UCS2612 Machine Learning Laboratory

Assignment1 : Working with Python packages - Numpy, Scipy, Scikit-learn, Matplotlib

Name : **Mega V**

Roll No : **3122 21 5001 051**

Aim :

To learning the python packages and working with that and to Explore the steps involved in the Learning process. For a dataset

Explore the various functions / methods that come under the following Python Libraries

a) Numpy – Numerical Python

Used for working with arrays.

Functions:

- Create a NumPy array

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

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

- Array Slicing

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

```
[5 6 7]
```

- Splitting is reverse operation of Joining

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

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

- Sorting arrays

```
import numpy as np
arr1 = np.array([3, 2, 0, 1])
arr2 = np.array(['banana', 'cherry', 'apple'])
arr3 = np.array([True, False, True])
print(np.sort(arr1))
print(np.sort(arr2))
print(np.sort(arr3))
```

```
[0 1 2 3]
['apple' 'banana' 'cherry']
[False  True  True]
```

- Filering Arrays

```
import numpy as np
arr = np.array([41, 42, 43, 44])
x = [True, False, True, False
newarr = arr[x]
print(newarr)
```

```
[41 43]
```

- Access Array Elements

```
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])
```

```
2nd element on 1st dim:  2
```

- Copy

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
x = arr.copy()
arr[0] = 42
print(arr)
print(x)
```

```
[42  2  3  4  5]
[1 2 3 4 5]
```

- View

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
x = arr.view()
arr[0] = 42
print(arr)
print(x)
```

```
[42  2  3  4  5]
[42  2  3  4  5]
```

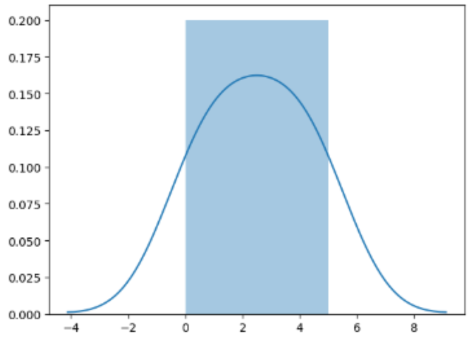
- Make changes in view

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
x = arr.view()
x[0] = 31
print(arr)
print(x)
```

```
[31  2  3  4  5]
[31  2  3  4  5]
```

- Seaborn - Seaborn is a library that uses Matplotlib underneath to plot graphs. It will be used to visualize random distributions.

```
import matplotlib.pyplot as plt
import seaborn as sns
sns.distplot([0, 1, 2, 3, 4, 5])
plt.show()
```



- Normal distribution
  - Use the `random.normal()` method to get a Normal Data Distribution.
  - It has three parameters:
    - `loc` - (Mean) where the peak of the bell exists.
    - `scale` - (Standard Deviation) how flat the graph distribution should be.
    - `size` - The shape of the returned array.

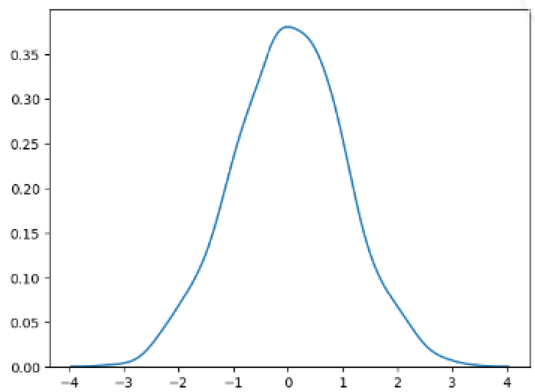
```
from numpy import random
x = random.normal(loc=1, scale=2, size=(2, 3))
print(x)
```

```
[[0.78604046 3.64592481 4.45391188]
 [0.68918731 1.22383744 4.65969041]]
```

```
from numpy import random
x = random.normal(size=(2, 3))
print(x)
```

```
[[ 0.70074132  0.24760778 -0.52154205]
 [-0.930514   -0.0909822  -0.95816753]]
```

```
from numpy import random
import matplotlib.pyplot as plt
import seaborn as sns
sns.distplot(random.normal(size=1000), hist=False)
plt.show()
```



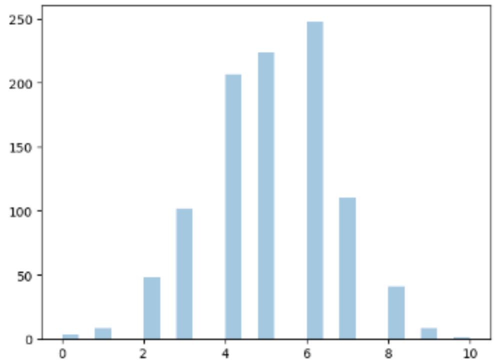
- Binomial Distribution
  - It describes the outcome of binary scenarios, e.g. toss of a coin, it will either be head or tails.

- It has three parameters:
    - n - number of trials.
    - p - probability of occurence of each trial (e.g. for toss of a coin 0.5 each).
    - size - The shape of the returned array.

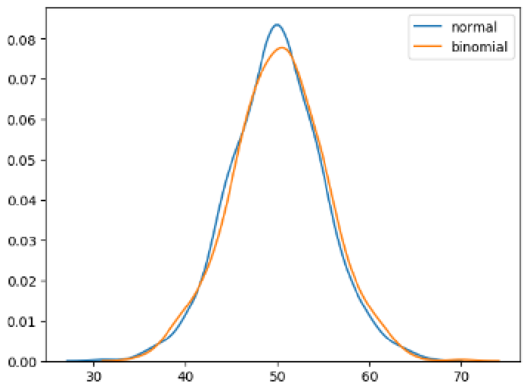
```
from numpy import random
x = random.binomial(n=10, p=0.5, size=10)
print(x)
```

```
[7 5 7 9 6 4 6 4 3 5]
```

```
from numpy import random
import matplotlib.pyplot as plt
import seaborn as sns
sns.distplot(random.binomial(n=10, p=0.5, size=1000),
hist=True, kde=False)
plt.show()
```




```
from numpy import random
import matplotlib.pyplot as plt
import seaborn as sns
sns.distplot(random.normal(loc=50, scale=5,
size=1000), hist=False, label='normal')
sns.distplot(random.binomial(n=100, p=0.5,
size=1000), hist=False, label='binomial')
plt.show()
```



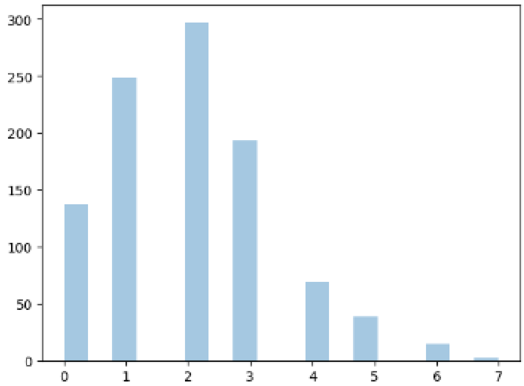
- Poisson Distribution
    - It estimates how many times an event can happen in a specified time. e.g. If someone eats twice a day what is the probability he will eat thrice?
    - It has two parameters:

- `lam` - rate or known number of occurrences e.g. 2 for above problem.
- `size` - The shape of the returned array.

```
from numpy import random
x = random.poisson(lam=2, size=10)
print(x)
```

```
[1 5 4 1 3 0 0 1 3 5]
```

```
from numpy import random
import matplotlib.pyplot as plt
import seaborn as sns
sns.distplot(random.poisson(lam=2, size=1000), kde=False)
plt.show()
```

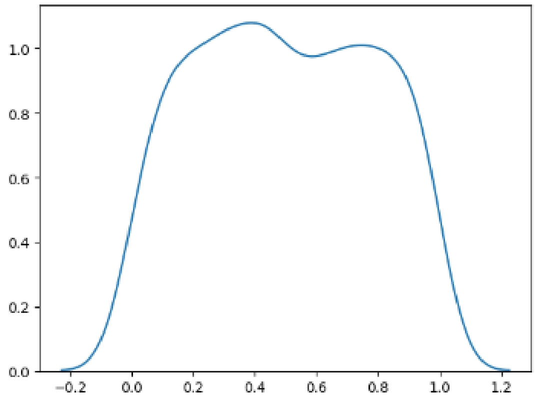


- Uniform Distribution
  - Used to describe probability where every event has equal chances of occuring.
  - E.g. Generation of random numbers.
  - It has three parameters:
    - `a` - lower bound - default 0 .0.
    - `b` - upper bound - default 1.0.
    - `size` - The shape of the returned array.

```
from numpy import random
x = random.uniform(size=(2, 3))
print(x)
```

```
[[0.8300512  0.23174099 0.13371372]
 [0.56232815 0.84961613 0.58218586]]
```

```
from numpy import random
import matplotlib.pyplot as plt
import seaborn as sns
sns.distplot(random.uniform(size=1000), hist=False)
plt.show()
```



- Exponential Distribution
    - Exponential distribution is used for describing time till next event e.g. failure/success etc.
    - It has two parameters:
        - scale - inverse of rate (see lam in poisson distribution) defaults to 1.0.
        - size - The shape of the returned array.

