Harshal Jaywant Chavan
Division: A
FYMCA
202121

LATE BHAUSAHEB HIRAY S.S. TRUST'S INSTITUTE OF COMPUTER APPLICATION

ISO 9001-2008 CERTIFIED

S.N. 341, Next to New English School, Govt. Colony, Bandra (East), Mumbai – 400051, Tel: 91-22-26570892/3181

Date:17/04/2021

CERTIFICATE

This is to certify that Mr./Ms.

________Harshal Jaywant Chavan____________________Roll No.202124

is a student of MCA of 1st year Semester-II has completed successfully full-semester practical/assignments of subject AI & ML for the academic year 2020 – 21.

Subject In-Charge **Director**

External Examiner

Harshal Jaywant Chavan
Division: A
FYMCA
202121

LBHSS's

Hiray Institute of Computer Application

AI &ML PRACTICAL – JOURNAL

F.Y.MCA

Harshal Jaywant Chavan
Division: A
FYMCA
202121

Index

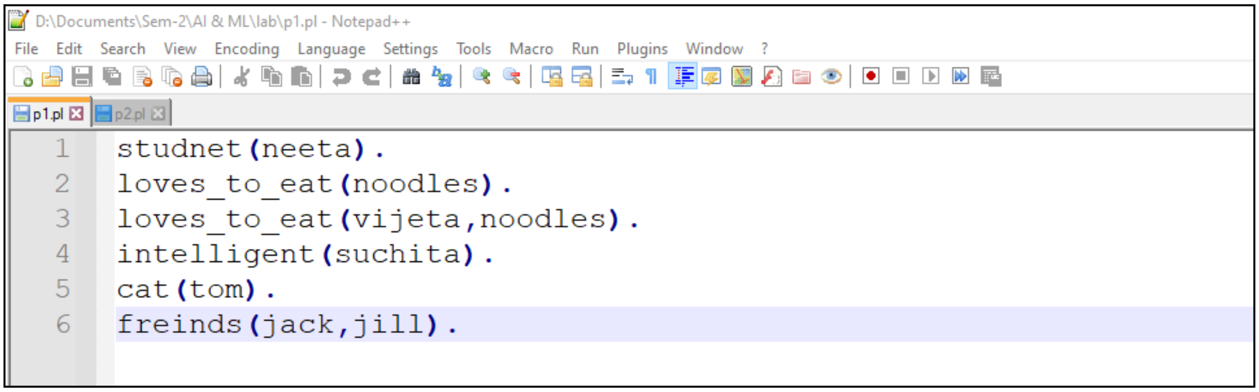
Sr. No.	Practicals	Date	Sign
1	Study of Logical Programming with Prolog.		
2	Study of Python Libraries: A) NumPy B) Pandas		
3	Study of Python Libraries: A) Matplotlib B) Scikit Learn		
4	Study of Supervised Learning: A) Linear Regression B) Logistic Regression C) K Nearest Neighbor Algorithm		
5	Study of Dimension Reduction: A) Feature Scaling B) Normalization (Feature Selection)		
6	Study of Principal of Component Analysis		
7	Study of K-mean Clustering.		
8	Study of Support Vector Machine.		

9	Study of Bagging Algorithms: A) Bagging algorithm B) Decision Tree Classifier C) Random Forest
10	Study of Boosting Algorithms: A) Ada Boost B) Stochastic Gradient Boosting C) Voting Ensemble
11	Study of Python Flask Library Any algorithm with Flask and Pickle Library

Harshal Jaywant Chavan
Division: A
FYMCA
202121

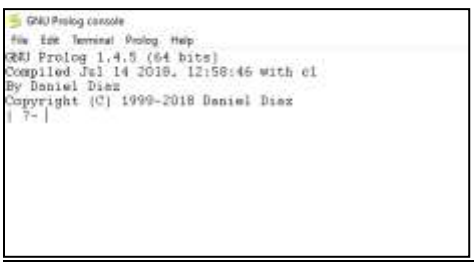
Practical 1: Study of Logical Programming with Prolog.

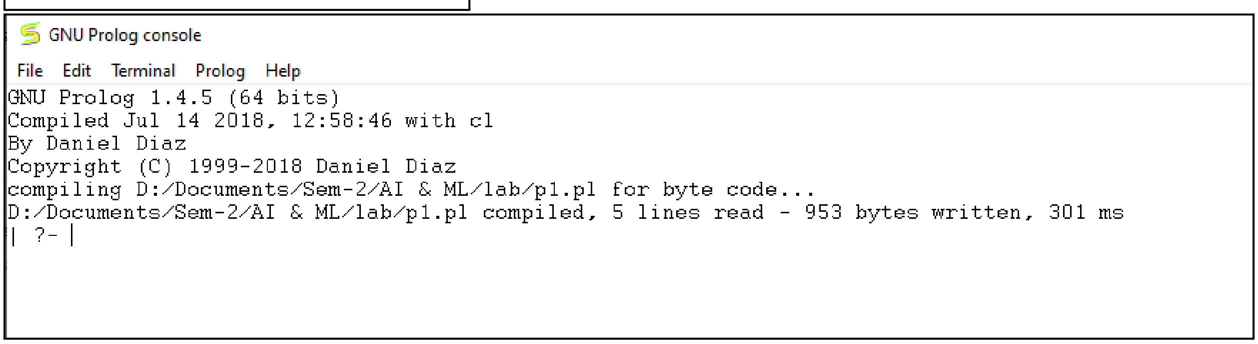
Step: - 1. Create one file in notepad with .pl extension and file type =All type.




```
studnet(neeta).
loves_to_eat(noodles).
loves_to_eat(vijeta,noodles).
intelligent(suchita).
cat(tom).
freinds(jack,jill).
```

Go to GNU → File → Change dir → Select the folder where notepad file is saved.






```
GNU Prolog 1.4.5 (64 bits)
Compiled Jul 14 2018, 12:58:46 with cl
By Daniel Diaz
Copyright (C) 1999-2018 Daniel Diaz
compiling D:/Documents/Sem-2/AI & ML/lab/p1.pl for byte code...
D:/Documents/Sem-2/AI & ML/lab/p1.pl compiled, 5 lines read - 953 bytes written, 301 ms
| ?- |
```

Code:-

→[p1].

→ Student(X).

Intelligent(Y).

```
GNU Prolog console
File   Edit   Terminal   Prolog   Help
GNU Prolog 1.4.5 (64 bits)
Compiled Jul 14 2018, 12:58:46 with cl
By Daniel Diaz
Copyright (C) 1999-2018 Daniel Diaz
| ?- change_directory('D:/AI & ML').

yes
| ?- [p1].
compiling D:/AI & ML/p1.pl for byte code...
D:/AI & ML/p1.pl:1: warning: singleton variables [Sameer] for student/1
D:/AI & ML/p1.pl:3: warning: singleton variables [Krishna] for loves_to_eat/2
D:/AI & ML/p1.pl:4: warning: singleton variables [Ritvik] for intelligent/1
D:/AI & ML/p1.pl compiled, 5 lines read - 826 bytes written, 390 ms

yes
| ?- student(X).

yes
| ?- intelligent(Y)
.

yes
| ?- |
```

2. AND (;) & OR (,) Function

Create new notepad file.

→ likes(pooja, geeta).
likes(geeta,pooja).
likes(neha,aliya).

freindship(X,Y) :- likes(X,Y);likes(Y,X).

```
| ?- [p2].
compiling D:/AI & ML/p2.pl for byte code...
D:/AI & ML/p2.pl compiled, 5 lines read - 884 bytes written, 17 ms

yes
| ?- freindship(X,Y).

X = pooja
Y = geeta ? ;

X = geeta
Y = pooja ? ;

X = neha
Y = aliya ? ;

X = geeta
Y = pooja ? ;

X = pooja
Y = geeta ? ;

X = aliya
Y = neha

(46 ms) yes
| ?- |
```

→ likes(pooja,geeta).
likes(geeta,pooja).
likes(neha,aliya).

freindship(X,Y) :- likes(X,Y),likes(Y,X).

```
| ?- [p2].
compiling D:/AI & ML/p2.pl for byte code...
D:/AI & ML/p2.pl compiled, 5 lines read - 763 bytes written, 16 ms

yes
| ?- freindship(X,Y).

X = pooja
Y = geeta ? ;

X = geeta
Y = pooja ? ;

no
| ?- |
```

→ next_to(mumbai,pune).
next_to(pune,satara).
next_to(mumbai,nashik).

travel(A,C) :- next_to(A,B),next_to(B,C).

```
| ?- [p2].
compiling D:/AI & ML/p2.pl for byte code...
D:/AI & ML/p2.pl compiled, 5 lines read - 738 bytes written, 16 ms

yes
| ?- freindship(X,Y).

X = pooja
Y = geeta ? ;

X = geeta
Y = pooja ? ;

(31 ms) no
```

3. **Relationship in prolog: - specify relationship between object and properties of objects.**
Relationship can also be a rule.

Create new notepad file.

→ p3.pl

```
female(scarlet).
female(alice).
female(katherine).
female(fiona).

male(bob).
male(sean).
male(chris).
male(dravis).

parent(bob, alice).
parent(bob, sean).
parent(scarlet,alice).
parent(scarlet, sean).

parent(alice, katherine).
parent(sean, chris).

parent(katherine,fiona).
parent(chris,dravis).

granparent(X,Y) :- parent(X,Z),parent(Z,Y).

sister(X,Y) :- parent(Z,X), parent(Z,Y), female(X), X\=Y.

brother(X,Y) :- parent(Z,X), parent(Z,Y), male(X), female(Y).

uncle(X,Y) :- parent(Z,Y), brother(X,Z).

aunt(X,Y) :- parent(Z,Y),sister(X,Z).

daughter(X,Y) :- parent(Y,X), female(X).

son(X,Y) :- parent(Y,X), male(X).

mother(X,Y) :- parent(X,Y), female(X).

father(X,Y) :- parent(X,Y), male(X).
```

GNU Prolog console

File Edit Terminal Prolog Help

```
GNU Prolog 1.4.5 (64 bits)
Compiled Jul 14 2018, 12:58:46 with cl
By Daniel Diaz
Copyright (C) 1999-2018 Daniel Diaz
| ?- change_directory('D:/AI & ML').

yes
| ?- [p3].
compiling D:/AI & ML/p3.pl for byte code...
D:/AI & ML/p3.pl compiled, 38 lines read - 4475 bytes written, 354 ms

(31 ms) yes
| ?- parent(X,Y)
.

X = bob
Y = alice ? ;

X = bob
Y = sean ? ;

X = scarlet
Y = alice ? ;

X = scarlet
Y = sean ? ;

X = alice
Y = katherine ? ;

X = sean
Y = chris ? ;

X = katherine
Y = fiona ? ;

X = chris
Y = dravis
```

```
| ?- brother(X,Y).

X = sean
Y = alice ? ;

X = sean
Y = alice ? ;

no
| ?- sister(X,Y).

X = alice
Y = sean ? ;

X = alice
Y = sean ? ;

(16 ms) no
| ?- grandparent(X,Y).

X = bob
Y = katherine ? ;

X = bob
Y = chris ? ;

X = scarlet
Y = katherine ? ;

X = scarlet
Y = chris ? ;

X = alice
Y = fiona ? ;

X = sean
Y = dravis ? ;
```

```
| ?- uncle(X,Y).

X = sean
Y = katherine ? ;

X = sean
Y = katherine ? ;

(15 ms) no
| ?- aunt(X,Y).

X = alice
Y = chris ? ;

X = alice
Y = chris ? ;

(31 ms) no
| ?- mother(X,Y).

X = scarlet
Y = alice ? ;

X = scarlet
Y = sean ? ;

X = alice
Y = katherine ? ;

X = katherine
Y = fiona ? ;

(47 ms) no
```

```
| ?- father(X,Y).

X = bob
Y = alice ? ;

X = bob
Y = sean ? ;

X = sean
Y = chris ? ;

X = chris
Y = dravis

yes
| ?- son(X,Y).

X = sean
Y = bob ? ;

X = sean
Y = scarlet ? ;

X = chris
Y = sean ? ;

X = dravis
Y = chris
```

```
| ?- daughter(X,Y).

X = alice
Y = bob ? ;

X = alice
Y = scarlet ? ;

X = katherine
Y = alice ? ;

X = fiona
Y = katherine ? ;

(31 ms) no
| ?- |
```

Practical 2: Study of Python Libraries:

a) NumPy b) Pandas

NumPy

```
import numpy as np

l = ['dog', 'cat', 'horse']

l
```

Output: ['dog', 'cat', 'horse']

```
type(l)
```

Output: list

```
l.sort()

l
```

Output: ['cat', 'dog', 'horse']

```
li = list(range(6))

li
```

Output: [0, 1, 2, 3, 4, 5]

```
while li:
    p=li.pop()
    print('p:', p)
    print('li:', li)
```

Output:

p: 5

li: [0, 1, 2, 3, 4]

p: 4

li: [0, 1, 2, 3]

p: 3

li: [0, 1, 2]

p: 2

li: [0, 1]

p: 1

li: [0]

p: 0

li: []

a = ('Ryan', 33, True)

b = 'Takaya', 25, False

type(b)

Output: tuple

type(a)

type(b)

Output: tuple

print(a[1])

Output: 33

print(b[0])

Output: Takaya

a = np.array([2,4,6,8])

a

Output: array([2, 4, 6, 8])

a.dtype

Output: dtype('int32')

a = np.array([2,4,6,8], np.int64)

a

Output: array([2, 4, 6, 8], dtype=int64)

a = np.array([[2,4,6,8]])

a

Output: array([[2, 4, 6, 8]])

a[0][3]

Output: 8

a.shape

Output: (1, 4)

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

listarr

Output:

```
array([[1, 1, 1],
       [2, 2, 2],
       [3, 3, 3]])
```

listarr.shape

Output: (3, 3)

listarr.size

Output: 9

z = np.zeros((2,4))

z

Output:

array([[0., 0., 0., 0.],

[0., 0., 0., 0.]])

z.shape

Output: (2, 4)

y = np.ones((3,4))

y

Output:

array([[1., 1., 1., 1.],

[1., 1., 1., 1.],

[1., 1., 1., 1.]])

y = np.ones((2,3,4))

y

Output:

array([[[1., 1., 1., 1.],

[1., 1., 1., 1.],

[1., 1., 1., 1.]],

```
       [[1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.]]])
```

x = np.arange(10)

x

Output: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

m = np.linspace(1,5,4)

m

Output: array([1.   , 2.33333333, 3.66666667, 5.   ])

m = np.linspace(1,7,3)

Output: Type Markdown and LaTeX: α2

m

Output: array([1., 4., 7.])

y

Output:

```
array([[[1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.]],

       [[1., 1., 1., 1.],
        [1., 1., 1., 1.],
```

```
       [1., 1., 1., 1.]]])
```

```
c = np.ones_like(y)
c
```

Output:

```
array([[[1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.]],

       [[1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.]]])
```

```
g = np.ones((2,3,4))
g
```

Output:

```
array([[[1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.]],

       [[1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.]]])
```

```
g.reshape
```

Output: <function ndarray.reshape>

g

Output:

```
array([[[1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.]],

       [[1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.]]])
```

```
h = np.arange(50)
h
```

Output:

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
       34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49])
```

```
h.reshape(2,25)
```

Output:

```
array([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15,
        16, 17, 18, 19, 20, 21, 22, 23, 24],
       [25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40,
        41, 42, 43, 44, 45, 46, 47, 48, 49]])
```

```
h.ravel()
```

Output:

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
```

17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,

34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49])

b = np.arange(3,10,2, dtype=np.int32)

b.itemsize

b

Output: 4

b = np.arange(3.4,10,2)

b.itemsize

Output: 8

b.shape

Output: (4,)

b.itemsize

Output: 8

t = np.linspace(3,10,3, dtype=np.int32)

t

Output: array([3. , 4.75, 6.5 , 8.25, 10.])

t = np.linspace(3,10,5, dtype=np.int32)

t

Output: array([3, 4, 6, 8, 10])

m = np.arange(6)

```
m
```

Output: array([0, 1, 2, 3, 4, 5])

```
m.reshape(2,3)
```

Output:

```
array([[0, 1, 2],
       [3, 4, 5]])
```

```
m.reshape(3,2)
```

Output:

```
array([[0, 1],
       [2, 3],
       [4, 5]])
```

Pandas

```
import numpy as np

import pandas as pd

dict = {"name":['aa', 'bb', 'cc'],
        "class":['fy','sy','ty'],
       "roll":[11, 22, 33]}
dict
```

Output: {'name': ['aa', 'bb', 'cc'], 'class': ['fy', 'sy', 'ty'], 'roll': [11, 22, 33]}

```
df = pd.DataFrame(dict)
df
```

Output:

	name	class	roll
0	aa	fy	11
1	bb	sy	22
2	cc	ty	33

```
df.to_csv('student.csv')
df.to_csv('index_false_student.csv', index=False)
df.head()
```

Output:

	name	class	roll
0	aa	fy	11
1	bb	sy	22
2	cc	ty	33

df.tail()

Output:

name	class	roll	
0	aa	fy	11
1	bb	sy	22
2	cc	ty	33

df.describe()

Output:

roll	
count	3.0
mean	22.0
std	11.0
min	11.0
25%	16.5
50%	22.0
75%	27.5
max	33.0

df.head(3)

Output:

	name	class	roll
0	aa	fy	11
1	bb	sy	22
2	cc	ty	33

df.to_csv('index.csv', index=False)

df

Output:

	name	class	roll
0	aa	fy	11
1	bb	sy	22
2	cc	ty	33

df.to_csv('index1.csv', index=False)

demo = pd.read_csv('index2.csv')

demo

Output:

	prod_id	name	area
0	2200	apple	andheri
1	3300	mango	parle
2	4400	orange	santacruz

demo['name']

Output:

0 apple

1 mango

2 orange

Name: name, dtype: object

demo['name'][1]

Output: 'mango'

demo['prod_id']

Output:

0   2200

1   3300

2   4400

Name: prod_id, dtype: int64

demo['prod_id'][2]

Output: 4400

demo['prod_id'][2] = 4004

Output: warning

<ipython-input-64-0c3a9eb8bc8c>:1: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

demo['prod_id'][2] = 4004

demo['prod_id']

Output:

0   2200

1   3300

2   4004

Name: prod_id, dtype: int64

demo.to_csv('new.csv')

demo

Output:

```
prod_id        name   area
0       2200   apple   andheri
1       3300   mango parle
2       4004   orange santacruz
```

demo.index = ['one', 'two', 'three']

demo

Output:

```
        prod_id        name   area
one     2200   apple   andheri
two     3300   mango parle
three   4004   orange Santacruz
```

s = pd.Series([2,3,4,5,6,7,8,9,10])

s

Output:

```
0    2
1    3
2    4
3    5
4    6
5    7
6    8
7    9
8   10
dtype: int64
```

```
s1 = pd.Series(np.random.rand(20))
s1
```

Output:

```
0     0.476242
1     0.332118
2     0.265113
3     0.722535
4     0.210917
5     0.204344
6     0.557794
7     0.585600
8     0.775989
9     0.555856
10    0.669544
11    0.874442
12    0.534156
13    0.260446
14    0.519634
15    0.776713
16    0.660476
17    0.748030
18    0.814161
19    0.366974
dtype: float64
```

```
df1 = pd.DataFrame(np.random.rand(20,10))
```

df1

Output:

	0	1	2	3	4	5	6	7	8	9
0	0.889829	0.217723	0.950464	0.114454	0.175260	0.171785	0.502882	0.431306	0.585802	0.824907
1	0.815695	0.961605	0.734357	0.617062	0.778672	0.737305	0.224034	0.792681	0.043488	0.755798
2	0.300321	0.297326	0.667170	0.810632	0.954124	0.527148	0.697780	0.679426	0.251948	0.124489
3	0.648760	0.770672	0.254008	0.025945	0.110265	0.602699	0.498752	0.413338	0.312994	0.293970
4	0.538527	0.630472	0.851454	0.061778	0.659211	0.565140	0.876626	0.598274	0.997209	0.087594
5	0.541544	0.934696	0.424254	0.602228	0.491561	0.614428	0.120711	0.491124	0.204725	0.973860
6	0.628961	0.302158	0.846598	0.068880	0.285089	0.233620	0.408571	0.277139	0.119807	0.524263
7	0.120473	0.407693	0.207758	0.042455	0.203260	0.605364	0.230598	0.450066	0.450713	0.003687
8	0.558722	0.927035	0.777533	0.483478	0.847846	0.096667	0.910407	0.327488	0.254891	0.337679
9	0.427066	0.629416	0.845941	0.008152	0.927802	0.945599	0.783255	0.626967	0.922936	0.155402
10	0.748707	0.909395	0.492470	0.046778	0.203244	0.102367	0.242721	0.370299	0.525937	0.410644
11	0.190404	0.602494	0.196155	0.650595	0.986109	0.680599	0.886406	0.262964	0.956797	0.719145
12	0.240944	0.520401	0.174845	0.756972	0.198388	0.355310	0.419668	0.514867	0.761939	0.560055
13	0.627101	0.535762	0.842373	0.963862	0.816623	0.052924	0.211294	0.368572	0.167157	0.388588
14	0.978139	0.237486	0.077492	0.209904	0.650783	0.663827	0.352613	0.130673	0.536371	0.074908

