Bharati Vidyapeeth's

Institute of Management & Information Technology

C.B.D. Belapur, Navi Mumbai – 400614

Vision:

Providing high quality, innovative and value-based education in information technology to build competent professionals.

Mission

M1. Technical Skills: To provide solid technical foundation theoretically as well as practically capable of providing quality services to industry.

M2. Development: -Department caters to the needs of students through comprehensive educational programs and promotes lifelong learning in the field of computer Applications.

M3. Ethical leadership: Department develops ethical leadership insight in the students to succeed in industry, government and academia.

CERTIFICATE

This is to certify that the journal is the work of **Ms. Pratiksha Manoj Manjrekar** Roll No. **31** of MCA (Sem- 2 , Div: A) for the academic year 2022 - 2023

Subject Code: **MCAL21**

Subject Name: **Artificial Intelligence & Machine Learning Lab**

Subject-in-charge Principal

Date: 16/06/2023 External Examiner

INDEX

Name: Pratiksha Manjrekar **MCA Sem II** **Roll no.: A 31**

Sr No.	Topic	Sign
1	**Logic programming with Prolog**	
1.1	Implementation of Logic programming using PROLOG-DFS for water jug problem.	
2	**Introduction to Python Programming**	
2.1	Implement Basics of Python Programming.	
	Write a program to Implement ML NumPy library.	
2.2	Write a program to Implement ML Pandas library.	
	Write a program to Implement ML SciPy, Matplotlib, Scikit Learn library	
3	**Supervised Learning**	
3.1	Write a program to implement Linear Regression.	
3.2	Write a program to implement Logistic regression.	
	Write a program to implement KNN- classification.	
4	**Unsupervised Learning:**	
4.1	Write a program to implement K-Means clustering algorithm.	
4.2	Write a program to implement K-medoid clustering algorithm.	
5	**Classifying data using Support Vector Machines (SVMs)**	
5.1	Write a program to Classifying data using Support Vector Machines (SVMs).	
6	**Bagging Algorithm**	
6.1	Write a program to implement Bagging Algorithm Decision Tree.	
6.2	Write a program to implement Bagging Algorithm Random Forest.	
7	**Boosting Algorithms:**	
7.1	Write a program to implement Boosting Algorithms: AdaBoost .	
7.2	Write a program to implement Boosting Algorithms: Stochastic Gradient Boosting,.	
8	**Deployment of Machine Learning Models**	

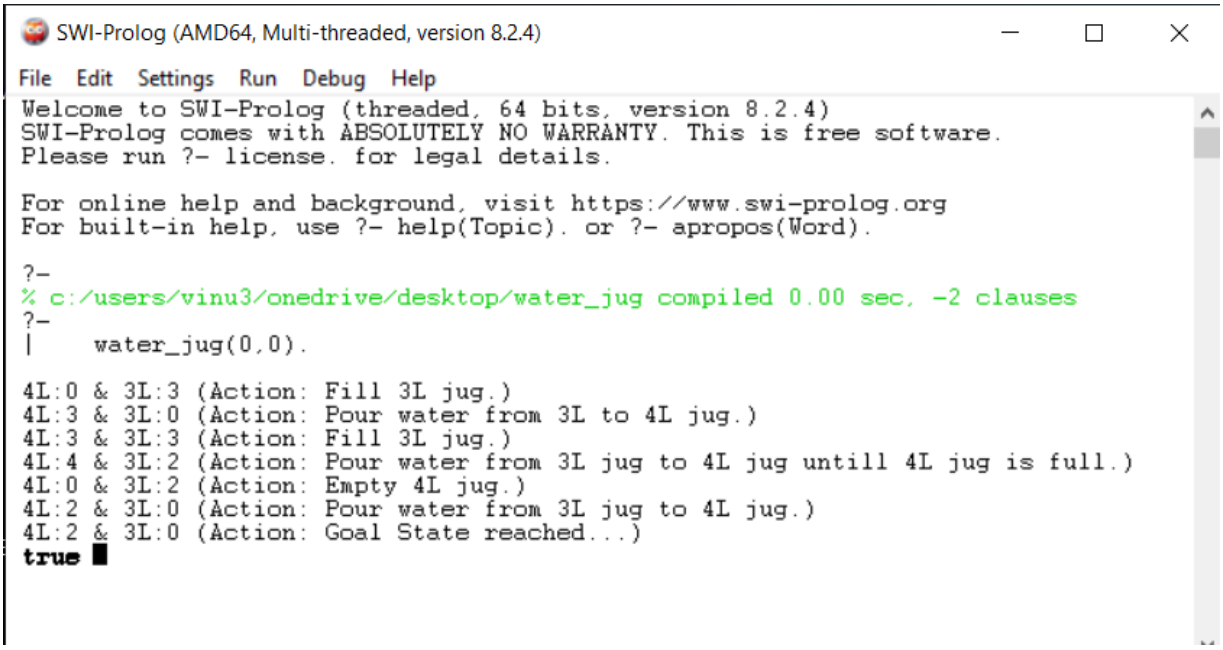
Practical No. 1
Logic programming with Prolog

Problem Statement 1.1: Implementation of Logic programming using PROLOG-DFS for water jug problem.

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 jugs 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 State 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 untill 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: Fill 4L jug.)'),XX is 4, water_jug(XX,Y));
                    (X=:=4, Y=:=1,nl,write('4L:2 & 3L:3 (Action: Pour water from 4L to 3L jug untill 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:

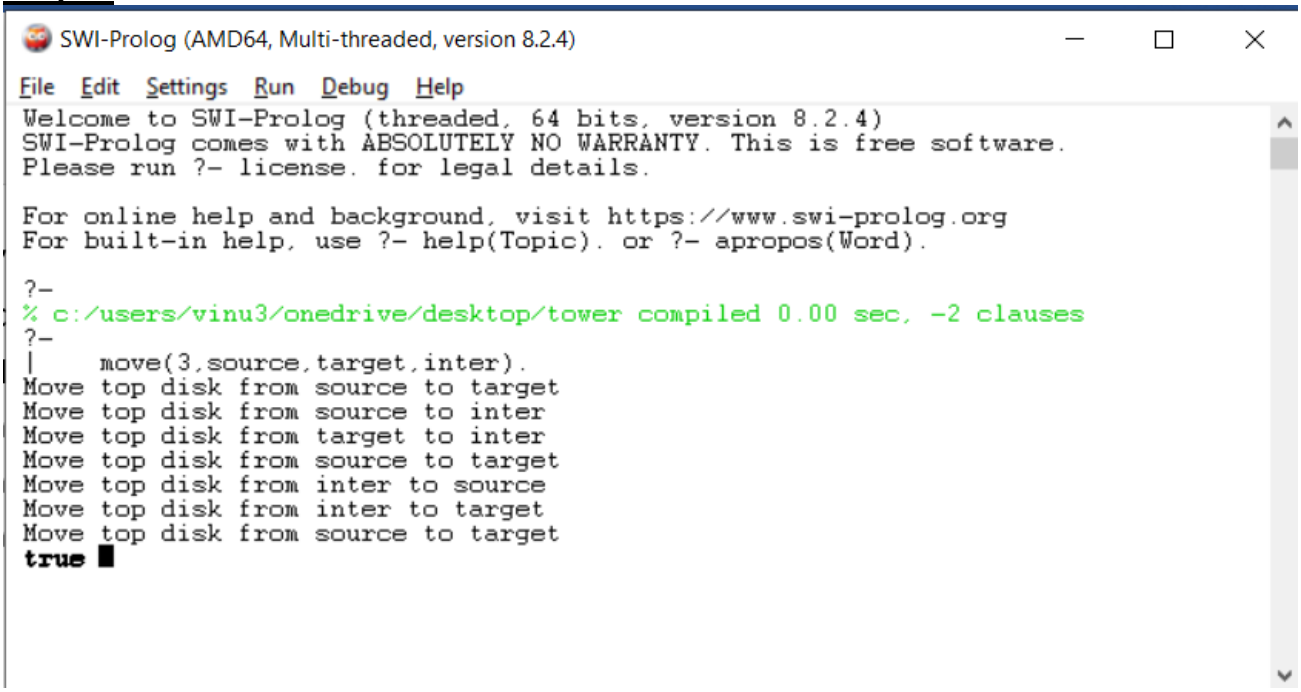

SWI-Prolog (AMD64, Multi-threaded, version 8.2.4)
File Edit Settings Run Debug Help
Welcome to SWI-Prolog (threaded, 64 bits, version 8.2.4)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.
For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).
?-
% c:/users/vinu3/onedrive/desktop/water_jug compiled 0.00 sec, -2 clauses
?-
| water_jug(0,0).
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 State reached...)
true

Problem Statement 1.2: Implementation of Logic programming using PROLOG-DFS for Tower of Hanoi problem.

Code:

```
move(1,X,Y,_):-
        write('Move top disk from')
        write(X)
        write(' to ')
        write(Y),
        nl.
move(N,X,Y,Z):-
        N>1,
        M is N-1,
        move(M,X,Z,Y),
        move(1,X,Y,_),
        move(M,Z,Y,X).
```

Output:



Practical No. 2
Introduction to Python Programming

Problem Statement 2.1 : Implement Basics of Python Programming.
Code:

Basics

In [4]:
```
id=[1,2,3,4]
emp_name=["Ram","Pratiksha","Sita","John"]
num_emp=4
```

In [5]:
```
emp_list=[id,emp_name,num_emp]
print(emp_list)
```

```
[[1, 2, 3, 4], ['Ram', 'Pratiksha', 'Sita', 'John'], 4]
```

In [6]:
```
print(emp_list[0])
```

```
[1, 2, 3, 4]
```

In [7]:
```
print(emp_list[1][1])
```

```
Pratiksha
```

In [8]:
```
emp_list[2]=5
print(emp_list)
```

```
[[1, 2, 3, 4], ['Ram', 'Pratiksha', 'Sita', 'John'], 5]
```

In [10]:
```
emp_list[1][3]="Vinita"
print(emp_list)
```

```
[[1, 2, 3, 4], ['Ram', 'Pratiksha', 'Sita', 'Vinita'], 5]
```

In [11]:
```
emp_list[1].append('Lekha')
print(emp_list)
```

```
[[1, 2, 3, 4], ['Ram', 'Pratiksha', 'Sita', 'Vinita', 'Lekha'], 5]
```

In [12]:
```
emp_list.append([23,25,36,43,53])
print(emp_list)
```

```
[[1, 2, 3, 4], ['Ram', 'Pratiksha', 'Sita', 'Vinita', 'Lekha'], 5, [23, 25, 36, 43, 53]]
```

In [13]:
```
emp_list[0].insert(0,5)
print(emp_list)
```

```
[[5, 1, 2, 3, 4], ['Ram', 'Pratiksha', 'Sita', 'Vinita', 'Lekha'], 5, [23, 25, 36, 43, 53]]
```

In [15]:
```
del emp_list[3]
print(emp_list)
```

```
[[5, 1, 2, 3, 4], ['Ram', 'Pratiksha', 'Sita', 'Vinita', 'Lekha'], 5]
```

In [16]:
```
emp_list[1].remove("Ram")
print(emp_list)
```

```
[[5, 1, 2, 3, 4], ['Pratiksha', 'Sita', 'Vinita', 'Lekha'], 5]
```

In [17]:
```
salary=['high','low','medium','low']
```

In [18]:
```
salary.remove('low')
print(salary)
```

```
['high', 'medium', 'low']
```

In [19]:
```
emp_list[0].pop(4)
```

Out[19]: 4

Problem Statement 2.2: Write a program to Implement ML NumPy library-:

In [21]:
```
import numpy as np
```

In [22]:
```
x=np.array([2,3,4,5])
print(type(x))
```

```
<class 'numpy.ndarray'>
```

