**Practical No. 1**

**Aim:** Implementation of Logic programming using LISP /PROLOG-DFS for water jug problem / BFS for tic-tac-toe problem/ Hill-climbing to solve 8- Puzzle 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));

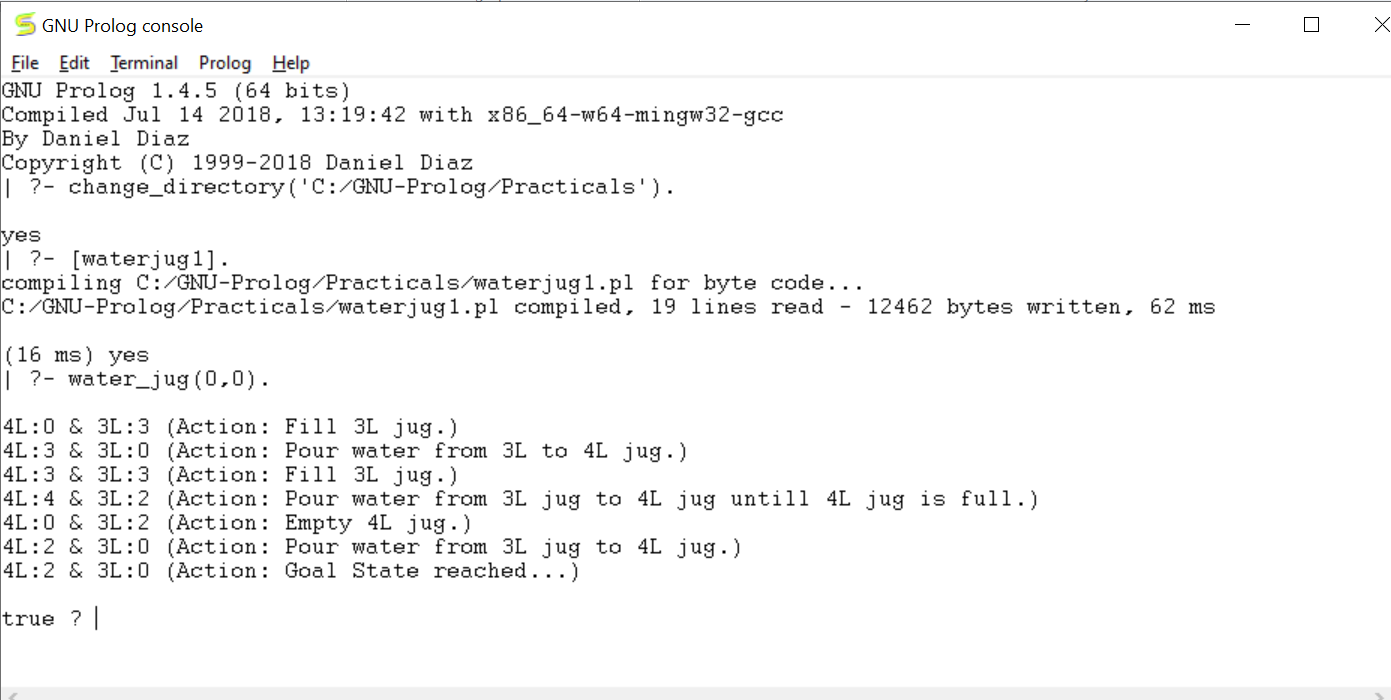
(X=:=4, Y=:=1,nl,write('4L:2 & 3L:3 (Action: Pour water from 4L to 3L jug untill 3L jug is full.)'),XX is X-2,YY is Y+2,water\_jug(XX,YY));

(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:**

****

**Practical No. 2**

**Aim:** Introduction to Python Programming: Learn the different libraries - NumPy, Pandas, SciPy, Matplotlib, Scikit Learn.