15	0.488940	0.336477	0.495782	0.341456	0.425742	0.461244	0.142852	0.294217	0.499867	0.226806
16	0.024142	0.726993	0.602587	0.815984	0.753234	0.515214	0.982483	0.124366	0.452646	0.757576
17	0.428680	0.481441	0.671396	0.437300	0.565147	0.387528	0.174145	0.295377	0.683534	0.326617
18	0.529209	0.236979	0.605650	0.002481	0.898732	0.043005	0.464004	0.849748	0.056447	0.424221
19	0.884170	0.725553	0.001559	0.273916	0.643806	0.102261	0.280440	0.360105	0.760108	0.674790

type(df1)

df1.describe()

Output:

	0	1	2	3	4	5	6	7	8	9
count	20.000000	20.000000	20.000000	20.000000	20.000000	20.000000	20.000000	20.000000	20.000000	20.000000
mean	0.530517	0.569589	0.535992	0.366716	0.578745	0.423202	0.470512	0.432950	0.477266	0.432250
std	0.263312	0.252397	0.294265	0.323563	0.296716	0.265350	0.286197	0.197626	0.300232	0.281850
min	0.024142	0.217723	0.001559	0.002481	0.110265	0.043005	0.120711	0.124366	0.043488	0.003687
25%	0.395380	0.327897	0.242445	0.058028	0.264632	0.154430	0.228957	0.295087	0.240142	0.208955
50%	0.540035	0.569128	0.604118	0.307686	0.647295	0.488229	0.414119	0.391818	0.476257	0.399616
75%	0.673747	0.737913	0.793743	0.625445	0.824429	0.607630	0.719149	0.535718	0.702677	0.685879
max	0.978139	0.961605	0.950464	0.963862	0.986109	0.945599	0.982483	0.849748	0.997209	0.973860

df1 [0][1] = "abc"

df1.head(10)

Output:

	0	1	2	3	4	5	6	7	8	9
0	0.889829	0.217723	0.950464	0.114454	0.175260	0.171785	0.502882	0.431306	0.585802	0.824907
1	abc	0.961605	0.734357	0.617062	0.778672	0.737305	0.224034	0.792681	0.043488	0.755798
2	0.300321	0.297326	0.667170	0.810632	0.954124	0.527148	0.697780	0.679426	0.251948	0.124489
3	0.64876	0.770672	0.254008	0.025945	0.110265	0.602699	0.498752	0.413338	0.312994	0.293970
4	0.538527	0.630472	0.851454	0.061778	0.659211	0.565140	0.876626	0.598274	0.997209	0.087594
5	0.541544	0.934696	0.424254	0.602228	0.491561	0.614428	0.120711	0.491124	0.204725	0.973860
6	0.628961	0.302158	0.846598	0.068880	0.285089	0.233620	0.408571	0.277139	0.119807	0.524263
7	0.120473	0.407693	0.207758	0.042455	0.203260	0.605364	0.230598	0.450066	0.450713	0.003687
8	0.558722	0.927035	0.777533	0.483478	0.847846	0.096667	0.910407	0.327488	0.254891	0.337679
9	0.427066	0.629416	0.845941	0.008152	0.927802	0.945599	0.783255	0.626967	0.922936	0.155402

df1.head(4)

Output:

	0	1	2	3	4	5	6	7	8	9
0	0.889829	efg	0.950464	0.114454	0.175260	0.171785	0.502882	0.431306	0.585802	0.824907

1 abc 0.961605 0.734357 0.617062 0.778672 0.737305 0.224034 0.792681 0.043488 0.755798

2 pqr 0.297326 0.667170 0.810632 0.954124 0.527148 0.697780 0.679426 0.251948 0.124489

3 0.64876 0.770672 0.254008 0.025945 0.110265 0.602699 0.498752 0.413338 0.312994 0.293970

df1[2][1]="aaa"

Output:

<ipython-input-95-93449a955d64>:1: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

 df1[2][1]="aaa"

C:\Users\aquis\anaconda3\lib\site-packages\pandas\core\indexing.py:1637: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

 self._setitem_single_block(indexer, value, name)

df1

Output:

0 1 2 3 4 5 6 7 8 9

0 0.889829 efg 0.950464 0.114454 0.175260 0.171785 0.502882 0.431306 0.585802 0.824907

1 abc 0.961605 aaa 0.617062 0.778672 0.737305 0.224034 0.792681 0.043488 0.755798

2	pqr	0.297326	0.66717	0.810632	0.954124	0.527148
	0.697780	0.679426	0.251948	0.124489		
3	0.64876	0.770672	0.254008	0.025945	0.110265	0.602699
	0.498752	0.413338	0.312994	0.293970		
4	0.538527	0.630472	0.851454	0.061778	0.659211	0.565140
	0.876626	0.598274	0.997209	0.087594		
5	0.541544	0.934696	0.424254	0.602228	0.491561	0.614428
	0.120711	0.491124	0.204725	0.973860		
6	0.628961	0.302158	0.846598	0.068880	0.285089	0.233620
	0.408571	0.277139	0.119807	0.524263		
7	0.120473	0.407693	0.207758	0.042455	0.203260	0.605364
	0.230598	0.450066	0.450713	0.003687		
8	0.558722	0.927035	0.777533	0.483478	0.847846	0.096667
	0.910407	0.327488	0.254891	0.337679		
9	0.427066	0.629416	0.845941	0.008152	0.927802	0.945599
	0.783255	0.626967	0.922936	0.155402		
10	0.748707	0.909395	0.49247	0.046778	0.203244	0.102367
	0.242721	0.370299	0.525937	0.410644		
11	0.190404	0.602494	0.196155	0.650595	0.986109	0.680599
	0.886406	0.262964	0.956797	0.719145		
12	0.240944	0.520401	0.174845	0.756972	0.198388	0.355310
	0.419668	0.514867	0.761939	0.560055		
13	0.627101	0.535762	0.842373	0.963862	0.816623	0.052924
	0.211294	0.368572	0.167157	0.388588		
14	0.978139	0.237486	0.077492	0.209904	0.650783	0.663827
	0.352613	0.130673	0.536371	0.074908		
15	0.48894	0.336477	0.495782	0.341456	0.425742	0.461244
	0.142852	0.294217	0.499867	0.226806		
16	0.024142	0.726993	0.602587	0.815984	0.753234	0.515214
	0.982483	0.124366	0.452646	0.757576		
17	0.42868	0.481441	0.671396	0.437300	0.565147	0.387528
	0.174145	0.295377	0.683534	0.326617		
18	0.529209	0.236979	0.60565	0.002481	0.898732	0.043005
	0.464004	0.849748	0.056447	0.424221		

19	0.88417	0.725553	0.001559	0.273916	0.643806	0.102261
	0.280440	0.360105	0.760108	0.674790		

demo

Output:

prod_id		name	area
one	2200	apple	andheri
two	3300	mango	parle
three	4004	grapes	santacruz

demo['prod_id'][1] = 5005

<ipython-input-98-547956110199>:1: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

demo['prod_id'][1] = 5005

demo

Output:

prod_id		name	area
one	2200	apple	andheri
two	5005	mango	parle
three	4004	grapes	santacruz

demo.dtypes

Output:

```
prod_id    int64
name       object
area       object
dtype: object
```

df1.dtypes

Output:

```
0     object
1     object
2     object
3    float64
4    float64
5    float64
6    float64
7    float64
8    float64
9    float64
dtype: object
```

df1

Output:

	0	1	2	3	4	5	6	7	8	9
0	0.889829	efg	0.950464	0.114454	0.175260	0.171785	0.502882	0.431306	0.585802	0.824907
1	abc	0.961605	aaa	0.617062	0.778672	0.737305	0.224034	0.792681	0.043488	0.755798

2	pqr	0.297326	0.66717	0.810632	0.954124	0.527148
	0.697780	0.679426	0.251948	0.124489		
3	0.64876	0.770672	0.254008	0.025945	0.110265	0.602699
	0.498752	0.413338	0.312994	0.293970		
4	0.538527	0.630472	0.851454	0.061778	0.659211	0.565140
	0.876626	0.598274	0.997209	0.087594		
5	0.541544	0.934696	0.424254	0.602228	0.491561	0.614428
	0.120711	0.491124	0.204725	0.973860		
6	0.628961	0.302158	0.846598	0.068880	0.285089	0.233620
	0.408571	0.277139	0.119807	0.524263		
7	0.120473	0.407693	0.207758	0.042455	0.203260	0.605364
	0.230598	0.450066	0.450713	0.003687		
8	0.558722	0.927035	0.777533	0.483478	0.847846	0.096667
	0.910407	0.327488	0.254891	0.337679		
9	0.427066	0.629416	0.845941	0.008152	0.927802	0.945599
	0.783255	0.626967	0.922936	0.155402		
10	0.748707	0.909395	0.49247	0.046778	0.203244	0.102367
	0.242721	0.370299	0.525937	0.410644		
11	0.190404	0.602494	0.196155	0.650595	0.986109	0.680599
	0.886406	0.262964	0.956797	0.719145		
12	0.240944	0.520401	0.174845	0.756972	0.198388	0.355310
	0.419668	0.514867	0.761939	0.560055		
13	0.627101	0.535762	0.842373	0.963862	0.816623	0.052924
	0.211294	0.368572	0.167157	0.388588		
14	0.978139	0.237486	0.077492	0.209904	0.650783	0.663827
	0.352613	0.130673	0.536371	0.074908		
15	0.48894	0.336477	0.495782	0.341456	0.425742	0.461244
	0.142852	0.294217	0.499867	0.226806		
16	0.024142	0.726993	0.602587	0.815984	0.753234	0.515214
	0.982483	0.124366	0.452646	0.757576		
17	0.42868	0.481441	0.671396	0.437300	0.565147	0.387528
	0.174145	0.295377	0.683534	0.326617		
18	0.529209	0.236979	0.60565	0.002481	0.898732	0.043005
	0.464004	0.849748	0.056447	0.424221		

19 0.88417 0.725553 0.001559 0.273916 0.643806 0.102261
0.280440 0.360105 0.760108 0.674790

df1.to_numpy()

Output:

```
array([[0.8898290587574625, 'efg', 0.9504637009415536,
        0.11445350753707939, 0.1752600347199531, 0.17178486545231497,
        0.5028824898103187, 0.431305684264646, 0.5858017094187148,
        0.8249071762869403],
       ['abc', 0.9616049518794304, 'aaa', 0.6170616309044902,
        0.7786715778059061, 0.7373050333549065, 0.22403425159455848,
        0.7926807836787797, 0.04348811153552434, 0.7557984441046045],
       ['pqr', 0.29732601721264285, 0.6671699248305508,
        0.8106324766594586, 0.9541240643323579, 0.5271480185664397,
        0.6977800757136665, 0.6794261624792959, 0.25194753570017625,
        0.12448932434428084],
       [0.6487599631982358, 0.7706715150467816, 0.2540076701928634,
        0.025944778723128806, 0.11026541677027257, 0.6026986958103315,
        0.4987517228774331, 0.4133382664044277, 0.31299444322854375,
        0.2939702728123903],
       [0.5385269288296717, 0.6304722606997493, 0.8514542001716628,
        0.06177752739395137, 0.6592108981215252, 0.5651400220152262,
        0.8766258842206889, 0.5982735952347097, 0.9972089870133152,
        0.08759388187127126],
       [0.5415438818693964, 0.9346959871836261, 0.4242535605595833,
        0.6022280894199854, 0.49156096274871275, 0.6144283361246711,
        0.1207113831461939, 0.49112392727947407, 0.20472539299943238,
```

0.9738604915353611],
[0.6289611558850483, 0.3021577039431248, 0.846597984603601,
0.06887979170151126, 0.2850892718126471, 0.23362048790526357,
0.4085709972198358, 0.27713931102605127, 0.11980742869681049,
0.5242625516236767],
[0.12047323514029218, 0.4076931276538779, 0.20775814224756806,
0.04245494922213655, 0.20325987717441063, 0.6053639969934034,
0.23059818844544866, 0.4500660481280979, 0.4507134591810671,
0.003687096791495703],
[0.558721730405733, 0.9270350875298212, 0.7775332056913566,
0.48347777029749706, 0.8478464104949196, 0.09666694338157167,
0.9104070401598463, 0.32748771998561244, 0.2548911720594357,
0.3376787878401263],
[0.42706601825584, 0.6294157375842707, 0.8459413328009987,
0.008151933416917778, 0.9278019823366778, 0.9455985340636448,
0.7832549150376663, 0.6269671160495615, 0.9229361644222673,
0.15540235785663703],
[0.7487069527367622, 0.9093947236386258, 0.49246977663430314,
0.04677816297810866, 0.20324374999174266, 0.1023672526008913,
0.24272067170750022, 0.3702986708319129, 0.5259374157967004,
0.41064368110825533],
[0.19040400501064036, 0.6024937826358194, 0.19615487722677982,
0.6505953528159473, 0.9861090888652927, 0.6805992380978194,
0.8864063677267591, 0.26296403260958523, 0.9567974966351677,
0.7191453275424041],
[0.24094430235912145, 0.5204007187411817, 0.1748447311932637,
0.7569724353480395, 0.19838767252402834, 0.3553101063124129,

0.4196679434769559, 0.5148666686957333, 0.7619392858693081,
0.5600548982309 45],
[0.6271005616304949, 0.5357621588665652, 0.8423727130622828,
0.9638621473539152, 0.8166226983755571, 0.05292449501022456,
0.21129437931396955, 0.36857150469345834, 0.1671565144741678,
0.38858763486196646],
[0.9781386628079035, 0.23748583454810612, 0.0774922086014398,
0.20990350008401104, 0.650783131011194, 0.6638271605151387,
0.35261309636887295, 0.1306729819085617, 0.5363706849354714,
0.07490764205042],
[0.48894020892946477, 0.33647710082718496, 0.4957819333797754,
0.3414556185184121, 0.4257420648833655, 0.46124407258671696,
0.1428515761588003, 0.294216975284216, 0.49986677703088356,
0.2268061817643775],
[0.02414249471968255, 0.7269933849502771, 0.6025866134022627,
0.8159838725163027, 0.7532337808624099, 0.515214318968477 7,
0.9824832756390718, 0.12436628539572403, 0.4526464668408925,
0.7575758731905909],
[0.4286800832564034, 0.48144143422024277, 0.6713958370200016,
0.43730039897594153, 0.5651468661339273, 0.38752775998992894,
0.1741454366748063, 0.2953770888681998, 0.6835335205019395,
0.32661688397358923],
[0.5292085630387563, 0.23697946207749765, 0.6056501407655644,
0.0024813922441588865, 0.8987318909075309, 0.04300455769905975,
0.46400399742872456, 0.8497477915770419, 0.05644742404326131,
0.4242205865280846],
[0.8841697571458046, 0.7255526457480526, 0.0015591409819755153,

0.2739157678767543, 0.6438064650372433, 0.10226113788566826,

0.2804404425187115, 0.36010547725733033, 0.7601084640428719,

0.6747895650580462]], dtype=object)

df1

Output:

	0	1	2	3	4	5	6	7	8	9
0	0.889829	efg	0.950464	0.114454	0.175260	0.171785	0.502882	0.431306	0.585802	0.824907
1	abc	0.961605	aaa	0.617062	0.778672	0.737305	0.224034	0.792681	0.043488	0.755798
2	pqr	0.297326	0.66717	0.810632	0.954124	0.527148	0.697780	0.679426	0.251948	0.124489
3	0.64876	0.770672	0.254008	0.025945	0.110265	0.602699	0.498752	0.413338	0.312994	0.293970
4	0.538527	0.630472	0.851454	0.061778	0.659211	0.565140	0.876626	0.598274	0.997209	0.087594
5	0.541544	0.934696	0.424254	0.602228	0.491561	0.614428	0.120711	0.491124	0.204725	0.973860
6	0.628961	0.302158	0.846598	0.068880	0.285089	0.233620	0.408571	0.277139	0.119807	0.524263
7	0.120473	0.407693	0.207758	0.042455	0.203260	0.605364	0.230598	0.450066	0.450713	0.003687
8	0.558722	0.927035	0.777533	0.483478	0.847846	0.096667	0.910407	0.327488	0.254891	0.337679
9	0.427066	0.629416	0.845941	0.008152	0.927802	0.945599	0.783255	0.626967	0.922936	0.155402
10	0.748707	0.909395	0.49247	0.046778	0.203244	0.102367	0.242721	0.370299	0.525937	0.410644
11	0.190404	0.602494	0.196155	0.650595	0.986109	0.680599	0.886406	0.262964	0.956797	0.719145

12	0.240944	0.520401	0.174845	0.756972	0.198388	0.355310	0.419668	0.514867	0.761939	0.560055
13	0.627101	0.535762	0.842373	0.963862	0.816623	0.052924	0.211294	0.368572	0.167157	0.388588
14	0.978139	0.237486	0.077492	0.209904	0.650783	0.663827	0.352613	0.130673	0.536371	0.074908
15	0.48894	0.336477	0.495782	0.341456	0.425742	0.461244	0.142852	0.294217	0.499867	0.226806
16	0.024142	0.726993	0.602587	0.815984	0.753234	0.515214	0.982483	0.124366	0.452646	0.757576
17	0.42868	0.481441	0.671396	0.437300	0.565147	0.387528	0.174145	0.295377	0.683534	0.326617
18	0.529209	0.236979	0.60565	0.002481	0.898732	0.043005	0.464004	0.849748	0.056447	0.424221
19	0.88417	0.725553	0.001559	0.273916	0.643806	0.102261	0.280440	0.360105	0.760108	0.674790

demo

Output:

	prod_id	name	area
one	2200	apple	andheri
two	5005	mango	parle
three	4004	grapes	santacruz

demo.T

Output:

	one	two	three
prod_id	2200	5005	4004
name	apple	mango	grapes
area	andheri	parle	santacruz

```
df2 = pd.DataFrame(np.random.rand(10,5))
df2
```

Output:

	0	1	2	3	4
0	0.988782	0.155982	0.163659	0.216378	0.338656
1	0.922171	0.810851	0.249822	0.283435	0.181059
2	0.069235	0.844811	0.165427	0.086819	0.301486
3	0.789741	0.358560	0.738854	0.373372	0.934196
4	0.405396	0.146483	0.516349	0.259770	0.846987
5	0.929204	0.212274	0.604740	0.422453	0.722843
6	0.247970	0.452907	0.853457	0.639186	0.590882
7	0.672903	0.397623	0.773096	0.071042	0.135975
8	0.139015	0.843306	0.936715	0.941274	0.551718
9	0.052673	0.486642	0.234463	0.257344	0.981282

```
df2.sort_index(axis=1, ascending=False)
```

Output:

	4	3	2	1	0
0	0.338656	0.216378	0.163659	0.155982	0.988782
1	0.181059	0.283435	0.249822	0.810851	0.922171
2	0.301486	0.086819	0.165427	0.844811	0.069235
3	0.934196	0.373372	0.738854	0.358560	0.789741
4	0.846987	0.259770	0.516349	0.146483	0.405396
5	0.722843	0.422453	0.604740	0.212274	0.929204
6	0.590882	0.639186	0.853457	0.452907	0.247970
7	0.135975	0.071042	0.773096	0.397623	0.672903

8	0.551718	0.941274	0.936715	0.843306	0.139015
9	0.981282	0.257344	0.234463	0.486642	0.052673

```
demo
```

Output:

prod_id		name	area
one	2200	apple	andheri
two	5005	mango	parle
three	4004	grapes	santacruz

```
p = demo.sort_values('name')
p
```

Output:

prod_id		name	area
one	2200	apple	andheri
three	4004	grapes	santacruz
two	5005	mango	parle

```
q = demo.groupby('prod_id')
q
```

Output:

<pandas.core.groupby.generic.DataFrameGroupBy object at 0x000001EE05FE2A30>

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

```
demo.hist('prod_id', bins=20)
```

ax = plt.gca()

ax.set_yscale('log')

Output: (graph)

demo.info()

Output:

```
<class 'pandas.core.frame.DataFrame'>
Index: 3 entries, one to three
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   prod_id  3 non-null      int64
 1   name     3 non-null      object
 2   area     3 non-null      object
dtypes: int64(1), object(2)
memory usage: 204.0+ bytes
```

df1

Output:

0	1	2	3	4	5	6	7	8	9
0	0.889829	efg	0.950464	0.114454	0.175260	0.171785	0.502882	0.431306	0.585802	0.824907
1	abc	0.961605	aaa	0.617062	0.778672	0.737305	0.224034	0.792681	0.043488	0.755798
2	pqr	0.297326	0.66717	0.810632	0.954124	0.527148	0.697780	0.679426	0.251948	0.124489
3	0.64876	0.770672	0.254008	0.025945	0.110265	0.602699	0.498752	0.413338	0.312994	0.293970

4	0.538527	0.630472	0.851454	0.061778	0.659211	0.565140
	0.876626	0.598274	0.997209	0.087594		
5	0.541544	0.934696	0.424254	0.602228	0.491561	0.614428
	0.120711	0.491124	0.204725	0.973860		
6	0.628961	0.302158	0.846598	0.068880	0.285089	0.233620
	0.408571	0.277139	0.119807	0.524263		
7	0.120473	0.407693	0.207758	0.042455	0.203260	0.605364
	0.230598	0.450066	0.450713	0.003687		
8	0.558722	0.927035	0.777533	0.483478	0.847846	0.096667
	0.910407	0.327488	0.254891	0.337679		
9	0.427066	0.629416	0.845941	0.008152	0.927802	0.945599
	0.783255	0.626967	0.922936	0.155402		
10	0.748707	0.909395	0.49247	0.046778	0.203244	0.102367
	0.242721	0.370299	0.525937	0.410644		
11	0.190404	0.602494	0.196155	0.650595	0.986109	0.680599
	0.886406	0.262964	0.956797	0.719145		
12	0.240944	0.520401	0.174845	0.756972	0.198388	0.355310
	0.419668	0.514867	0.761939	0.560055		
13	0.627101	0.535762	0.842373	0.963862	0.816623	0.052924
	0.211294	0.368572	0.167157	0.388588		
14	0.978139	0.237486	0.077492	0.209904	0.650783	0.663827
	0.352613	0.130673	0.536371	0.074908		
15	0.48894	0.336477	0.495782	0.341456	0.425742	0.461244
	0.142852	0.294217	0.499867	0.226806		
16	0.024142	0.726993	0.602587	0.815984	0.753234	0.515214
	0.982483	0.124366	0.452646	0.757576		
17	0.42868	0.481441	0.671396	0.437300	0.565147	0.387528
	0.174145	0.295377	0.683534	0.326617		
18	0.529209	0.236979	0.60565	0.002481	0.898732	0.043005
	0.464004	0.849748	0.056447	0.424221		
19	0.88417	0.725553	0.001559	0.273916	0.643806	0.102261
	0.280440	0.360105	0.760108	0.674790		

df1.loc[0,0]=990

df1

Output:

	0	1	2	3	4	5	6	7	8	9
0	990	efg	0.950464	0.114454	0.175260	0.171785	0.502882	0.431306	0.585802	0.824907
1	abc	0.961605	aaa	0.617062	0.778672	0.737305	0.224034	0.792681	0.043488	0.755798
2	pqr	0.297326	0.66717	0.810632	0.954124	0.527148	0.697780	0.679426	0.251948	0.124489
3	0.64876	0.770672	0.254008	0.025945	0.110265	0.602699	0.498752	0.413338	0.312994	0.293970
4	0.538527	0.630472	0.851454	0.061778	0.659211	0.565140	0.876626	0.598274	0.997209	0.087594
5	0.541544	0.934696	0.424254	0.602228	0.491561	0.614428	0.120711	0.491124	0.204725	0.973860
6	0.628961	0.302158	0.846598	0.068880	0.285089	0.233620	0.408571	0.277139	0.119807	0.524263
7	0.120473	0.407693	0.207758	0.042455	0.203260	0.605364	0.230598	0.450066	0.450713	0.003687
8	0.558722	0.927035	0.777533	0.483478	0.847846	0.096667	0.910407	0.327488	0.254891	0.337679
9	0.427066	0.629416	0.845941	0.008152	0.927802	0.945599	0.783255	0.626967	0.922936	0.155402
10	0.748707	0.909395	0.49247	0.046778	0.203244	0.102367	0.242721	0.370299	0.525937	0.410644
11	0.190404	0.602494	0.196155	0.650595	0.986109	0.680599	0.886406	0.262964	0.956797	0.719145
12	0.240944	0.520401	0.174845	0.756972	0.198388	0.355310	0.419668	0.514867	0.761939	0.560055
13	0.627101	0.535762	0.842373	0.963862	0.816623	0.052924	0.211294	0.368572	0.167157	0.388588

14	0.978139	0.237486	0.077492	0.209904	0.650783	0.663827	0.352613	0.130673	0.536371	0.074908
15	0.48894	0.336477	0.495782	0.341456	0.425742	0.461244	0.142852	0.294217	0.499867	0.226806
16	0.024142	0.726993	0.602587	0.815984	0.753234	0.515214	0.982483	0.124366	0.452646	0.757576
17	0.42868	0.481441	0.671396	0.437300	0.565147	0.387528	0.174145	0.295377	0.683534	0.326617
18	0.529209	0.236979	0.60565	0.002481	0.898732	0.043005	0.464004	0.849748	0.056447	0.424221
19	0.88417	0.725553	0.001559	0.273916	0.643806	0.102261	0.280440	0.360105	0.760108	0.674790

```
df2
```

Output:

	0	1	2	3	4
0	0.988782	0.155982	0.163659	0.216378	0.338656
1	0.922171	0.810851	0.249822	0.283435	0.181059
2	0.069235	0.844811	0.165427	0.086819	0.301486
3	0.789741	0.358560	0.738854	0.373372	0.934196
4	0.405396	0.146483	0.516349	0.259770	0.846987
5	0.929204	0.212274	0.604740	0.422453	0.722843
6	0.247970	0.452907	0.853457	0.639186	0.590882
7	0.672903	0.397623	0.773096	0.071042	0.135975
8	0.139015	0.843306	0.936715	0.941274	0.551718
9	0.052673	0.486642	0.234463	0.257344	0.981282

```
df2.loc[0,0]=990
df2
```

Output:

	0	1	2	3	4
0	990.000000	0.155982	0.163659	0.216378	0.338656
1	0.922171	0.810851	0.249822	0.283435	0.181059
2	0.069235	0.844811	0.165427	0.086819	0.301486
3	0.789741	0.358560	0.738854	0.373372	0.934196
4	0.405396	0.146483	0.516349	0.259770	0.846987
5	0.929204	0.212274	0.604740	0.422453	0.722843
6	0.247970	0.452907	0.853457	0.639186	0.590882
7	0.672903	0.397623	0.773096	0.071042	0.135975
8	0.139015	0.843306	0.936715	0.941274	0.551718
9	0.052673	0.486642	0.234463	0.257344	0.981282

```
df2.columns = list("ABCDE")
df2
```

Output:

	A	B	C	D	E
0	990.000000	0.155982	0.163659	0.216378	0.338656
1	0.922171	0.810851	0.249822	0.283435	0.181059
2	0.069235	0.844811	0.165427	0.086819	0.301486
3	0.789741	0.358560	0.738854	0.373372	0.934196
4	0.405396	0.146483	0.516349	0.259770	0.846987
5	0.929204	0.212274	0.604740	0.422453	0.722843
6	0.247970	0.452907	0.853457	0.639186	0.590882
7	0.672903	0.397623	0.773096	0.071042	0.135975
8	0.139015	0.843306	0.936715	0.941274	0.551718
9	0.052673	0.486642	0.234463	0.257344	0.981282

```
df2.loc[0,'A']=89
df2
```

Output:

	A	B	C	D	E
0	89.000000	899.000000	0.163659	0.216378	0.338656
1	0.922171	0.810851	0.249822	0.283435	0.181059
2	0.069235	0.844811	0.165427	0.086819	0.301486
3	0.789741	0.358560	0.738854	0.373372	0.934196
4	0.405396	0.146483	0.516349	0.259770	0.846987
5	0.929204	0.212274	0.604740	0.422453	0.722843
6	0.247970	0.452907	0.853457	0.639186	0.590882
7	0.672903	0.397623	0.773096	0.071042	0.135975
8	0.139015	0.843306	0.936715	0.941274	0.551718
9	0.052673	0.486642	0.234463	0.257344	0.981282

```
dt = pd.DataFrame(np.random.rand(10,5))
dt
```

Ouput:

	0	1	2	3	4
0	0.514973	0.132473	0.662300	0.870011	0.099254
1	0.505812	0.655760	0.709748	0.459002	0.258930
2	0.446541	0.850593	0.959236	0.653753	0.742279
3	0.364539	0.001264	0.233297	0.904143	0.396865
4	0.214473	0.344468	0.010521	0.403364	0.834405
5	0.543493	0.511075	0.517688	0.971037	0.386030
6	0.757976	0.310684	0.385691	0.767525	0.537692

7	0.532578	0.294248	0.438818	0.581528	0.483544
8	0.383618	0.366597	0.258645	0.600649	0.044865
9	0.649240	0.894046	0.534226	0.551215	0.025614

```
dt[0][0]=88
dt
```

Output:

	0	1	2	3	4
0	88.000000	0.132473	0.662300	0.870011	0.099254
1	0.505812	0.655760	0.709748	0.459002	0.258930
2	0.446541	0.850593	0.959236	0.653753	0.742279
3	0.364539	0.001264	0.233297	0.904143	0.396865
4	0.214473	0.344468	0.010521	0.403364	0.834405
5	0.543493	0.511075	0.517688	0.971037	0.386030
6	0.757976	0.310684	0.385691	0.767525	0.537692
7	0.532578	0.294248	0.438818	0.581528	0.483544
8	0.383618	0.366597	0.258645	0.600649	0.044865
9	0.649240	0.894046	0.534226	0.551215	0.025614

```
dt.sort_index(axis=1, ascending=False)
```

Output:

	4	3	2	1	0
0	0.099254	0.870011	0.662300	0.132473	88.000000
1	0.258930	0.459002	0.709748	0.655760	0.505812
2	0.742279	0.653753	0.959236	0.850593	0.446541
3	0.396865	0.904143	0.233297	0.001264	0.364539
4	0.834405	0.403364	0.010521	0.344468	0.214473

5	0.386030	0.971037	0.517688	0.511075	0.543493
6	0.537692	0.767525	0.385691	0.310684	0.757976
7	0.483544	0.581528	0.438818	0.294248	0.532578
8	0.044865	0.600649	0.258645	0.366597	0.383618
9	0.025614	0.551215	0.534226	0.894046	0.649240

```
dt[0][0]=0.9
dt
```

Output:

	0	1	2	3	4
0	0.900000	0.132473	0.662300	0.870011	0.099254
1	0.505812	0.655760	0.709748	0.459002	0.258930
2	0.446541	0.850593	0.959236	0.653753	0.742279
3	0.364539	0.001264	0.233297	0.904143	0.396865
4	0.214473	0.344468	0.010521	0.403364	0.834405
5	0.543493	0.511075	0.517688	0.971037	0.386030
6	0.757976	0.310684	0.385691	0.767525	0.537692
7	0.532578	0.294248	0.438818	0.581528	0.483544
8	0.383618	0.366597	0.258645	0.600649	0.044865
9	0.649240	0.894046	0.534226	0.551215	0.025614

```
dt.columns = list("abcde")
dt
```

Output:

	a	b	c	d	e
0	0.900000	0.132473	0.662300	0.870011	0.099254
1	0.505812	0.655760	0.709748	0.459002	0.258930

2	0.446541	0.850593	0.959236	0.653753	0.742279
3	0.364539	0.001264	0.233297	0.904143	0.396865
4	0.214473	0.344468	0.010521	0.403364	0.834405
5	0.543493	0.511075	0.517688	0.971037	0.386030
6	0.757976	0.310684	0.385691	0.767525	0.537692
7	0.532578	0.294248	0.438818	0.581528	0.483544
8	0.383618	0.366597	0.258645	0.600649	0.044865
9	0.649240	0.894046	0.534226	0.551215	0.025

dt.loc[0,'b']=68

dt

Output:

	a	b	c	d	e
0	0.900000	68.000000	0.662300	0.870011	0.099254
1	0.505812	0.655760	0.709748	0.459002	0.258930
2	0.446541	0.850593	0.959236	0.653753	0.742279
3	0.364539	0.001264	0.233297	0.904143	0.396865
4	0.214473	0.344468	0.010521	0.403364	0.834405
5	0.543493	0.511075	0.517688	0.971037	0.386030
6	0.757976	0.310684	0.385691	0.767525	0.537692
7	0.532578	0.294248	0.438818	0.581528	0.483544
8	0.383618	0.366597	0.258645	0.600649	0.044865
9	0.649240	0.894046	0.534226	0.551215	0.025614

dt.loc[0,0]=98

dt

Output:

a	b	c	d	e	0	
0	0.900000	68.000000	0.662300	0.870011	0.099254	98.0
1	0.505812	0.655760	0.709748	0.459002	0.258930	NaN
2	0.446541	0.850593	0.959236	0.653753	0.742279	NaN
3	0.364539	0.001264	0.233297	0.904143	0.396865	NaN
4	0.214473	0.344468	0.010521	0.403364	0.834405	NaN
5	0.543493	0.511075	0.517688	0.971037	0.386030	NaN
6	0.757976	0.310684	0.385691	0.767525	0.537692	NaN
7	0.532578	0.294248	0.438818	0.581528	0.483544	NaN
8	0.383618	0.366597	0.258645	0.600649	0.044865	NaN
9	0.649240	0.894046	0.534226	0.551215	0.025614	NaN

```
dt.drop(0,axis=1)
```

Output:

a	b	c	d	e	
0	0.900000	68.000000	0.662300	0.870011	0.099254
1	0.505812	0.655760	0.709748	0.459002	0.258930
2	0.446541	0.850593	0.959236	0.653753	0.742279
3	0.364539	0.001264	0.233297	0.904143	0.396865
4	0.214473	0.344468	0.010521	0.403364	0.834405
5	0.543493	0.511075	0.517688	0.971037	0.386030
6	0.757976	0.310684	0.385691	0.767525	0.537692
7	0.532578	0.294248	0.438818	0.581528	0.483544
8	0.383618	0.366597	0.258645	0.600649	0.044865
9	0.649240	0.894046	0.534226	0.551215	0.025614

```
newdt = dt.drop(0,axis=1)
```

```
newdt
```

Output:

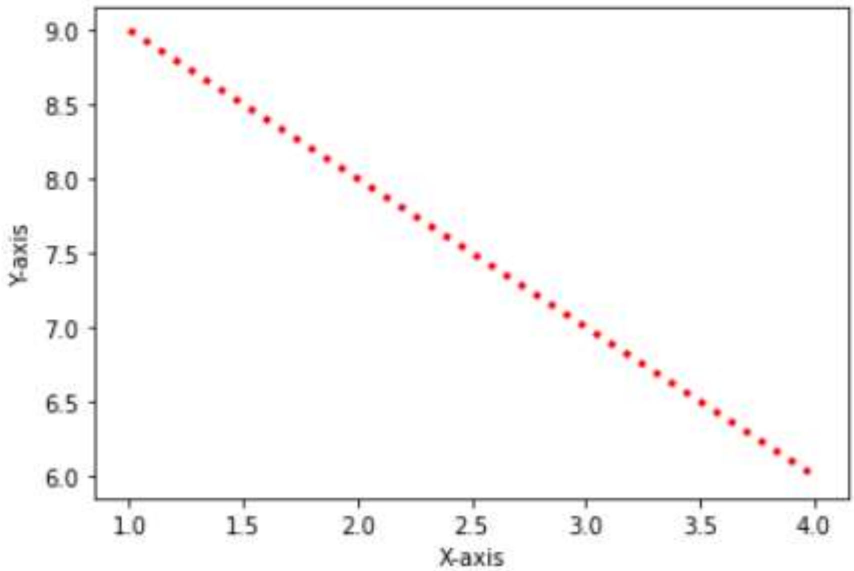
	a	b	c	d	e
0	0.900000	68.000000	0.662300	0.870011	0.099254
1	0.505812	0.655760	0.709748	0.459002	0.258930
2	0.446541	0.850593	0.959236	0.653753	0.742279
3	0.364539	0.001264	0.233297	0.904143	0.396865
4	0.214473	0.344468	0.010521	0.403364	0.834405
5	0.543493	0.511075	0.517688	0.971037	0.386030
6	0.757976	0.310684	0.385691	0.767525	0.537692
7	0.532578	0.294248	0.438818	0.581528	0.483544
8	0.383618	0.366597	0.258645	0.600649	0.044865
9	0.649240	0.894046	0.534226	0.551215	0.025614

Practical 3: Study of Python Libraries:

a) Matplotlib b) Scikit Learn

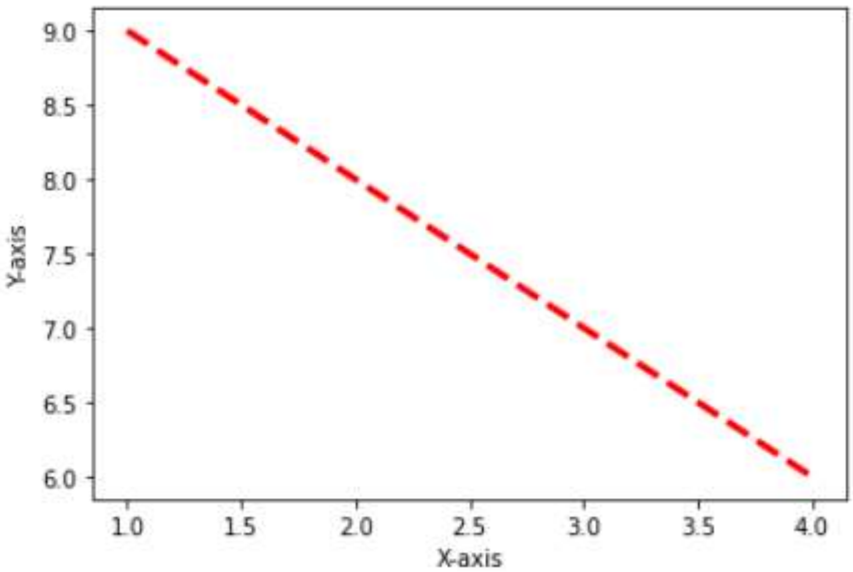
```
import matplotlib.pyplot as pl

x = [1, 2, 3, 4]
y = [9, 8, 7, 6]
pl.xlabel("X-axis")
pl.ylabel("Y-axis")
pl.plot(x,y,'r',linewidth = 3, linestyle = 'dashdot')
pl.show()
```

Output:




```
pl.xlabel("X-axis")
pl.ylabel("Y-axis")
pl.plot(x,y,'r',linewidth = 3, linestyle = 'dashed')
pl.show()
```

Output:



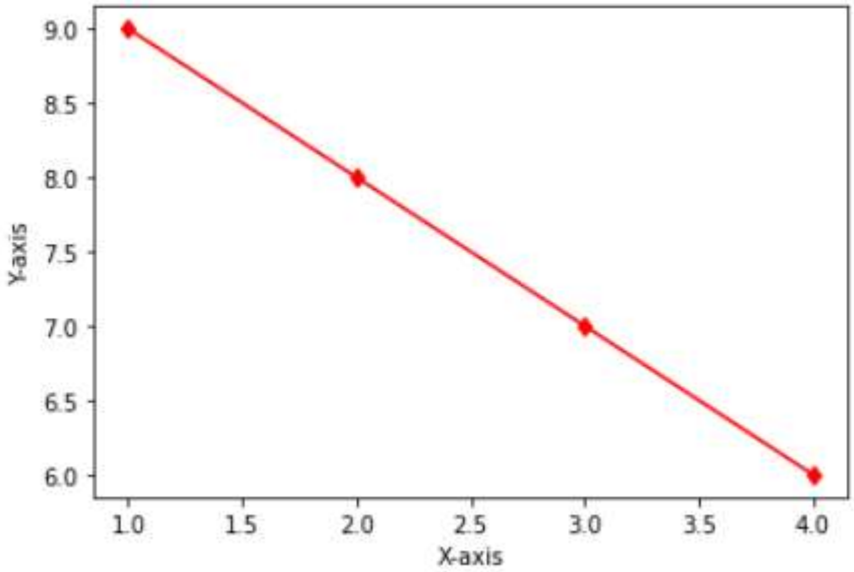
pl.xlabel("X-axis")

pl.ylabel("Y-axis")

pl.plot(x,y,'r',marker='d')