```
from numpy import random
x = random.exponential(scale=2, size=(2, 3))
print(x)
```

```
[[2.08340211 3.71240206 0.21520718]
 [1.78793833 4.17123932 5.64152689]]
```

- Summations

```
import numpy as np
arr1 = np.array([1, 2, 3])
arr2 = np.array([1, 2, 3])
newarr = np.add(arr1, arr2)
print(newarr)
```

```
[2 4 6]
```

```
import numpy as np
arr1 = np.array([1, 2, 3])
arr2 = np.array([1, 2, 3])
newarr = np.sum([arr1, arr2])
print(newarr)
```

```
12
```

```
import numpy as np
arr = np.array([1, 2, 3])
newarr = np.cumsum(arr)
print(newarr)
```

```
[1 3 6]
```

- Trigonometric

```
import numpy as np
arr = np.array([np.pi/2, np.pi/3, np.pi/4, np.pi/5])
x = np.sin(arr)
print(x)
```

```
[1.         0.8660254  0.70710678 0.58778525]
```

b) Pandas

Used to analyze data.

Functions:

- Create an alias with the as keyword

```
import pandas as pd
mydataset = {
  'cars': ["BMW", "Volvo", "Ford"],
  'passings': [3, 7, 2]
}
myvar = pd.DataFrame(mydataset)
print(myvar)
```

```
    cars  passings
0    BMW         3
1  Volvo         7
2   Ford         2
```

- Create simple pandas series from the list.

```
import pandas as pd
a = [1, 7, 2]
myvar = pd.Series(a)
print(myvar)
```

```
0    1
1    7
2    2
dtype: int64
```

- Index argument

```
import pandas as pd
a = [1, 7, 2]
myvar = pd.Series(a, index = ["x", "y", "z"])
print(myvar)
```

```
x    1
y    7
z    2
dtype: int64
```

```
import pandas as pd
calories = {"day1": 420, "day2": 380, "day3": 390}
myvar = pd.Series(calories, index = ["day1", "day2"])
print(myvar)
```

```
day1    420
day2    380
dtype: int64
```

- DataFrame from two series

```
import pandas as pd
data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}
df = pd.DataFrame(data)
print(myvar)
```

```
   calories  duration
0       420        50
1       380        40
2       390        45
```

- Locate row

```python
print(df.loc[0])S
```

```
calories    420
duration     50
Name: 0, dtype: int64
```

```python
print(df.loc[[0, 1]])
```

```
   calories  duration
0       420        50
1       380        40
```

- Named indexes

```python
import pandas as pd
data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}
df = pd.DataFrame(data, index =
["day1", "day2", "day3"])
print(df)
```

```
      calories  duration
day1       420        50
day2       380        40
day3       390        45
```

- Load files into a dataframe

```python
import pandas as pd
df = pd.read_csv('data.csv')
print(df)
```

```
     Duration  Pulse  Maxpulse  Calories
0          60    110       130     409.1
1          60    117       145     479.0
2          60    103       135     340.0
3          45    109       175     282.4
4          45    117       148     406.0
..        ...    ...       ...       ...
164        60    105       140     290.8
165        60    110       145     300.4
166        60    115       145     310.2
167        75    120       150     320.4
168        75    125       150     330.4

[169 rows x 4 columns]
```

- Max rows

```
import pandas as pd
print(pd.options.display.max_rows)
```

```
60
```

```
import pandas as pd
pd.options.display.max_rows = 6
df = pd.read_csv('data.csv')
print(df)
```

```
     Duration  Pulse  Maxpulse  Calories
0          60    110       130     409.1
1          60    117       145     479.0
2          60    103       135     340.0
..        ...    ...       ...       ...
166        60    115       145     310.2
167        75    120       150     320.4
168        75    125       150     330.4

[169 rows x 4 columns]
```

- Read JSON

```
import pandas as pd
df = pd.read_json('data.json')
print(df.to_string())
```

```
    Duration  Pulse  Maxpulse  Calories
0         60    110       130     409.1
1         60    117       145     479.0
2         60    103       135     340.0
3         45    109       175     282.4
4         45    117       148     406.0
5         60    102       127     300.5
6         60    110       136     374.0
7         45    104       134     253.3
8         30    109       133     195.1
9         60     98       124     269.0
10        60    103       147     329.3
```

- Dictionary as JSON

```
import pandas as pd
data = {
  "Duration":{
    "0":60,
    "1":60,
    "2":60,
    "3":45,
    "4":45,
    "5":60
  },
  "Pulse":{
    "0":110,
    "1":117,
    "2":103,
    "3":109,
```

```
    "4":117,
    "5":102
  },
  "Maxpulse":{
    "0":130,
    "1":145,
    "2":135,
    "3":175,
    "4":148,
    "5":127
  },
  "Calories":{
    "0":409,
    "1":479,
    "2":340,
    "3":282,
    "4":406,
    "5":300
  }
}
df = pd.DataFrame(data)
print(df)
```

```
   Duration  Pulse  Maxpulse  Calories
0        60    110       130     409.1
1        60    117       145     479.0
2        60    103       135     340.0
3        45    109       175     282.4
4        45    117       148     406.0
5        60    102       127     300.5
```

- Viewing data – first 5 rows, last 5 rows

```
import pandas as pd
df = pd.read_csv('data.csv')
print(df.head())
print(df.tail())
print(df.info())
```

```
   Duration  Pulse  Maxpulse  Calories
0        60    110       130     409.1
1        60    117       145     479.0
2        60    103       135     340.0
3        45    109       175     282.4
4        45    117       148     406.0
```

```
     Duration  Pulse  Maxpulse  Calories
164        60    105       140     290.8
165        60    110       145     300.4
166        60    115       145     310.2
167        75    120       150     320.4
168        75    125       150     330.4
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 169 entries, 0 to 168
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Duration  169 non-null    int64
 1   Pulse     169 non-null    int64
 2   Maxpulse  169 non-null    int64
 3   Calories  164 non-null    float64
dtypes: float64(1), int64(3)
memory usage: 5.4 KB
None
```

- Data cleaning
  - Data cleaning means fixing bad data in your data set. Bad data could be:
    - Empty cells
    - Data in wrong format
    - Wrong data
    - Duplicates

Cleaning empty cells (has NaN values) - Remove rows with NaN values:

```
import pandas as pd
df = pd.read_csv('data.csv')
new_df = df.dropna()
print(new_df.to_string())
```

Data of wrong format converted into correct format:

```
19    60  '2020/12/19'   103   123   323.0
20    45  '2020/12/20'    97   125   243.0
21    60  '2020/12/21'   108   131   364.2
22    45           NaN   100   119   282.0
23    60  '2020/12/23'   130   101   300.0
24    45  '2020/12/24'   105   132   246.0
25    60  '2020/12/25'   102   126   334.5
26    60      20201226   100   120   250.0
27    60  '2020/12/27'    92   118   241.0
```

```
import pandas as pd
df = pd.read_csv('data.csv')
df['Date'] = pd.to_datetime(df['Date'])
print(df.to_string())
```

19	60	'2020/12/19'	103	123	323.0
20	45	'2020/12/20'	97	125	243.0
21	60	'2020/12/21'	108	131	364.2
22	45	NaT	100	119	282.0
23	60	'2020/12/23'	130	101	300.0
24	45	'2020/12/24'	105	132	246.0
25	60	'2020/12/25'	102	126	334.5
26	60	'2020/12/26'	100	120	250.0
27	60	'2020/12/27'	92	118	241.0

```
df.dropna(subset=['Date'], inplace = True)
```

19	60	2020-12-19	103	123	323.0
20	45	2020-12-20	97	125	243.0
21	60	2020-12-21	108	131	364.2
23	60	2020-12-23	130	101	300.0
24	45	2020-12-24	105	132	246.0
25	60	2020-12-25	102	126	334.5
26	60	2020-12-26	100	120	250.0
27	60	2020-12-27	92	118	241.0

- Fixing wrong data

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

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:

```
import pandas as pd
df = pd.read_csv('data.csv')
df.loc[7, 'Duration'] = 45
print(df.to_string())
```

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

- Discover duplicates

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
13	60	'2020/12/13'	106	128	345.3

```
print(df.duplicated())
```

```
9       False
10      False
11      False
12       True
13      False
```

- Remove duplicates

```
df.drop_duplicates(inplace = True)
```

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

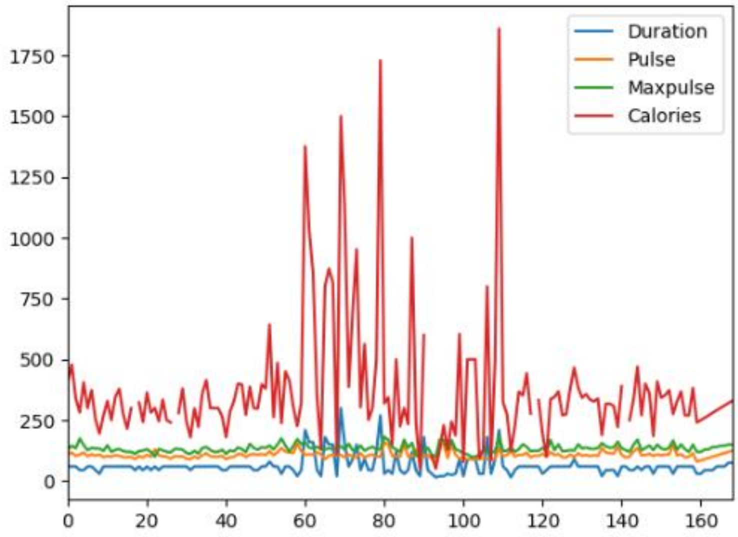
- Finding relationships

