a **low level graph plotting library** in python that serves as a **visualization utility**.
- Matplotlib is **open source** and we can use it freely.
- Matplotlib is mostly written in python, a few segments are written in C, Objective-C and Javascript for Platform compatibility.

### 1. Pyplot.plot:

Ans:

- **Pyplot**
  - Most of the Matplotlib utilities lies under the **pyplot** submodule, and are usually imported under the **plt** alias:

```
import matplotlib.pyplot as plt
```

### • Plotting x and y points

- The **plot()** function is used to draw points (markers) in a diagram.
- By default, the **plot()** function draws a line from point to point.
- The function takes parameters for specifying points in the diagram.
- **Parameter 1** is an array containing the points on the **x-axis**.
- **Parameter 2** is an array containing the points on the **y-axis**.
- If we need to plot a line from (1, 3) to (8, 10), we have to pass two arrays [1, 8] and [3, 10] to the plot function.

- **Example:** Draw a line in a diagram from position (1, 3) to position (8, 10):

```
#Three lines to make our compiler able to draw:
import sys
import matplotlib
matplotlib.use('Agg')

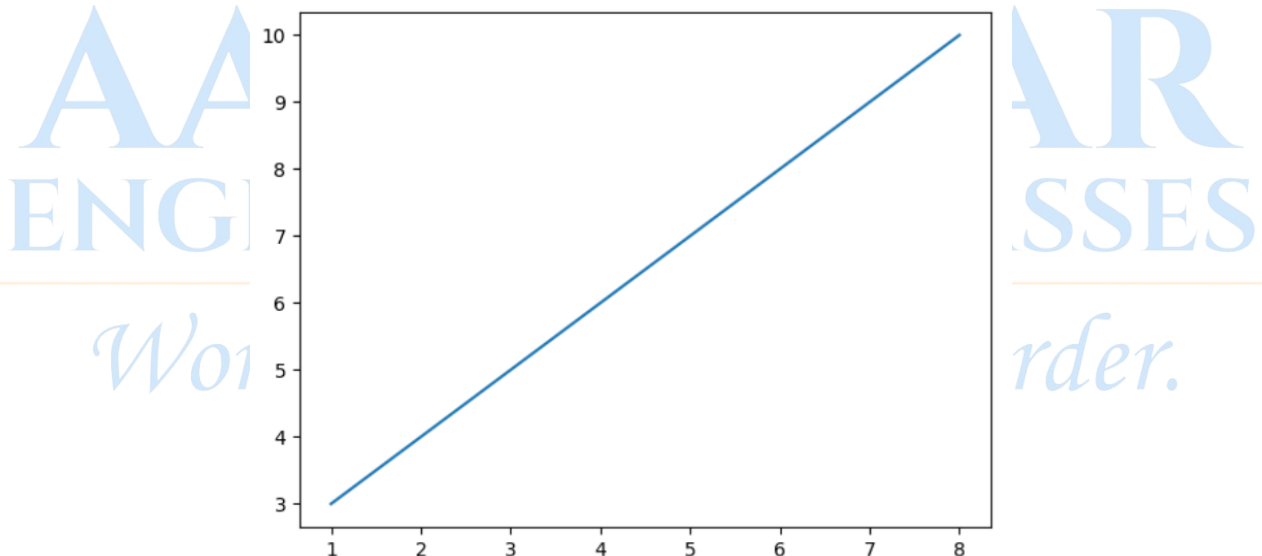
import matplotlib.pyplot as plt
import numpy as np

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

plt.plot(xpoints, ypoints)
plt.show()

#Two lines to make our compiler able to draw:
plt.savefig(sys.stdout.buffer)
sys.stdout.flush()
```

- **Output:**



## 2. labels

Ans:

- **Create Labels for a Plot**

- With Pyplot, you can use the **xlabel()** and **ylabel()** functions to set a label for the x- and y-axis.

- **Create a Title for a Plot**

- With Pyplot, you can use the **title()** function to set a title for the plot.

