ROLL NO:102
FYMCA DIV: B

Practical No: 01
Aim: Implementation Of Prolog

Code:
precious(gold)precious(silver).
precious(pearl).

Output:
```
?- [prac1].
true.

?- precious(gold).
true.

?- precious(silver).
true.

?- precious(platinum).
false.

?- precious(silve).
false.
```

Code:
female (mary).
father (surya, ramesh).
likes (surya, food).
likes (surya, car).
likes (surya, bike).
likes (surya, book).
likes (shreya, chocolate).
likes (sudha, food).
likes (surya, coffee).
likes (samudra, chocolate).
likes (samanta, X): likes (X, chocolate).

Output:
```
Welcome to SWI-Prolog (threaded, 64 bits, version 9.2.4)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- [prac1].
true.

?- likes(surya,food).
true ;
false.

?- likes(shreya,book).
false.

?- likes(samanta,X).
X = shreya ,

?- likes(samudra,X).
X = chocolate.

?-
```

Practical No: 02

Aim: Implementation of Water jug problem using Prolog

Code:

```
water_jug (X, Y): - X>4, Y<3, write ('4L jug overflow'), nl.
water_jug (X, Y): - X>4, Y>3, write ('3L jug overflow'), nl.
water_jug (X, Y): - X>4, Y>3, write ('Both jug overflow'), nl.
water_jug (X, Y): - (X=: =0, Y=: =0, nl, write ('4L:0 & 3L:3 (Action: Fill 3L
jug.)'), YY is 3,
water_jug (X, YY));
(X=: =0, Y=: =0, nl, write ('4L:4 & 3L:0 (Action: Fill 4L jug.)'), XX is 4,
water_jug (XX, Y));
(X=: =2, Y=: =0, nl, write ('4L:2 & 3L:0 (Action: Goal Stage reached...)'));
(X=: =4, Y=: =0, nl, write ('4L:1 & 3L:3 (Action: Pour water from 4L to 3L
jug.)'), XX is X-3,
YY is 3, water_jug (XX, YY));
(X=: =0, Y=: =3, nl, write ('4L:3 & 3L:0 (Action: Pour water from 3L to 4L
jug.)'), XX is 3, YY
is 0, water_jug (XX, YY));
(X=: =1, Y=: =3, nl, write ('4L:1 & 3L:0 (Action: Empty 3L jug.)'), YY is 0,
water_jug (X, YY));
(X=: =3, Y=: =0, nl, write ('4L:3 & 3L:3 (Action: Fill 3L jug.)'), YY is 3,
water_jug (X, YY));
(X=: =3, Y=: =3, nl, write ('4L:4 & 3L:2 (Action: Pour water from 3L jug to 4L
jug until 4L jug is full.)'), XX is X+1, YY is Y-1, water_jug (XX, YY));
(X=: =1, Y=: =0, nl, write ('4L:0 & 3L:1 (Action: Pour water from 4L jug to 3L
jug.)'), XX is Y,
YY is X, water_jug (XX, YY));
(X=:=0,Y=:=1, nl, write ('4L:4 & 3L:1 (Action: Pour water from 4L.)'), XX is 4,
water_jug (XX, Y));
(X=: =4, Y=: =1, nl, write ('4L:2 & 3L:3 (Action: Pour water from 4L jug to 3L
jug until 3L jug
is full.)'), XX is X-2, YY is Y+2, water_jug (XX, YY));
(X=: =2, Y=: =3, nl, write ('4L:2 & 3L:0 (Action: Empty 3L jug.)'), YY is 0,
water_jug (X, YY));
(X=: = 4, Y=: =2, nl, write ('4L:0 & 3L:2 (Action: Empty 4L jug.)'), XX is 0,
water_jug (XX, Y));
(X=: =0, Y=: =2, nl, write ('4L:2 & 3L:0 (Action: Pour water from 3L jug to 4L
jug.)'), XX is Y,
YY is X, water_jug (XX, YY)).
```

Output:

```
Welcome to SWI-Prolog (threaded, 64 bits, version 9.2.4)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- [prac2].
true.

?- waterjug(9,0).
Correct to: "water_jug(9,0)"? yes
4L jug overflow
true
Unknown action: e (h for help)
Action?
Unknown action: s (h for help)
Action? .

?- waterjug(0,6).
Correct to: "water_jug(0,6)"?
Please answer 'y' or 'n'? yes
false.

?- waterjug(6,6).
Correct to: "water_jug(6,6)"?
Please answer 'y' or 'n'? yes
3L jug overflow
true .

?- waterjug(0,0).
Correct to: "water_jug(0,0)"? yes

4L:0 & 3L:3 (Action : Fill 3L jug.)
4L:3 & 3L:0 (Action : Pour water from 3L to 4L jug.)
4L:3 & 3L:3 (Action : Fill 3L jug.)
4L:4 & 3L:2 (Action : Pour water from 3L jug to 4L jug untill 4L jug
is full.)
4L:0 & 3L:2 (Action : Empty 4L jug .)
4L:2 & 3L:0 (Action : Pour water from 3L jug to 4L jug.)
4L:2 & 3L:0 (Action : Goal Stage reached...)
true .

?-
|
```

ROLL NO:102
FYMCA DIV: B

Practical No:3

Aim: Introduction to Python Programming

Output:

jupyter Untitled33 Last Checkpoint: 3 minutes ago (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help

In [7]:
```python
# Assigning value to a string variable
myString = "Hello World!"

# Setting working directory
import os
print(os.getcwd())
```

```
C:\Users\Lab2_51
```

In [8]:
```python
# Assigning value to x
x = 1
print(type(x))

# Checking if x is an integer
print(isinstance(x, int))
```

```
<class 'int'>
True
```

In [9]:
```python
# Rounding x and assigning it to y
y = round(x)
print(y)

# Assigning value to z and converting it to integer
z = 3.14
z = int(z)
print(z)

# Creating a vector using list
x1 = [34, 52.5, 45.2]
print(x1)
```

```
1
3
[34, 52.5, 45.2]
```

In [10]:
```python
# Arithmetic operations
r1 = 21
r2 = 20
add = r1 + r2
print(add)
```

```
41
```

In [11]:
```python
s1 = 22
s2 = 12
sub = s2 - s1
print(sub)
```

```
-10
```

In [12]:
```python
m1 = 2
m2 = 5
mul = m1 * m2
print(mul)
```

```
10
```

In [13]:
```python
d1 = 20
d2 = 10
div = d1 / d2
print(div)
```

```
2.0
```

ROLL NO:102
FYMCA DIV: B

Practical No:4
Aim: Introduction to Python Libraries

Output:

```
In [1]: import numpy as np
        import pandas as pd

In [2]: df = pd.read_csv('C:\\Users\\USER\\Desktop\\eda\\mtcars.csv')

In [3]: print(df)

                  model   mpg  cyl   disp   hp  drat     wt   qsec  vs  am  \
0             Mazda RX4  21.0    6  160.0  110  3.90  2.620  16.46   0   1
1         Mazda RX4 Wag  21.0    6  160.0  110  3.90  2.875  17.02   0   1
2            Datsun 710  22.8    4  108.0   93  3.85  2.320  18.61   1   1
3        Hornet 4 Drive  21.4    6  258.0  110  3.08  3.215  19.44   1   0
4     Hornet Sportabout  18.7    8  360.0  175  3.15  3.440  17.02   0   0
5               Valiant  18.1    6  225.0  105  2.76  3.460  20.22   1   0
6            Duster 360  14.3    8  360.0  245  3.21  3.570  15.84   0   0
7             Merc 240D  24.4    4  146.7   62  3.69  3.190  20.00   1   0
8              Merc 230  22.8    4  140.8   95  3.92  3.150  22.90   1   0
9              Merc 280  19.2    6  167.6  123  3.92  3.440  18.30   1   0
10            Merc 280C  17.8    6  167.6  123  3.92  3.440  18.90   1   0
11           Merc 450SE  16.4    8  275.8  180  3.07  4.070  17.40   0   0
12           Merc 450SL  17.3    8  275.8  180  3.07  3.730  17.60   0   0
13          Merc 450SLC  15.2    8  275.8  180  3.07  3.780  18.00   0   0
14   Cadillac Fleetwood  10.4    8  472.0  205  2.93  5.250  17.98   0   0
15  Lincoln Continental  10.4    8  460.0  215  3.00  5.424  17.82   0   0
16    Chrysler Imperial  14.7    8  440.0  230  3.23  5.345  17.42   0   0
17             Fiat 128  32.4    4   78.7   66  4.08  2.200  19.47   1   1
18          Honda Civic  30.4    4   75.7   52  4.93  1.615  18.52   1   1
19       Toyota Corolla  33.9    4   71.1   65  4.22  1.835  19.90   1   1
20        Toyota Corona  21.5    4  120.1   97  3.70  2.465  20.01   1   0
21     Dodge Challenger  15.5    8  318.0  150  2.76  3.520  16.87   0   0
22          AMC Javelin  15.2    8  304.0  150  3.15  3.435  17.30   0   0
23           Camaro Z28  13.3    8  350.0  245  3.73  3.840  15.41   0   0
24     Pontiac Firebird  19.2    8  400.0  175  3.08  3.845  17.05   0   0
25            Fiat X1-9  27.3    4   79.0   66  4.08  1.935  18.90   1   1
26        Porsche 914-2  26.0    4  120.3   91  4.43  2.140  16.70   0   1
27         Lotus Europa  30.4    4   95.1  113  3.77  1.513  16.90   1   1
28       Ford Pantera L  15.8    8  351.0  264  4.22  3.170  14.50   0   1
29         Ferrari Dino  19.7    6  145.0  175  3.62  2.770  15.50   0   1
30        Maserati Bora  15.0    8  301.0  335  3.54  3.570  14.60   0   1
31           Volvo 142E  21.4    4  121.0  109  4.11  2.780  18.60   1   1

    gear  carb
0      4     4
1      4     4
2      4     1
3      3     1
4      3     2
5      3     1
6      3     4
7      4     2
8      4     2
9      4     4
10     4     4
11     3     3
12     3     3
```

In [5]:
```
df.head()
```

Out[5]:

	model	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
1	Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
2	Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
3	Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
4	Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
5	Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1