```
df.corr()
```

	Duration	Pulse	Maxpulse	Calories
Duration	1.000000	-0.155408	0.009403	0.922721
Pulse	-0.155408	1.000000	0.786535	0.025120
Maxpulse	0.009403	0.786535	1.000000	0.203814
Calories	0.922721	0.025120	0.203814	1.000000

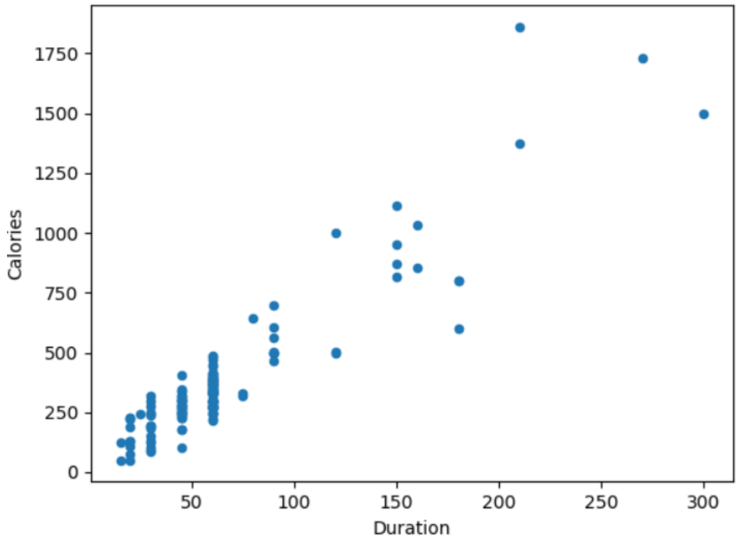
- Plotting

```
import pandas as pd
import matplotlib.pyplot as plt
df = pd.read_csv('data.csv')
df.plot()
plt.show()
```



- Scatter plot

```
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()
```



c) Scipy

Working with graphs

Graphs are an essential data structure.

SciPy provides us with the module scipy.sparse.csgraph for working with such data structures.

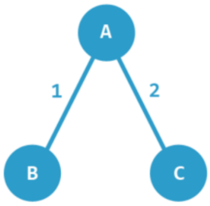
SciPy is built on NumPy and provides additional functionality for scientific computing.

It includes modules for optimization, integration, interpolation, eigenvalue problems, and more.Adjacency matrix

- Key Functions/Methods:

- Integration: scipy.integrate.quad(), scipy.integrate.simps().
- Interpolation: scipy.interpolate.interp1d().
- Optimization: scipy.optimize.minimize().
- Signal Processing: scipy.signal.convolve(), scipy.signal.spectrogram().
- Special functions: scipy.special.erf(), scipy.special.gamma().

```
   A B C
A:[0 1 2]
B:[1 0 0]
C:[2 0 0]
```



- Connected components

```
import numpy as np
from scipy.sparse.csgraph import connected_components
from scipy.sparse import csr_matrix
arr = np.array([
  [0, 1, 2],
  [1, 0, 0],
  [2, 0, 0]
])
newarr = csr_matrix(arr)
print(connected_components(newarr))
```

```
(1, array([0, 0, 0], dtype=int32))
```

- Dijksra

```
import numpy as np
from scipy.sparse.csgraph import dijkstra
from scipy.sparse import csr_matrix
arr = np.array([
  [0, 1, 2],
  [1, 0, 0],
  [2, 0, 0]
])
newarr = csr_matrix(arr)
print(dijkstra(newarr, return_predecessors=True,
indices=0))
```

```
(array([ 0.,  1.,  2.]), array([-9999,     0,     0], dtype=int32))
```

- Floyd warshall

```
import numpy as np
from scipy.sparse.csgraph import floyd_warshall
from scipy.sparse import csr_matrix
arr = np.array([
  [0, 1, 2],
  [1, 0, 0],
  [2, 0, 0]
])
newarr = csr_matrix(arr)
```

```
print(floyd_warshall(newarr,
return_predecessors=True))
```

```
(array([[ 0.,  1.,  2.],
       [ 1.,  0.,  3.],
       [ 2.,  3.,  0.]]), array([[-9999,     0,     0],
       [    1, -9999,     0],
       [    2,     0, -9999]], dtype=int32))
```

- Bellman Ford

```
import numpy as np
from scipy.sparse.csgraph import bellman_ford
from scipy.sparse import csr_matrix
arr = np.array([
  [0, -1, 2],
  [1, 0, 0],
  [2, 0, 0]
])
newarr = csr_matrix(arr)
print(bellman_ford(newarr, return_predecessors=True,
indices=0)
```

```
(array([ 0., -1.,  2.]), array([-9999,     0,     0], dtype=int32))
```

- Depth first order

```
import numpy as np
from scipy.sparse.csgraph import depth_first_order
from scipy.sparse import csr_matrix
arr = np.array([
  [0, 1, 0, 1],
  [1, 1, 1, 1],
  [2, 1, 1, 0],
  [0, 1, 0, 1]
])
newarr = csr_matrix(arr)
print(depth_first_order(newarr, 1))
```

```
(array([1, 0, 3, 2], dtype=int32), array([    1, -9999,     1,     0], dtype=int32))
```

- Breadth first order

```
import numpy as np
from scipy.sparse.csgraph import breadth_first_order
from scipy.sparse import csr_matrix
arr = np.array([
  [0, 1, 0, 1],
  [1, 1, 1, 1],
  [2, 1, 1, 0],
  [0, 1, 0, 1]
])
newarr = csr_matrix(arr)
print(breadth_first_order(newarr, 1))
```

```
(array([1, 0, 2, 3], dtype=int32), array([    1, -9999,     1,     1], dtype=int32))
```

- Binary Entropy Function

```
def binary_entropy(x):
    return -(sc.xlogy(x, x) + sc.xlog1py(1 - x, -
x))/np.log(2)
```

- A rectangular step function on [0,1]

```
def step(x):
    return 0.5*(np.sign(x) + np.sign(1 - x))
```

- Ramp function

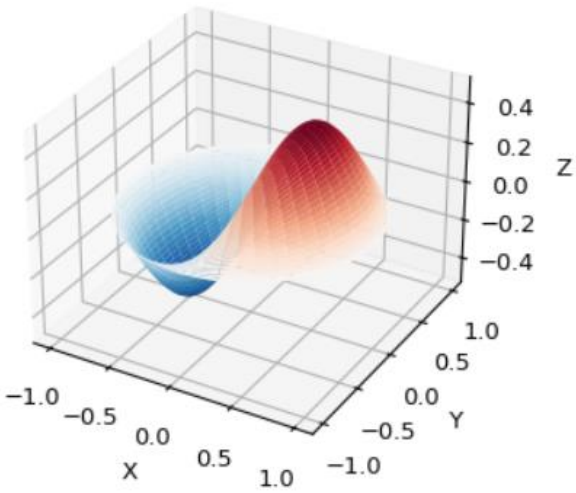
```
def ramp(x):
    return np.maximum(0, x)
```

- Bessel functions of real order (jv, jn_zeros)

Bessel functions are a family of solutions to Bessel's differential equation with real or complex order alpha:

$$x^2 \frac{d^2 y}{dx^2} + x \frac{dy}{dx} + (x^2 - \alpha^2)y = 0$$

```
>>> from scipy import special
>>> import numpy as np
>>> def drumhead_height(n, k, distance, angle, t):
...     kth_zero = special.jn_zeros(n, k)[-1]
...     return np.cos(t) * np.cos(n*angle) *
special.jn(n, distance*kth_zero)
>>> theta = np.r_[0:2*np.pi:50j]
>>> radius = np.r_[0:1:50j]
>>> x = np.array([r * np.cos(theta) for r in
radius])
>>> y = np.array([r * np.sin(theta) for r in
radius])
>>> z = np.array([drumhead_height(1, 1, r, theta,
0.5) for r in radius])
>>> import matplotlib.pyplot as plt
>>> fig = plt.figure()
>>> ax = fig.add_axes(rect=(0, 0.05, 0.95, 0.95),
projection='3d')
>>> ax.plot_surface(x, y, z, rstride=1, cstride=1,
cmap='RdBu_r', vmin=-0.5, vmax=0.5)
>>> ax.set_xlabel('X')
>>> ax.set_ylabel('Y')
>>> ax.set_xticks(np.arange(-1, 1.1, 0.5))
>>> ax.set_yticks(np.arange(-1, 1.1, 0.5))
>>> ax.set_zlabel('Z')
>>> plt.show()
```



- Cython Bindings for Special Functions (scipy.special.cython_special)

```
ciimport scipy.special.cython_special as csc
cdef:
    double x = 1
    double complex z = 1 + 1j
    double si, ci, rgam
    double complex cgam
rgam = csc.gamma(x)
print(rgam)
cgam = csc.gamma(z)
print(cgam)
csc.sici(x, &si, &ci)
print(si, ci)
```

- Nelder-Mead Simplex algorithm(method= 'Nelder-Mead')