- **Set Font Properties for Title and Labels**

- You can use the fontdict parameter in **xlabel()**, **ylabel()**, and **title()** to set font properties for the title and labels.
- **Example:**

```
#Three lines to make our compiler able to draw:
import sys
import matplotlib
matplotlib.use('Agg')

import numpy as np
import matplotlib.pyplot as plt

x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

font1 = {'family':'serif','color':'blue','size':20}
font2 = {'family':'serif','color':'darkred','size':15}

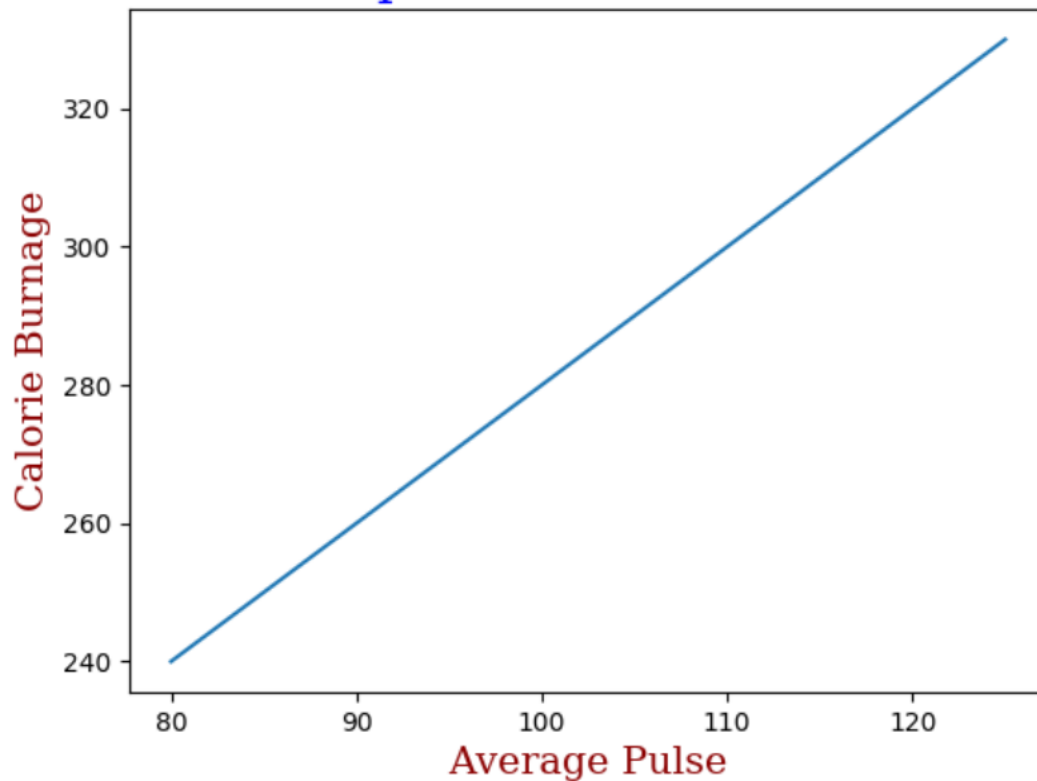
plt.title("Sports Watch Data", fontdict = font1)
plt.xlabel("Average Pulse", fontdict = font2)
plt.ylabel("Calorie Burnage", fontdict = font2)

plt.plot(x, y)
plt.show()

#Two lines to make our compiler able to draw:
plt.savefig(sys.stdout.buffer)
sys.stdout.flush()
```

○ **Output:**

## Sports Watch Data



**AAVISHKAR**  
ENGINEERING CLASSES

*Work Smarter, Not Harder.*



### 3. grid

Ans:

- **Add Grid Lines to a Plot**
  - With Pyplot, you can use the **grid()** function to add grid lines to the plot.
- **Specify Which Grid Lines to Display**
  - You can use the axis parameter in the grid() function to specify which grid lines to display.
  - Legal values are: 'x', 'y', and 'both'.
  - Default value is 'both'.
- **Set Line Properties for the Grid**
  - You can also set the line properties of the grid, like this: **grid(color = 'color', linestyle = 'linestyle', linewidth = number)**.
- **Example:**

```
#Three lines to make our compiler able to draw:
import sys
import matplotlib
matplotlib.use('Agg')

import numpy as np
import matplotlib.pyplot as plt

x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

plt.title("Sports Watch Data")
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")

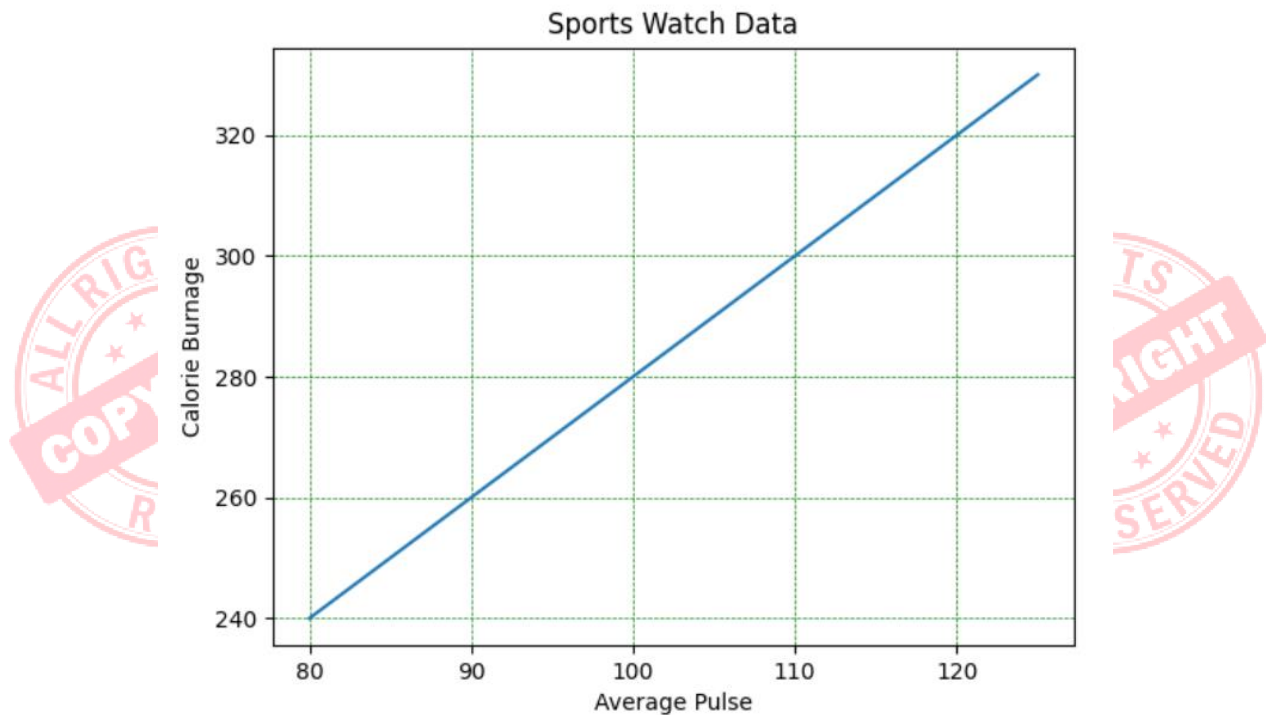
plt.plot(x, y)

plt.grid(color = 'green', linestyle = '--', linewidth = 0.5)

plt.show()

#Two lines to make our compiler able to draw:
plt.savefig(sys.stdout.buffer)
sys.stdout.flush()
```

- **Output:**



# AAVISHKAR

## ENGINEERING CLASSES

*Work Smarter, Not Harder.*

📍: 202, I - Address. Panchamrut Bunglows II, Sola Bridge S.G.Highway,  
Opp. Science City Approach BRTS Stop, Science City, Ahmedabad -380060.

☎: + 91 7600449191, ✉: aavishkar2406@gmail.com

## 4. bars

**Ans:**

- **Creating Bars**

- With Pyplot, you can use the bar() function to draw bar graphs:
- The bar() function takes arguments that describes the layout of the bars.
- The categories and their values represented by the first and second argument as arrays.