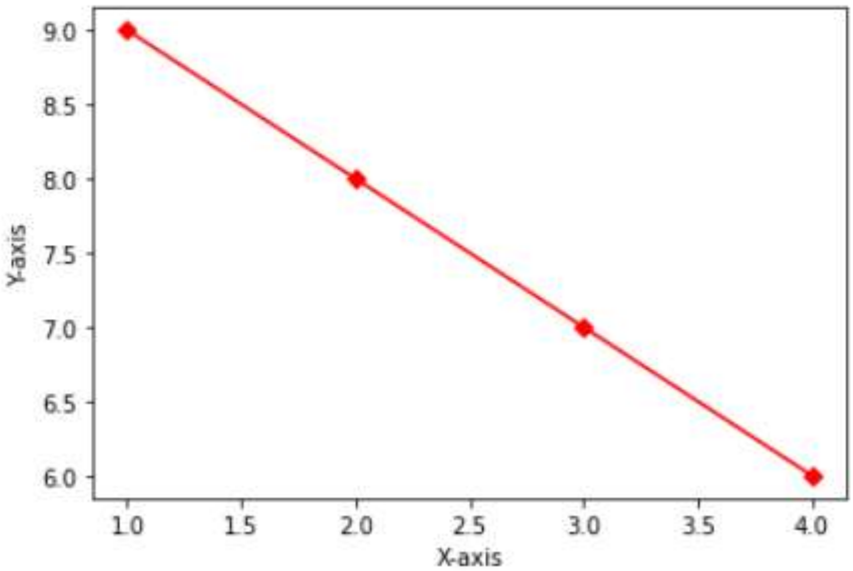
pl.show()

Output:




```
pl.xlabel("X-axis")

pl.ylabel("Y-axis")

pl.plot(x,y,'r',marker='D')

pl.show()
```

Output:



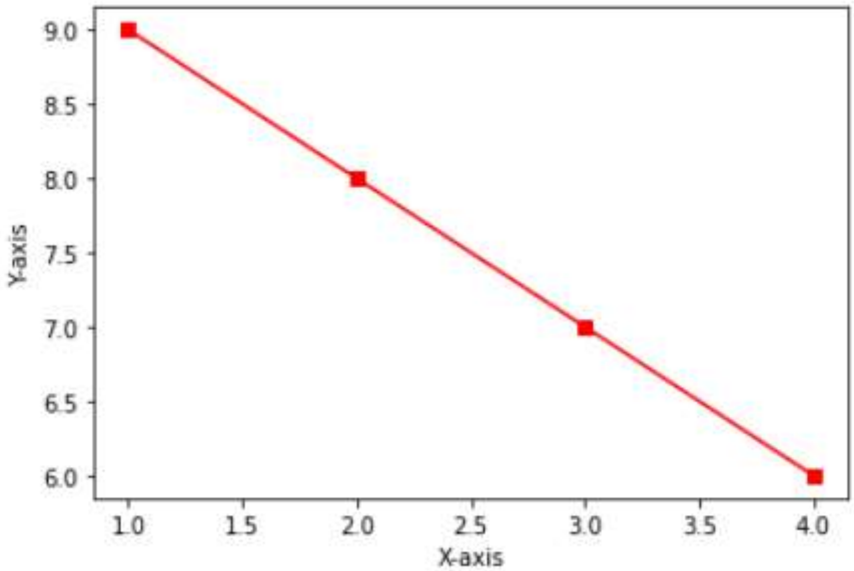

```
pl.xlabel("X-axis")
```

pl.ylabel("Y-axis")

pl.plot(x,y,'r',marker='s')

pl.show()

Output:



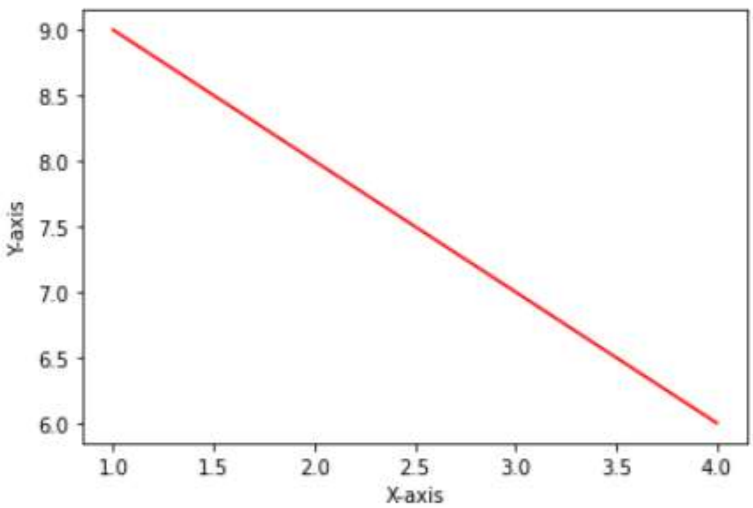
pl.xlabel("X-axis")

pl.ylabel("Y-axis")

pl.plot(x,y,'r',marker='')

pl.show()

Output:



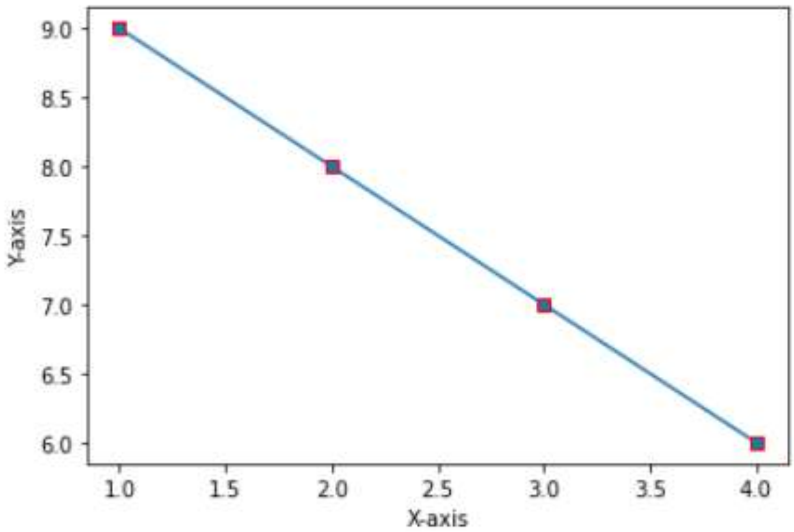
pl.xlabel("X-axis")

pl.ylabel("Y-axis")

pl.plot(x,y,marker='s', markeredgecolor='red')

pl.show()

Output:



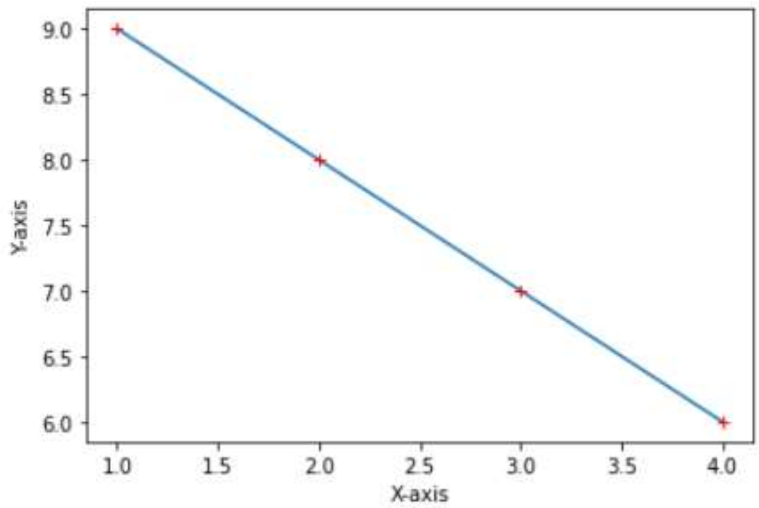
pl.xlabel("X-axis")

pl.ylabel("Y-axis")

pl.plot(x,y,marker='+', markeredgecolor='red')

pl.show()

Output:



pl.xlabel("X-axis")

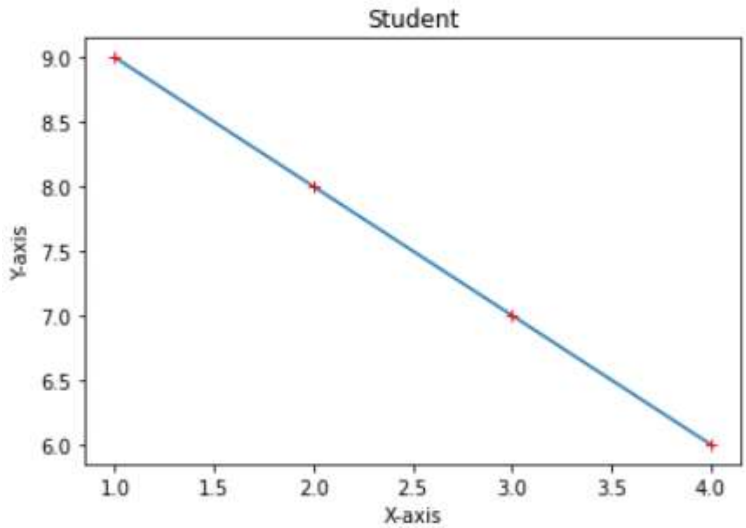
pl.ylabel("Y-axis")

pl.title("Student")

pl.plot(x,y,marker='+', markeredgecolor='red')

pl.show()

Output:



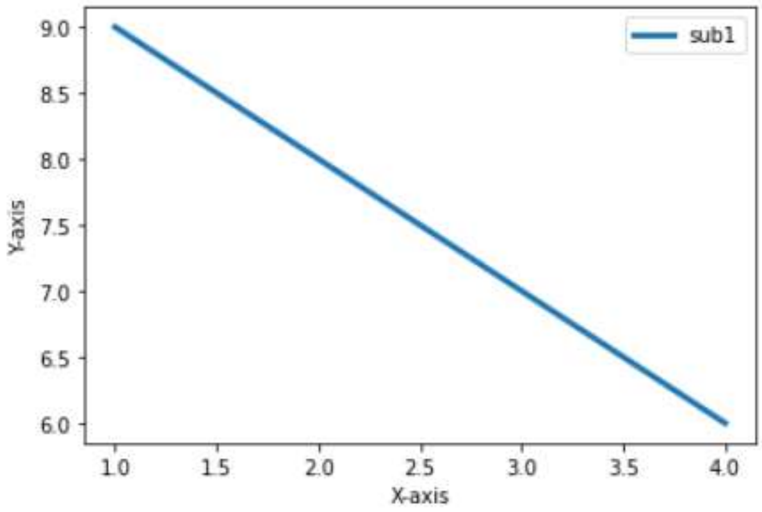
pl.xlabel("X-axis")

pl.ylabel("Y-axis")

pl.plot(x,y,linewidth=3, label="sub1")

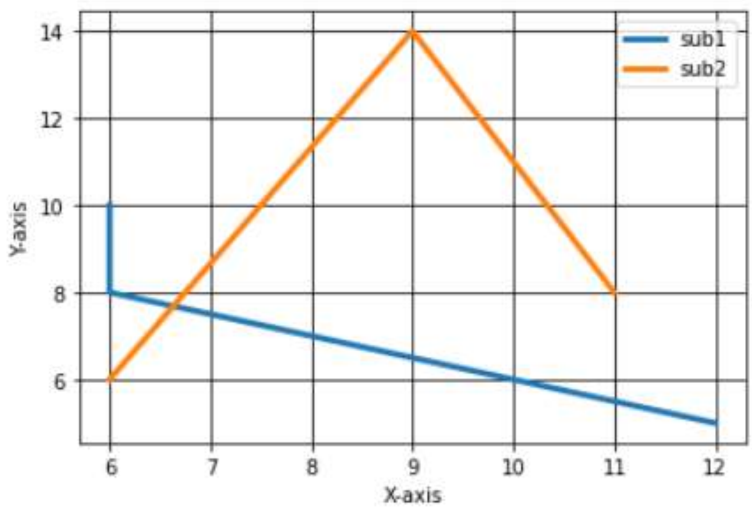
```
pl.legend()
pl.show()
```

Output:

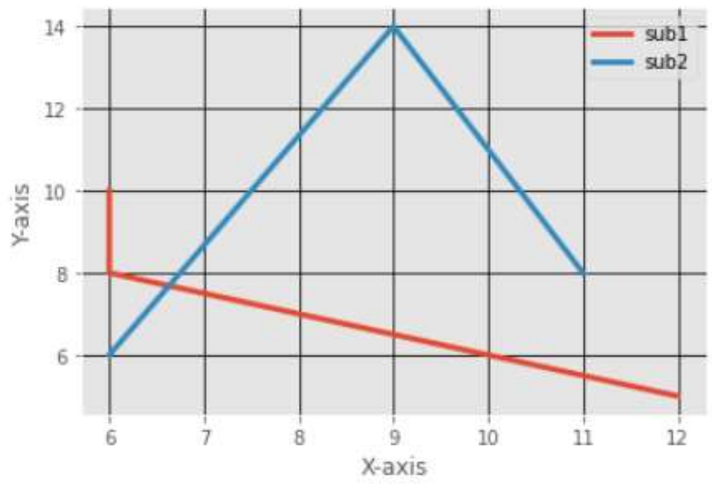



```
x2 = [6, 9, 11]
y = [5, 8, 10]
x = [12, 6, 6]
y2 = [6, 14, 8]

pl.xlabel("X-axis")
pl.ylabel("Y-axis")
pl.plot(x,y,linewidth=3, label="sub1")
pl.plot(x2,y2,linewidth=3, label="sub2")
pl.legend()
pl.show()
```

Output:




```
from matplotlib import style

style.use('ggplot')
pl.xlabel("X-axis")
pl.ylabel("Y-axis")
pl.plot(x,y,linewidth=3, label="sub1")
pl.plot(x2,y2,linewidth=3, label="sub2")
pl.legend()
pl.grid(True, color = 'k')
pl.show()
```



x = [1,2,3,4]

y = [9,8,7,6]

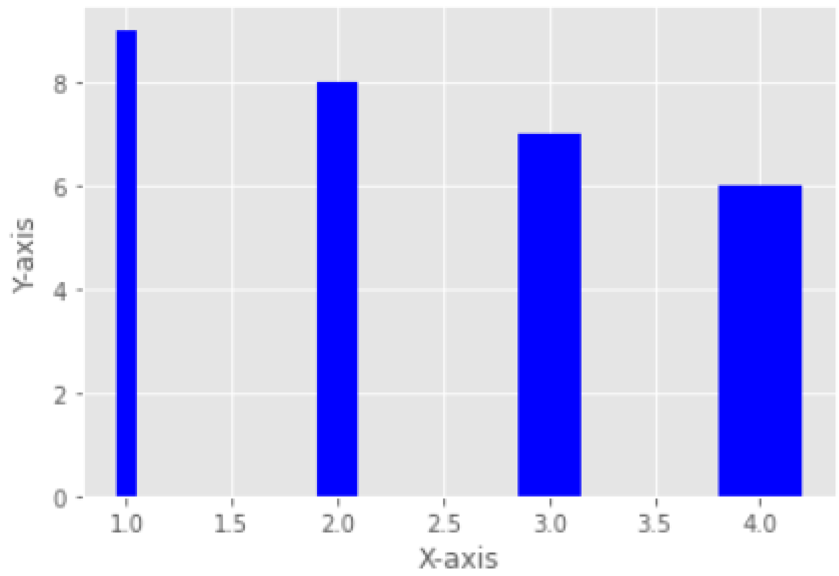
pl.xlabel("X-axis")

pl.ylabel("Y-axis")

pl.bar(x,y, width=[0.1,0.2,0.3,0.4], color = 'b')

pl.show()

Output:



x = [1,2,3,4]

y = [9,8,7,6]

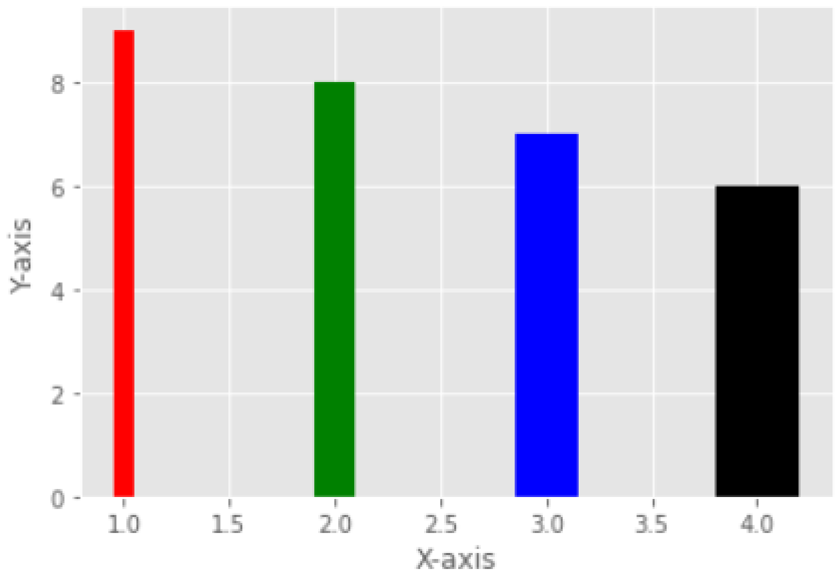
pl.xlabel("X-axis")

pl.ylabel("Y-axis")

pl.bar(x,y, width=[0.1,0.2,0.3,0.4], color = ['r','g','b','k'])

pl.show()

Output:



population_age = [50,30,60,76,34,29,90,100,9,8,5,23,65,34,21,54,87,98,43,56,45,99,12,10,44,35,101]

bins = [0,10,20,30,40,50,60,70,80,90,101]

pl.hist(population_age,bins,histtype = 'bar', rwidth=0.5)

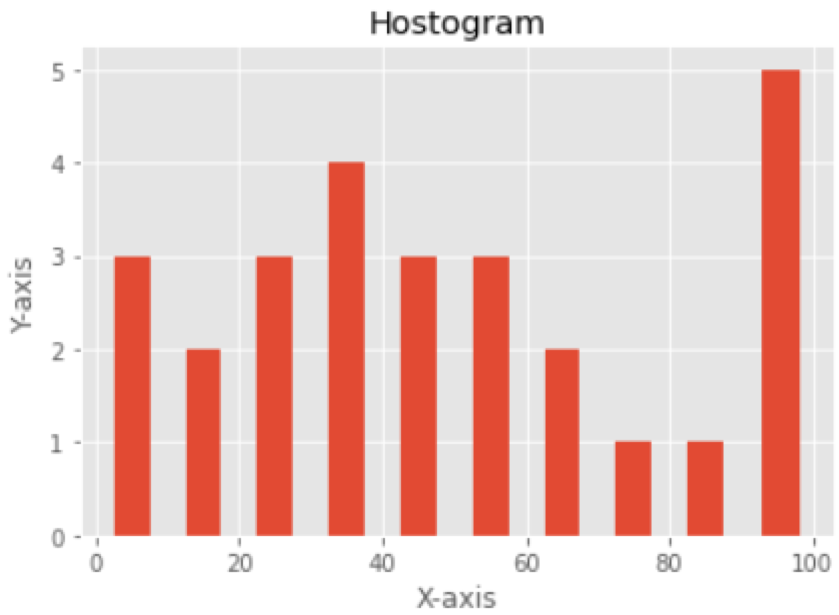
pl.xlabel("X-axis")

pl.ylabel("Y-axis")

pl.title("Hostogram")

pl.show()

Output:



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

y = [4,6,7,6,1,4]