In [23]:
```
print(x)
```

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

In [24]:
```
x=np.array([2,3,'n',5])
print(x)
```

```
['2' '3' 'n' '5']
```

In [25]:
```
b=np.linspace(start=1,stop=5,num=10,endpoint=True,retstep=False)
print(b)
```

```
[1.         1.44444444 1.88888889 2.33333333 2.77777778 3.22222222
 3.66666667 4.11111111 4.55555556 5.        ]
```

In [26]:
```
c=np.linspace(start=1,stop=5,num=10,endpoint=True,retstep=True)
print(c)
```

```
(array([1.        , 1.44444444, 1.88888889, 2.33333333, 2.77777778,
       3.22222222, 3.66666667, 4.11111111, 4.55555556, 5.        ]), 0.4444444444444444)
```

In [28]:
```
d=np.arange(start=1,stop=10,step=2)
print(d)
```

```
[1 3 5 7 9]
```

In [29]:
```
i=np.ones((3,4))
print(i)
```

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

In [30]:
```
j=np.zeros((3,4))
print(j)
```

```
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
```

In [32]:
```
l=np.random.rand(5)
print(l)
```

```
[0.23303178 0.34801942 0.80458391 0.46611172 0.38411893]
```

In [33]:
```
p=np.random.rand(5,2)
print(p)
```

```
[[0.12520274 0.59794909]
 [0.9099957  0.7799197 ]
 [0.40141704 0.2507982 ]
 [0.42565038 0.71075717]
 [0.89288362 0.68388405]]
```

In [35]:
```
q=np.logspace(1,10,num=5,endpoint=True,base=10.0)
print(q)
```

```
[1.00000000e+01 1.77827941e+03 3.16227766e+05 5.62341325e+07
 1.00000000e+10]
```

In [36]:
```
import sys
x=range(1000)
sys.getsizeof(1)*len(x)
```

Out[36]:
```
28000
```

In [38]:
```
y=np.array(x)
y.itemsize*y.size
```

Out[38]:
```
4000
```

In [39]:
```
grid=np.arange(start=1,stop=10).reshape(3,3)
print(grid)
```

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

In [40]:
```
a=np.array([[1,2,3],[4,5,6],[7,8,9]])
a.shape
```

Out[40]:
```
(3, 3)
```

In [41]:
```
np.sum(a)
np.sum(a,axis=0)
```

Out[41]:
```
array([12, 15, 18])
```

In [42]:
```
a.sum()
```

Out[42]:
```
45
```

In [43]:
```
np.sum(a,axis=1)
```

Out[43]:
```
array([ 6, 15, 24])
```

In [44]:
```
b=np.arange(start=11,stop=20).reshape(3,3)
print(b)
```

```
[[11 12 13]
 [14 15 16]
 [17 18 19]]
```

In [45]:
```
print(a)
```

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

In [46]:
```
np.add(a,b)
```

Out[46]:
```
array([[12, 14, 16],
       [18, 20, 22],
       [24, 26, 28]])
```

In [47]:
```
np.multiply(a,b)
```

Out[47]:
```
array([[ 11,  24,  39],
       [ 56,  75,  96],
       [119, 144, 171]])
```

In [48]:
```
a[1:3]
```

Out[48]:
```
array([[4, 5, 6],
       [7, 8, 9]])
```

In [49]:
```
np.transpose(a)
```

Out[49]:
```
array([[1, 4, 7],
       [2, 5, 8],
       [3, 6, 9]])
```

In [50]:
```
a_row=np.append(a,[[10,11,14]],axis=0)
print(a_row)
```

```
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 14]]
```

In [51]:
```
col=np.array([21,22,23]).reshape(3,1)
```

In [52]:
```
a_col=np.append(a,col,axis=1)
print(a_col)
```

```
[[ 1  2  3 21]
 [ 4  5  6 22]
 [ 7  8  9 23]]
```

In [53]:
```
a_ins=np.insert(a,1,[13,15,16],axis=0)
print(a_ins)
```

```
[[ 1  2  3]
 [13 15 16]
 [ 4  5  6]
 [ 7  8  9]]
```

In [54]:
```
a_del=np.delete(a_ins,2,axis=0)
print(a_del)
```

```
[[ 1  2  3]
 [13 15 16]
 [ 7  8  9]]
```

In [56]:
```
print(x)
timeit=np.sum(y)
```

```
range(0, 1000)
```

Problem Statement 2.3: Write a program to Implement ML Pandas library :

First download the “mtcars.csv” dataset

In [58]:
```
import os
import pandas as pd
```

In [63]:
```
data1 = pd.read_csv('C:/Users/Pratiksha/Data Analytics/Datasets/mtcars.csv')
```

In [64]:
```
data1
```

Out[64]:

	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
6	Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
7	Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
8	Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
9	Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4
10	Merc 280C	17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4
11	Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.40	0	0	3	3
12	Merc 450SL	17.3	8	275.8	180	3.07	3.730	17.60	0	0	3	3
13	Merc 450SLC	15.2	8	275.8	180	3.07	3.780	18.00	0	0	3	3
14	Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250	17.98	0	0	3	4
15	Lincoln Continental	10.4	8	460.0	215	3.00	5.424	17.82	0	0	3	4
16	Chrysler Imperial	14.7	8	440.0	230	3.23	5.345	17.42	0	0	3	4
17	Fiat 128	32.4	4	78.7	66	4.08	2.200	19.47	1	1	4	1
18	Honda Civic	30.4	4	75.7	52	4.93	1.615	18.52	1	1	4	2
19	Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1
20	Toyota Corona	21.5	4	120.1	97	3.70	2.465	20.01	1	0	3	1
21	Dodge Challenger	15.5	8	318.0	150	2.76	3.520	16.87	0	0	3	2
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 [65]:
```
data1.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 [66]:
```
data1.head()
```

Out[66]:

	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

In [67]:
```
data1.size
```

Out[67]:
```
384
```

```
In [68]: data1.shape
Out[68]: (32, 12)

In [69]: data1.ndim
Out[69]: 2

In [70]: data1.at[4,'model']
Out[70]: 'Hornet Sportabout'

In [71]: data1.iat[4,3]
Out[71]: 360.0

In [72]: data1.loc[:,'model']
Out[72]: 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 [73]:
```
data1.loc[0:5,'model']
```

Out[73]:
```
0            Mazda RX4
1        Mazda RX4 Wag
2           Datsun 710
3       Hornet 4 Drive
4    Hornet Sportabout
5              Valiant
Name: model, dtype: object
```

In [74]:
```
data1.iloc[0:5,0:2]
```

Out[74]:

	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 [75]:
```
data1.iloc[0:10,0:10]
```

Out[75]:

	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

In [76]:
```
data1.dtypes
```

Out[76]:
```
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 [77]:
```
data1['model'].dtype
```

Out[77]:
```
dtype('O')
```

In [78]:
```
data1.axes
```

Out[78]:
```
[RangeIndex(start=0, stop=32, step=1),
 Index(['model', 'mpg', 'cyl', 'disp', 'hp', 'drat', 'wt', 'qsec', 'vs', 'am',
        'gear', 'carb'],
       dtype='object')]
```

In [79]:
```
data1.columns
```

Out[79]:
```
Index(['model', 'mpg', 'cyl', 'disp', 'hp', 'drat', 'wt', 'qsec', 'vs', 'am',
       'gear', 'carb'],
      dtype='object')
```

In [80]:
```
data1['hp'].std()
```

Out[80]:
```
68.56286848932059
```

In [81]:
```
data1['mpg'].mean()
```

Out[81]:
```
20.090624999999996
```

In [82]:
```
data1['mpg'].median()
```

Out[82]:
```
19.2
```

In [83]:
```
data1['hp'].describe()
```

Out[83]:
```
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
```

In [84]:
```
data1.head(15)
```

Out[84]:

	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
6	Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
7	Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
8	Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
9	Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4
10	Merc 280C	17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4
11	Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.40	0	0	3	3
12	Merc 450SL	17.3	8	275.8	180	3.07	3.730	17.60	0	0	3	3
13	Merc 450SLC	15.2	8	275.8	180	3.07	3.780	18.00	0	0	3	3
14	Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250	17.98	0	0	3	4

```
In [85]: data1.iloc[1]
```

```
Out[85]: model    Mazda RX4 Wag
         mpg                21.0
         cyl                   6
         disp              160.0
         hp                  110
         drat                3.9
         wt                2.875
         qsec              17.02
         vs                    0
         am                    1
         gear                  4
         carb                  4
         Name: 1, dtype: object
```

```
In [86]: data1.iloc[:,-1]
```

```
Out[86]: 0     4
         1     4
         2     1
         3     1
         4     2
         5     1
         6     4
         7     2
         8     2
         9     4
         10    4
         11    3
         12    3
         13    3
         14    4
         15    4
         16    4
         17    1
         18    2
         19    1
         20    1
         21    2
         22    2
         23    4
         24    2
         25    1
         26    2
         27    2
         28    4
         29    6
         30    8
         31    2
         Name: carb, dtype: int64
```

In [87]:
```
data1.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 [88]:
```
data1.iloc[-1]
```

Out[88]:
```
model    Volvo 142E
mpg            21.4
cyl               4
disp          121.0
hp              109
drat           4.11
wt             2.78
qsec           18.6
vs                1
am                1
gear              4
carb              2
Name: 31, dtype: object
```

In [89]:
```
data1.iloc[1]
```

Out[89]:
```
model    Mazda RX4 Wag
mpg               21.0
cyl                  6
disp             160.0
hp                 110
drat               3.9
wt               2.875
qsec             17.02
vs                   0
am                   1
gear                 4
carb                 4
Name: 1, dtype: object
```

In [90]:
```
data1_sorted=data1.sort_values(by='mpg')
```

In [91]:
```
data1_sorted.head()
```

Out[91]:

	model	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
15	Lincoln Continental	10.4	8	460.0	215	3.00	5.424	17.82	0	0	3	4
14	Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250	17.98	0	0	3	4
23	Camaro Z28	13.3	8	350.0	245	3.73	3.840	15.41	0	0	3	4
6	Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
16	Chrysler Imperial	14.7	8	440.0	230	3.23	5.345	17.42	0	0	3	4

In [92]:
```
data1[data1['carb']==1]
```

Out[92]:

	model	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
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
5	Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
17	Fiat 128	32.4	4	78.7	66	4.08	2.200	19.47	1	1	4	1
19	Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1
20	Toyota Corona	21.5	4	120.1	97	3.70	2.465	20.01	1	0	3	1
25	Fiat X1-9	27.3	4	79.0	66	4.08	1.935	18.90	1	1	4	1

In [93]:
```
data1[data1['carb']==1].count()
```

Out[93]:
```
model    7
mpg      7
cyl      7
disp     7
hp       7
drat     7
wt       7
qsec     7
vs       7
am       7
gear     7
carb     7
dtype: int64
```

In [94]:
```
data1.describe()
```

Out[94]:

	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	32.0000
mean	20.090625	6.187500	230.721875	146.687500	3.596563	3.217250	17.848750	0.437500	0.406250	3.687500	2.8125
std	6.026948	1.785922	123.938694	68.562868	0.534679	0.978457	1.786943	0.504016	0.498991	0.737804	1.6152
min	10.400000	4.000000	71.100000	52.000000	2.760000	1.513000	14.500000	0.000000	0.000000	3.000000	1.0000
25%	15.425000	4.000000	120.825000	96.500000	3.080000	2.581250	16.892500	0.000000	0.000000	3.000000	2.0000
50%	19.200000	6.000000	196.300000	123.000000	3.695000	3.325000	17.710000	0.000000	0.000000	4.000000	2.0000
75%	22.800000	8.000000	326.000000	180.000000	3.920000	3.610000	18.900000	1.000000	1.000000	4.000000	4.0000
max	33.900000	8.000000	472.000000	335.000000	4.930000	5.424000	22.900000	1.000000	1.000000	5.000000	8.0000