- **Horizontal Bars**

- If you want the bars to be displayed horizontally instead of vertically, use the barh() function:

**plt.barh(x, y)**

- **Bar Color**

- The bar() and barh() take the keyword argument color to set the color of the bars:

**plt.bar(x, y, color = "red")**

- **Bar Width**

- The bar() takes the keyword argument width to set the width of the bars:

**plt.bar(x, y, width = 0.1)**

- **Bar Height**

- The barh() takes the keyword argument height to set the height of the bars:

**plt.barh(x, y, height = 0.1)**

- **Example:**

```
#Three lines to make our compiler able to draw:
import sys
import matplotlib
matplotlib.use('Agg')

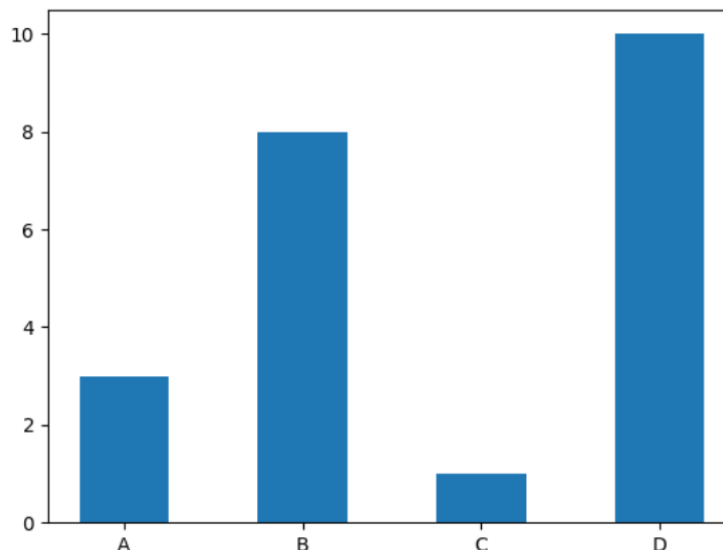
import matplotlib.pyplot as plt
import numpy as np

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.bar(x, y, width = 0.5)
plt.show()

#Two lines to make our compiler able to draw:
plt.savefig(sys.stdout.buffer)
sys.stdout.flush()
```

- **Output:**

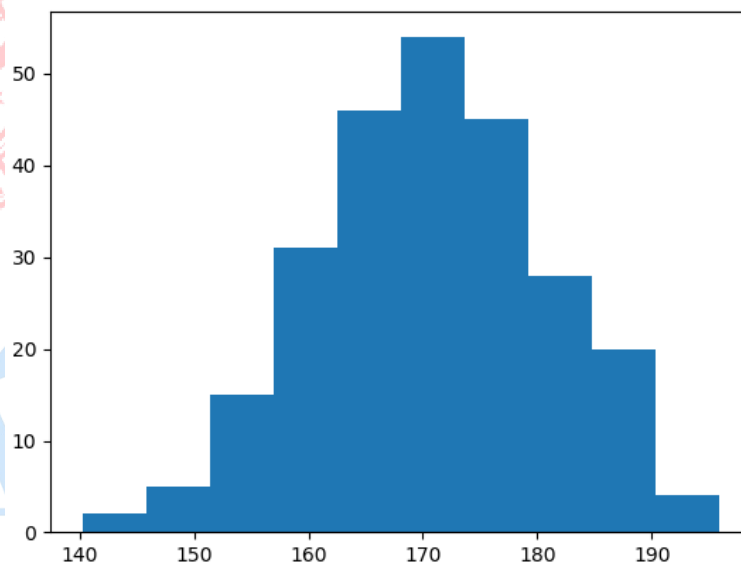




## 5. Histogram

**Ans:**

- A **histogram** is a **graph showing frequency distributions**.
- It is a graph showing the number of observations within each given interval.
  - **Example:** Say you ask for the height of 250 people, you might end up with a histogram like this:



- You can read from the histogram that there are approximately:
  - 2 people from 140 to 145cm
  - 5 people from 145 to 150cm
  - 15 people from 151 to 156cm
  - 31 people from 157 to 162cm
  - 46 people from 163 to 168cm
  - 53 people from 168 to 173cm
  - 45 people from 173 to 178cm
  - 28 people from 179 to 184cm
  - 21 people from 185 to 190cm
  - 4 people from 190 to 195cm

- **Create Histogram**

- In Matplotlib, we use the **hist()** function to create histograms.
- The **hist()** function will use an array of numbers to create a histogram, the array is sent into the function as an argument.