In [ ]:
```
df.head(10)
```

In [6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32 entries, 0 to 31
Data columns (total 12 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   model   32 non-null     object
 1   mpg     32 non-null     float64
 2   cyl     32 non-null     int64
 3   disp    32 non-null     float64
 4   hp      32 non-null     int64
 5   drat    32 non-null     float64
 6   wt      32 non-null     float64
 7   qsec    32 non-null     float64
 8   vs      32 non-null     int64
 9   am      32 non-null     int64
 10  gear    32 non-null     int64
 11  carb    32 non-null     int64
dtypes: float64(5), int64(6), object(1)
memory usage: 3.1+ KB
```

In [9]:
```
df.isnull()
```

Out[9]:

	model	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	False	False	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	False	False
5	False	False	False	False	False	False	False	False	False	False	False	False
6	False	False	False	False	False	False	False	False	False	False	False	False
7	False	False	False	False	False	False	False	False	False	False	False	False
8	False	False	False	False	False	False	False	False	False	False	False	False
9	False	False	False	False	False	False	False	False	False	False	False	False
10	False	False	False	False	False	False	False	False	False	False	False	False

In [10]:
```
df.isnull().sum()
```

Out[10]:
```
model    0
mpg      0
cyl      0
disp     0
hp       0
drat     0
wt       0
qsec     0
vs       0
am       0
gear     0
carb     0
dtype: int64
```

In [11]:
```
df.tail()
```

Out[11]:

	model	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
27	Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.9	1	1	5	2
28	Ford Pantera L	15.8	8	351.0	264	4.22	3.170	14.5	0	1	5	4
29	Ferrari Dino	19.7	6	145.0	175	3.62	2.770	15.5	0	1	5	6
30	Maserati Bora	15.0	8	301.0	335	3.54	3.570	14.6	0	1	5	8
31	Volvo 142E	21.4	4	121.0	109	4.11	2.780	18.6	1	1	4	2

In [12]:
```
df.describe()
```

Out[12]:

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
count	32.000000	32.000000	32.000000	32.000000	32.000000	32.000000	32.000000	32.000000	32.000000	32.000000	32.000
mean	20.090625	6.187500	230.721875	146.687500	3.596563	3.217250	17.848750	0.437500	0.406250	3.687500	2.812
std	6.026948	1.785922	123.938694	68.562868	0.534679	0.978457	1.786943	0.504016	0.498991	0.737804	1.615
min	10.400000	4.000000	71.100000	52.000000	2.760000	1.513000	14.500000	0.000000	0.000000	3.000000	1.000
25%	15.425000	4.000000	120.825000	96.500000	3.080000	2.581250	16.892500	0.000000	0.000000	3.000000	2.000
50%	19.200000	6.000000	196.300000	123.000000	3.695000	3.325000	17.710000	0.000000	0.000000	4.000000	2.000
75%	22.800000	8.000000	326.000000	180.000000	3.920000	3.610000	18.900000	1.000000	1.000000	4.000000	4.000
max	33.900000	8.000000	472.000000	335.000000	4.930000	5.424000	22.900000	1.000000	1.000000	5.000000	8.000

In [13]:
```
df.size
```

Out[13]: 384

In [14]:
```
df.shape
```

Out[14]: (32, 12)

In [15]:
```
df.ndim
```

Out[15]: 2

In [16]:
```
df.at[4,'model']
```

Out[16]: 'Hornet Sportabout'

In [17]:
```
df.at[4,'disp']
```

Out[17]: 360.0

In [18]:
```
df.iat[3,4]
```

Out[18]: 110

In [20]:
```
df.loc[:,'model']
```

Out[20]:
```
0             Mazda RX4
1         Mazda RX4 Wag
2            Datsun 710
3        Hornet 4 Drive
4     Hornet Sportabout
5               Valiant
6            Duster 360
7             Merc 240D
8              Merc 230
9              Merc 280
10            Merc 280C
11           Merc 450SE
12           Merc 450SL
13          Merc 450SLC
14   Cadillac Fleetwood
15  Lincoln Continental
16    Chrysler Imperial
17             Fiat 128
18          Honda Civic
19       Toyota Corolla
20        Toyota Corona
21     Dodge Challenger
22          AMC Javelin
23           Camaro Z28
24     Pontiac Firebird
25            Fiat X1-9
26        Porsche 914-2
27         Lotus Europa
28       Ford Pantera L
29         Ferrari Dino
30        Maserati Bora
31           Volvo 142E
Name: model, dtype: object
```

In [23]:
```
df.iloc[0:5,0:2]
```

Out[23]:

	model	mpg
0	Mazda RX4	21.0
1	Mazda RX4 Wag	21.0
2	Datsun 710	22.8
3	Hornet 4 Drive	21.4
4	Hornet Sportabout	18.7

In [26]:
```
df.iloc[22:32,:]
```

Out[26]:

	model	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
22	AMC Javelin	15.2	8	304.0	150	3.15	3.435	17.30	0	0	3	2
23	Camaro Z28	13.3	8	350.0	245	3.73	3.840	15.41	0	0	3	4
24	Pontiac Firebird	19.2	8	400.0	175	3.08	3.845	17.05	0	0	3	2
25	Fiat X1-9	27.3	4	79.0	66	4.08	1.935	18.90	1	1	4	1
26	Porsche 914-2	26.0	4	120.3	91	4.43	2.140	16.70	0	1	5	2
27	Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.90	1	1	5	2
28	Ford Pantera L	15.8	8	351.0	264	4.22	3.170	14.50	0	1	5	4
29	Ferrari Dino	19.7	6	145.0	175	3.62	2.770	15.50	0	1	5	6
30	Maserati Bora	15.0	8	301.0	335	3.54	3.570	14.60	0	1	5	8
31	Volvo 142E	21.4	4	121.0	109	4.11	2.780	18.60	1	1	4	2

In [27]:
```
df.dtypes
```

Out[27]:
```
model     object
mpg      float64
cyl        int64
disp     float64
hp         int64
drat     float64
wt       float64
qsec     float64
vs         int64
am         int64
gear       int64
carb       int64
dtype: object
```

```
In [28]: df['model'].dtype

Out[28]: dtype('O')

In [29]: df.axes

Out[29]: [RangeIndex(start=0, stop=32, step=1),
          Index(['model', 'mpg', 'cyl', 'disp', 'hp', 'drat', 'wt', 'qsec', 'vs', 'am',
                 'gear', 'carb'],
                dtype='object')]

In [30]: df.columns

Out[30]: Index(['model', 'mpg', 'cyl', 'disp', 'hp', 'drat', 'wt', 'qsec', 'vs', 'am',
                'gear', 'carb'],
               dtype='object')

In [31]: df['hp'].std()

Out[31]: 68.56286848932059

In [32]: df['mpg'].mean()

Out[32]: 20.090625000000003

In [33]: df['mpg'].median()

Out[33]: 19.2

In [34]: df['hp'].describe()

Out[34]: count     32.000000
         mean     146.687500
         std       68.562868
         min       52.000000
         25%       96.500000
         50%      123.000000
         75%      180.000000
         max      335.000000
         Name: hp, dtype: float64
```

Practical No:5

Aim: Program to Implement Linear Regression

Output:

```
[1]: import pandas as pd
```

```
[2]: import numpy as np
```

```
[3]: from sklearn import linear_model
```

```
[4]: import matplotlib.pyplot as plt
```

```
[5]: df = pd.read_csv(r'C:\Users\nisha\OneDrive\Desktop\area.csv')
```

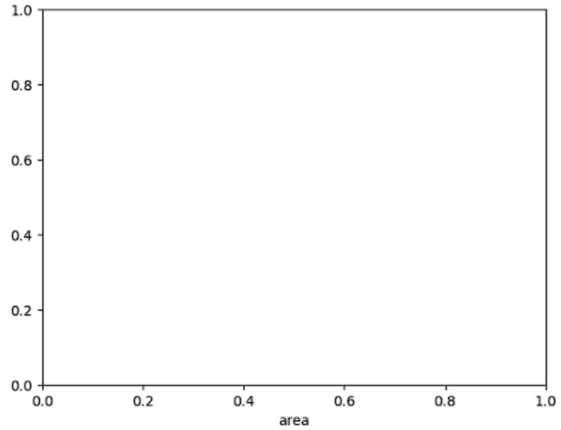
```
[6]: df
```

[6]:

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

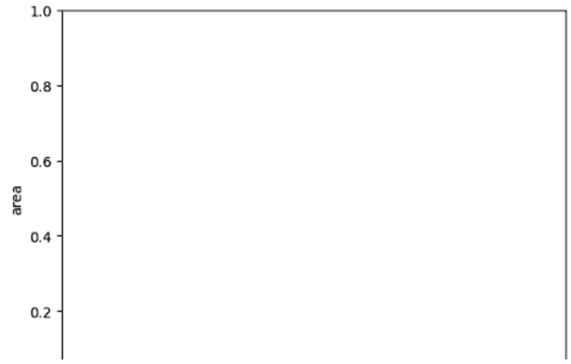
```
[7]: plt.xlabel('area')
```

```
[7]: Text(0.5, 0, 'area')
```

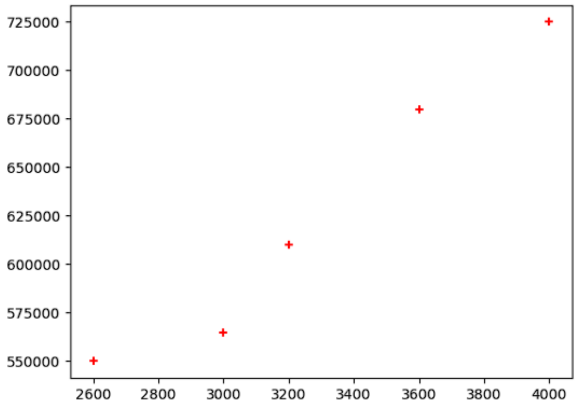



```
[8]: plt.ylabel('area')
```

```
[8]: Text(0, 0.5, 'area')
```



```
[13]: plt.scatter(df.area,df.price,color='red',marker='+')
[13]: <matplotlib.collections.PathCollection at 0x19db6be6410>
```




```
[14]: new_df = df.drop('price',axis='columns')
[15]: new_df
```

[15]:

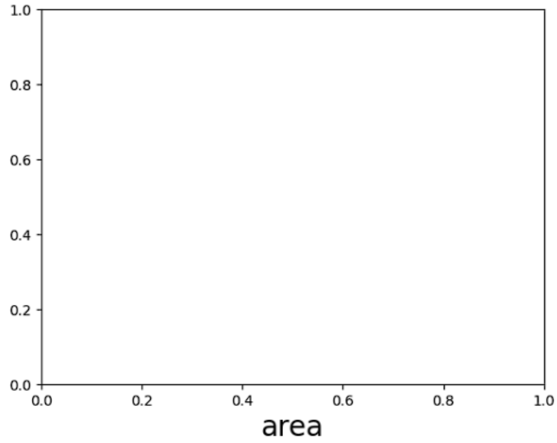
	area
0	2600
1	3000
2	3200
3	3600
4	4000

```
[16]: price = df.price
[17]: price
[17]: 0    550000
      1    565000
      2    610000
      3    680000
      4    725000
      Name: price, dtype: int64
[18]: reg = linear_model.LinearRegression()
[19]: reg.fit(new_df,price)
[19]: ▾ LinearRegression
      LinearRegression()
[20]: reg.predict([[3300]])
      C:\Users\nisha\anaconda3\Lib\site-packages\sklearn\base.py:439: UserWarning: X does not have valid feature names, but LinearRegression was fitted with f
      eature names
        warnings.warn(
[20]: array([628715.75342466])
[21]: reg.coef_
[21]: array([135.78767123])
[22]: reg.intercept_
[22]: 180616.43835616432
[23]: 3300*135.78767123 + 180616.43835616432
[23]: 628715.7534151643
[24]: reg.predict([[5000]])
      C:\Users\nisha\anaconda3\Lib\site-packages\sklearn\base.py:439: UserWarning: X does not have valid feature names, but LinearRegression was fitted with f
      eature names
        warnings.warn(
[24]: array([859554.79452055])
[25]: plt.xlabel('area',fontsize=20)
```

```
[24]: array([859554.79452055])
```

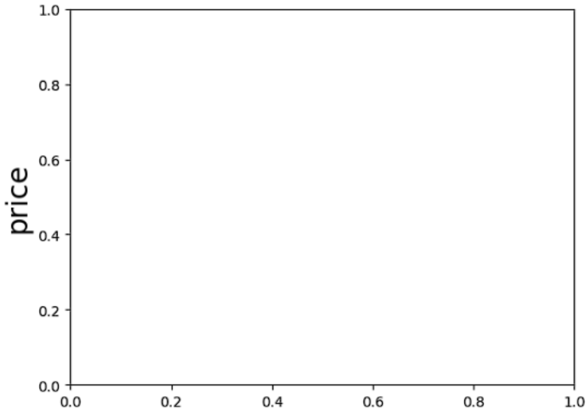
```
[25]: plt.xlabel('area',fontsize=20)
```