**Code:**

**Basics**

In [1]:

id = [1,2,3,4]

emp\_name = ["ram","preeti","satish","john"]

num\_emp = 4

In [2]:

emp\_list = [id, emp\_name, num\_emp]

print(emp\_list)

[[1, 2, 3, 4], ['ram', 'preeti', 'satish', 'john'], 4]

In [3]:

print(emp\_list[0])

[1, 2, 3, 4]

In [4]:

print(emp\_list[1][1])

preeti

In [5]:

emp\_list[2] = 5

print(emp\_list)

[[1, 2, 3, 4], ['ram', 'preeti', 'satish', 'john'], 5]

In [6]:

emp\_list[1][3] = "vinit"

print(emp\_list)

[[1, 2, 3, 4], ['ram', 'preeti', 'satish', 'vinit'], 5]

In [7]:

emp\_list[1].append('omkar')

print(emp\_list)

[[1, 2, 3, 4], ['ram', 'preeti', 'satish', 'vinit', 'omkar'], 5]

In [8]:

emp\_list.append([23,25,36,43,53])

print(emp\_list)

[[1, 2, 3, 4], ['ram', 'preeti', 'satish', 'vinit', 'omkar'], 5, [23, 25, 36, 43, 53]]

In [9]:

emp\_list[0].insert(0,5)

print(emp\_list)

[[5, 1, 2, 3, 4], ['ram', 'preeti', 'satish', 'vinit', 'omkar'], 5, [23, 25, 36, 43, 53]]

In [10]:

del emp\_list[3]

print(emp\_list)

[[5, 1, 2, 3, 4], ['ram', 'preeti', 'satish', 'vinit', 'omkar'], 5]

In [11]:

emp\_list[1].remove("ram")

print(emp\_list)

[[5, 1, 2, 3, 4], ['preeti', 'satish', 'vinit', 'omkar'], 5]

In [12]:

salary = ['high', 'low', 'medium', 'low']

In [13]:

salary.remove('low')

print(salary)

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

In [14]:

emp\_list[0].pop(4)

Out[14]:

4

**Numpy**

In [1]:

import numpy as np

In [2]:

x = np.array([2, 3, 4, 5])

print(type(x))

<class 'numpy.ndarray'>

In [3]:

print(x)

[2 3 4 5]

In [4]:

x = np.array([2, 3, 'n', 5])

print(x)

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

In [5]:

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 [6]:

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 [7]:

d = np.arange(start =1, stop =10, step =2)

print(d)

[1 3 5 7 9]

In [8]:

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

print(i)

[[1. 1. 1. 1.]

[1. 1. 1. 1.]

[1. 1. 1. 1.]]

In [9]:

j = np.zeros((3, 4))

print(j)

[[0. 0. 0. 0.]

[0. 0. 0. 0.]

[0. 0. 0. 0.]]

In [10]:

l = np.random.rand(5)

print(l)

[0.11678631 0.05853044 0.37605417 0.53108771 0.41714407]

In [11]:

p = np.random.rand(5,2)

print(p)

[[0.57274246 0.80793859]

[0.19270974 0.71137196]

[0.78653993 0.59157179]

[0.31900428 0.8219432 ]

[0.15661479 0.72497422]]

In [12]:

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 [13]:

import sys

x = range(1000)

sys.getsizeof(1)\*len(x)

Out[13]:

28000

In [14]:

y = np.array(x)

y.itemsize\*y.size

Out[14]:

4000

In [15]:

grid = np.arange(start =1, stop =10).reshape(3,3)

print(grid)

[[1 2 3]

[4 5 6]

[7 8 9]]

In [16]:

a = np.array([[1,2,3],[4,5,6],[7,8,9]])

a.shape

Out[16]:

(3, 3)

In [17]:

np.sum(a)

np.sum(a, axis=0)

Out[17]:

array([12, 15, 18])

In [18]:

a.sum()

Out[18]:

45

In [19]:

np.sum(a,axis=1)

Out[19]:

array([ 6, 15, 24])

In [20]:

b = np.arange(start =11, stop =20).reshape(3,3)

print(b)

[[11 12 13]

[14 15 16]

[17 18 19]]

In [21]:

print(a)

[[1 2 3]

[4 5 6]

[7 8 9]]

In [22]:

np.add(a,b)

Out[22]:

array([[12, 14, 16],

[18, 20, 22],

[24, 26, 28]])

In [23]:

np.multiply(a,b)

Out[23]:

array([[ 11, 24, 39],

[ 56, 75, 96],

[119, 144, 171]])

In [24]:

a[1:3]

Out[24]:

array([[4, 5, 6],

[7, 8, 9]])

In [25]:

np.transpose(a)

Out[25]:

array([[1, 4, 7],

[2, 5, 8],

[3, 6, 9]])

In [26]:

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 [27]:

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

In [28]:

a\_col = np.append(a,col,axis=1)

print(a\_col)