```
>>> import numpy as np
>>> from scipy.optimize import minimize
```

```
>>> def rosen(x):
...     """The Rosenbrock function"""
...     return sum(100.0*(x[1:]-x[:-1]**2.0)**2.0 + (1-x[:-1])**2.0)
```

```
>>> x0 = np.array([1.3, 0.7, 0.8, 1.9, 1.2])
>>> res = minimize(rosen, x0, method='nelder-mead',
...                options={'xatol': 1e-8, 'disp': True})
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 339
         Function evaluations: 571
```

```
>>> print(res.x)
[1. 1. 1. 1. 1.]
```

d) Scikit-Learn

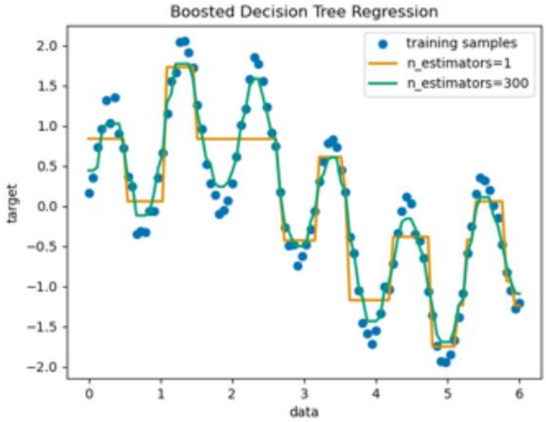
Identify which category an object belongs to.

Applications: spam detection, image recognition

Algorithms: Gradient boosting, nearest neighbors, random forest, logistic regression, classification, regression, clustering, and dimensionality reduction.

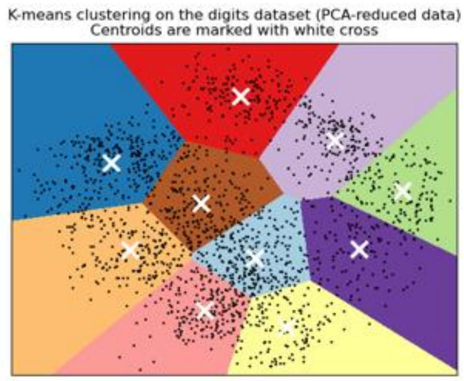
- Regression

  Applications: drug response, stock prices.
  Algorithms: Gradient boosting, nearest neighbors, random forest, ridge, and more...



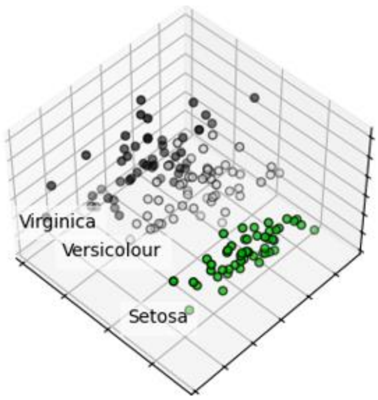
- Clustering

  Applications : Customer segmentation, Grouping experiment outcomes
  Algorithms: k-Means, HDBSCAN, hierarchical clustering, and more...



- Dimensionality Reduction

  Reducing the number of random variables to consider.
  Applications: Visualization, Increased efficiency
  Algorithms: PCA, feature selection, non-negative matrix factorization, and ...

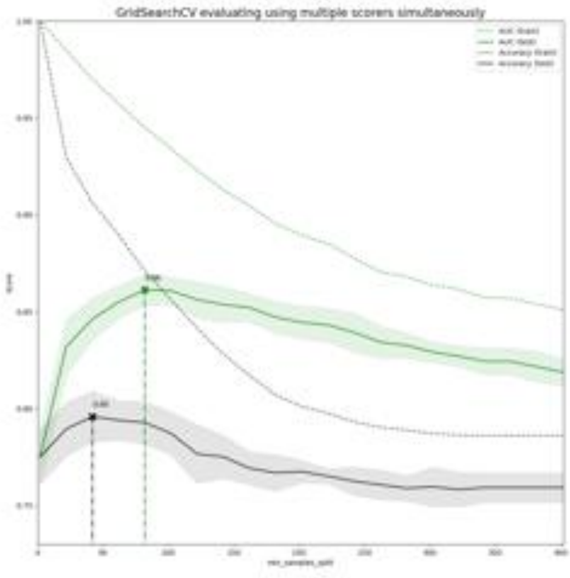


- Model selection

  Comparing, validating and choosing parameters and models.

  Applications: Improved accuracy via parameter tuning.

  Algorithms: grid search, cross validation, metrics, and more...



- Preprocessing

  Feature extraction and normalization.

  Applications: Transforming input data such as text for use with machine learning algorithms.

  Algorithms: preprocessing, feature extraction, and more...

- Key Functions/Methods:
  - sklearn.model_selection.train_test_split(): Split data into training and testing sets.
  - Model building and training: fit(), predict().
  - Evaluation metrics: sklearn.metrics.accuracy_score(), sklearn.metrics.confusion_matrix().
  - Preprocessing: sklearn.preprocessing.StandardScaler(), sklearn.preprocessing.LabelEncoder().
- Fitting and predicting: estimator basics
  - Scikit-learn provides dozens of built-in machine learning algorithms and models, called estimators.
  - Each estimator can be fitted to some data using its fit method.
    - RandonForestClassifier

```
from sklearn.ensemble import
RandomForestClassifier
clf = RandomForestClassifier(random_state=0)
X = [[ 1,  2,  3],  # 2 samples, 3 features
```

```
...       [11, 12, 13]]
y = [0, 1]  # classes of each sample
clf.fit(X, y)
RandomForestClassifier(random_state=0)
```

Once the estimator is fitted, it can be used for predicting target values of new data. You don't need to re-train the estimator:

```
clf.predict(X)  # predict classes of the
training data
array([0, 1])
clf.predict([[4, 5, 6], [14, 15, 16]])  #
predict classes of new data
array([0, 1])
```

- Transformers and pre-processors

  Pre-processors and transformers follow the same API as the estimator objects (they actually all inherit from the same BaseEstimator class). The transformer objects don't have a predict method but rather a transform method that outputs a newly transformed sample matrix X:

```
>>> from sklearn.preprocessing import
StandardScaler
>>> X = [[0, 15],
...      [1, -10]]
>>> # scale data according to computed scaling
values
>>> StandardScaler().fit(X).transform(X)
array([[-1.,  1.],
       [ 1., -1.]])
```

- Pipelines: chaining pre-processors and estimators

  Transformers and estimators (predictors) can be combined together into a single unifying object: a Pipeline.

```
>>> from sklearn.preprocessing import StandardScaler
>>> from sklearn.linear_model import LogisticRegression
>>> from sklearn.pipeline import make_pipeline
>>> from sklearn.datasets import load_iris
>>> from sklearn.model_selection import train_test_split
>>> from sklearn.metrics import accuracy_score
...
>>> # create a pipeline object
>>> pipe = make_pipeline(
...     StandardScaler(),
...     LogisticRegression()
... )
...
>>> # load the iris dataset and split it into train and
test sets
>>> X, y = load_iris(return_X_y=True)
>>> X_train, X_test, y_train, y_test = train_test_split(X,
y, random_state=0)
...
```

```
>>> # fit the whole pipeline
>>> pipe.fit(X_train, y_train)
Pipeline(steps=[('standardscaler', StandardScaler()),
                ('logisticregression',
LogisticRegression())])
>>> # we can now use it like any other estimator
>>> accuracy_score(pipe.predict(X_test), y_test)
0.97...
```

- Model Evaluation

  Fitting a model to some data does not entail that it will predict well on unseen data. This needs to be directly evaluated. We have just seen the train_test_split helper that splits a dataset into train and test sets, but scikit-learn provides many other tools for model evaluation, in particular for cross-validation.

```
>>> from sklearn.datasets import make_regression
>>> from sklearn.linear_model import LinearRegression
>>> from sklearn.model_selection import cross_validate
...
>>> X, y = make_regression(n_samples=1000,
random_state=0)
>>> lr = LinearRegression()
...
>>> result = cross_validate(lr, X, y)  # defaults to
5-fold CV
>>> result['test_score']  # r_squared score is high
because dataset is easy
array([1., 1., 1., 1., 1.])
```

- Automatic parameter searches

  The generalization power of an estimator often critically depends on a few parameters. For example a RandomForestRegressor has a n_estimators parameter that determines the number of trees in the forest, and a max_depth parameter that determines the maximum depth of each tree.