```
[25]: Text(0.5, 0, 'area')
```



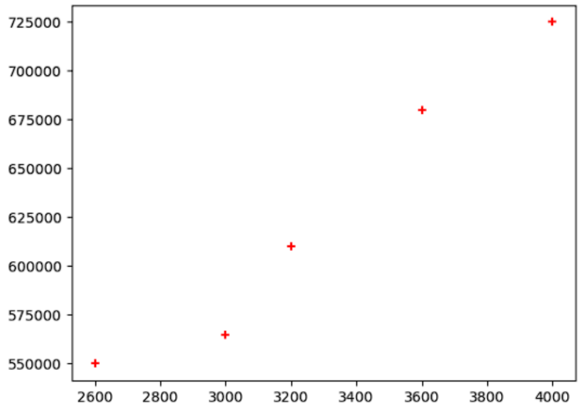

```
[26]: plt.ylabel('price',fontsize=20)
```

```
[26]: Text(0, 0.5, 'price')
```



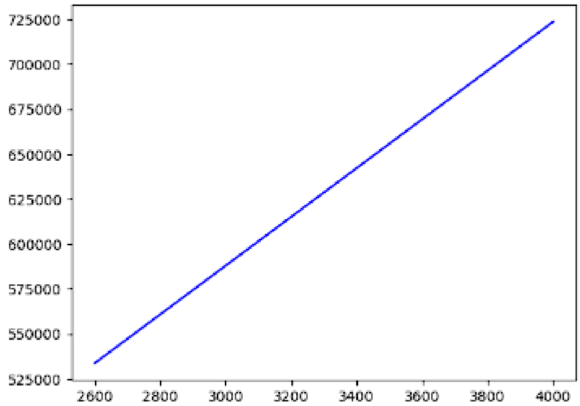
[27]:
```
plt.scatter(df.area,df.price,color='red',marker='+')
```

[27]:
```
<matplotlib.collections.PathCollection at 0x19db72ee490>
```



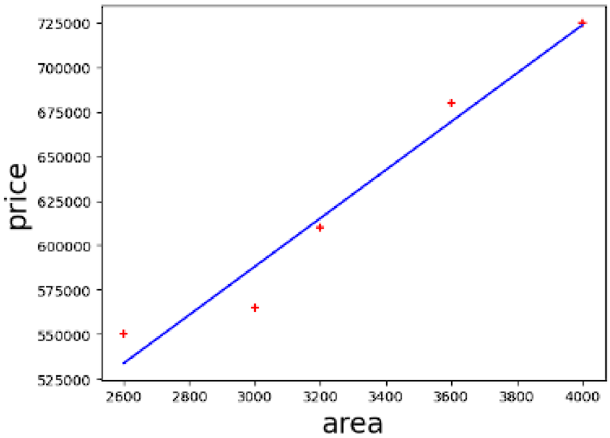
[28]:
```
plt.plot(df.area,reg.predict(df[['area']]),color='blue')
```

[28]:
```
[<matplotlib.lines.Line2D at 0x19db4d44810>]
```



[29]:
```
plt.xlabel('area',fontsize=20)
plt.ylabel('price',fontsize=20)
plt.scatter(df.area,df.price,color='red',marker='+')
plt.plot(df.area,reg.predict(df[['area']]),color='blue')
```

[29]:
```
[<matplotlib.lines.Line2D at 0x19db7589890>]
```



[ ]:

Practical No:6

Aim: Program to Implement Logistic Regression

Output:

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

[2]:
```
credit_df = pd.read_csv('C:/Users/nisha/OneDrive/Desktop/CreditRisk.csv')
```

[3]:
```
credit_df.shape
```

[3]: (614, 13)

In [4]:
```
credit_df.head()
```

Out[4]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Pr
0	LP001002	Male	No	0	Graduate	No	5849	0.0	0	360.0	1.0	
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128	360.0	1.0	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66	360.0	1.0	
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120	360.0	1.0	
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141	360.0	1.0	

In [5]:
```
credit_df.tail()
```

Out[5]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
609	LP002978	Female	No	0	Graduate	No	2900	0.0	71	360.0	1.0
610	LP002979	Male	Yes	3+	Graduate	No	4106	0.0	40	180.0	1.0
611	LP002983	Male	Yes	1	Graduate	No	8072	240.0	253	360.0	1.0
612	LP002984	Male	Yes	2	Graduate	No	7583	0.0	187	360.0	1.0
613	LP002990	Female	No	0	Graduate	Yes	4583	0.0	133	360.0	0.0

In [6]:
```
credit_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Loan_ID            614 non-null    object
 1   Gender             601 non-null    object
 2   Married            611 non-null    object
 3   Dependents         599 non-null    object
 4   Education          614 non-null    object
 5   Self_Employed      582 non-null    object
 6   ApplicantIncome    614 non-null    int64
 7   CoapplicantIncome  614 non-null    float64
 8   LoanAmount         614 non-null    int64
 9   Loan_Amount_Term   600 non-null    float64
 10  Credit_History     564 non-null    float64
 11  Property_Area      614 non-null    object
 12  Loan_Status        614 non-null    int64
dtypes: float64(3), int64(3), object(7)
memory usage: 62.5+ KB
```

In [7]:
```
credit_df.describe()
```

Out[7]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Loan_Status
count	614.000000	614.000000	614.000000	600.00000	564.000000	614.000000
mean	5403.459283	1621.245798	141.166124	342.00000	0.842199	0.687296
std	6109.041673	2926.248369	88.340630	65.12041	0.364878	0.463973
min	150.000000	0.000000	0.000000	12.00000	0.000000	0.000000
25%	2877.500000	0.000000	98.000000	360.00000	1.000000	0.000000
50%	3812.500000	1188.500000	125.000000	360.00000	1.000000	1.000000
75%	5795.000000	2297.250000	164.750000	360.00000	1.000000	1.000000
max	81000.000000	41667.000000	700.000000	480.00000	1.000000	1.000000

In [8]:
```
credit_df.Loan_Status.value_counts()
```

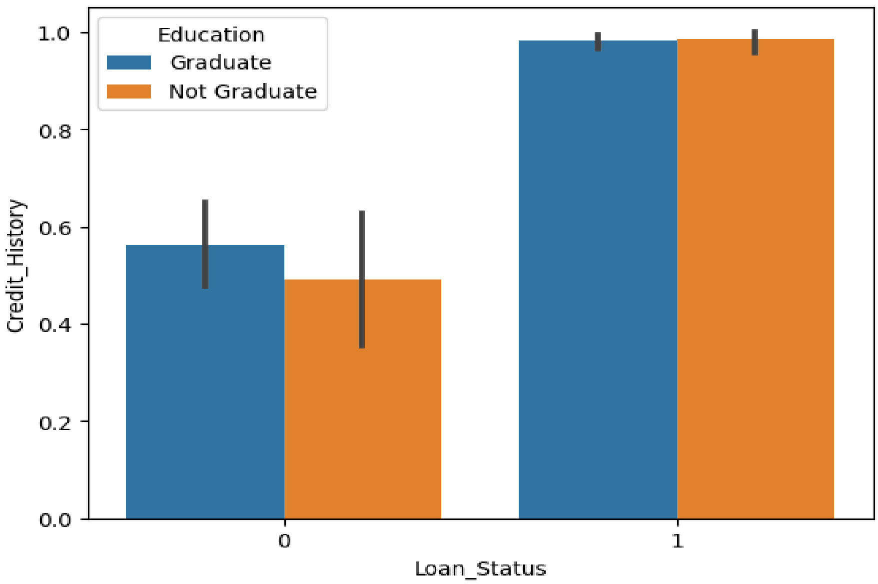
Out[8]:
```
1    422
0    192
Name: Loan_Status, dtype: int64
```

In [9]:
```
credit_df.groupby(['Education', 'Loan_Status']).Education.count()
```

Out[9]:
```
Education     Loan_Status
Graduate      0              140
              1              340
Not Graduate  0               52
              1               82
Name: Education, dtype: int64
```

In [10]:
```
sns.barplot(y = 'Credit_History' , x='Loan_Status', hue='Education', data=credit_df)
```

Out[10]: <Axes: xlabel='Loan_Status', ylabel='Credit_History'>



In [11]:
```
100 * credit_df.isnull().sum() / credit_df.shape[0]
```

Out[11]:
```
Loan_ID              0.000000
Gender               2.117264
Married              0.488599
Dependents           2.442997
Education            0.000000
Self_Employed        5.211726
ApplicantIncome      0.000000
CoapplicantIncome    0.000000
LoanAmount           0.000000
Loan_Amount_Term     2.280130
Credit_History       8.143322
Property_Area        0.000000
Loan_Status          0.000000
dtype: float64
```

In [12]:
```
DF=credit_df.drop(credit_df.columns[0],axis=1)
```

In [13]:
```
DF.head()
```

Out[13]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area
0	Male	No	0	Graduate	No	5849	0.0	0	360.0	1.0	Urbar
1	Male	Yes	1	Graduate	No	4583	1508.0	128	360.0	1.0	Rura
2	Male	Yes	0	Graduate	Yes	3000	0.0	66	360.0	1.0	Urbar
3	Male	Yes	0	Not Graduate	No	2583	2358.0	120	360.0	1.0	Urbar
4	Male	No	0	Graduate	No	6000	0.0	141	360.0	1.0	Urbar

In [14]:
```
object_columns = DF.select_dtypes(include=['object']).columns
numeric_columns = DF.select_dtypes(exclude=['object']).columns
```

In [15]:
```
for column in object_columns:
    majority = DF[column].value_counts().iloc[0]
    DF[column].fillna(majority, inplace=True)
```

In [17]:
```
for column in numeric_columns:
    mean = DF[column].mean()
    DF[column].fillna(mean, inplace=True)
```

In [18]:
```
DF.head()
```

Out[18]:

arried	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status
No	0	Graduate	No	5849	0.0	0	360.0	1.0	Urban	1
Yes	1	Graduate	No	4583	1508.0	128	360.0	1.0	Rural	0
Yes	0	Graduate	Yes	3000	0.0	66	360.0	1.0	Urban	1
Yes	0	Not Graduate	No	2583	2358.0	120	360.0	1.0	Urban	1
No	0	Graduate	No	6000	0.0	141	360.0	1.0	Urban	1

In [20]:
```
DF[object_columns].Property_Area #Categorical Columns
```

Out[20]:
```
0          Urban
1          Rural
2          Urban
3          Urban
4          Urban
         ...
609        Rural
610        Rural
611        Urban
612        Urban
613    Semiurban
Name: Property_Area, Length: 614, dtype: object
```

In [21]:
```
DF[object_columns].Property_Area.head()
```

Out[21]:
```
0    Urban
1    Rural
2    Urban
3    Urban
4    Urban
Name: Property_Area, dtype: object
```

In [22]:
```
Df_dummy = pd.get_dummies(DF, columns=object_columns)
```

In [23]:
```
Df_dummy.head()
```

Out[23]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Loan_Status	Gender_489	Gender_Female	Gender_Male	Married_398
0	5849	0.0	0	360.0	1.0	1	0	0	1	0
1	4583	1508.0	128	360.0	1.0	0	0	0	1	0
2	3000	0.0	66	360.0	1.0	1	0	0	1	0
3	2583	2358.0	120	360.0	1.0	1	0	0	1	0
4	6000	0.0	141	360.0	1.0	1	0	0	1	0

5 rows × 25 columns

In [24]:
```
Df_dummy.shape
```

Out[24]: (614, 25)

In [25]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

In [29]:
```
x = Df_dummy.drop('Loan_Status', axis=1)
y = Df_dummy.Loan_Status
train_x, test_x, train_y, test_y = train_test_split(x, y, test_size=0.3, random_state=42)
```

In [30]:
```
train_x.shape, test_x.shape
```

Out[30]: ((429, 24), (185, 24))

```
model = LogisticRegression() #LogisticRegression_Model
```

```
model.fit(train_x, train_y)
```

```
train_y_hat = model.predict(train_x)
test_y_hat = model.predict(test_x)
```

```
print('train_accuracy', accuracy_score(train_y, train_y_hat))
print('test accuracy', accuracy_score(test_y, test_y_hat))
```