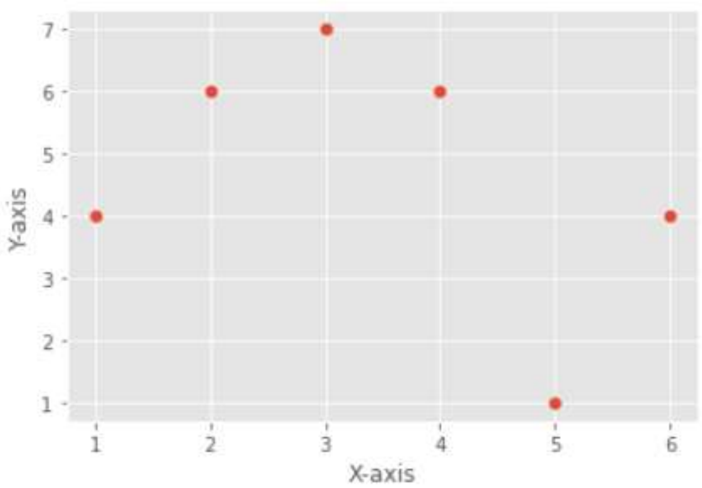
pl.scatter(x,y)

pl.xlabel("X-axis")

pl.ylabel("Y-axis")

pl.show()

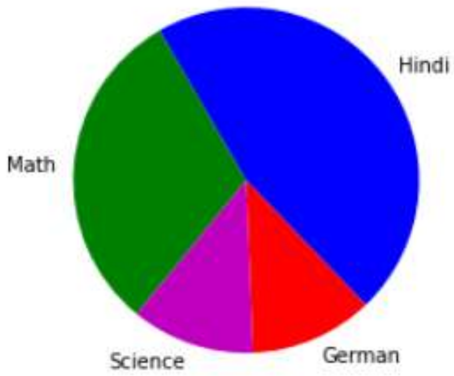
Output:



slices = [8,3,3,12]

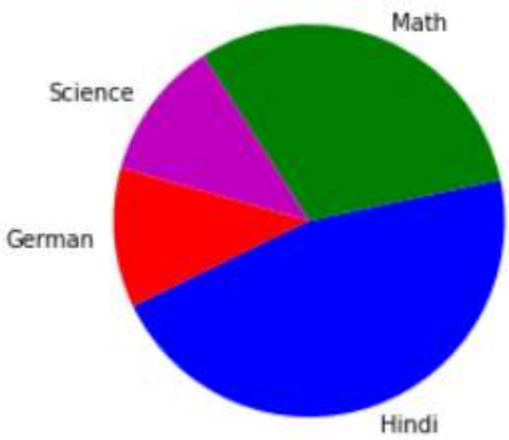
```
subjects = ['Math','Science','German','Hindi']

cols = ['g','m','r','b']

pl.pie(slices, labels = subjects, colors = cols, startangle = 120)

pl.show()
```

Output:




```
slices = [8,3,3,12]

subjects = ['Math','Science','German','Hindi']

cols = ['g','m','r','b']

pl.pie(slices, labels = subjects, colors = cols, startangle = 12)

pl.show()
```

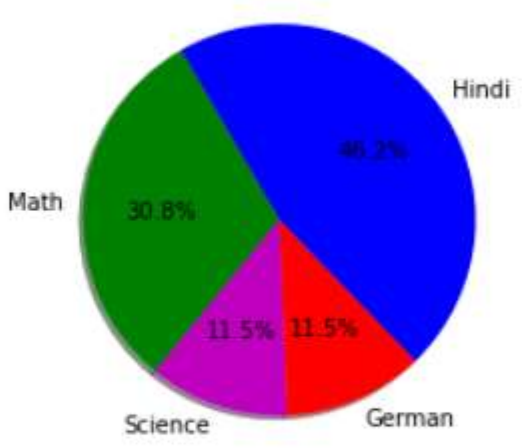
Output:




```
slices = [8,3,3,12]
```

```
subjects = ['Math','Science','German','Hindi']

cols = ['g','m','r','b']

pl.pie(slices, labels = subjects, colors = cols, startangle = 120, shadow = True, autopct = '%1.1f%%')

pl.show()
```

Output:




```
from numpy import poly1d

p = poly1d([9,8,7])

p
```

Output: 2

```
print(p)
```

Output: 9 x + 8 x + 7

```
print(p*p)
```

Output:

```
   4     3     2
81 x + 144 x + 190 x + 112 x + 49
```

```
print(2*p)
```

Output:

```
   2
18 x + 16 x + 14
```

```
p.r
```

Output: array([-0.44444444+0.7617394j, -0.44444444-0.7617394j])

```
import scipy.special as spl
import numpy as np
a = 8
spl.cbrt(a)
```

Output: 2.0

```
x = spl.sindg(90)
x
```

Output:1.0

```
y = spl.cosdg(0)
y
```

Output: 1.0

```
z = spl.tandg(45)
z
```

Output: 1.0

```
r = spl.perm(6,2)
r
```

Output: 30.0

```
x = 4
spl.exp10(x)
```

Output: 10000.0

```
spl.exp2(2)
```

Output: 4.0

```
f = lambda x : x**3
f(3)
```

Output: 27

```
u = lambda x,y : x**2 + y**2 + 2*x*y
u(4,5)
```

Output: 81

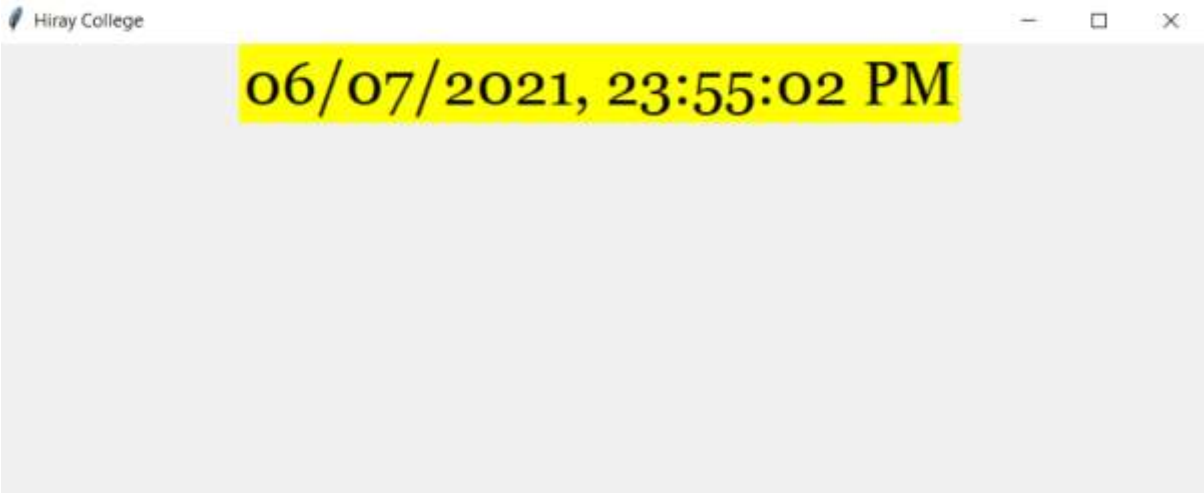
```
from scipy import integrate
f = lambda x: x**3
i = integrate.quad(f,1,2)
i
```

Output: (3.7500000000000004, 4.1633363423443377e-14)

```
from tkinter import *
import time as t
dc = Tk()
dc.title("Hiray College")
dc.geometry("800x300")


def time():
    d = t.strftime("%d/%m/%Y, %H:%M:%S %p")
    l.config(text = d)
    l.after(1000,time)
```

```
l = Label(dc, font=('Georgia',30), bg = "yellow", fg = "black")
l.pack()

time()
mainloop()
```

Output:




```
import math
print(math.radians(180))
```

Output: 3.141592653589793

Practical 4: Study of Supervised Learning:

a) Linear Regression
b) Logistic Regression
c) K Nearest Neighbour Algorithm

a) Linear Regression

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as pl

from sklearn import linear_model


df = pd.read_csv("dataset1.csv")


df
```

Output:

	area	price
0	2600	550000
1	3000	565000
2	3200	610000
3	3600	680000
4	4000	725000

```
df.shape
```

Output: (5, 2)

```
%matplotlib inline
```

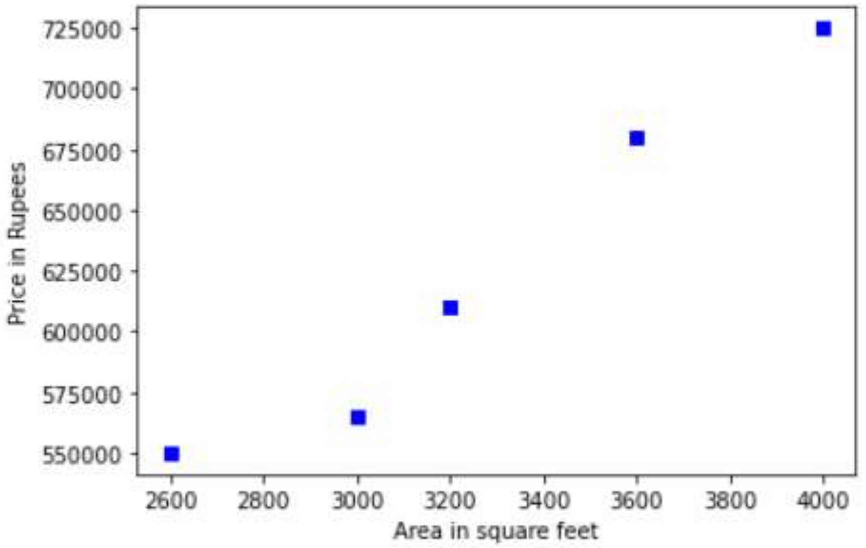
pl.xlabel('Area in square feet')

pl.ylabel('Price in Rupees')

pl.scatter(df.area,df.price, color='b', marker='s')

Output:

[24]: <matplotlib.collections.PathCollection at 0x13360421a90>



reg = linear_model.LinearRegression()

reg.fit(df[['area']],df.price)

Output: LinearRegression()

reg.coef_ #coef is m (slope)

Output: array([135.78767123])

reg.predict([[3300]])

Output: array([628715.75342466])

reg.intercept_ #intercept is b

Output: 180616.43835616432

y = m * x + b

135.78767123 * 3300 + 180616.43835616432

Output: 628715.7534151643

b) Logistic Regression

```
import numpy as np

import pandas as pd

from matplotlib import pyplot as pl


df1 = pd.read_csv("loan.csv")


df1.head()
```

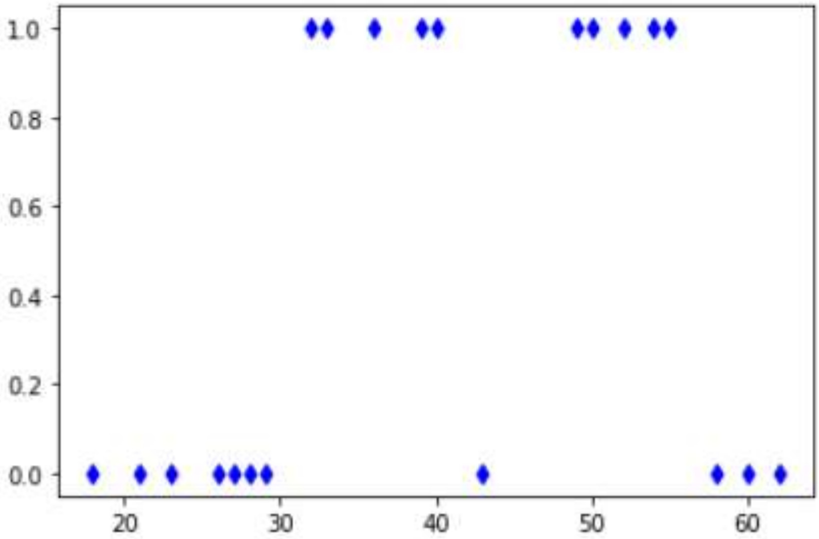
Output:

	age	bought_loan
0	23	0
1	18	0
2	55	1
3	43	0
4	36	1

```
pl.scatter(df1.age,df1.bought_loan, marker='d', color='b')
```

Output:

<matplotlib.collections.PathCollection at 0x25b5e99d160>



from sklearn.model_selection import train_test_split

x_train, x_test,y_train,y_test = train_test_split(df1[['age']], df1.bought_loan, train_size=0.9, shuffle=False)

x_test

Output:

	age
18	33
19	60
20	52

from sklearn.linear_model import LogisticRegression

logreg = LogisticRegression()

logreg.fit(x_train,y_train)

Output: LogisticRegression()

logreg.predict(x_test)

Output: array([0, 1, 1], dtype=int64)

c) K Nearest Neighbour Algorithm

```
import numpy as np

import pandas as pd

ds = pd.read_csv("iris.csv")

ds
```

Output:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...	...
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 6 columns

```
ds.iloc[: ,1:5]
```

Output:

	SepalLengthCm	SepalWidthCm	PetalLengthCm
0	5.1	3.5	1.4
1	4.9	3.0	1.4
2	4.7	3.2	1.3
3	4.6	3.1	1.5
4	5.0	3.6	1.4
...	...	...	...
145	6.7	3.0	5.2
146	6.3	2.5	5.0
147	6.5	3.0	5.2
148	6.2	3.4	5.4
149	5.9	3.0	5.1

150 rows × 3 columns

x = ds.iloc[:, 1:5].values

y = ds.iloc[:, 5].values

from sklearn.preprocessing import LabelEncoder   #0 and nclass-1

lblenc_y = LabelEncoder()

y = lblenc_y.fit_transform(y)

y

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.2)

from sklearn.neighbors import KNeighborsClassifier

knn_model = KNeighborsClassifier(n_neighbors=5)

knn_model.fit(x_train, y_train)

Output: KNeighborsClassifier()

y_predict = knn_model.predict(x_test)

from sklearn.metrics import confusion_matrix, classification_report

print(confusion_matrix(y_test,y_predict))

Output:

[[13 0 0]

 [0 7 2]

 [0 0 8]]

#accuracy

print(28/30)

Output: 0.9333333333333333

print(classification_report(y_test, y_predict))

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         9
           1       0.91      0.91      0.91        11
           2       0.90      0.90      0.90        10

    accuracy                           0.93        30
   macro avg       0.94      0.94      0.94        30
weighted avg       0.93      0.93      0.93        30
```

Practical 5: Study of Dimension Reduction:

a) Feature Scaling
b) Normalization (Feature Selection)

a) Feature Selection

```
import numpy as np

import pandas as pd

from sklearn.feature_selection import SelectKBest

from sklearn.feature_selection import chi2

df = pd.read_csv("FeatureSelection.csv")

df.head()
```

	age	weight	height	cholestrol	sugar	Target
0	35	70	150	233	250	1
1	56	75	100	250	300	0
2	67	68	180	204	260	0
3	72	60	170	236	450	1
4	39	77	190	354	220	1

```
X = df.iloc[:,0:5]

X.head()
```

	age	weight	height	cholestrol	sugar
0	35	70	150	233	250
1	56	75	100	250	300
2	67	68	180	204	260
3	72	60	170	236	450
4	39	77	190	354	220

```
y = df.iloc[:,-1]

y.head()
```

```
0    1
1    0
2    0
3    1
4    1
Name: Target, dtype: int64
```

```
bestfeat = SelectKBest(score_func=chi2, k=4)
fit = bestfeat.fit(X,y)
datascore = pd.DataFrame(fit.scores_)
datacol = pd.DataFrame(X.columns)
fscore = pd.concat([datacol, datascore], axis = 1)
fscore.columns = ["best", "score"]
fscore
```

	best	score
0	age	5.807560
1	weight	0.036697
2	height	1.763191
3	cholestrol	5.039769
4	sugar	0.024155

```
from sklearn.ensemble import ExtraTreesClassifier
import matplotlib.pyplot as pl
model= ExtraTreesClassifier()
model.fit(X,y)
```

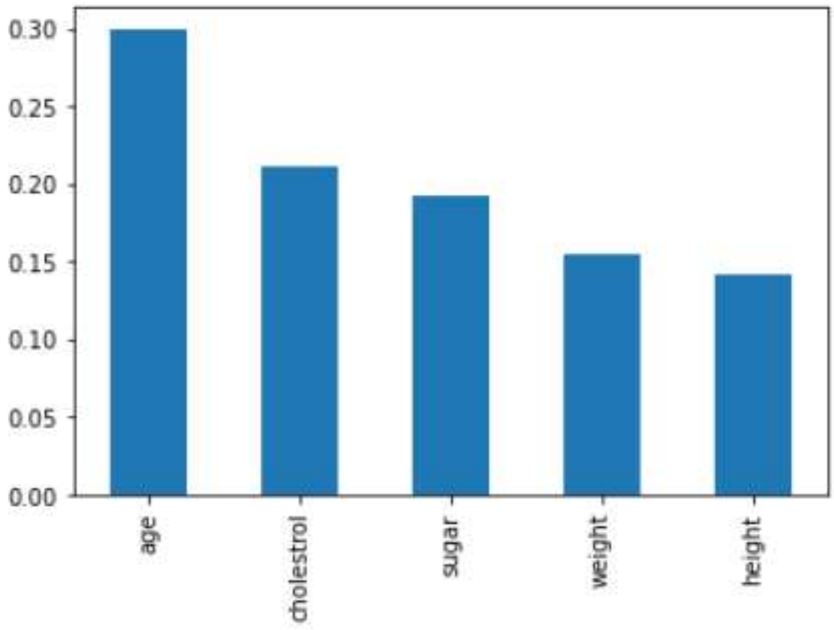
ExtraTreesClassifier()

```
model.feature_importances_
```

array([0.29935847, 0.15434854, 0.14141038, 0.21183862, 0.19304398])

```
featimport = pd.Series(model.feature_importances_, index=X.columns)
featimport.nlargest(5).plot(kind='bar')
```

pl.show()



 # Seaborn -----> Heatmap ---->

import seaborn as sns

corrmat = df.corr()

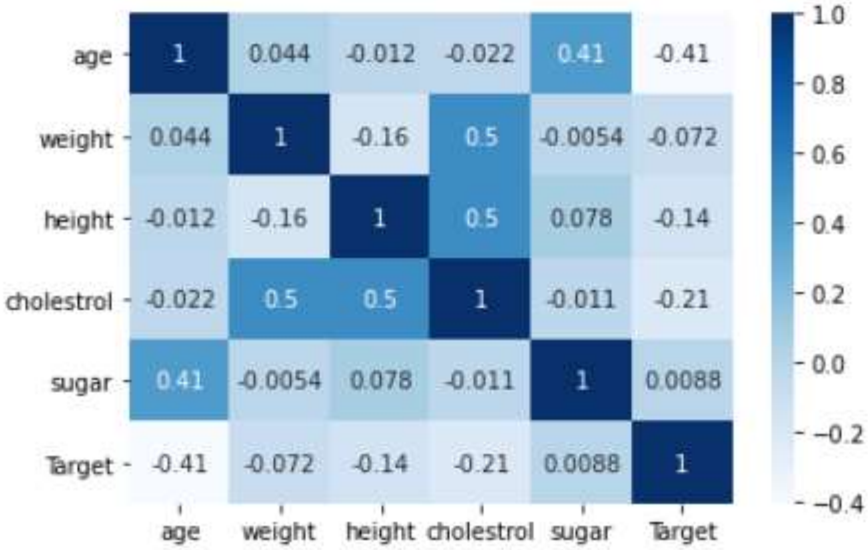
corrfeat = corrmat.index

pl.figure(figsize=(20,20))

<Figure size 1440x1440 with 0 Axes>

<Figure size 1440x1440 with 0 Axes>

a= sns.heatmap(df[corrfeat].corr(), annot=True, cmap="Blues")



age
weight
height
cholestrol
sugar
Target
1
0.044
-0.012
-0.022
0.41
-0.41
0.044
1
-0.16
0.5
-0.0054
-0.072
-0.012
-0.16
1
0.5
0.078
-0.14
-0.022
0.5
0.5
1
-0.011
-0.21
0.41
-0.0054
0.078
-0.011
1
0.0088
-0.41
-0.072
-0.14
-0.21
0.0088
1
age
weight
height
cholestrol
sugar
Target
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4

b) Normalization

```
import numpy as np

import pandas as pd

ds = pd.read_csv("FeatureSelection.csv")

ds.head()
```

	age	weight	height	cholestrol	sugar	Target
0	35	70	150	233	250	1
1	56	75	100	250	300	0
2	67	68	180	204	260	0
3	72	60	170	236	450	1
4	39	77	190	354	220	1