- **Example:**

```
#Three lines to make our compiler able to draw:
import sys
import matplotlib
matplotlib.use('Agg')

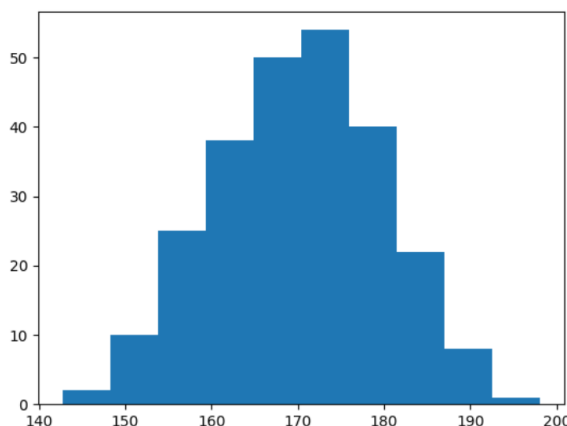
import matplotlib.pyplot as plt
import numpy as np

x = np.random.normal(170, 10, 250)

plt.hist(x)
plt.show()

#Two lines to make our compiler able to draw:
plt.savefig(sys.stdout.buffer)
sys.stdout.flush()
```

- **Output:**



## 6. Subplot

Ans:

- **Display Multiple Plots**
  - With the **subplot()** function you can draw multiple plots in one figure
- **The subplot() Function**
  - The **subplot()** function takes three arguments that describes the layout of the figure.
  - The layout is organized in **rows and columns**, which are represented by the **first and second argument**.
  - The **third argument represents the index of the current plot**.
- **Title**
  - You can add a title to each plot with the **title()** function.
- **Super Title**
  - You can add a title to the **entire figure with the suptitle()** function.
- **Example:**

```
#Three lines to make our compiler able to draw:
import sys
import matplotlib
matplotlib.use('Agg')

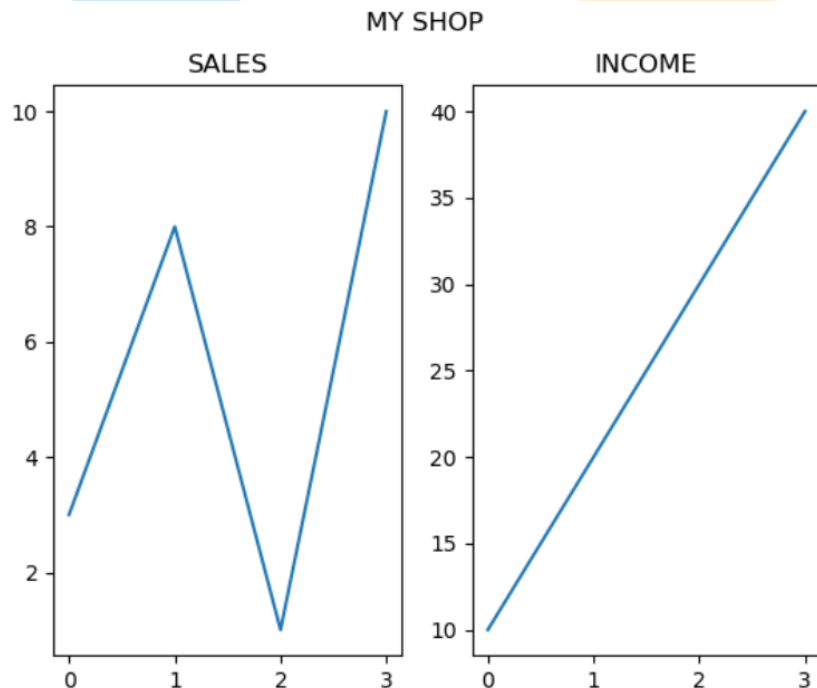
import matplotlib.pyplot as plt
import numpy as np

#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()  
  
#Two lines to make our compiler able to draw:  
plt.savefig(sys.stdout.buffer)  
sys.stdout.flush()
```

- **Output:**





## 7. pie chart

Ans:

- **Creating Pie Charts**

- With Pyplot, you can use the pie() function to draw pie charts

- **Labels**

- Add labels to the pie chart with the label parameter.
- The label parameter must be an array with one label for each wedge

- **Start Angle**

- As mentioned the default start angle is at the x-axis, but you can change the start angle by specifying a startangle parameter.
- The startangle parameter is defined with an angle in degrees, default angle is 0

**plt.pie(y, labels = mylabels, startangle = 90)**

- **Explode**

- Maybe you want one of the wedges to stand out? The explode parameter allows you to do that.
- The explode parameter, if specified, and not None, must be an array with one value for each wedge.
- Each value represents how far from the center each wedge is displayed

**myexplode = [0.2, 0, 0, 0]**

- **Shadow**

- Add a shadow to the pie chart by setting the shadows parameter to True.