```
train_accuracy 0.8205128205128205
test accuracy 0.7837837837837838
```

In [35]:
```
print(confusion_matrix(train_y, train_y_hat))
```

```
[[ 57  70]
 [  7 295]]
```

In [36]:
```
print(confusion_matrix(test_y, test_y_hat))
```

```
[[ 27  38]
 [  2 118]]
```

In [37]:
```
test_y.value_counts()
```

Out[37]:
```
1    120
0     65
Name: Loan_Status, dtype: int64
```

In [38]:
```
pd.Series(test_y_hat).value_counts()
```

Out[38]:
```
1    156
0     29
dtype: int64
```

In [39]:
```
(57 + 295) / train_y.shape[0]
```

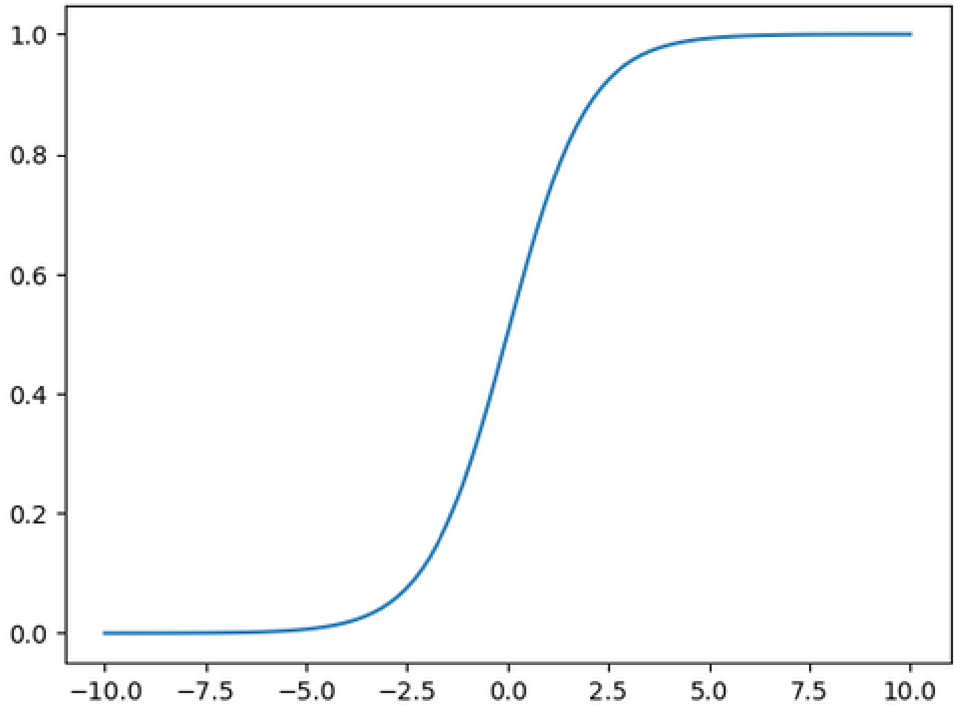
Out[39]:
```
0.8205128205128205
```

In [40]:
```
print(classification_report(test_y, test_y_hat))
```

```
              precision    recall  f1-score   support

           0       0.93      0.42      0.57        65
           1       0.76      0.98      0.86       120

    accuracy                           0.78       185
   macro avg       0.84      0.70      0.71       185
weighted avg       0.82      0.78      0.76       185
```

In [41]:
```
x = np.linspace(-10, 10, 100)
y = 1 / (1 + np.exp(-x)) #Sigmoid
plt.plot(x, y)
```

Out[41]: [<matplotlib.lines.Line2D at 0x1d457f4c460>]



ROLL NO:102
FYMCA DIV: B

Practical No:7

Aim: Implementation Of KNN Classification.

jupyter KNN Last Checkpoint: 3 minutes ago

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)

[1]:
```
import pandas as pd
from sklearn.datasets import load_iris
iris=load_iris()
```

[2]:
```
iris.feature_names
```

[2]:
```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

[3]:
```
iris.target_names
```

[3]:
```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

[5]:
```
df=pd.DataFrame(iris.data,columns=iris.feature_names)
df.head()
```

[5]:

	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

[6]:
```
df['target']=iris.target
df.head()
```

[6]:

	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

jupyter KNN Last Checkpoint: 4 minutes ago

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)

[6]:
```
df['target']=iris.target
df.head()
```

[6]:

	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

[7]:
```
df[df.target==1].head()
```

[7]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
50	7.0	3.2	4.7	1.4	1
51	6.4	3.2	4.5	1.5	1
52	6.9	3.1	4.9	1.5	1
53	5.5	2.3	4.0	1.3	1
54	6.5	2.8	4.6	1.5	1

[8]:
```
df['flower_name']=df.target.apply(lambda x:iris.target_names[x])
df.head()
```

[8]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	flower_name
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

jupyter KNN Last Checkpoint: 5 minutes ago

File Edit View Run Kernel Settings Help Trusted

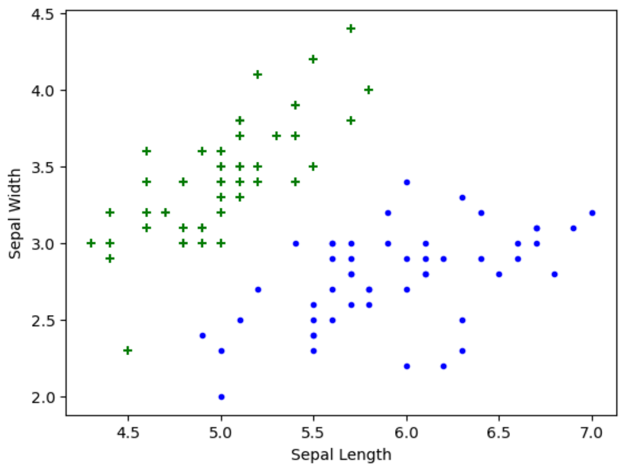
Code JupyterLab Python 3 (ipykernel)

[9]:
```
df0=df[:50]
df1=df[50:100]
df2=df[100:]
```

[10]:
```
import matplotlib.pyplot as plt
%matplotlib inline
```

[12]:
```
plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')
plt.scatter(df0['sepal length (cm)'],df0['sepal width (cm)'],color="green",marker='+')
plt.scatter(df1['sepal length (cm)'],df1['sepal width (cm)'],color="blue",marker='.')
```

[12]:
```
<matplotlib.collections.PathCollection at 0x15d77525310>
```



[13]:
```
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
plt.scatter(df0['petal length (cm)'],df0['petal width (cm)'],color="green",marker='+')
```

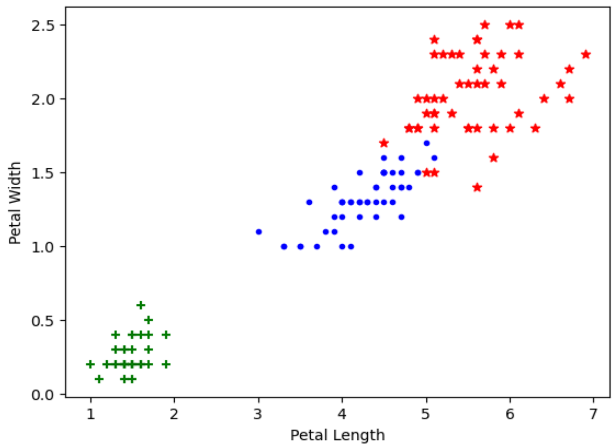
jupyter KNN Last Checkpoint: 5 minutes ago

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)

```
plt.scatter(df1['petal length (cm)'],df1['petal width (cm)'],color="blue",marker='.')
plt.scatter(df2['petal length (cm)'],df2['petal width (cm)'],color="red",marker='*')
```

[13]:
```
<matplotlib.collections.PathCollection at 0x15d7caccf50>
```



[14]:
```
from sklearn.model_selection import train_test_split
```

[15]:
```
x=df.drop(['target','flower_name'],axis='columns')
y=df.target
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=1)
```

[16]:
```
len(x_train)
```

[16]:
```
120
```

[17]:
```
len(x_test)
```

[17]:
```
30
```

ROLL NO:102
FYMCA DIV: B

jupyter KNN Last Checkpoint: 6 minutes ago

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)

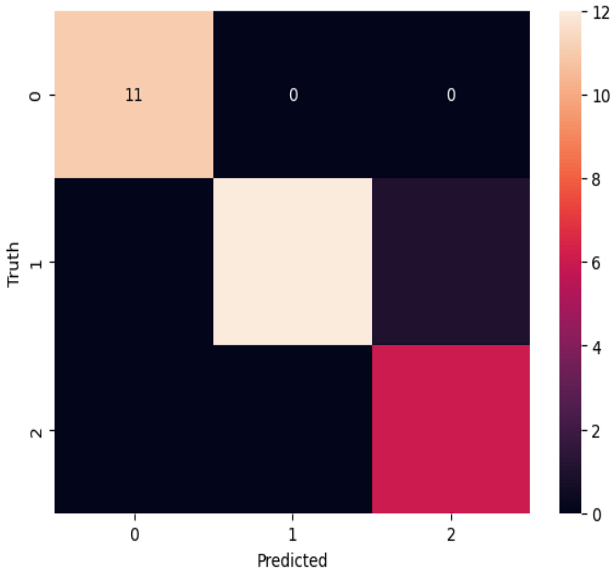
```
[17]: 30

[19]: from sklearn.neighbors import KNeighborsClassifier
      knn=KNeighborsClassifier(n_neighbors=10)

[20]: knn.fit(x_train,y_train)

[20]: KNeighborsClassifier
      KNeighborsClassifier(n_neighbors=10)

[21]: knn.score(x_test,y_test)

[21]: 0.9666666666666667

[22]: knn.predict([[4.8,3.0,1.5,0.3]])

C:\Users\nisha\anaconda3\Lib\site-packages\sklearn\base.py:439: UserWarning: X does not have valid feature names, but KNeighborsClassifier was fitted wi
th feature names
  warnings.warn(

[22]: array([0])

[23]: from sklearn.metrics import confusion_matrix
      y_pred=knn.predict(x_test)
      cm=confusion_matrix(y_test,y_pred)
      cm

[23]: array([[11,  0,  0],
             [ 0, 12,  1],
             [ 0,  0,  6]], dtype=int64)

[25]: %matplotlib inline
      import matplotlib.pyplot as plt
      import seaborn as sn
      plt.figure(figsize=(7,5))
      sn.heatmap(cm,annot=True)
      plt.xlabel('Predicted')
      plt.ylabel('Truth')

[25]: Text(58.222222222222214, 0.5, 'Truth')
```

jupyter KNN Last Checkpoint: 6 minutes ago

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)

```
      plt.figure(figsize=(7,5))
      sn.heatmap(cm,annot=True)
      plt.xlabel('Predicted')
      plt.ylabel('Truth')

[25]: Text(58.222222222222214, 0.5, 'Truth')
```



[ ]:

ROLL NO:102
FYMCA DIV: B

Practical No:8

Aim: Program to Implement Principal Component Analysis

Output:

jupyter PrincipalComponentAnalysis Last Checkpoint: 16 hours ago

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)

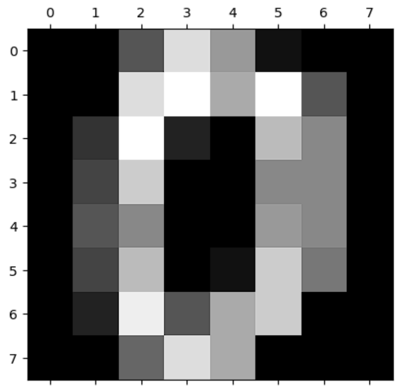
```
[1]: from sklearn.datasets import load_digits

[2]: import pandas as pd

[3]: datasets=load_digits()

[4]: datasets.keys()

[4]: dict_keys(['data', 'target', 'frame', 'feature_names', 'target_names', 'images', 'DESCR'])

[5]: datasets.data.shape

[5]: (1797, 64)

[6]: datasets.data[0]

[6]: 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.])