[[ 1 2 3 21]

[ 4 5 6 22]

[ 7 8 9 23]]

In [29]:

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 [30]:

a\_del = np.delete(a\_ins,2,axis=0)

print(a\_del)

[[ 1 2 3]

[13 15 16]

[ 7 8 9]]

In [33]:

print(x)

timeit=np.sum(y)

range(0, 1000)

**Pandas**

In [1]:

import os

In [2]:

import pandas as pd

In [3]:

pd.read\_csv('C:/Users/Kartik/Downloads/mtcars.csv')

Out[3]:

|  | 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 [4]:

data1=pd.read\_csv('C:/Users/Kartik/Downloads/mtcars.csv')

In [5]:

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 [6]:

data1.head()

Out[6]:

|  | 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 [7]:

data1.size

Out[7]:

384

In [8]:

data1.shape

Out[8]:

(32, 12)

In [9]:

data1.ndim

Out[9]:

2

In [10]:

data1.at[4,'model']

Out[10]:

'Hornet Sportabout'

In [11]:

data1.iat[4,3]

Out[11]:

360.0

In [12]:

data1.loc[:,'model']

Out[12]:

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 [13]:

data1.loc[0:5,'model']

Out[13]:

0 Mazda RX4

1 Mazda RX4 Wag

2 Datsun 710

3 Hornet 4 Drive

4 Hornet Sportabout

5 Valiant

Name: model, dtype: object

In [14]:

data1.iloc[0:5,0:2]

Out[14]:

|  | 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 [15]:

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

Out[15]:

|  | 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 [16]:

data1.dtypes

Out[16]:

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 [17]:

data1['model'].dtype

Out[17]:

dtype('O')

In [18]:

data1.axes

Out[18]:

[RangeIndex(start=0, stop=32, step=1),

Index(['model', 'mpg', 'cyl', 'disp', 'hp', 'drat', 'wt', 'qsec', 'vs', 'am',

'gear', 'carb'],

dtype='object')]

In [19]:

data1.columns

Out[19]:

Index(['model', 'mpg', 'cyl', 'disp', 'hp', 'drat', 'wt', 'qsec', 'vs', 'am',

'gear', 'carb'],

dtype='object')

In [20]:

data1['hp'].std()

Out[20]:

68.56286848932059

In [21]:

data1['mpg'].mean()

Out[21]:

20.090624999999996

In [22]:

data1['mpg'].median()

Out[22]:

19.2

In [23]:

data1['hp'].describe()

Out[23]:

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 [24]:

data1.head(15)

Out[24]:

|  | 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 [25]:

data1.iloc[1]

Out[25]:

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 [26]:

data1.iloc[:,-1]

Out[26]:

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 [27]:

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 [28]:

data1.iloc[-1]

Out[28]:

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 [29]:

data1.iloc[1]

Out[29]:

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 [30]:

data1\_sorted=data1.sort\_values(by='mpg')

In [31]:

data1\_sorted.head()

Out[31]:

|  | 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 [32]:

data1[data1['carb']==1]

Out[32]:

|  | 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 [33]:

data1[data1['carb']==1].count()

Out[33]:

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 [34]:

data1.describe()

Out[34]:

|  | 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 |

In [ ]:

**Practical No. 3**

**Aim:** Implementation of Linear Regression, Logistic regression, KNN- classification.

**Code:**

**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 [4]:

boston.keys()

Out[4]:

dict\_keys(['data', 'target', 'feature\_names', 'DESCR', 'filename'])

In [20]:

bosDf= pd.DataFrame(boston['data'],columns = boston['feature\_names'])

In [24]:

bosDf['Median\_Price']=boston['target']

bosDf.head()

Out[24]:

|  | **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 [45]:

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 [11]:

bosDf.shape

Out[11]:

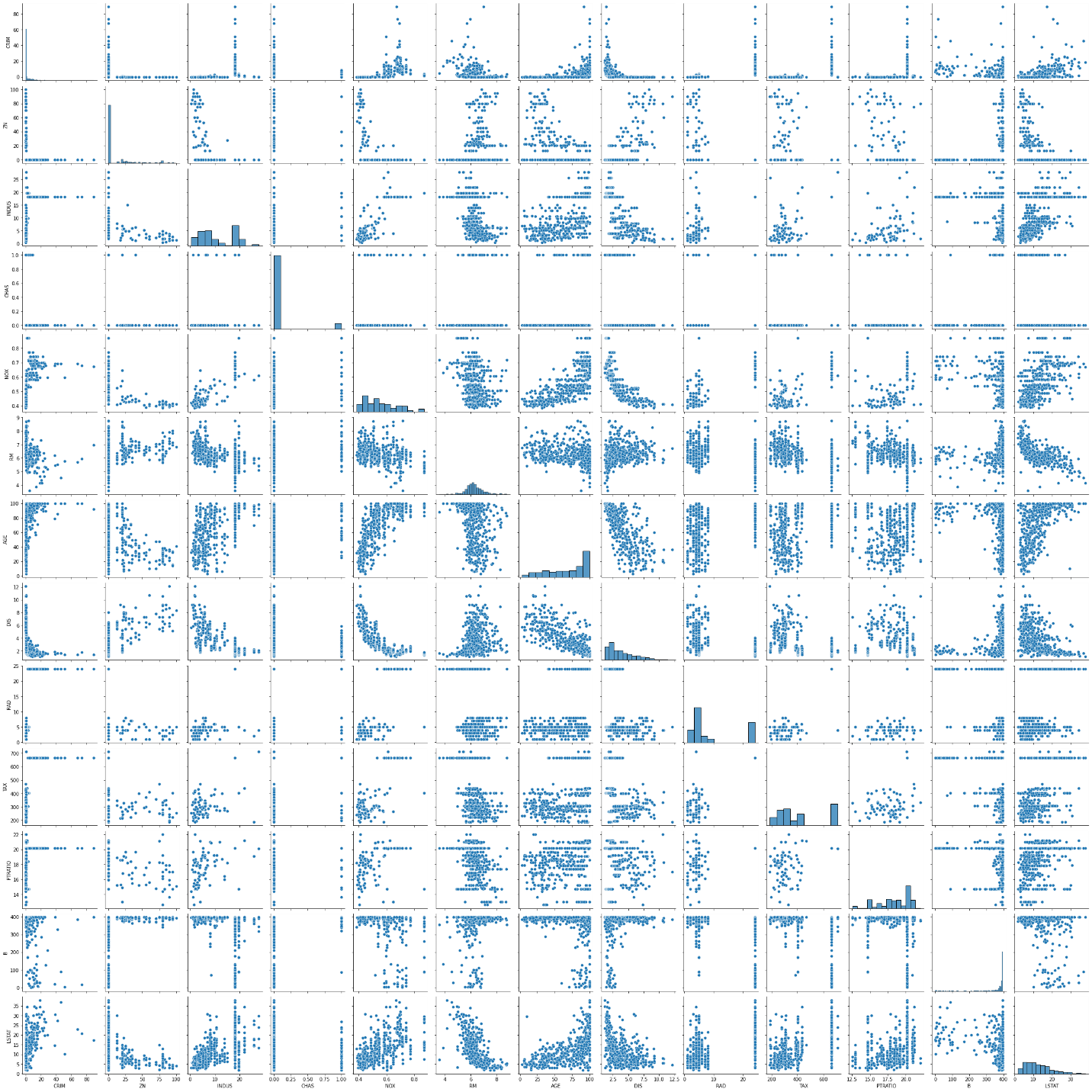
(506, 13)

In [12]:

sns.pairplot(bosDf)

Out[12]:

<seaborn.axisgrid.PairGrid at 0x2ec3f2a53d0>



In [25]:

plt.figure(figsize=(14,10))

sns.heatmap(bosDf.corr(), annot=True)

Out[25]:

<AxesSubplot:>



In [15]:

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 [26]:

X = bosDf.drop('Median\_Price',axis=1)

Y = bosDf.Median\_Price

In [29]:

train\_X , test\_X , train\_Y , test\_Y =tts(X,Y,test\_size=0.3, random\_state=42)

In [31]:

train\_X.head()

Out[31]:

|  | **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 [32]:

train\_Y.head()

Out[32]:

5 28.7

116 21.2

45 19.3

16 23.1

468 19.1

Name: Median\_Price, dtype: float64

In [34]:

Model = LR()

In [35]:

Model.fit(train\_X,train\_Y)

Out[35]:

LinearRegression()

In [36]:

Model.coef\_

Out[36]:

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 [38]:

Model.intercept\_

Out[38]:

31.63108403569189

In [41]:

train\_Y\_hat = Model.predict(train\_X)

test\_Y\_hat = Model.predict(test\_X)

In [42]:

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

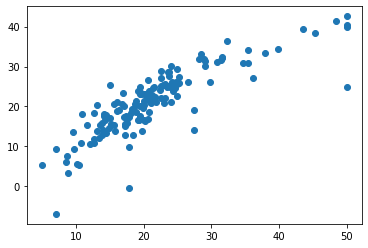
Test MSE 4.6386899261728525

In [43]:

plt.scatter(test\_Y,test\_Y\_hat)

Out[43]:

<matplotlib.collections.PathCollection at 0x2ec4809e220>

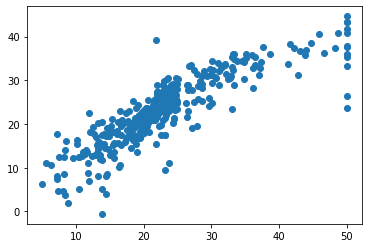


In [44]:

plt.scatter(train\_Y,train\_Y\_hat)

Out[44]:

<matplotlib.collections.PathCollection at 0x2ec488ee370>



**Logistic Regression**

In [4]:

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

In [5]:

credit\_df = pd.read\_csv('CreditRisk.csv')

credit\_df.head()

Out[5]:

|  | **Loan\_ID** | **Gender** | **Married** | **Dependents** | **Education** | **Self\_Employed** | **ApplicantIncome** | **CoapplicantIncome** | **LoanAmount** | **Loan\_Amount\_Term** | **Credit\_History** | **Property\_Area** | **Loan\_Status** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | LP001002 | Male | No | 0 | Graduate | No | 5849 | 0.0 | 0 | 360.0 | 1.0 | Urban | 1 |
| **1** | LP001003 | Male | Yes | 1 | Graduate | No | 4583 | 1508.0 | 128 | 360.0 | 1.0 | Rural | 0 |
| **2** | LP001005 | Male | Yes | 0 | Graduate | Yes | 3000 | 0.0 | 66 | 360.0 | 1.0 | Urban | 1 |
| **3** | LP001006 | Male | Yes | 0 | Not Graduate | No | 2583 | 2358.0 | 120 | 360.0 | 1.0 | Urban | 1 |
| **4** | LP001008 | Male | No | 0 | Graduate | No | 6000 | 0.0 | 141 | 360.0 | 1.0 | Urban | 1 |

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 [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 [7]:

credit\_df.groupby(['Education', 'Loan\_Status']).Education.count()

Out[7]:

Education Loan\_Status

Graduate 0 140

1 340

Not Graduate 0 52

1 82

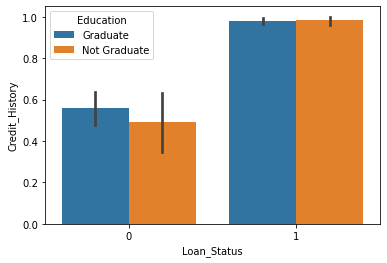
Name: Education, dtype: int64

In [8]:

sns.barplot(y='Credit\_History', x='Loan\_Status', hue='Education',data = credit\_df)

Out[8]:

<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

In [11]:

numeric\_columns = DF.select\_dtypes(exclude =['object']).columns

A technique for filling missing values

In [12]:

for column in object\_columns:

majority = DF[column].value\_counts().iloc[0]

DF[column].fillna(majority, inplace=True)

In [13]:

for column in numeric\_columns:

mean = DF[column].mean()

DF[column].fillna(mean, inplace=True)

In [14]:

DF.head()

Out[14]:

|  | **Gender** | **Married** | **Dependents** | **Education** | **Self\_Employed** | **ApplicantIncome** | **CoapplicantIncome** | **LoanAmount** | **Loan\_Amount\_Term** | **Credit\_History** | **Property\_Area** | **Loan\_Status** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | Male | No | 0 | Graduate | No | 5849 | 0.0 | 0 | 360.0 | 1.0 | Urban | 1 |
| **1** | Male | Yes | 1 | Graduate | No | 4583 | 1508.0 | 128 | 360.0 | 1.0 | Rural | 0 |
| **2** | Male | Yes | 0 | Graduate | Yes | 3000 | 0.0 | 66 | 360.0 | 1.0 | Urban | 1 |
| **3** | Male | Yes | 0 | Not Graduate | No | 2583 | 2358.0 | 120 | 360.0 | 1.0 | Urban | 1 |
| **4** | Male | No | 0 | Graduate | No | 6000 | 0.0 | 141 | 360.0 | 1.0 | Urban | 1 |