Practical 3
Supervised Learning

Problem Statement 3.1 Write a program to implement Linear Regression.
Linear Regression

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_boston
```

In [2]:
```
boston=load_boston()
```

In [3]:
```
boston.keys()
```

Out[3]:
```
dict_keys(['data', 'target', 'feature_names', 'DESCR', 'filename', 'data_module'])
```

In [4]:
```
bosDf=pd.DataFrame(boston['data'],columns=boston['feature_names'])
```

In [5]:
```
bosDf['Median_Price']=boston['target']
bosDf.head()
```

Out[5]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	Median_Price
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	24.0
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	21.6
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	34.7
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	33.4
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	36.2

In [6]:
```
bosDf.size
bosDf.info()
```

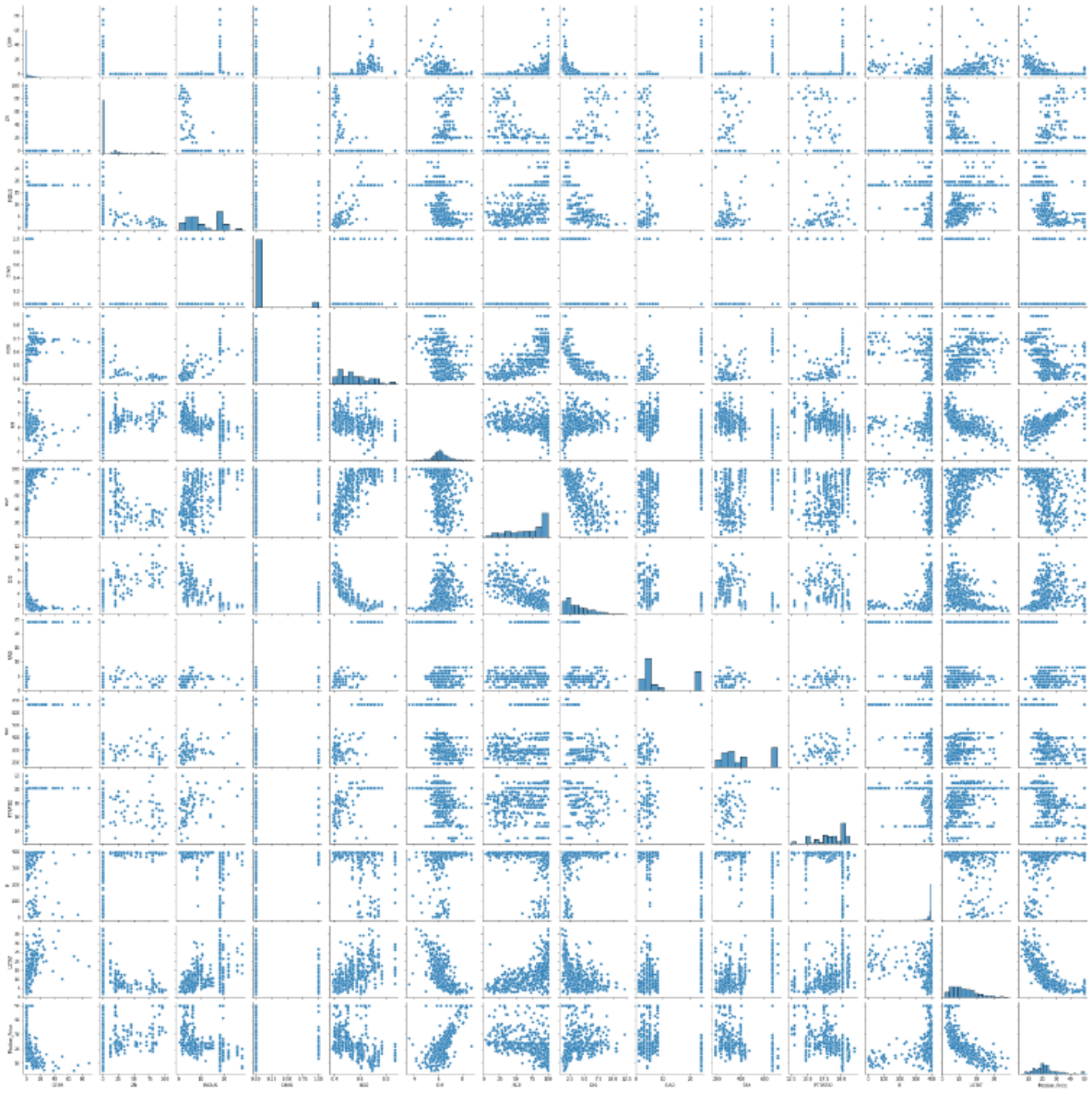
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   CRIM          506 non-null    float64
 1   ZN            506 non-null    float64
 2   INDUS         506 non-null    float64
 3   CHAS          506 non-null    float64
 4   NOX           506 non-null    float64
 5   RM            506 non-null    float64
 6   AGE           506 non-null    float64
 7   DIS           506 non-null    float64
 8   RAD           506 non-null    float64
 9   TAX           506 non-null    float64
 10  PTRATIO       506 non-null    float64
 11  B             506 non-null    float64
 12  LSTAT         506 non-null    float64
 13  Median_Price  506 non-null    float64
dtypes: float64(14)
memory usage: 55.5 KB
```

In [7]:
```
bosDf.shape
```

Out[7]:
```
(506, 14)
```

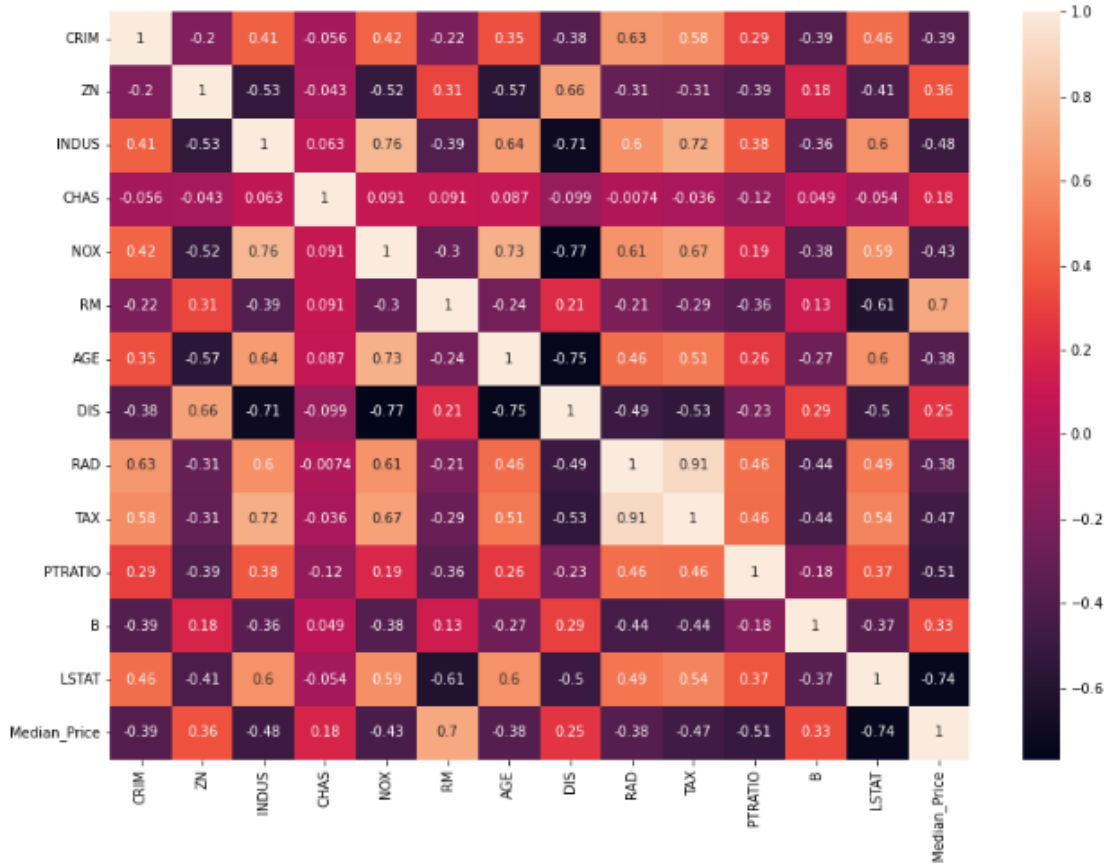
In [8]:
```
sns.pairplot(bosDf)
```

Out[8]:
```
<seaborn.axisgrid.PairGrid at 0x14bc00ac6d0>
```



In [10]:
```
plt.figure(figsize=(14,10))
sns.heatmap(bosDf.corr(),annot=True)
```

Out[10]: <AxesSubplot:>



In [11]:
```
from sklearn.model_selection import train_test_split as tts
from sklearn.linear_model import LinearRegression as LR
from sklearn.metrics import mean_squared_error
import math
```

In [12]:
```
X=bosDf.drop('Median_Price',axis=1)
Y=bosDf.Median_Price
```

In [13]:
```
train_X,test_X,train_Y,test_Y=tts(X,Y,test_size=0.3,random_state=42)
```

In [14]:
```
train_X.head()
```

Out[14]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
5	0.02985	0.0	2.18	0.0	0.458	6.430	58.7	6.0622	3.0	222.0	18.7	394.12	5.21
116	0.13158	0.0	10.01	0.0	0.547	6.176	72.5	2.7301	6.0	432.0	17.8	393.30	12.04
45	0.17142	0.0	6.91	0.0	0.448	5.682	33.8	5.1004	3.0	233.0	17.9	396.90	10.21
16	1.05393	0.0	8.14	0.0	0.538	5.935	29.3	4.4986	4.0	307.0	21.0	386.85	6.58
468	15.57570	0.0	18.10	0.0	0.580	5.926	71.0	2.9084	24.0	666.0	20.2	368.74	18.13

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

Out[15]:
```
5      28.7
116    21.2
45     19.3
16     23.1
468    19.1
Name: Median_Price, dtype: float64
```

In [16]:
```
Model=LR()
```

In [17]:
```
Model.fit(train_X,train_Y)
```

Out[17]:
```
LinearRegression()
```

In [18]:
```
Model.coef_
```

Out[18]:
```
array([-1.33470103e-01,  3.58089136e-02,  4.95226452e-02,  3.11983512e+00,
       -1.54170609e+01,  4.05719923e+00, -1.08208352e-02, -1.38599824e+00,
        2.42727340e-01, -8.70223437e-03, -9.10685208e-01,  1.17941159e-02,
       -5.47113313e-01])
```

In [21]:
```
Model.intercept_
```

Out[21]:
```
31.631084035693547
```

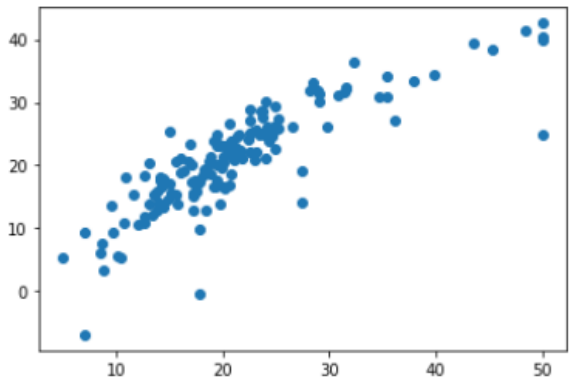
In [22]:
```
train_Y_hat=Model.predict(train_X)
test_Y_hat=Model.predict(test_X)
```