[7]: datasets.data[0].reshape(8,8)

[7]: 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.]])

[9]: from matplotlib import pyplot as plt
     %matplotlib inline
     plt.gray()
     plt.matshow(datasets.data[0].reshape(8,8))

[9]: <matplotlib.image.AxesImage at 0x19a3c5b46d0>
     <Figure size 640x480 with 0 Axes>
```

jupyter PrincipalComponentAnalysis Last Checkpoint: 16 hours ago

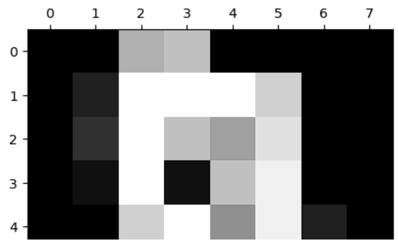
File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)

```
<Figure size 640x480 with 0 Axes>
```



```
[10]: plt.matshow(datasets.data[9].reshape(8,8))

[10]: <matplotlib.image.AxesImage at 0x19a3d7ec9d0>
```

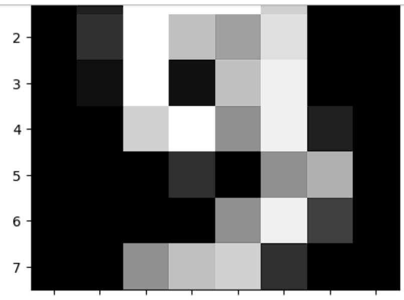


ROLL NO:102
FYMCA DIV: B

jupyter PrincipalComponentAnalysis Last Checkpoint: 16 hours ago

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)



[11]:
```
datasets.target[:5]
```

[11]:
```
array([0, 1, 2, 3, 4])
```

[13]:
```
df=pd.DataFrame(datasets.data,columns=datasets.feature_names)
df.head()
```

[13]:

	pixel_0_0	pixel_0_1	pixel_0_2	pixel_0_3	pixel_0_4	pixel_0_5	pixel_0_6	pixel_0_7	pixel_1_0	pixel_1_1	...	pixel_6_6	pixel_6_7	pixel_7_0	pixel_7_1	pixel_7_2	pixe
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	
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	
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	0.0	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	
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	

5 rows × 64 columns

[14]:
```
datasets.target
```

[14]:
```
array([0, 1, 2, ..., 8, 9, 8])
```

jupyter PrincipalComponentAnalysis Last Checkpoint: 16 hours ago

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)

[15]:
```
df.describe()
```

[15]:

	pixel_0_0	pixel_0_1	pixel_0_2	pixel_0_3	pixel_0_4	pixel_0_5	pixel_0_6	pixel_0_7	pixel_1_0	pixel_1_1	...	pixel_6_6	pixel_6_7	
count	1797.0	1797.000000	1797.000000	1797.000000	1797.000000	1797.000000	1797.000000	1797.000000	1797.000000	1797.000000	...	1797.000000	1797.000000	1
mean	0.0	0.303840	5.204786	11.835838	11.848080	5.781859	1.362270	0.129661	0.005565	1.993879	...	3.725097	0.206455	
std	0.0	0.907192	4.754826	4.248842	4.287388	5.666418	3.325775	1.037383	0.094222	3.196160	...	4.919406	0.984401	
min	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	
25%	0.0	0.000000	1.000000	10.000000	10.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	
50%	0.0	0.000000	4.000000	13.000000	13.000000	4.000000	0.000000	0.000000	0.000000	0.000000	...	1.000000	0.000000	
75%	0.0	0.000000	9.000000	15.000000	15.000000	11.000000	0.000000	0.000000	0.000000	3.000000	...	7.000000	0.000000	
max	0.0	8.000000	16.000000	16.000000	16.000000	16.000000	16.000000	15.000000	2.000000	16.000000	...	16.000000	13.000000	

8 rows × 64 columns

[16]:
```
x=df
y=datasets.target
```

[17]:
```
from sklearn.preprocessing import StandardScaler
```

[19]:
```
scaler=StandardScaler()
x_scaled=scaler.fit_transform(x)
x_scaled
```

[19]:
```
array([[ 0.        , -0.33501649, -0.04308102, ..., -1.14664746,
        -0.5056698 , -0.19600752],
       [ 0.        , -0.33501649, -1.09493684, ...,  0.54856067,
        -0.5056698 , -0.19600752],
       [ 0.        , -0.33501649, -1.09493684, ...,  1.56568555,
         1.6951369 , -0.19600752],
       ...,
       [ 0.        , -0.33501649, -0.88456568, ..., -0.12952258,
        -0.5056698 , -0.19600752],
       [ 0.        , -0.33501649, -0.67419451, ...,  0.8876023 ,
        -0.5056698 , -0.19600752],
```

ROLL NO:102
FYMCA DIV: B

jupyter PrincipalComponentAnalysis Last Checkpoint: 16 hours ago

File Edit View Run Kernel Settings Help — Trusted

JupyterLab — Python 3 (ipykernel)

```
        -0.5056698 , -0.19600752],
       [ 0.        , -0.33501649,  1.00877481, ...,  0.8876023 ,
        -0.26113572, -0.19600752]])
```

```
[20]: from sklearn.model_selection import train_test_split

[21]: x_train,x_test,y_train,y_test=train_test_split(x_scaled,y,test_size=0.2, random_state=30)

[22]: from sklearn.linear_model import LogisticRegression

[23]: model = LogisticRegression()
      model.fit(x_train, y_train)
      model.score(x_test, y_test)

[23]: 0.9722222222222222

[24]: from sklearn.decomposition import PCA

[26]: pca = PCA(0.95)
      x_pca = pca.fit_transform(x)
      x_pca.shape

[26]: (1797, 29)

[27]: pca.explained_variance_ratio_

[27]: array([0.14890594, 0.13618771, 0.11794594, 0.08409979, 0.05782415,
             0.0491691 , 0.04315987, 0.03661373, 0.03353248, 0.03078806,
             0.02372341, 0.02272697, 0.01821863, 0.01773855, 0.01467101,
             0.01409716, 0.01318589, 0.01248138, 0.01017718, 0.00905617,
             0.00889538, 0.00797123, 0.00767493, 0.00722904, 0.00695889,
             0.00596081, 0.00575615, 0.00515158, 0.0048954 ])

[28]: pca.n_components_

[28]: 29

[29]: x_train_pca, x_test_pca, y_train, y_test = train_test_split(x_pca, y, test_size=0.2, random_state=30)

[30]: from sklearn.linear_model import LogisticRegression

[31]: model = LogisticRegression(max_iter=1000)
```

jupyter PrincipalComponentAnalysis Last Checkpoint: 16 hours ago

File Edit View Run Kernel Settings Help — Trusted

JupyterLab — Python 3 (ipykernel)

```
[29]: x_train_pca, x_test_pca, y_train, y_test = train_test_split(x_pca, y, test_size=0.2, random_state=30)

[30]: from sklearn.linear_model import LogisticRegression

[31]: model = LogisticRegression(max_iter=1000)

[36]: model = LogisticRegression(max_iter=1000)
      model.fit(x_train_pca, y_train)
      model.score(x_test_pca, y_test)

[36]: 0.6083333333333333

[37]: pca = PCA(n_components=2)
      x_pca = pca.fit_transform(x)
      x_pca.shape

[37]: (1797, 2)

[38]: pca.explained_variance_ratio_

[38]: array([0.14890594, 0.13618771])

[39]: x_train_pca, x_test_pca, y_train, y_test = train_test_split(x_pca, y, test_size=0.2, random_state=30)

[40]: model = LogisticRegression(max_iter=1000)
      model.fit(x_train_pca, y_train)
      model.score(x_test_pca, y_test)

[40]: 0.6083333333333333

[ ]:
```

Practical No: 9

Aim: Program to Implement K-Means Algorithm

Output:

In [3]:
```
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
%matplotlib inline
```

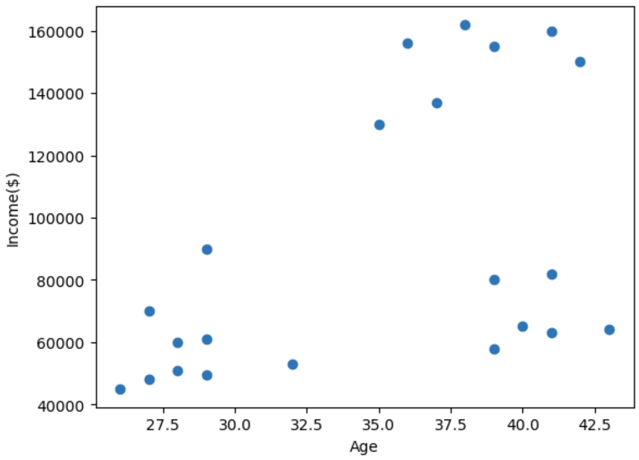
In [5]:
```
df=pd.read_csv('C:/Users/LAB2_41/Documents/Siddharth_63/income.csv')
df.head()
```

Out[5]:

	Name	Age	Income($)
0	Rob	27	70000
1	Michael	29	90000
2	Mohan	29	61000
3	Ismail	28	60000
4	Kory	42	150000

In [6]:
```
plt.scatter(df.Age,df['Income($)'])
plt.xlabel('Age')
plt.ylabel('Income($)')
```

Out[6]: Text(0, 0.5, 'Income($)')



In [7]:
```
km=KMeans(n_clusters=3)
y_predicted=km.fit_predict(df[['Age','Income($)']])
y_predicted
```

```
C:\Users\LAB2_41\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning: KMeans is known to have a memory lea
k on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=1.
  warnings.warn(
```

Out[7]: array([2, 2, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 0])

In [8]:
```
df['clusters']=y_predicted
df.head()
```

Out[8]:

	Name	Age	Income($)	clusters
0	Rob	27	70000	2
1	Michael	29	90000	2
2	Mohan	29	61000	0
3	Ismail	28	60000	0
4	Kory	42	150000	1

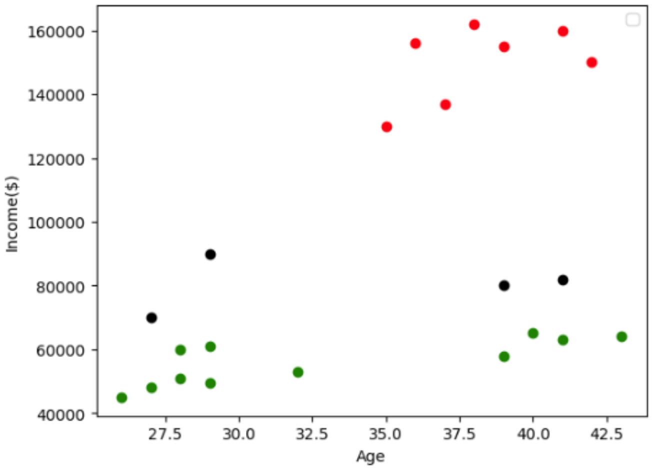
In [9]:
```
km.cluster_centers_
```

Out[9]:
```
array([[3.29090909e+01, 5.61363636e+04],
       [3.82857143e+01, 1.50000000e+05],
       [3.40000000e+01, 8.05000000e+04]])
```

In [13]:
```
df1=df[df.clusters==0]
df2=df[df.clusters==1]
df3=df[df.clusters==2]
plt.scatter(df1.Age,df1['Income($)'],color='green')
plt.scatter(df2.Age,df2['Income($)'],color='red')
plt.scatter(df3.Age,df3['Income($)'],color='black')
plt.xlabel('Age')
plt.ylabel('Income($)')
plt.legend()
```