```
>>> from sklearn.datasets import fetch_california_housing
>>> from sklearn.ensemble import RandomForestRegressor
>>> from sklearn.model_selection import RandomizedSearchCV
>>> from sklearn.model_selection import train_test_split
>>> from scipy.stats import randint
...
>>> X, y = fetch_california_housing(return_X_y=True)
>>> X_train, X_test, y_train, y_test = train_test_split(X, y,
random_state=0)
...
>>> # define the parameter space that will be searched over
>>> param_distributions = {'n_estimators': randint(1, 5),
...                        'max_depth': randint(5, 10)}
...
```

```
>>> # now create a searchCV object and fit it to the data
>>> search =
RandomizedSearchCV(estimator=RandomForestRegressor(random_stat
e=0),
...                                n_iter=5,
...
param_distributions=param_distributions,
...                                random_state=0)
>>> search.fit(X_train, y_train)
RandomizedSearchCV(estimator=RandomForestRegressor(random_stat
e=0), n_iter=5,
                   param_distributions={'max_depth': ...,
                                        'n_estimators': ...},
                   random_state=0)
>>> search.best_params_
{'max_depth': 9, 'n_estimators': 4}

>>> # the search object now acts like a normal random forest
estimator
>>> # with max_depth=9 and n_estimators=4
>>> search.score(X_test, y_test)
0.73...
```

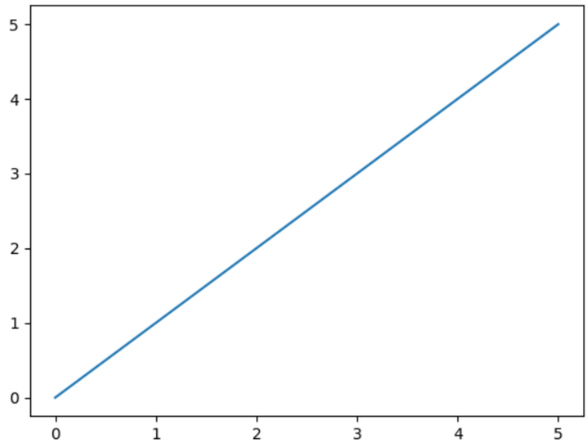
e) Matplotlib

Matplotlib is a 2D plotting library for creating static, animated, and interactive visualizations in Python.

- Key Functions/Methods:
  - matplotlib.pyplot.plot(), matplotlib.pyplot.scatter(): Create line plots and scatter plots.
  - matplotlib.pyplot.xlabel(), matplotlib.pyplot.ylabel(): Set axis labels.
  - matplotlib.pyplot.legend(): Add legends to plots.
  - matplotlib.pyplot.subplot(), matplotlib.pyplot.figure(): Create subplots and figures.
  - matplotlib.pyplot.savefig(): Save figures to a file.

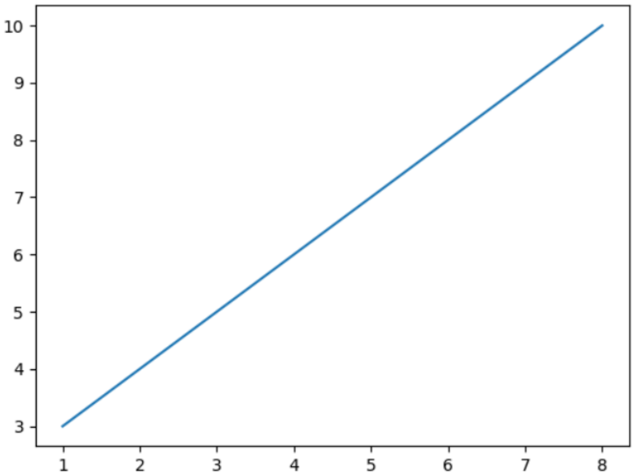
```
import matplotlib.pyplot as plt
import numpy as np

xpoints = np.array([0, 6])
ypoints = np.array([0, 250])

plt.plot(xpoints, ypoints)
```

```
plt.show()
```



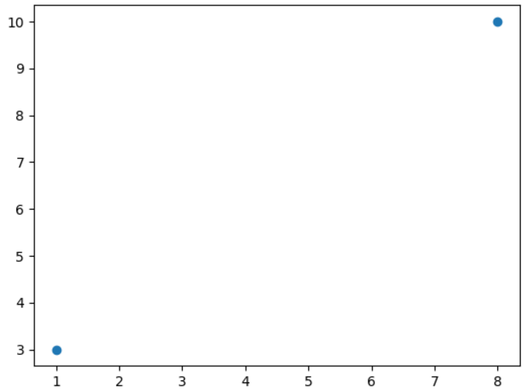
- Plotting x and y points

```
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()
```



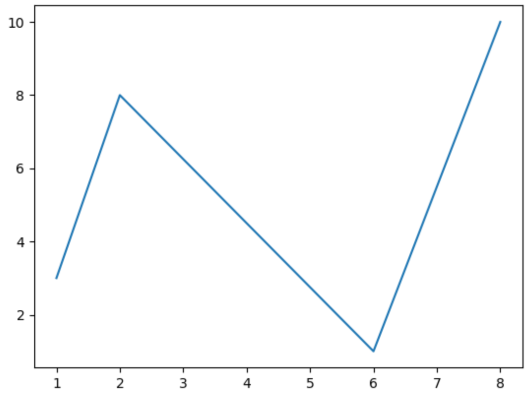
- Plotting Without Line

```
import matplotlib.pyplot as plt
import numpy as np
xpoints = np.array([1, 8])
ypoints = np.array([3, 10])
plt.plot(xpoints, ypoints, 'o')
plt.show()
```



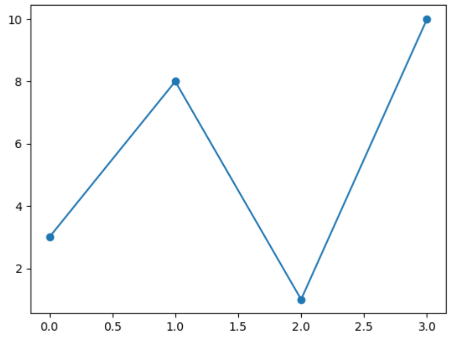
- Multiple Points

```
import matplotlib.pyplot as plt
import numpy as np
xpoints = np.array([1, 2, 6, 8])
ypoints = np.array([3, 8, 1, 10])
plt.plot(xpoints, ypoints)
plt.show()
```

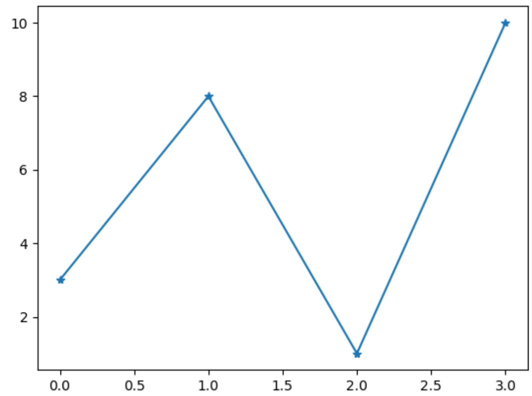


- Markers

```
import matplotlib.pyplot as plt
import numpy as np
ypoints = np.array([3, 8, 1, 10])
plt.plot(ypoints, marker = 'o')
plt.show()
```

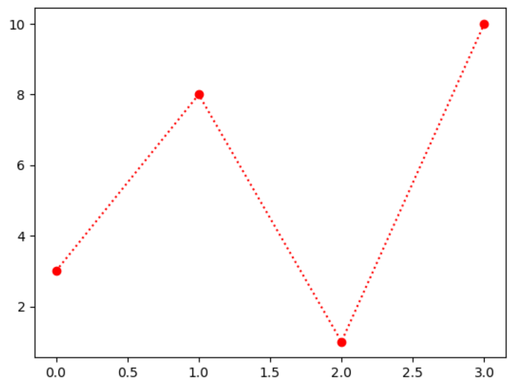


```
plt.plot(ypoints, marker = '*')
```



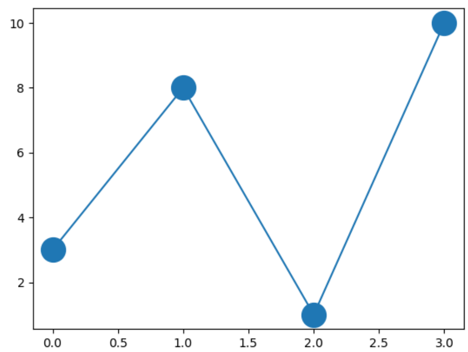
- Format Strings fmt

```
import matplotlib.pyplot as plt
import numpy as np
ypoints = np.array([3, 8, 1, 10])
plt.plot(ypoints, 'o:r')
plt.show()
```



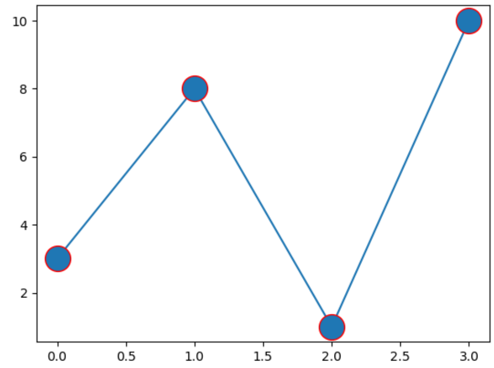
- Marker Size

```
import matplotlib.pyplot as plt
import numpy as np
ypoints = np.array([3, 8, 1, 10])
plt.plot(ypoints, marker = 'o', ms = 20)
plt.show()
```

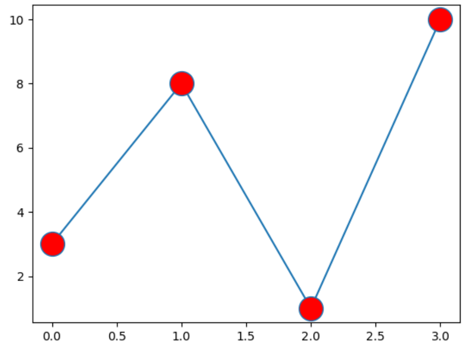


- Marker Color

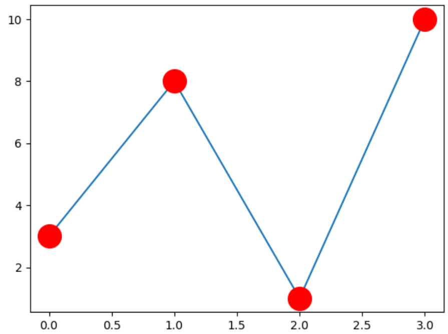
```
import matplotlib.pyplot as plt
import numpy as np
ypoints = np.array([3, 8, 1, 10])
plt.plot(ypoints, marker = 'o', ms = 20, mec = 'r')
plt.show()
```