In [23]:
```
print('Train MSE',math.sqrt(mean_squared_error(train_Y,train_Y_hat)))
print('Test MSE',math.sqrt(mean_squared_error(test_Y,test_Y_hat)))
```

```
Train MSE 4.748208239685937
Test MSE 4.638689926172827
```

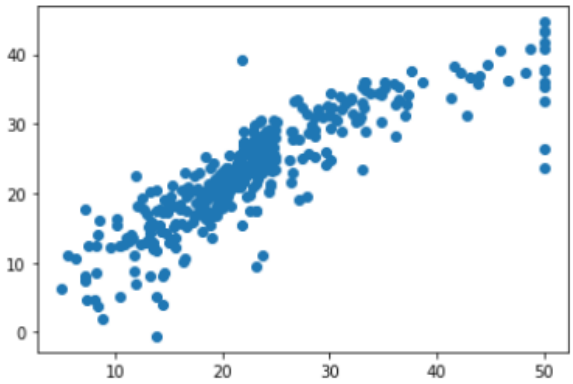
In [24]:
```
plt.scatter(test_Y,test_Y_hat)
```

Out[24]:
```
<matplotlib.collections.PathCollection at 0x14bcc8db5b0>
```



In [25]:
```
plt.scatter(train_Y,train_Y_hat)
```

Out[25]:
```
<matplotlib.collections.PathCollection at 0x14bcd55a640>
```



Problem Statement 3.2 Write a program to implement Logistic regression

In [26]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [30]:
```
credit_df=pd.read_csv('C:\\Users\\Pratiksha\\Data Analytics\\Datasets\\CreditRisk.csv')
credit_df.head()
```

Out[30]:

	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 [31]:
```
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 [32]:
```
credit_df.describe()
```

Out[32]:

	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 [33]:
```
credit_df.Loan_Status.value_counts()
```

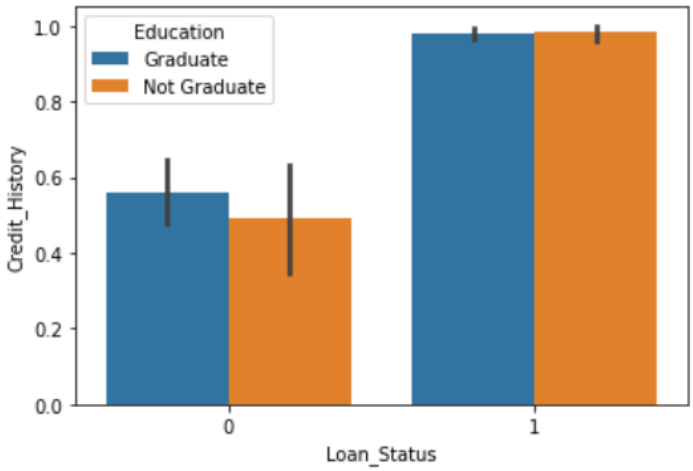
Out[33]:
```
1    422
0    192
Name: Loan_Status, dtype: int64
```

In [34]:
```
credit_df.groupby(['Education','Loan_Status']).Education.count()
```

Out[34]:
```
Education     Loan_Status
Graduate      0              140
              1              340
Not Graduate  0               52
              1               82
Name: Education, dtype: int64
```

In [35]:
```
sns.barplot(y='Credit_History',x='Loan_Status',hue='Education',data=credit_df)
```

Out[35]:
```
<AxesSubplot:xlabel='Loan_Status', ylabel='Credit_History'>
```



In [36]:
```
100*credit_df.isnull().sum()/credit_df.shape[0]
```

Out[36]:
```
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 [37]:
```
DF=credit_df.drop('Loan_ID',axis=1)
```

In [38]:
```
object_columns=DF.select_dtypes(include=['object']).columns
```

In [39]:
```
numeric_columns=DF.select_dtypes(exclude=['object']).columns
```

In [41]:
```
# A technique for filling missing values

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

In [42]:
```
for column in numeric_columns:
    mean=DF[column].mean()
    DF[column].fillna(mean,inplace=True)
```

In [43]:
```
DF.head()
```

Out[43]:

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

In [44]:
```
credit_df.head()
```

Out[44]:

	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 [45]:
```
DF[object_columns].Property_Area
```

Out[45]:
```
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 [46]:
```
DF[object_columns].Property_Area.head()
```

Out[46]:
```
0    Urban
1    Rural
2    Urban
3    Urban
4    Urban
Name: Property_Area, dtype: object
```

In [47]:
```
DF_dummy = pd.get_dummies(DF,columns = object_columns)
```

In [48]:
```
DF_dummy.shape
```

Out[48]:
```
(614, 25)
```

In [49]:
```
DF_dummy.head()
```

Out[49]:

	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 [50]:
```
from sklearn.model_selection import train_test_split as TTS
from sklearn.linear_model import LogisticRegression as LoR
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
```

In [51]:
```
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 [52]:
```
train_x.shape,test_x.shape
```

Out[52]: ((429, 24), (185, 24))

In [53]:
```
model = LoR()
```

In [54]:
```
model.fit(train_x,train_y)
```

Out[54]: LogisticRegression()

In [56]:
```
train_y_hat = model.predict(train_x)
test_y_hat = model.predict(test_x)
```

In [57]:
```
print('train accuracy',accuracy_score(train_y,train_y_hat))
print('train accuracy',accuracy_score(test_y,test_y_hat))
```

```
train accuracy 0.8205128205128205
train accuracy 0.7837837837837838
```

In [58]:
```
print(confusion_matrix(train_y,train_y_hat))
```

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

In [59]:
```
test_y.value_counts()
```

Out[59]:
```
1    120
0     65
Name: Loan_Status, dtype: int64
```

In [60]:
```
pd.Series(test_y_hat).value_counts()
```

Out[60]:
```
1    156
0     29
dtype: int64
```

In [61]:
```
(57+295)/train_y.shape[0]
```

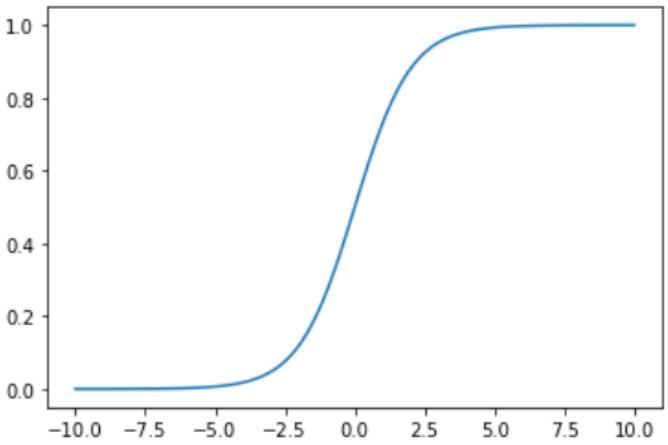
Out[61]: 0.8205128205128205

In [62]:
```
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 [64]:
```
x=np.linspace(-10,10,100)
y=1/(1+np.exp(-x))#sigmoid
plt.plot(x,y)
```

Out[64]:
```
[<matplotlib.lines.Line2D at 0x14bd05bb160>]
```



In [65]:
```
test_y_hat_5=(model.predict_proba(test_x)[:,1]>0.5).astype(int)
test_y_hat_7=(model.predict_proba(test_x)[:,1]>0.7).astype(int)
test_y_hat_3=(model.predict_proba(test_x)[:,1]>0.3).astype(int)
```

In [67]:
```
print(confusion_matrix(test_y,test_y_hat_5))
print(confusion_matrix(test_y,test_y_hat_7))
print(confusion_matrix(test_y,test_y_hat_3))
```

```
[[ 27  38]
 [  2 118]]
[[39 26]
 [27 93]]
[[ 15  50]
 [  1 119]]
```

In [68]:
```
print(classification_report(test_y,test_y_hat_5))
print(classification_report(test_y,test_y_hat_7))
print(classification_report(test_y,test_y_hat_3))
```

```
              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

              precision    recall  f1-score   support

           0       0.59      0.60      0.60        65
           1       0.78      0.78      0.78       120

    accuracy                           0.71       185
   macro avg       0.69      0.69      0.69       185
weighted avg       0.71      0.71      0.71       185

              precision    recall  f1-score   support

           0       0.94      0.23      0.37        65
           1       0.70      0.99      0.82       120

    accuracy                           0.72       185
   macro avg       0.82      0.61      0.60       185
weighted avg       0.79      0.72      0.66       185
```

Problem Statement 3.3: Write a program to implement KNN- classification

In [70]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [71]:
```
credit_df=pd.read_csv('C:\\Users\\Pratiksha\\Data Analytics\\Datasets\\CreditRisk.csv')
credit_df.head()
```

Out[71]:

	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 [72]:
```
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 [73]:
```
credit_df.Loan_Status.value_counts()
```

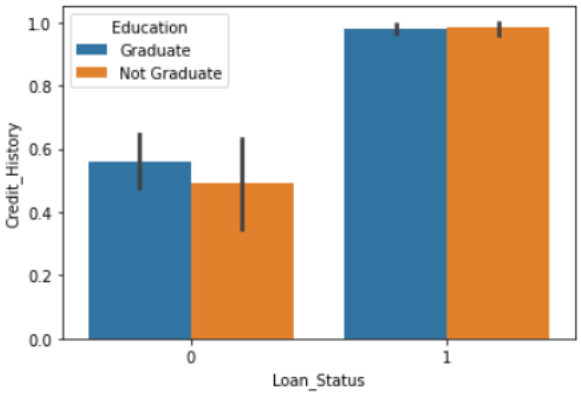
Out[73]:
```
1    422
0    192
Name: Loan_Status, dtype: int64
```

In [74]:
```
credit_df.groupby(['Education','Loan_Status']).Education.count()
```

Out[74]:
```
Education     Loan_Status
Graduate      0              140
              1              340
Not Graduate  0               52
              1               82
Name: Education, dtype: int64
```

In [75]:
```
sns.barplot(y='Credit_History',x='Loan_Status',hue='Education',data=credit_df)
```

Out[75]:
```
<AxesSubplot:xlabel='Loan_Status', ylabel='Credit_History'>
```



In [76]:
```
100*credit_df.isnull().sum()/credit_df.shape[0]
```

Out[76]:
```
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 [77]:
```
DF=credit_df.drop('Loan_ID',axis=1)
```

In [79]:
```
object_columns=DF.select_dtypes(include=['object']).columns
numeric_columns=DF.select_dtypes(exclude=['object']).columns
```

In [80]:
```
for column in object_columns:
    majority=DF[column].value_counts().iloc[0]
    DF[column].fillna(majority,inplace=True)
```

In [81]:
```
for column in numeric_columns:
    mean=DF[column].mean()
    DF[column].fillna(mean,inplace=True)
```

In [82]:
```
DF.head()
```

Out[82]:

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

In [83]:
```
DF[object_columns].Property_Area
```

Out[83]:
```
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 [84]:
```
DF[object_columns].Property_Area.head()
```

Out[84]:
```
0    Urban
1    Rural
2    Urban
3    Urban
4    Urban
Name: Property_Area, dtype: object
```

In [85]:
```
DF_dummy=pd.get_dummies(DF,columns=object_columns)
```

In [86]:
```
DF_dummy.shape
```

Out[86]: (614, 25)

In [87]:
```
DF_dummy.head()
```

Out[87]:

	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 [88]:
```
from sklearn.model_selection import train_test_split as TTS
from sklearn.neighbors import KNeighborsClassifier as KNN
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
```

In [89]:
```
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 [90]:
```
train_x.shape,test_x.shape
```

Out[90]: ((429, 24), (185, 24))

KNN Model

In [91]:
```
knn_model=KNN(n_neighbors=7)
```

In [92]:
```
knn_model.fit(train_x,train_y)
```

Out[92]: KNeighborsClassifier(n_neighbors=7)

In [94]:
```
train_y_hat=knn_model.predict(train_x)
test_y_hat=knn_model.predict(test_x)
```

In [95]:
```
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       0.70      0.24      0.35       127
           1       0.75      0.96      0.84       302

    accuracy                           0.74       429
   macro avg       0.72      0.60      0.60       429
weighted avg       0.73      0.74      0.70       429

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

           0       0.36      0.12      0.18        65
           1       0.65      0.88      0.75       120

    accuracy                           0.62       185
   macro avg       0.51      0.50      0.47       185
weighted avg       0.55      0.62      0.55       185
```

Practical 4
Unsupervised Learning

Problem Statement 4.1 Write a program to implement K-Means clustering algorithm.