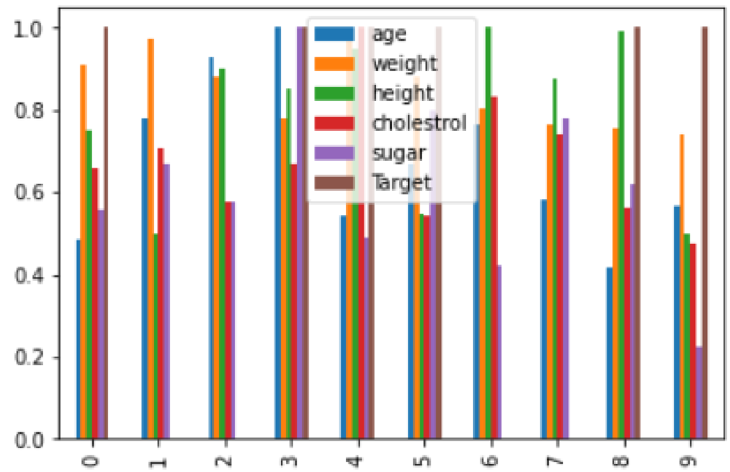
 # Simple feature scaling

```
for column in ds.columns:
    ds[column]=ds[column]/ds[column].abs().max()

ds.head()
```

	age	weight	height	cholestrol	sugar	Target
0	0.486111	0.909091	0.75	0.658192	0.555556	1.0
1	0.777778	0.974026	0.50	0.706215	0.666667	0.0
2	0.930556	0.883117	0.90	0.576271	0.577778	0.0
3	1.000000	0.779221	0.85	0.666667	1.000000	1.0
4	0.541667	1.000000	0.95	1.000000	0.488889	1.0

```
import matplotlib.pyplot as pl

ds.plot(kind='bar')
```

<AxesSubplot:>



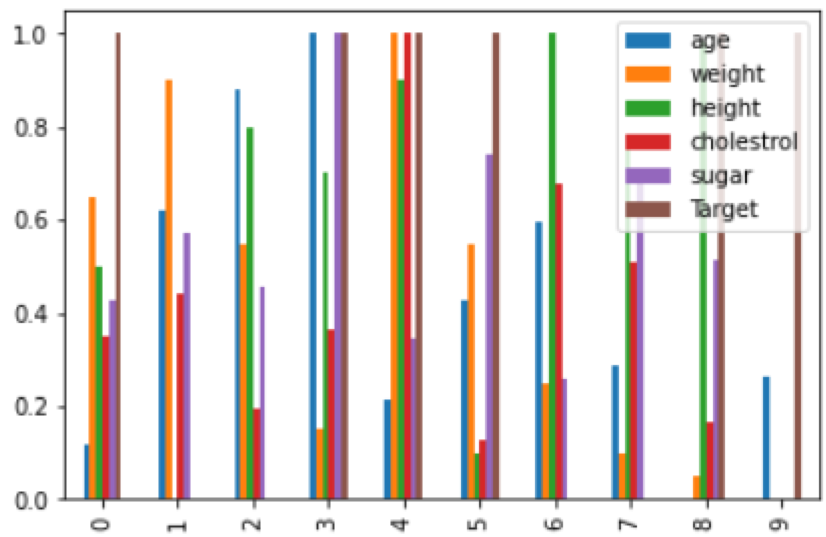
 # Min Max method xold=(xold-xmin)/(xmax-xmin)

```
ds1= ds.copy()
for column in ds1.columns:
  ds1[column]= (ds1[column] - ds1[column].min()) / (ds1[column].max() - ds1[column].min())
ds1.head()
```

	age	weight	height	cholestrol	sugar	Target
0	0.119048	0.65	0.5	0.349462	0.428571	1.0
1	0.619048	0.90	0.0	0.440860	0.571429	0.0
2	0.880952	0.55	0.8	0.193548	0.457143	0.0
3	1.000000	0.15	0.7	0.365591	1.000000	1.0
4	0.214286	1.00	0.9	1.000000	0.342857	1.0

```
ds1.plot(kind='bar')
```

<AxesSubplot:>



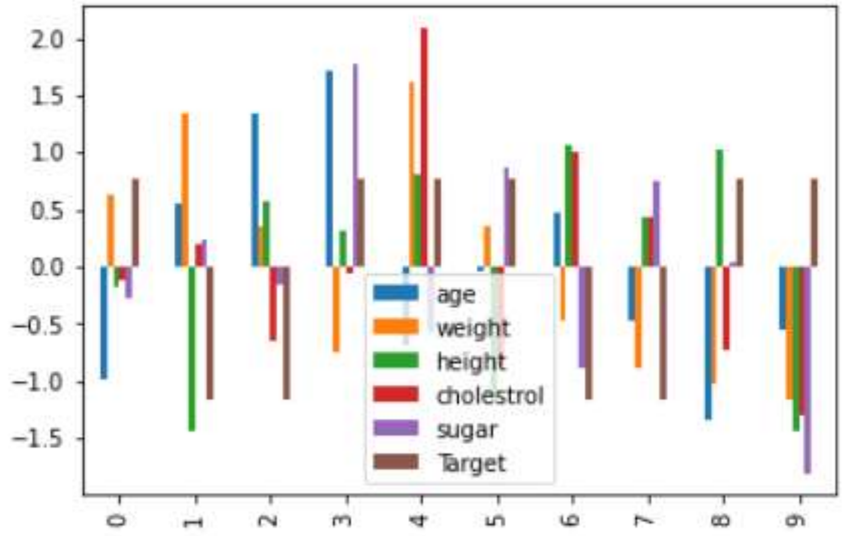
 # Standardization (Z score method or 0 mean)

```
ds2=ds.copy()
for column in ds2.columns:
   ds2[column]=(ds2[column] - ds2[column].mean()) / ds2[column].std()
ds2.head()
```

	age	weight	height	cholestrol	sugar	Target
0	-0.980397	0.637633	-0.182525	-0.114881	-0.266226	0.774597
1	0.544665	1.330712	-1.432694	0.195116	0.245747	-1.161895
2	1.343507	0.360401	0.567577	-0.643699	-0.163831	-1.161895
3	1.706617	-0.748525	0.317543	-0.060176	1.781663	0.774597
4	-0.689909	1.607944	0.817611	2.091567	-0.573409	0.774597

ds2.plot(kind='bar')

```
<AxesSubplot:>
```



Practical 6: Study of Principal Components Analysis

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as pl
ds = pd.read_csv("wine.csv")
ds.head()
```

	Alcohol	Malic.acid	Ash	Acl	Mg	Phenols	Flavanoids	Nonflavanoid.phenols	Proanth	Color.int	Hue	OD	Proline	Wine
0	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065	1
1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050	1
2	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185	1
3	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480	1
4	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735	1

```
X = ds.iloc[:,0:13]
X.head()
```

	Alcohol	Malic.acid	Ash	Acl	Mg	Phenols	Flavanoids	Nonflavanoid.phenols	Proanth	Color.int	Hue	OD	Proline
0	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065
1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050
2	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185
3	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480
4	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735

```
y = ds.iloc[:,-1]
y.head()
```

```
0    1
1    1
2    1
3    1
4    1
Name: Wine, dtype: int64
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.2)
```

 # We will extract any two columns as Principal Component

 # With the help of EigenValue

 # First we will standardize the data

```
from sklearn.preprocessing import StandardScaler
st = StandardScaler()
```

 # fit and transform

```
X_train = st.fit_transform(X_train)
X_train
```

```
array([[ 0.86085012, -0.83342199,  0.47848374, ..., -0.21754034,
         0.38175836,  1.84584136],
       [-0.73835419,  1.70964634,  1.29881334, ..., -0.08954008,
         0.73014966, -1.20811907],
       [-0.22168818,  0.46354285,  0.87081528, ..., -0.21754034,
        -0.62160859, -0.46122657],
       ...,
       [-1.47644849, -0.58759206, -1.73283953, ..., -0.00420658,
        -0.21747468, -1.04214296],
       [-1.15660763, -1.08772883,  0.51415024, ...,  1.57446322,
         0.17272357, -0.32844568],
       [ 1.27910356, -0.66388411, -0.30617936, ...,  0.63579469,
         1.55235313,  0.16948265]])
```

```
X_test = st.transform(X_test)
```

 # Applying PCA

```
from sklearn.decomposition import PCA
```

 # Check Eigenvalue of components

```
pca = PCA(n_components = 2)
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)
pca.explained_variance_ratio_
```

```
array([0.36138769, 0.1937306 ])
```

 # Sorted ..... Descending order

 # Now classification

```
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
LogisticRegression()
y_test
```

```
29     1
149    3
63     2
158    3
22     1
Name: Wine, dtype: int64
```

y_predict = logreg.predict(X_test)

y_predict

```
array([1, 3, 2, 3, 1, 1, 2, 2, 2, 1, 1, 3, 2, 3, 2, 1, 3, 3, 1, 1, 3, 2,
       1, 1, 2, 1, 1, 2, 2, 3, 3, 1, 2, 3, 1, 2], dtype=int64)
```

check with Confusion Metrix for Actual vs Predict

from sklearn.metrics import confusion_matrix

c = confusion_matrix(y_test, y_predict)

c

```
array([[13,  1,  0],
       [ 1, 11,  1],
       [ 0,  0,  9]], dtype=int64)
```

 # 3 category

 # Visualize 2 components (0th and 1st)

X_disp, y_disp = X_train, y_train

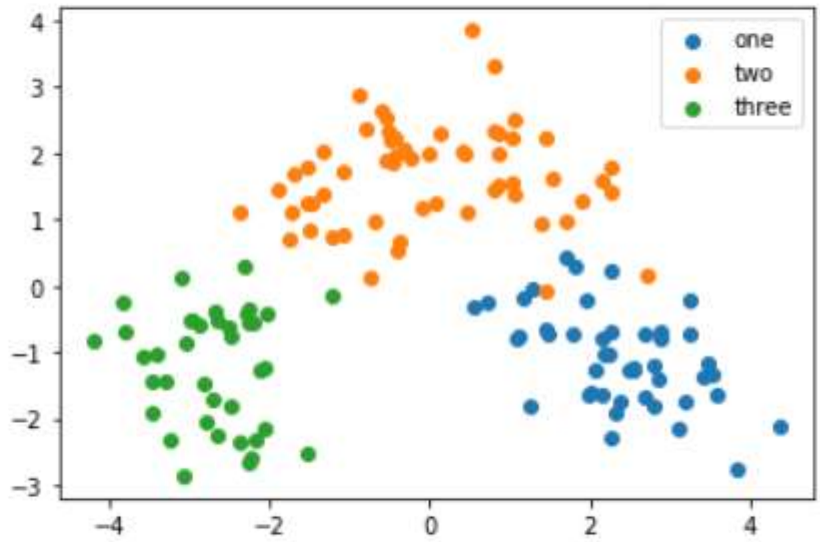
pl.scatter(X_disp[y_disp == 1,0], X_disp[y_disp == 1,1], label = 'one')

pl.scatter(X_disp[y_disp == 2,0], X_disp[y_disp == 2,1], label = 'two')

pl.scatter(X_disp[y_disp == 3,0], X_disp[y_disp == 3,1], label = 'three')

pl.legend()

pl.show()



one
two
three
4
3
2
1
0
−1
−2
−3
−4
−2
0
2
4

Practical 7: Implementation of K-Means Clustering

 # Find the exact / proper K

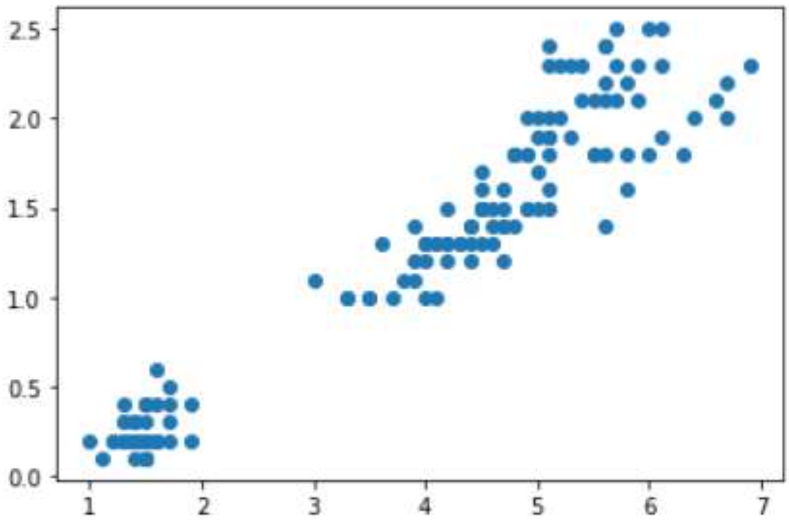
 # Elbow method

```
import pandas as pd

import numpy as np

ds = pd.read_csv("Iris.csv")

ds.head()
```

	PetalLength	PetalWidth	Species
0	1.4	0.2	Iris-setosa
1	1.4	0.2	Iris-setosa
2	1.3	0.2	Iris-setosa
3	1.5	0.2	Iris-setosa
4	1.4	0.2	Iris-setosa

```
from matplotlib import pyplot as pl

pl.scatter(ds['PetalLength'],ds['PetalWidth'])
```

```
<matplotlib.collections.PathCollection at 0x25124fdc3d0>
```




```
from sklearn.cluster import KMeans

kmean = KMeans(n_clusters = 3)

kmean
```

KMeans(n_clusters=3)

y_predict = kmean.fit_predict(ds[['PetalLength','PetalWidth']])

y_predict

```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2, 0, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

ds['cluster'] = y_predict

ds

	PetalLength	PetalWidth	Species	cluster
0	1.4	0.2	Iris-setosa	1
1	1.4	0.2	Iris-setosa	1
2	1.3	0.2	Iris-setosa	1
3	1.5	0.2	Iris-setosa	1
4	1.4	0.2	Iris-setosa	1
...	...	...	...	...
145	5.2	2.3	Iris-virginica	2
146	5.0	1.9	Iris-virginica	2
147	5.2	2.0	Iris-virginica	2
148	5.4	2.3	Iris-virginica	2
149	5.1	1.8	Iris-virginica	2

150 rows × 4 columns

kmean.cluster_centers_

```
array([[4.26923077, 1.34230769],
       [1.464     , 0.244     ],
       [5.59583333, 2.0375    ]])
```

ds1 = ds[ds.cluster == 0]

ds2 = ds[ds.cluster == 1]

ds3 = ds[ds.cluster == 2]

pl.scatter(ds1.PetalLength,ds1.PetalWidth, color = 'blue') pl.scatter(ds2.PetalLength,ds2.PetalWidth, color = 'red')

pl.scatter(ds3.PetalLength,ds3.PetalWidth, color = 'green')

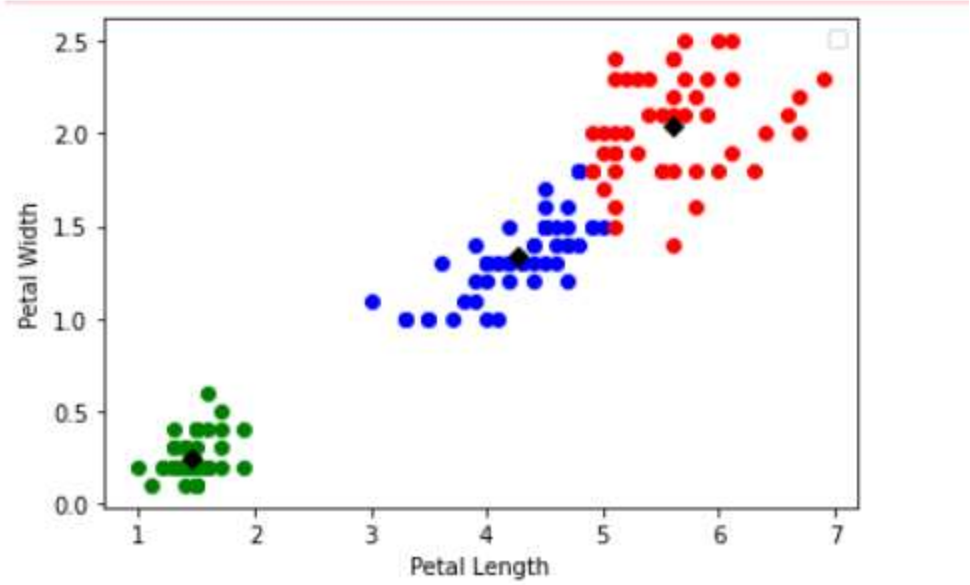
pl.scatter(kmean.cluster_centers_[:,0], kmean.cluster_centers_[:,1], color = 'black', marker='D')

pl.xlabel('Petal Length')

pl.ylabel('Petal Width')

pl.legend()

pl.show()



 # since values of x and y are mismatch

 # so we need to scale the values

from sklearn.preprocessing import MinMaxScaler

scl = MinMaxScaler()

scl.fit(ds[['PetalLength']])

ds['PetalLength'] = scl.transform(ds[['PetalLength']])

scl.fit(ds[['PetalWidth']])

ds['PetalWidth'] = scl.transform(ds[['PetalWidth']])

 # Applying KMean once again

kmean = KMeans(n_clusters = 3)

y_predict = kmean.fit_predict(ds[['PetalLength', 'PetalWidth']])

y_predict

```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2, 0, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 0, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

ds['cluster'] = y_predict

ds

	PetalLength	PetalWidth	Species	cluster
0	0.067797	0.041667	Iris-setosa	1
1	0.067797	0.041667	Iris-setosa	1
2	0.050847	0.041667	Iris-setosa	1
3	0.084746	0.041667	Iris-setosa	1
4	0.067797	0.041667	Iris-setosa	1
...	...	...	...	...
145	0.711864	0.916667	Iris-virginica	2
146	0.677966	0.750000	Iris-virginica	2
147	0.711864	0.791667	Iris-virginica	2
148	0.745763	0.916667	Iris-virginica	2
149	0.694915	0.708333	Iris-virginica	2

150 rows × 4 columns

ds1 = ds[ds.cluster == 0]

ds2 = ds[ds.cluster == 1]

ds3 = ds[ds.cluster == 2]

pl.scatter(ds1.PetalLength,ds1.PetalWidth, color = 'blue')

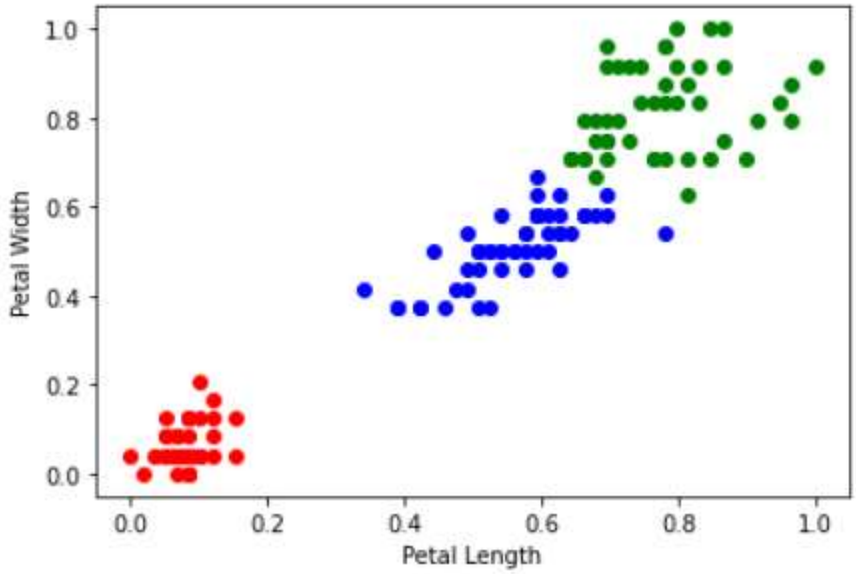
pl.scatter(ds2.PetalLength,ds2.PetalWidth, color = 'red')

pl.scatter(ds3.PetalLength,ds3.PetalWidth, color = 'green')

pl.xlabel('Petal Length')

pl.ylabel('Petal Width')

pl.show()

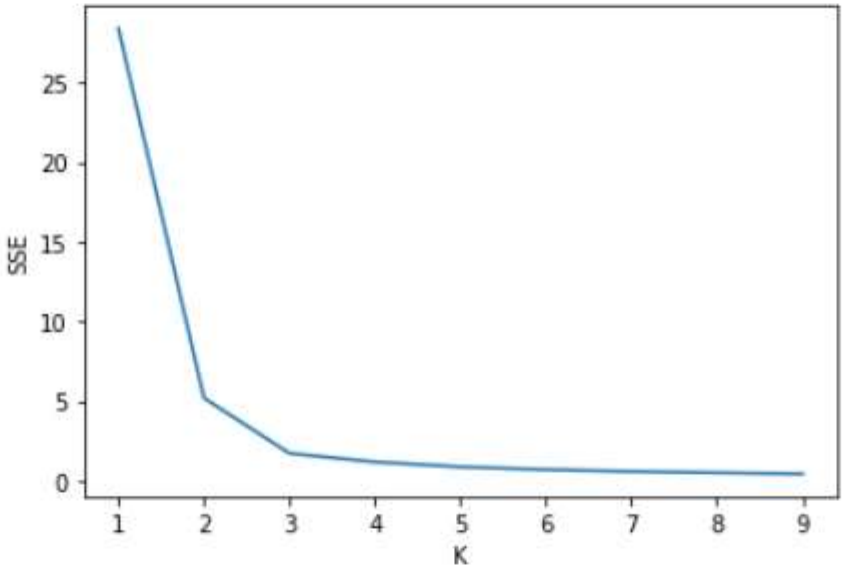



```
k_range = range(1,10)

sse = []

for k in k_range:

    kmean = KMeans(n_clusters = k)

    kmean.fit(ds[['PetalLength','PetalWidth']])

    sse.append(kmean.inertia_)

sse
```

```
[28.391514358368717,
 5.179687509974783,
 1.7050986081225123,
 1.1621031930971286,
 0.8570856553216398,
 0.6833274904190353,
 0.5683512655008139,
 0.48911635449076774,
 0.4155388630360096]
```

```
pl.xlabel('K')

pl.ylabel('SSE')

pl.plot(k_range, sse)
```

[<matplotlib.lines.Line2D at 0x1c46ba07d90>]



Practical 8: Study of Support Vector Machines (SVMs).