```
import matplotlib.pyplot as plt
import numpy as np
ypoints = np.array([3, 8, 1, 10])
plt.plot(ypoints, marker = 'o', ms = 20, mfc = 'r')
plt.show()
```

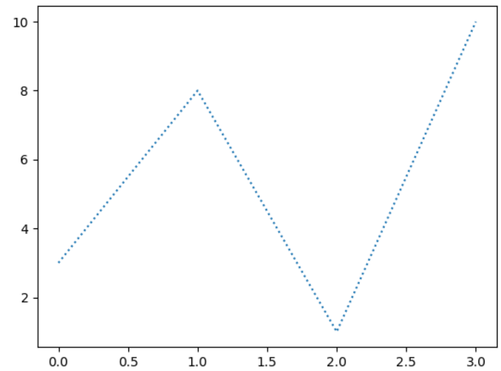


```
import matplotlib.pyplot as plt
import numpy as np
ypoints = np.array([3, 8, 1, 10])
plt.plot(ypoints, marker = 'o', ms = 20, mec = 'r',
mfc = 'r')
plt.show()
```



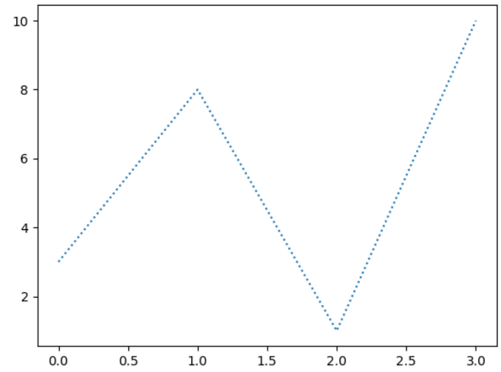
- Linestyle

```python
import matplotlib.pyplot as plt
import numpy as np
ypoints = np.array([3, 8, 1, 10])
plt.plot(ypoints, linestyle = 'dotted') #”dashed”
plt.show()
```

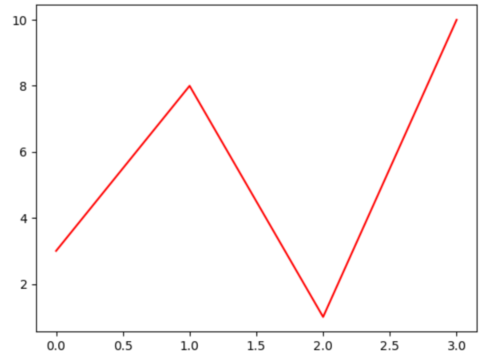


- Shorter syntax

```python
plt.plot(ypoints, ls = ':')
```



- Line color
- ```python
  import matplotlib.pyplot as plt
  import numpy as np
  ypoints = np.array([3, 8, 1, 10])
  plt.plot(ypoints, color = 'r') # c='#4CAF50',c='hotpink'
  plt.show()
  ```

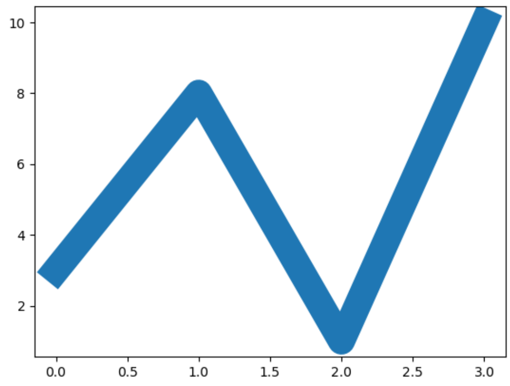

  ```

- Line width

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

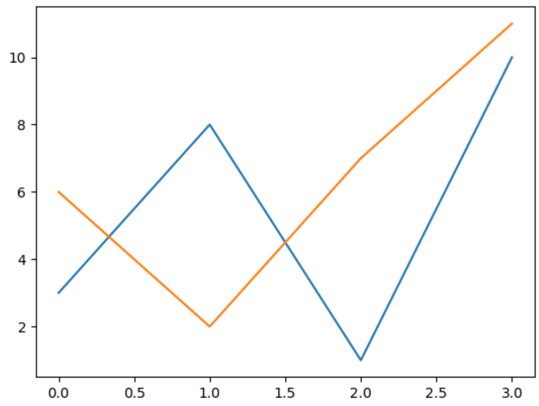
ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, linewidth = '20.5')
plt.show()
```



- Multiple Lines

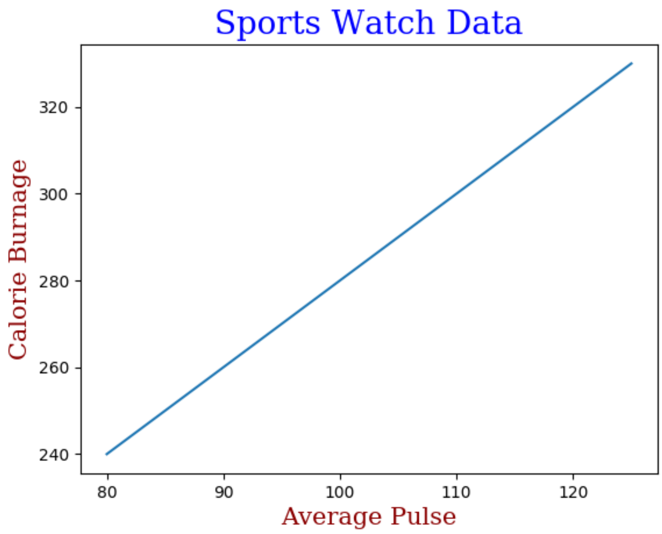
```
import matplotlib.pyplot as plt
import numpy as np
y1 = np.array([3, 8, 1, 10])
y2 = np.array([6, 2, 7, 11])
plt.plot(y1)
plt.plot(y2)
plt.show()
```



- Set font properties for title and labels

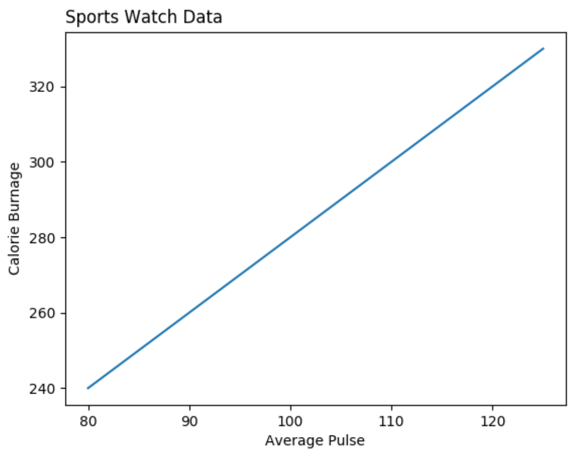
```
import numpy as np
import matplotlib.pyplot as plt
x =
np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 12
5])
y =
np.array([240, 250, 260, 270, 280, 290, 300, 310, 320
, 330])
```

```
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()
```



- Position of the title

```
import numpy as np
import matplotlib.pyplot as plt
x =
np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 12
5])
y =
np.array([240, 250, 260, 270, 280, 290, 300, 310, 320
, 330])
plt.title("Sports Watch Data", loc = 'left')
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")
plt.plot(x, y)
plt.show()
```



- Grid lines

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

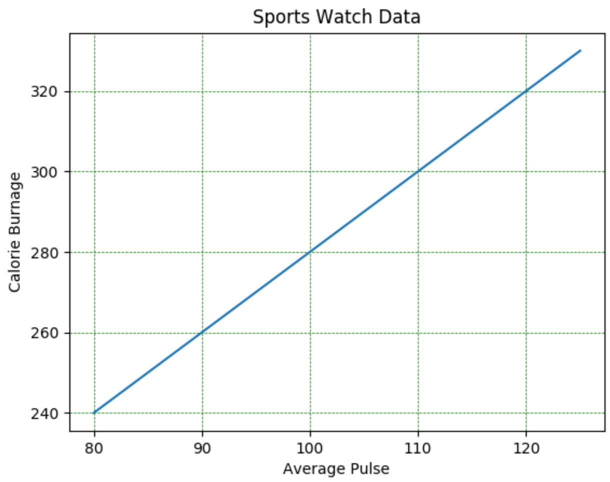
x =
np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 12
5])
y =
np.array([240, 250, 260, 270, 280, 290, 300, 310, 320
, 330])

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()
```



- Subplot for 6 charts

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

x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(2, 3, 1)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(2, 3, 2)
plt.plot(x,y)
```

```
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(2, 3, 3)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(2, 3, 4)
plt.plot(x,y)