In [1]:
```
import pandas as pd
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
```

In [4]:
```
df=pd.read_csv('C:\\Users\\Pratiksha\\Data Analytics\\Datasets\\driver-data.csv')
df
```

Out[4]:

	Driver_ID	Distance_Feature	Speeding_Feature
0	3423311935	71.24	28
1	3423313212	52.53	25
2	3423313724	64.54	27
3	3423311373	55.69	22
4	3423310999	54.58	25
...	...	...	...
3995	3423310685	160.04	10
3996	3423312600	176.17	5
3997	3423312921	170.91	12
3998	3423313630	176.14	5
3999	3423311533	168.03	9

4000 rows × 3 columns

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

Out[5]:

	Driver_ID	Distance_Feature	Speeding_Feature
0	3423311935	71.24	28
1	3423313212	52.53	25
2	3423313724	64.54	27
3	3423311373	55.69	22
4	3423310999	54.58	25

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

Out[6]:

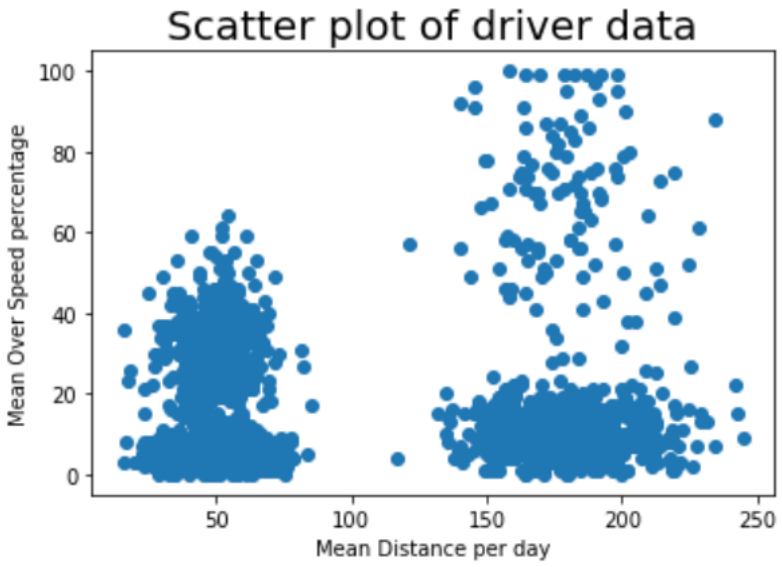
	Driver_ID	Distance_Feature	Speeding_Feature
count	4.000000e+03	4000.000000	4000.000000
mean	3.423312e+09	76.041523	10.721000
std	1.154845e+03	53.469563	13.708543
min	3.423310e+09	15.520000	0.000000
25%	3.423311e+09	45.247500	4.000000
50%	3.423312e+09	53.330000	6.000000
75%	3.423313e+09	65.632500	9.000000
max	3.423314e+09	244.790000	100.000000

In [7]:
```
df.shape
```

Out[7]:
```
(4000, 3)
```

In [10]:
```
plt.plot(df.Distance_Feature,df.Speeding_Feature,'o')
plt.xlabel('Mean Distance per day')
plt.ylabel('Mean Over Speed percentage')
plt.title('Scatter plot of driver data',fontsize=20)
plt.show
```

Out[10]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



In [12]:
```
data=df.drop(['Driver_ID'],axis=1)
cluster_model=KMeans(n_clusters=2)
cluster_model.fit(data)
```

Out[12]:
```
KMeans(n_clusters=2)
```

In [13]:
```
df['labels']=cluster_model.labels_
```

In [14]:
```
df.head()
```

Out[14]:

	Driver_ID	Distance_Feature	Speeding_Feature	labels
0	3423311935	71.24	28	0
1	3423313212	52.53	25	0
2	3423313724	64.54	27	0
3	3423311373	55.69	22	0
4	3423310999	54.58	25	0

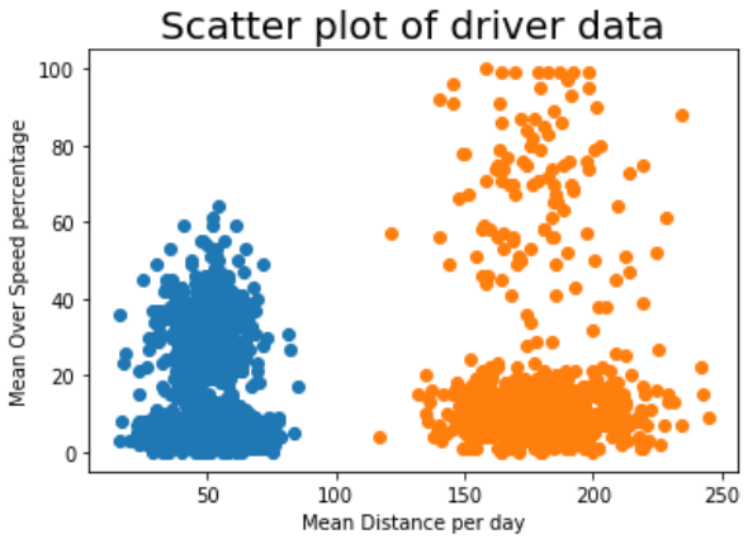
In [15]:
```
df.labels.unique()
```

Out[15]:
```
array([0, 1])
```

In [16]:
```
df['labels'].value_counts()
```

Out[16]:
```
0    3200
1     800
Name: labels, dtype: int64
```

In [20]:
```
for label in df.labels.unique():
    plt.plot(df.loc[df.labels == label,'Distance_Feature'],
    df.loc[df.labels == label,'Speeding_Feature'],'o',label = label)
    plt.xlabel('Mean Distance per day')
    plt.ylabel('Mean Over Speed percentage')
    plt.title('Scatter plot of driver data',fontsize=20)
    plt.show
```

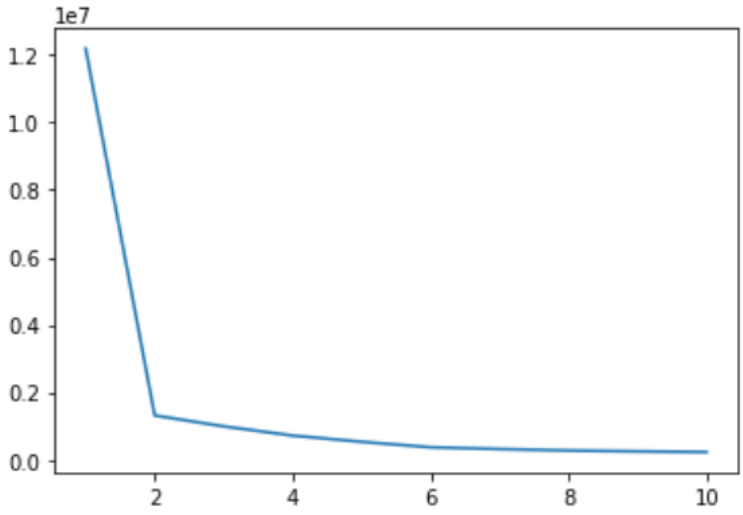


In [21]:
```
cluster_model.cluster_centers_
```

Out[21]:
```
array([[ 50.04763438,   8.82875   ],
       [180.017075  ,  18.29      ]])
```

In [22]:
```
error = []
for k in range(1,11):
    cluster_model = KMeans(k)
    cluster_model.fit(data)
    error.append(cluster_model.inertia_)
```

In [23]:
```
plt.plot(range(1,11),error)
plt.show()
```



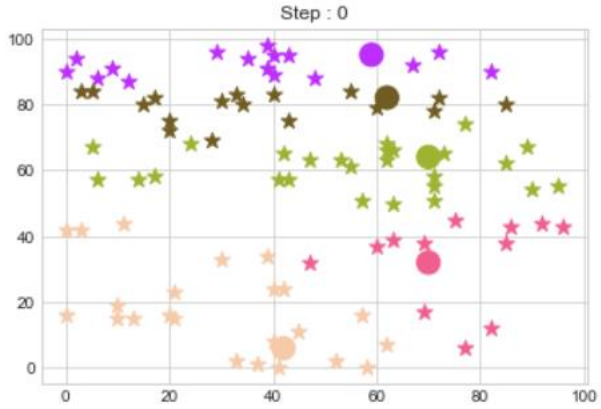
Problem statement 4.2: Write a program to implement K-medoid clustering algorithm

Code:

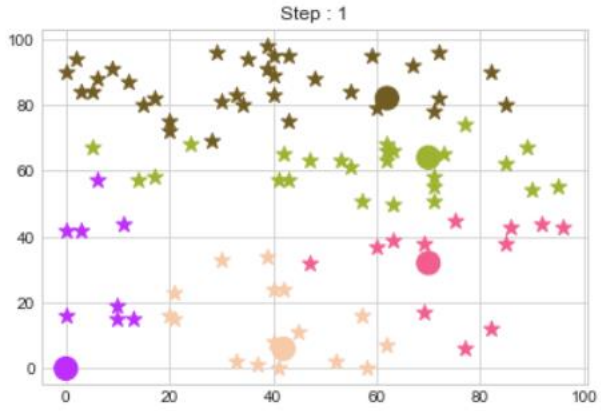
```
import numpy as np
import matplotlib.pyplot as plt

plt.style.use("seaborn-whitegrid")

class KMedoidsClass:
    def __init__(self,data,k,iters):
        self.data= data
        self.k = k
        self.iters = iters
        self.medoids = np.array([data[i] for i in range(self.k)])
        self.colors = np.array(np.random.randint(0, 255, size =(self.k, 4)))/255
        self.colors[:,3]=1

    def manhattan(self,p1, p2):
        return np.abs((p1[0]-p2[0])) + np.abs((p1[1]-p2[1]))

    def get_costs(self, medoids, data):
        tmp_clusters = {i:[] for i in range(len(medoids))}
        cst = 0
        for d in data:
            dst = np.array([self.manhattan(d, md) for md in medoids])
            c = dst.argmin()
            tmp_clusters[c].append(d)
            cst+=dst.min()

        tmp_clusters = {k:np.array(v) for k,v in tmp_clusters.items()}
        return tmp_clusters, cst

    def fit(self):

        samples,_ = self.data.shape

        self.clusters, cost = self.get_costs(data=self.data, medoids=self.medoids)
        count = 0


        colors =  np.array(np.random.randint(0, 255, size =(self.k, 4)))/255
        colors[:,3]=1

        plt.title(f"Step : 0")
        [plt.scatter(self.clusters[t][:, 0], self.clusters[t][:, 1], marker="*", s=100,
                                color = colors[t]) for t in range(self.k)]
        plt.scatter(self.medoids[:, 0], self.medoids[:, 1], s=200, color=colors)
        plt.show()
```

```
        while True:
            swap = False
            for i in range(samples):
                if not i in self.medoids:
                    for j in range(self.k):
                        tmp_meds = self.medoids.copy()
                        tmp_meds[j] = i
                        clusters_, cost_ = self.get_costs(data=self.data, medoids=tmp_meds)

                        if cost_<cost:
                            self.medoids = tmp_meds
                            cost = cost_
                            swap = True
                            self.clusters = clusters_
                            print(f"Medoids Changed to: {self.medoids}.")
                            plt.title(f"Step : {count+1}")
                            [plt.scatter(self.clusters[t][:, 0], self.clusters[t][:, 1], marker="*", s=100,
                                    color = colors[t]) for t in range(self.k)]
                            plt.scatter(self.medoids[:, 0], self.medoids[:, 1], s=200, color=colors)
                            plt.show()
            count+=1

            if count>=self.iters:
                print("End of the iterations.")
                break
            if not swap:
                print("No changes.")
                break

dt = np.random.randint(0,100, (100,2))
kmedoid = KMedoidsClass(dt,5,5)
kmedoid.fit()
```