**plt.pie(y, labels = mylabels, explode = myexplode, shadow = True)**

- **Colors**

- You can set the color of each wedge with the colors parameter.
- The colors parameter, if specified, must be an array with one value for each wedge

```
mycolors = ["black", "hotpink", "b", "#4CAF50"]  
plt.pie(y, labels = mylabels, colors = mycolors)
```

- **Legend**

- To add a list of explanation for each wedge, use the legend() function

```
plt.legend()
```

- **Legend With Header**

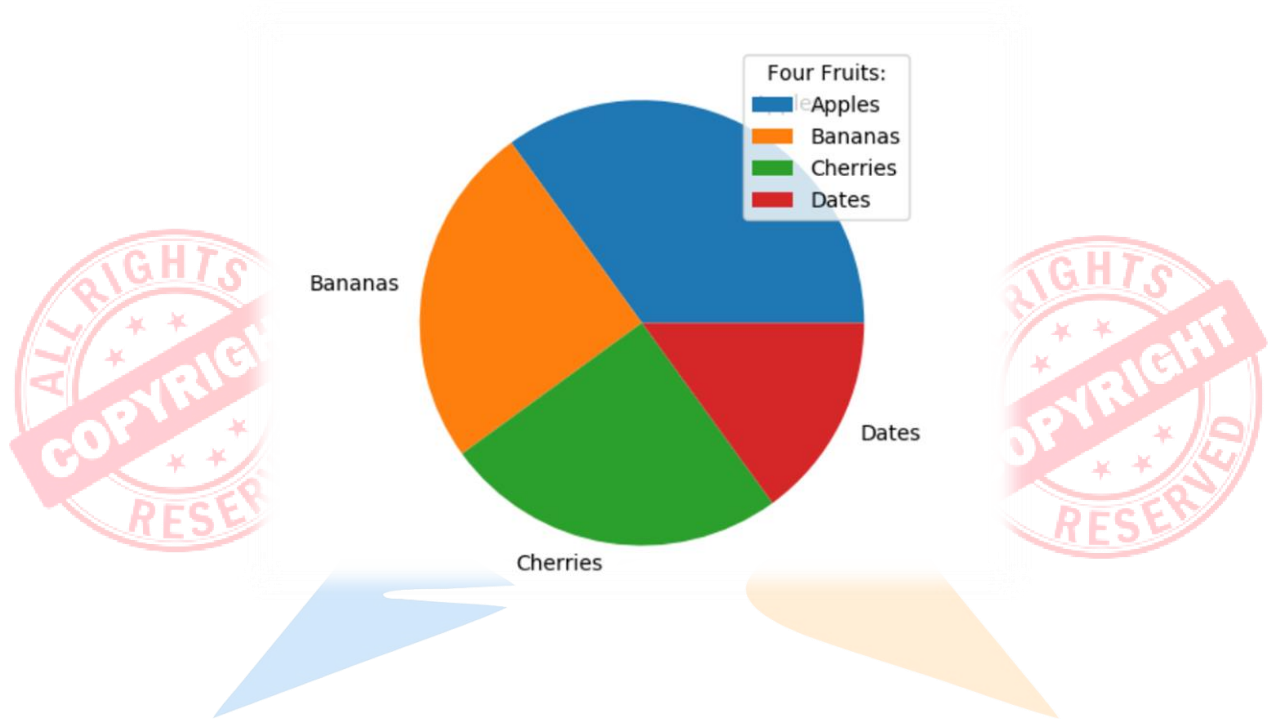
- To add a header to the legend, add the title parameter to the legend function.

```
plt.legend(title = "Four Fruits:")
```

- **Example:**

```
#Three lines to make our compiler able to draw:  
import sys  
import matplotlib  
matplotlib.use('Agg')  
  
import matplotlib.pyplot as plt  
import numpy as np  
  
y = np.array([35, 25, 25, 15])  
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]  
  
plt.pie(y, labels = mylabels)  
plt.legend(title = "Four Fruits:")  
plt.show()  
  
#Two lines to make our compiler able to draw:  
plt.savefig(sys.stdout.buffer)  
sys.stdout.flush()
```

- **Output:**



# AAVISHKAR ENGINEERING CLASSES

*Work Smarter, Not Harder.*

## 4. Sklearn:

**Ans:**

- Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python.
- It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction via a consistency interface in Python.
- This library, which is largely written in Python, is built upon NumPy, SciPy and Matplotlib.