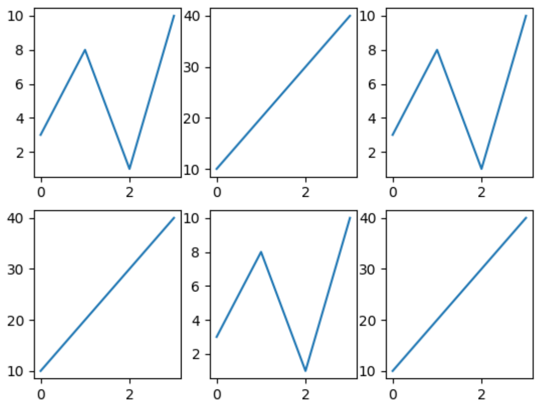
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(2, 3, 5)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(2, 3, 6)
plt.plot(x,y)

plt.show()
```



- Subtitle

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

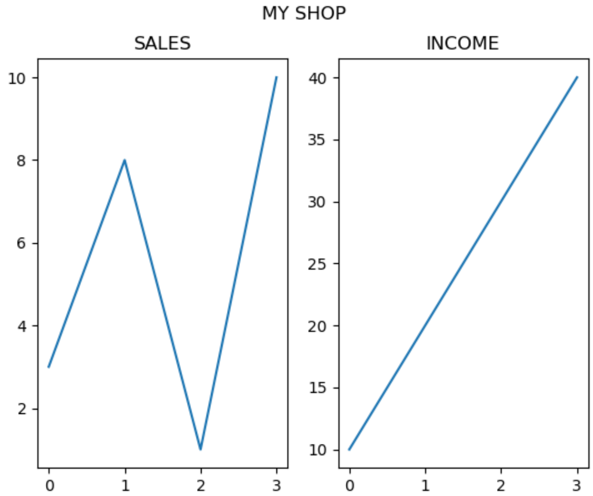
#plot 1:
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

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()
```

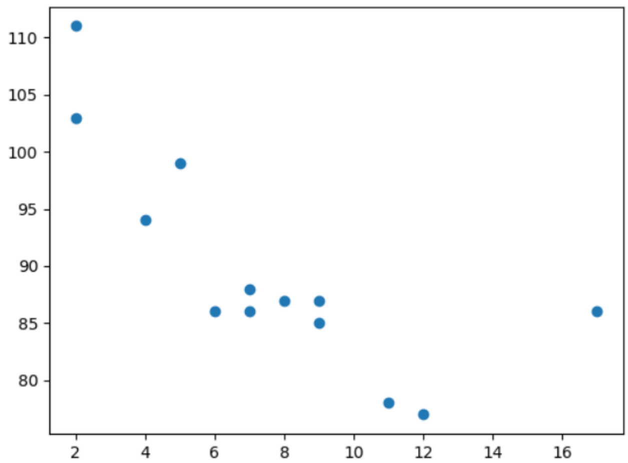


- Scatter Plots()

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

x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y =
np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])

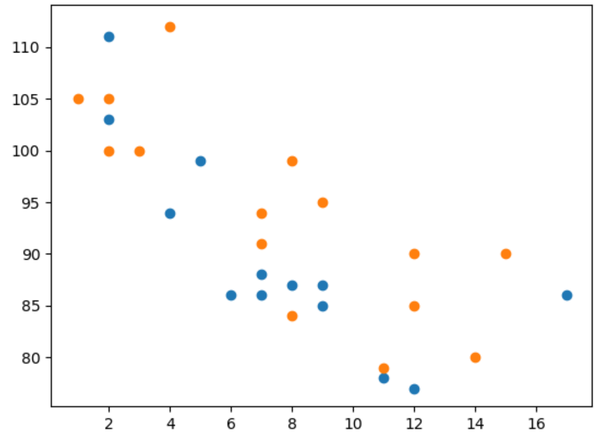
plt.scatter(x, y)
plt.show()
```



- Compare plots

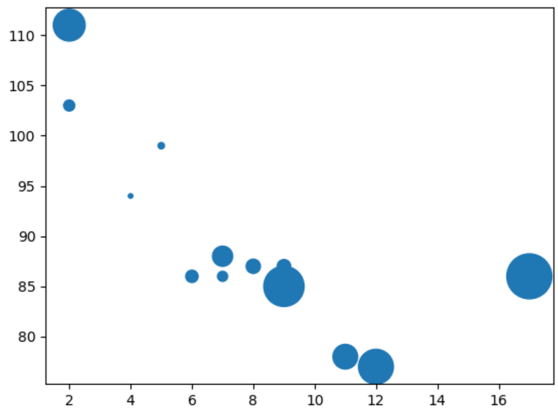
```
import matplotlib.pyplot as plt
import numpy as np
```

```
#day one, the age and speed of 13 cars:
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y =
np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
plt.scatter(x, y)
#day two, the age and speed of 15 cars:
x = np.array([2,2,8,1,15,8,12,9,7,3,11,4,7,14,12])
y =
np.array([100,105,84,105,90,99,90,95,94,100,79,112,91
,80,85])
plt.scatter(x, y)
plt.show()
```



- Varied plot size

```
import matplotlib.pyplot as plt
import numpy as np
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y =
np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
sizes
= np.array([20,50,100,200,500,1000,60,90,10,300,600,8
00,75])
plt.scatter(x, y, s=sizes)
plt.show()
```

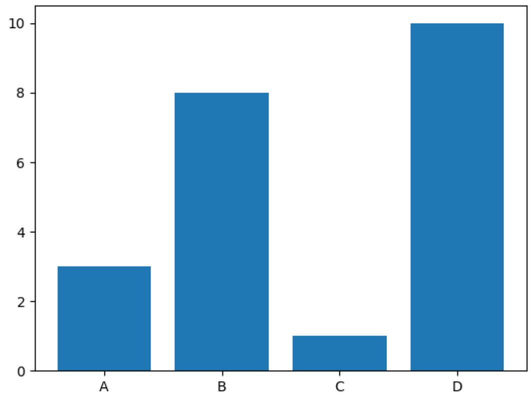


- Bars – For x-axis: bar(x,y), For y-axis: barh(x,y)

```
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)
plt.show()
```

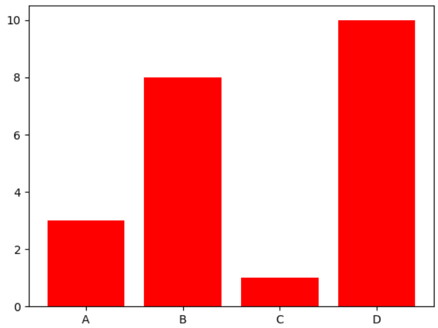


- Bar color

```
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, color = "red")
plt.show()
```

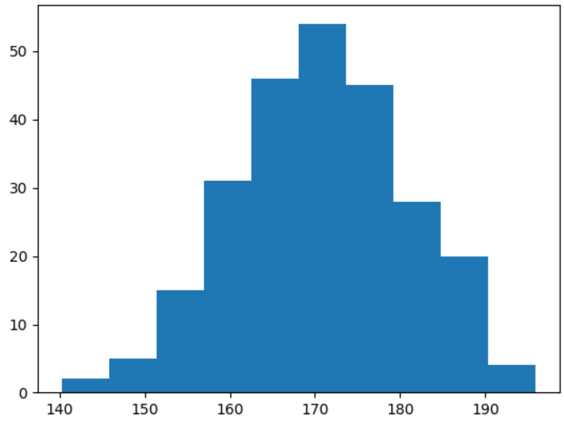


- Histogram

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

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

plt.hist(x)
plt.show()
```

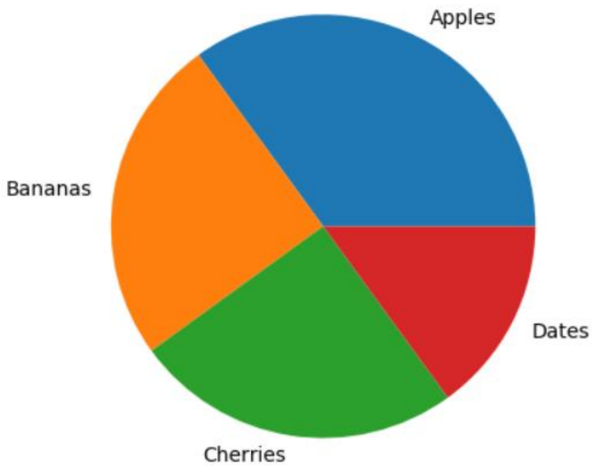


- Pie Chart

```
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.show()
```



List down the features and class labels from the dataset. Explore the steps involved in the Learning process.