```
No artists with labels found to put in legend.  Note that artists whose label start with an underscore are ignored when legend
() is called with no argument.
```

Out[13]: `<matplotlib.legend.Legend at 0x2011fb12da0>`



In [14]:
```
scaler=MinMaxScaler()

scaler.fit(df[['Income($)']])
df['Income($)']=scaler.transform(df[['Income($)']])

scaler.fit(df[['Age']])
df['Age']=scaler.transform(df[['Age']])
```

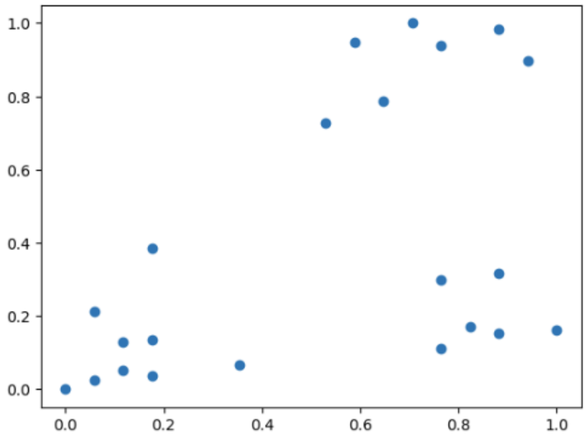
In [16]:
```
df.head(10)
```

Out[16]:

	Name	Age	Income($)	clusters
0	Rob	0.058824	0.213675	2
1	Michael	0.176471	0.384615	2
2	Mohan	0.176471	0.136752	0
3	Ismail	0.117647	0.128205	0
4	Kory	0.941176	0.897436	1
5	Gautam	0.764706	0.940171	1
6	David	0.882353	0.982906	1
7	Andrea	0.705882	1.000000	1
8	Brad	0.588235	0.948718	1
9	Angelina	0.529412	0.726496	1

In [19]:
```
plt.scatter(df.Age,df['Income($)'])
```

Out[19]: <matplotlib.collections.PathCollection at 0x20123121c90>



In [24]:
```
km=KMeans(n_clusters=3)
y_predicted=km.fit_predict(df[['Age','Income($)']])
y_predicted
```

```
C:\Users\LAB2_41\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning: KMeans is known to have a memory lea
k on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=1.
  warnings.warn(
```

Out[24]: array([1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2])

In [25]:
```
df['cluster']=y_predicted
df.head()
```

Out[25]:

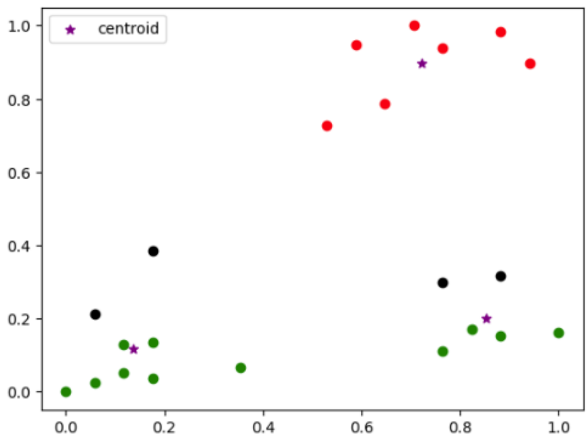
	Name	Age	Income($)	clusters	cluster
0	Rob	0.058824	0.213675	2	1
1	Michael	0.176471	0.384615	2	1
2	Mohan	0.176471	0.136752	0	1
3	Ismail	0.117647	0.128205	0	1
4	Kory	0.941176	0.897436	1	0

In [26]:
```
km.cluster_centers_
```

Out[26]:
```
array([[0.72268908, 0.8974359 ],
       [0.1372549 , 0.11633428],
       [0.85294118, 0.2022792 ]])
```


In [29]:
```
df1=df[df.clusters==0]
df2=df[df.clusters==1]
df3=df[df.clusters==2]
plt.scatter(df1.Age,df1['Income($)'],color='green')
plt.scatter(df2.Age,df2['Income($)'],color='red')
plt.scatter(df3.Age,df3['Income($)'],color='black')
plt.scatter(km.cluster_centers_[:,0],km.cluster_centers_[:,1],color='purple',marker='*',label='centroid')
plt.legend()
```

Out[29]: <matplotlib.legend.Legend at 0x201232ab910>

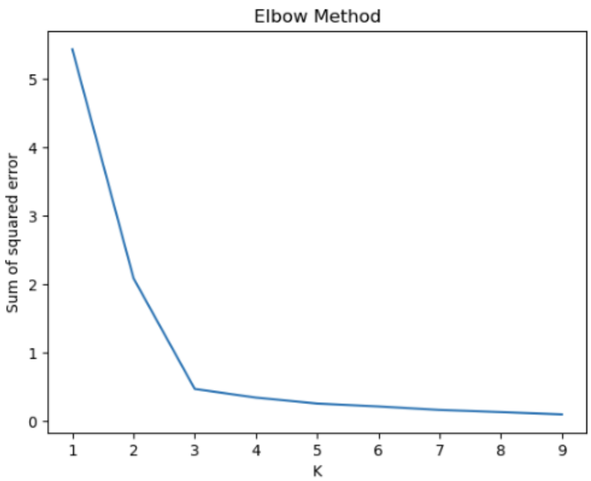


In [30]:
```
sse=[]
k_rng=range(1,10)
for k in k_rng:
    km=KMeans(n_clusters=k)
    km.fit(df[['Age','Income($)']])
    sse.append(km.inertia_)
```

```
C:\Users\LAB2_41\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning: KMeans is known to have a memory lea
k on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\LAB2_41\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning: KMeans is known to have a memory lea
k on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\LAB2_41\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning: KMeans is known to have a memory lea
k on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\LAB2_41\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning: KMeans is known to have a memory lea
k on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\LAB2_41\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1334: UserWarning: KMeans is known to have a memory lea
k on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable
```

In [32]:
```
plt.xlabel('K')
plt.ylabel('Sum of squared error')
plt.plot(k_rng,sse)
plt.title('Elbow Method')
```

Out[32]: `Text(0.5, 1.0, 'Elbow Method')`



ROLL NO:102
FYMCA DIV: B

Practical No: 10
Aim: Implementation Of Support Vector Machine.

Output:

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

[3]:
```
credit_df=pd.read_csv(r'C:\Users\nisha\OneDrive\Desktop\SEM 2\AIML LAB\CreditRisk.csv')
credit_df.head()
```

[3]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property
0	LP001002	Male	No	0	Graduate	No	5849	0.0	0	360.0	1.0	
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128	360.0	1.0	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66	360.0	1.0	
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120	360.0	1.0	
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141	360.0	1.0	

[4]:
```
credit_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Loan_ID            614 non-null    object
 1   Gender             601 non-null    object
 2   Married            611 non-null    object
 3   Dependents         599 non-null    object
 4   Education          614 non-null    object
 5   Self_Employed      582 non-null    object
 6   ApplicantIncome    614 non-null    int64
 7   CoapplicantIncome  614 non-null    float64
 8   LoanAmount         614 non-null    int64
 9   Loan_Amount_Term   600 non-null    float64
 10  Credit_History     564 non-null    float64
 11  Property_Area      614 non-null    object
 12  Loan_Status        614 non-null    int64
dtypes: float64(3), int64(3), object(7)
memory usage: 62.5+ KB
```

[5]:
```
credit_df.Loan_Status.value_counts()
```

[5]:
```
Loan_Status
1    422
0    192
Name: count, dtype: int64
```

[6]:
```
credit_df.describe()
```

[6]:

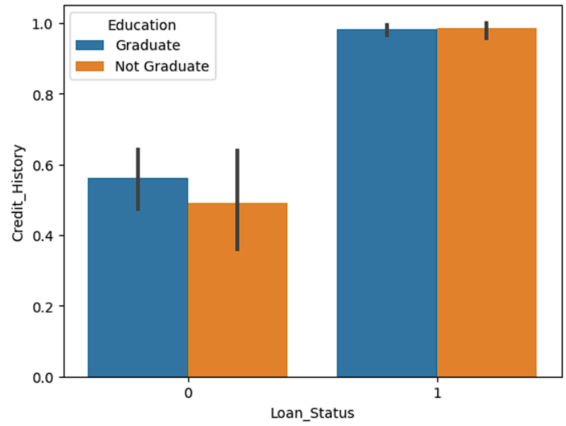
	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Loan_Status
count	614.000000	614.000000	614.000000	600.00000	564.000000	614.000000
mean	5403.459283	1621.245798	141.166124	342.00000	0.842199	0.687296
std	6109.041673	2926.248369	88.340630	65.12041	0.364878	0.463973
min	150.000000	0.000000	0.000000	12.00000	0.000000	0.000000
25%	2877.500000	0.000000	98.000000	360.00000	1.000000	0.000000
50%	3812.500000	1188.500000	125.000000	360.00000	1.000000	1.000000
75%	5795.000000	2297.250000	164.750000	360.00000	1.000000	1.000000
max	81000.000000	41667.000000	700.000000	480.00000	1.000000	1.000000

[7]:
```
credit_df.groupby(['Education','Loan_Status']).Education.count()
```

[7]:
```
Education     Loan_Status
Graduate      0              140
              1              340
Not Graduate  0               52
              1               82
Name: Education, dtype: int64
```

[8]:
```
sns.barplot(y='Credit_History', x='Loan_Status', hue='Education',data = credit_df)
```

[8]: <Axes: xlabel='Loan_Status', ylabel='Credit_History'>



[9]:
```
100 * credit_df.isnull().sum() / credit_df.shape[0]
```

[9]:
```
Loan_ID              0.000000
Gender               2.117264
Married              0.488599
Dependents           2.442997
Education            0.000000
Self_Employed        5.211726
ApplicantIncome      0.000000
CoapplicantIncome    0.000000
LoanAmount           0.000000
Loan_Amount_Term     2.280130
Credit_History       8.143322
Property_Area        0.000000
Loan_Status          0.000000
dtype: float64
```

[10]:
```
DF = credit_df.drop('Loan_ID', axis = 1)
```

[11]:
```
object_columns = DF.select_dtypes(include =['object']).columns
numeric_columns = DF.select_dtypes(exclude =['object']).columns
```

[12]:
```
for column in object_columns:
    majority = DF[column].value_counts().iloc[0]
    DF[column].fillna(majority, inplace=True)
```

[13]:
```
for column in numeric_columns:
    mean =  DF[column].mean()
    DF[column].fillna(mean, inplace=True)
```

[14]:
```
mean
```

[14]:
```
0.6872964169381107
```

[15]:
```
DF.head()
```

[15]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loa
0	Male	No	0	Graduate	No	5849	0.0	0	360.0	1.0	Urban	
1	Male	Yes	1	Graduate	No	4583	1508.0	128	360.0	1.0	Rural	
2	Male	Yes	0	Graduate	Yes	3000	0.0	66	360.0	1.0	Urban	
3	Male	Yes	0	Not Graduate	No	2583	2358.0	120	360.0	1.0	Urban	
4	Male	No	0	Graduate	No	6000	0.0	141	360.0	1.0	Urban	

[16]:
```
DF[object_columns].Married
```

[16]:
```
0       No
1      Yes
2      Yes
3      Yes
4       No
      ...
609     No
610    Yes
611    Yes
612    Yes
613     No
Name: Married, Length: 614, dtype: object
```

[17]:
```
DF[object_columns].Property_Area.head()
```

[17]:
```
DF[object_columns].Property_Area.head()
```

[17]:
```
0    Urban
1    Rural
2    Urban
3    Urban
4    Urban
Name: Property_Area, dtype: object
```

[18]:
```
DF_dummy = pd.get_dummies(DF, columns = object_columns)
```

[19]:
```
DF_dummy.shape
```

[19]:
```
(614, 25)
```

[20]:
```
DF_dummy.head()
```

[20]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Loan_Status	Gender_489	Gender_Female	Gender_Male	Married_398	...	De
0	5849	0.0	0	360.0	1.0	1	False	False	True	False	...	
1	4583	1508.0	128	360.0	1.0	0	False	False	True	False	...	
2	3000	0.0	66	360.0	1.0	1	False	False	True	False	...	
3	2583	2358.0	120	360.0	1.0	1	False	False	True	False	...	
4	6000	0.0	141	360.0	1.0	1	False	False	True	False	...	

5 rows × 25 columns

[21]:
```
from sklearn.model_selection import train_test_split as TTS
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