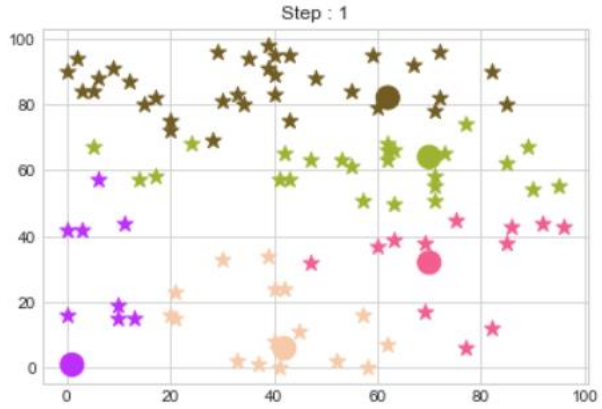
Output:



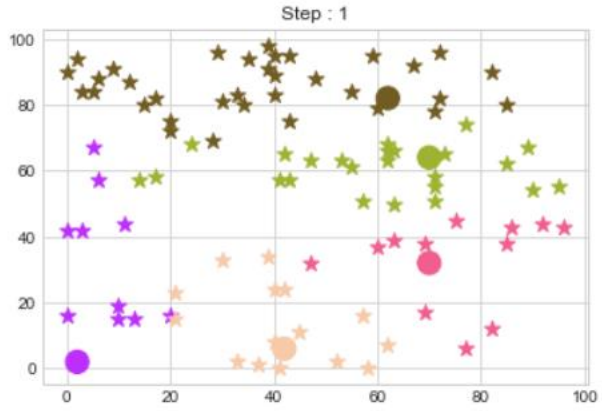

```
Medoids Changed to: [[ 0  0]
 [62 82]
 [70 64]
 [70 32]
 [42  6]].
```



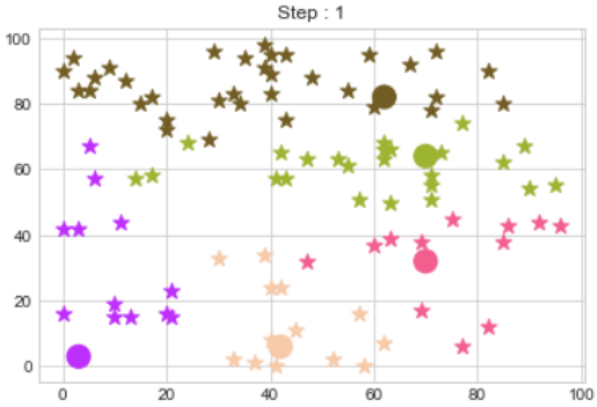

```
Medoids Changed to: [[ 1  1]
 [62 82]
 [70 64]
 [70 32]
 [42  6]].
```



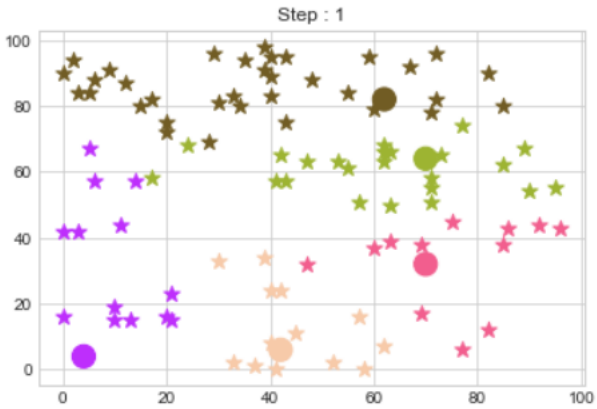

```
Medoids Changed to: [[ 2  2]
 [62 82]
 [70 64]
 [70 32]
 [42  6]].
```



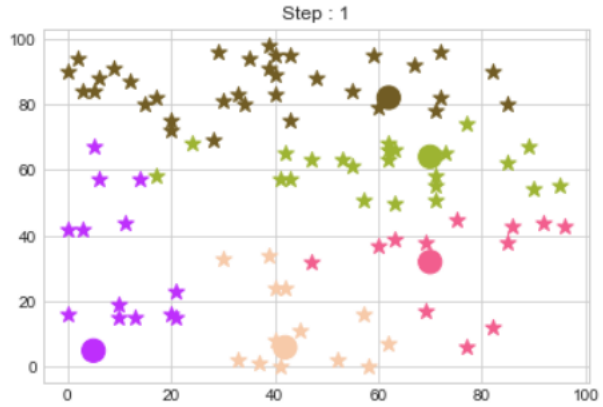

```
Medoids Changed to: [[ 3  3]
 [62 82]
 [70 64]
 [70 32]
 [42  6]].
```



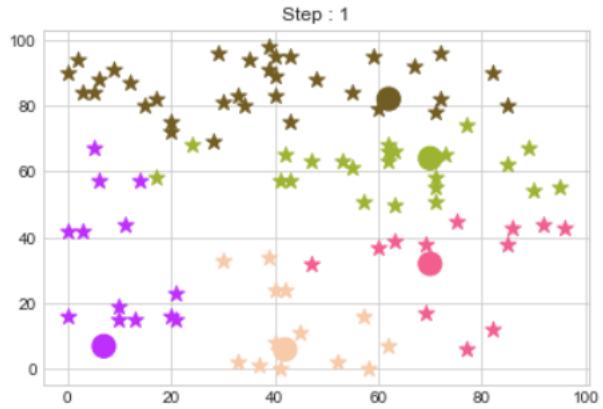

```
Medoids Changed to: [[ 4  4]
 [62 82]
 [70 64]
 [70 32]
 [42  6]].
```



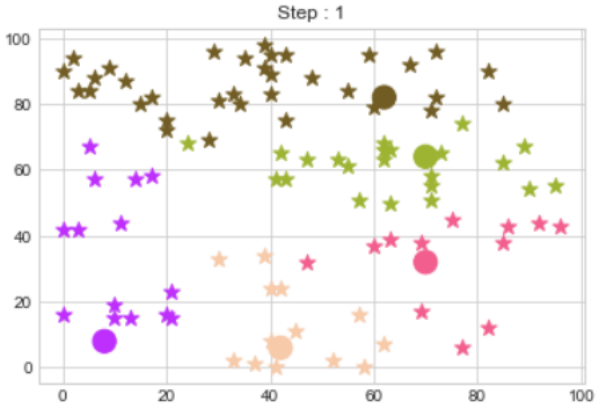

```
Medoids Changed to: [[ 5  5]
 [62 82]
 [70 64]
 [70 32]
 [42  6]].
```



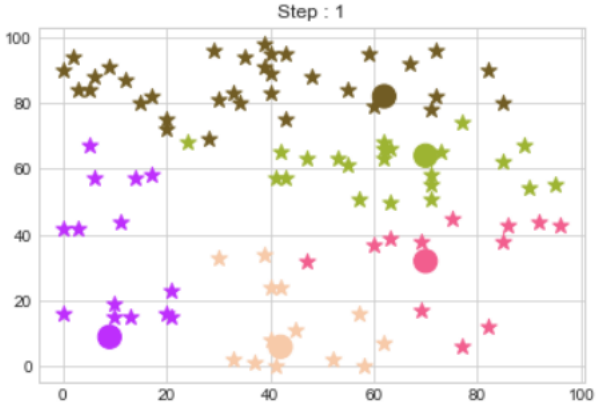

```
Medoids Changed to: [[ 7  7]
 [62 82]
 [70 64]
 [70 32]
 [42  6]].
```




```
Medoids Changed to: [[ 8  8]
 [62 82]
 [70 64]
 [70 32]
 [42  6]].
```



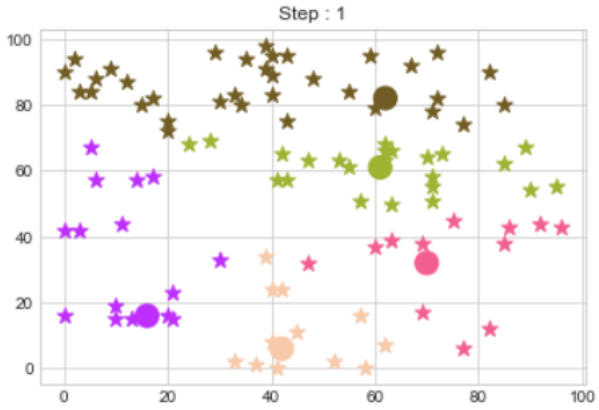

```
Medoids Changed to: [[ 9  9]
 [62 82]
 [70 64]
 [70 32]
 [42  6]].
```



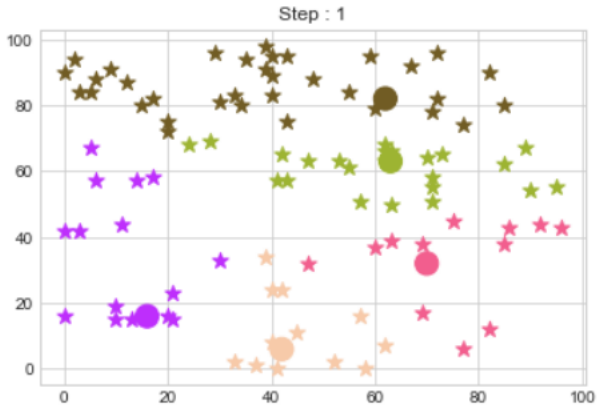

```
Medoids Changed to: [[10 10]
 [62 82]
 [70 64]
 [70 32]
 [42  6]].
```

-
-
-
-
-
-

```
Medoids Changed to: [[16 16]
 [62 82]
 [61 61]
 [70 32]
 [42  6]].
```




```
Medoids Changed to: [[16 16]
 [62 82]
 [63 63]
 [70 32]
 [42  6]].
```




```
No changes.
```

•

Practical 5

Classifying data using Support Vector Machines (SVMs)

Problem Statement 5.1 Write a program to Classifying data using Support Vector Machines (SVMs).