```
import pandas as pd
from sklearn.datasets import load_iris
iris = load_iris()
iris.feature_names
```

```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

```
ds = pd.DataFrame(iris.data, columns=iris.feature_names)
ds.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

 # Appending one column as target column

```
ds['target'] = iris.target
ds.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

 # 0 ----> Setosa 1-->Versicolor 2---> Virginica

```
iris.target_names
```

```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

#Want to see the number of rows for each flower type

```
ds[ds.target==2]
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
100	6.3	3.3	6.0	2.5	2
101	5.8	2.7	5.1	1.9	2
102	7.1	3.0	5.9	2.1	2
103	6.3	2.9	5.6	1.8	2
104	6.5	3.0	5.8	2.2	2
105	7.6	3.0	6.6	2.1	2
106	4.9	2.5	4.5	1.7	2
…					
…					
…					
…					
…					
146	6.3	2.5	5.0	1.9	2
147	6.5	3.0	5.2	2.0	2
148	6.2	3.4	5.4	2.3	2
149	5.9	3.0	5.1	1.8	2

 # Want to append flower name column ---> based on index (0, 1, 2) i.e Setosa and all

```
ds['fname'] = ds.target.apply(lambda x: iris.target_names[x])
ds.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	fname
0	5.1	3.5	1.4	0.2	0	setosa
1	4.9	3.0	1.4	0.2	0	setosa
2	4.7	3.2	1.3	0.2	0	setosa
3	4.6	3.1	1.5	0.2	0	setosa
4	5.0	3.6	1.4	0.2	0	setosa

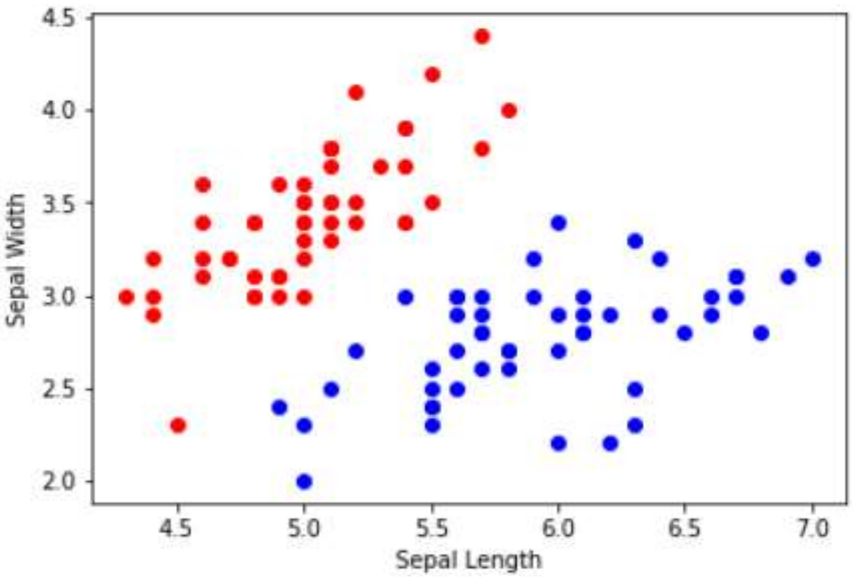
```
from matplotlib import pyplot as pl
```

 # Create 3 DF for three flowers

```
ds1 = ds[ds.target==0]

ds2 = ds[ds.target==1]

ds3 = ds[ds.target==2]
```

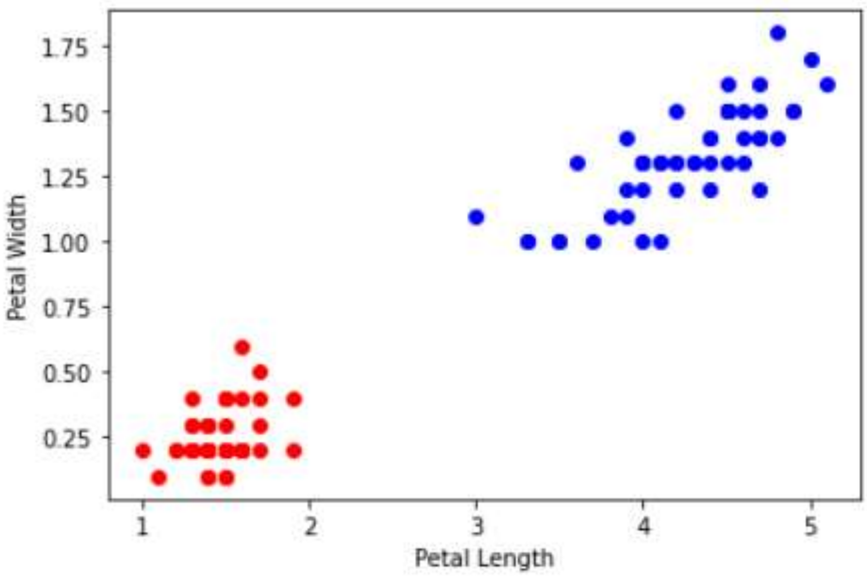
```
pl.scatter(ds1['sepal length (cm)'], ds1['sepal width (cm)'], color = 'red', marker = 'o')

pl.scatter(ds2['sepal length (cm)'], ds2['sepal width (cm)'], color = 'blue', marker = 'o')

pl.xlabel('Sepal Length')

pl.ylabel('Sepal Width')
```

```
Text(0, 0.5, 'Sepal Width')
```




```
pl.scatter(ds1['petal length (cm)'], ds1['petal width (cm)'], color = 'red', marker = 'o')

pl.scatter(ds2['petal length (cm)'], ds2['petal width (cm)'], color = 'blue', marker = 'o')

pl.xlabel('Petal Length')

pl.ylabel('Petal Width')
```

```
Text(0, 0.5, 'Petal Width')
```



from sklearn.model_selection import train_test_split

Remove fname column

X = ds.drop(['target','fname'], axis = 'columns')

X.head()

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

y = ds['target']

y.head()

```
0    0
1    0
2    0
3    0
4    0
Name: target, dtype: int32
```

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=False)

from sklearn.svm import SVC

```
svmodel = SVC(kernel='linear')
svmodel.fit(X_train, y_train)
```

```
SVC(kernel='linear')
```

Accuracy of model

```
svmodel.score(X_train, y_train)
```

```
0.9916666666666667
```

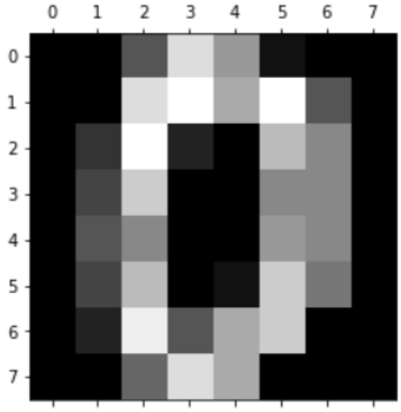
Practical 9. Study of Bagging Algorithm:

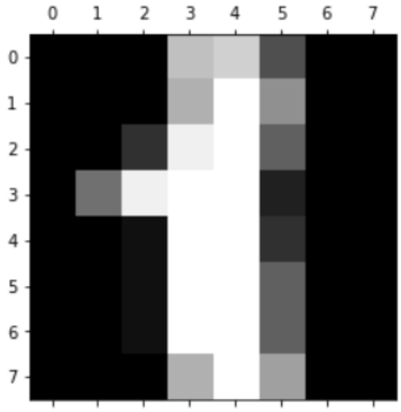
a) Random Forest
b) Decision Tree Classifier
c) Bagging Classifier

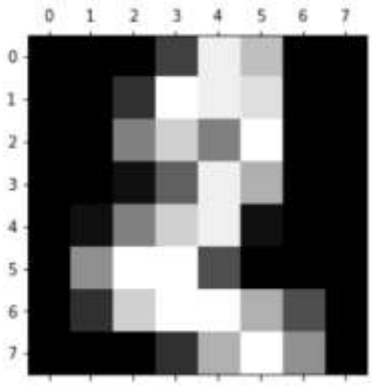
a) Random Forest

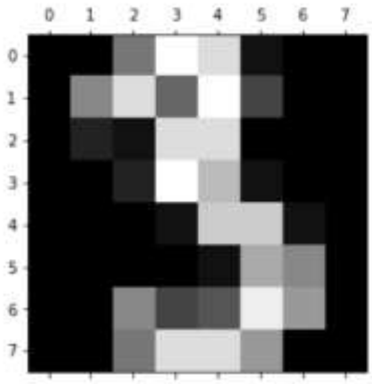
```
import pandas as pd

import numpy as np

from sklearn.datasets import load_digits


dig = load_digits()


import matplotlib.pyplot as pl



pl.gray()

for i in range(4):

   pl.matshow(dig.images[i])
```

<Figure size 432x288 with 0 Axes>









dig.data[:3]

```
array([[ 0.,  0.,  5., 13.,  9.,  1.,  0.,  0.,  0.,  0., 13., 15., 10.,
        15.,  5.,  0.,  0.,  3., 15.,  2.,  0., 11.,  8.,  0.,  0.,  4.,
        12.,  0.,  0.,  8.,  8.,  0.,  0.,  5.,  8.,  0.,  0.,  9.,  8.,
         0.,  0.,  4., 11.,  0.,  1., 12.,  7.,  0.,  0.,  2., 14.,  5.,
        10., 12.,  0.,  0.,  0.,  0.,  6., 13., 10.,  0.,  0.,  0.],
       [ 0.,  0.,  0., 12., 13.,  5.,  0.,  0.,  0.,  0.,  0., 11., 16.,
         9.,  0.,  0.,  0.,  0.,  3., 15., 16.,  6.,  0.,  0.,  0.,  7.,
        15., 16., 16.,  2.,  0.,  0.,  0.,  0.,  1., 16., 16.,  3.,  0.,
         0.,  0.,  0.,  1., 16., 16.,  6.,  0.,  0.,  0.,  0.,  1., 16.,
        16.,  6.,  0.,  0.,  0.,  0.,  0., 11., 16., 10.,  0.,  0.],
       [ 0.,  0.,  0.,  4., 15., 12.,  0.,  0.,  0.,  0.,  3., 16., 15.,
        14.,  0.,  0.,  0.,  0.,  8., 13.,  8., 16.,  0.,  0.,  0.,  0.,
         1.,  6., 15., 11.,  0.,  0.,  0.,  1.,  8., 13., 15.,  1.,  0.,
         0.,  0.,  9., 16., 16.,  5.,  0.,  0.,  0.,  0.,  3., 13., 16.,
        16., 11.,  5.,  0.,  0.,  0.,  0.,  3., 11., 16.,  9.,  0.]])
```

ds = pd.DataFrame(dig.data)

ds.head()

	0	1	2	3	4	5	6	7	8	9	...	54	55	56	57	58	59	60	61	62	63
0	0.0	0.0	5.0	13.0	9.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	6.0	13.0	10.0	0.0	0.0	0.0
1	0.0	0.0	0.0	12.0	13.0	5.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	11.0	16.0	10.0	0.0	0.0
2	0.0	0.0	0.0	4.0	15.0	12.0	0.0	0.0	0.0	0.0	...	5.0	0.0	0.0	0.0	0.0	3.0	11.0	16.0	9.0	0.0
3	0.0	0.0	7.0	15.0	13.0	1.0	0.0	0.0	0.0	8.0	...	9.0	0.0	0.0	0.0	7.0	13.0	13.0	9.0	0.0	0.0
4	0.0	0.0	0.0	1.0	11.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	2.0	16.0	4.0	0.0	0.0

5 rows × 64 columns

ds['target'] = dig.target

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(ds.drop(['target'], axis = 'columns'), dig.target, test_size=0.2)

y_test

```
array([9, 9, 4, 6, 0, 0, 3, 6, 3, 8, 9, 7, 3, 9, 5, 4, 1, 6, 7, 6, 7, 1,
       5, 0, 6, 1, 5, 6, 0, 2, 6, 3, 2, 9, 6, 7, 2, 4, 1, 5, 4, 5, 3, 4,
       7, 2, 8, 6, 1, 9, 5, 9, 6, 7, 5, 8, 5, 5, 2, 3, 7, 1, 3, 5, 5, 6,
       5, 1, 6, 1, 4, 0, 5, 6, 3, 3, 1, 7, 3, 3, 3, 3, 2, 4, 4, 4, 4, 8,
       3, 5, 3, 0, 3, 7, 9, 0, 3, 8, 5, 2, 4, 9, 7, 6, 6, 5, 4, 6, 3, 2,
       5, 8, 4, 6, 9, 3, 5, 2, 3, 1, 4, 6, 3, 3, 4, 4, 1, 6, 1, 5, 4, 7,
       8, 1, 1, 3, 5, 1, 6, 7, 9, 4, 3, 7, 9, 0, 8, 8, 8, 3, 6, 2, 4, 2,
       8, 6, 8, 3, 2, 7, 0, 0, 4, 5, 3, 6, 8, 4, 5, 0, 5, 4, 5, 8, 0, 2,
       6, 8, 8, 9, 9, 5, 2, 4, 8, 4, 7, 8, 6, 9, 3, 4, 9, 1, 9, 0, 7, 4,
       6, 7, 2, 9, 9, 1, 3, 7, 5, 8, 2, 2, 2, 2, 6, 8, 4, 7, 1, 1, 1, 6,
       4, 9, 2, 8, 4, 4, 8, 5, 8, 7, 2, 9, 6, 6, 3, 2, 5, 6, 7, 0, 1, 7,
       4, 7, 7, 7, 4, 6, 2, 0, 2, 3, 5, 5, 1, 2, 9, 5, 4, 4, 4, 4, 8, 8,
       8, 0, 1, 0, 2, 7, 8, 3, 9, 6, 8, 5, 2, 8, 0, 4, 0, 8, 9, 3, 2, 6,
       6, 3, 2, 9, 1, 0, 1, 6, 9, 1, 4, 8, 3, 8, 5, 9, 0, 5, 1, 8, 0, 3,
       6, 9, 6, 7, 5, 4, 4, 5, 1, 1, 9, 1, 0, 3, 4, 4, 7, 7, 5, 0, 9, 5,
       5, 1, 7, 5, 0, 8, 7, 9, 9, 6, 1, 3, 5, 1, 2, 9, 4, 3, 0, 5, 8, 3,
       1, 4, 0, 0, 5, 3, 0, 6])
```

from sklearn.ensemble import RandomForestClassifier rfmodel = RandomForestClassifier(n_estimators = 30)

rfmodel.fit(X_train, y_train)

RandomForestClassifier(n_estimators=30)

rfmodel.score(X_test, y_test)

0.9638888888888889

 # Draw heatmap and confusion metrix

y_pred = rfmodel.predict(X_test)

y_pred

```
array([9, 9, 4, 6, 0, 0, 3, 6, 3, 8, 9, 7, 3, 9, 5, 4, 1, 6, 7, 6, 7, 1,
       5, 0, 6, 1, 5, 6, 0, 2, 6, 3, 2, 9, 6, 7, 2, 4, 1, 5, 4, 5, 3, 4,
       7, 2, 8, 6, 1, 9, 5, 9, 6, 7, 5, 8, 5, 5, 2, 3, 7, 1, 3, 5, 5, 6,
       5, 1, 6, 1, 4, 0, 5, 6, 3, 3, 1, 7, 7, 3, 3, 3, 2, 9, 4, 9, 4, 8,
       3, 5, 3, 0, 3, 7, 9, 0, 3, 8, 5, 2, 4, 9, 7, 6, 6, 5, 4, 6, 3, 2,
       5, 8, 4, 6, 9, 3, 5, 2, 3, 1, 4, 6, 3, 3, 4, 4, 1, 6, 1, 5, 4, 7,
       8, 1, 1, 3, 5, 1, 6, 7, 9, 4, 3, 7, 9, 0, 8, 8, 8, 3, 6, 2, 4, 2,
       8, 6, 8, 9, 2, 7, 0, 0, 4, 5, 3, 6, 8, 4, 5, 0, 5, 4, 5, 8, 0, 2,
       6, 8, 8, 9, 9, 5, 2, 4, 8, 5, 7, 8, 6, 9, 3, 4, 9, 1, 9, 0, 7, 4,
       6, 7, 2, 9, 9, 1, 3, 7, 5, 1, 2, 2, 2, 2, 6, 3, 4, 7, 1, 1, 1, 6,
       4, 9, 2, 8, 4, 4, 2, 5, 8, 7, 2, 9, 6, 6, 3, 2, 5, 6, 7, 0, 1, 7,
       4, 7, 7, 7, 4, 6, 2, 0, 2, 8, 5, 5, 1, 2, 9, 5, 4, 4, 4, 4, 8, 8,
       7, 0, 1, 0, 2, 7, 8, 3, 9, 6, 8, 5, 2, 8, 0, 4, 0, 8, 5, 3, 2, 6,
       6, 3, 2, 9, 1, 0, 1, 6, 9, 1, 4, 8, 3, 8, 5, 9, 0, 5, 1, 8, 0, 3,
       6, 9, 6, 7, 6, 4, 4, 5, 1, 1, 9, 1, 0, 3, 4, 4, 7, 7, 5, 0, 9, 5,
       5, 1, 7, 5, 0, 8, 7, 3, 9, 6, 1, 3, 5, 1, 2, 9, 4, 3, 0, 5, 8, 3,
       1, 4, 0, 0, 5, 3, 0, 6])
```

from sklearn.metrics import confusion_matrix

cf_matrix = confusion_matrix(y_test, y_pred)

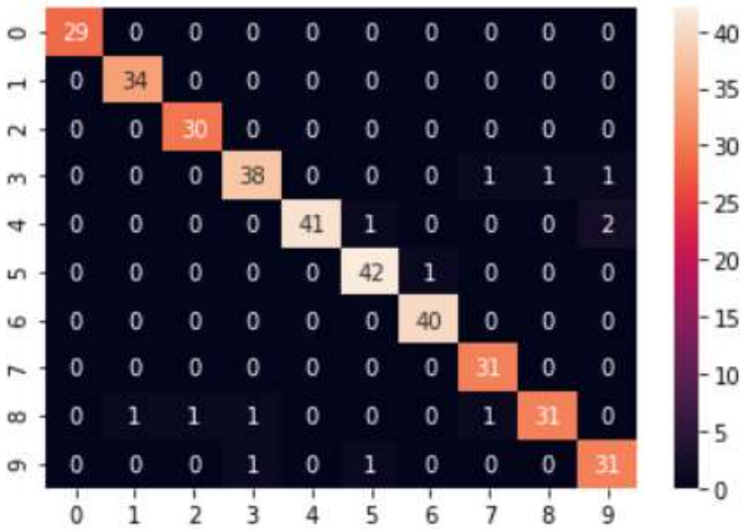
cf_matrix