1) **Loading the dataset.**
2) **Pre-Processing the data (Handling missing values, Normalization, Standardization).**
3) **Exploratory Data Analysis.**
4) **Feature Selection Techniques.**
5) **Split the data into training, testing and validation sets.**

CODE and Output:

```
import numpy as np
import pandas as pd
import scipy as sl
import matplotlib.pyplot as mat

Loading The Dataset

df = pd.read_csv("C:\\Users\\SSN\\Desktop\\ML
Lab\\Ex1\\diabetes_dataset.csv",)
```

```
print("The Shape of the Data Frame (rows , columns)  :  ",df.shape)
print()
print()
print("The DataType for Each Attributes in Data Frame\n",df.dtypes)
```

```
The Shape of the Data Frame (rows , columns)  :   (100000, 9)


The DataType for Each Attributes in Data Frame
 gender                     object
age                       float64
hypertension              float64
heart_disease             float64
smoking_history            object
bmi                       float64
HbA1c_level               float64
blood_glucose_level       float64
diabetes                  float64
dtype: object
```

```
print(df.head)
```

```
... <bound method NDFrame.head of        gender   age  hypertension  heart_disease smoking_history    bmi  \
    0      Female  80.0           0.0            1.0           never  25.19
    1      Female  54.0           0.0            0.0         No Info  27.32
    2        Male  28.0           0.0            0.0           never  27.32
    3      Female  36.0           0.0            0.0         current  23.45
    4        Male  76.0           1.0            1.0         current  20.14
    ...       ...   ...           ...            ...             ...    ...
    99995  Female  80.0           0.0            0.0         No Info  27.32
    99996  Female   2.0           0.0            0.0         No Info  17.37
    99997    Male  66.0           0.0            0.0          former  27.83
    99998  Female  24.0           0.0            0.0           never  35.42
    99999  Female  57.0           0.0            0.0         current  22.43

           HbA1c_level  blood_glucose_level  diabetes
    0              6.6                140.0       0.0
    1              6.6                 80.0       0.0
    2              5.7                158.0       0.0
    3              5.0                155.0       0.0
    4              4.8                155.0       0.0
    ...            ...                  ...       ...
    99995          6.2                 90.0       0.0
    99996          6.5                100.0       0.0
    99997          5.7                155.0       0.0
    99998          4.0                100.0       0.0
    99999          6.6                 90.0       0.0

    [100000 rows x 9 columns]>
```

```
Finding The Missing Values in the Dataframe
```

```
print("The Number of Missing Values in Each Columns\n",df.isnull().sum())
```

```
... The Number of Missing Values in Each Columns
     gender                 1
    age                    2
    hypertension           2
    heart_disease          1
    smoking_history       16
    bmi                   14
    HbA1c_level            7
    blood_glucose_level   16
    diabetes               4
    dtype: int64
```

```
Handling The Missing Values for The Attributes as follow
Gender              Fill as Unknown
Age                 Fill a Median Value
Hypertension        Fill a Null Value 0
Heart_diease        Fill a Null value 0
Smoking History     Fill a No Info
```

```
BMI                 Fill a Median Value
HbA1c_level         Fill a Median Value
blood_glucose_level Fill a Median Value
diabetes            Remopve the Row
```

```
print("Missing values handling for 'gender'")
df['gender']=df['gender'].fillna('Unknown')
```

```
print("Missing values handling for Age")
age_median = df['age'].median()
df['age']=df['age'].fillna(age_median)
```

```
print("Missing values handling for Hyper Tension and Heart Disease")
df['hypertension']=df['hypertension'].fillna(0)
df['heart_disease']=df['heart_disease'].fillna(0)
```

```
print("Missing values handling for Smoking history")
df['smoking_history']=df['smoking_history'].fillna('No Info')
```

```
print("Missing values handling for bmi")
bmi_median = df['bmi'].median()
df['bmi']=df['bmi'].fillna(bmi_median)
```

```
print("Missing values handling for HbA1c_level")
hba1c_median = df['HbA1c_level'].median()
df['HbA1c_level']=df['HbA1c_level'].fillna(hba1c_median)
```

```
print("Missing values handling for HbA1c_level")
blood_glucose_median = df['blood_glucose_level'].median()
df['blood_glucose_level'] =
df['blood_glucose_level'].fillna(blood_glucose_median)
```

```
print("Missing values handling for diabetes_status")
df.dropna(subset=['diabetes'], inplace=True)
```

```
print("The Number of Missing Values in Each Columns\n",df.isnull().sum())
```

```
The Number of Missing Values in Each Columns
 gender                0
age                   0
hypertension          0
heart_disease         0
smoking_history       0
bmi                   0
HbA1c_level           0
blood_glucose_level   0
diabetes              0
dtype: int64
```

```
Normalization The attributes Age and BMI
from sklearn.preprocessing import MinMaxScaler, StandardScaler
attributes_to_normalize = ['age', 'bmi']
min_max_scaler = MinMaxScaler()
df[attributes_to_normalize] =
min_max_scaler.fit_transform(df[attributes_to_normalize])
```

```
       gender       age  hypertension  heart_disease smoking_history  \
0      Female  1.000000           0.0            1.0           never
1      Female  0.674675           0.0            0.0         No Info
2        Male  0.349349           0.0            0.0           never
3      Female  0.449449           0.0            0.0         current
4        Male  0.949950           1.0            1.0         current
...       ...       ...           ...            ...             ...
99995  Female  1.000000           0.0            0.0         No Info
99996  Female  0.024024           0.0            0.0         No Info
99997    Male  0.824825           0.0            0.0          former
99998  Female  0.299299           0.0            0.0           never
99999  Female  0.712212           0.0            0.0         current

            bmi  HbA1c_level  blood_glucose_level  diabetes
0      0.263246          6.6                140.0       0.0
1      0.285505          6.6                 80.0       0.0
2      0.285505          5.7                158.0       0.0
3      0.245062          5.0                155.0       0.0
4      0.210471          4.8                155.0       0.0
...         ...          ...                  ...       ...
99995  0.285505          6.2                 90.0       0.0
99996  0.181524          6.5                100.0       0.0
99997  0.290835          5.7                155.0       0.0
99998  0.370154          4.0                100.0       0.0
99999  0.234403          6.6                 90.0       0.0

[99996 rows x 9 columns]
```

```
Standardization the Attributes HbA1c Level and Blod Glucose Level
```

```python
# List of attributes to be standardized
attributes_to_standardize = ['HbA1c_level', 'blood_glucose_level']

standard_scaler = StandardScaler()

df[attributes_to_standardize] =
standard_scaler.fit_transform(df[attributes_to_standardize])
```

```
       gender       age  hypertension  heart_disease smoking_history  \
0      Female  1.000000           0.0            1.0           never
1      Female  0.674675           0.0            0.0         No Info
2        Male  0.349349           0.0            0.0           never
3      Female  0.449449           0.0            0.0         current
4        Male  0.949950           1.0            1.0         current
...       ...       ...           ...            ...             ...
99995  Female  1.000000           0.0            0.0         No Info
99996  Female  0.024024           0.0            0.0         No Info
99997    Male  0.824825           0.0            0.0          former
99998  Female  0.299299           0.0            0.0           never
99999  Female  0.712212           0.0            0.0         current

            bmi  HbA1c_level  blood_glucose_level  diabetes
0      0.263246     1.001694             0.047797       0.0
1      0.285505     1.001694            -1.426171       0.0
2      0.285505     0.161103             0.489988       0.0
3      0.245062    -0.492690             0.416289       0.0
4      0.210471    -0.679488             0.416289       0.0
...         ...          ...                  ...       ...
99995  0.285505     0.628098            -1.180510       0.0
99996  0.181524     0.908295            -0.934848       0.0
99997  0.290835     0.161103             0.416289       0.0
99998  0.370154    -1.426680            -0.934848       0.0
99999  0.234403     1.001694            -1.180510       0.0

[99996 rows x 9 columns]
```

```
Extract The Features Using SelectFromModel imported from sklearn package
```

```python
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.ensemble import RandomForestClassifier
```

```
from sklearn.feature_selection import SelectFromModel

X = df.drop(columns=['diabetes'])
y = df['diabetes']

categorical_features = ['gender', 'smoking_history']

numerical_features = ['age', 'hypertension', 'bmi', 'HbA1c_level',
'blood_glucose_level', 'heart_disease']

categorical_preprocessor = OneHotEncoder(handle_unknown='ignore')

numerical_preprocessor = SimpleImputer(strategy='median')

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_preprocessor, numerical_features),
        ('cat', categorical_preprocessor, categorical_features)
    ])

rf_clf = RandomForestClassifier()
selector = SelectFromModel(estimator=rf_clf)

pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('selector', selector)
])

pipeline.fit(X, y)
selected_features_indices =
pipeline.named_steps['selector'].get_support(indices=True)

selected_features = df.columns[selected_features_indices]

print("The Selected Featurs are : ",selected_features)
```

```
The Selected Features are: Index(['gender', 'hypertension', 'heart_disease', 'smoking_history'], dtype='object')
```

```
Divide the Train and Test data using the train_test_split method imported from
sklearn
```

```
from sklearn.model_selection import train_test_split
```

```python
X_selected = df[selected_features]
y = df['diabetes']

X_train, X_test, y_train, y_test = train_test_split(X_selected, y,
test_size=0.2, random_state=42)

print("Shape of X_train:", X_train.shape)
print("Shape of X_test:", X_test.shape)
print("Shape of y_train:", y_train.shape)
print("Shape of y_test:", y_test.shape)
```

```
Shape of X_train: (79996, 4)
Shape of X_test: (20000, 4)
Shape of y_train: (79996,)
Shape of y_test: (20000,)
```

Learning Outcome:

1) Learning the python packages such as Numpy, Scipy, Scikitlearn, Matplotlib
2) Learning about Loading the dataset using the pandas
3) Learning about the Pre processing steps in ML
4) Learning about Feature Selection Techniques.
5) Learning about Split the data into training, testing and validation sets