In [8]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [10]:
```
credit_df=pd.read_csv('C:\\Users\\Pratiksha\\Data Analytics\\Datasets\\CreditRisk.csv')
credit_df.head()
```

Out[10]:

	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 [11]:
```
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 [12]:
```
credit_df.describe()
```

Out[12]:

	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 [13]:
```
credit_df.Loan_Status.value_counts()
```

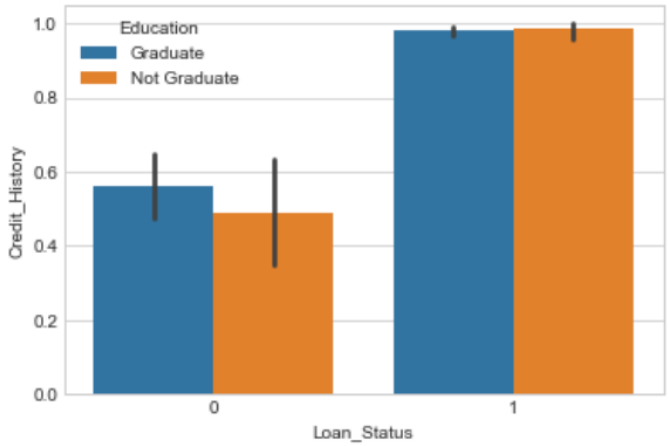
Out[13]:
```
1    422
0    192
Name: Loan_Status, dtype: int64
```

In [14]:
```
credit_df.groupby(['Education','Loan_Status']).Education.count()
```

Out[14]:
```
Education     Loan_Status
Graduate      0              140
              1              340
Not Graduate  0               52
              1               82
Name: Education, dtype: int64
```

In [15]:
```
sns.barplot(y='Credit_History',x='Loan_Status',hue='Education',data=credit_df)
```

Out[15]:
```
<AxesSubplot:xlabel='Loan_Status', ylabel='Credit_History'>
```



In [16]:
```
100*credit_df.isnull().sum()/credit_df.shape[0]
```

Out[16]:
```
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 [17]:
```
DF=credit_df.drop('Loan_ID',axis=1)
```

In [18]:
```
object_columns=DF.select_dtypes(include=['object']).columns
numeric_columns=DF.select_dtypes(exclude=['object']).columns
```

In [19]:
```
for column in object_columns:
    majority=DF[column].value_counts().iloc[0]
    DF[column].fillna(majority,inplace=True)
```

In [20]:
```
for column in numeric_columns:
    mean=DF[column].mean()
    DF[column].fillna(mean,inplace=True)
```

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

Out[21]:

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

In [22]:
```
DF[object_columns].Property_Area
```

Out[22]:
```
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 [23]:
```
DF[object_columns].Property_Area.head()
```

Out[23]:
```
0    Urban
1    Rural
2    Urban
3    Urban
4    Urban
Name: Property_Area, dtype: object
```

In [24]:
```
DF_dummy = pd.get_dummies(DF,columns = object_columns)
```

In [25]:
```
DF_dummy.shape
```

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

In [26]:
```
DF_dummy.head()
```

Out[26]:

	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 [27]:
```
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 [28]:
```
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 [29]:
```
train_x.shape,test_x.shape
```

Out[29]:
```
((429, 24), (185, 24))
```

SVM Model

In [30]:
```
svm_model=SVC(kernel='rbf',gamma=0.00001,C=1000)
```

In [31]:
```
svm_model.fit(train_x,train_y)
```

Out[31]:
```
SVC(C=1000, gamma=1e-05)
```

In [32]:
```
train_y_hat=svm_model.predict(train_x)
test_y_hat=svm_model.predict(test_x)
```

In [33]:
```
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
```

In [34]:
```
confusion_matrix(test_y,test_y_hat)
```

Out[34]:
```
array([[12, 53],
       [21, 99]], dtype=int64)
```

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

Out[35]:
```
array([[121,   6],
       [  7, 295]], dtype=int64)
```

Practical 6
Bagging Algorithm

Problem Statement 6.1 Write a program to implement Bagging Algorithm Decision Tree.

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

In [2]:
```
credit_df=pd.read_csv('C:\\Users\\Pratiksha\\Data Analytics\\Datasets\\CreditRisk.csv')
credit_df.head()
```

Out[2]:

	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 [3]:
```
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 [4]:
```
credit_df.describe()
```

Out[4]:

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

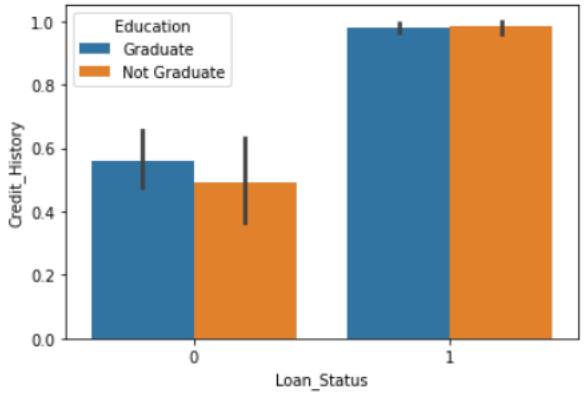
Out[5]:
```
1    422
0    192
Name: Loan_Status, dtype: int64
```

In [6]:
```
credit_df.groupby(['Education','Loan_Status']).Education.count()
```

Out[6]:
```
Education     Loan_Status
Graduate      0              140
              1              340
Not Graduate  0               52
              1               82
Name: Education, dtype: int64
```

In [7]:
```
sns.barplot(y='Credit_History',x='Loan_Status',hue='Education',data=credit_df)
```

Out[7]:
```
<AxesSubplot:xlabel='Loan_Status', ylabel='Credit_History'>
```



In [8]:
```
100*credit_df.isnull().sum()/credit_df.shape[0]
```

Out[8]:
```
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 [9]:
```
DF=credit_df.drop('Loan_ID',axis=1)
```

In [10]:
```
object_columns=DF.select_dtypes(include=['object']).columns
numeric_columns=DF.select_dtypes(exclude=['object']).columns
```

In [11]:
```
for column in object_columns:
    majority=DF[column].value_counts().iloc[0]
    DF[column].fillna(majority,inplace=True)
```

In [12]:
```
for column in numeric_columns:
    mean=DF[column].mean()
    DF[column].fillna(mean,inplace=True)
```

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

Out[13]:

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

In [14]:
```
credit_df.head()
```

Out[14]:

	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 [15]:
```
DF[object_columns].Property_Area
```

Out[15]:
```
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 [16]:
```
DF[object_columns].Property_Area.head()
```

Out[16]:
```
0    Urban
1    Rural
2    Urban
3    Urban
4    Urban
Name: Property_Area, dtype: object
```

In [17]:
```
DF_dummy=pd.get_dummies(DF,columns=object_columns)
```

In [18]:
```
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	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

Model Construction

In [20]:
```
from sklearn.model_selection import train_test_split as TTS
from sklearn.tree import DecisionTreeClassifier as DTC
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
```

In [21]:
```
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 [22]:
```
train_x.shape,test_x.shape
```

Out[22]: ((429, 24), (185, 24))

In [23]:
```
dt_model = DTC(max_depth = 14)
```

In [24]:
```
dt_model.fit(train_x,train_y)
```

Out[24]: DecisionTreeClassifier(max_depth=14)

In [25]:
```
train_y_hat = dt_model.predict(train_x)
test_y_hat = dt_model.predict(test_x)
```

In [26]:
```
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       1.00      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      0.51      0.53        65
           1       0.75      0.78      0.76       120

    accuracy                           0.69       185
   macro avg       0.65      0.65      0.65       185
weighted avg       0.68      0.69      0.68       185
```

In [27]:
```
confusion_matrix(train_y,train_y_hat)
```

Out[27]:
```
array([[127,   0],
       [  0, 302]], dtype=int64)
```

In [28]:
```
confusion_matrix(test_y,test_y_hat)
```

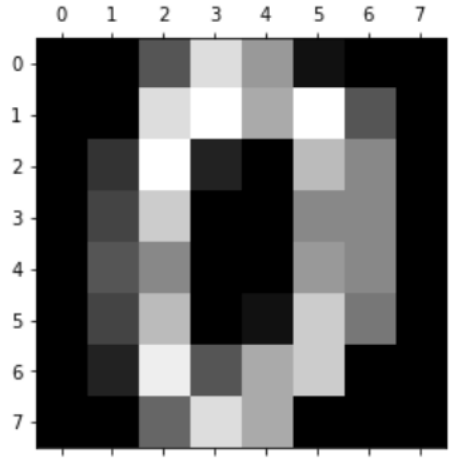
Out[28]:
```
array([[33, 32],
       [26, 94]], dtype=int64)
```

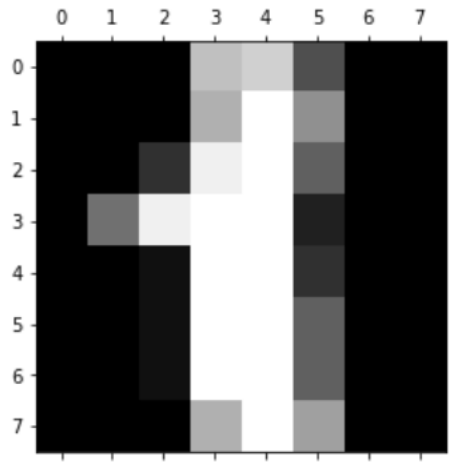
Problem Statement 6.2 Write a program to implement Bagging Algorithm Random Forest.

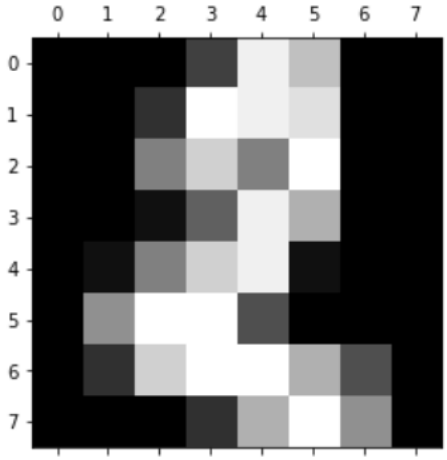
In [30]:
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_digits
digits=load_digits()
```

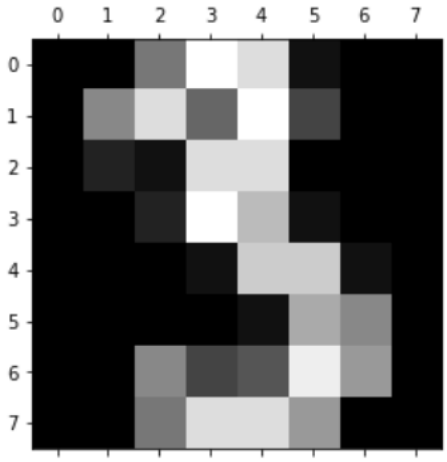
In [33]:
```
plt.gray()
for i in range(4):
    plt.matshow(digits.images[i])
```