```
array([[29,  0,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 0, 34,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0, 30,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0, 38,  0,  0,  0,  1,  1,  1],
       [ 0,  0,  0,  0, 41,  1,  0,  0,  0,  2],
       [ 0,  0,  0,  0,  0, 42,  1,  0,  0,  0],
       [ 0,  0,  0,  0,  0,  0, 40,  0,  0,  0],
       [ 0,  0,  0,  0,  0,  0,  0, 31,  0,  0],
       [ 0,  1,  1,  1,  0,  0,  0,  1, 31,  0],
       [ 0,  0,  0,  1,  0,  1,  0,  0,  0, 31]], dtype=int64)
```

import seaborn as sns

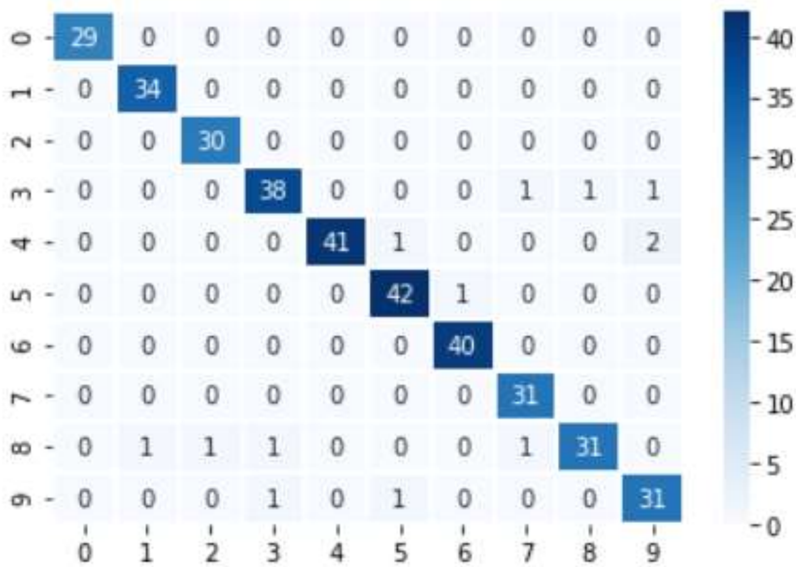
sns.heatmap(cf_matrix, annot = True)

<AxesSubplot:>



sns.heatmap(cf_matrix, annot = True, cmap = 'Blues', linewidths = 1)

<AxesSubplot:>



b) Decision Tree classifier

import pandas as pd

ds = pd.read_csv("IrisNew.csv")

ds.head()

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

X = ds.iloc[:,1:5]

y = ds.iloc[:,5]

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3) # 70% training and 30% test

from sklearn.tree import DecisionTreeClassifier

dt_classifier = DecisionTreeClassifier()

dt_classifier.fit(X_train, y_train)

DecisionTreeClassifier()

y_pred = dt_classifier.predict(X_test)

from sklearn.metrics import confusion_matrix, accuracy_score

cf = confusion_matrix(y_test, y_pred)

cf

```
array([[14,  0,  0],
       [ 0, 17,  1],
       [ 0,  3, 10]], dtype=int64)
```

ac = accuracy_score(y_test, y_pred)

ac

0.9111111111111111

c) Bagging classifier

```
import pandas as pd

ds = pd.read_csv("IrisNew.csv")

ds.head()
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

```
X = ds.iloc[:,1:5]

y = ds.iloc[:,5]

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3) # 70% training and 30% test

from sklearn.ensemble import BaggingClassifier

bagclassifier = BaggingClassifier()

bagclassifier.fit(X_train, y_train)
```

BaggingClassifier()

```
y_pred = bagclassifier.predict(X_test)

from sklearn.metrics import confusion_matrix, accuracy_score

bag_cs = confusion_matrix(y_test, y_pred)

bag_cs
```

```
array([[15,  0,  0],
       [ 0, 13,  1],
       [ 0,  2, 14]], dtype=int64)
```

```
bag_ac = accuracy_score(y_test, y_pred)
```

bag_ac

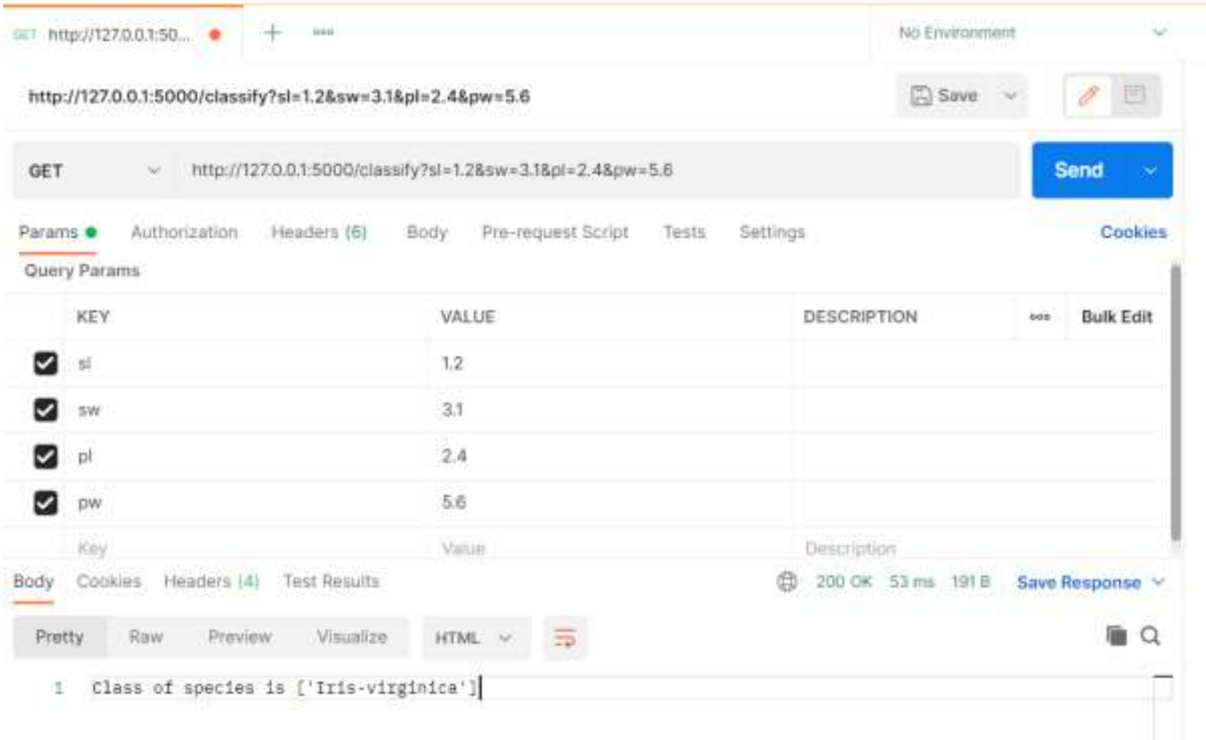
0.9333333333333333

```
import pickle
with open('bagModelIris.pkl','wb') as file:
    pickle.dump(bagclassifier, file)
import flask
from flask import Flask, request
model_bagging = pickle.load(open('bagModelIris.pkl','rb'))
app = Flask(__name__)
# Two parts (base address + route address)
@app.route('/', methods = ['GET','POST'])
def main():
    return "Begging Flask API Development"
@app.route('/classify', methods = ['GET'])
def classify():
    if flask.request.method == 'GET':
        SepalLengthCm = request.args.get('sl')
        SepalWidthCm = request.args.get('sw')
        PetalLengthCm = request.args.get('pl')
        PetalWidthCm = request.args.get('pw')

        prediction = model_bagging.predict([[SepalLengthCm, SepalWidthCm, PetalLengthCm, PetalWidthCm]])

        return 'Class of species is '+str(prediction)

if __name__ == '__main__':
```

```
app.run()
```

```
 * Serving Flask app "__main__" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [15/Aug/2021 22:55:00] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [15/Aug/2021 22:56:36] "GET /classify?sl=1.2&sw=3.1&pl=2.4&pw=5.6 HTTP/1.1" 200 -
```

Testing using Postman



Practical 10: Study of Boosting Algorithms:

a) AdaBoost
b) Stochastic Gradient Boosting
c) Voting Ensemble

a) AdaBoost

```
import pandas as pd

ds = pd.read_csv(r'addsdataset.csv')

# X is for input, y for output

X = ds.iloc[:,[2,3]].values

y = ds.iloc[:, 4].values


from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0,
shuffle = False)


# Features scaling

from sklearn.preprocessing import StandardScaler

sd = StandardScaler()

X_train = sd.fit_transform(X_train)

X_test = sd.transform(X_test)


from sklearn.ensemble import AdaBoostClassifier

classifier = AdaBoostClassifier()

classifier.fit(X_train, y_train)
```

AdaBoostClassifier()

y_pred = classifier.predict(X_test)

y_test

```
array([1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1,
       0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1,
       1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1,
       1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0,
       1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1], dtype=int64)
```

y_pred

```
array([1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1,
       0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0,
       1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1,
       0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0,
       1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1], dtype=int64)
```

from sklearn.metrics import confusion_matrix, accuracy_score

ac = accuracy_score(y_test, y_pred)

ac

0.82

 # Improvement with Random Forest Algorithm

from sklearn.ensemble import RandomForestClassifier

RF = RandomForestClassifier(max_depth = 2, random_state = 0)

classifierNew = AdaBoostClassifier(base_estimator = RF, n_estimators = 100, learning_rate = 0.01, random_state = 0)

classifierNew.fit(X_train, y_train)

```
AdaBoostClassifier(base_estimator=RandomForestClassifier(max_depth=2,
                                                         random_state=0),
                   learning_rate=0.01, n_estimators=100, random_state=0)
```

y_pred = classifierNew.predict(X_test)

ac = accuracy_score(y_test, y_pred)

ac

0.94

 # Deployment

 # In ANACONDA PROMT type below command

 # conda install -c anaconda flask

 # FROM GOOGLE INSTALL POSTMAN

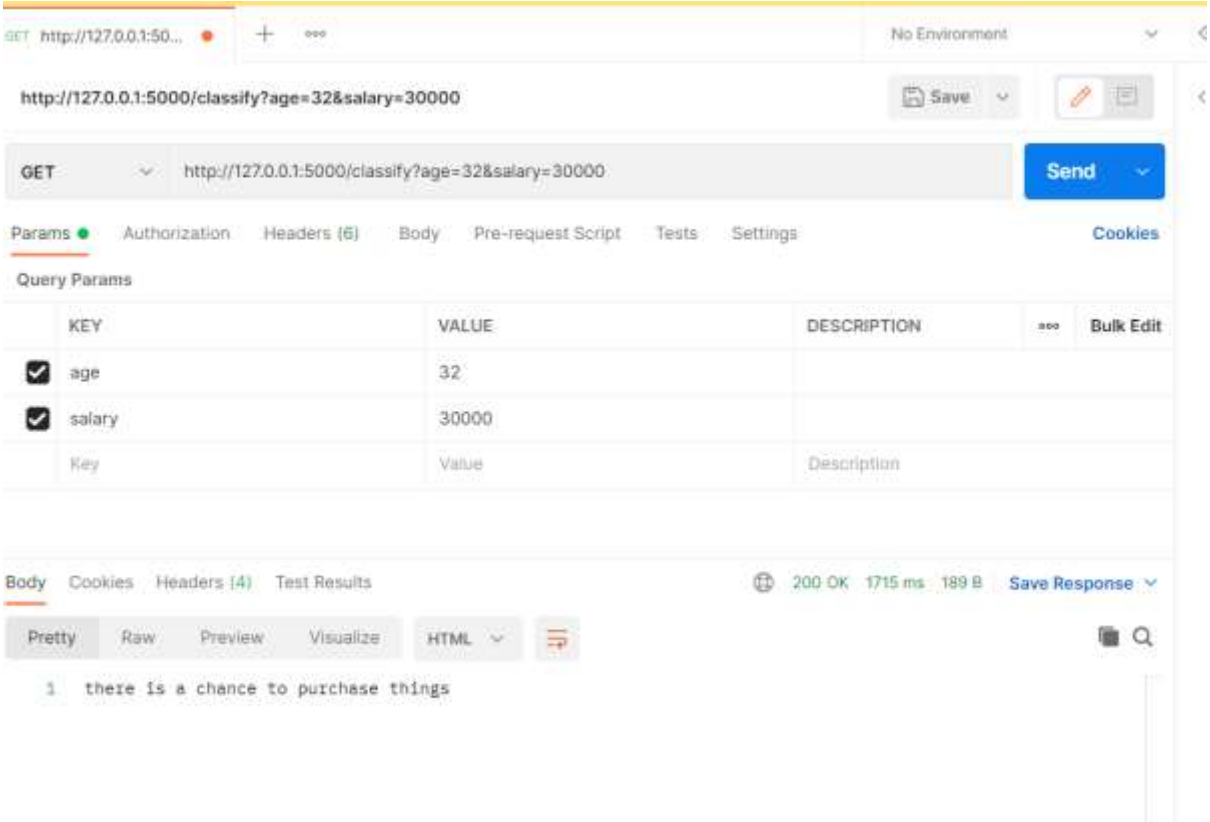
 # Creating pkl file

```
with open('model.pkl','wb') as file:
    pickle.dump(classifier, file)
with open('modelNew.pkl','wb') as file:
    pickle.dump(classifierNew, file)

import flask
from flask import Flask, request
import pickle

model_adaboost = pickle.load(open('modelNew.pkl', 'rb'))

app = Flask(__name__)

#Get method -> Read / Retrieve

@app.route('/', methods = ['GET', 'POST'])
def main():
    return "Ada boost with flask"
```

```
@app.route('/classify', methods = ['GET'])

def classify():

    if flask.request.method == 'GET':

        Age = request.args.get('age') # we will call the data from API using Postman

        EstimatedSalary = request.args.get('salary')

        prediction = model_adaboost.predict([[Age, EstimatedSalary]])

        print(prediction)


        if prediction == 1:

            return "there is a chance to purchase things"

        else:

            return "sorry, no chance"

    else:

        return "Select GET method"


if __name__ == '__main__':

    app.run()
```

```
 * Serving Flask app "__main__" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [22/Jul/2021 21:45:11] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [22/Jul/2021 21:45:43] "GET /classify?age=32&salary=30000 HTTP/1.1" 200 -
[1]
```

Testing the service using Postman



b) Stochastic Gradient Boosting

```
import pandas as pd

ds = pd.read_csv("IrisNew.csv")

ds.head()
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

```
X = ds.iloc[:,1:5]

y = ds.iloc[:,5]

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

from sklearn.linear_model import SGDClassifier

sgd = SGDClassifier(loss = 'hinge',penalty='l2',max_iter=1000,
random_state=None,learning_rate= 'optimal')

sgd.fit(X_train, y_train)
```

SGDClassifier()

```
y_pred = sgd.predict(X_test)

from sklearn.metrics import confusion_matrix, accuracy_score

sgd = confusion_matrix(y_test, y_pred)

sgd
```

```
array([[15,  0,  0],
       [ 1, 10,  1],
       [ 0,  1, 17]], dtype=int64)
```

```
ac_sgd = accuracy_score(y_test, y_pred)

ac_sgd
```

0.9333333333333333

c) Voting Ensemble.

```
import pandas as pd

import numpy as np

ds = pd.read_csv('diabetes.csv')

ds.head()'
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

```
X = ds.iloc[:,0:8]

y = ds.iloc[:, 8]

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X = sc.fit_transform(X)

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
```

 # Applying 5 different algrithm on single dataset

 # Applying Logistic Regression [82%]

```
from sklearn.linear_model import LogisticRegression

logreg = LogisticRegression()

logreg.fit(X_train, y_train)

y_pred_log = logreg.predict(X_test)


from sklearn.metrics import accuracy_score
```

```
ac = accuracy_score(y_test, y_pred_log)
ac
```

0.8246753246753247

 # Applying Decision Tree [79%]

```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier()
dt.fit(X_train, y_train)
y_pred_dt = dt.predict(X_test)

from sklearn.metrics import accuracy_score
ac = accuracy_score(y_test, y_pred_dt)
ac
```

0.7922077922077922

 # Applying KNN [80%]

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
y_pred_knn = knn.predict(X_test)

from sklearn.metrics import accuracy_score
ac = accuracy_score(y_test, y_pred_knn)
ac
```

0.8051948051948052

 # Applying Random Forest [81%]

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators=1000, random_state=0)
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)

from sklearn.metrics import accuracy_score
ac = accuracy_score(y_test, y_pred_rf)
ac
```

0.8181818181818182

 # Applying Adaboost [74%]

```
from sklearn.ensemble import AdaBoostClassifier
adb = AdaBoostClassifier(n_estimators=1000, random_state=0)
adb.fit(X_train, y_train)
y_pred_adb = adb.predict(X_test)

from sklearn.metrics import accuracy_score
ac = accuracy_score(y_test, y_pred_adb)
ac
```

0.7467532467532467

 # Voting

 # Soft Voting without weight

```
from sklearn.ensemble import VotingClassifier
vc = VotingClassifier(estimators = [('LogisticReg', logreg),
                              ('DecisionTree',dt),
```

```
                ('RandomForest',rf),

                ('AdaBoost',adb),

                ('Kneighbor',knn)], voting = 'soft')


vc.fit(X_train, y_train)
```

```
VotingClassifier(estimators=[('LogisticReg', LogisticRegression()),
                             ('DecisionTree', DecisionTreeClassifier()),
                             ('RandomForest',
                              RandomForestClassifier(n_estimators=1000,
                                                     random_state=0)),
                             ('AdaBoost',
                              AdaBoostClassifier(n_estimators=1000,
                                                 random_state=0)),
                             ('Kneighbor', KNeighborsClassifier())],
                 voting='soft')
```

```
y_pred_vc = vc.predict(X_test)

from sklearn.metrics import accuracy_score

ac = accuracy_score(y_test, y_pred_vc)

ac
```

0.8311688311688312

 # Soft Voting with weight (Scores ----0%) name, model

```
def get_model():

    models = list()

    models.append(('lr', LogisticRegression()))

    models.append(('dt', DecisionTreeClassifier()))

    models.append(('knn', KNeighborsClassifier()))

    models.append(('rf', RandomForestClassifier()))

    models.append(('adb', AdaBoostClassifier()))

    return models
```

```
def evaluate_model(models, X_train, X_test, y_train, y_test):
    scores = list()
    for name, model in models:
        model.fit(X_train, y_train)
        yhat = model.predict(X_test)
        acc = accuracy_score(y_test, yhat)
        scores.append(acc)
    return scores

models = get_model() # create a base model
scores = evaluate_model(models, X_train, X_test, y_train, y_test)
vc1 = VotingClassifier(estimators = models, voting = 'soft', weights = scores)
vc1.fit(X_train, y_train)
y_pred_vc1 = vc1.predict(X_test)
```

0.8181818181818182

 # Hard voting with weights

```
vc2 = VotingClassifier(estimators = models, voting = 'hard', weights = scores)
vc2.fit(X_train, y_train)
y_pred_vc2 = vc2.predict(X_test)
acc_hd = accuracy_score(y_test, y_pred_vc2)
acc_hd
```

0.8441558441558441

Practical 11: Study of Python Flask Library

AdaBoostIrisModel

```
import pandas as pd

import pickle

ds = pd.read_csv('IrisNew.csv')

ds.head()
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

```
X = ds.iloc[:,1:5]

y = ds.iloc[:,5]

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3)

from sklearn.ensemble import AdaBoostClassifier

classifier = AdaBoostClassifier()

classifier.fit(X_train, y_train)
```

AdaBoostClassifier()

```
y_pred = classifier.predict(X_test)

from sklearn.metrics import accuracy_score, confusion_matrix

cf_matrix = confusion_matrix(y_test,y_pred)

cf_matrix
```

```
array([[16,  0,  0],
       [ 0, 11,  0],
       [ 0,  2, 16]], dtype=int64)
```

```
ac = accuracy_score(y_test,y_pred)

ac
```

0.9555555555555556

```
with open('modelIris.pkl','wb') as file:
    pickle.dump(classifier, file)
```

Client App (Testing the Model)

```
import flask
from flask import Flask, request
import pickle
model_adaboost = pickle.load(open('modelIris.pkl','rb'))
app = Flask(__name__)
```

 # Two parts (base address + route address)

```
@app.route('/', methods = ['GET','POST'])
def main():
    return "Ada Boost Flask API Development"


@app.route('/classify', methods = ['GET'])
def classify():
    if flask.request.method == 'GET':
        SepalLengthCm = request.args.get('sl')
        SepalWidthCm = request.args.get('sw')
```

```python
    PetalLengthCm = request.args.get('pl')
    PetalWidthCm = request.args.get('pw')

    prediction = model_adaboost.predict([[SepalLengthCm, SepalWidthCm, PetalLengthCm, PetalWidthCm]])

    return 'Class of species is '+str(prediction)
if __name__ == '__main__':
  app.run()
```

```
 * Serving Flask app "__main__" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [23/Jul/2021 10:17:10] "GET /classify?sl=5&sw=5&pl=5&pw=5 HTTP/1.1" 200 -
```

Testing using Postman