### 1. Key concepts and features

**Ans:**

- **Key concepts:**
  - Algorithms for making decisions, such as:
    - Data are identified and categorised by classification as per the patterns.
    - Regression is the process of forecasting or predicting data values using the historical and anticipated data average.
    - Clustering is the automatic collection of datasets with related data.
    - Predictive analysis is supported by various algorithms, including neural networks for pattern recognition and straightforward linear regression.
- **Features:**
  - **Supervised Learning algorithms:**
    - Almost all the popular supervised learning algorithms, like Linear Regression, Support Vector Machine (SVM), Decision Tree etc., are the part of scikit-learn.



- **Unsupervised Learning algorithms:**

- On the other hand, it also has all the popular unsupervised learning algorithms from clustering, factor analysis, PCA (Principal Component Analysis) to unsupervised neural networks.

- **Clustering**

- This model is used for grouping unlabeled data.

- **Cross Validation**

- It is used to check the accuracy of supervised models on unseen data.

- **Dimensionality Reduction:**

- It is used for reducing the number of attributes in data which can be further used for summarization, visualization and feature selection.

- **Ensemble methods:**

- As name suggest, it is used for combining the predictions of multiple supervised models.

- **Feature extraction:**

- It is used to extract the features from data to define the attributes in image and text data.

- **Feature selection:**

- It is used to identify useful attributes to create supervised models.

- **Open Source:**

- It is open-source library and also commercially usable under BSD license.



## 2. Steps to Build a Model in Sklearn:

### i. Loading a Dataset

### ii. Train\_test\_split

**Ans:**

### Step 1: Loading a Dataset

- Simply put, a dataset is a collection of sample data points. A dataset typically consists of two primary parts:
- **Features:**
  - Features are essentially the variables in our dataset, often called predictors, data inputs, or attributes.
  - A feature matrix, which is frequently symbolised by the letter "X," can be used to represent them since many of them may exist.
  - The term "feature names" refers to a list of names of all the features.
- **Response:**
  - (sometimes referred to as the target feature, label, or output) Based on the variables feature, this variable is the output.
  - In most cases, we only have one response column, which is depicted by a response column or vector (the letter 'y' is frequently used to denote a response vector).
  - Target names refer to all the various values a response vector could take.

### Step 2: Splitting the Dataset

- The correctness of each machine learning model is a crucial consideration.

- Now, one may train a model with the provided dataset and then use that model to predict the target values for another set of the dataset to ascertain the correctness of the model.
- **To sum it up:**
  - Make a training dataset and a testing dataset out of the given dataset.
  - On the practise set, train the model.
  - Test the model using the testing dataset and assess its performance.

### Step 3: Training the Model

- It's time to use the training dataset to train the model, which will make predictions.
- A variety of machine learning techniques with an easy-to-use interface for fitting, prediction accuracy, etc., are offered by Scikit-learn.
- Our classifier must now be tested using the testing dataset.
- For this, we can use the `.predict()` model class method, giving back the predicted values.
- By comparing the actual values of the testing dataset and the predicted values, we can assess the model's performance with the help of sklearn methods.
- The `accuracy_score` function of the metrics package is used for this.

## • Subject Overview:

Unit No.	Unit Title	Total Marks
1	Introduction to Machine Learning	08
2	Preparing to Model	12
3	Modeling and Evaluation	12
4	Supervised Learning - Classification and Regression	14
5	Unsupervised Learning	12
6	Python Libraries for Machine Learning	12

**AAVISHKAR**  
ENGINEERING CLASSES

*Work Smarter, Not Harder.*

**DIPLOMA, DEGREE AND M.E. CLASSES**

**ALL SUBJECTS OF ALL SEMESTERS**



AAVISHKAR2406



AAVISHKAR2406

📍: 202, I - Address. Panchamrut Bunglows II, Sola Bridge S.G.Highway,  
Opp. Science City Approach BRTS Stop, Science City, Ahmedabad -380060.

☎ : + 91 7600449191 , ✉ : aavishkar2406@gmail.com