[22]:
```
X = DF_dummy.drop('Loan_Status', axis=1)
Y = DF_dummy.Loan_Status
train_x, test_x, train_y, test_y = TTS(X, Y, test_size = 0.3, random_state=42)
```

[23]:
```
train_x.shape, test_x.shape
```

[23]:
```
((429, 24), (185, 24))
```

[24]:
```
svm_model = SVC(kernel='rbf', gamma=0.00001,C=1000)
```

[25]:
```
svm_model.fit(train_x, train_y)
```

[25]:
```
▾          SVC
SVC(C=1000, gamma=1e-05)
```

[26]:
```
train_y_hat = svm_model.predict(train_x)
test_y_hat = svm_model.predict(test_x)
```

[27]:
```
print('-'*20, 'Train','-'*20)
print(classification_report(train_y, train_y_hat))
print('-'*20, 'Test', '-'*20)
print(classification_report(test_y, test_y_hat))
```

```
-------------------- Train --------------------
              precision    recall  f1-score   support

           0       0.95      0.95      0.95       127
           1       0.98      0.98      0.98       302

    accuracy                           0.97       429
   macro avg       0.96      0.96      0.96       429
weighted avg       0.97      0.97      0.97       429

-------------------- Test --------------------
              precision    recall  f1-score   support

           0       0.36      0.18      0.24        65
           1       0.65      0.82      0.73       120

    accuracy                           0.60       185
   macro avg       0.51      0.50      0.49       185
weighted avg       0.55      0.60      0.56       185
```

[28]:
```
confusion_matrix(train_y, train_y_hat)
```

[28]:
```
array([[121,   6],
       [  7, 295]], dtype=int64)
```

[ ]:

ROLL NO:102
FYMCA DIV: B

Practical No: 11

Aim: Program to Implement Decision Tree

Output:

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [3]:
```
credit_df=pd.read_csv(r"C:\Users\USER\Documents\2155aiml\CreditRisk.csv")
credit_df.head()
```

Out[3]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
0	LP001002	Male	No	0	Graduate	No	5849	0.0	0	360.0	1.0
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128	360.0	1.0
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66	360.0	1.0
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120	360.0	1.0
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141	360.0	1.0

In [5]:
```
credit_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Loan_ID            614 non-null    object 
 1   Gender             601 non-null    object 
 2   Married            611 non-null    object 
 3   Dependents         599 non-null    object 
 4   Education          614 non-null    object 
 5   Self_Employed      582 non-null    object 
 6   ApplicantIncome    614 non-null    int64  
 7   CoapplicantIncome  614 non-null    float64
 8   LoanAmount         614 non-null    int64  
 9   Loan_Amount_Term   600 non-null    float64
 10  Credit_History     564 non-null    float64
 11  Property_Area      614 non-null    object 
 12  Loan_Status        614 non-null    int64  
dtypes: float64(3), int64(3), object(7)
memory usage: 62.5+ KB
```

In [7]:
```
credit_df.Loan_Status.value_counts()
```

Out[7]:
```
Loan_Status
1    422
0    192
Name: count, dtype: int64
```

In [8]:
```
credit_df.describe()
```

Out[8]:

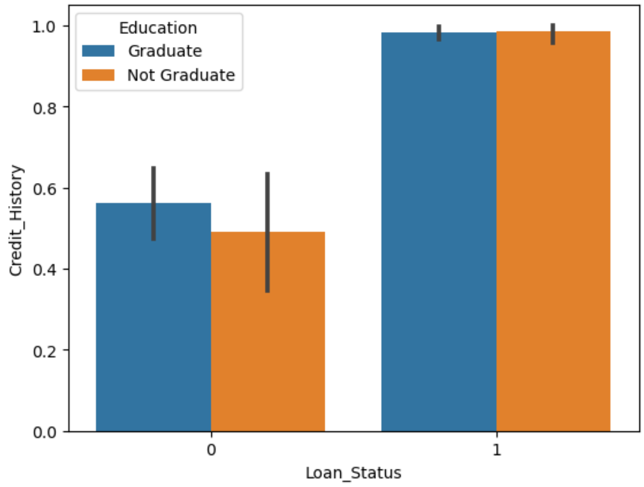
	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Loan_Status
count	614.000000	614.000000	614.000000	600.00000	564.000000	614.000000
mean	5403.459283	1621.245798	141.166124	342.00000	0.842199	0.687296
std	6109.041673	2926.248369	88.340630	65.12041	0.364878	0.463973
min	150.000000	0.000000	0.000000	12.00000	0.000000	0.000000
25%	2877.500000	0.000000	98.000000	360.00000	1.000000	0.000000
50%	3812.500000	1188.500000	125.000000	360.00000	1.000000	1.000000
75%	5795.000000	2297.250000	164.750000	360.00000	1.000000	1.000000
max	81000.000000	41667.000000	700.000000	480.00000	1.000000	1.000000

In [10]:
```
credit_df.groupby(['Education','Loan_Status']).Education.count()
```

Out[10]:
```
Education     Loan_Status
Graduate      0              140
              1              340
Not Graduate  0               52
              1               82
Name: Education, dtype: int64
```

In [11]:
```
sns.barplot(y='Credit_History',x='Loan_Status',hue='Education',data=credit_df)
```

Out[11]: <Axes: xlabel='Loan_Status', ylabel='Credit_History'>



In [12]:
```
100*credit_df.isnull().sum()/credit_df.shape[0]
```

Out[12]:
```
Loan_ID              0.000000
Gender               2.117264
Married              0.488599
Dependents           2.442997
Education            0.000000
Self_Employed        5.211726
ApplicantIncome      0.000000
CoapplicantIncome    0.000000
LoanAmount           0.000000
Loan_Amount_Term     2.280130
Credit_History       8.143322
Property_Area        0.000000
Loan_Status          0.000000
dtype: float64
```

In [13]:
```
DF = credit_df.drop('Loan_ID', axis = 1)
object_columns = DF.select_dtypes(include =['object']).columns
numeric_columns = DF.select_dtypes(exclude =['object']).columns
for column in object_columns:
    majority = DF[column].value_counts().iloc[0]
    DF[column].fillna(majority, inplace=True)
for column in numeric_columns:
    mean = DF[column].mean()
    DF[column].fillna(mean, inplace=True)
```

In [14]:
```
mean
```

```
In [15]: DF.head()
```

Out[15]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Ai
0	Male	No	0	Graduate	No	5849	0.0	0	360.0	1.0	Urt
1	Male	Yes	1	Graduate	No	4583	1508.0	128	360.0	1.0	Rι
2	Male	Yes	0	Graduate	Yes	3000	0.0	66	360.0	1.0	Urt
3	Male	Yes	0	Not Graduate	No	2583	2358.0	120	360.0	1.0	Urt
4	Male	No	0	Graduate	No	6000	0.0	141	360.0	1.0	Urt

```
In [16]: DF[object_columns].Married
```

```
Out[16]: 0       No
1       Yes
2       Yes
3       Yes
4        No
       ...
609      No
610     Yes
611     Yes
612     Yes
613      No
Name: Married, Length: 614, dtype: object
```

```
In [17]: DF[object_columns].Property_Area.head()
```

```
Out[17]: 0    Urban
1    Rural
2    Urban
3    Urban
4    Urban
Name: Property_Area, dtype: object
```

```
In [18]: DF_dummy = pd.get_dummies(DF, columns = object_columns)
DF_dummy.shape
```

```
Out[18]: (614, 25)
```

```
In [19]: DF_dummy.head()
```

Out[19]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Loan_Status	Gender_489	Gender_Female	Gender_Male	Married_398
0	5849	0.0	0	360.0	1.0	1	False	False	True	False
1	4583	1508.0	128	360.0	1.0	0	False	False	True	False
2	3000	0.0	66	360.0	1.0	1	False	False	True	False
3	2583	2358.0	120	360.0	1.0	1	False	False	True	False
4	6000	0.0	141	360.0	1.0	1	False	False	True	False

5 rows × 25 columns

```
In [20]: from sklearn.model_selection import train_test_split as TTS
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

```
In [24]: X = DF_dummy.drop('Loan_Status',axis=1)
Y = DF_dummy.Loan_Status
train_x, test_x, train_y, test_y=TTS(X,Y,test_size=0.3,random_state=42)
```

```
In [25]: train_x.shape, test_x.shape
Out[25]: ((429, 24), (185, 24))

In [28]: from sklearn.tree import DecisionTreeClassifier
         dt_model = DecisionTreeClassifier(max_depth=14)

In [29]: dt_model.fit(train_x, train_y)
Out[29]: ▾ DecisionTreeClassifier
         DecisionTreeClassifier(max_depth=14)

In [30]: train_y_hat = dt_model.predict(train_x)
         test_y_hat = dt_model.predict(test_x)

In [31]: print('-'*20,'Train','-'*20)
         print(classification_report(train_y,train_y_hat))
         print('-'*20,'Test','-'*20)
         print(classification_report(test_y,test_y_hat))
```

```
-------------------- Train --------------------
              precision    recall  f1-score   support

           0       0.99      1.00      1.00       127
           1       1.00      1.00      1.00       302

    accuracy                           1.00       429
   macro avg       1.00      1.00      1.00       429
weighted avg       1.00      1.00      1.00       429

-------------------- Test --------------------
              precision    recall  f1-score   support

           0       0.56      0.54      0.55        65
           1       0.76      0.78      0.77       120

    accuracy                           0.69       185
   macro avg       0.66      0.66      0.66       185
weighted avg       0.69      0.69      0.69       185
```

```
In [32]: confusion_matrix(train_y, train_y_hat)
Out[32]: array([[127,   0],
                [  1, 301]], dtype=int64)

In [ ]:
```

ROLL NO:102
FYMCA DIV: B

Practical No: 12

Aim: Program to Implement Random Forest

Output:

In [3]:
```
import numpy as np
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
```

In [5]:
```
train = pd.read_csv(r"D:\Siddharth_63\AIML\titanic - titanic.csv")
print(train.shape)
```

```
(891, 12)
```

In [6]:
```
#checking for missing data
NAs = pd.concat([train.isnull().sum()], axis=1, keys=["Train"])
NAs[NAs.sum(axis=1) > 0]
```

Out[6]:

	Train
Age	177
Cabin	687
Embarked	2

In [7]:
```
train.pop("Cabin")
train.pop("Name")
train.pop("Ticket")
```

Out[7]:
```
0             A/5 21171
1              PC 17599
2      STON/O2. 3101282
3                113803
4                373450
             ...       
886              211536
887              112053
888          W./C. 6607
889              111369
890              370376
Name: Ticket, Length: 891, dtype: object
```

In [8]:
```
# Filling missing Age values with mean
train["Age"] = train["Age"].fillna(train["Age"].mean())
```

In [9]:
```
# Filling missing Embarked values with most common values
train["Embarked"] = train["Embarked"].fillna(train["Embarked"].mode()[0])
```

In [10]:
```
train["Pclass"] = train["Pclass"].apply(str)
```

In [11]:
```
# Getting Dummies from all other categories vars
for col in train.dtypes[train.dtypes == "object"].index:
    for_dummy = train.pop(col)
    train = pd.concat([train, pd.get_dummies(for_dummy, prefix=col)], axis=1)
train.head()
```

Out[11]:

	PassengerId	Survived	Age	SibSp	Parch	Fare	Pclass_1	Pclass_2	Pclass_3	Sex_female	Sex_male	Embarked_C	Embarked_Q	Embarked_S
0	1	0	22.0	1	0	7.2500	0	0	1	0	1	0	0	1
1	2	1	38.0	1	0	71.2833	1	0	0	1	0	1	0	0
2	3	1	26.0	0	0	7.9250	0	0	1	1	0	0	0	1
3	4	1	35.0	1	0	53.1000	1	0	0	1	0	0	0	1
4	5	0	35.0	0	0	8.0500	0	0	1	0	1	0	0	1

In [12]:
```
labels = train.pop("Survived")
```

In [13]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(train, labels, test_size=0.25)
```