```
<Figure size 432x288 with 0 Axes>
```






0 1 2 3 4 5 6 7
0
1
2
3
4
5
6
7


0 1 2 3 4 5 6 7
0
1
2
3
4
5
6
7

In [34]:
```
df=pd.DataFrame(digits.data)
df.head()
```

Out[34]:

	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

In [35]:
```
df['target'] = digits.target
```

In [36]:
```
df[0:12]
```

Out[36]:

	0	1	2	3	4	5	6	7	8	9	...	55	56	57	58	59	60	61	62	63	target
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	6.0	13.0	10.0	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	11.0	16.0	10.0	0.0	0.0	1
2	0.0	0.0	0.0	4.0	15.0	12.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	3.0	11.0	16.0	9.0	0.0	2
3	0.0	0.0	7.0	15.0	13.0	1.0	0.0	0.0	0.0	8.0	...	0.0	0.0	0.0	7.0	13.0	13.0	9.0	0.0	0.0	3
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	2.0	16.0	4.0	0.0	0.0	4
5	0.0	0.0	12.0	10.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	9.0	16.0	16.0	10.0	0.0	0.0	5
6	0.0	0.0	0.0	12.0	13.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	1.0	9.0	15.0	11.0	3.0	0.0	6
7	0.0	0.0	7.0	8.0	13.0	16.0	15.0	1.0	0.0	0.0	...	0.0	0.0	0.0	13.0	5.0	0.0	0.0	0.0	0.0	7
8	0.0	0.0	9.0	14.0	8.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	11.0	16.0	15.0	11.0	1.0	0.0	8
9	0.0	0.0	11.0	12.0	0.0	0.0	0.0	0.0	0.0	2.0	...	0.0	0.0	0.0	9.0	12.0	13.0	3.0	0.0	0.0	9
10	0.0	0.0	1.0	9.0	15.0	11.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	1.0	10.0	13.0	3.0	0.0	0.0	0
11	0.0	0.0	0.0	0.0	14.0	13.0	1.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	1.0	13.0	16.0	1.0	0.0	1

12 rows × 65 columns

In [37]:
```
X = df.drop('target',axis='columns')
y = df.target
```

In [38]:
```
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)
```

In [39]:
```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(n_estimators=40)
model.fit(X_train,y_train)
```

Out[39]:
```
RandomForestClassifier(n_estimators=40)
```

In [41]:
```
model.score(X_test,y_test)
```

Out[41]:
```
0.9888888888888889
```

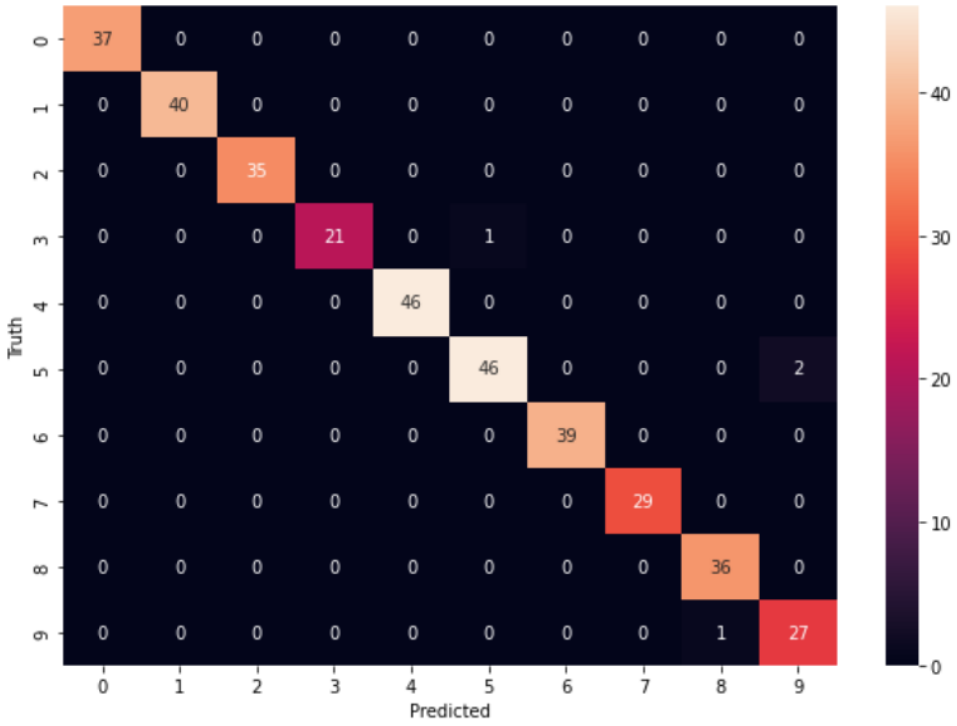
In [42]:
```
y_predicted = model.predict(X_test)
```

In [43]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,y_predicted)
cm
```

Out[43]:
```
array([[37,  0,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 0, 40,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0, 35,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0, 21,  0,  1,  0,  0,  0,  0],
       [ 0,  0,  0,  0, 46,  0,  0,  0,  0,  0],
       [ 0,  0,  0,  0,  0, 46,  0,  0,  0,  2],
       [ 0,  0,  0,  0,  0,  0, 39,  0,  0,  0],
       [ 0,  0,  0,  0,  0,  0,  0, 29,  0,  0],
       [ 0,  0,  0,  0,  0,  0,  0,  0, 36,  0],
       [ 0,  0,  0,  0,  0,  0,  0,  0,  1, 27]], dtype=int64)
```

In [44]:
```
import seaborn as sn
plt.figure(figsize=(10,7))
sn.heatmap(cm,annot=True)
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

Out[44]: Text(69.0, 0.5, 'Truth')



Practical 7
Boosting Algorithms

Problem Statement 7.1 Write a program to implement Boosting Algorithms: AdaBoost .

In [2]:
```
import pandas as pd
from sklearn.ensemble import AdaBoostClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
```

In [3]:
```
data = pd.read_csv("C:\\Users\\Pratiksha\\Data Analytics\\Datasets\\income.csv")
data.head()
```

Out[3]:

	age	fnlwgt	education_num	capital_gain	capital_loss	hours_per_week	income_level
0	39	77516	13	2174	0	40	0
1	50	83311	13	0	0	13	0
2	38	215646	9	0	0	40	0
3	53	234721	7	0	0	40	0
4	28	338409	13	0	0	40	0

In [6]:
```
X = data.iloc[:,0:6]
y = data.iloc[:,6]
```

In [9]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

In [10]:
```
AdaModel = AdaBoostClassifier(n_estimators=100, learning_rate=1)
model = AdaModel.fit(X_train, y_train)

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

In [11]:
```
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.8419490224178524
```

Problem Statement 7.2 Write a program to implement Boosting Algorithms: Stochastic Gradient Boosting.

Stochastic Gradient Boosting:-

In [12]:
```
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.metrics import r2_score
from sklearn.datasets import load_boston
from sklearn.metrics import mean_absolute_error
import warnings
warnings.filterwarnings('ignore')
```

In [16]:
```
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 [17]:
```
X.head()
```

Out[17]:

	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 [18]:
```
y[1:10]   # Response
```

Out[18]:
```
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 [19]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

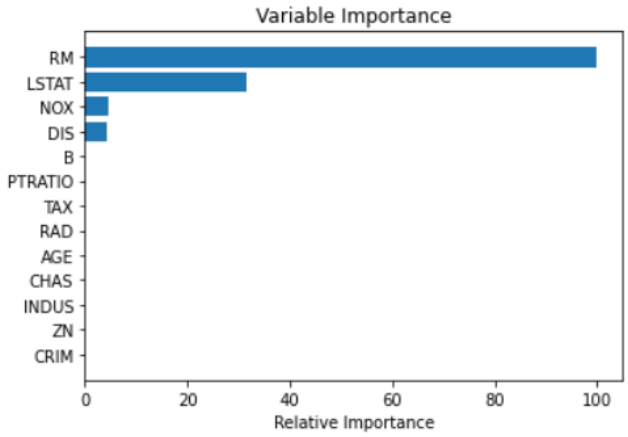
In [20]:
```
# Create gradientboose REGRESSOR object
gradientregressor = GradientBoostingRegressor(max_depth=2, n_estimators=3, learning_rate=1)
```

In [23]:
```
model = gradientregressor.fit(X_train,y_train)

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

In [24]:
```
r2_score(y_pred, y_test)
```

Out[24]:
```
0.5268677744769721
```

In [26]:
```
import matplotlib.pyplot as plt
%matplotlib inline

# Plot deature 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.xlabel('Relative Importance')
plt.yticks(pos, boston.feature_names[sorted_idx])
plt.title('Variable Importance')
plt.show()
```



Practical 8
Deployment of Machine Learning Models

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import pickle
```

In [2]:
```
# 1. Given the dataset, hiring.csv, construct a linear regression model. #Deploy the model using flask.
df  =  pd.read_csv('C:\\Users\\Pratiksha\\Data Analytics\\Datasets\\hiring.csv')
```

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

Out[6]:

	experience	test_score(out of 10)	interview_score(out of 10)	salary($)
0	0	8.0	9	50000
1	0	8.0	6	45000
2	five	6.0	7	60000
3	two	10.0	10	65000
4	seven	9.0	6	70000

In [9]:
```
df['experience'].fillna(0,  inplace=True)
df['test_score(out of 10)'].fillna(df['test_score(out of 10)'].mean(),inplace=True)
df
```

Out[9]:

	experience	test_score(out of 10)	interview_score(out of 10)	salary($)
0	0	8.000000	9	50000
1	0	8.000000	6	45000
2	five	6.000000	7	60000
3	two	10.000000	10	65000
4	seven	9.000000	6	70000
5	three	7.000000	10	62000
6	ten	7.857143	7	72000
7	eleven	7.000000	8	80000

In [12]:
```
#Converting words to integer values
def convert_to_int(word):
    word_dict  =  {'one':1,  'two':2,  'three':3,  'four':4,  'five':5,  'six':6,  'seven':7,  'eight':8,  'nine':9,  'ten':10,
    return word_dict[word]
df['experience']  =  df['experience'].apply(lambda  x  :  convert_to_int(x))
```

In [13]:
```
df
```

Out[13]:

	experience	test_score(out of 10)	interview_score(out of 10)	salary($)
0	0	8.000000	9	50000
1	0	8.000000	6	45000
2	5	6.000000	7	60000
3	2	10.000000	10	65000
4	7	9.000000	6	70000
5	3	7.000000	10	62000
6	10	7.857143	7	72000
7	11	7.000000	8	80000

In [14]:
```
X  =  df.iloc[:,  :3]
y  =  df.iloc[:,-1]

#Splitting Training and Test Set
#Since we have a very small dataset, we will train our model with all availabe data.
```

In [15]:
```
X
```

Out[15]:

	experience	test_score(out of 10)	interview_score(out of 10)
0	0	8.000000	9
1	0	8.000000	6
2	5	6.000000	7
3	2	10.000000	10
4	7	9.000000	6
5	3	7.000000	10
6	10	7.857143	7
7	11	7.000000	8

In [16]:
```
y
```

Out[16]:
```
0    50000
1    45000
2    60000
3    65000
4    70000
5    62000
6    72000
7    80000
Name: salary($), dtype: int64
```

In [17]:
```
from sklearn.linear_model import LinearRegression
lmodel = LinearRegression()

#Fitting model with trainig data
lmodel.fit(X,  y)
```

Out[17]:
```
LinearRegression()
```

In [17]:
```
from sklearn.linear_model import LinearRegression
lmodel = LinearRegression()

#Fitting model with trainig data
lmodel.fit(X,  y)
```

Out[17]:
```
LinearRegression()
```

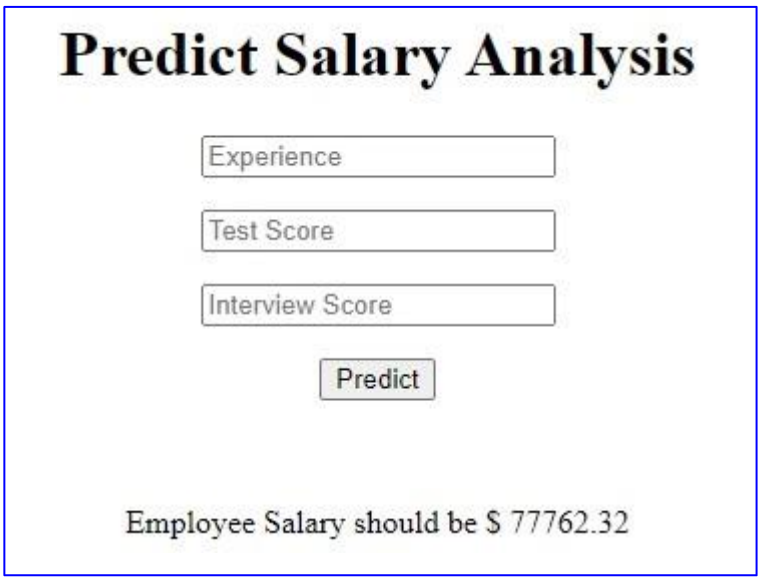
In [18]:
```
# Saving model to disk
pickle.dump(lmodel,  open('model.pkl','wb'))
```

In [19]:
```
# Loading model to compare the results
model  =  pickle.load(open('model.pkl','rb'))
print(model.predict([[2,  9,  6]]))
```

```
[53290.89255945]
```

 # 2. Run the model on browser for and predict the salary

a. experience =9, test_score=8 and interview_score=9



b. experience =1, test_score=4 and interview_score=3