In [14]:
```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators=100)
rf.fit(x_train, y_train)
```

Out[14]:
```
▾ RandomForestClassifier
RandomForestClassifier()
```

In [15]:
```
y_pred = rf.predict(x_test)
```

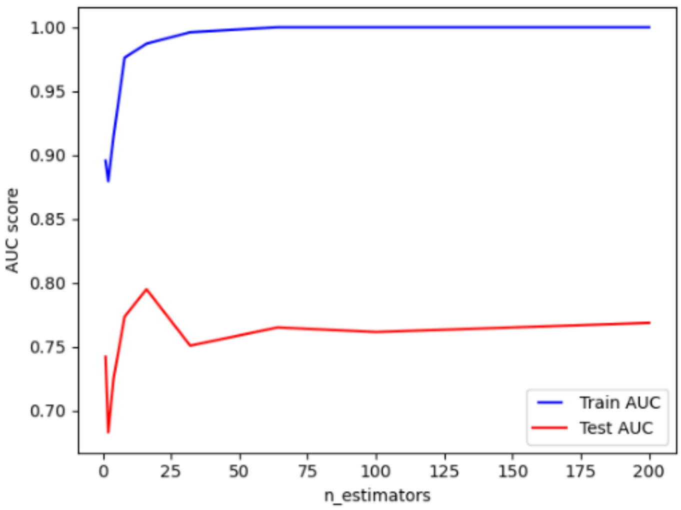
In [16]:
```
from sklearn.metrics import roc_curve, auc
false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, y_pred)
roc_auc = auc(false_positive_rate, true_positive_rate)
roc_auc
```

Out[16]: 0.798442760942761

In [17]:
```
n_estimators = [1, 2, 4, 8, 16, 32, 64, 100, 200]
train_results = []
test_results = []
```

In [18]:
```
for estimator in n_estimators:
    rf = RandomForestClassifier(n_estimators=estimator, n_jobs=-1)
    rf.fit(x_train, y_train)
    train_pred = rf.predict(x_train)
    false_positive_rate, true_positive_rate, thresholds = roc_curve(y_train, train_pred)
    roc_auc = auc(false_positive_rate, true_positive_rate)
    train_results.append(roc_auc)
    y_pred = rf.predict(x_test)
    false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, y_pred)
    roc_auc = auc(false_positive_rate, true_positive_rate)
    test_results.append(roc_auc)
```

In [20]:
```
from matplotlib.legend_handler import HandlerLine2D
line1, = plt.plot(n_estimators, train_results, "b", label="Train AUC")
line2, = plt.plot(n_estimators, test_results, "r", label="Test AUC")
plt.legend(handler_map={line1: HandlerLine2D(numpoints=2)})
plt.ylabel("AUC score")
plt.xlabel("n_estimators")
plt.show()
```



In [19]:
```
from sklearn.ensemble import RandomForestClassifier
rgf = RandomForestClassifier(n_estimators=200)
rf.fit(x_train, y_train)
```

Out[19]:
```
RandomForestClassifier
RandomForestClassifier(n_estimators=200, n_jobs=-1)
```

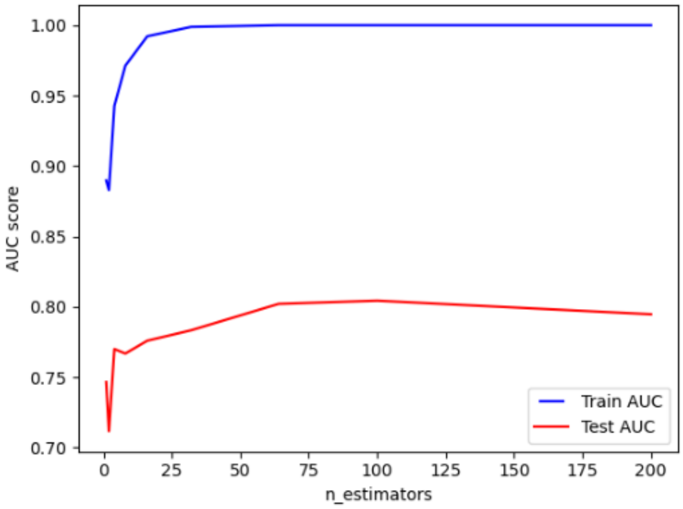
In [20]:
```
y_predict = rf.predict(x_test)
from sklearn.metrics import roc_curve, auc
false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, y_pred)
roc_auc = auc(false_positive_rate, true_positive_rate)
roc_auc
```

Out[20]: 0.7986952861952863

In [21]:
```
n_estimators = [1, 2, 4, 8, 16, 32, 64, 100, 200]
train_results = []
test_results = []
```

In [22]:
```
for estimator in n_estimators:
    rf = RandomForestClassifier(n_estimators=estimator, n_jobs=-1)
    rf.fit(x_train, y_train)
    train_pred = rf.predict(x_train)
    false_positive_rate, true_positive_rate, thresholds = roc_curve(y_train,train_pred)
    roc_auc = auc(false_positive_rate, true_positive_rate)
    train_results.append(roc_auc)
    y_pred = rf.predict(x_test)
    false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, y_pred)
    roc_auc = auc(false_positive_rate, true_positive_rate)
    test_results.append(roc_auc)
```

In [23]:
```
from matplotlib.legend_handler import HandlerLine2D
line1, = plt.plot(n_estimators, train_results, "b", label="Train AUC")
line2, = plt.plot(n_estimators, test_results, "r", label="Test AUC")
plt.legend(handler_map={line1: HandlerLine2D(numpoints=2)})
plt.ylabel("AUC score")
plt.xlabel("n_estimators")
plt.show()
```



Practical No: 13

Aim: Program to Implement AdaBoost

Output:

```
In [17]: from sklearn.ensemble import AdaBoostClassifier
         from sklearn import datasets
         from sklearn.model_selection import train_test_split
         from sklearn import metrics
```

```
In [18]: #Load data
         iris = datasets.load_iris()
         X = iris.data
         y = iris.target
```

```
In [19]: # Split dataset into training set and test set
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
         # 80% training and 20% test
```

```
In [20]: # Create adaboost classifer object
         AdaModel = AdaBoostClassifier(n_estimators=100,learning_rate=1)

         # Train Adaboost Classifer
         model = AdaModel.fit(X_train, y_train)

         # Predict the response for test dataset
         y_pred = model.predict(X_test)
```

```
In [21]: # Model Accuracy, how often is the classifier correct?
         print("Accuracy:",metrics.accuracy_score(y_test, y_pred))

         Accuracy: 0.9333333333333333
```



```
In [22]: # Import Support Vector Classifier
         from sklearn.svm import SVC
         #Import scikit-learn metrics module for accuracy calculation
         from sklearn import metrics
         svc=SVC(probability=True, kernel='linear')
         # Create adaboost classifer object
         abc =AdaBoostClassifier(n_estimators=50, base_estimator=svc,learning_rate=1)
```

```
In [23]: # Train Adaboost Classifer
         model = abc.fit(X_train, y_train)

         #Predict the response for test dataset
         y_pred = model.predict(X_test)
```

```
In [24]: # Model Accuracy, how often is the classifier correct?
         print("Accuracy:",metrics.accuracy_score(y_test, y_pred))

         Accuracy: 1.0
```

ROLL NO:102
FYMCA DIV: B

Practical No: 14

Aim: Program to Implement Gradient Boosting

Output:

In [1]:
```
#Importing neccesary packages
# Load libraries
from sklearn.ensemble import GradientBoostingRegressor
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.datasets import load_boston
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import r2_score
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
# Load data - Reading Boston Data
boston = load_boston()
X = pd.DataFrame(boston.data, columns=boston.feature_names) #Independent columns

y = pd.Series(boston.target) #Dependent column - Median value of House
```

In [3]:
```
#Viewing Data - predictors
X.head()
```

Out[3]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

In [4]:
```
y[1:10] #response
```

Out[4]:
```
1    21.6
2    34.7
3    33.4
4    36.2
5    28.7
6    22.9
7    27.1
8    16.5
9    18.9
dtype: float64
```

In [5]:
```
# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2) # 80% training and 20% test
```

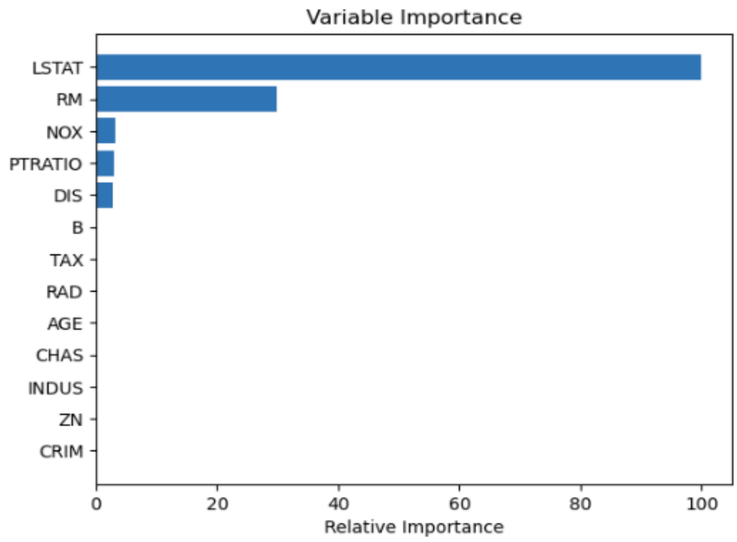
In [6]:
```
# Create gradientboost REGRESSOR object
gradientregressor = GradientBoostingRegressor(max_depth=2,n_estimators=3,learning_rate=1.0)
```

In [7]:
```
# Train gradientboost REGRESSOR
model = gradientregressor.fit(X_train, y_train)

#Predict the response for test dataset
y_pred = model.predict(X_test)
```

In [8]:
```
r2_score(y_pred,y_test)
```

Out[8]:
```
0.4839117761877929
```

In [9]:
```
import matplotlib.pyplot as plt
%matplotlib inline

# Plot feature importance
feature_importance = model.feature_importances_

# make importances relative to max importance
feature_importance = 100.0 * (feature_importance / feature_importance.max())
sorted_idx = np.argsort(feature_importance)
pos = np.arange(sorted_idx.shape[0]) + .5
plt.barh(pos, feature_importance[sorted_idx], align='center')
plt.yticks(pos, boston.feature_names[sorted_idx])
plt.xlabel('Relative Importance')
plt.title('Variable Importance')
plt.show()
```



In [10]:
```
from sklearn.model_selection import GridSearchCV
LR = {'learning_rate':[0.15,0.1,0.10,0.05], 'n_estimators':[100,150,200,250]}

tuning = GridSearchCV(estimator =GradientBoostingRegressor(),
            param_grid = LR, scoring='r2')
tuning.fit(X_train,y_train)
tuning.best_params_, tuning.best_score_
```

Out[10]: ({'learning_rate': 0.1, 'n_estimators': 150}, 0.8643406323195935)