In [15]:

credit\_df.head()

Out[15]:

|  | **Loan\_ID** | **Gender** | **Married** | **Dependents** | **Education** | **Self\_Employed** | **ApplicantIncome** | **CoapplicantIncome** | **LoanAmount** | **Loan\_Amount\_Term** | **Credit\_History** | **Property\_Area** | **Loan\_Status** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | LP001002 | Male | No | 0 | Graduate | No | 5849 | 0.0 | 0 | 360.0 | 1.0 | Urban | 1 |
| **1** | LP001003 | Male | Yes | 1 | Graduate | No | 4583 | 1508.0 | 128 | 360.0 | 1.0 | Rural | 0 |
| **2** | LP001005 | Male | Yes | 0 | Graduate | Yes | 3000 | 0.0 | 66 | 360.0 | 1.0 | Urban | 1 |
| **3** | LP001006 | Male | Yes | 0 | Not Graduate | No | 2583 | 2358.0 | 120 | 360.0 | 1.0 | Urban | 1 |
| **4** | LP001008 | Male | No | 0 | Graduate | No | 6000 | 0.0 | 141 | 360.0 | 1.0 | Urban | 1 |

In [16]:

DF[object\_columns].Property\_Area

Out[16]:

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 [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)

In [19]:

DF\_dummy.shape

Out[19]:

(614, 25)

In [20]:

DF\_dummy.head()

Out[20]:

|  | ApplicantIncome | CoapplicantIncome | LoanAmount | Loan\_Amount\_Term | Credit\_History | Loan\_Status | Gender\_489 | Gender\_Female | Gender\_Male | Married\_398 |  | Dependents\_2 | Dependents\_3+ | Education\_Graduate | Education\_Not Graduate | Self\_Employed\_500 | Self\_Employed\_No | Self\_Employed\_Yes | Property\_Area\_Rural | Property\_Area\_Semiurban | Property\_Area\_Urban |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 5849 | 0.0 | 0 | 360.0 | 1.0 | 1 | 0 | 0 | 1 | 0 |  | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| **1** | 4583 | 1508.0 | 128 | 360.0 | 1.0 | 0 | 0 | 0 | 1 | 0 |  | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| **2** | 3000 | 0.0 | 66 | 360.0 | 1.0 | 1 | 0 | 0 | 1 | 0 |  | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| **3** | 2583 | 2358.0 | 120 | 360.0 | 1.0 | 1 | 0 | 0 | 1 | 0 |  | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 |
| **4** | 6000 | 0.0 | 141 | 360.0 | 1.0 | 1 | 0 | 0 | 1 | 0 |  | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |

5 rows × 25 columns

In [21]:

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 [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)

In [23]:

train\_x.shape, test\_x.shape

Out[23]:

((429, 24), (185, 24))

In [24]:

model = LoR()

In [25]:

model.fit(train\_x,train\_y)

Out[25]:

LogisticRegression()

In [26]:

train\_y\_hat = model.predict(train\_x)

test\_y\_hat = model.predict(test\_x)

In [27]:

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 [28]:

print(confusion\_matrix(train\_y,train\_y\_hat))

[[ 57 70]

[ 7 295]]

In [29]:

print(confusion\_matrix(test\_y,test\_y\_hat))

[[ 27 38]

[ 2 118]]

In [30]:

test\_y.value\_counts()

Out[30]:

1 120

0 65

Name: Loan\_Status, dtype: int64

In [31]:

pd.Series(test\_y\_hat).value\_counts()

Out[31]:

1 156

0 29

dtype: int64

In [32]:

(57+295)/train\_y.shape[0]

Out[32]:

0.8205128205128205

In [33]:

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 [34]:

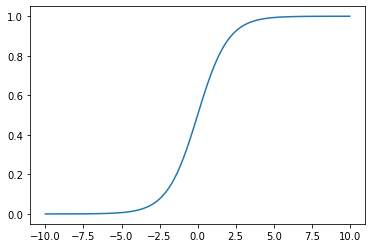
x=np.linspace(-10,10,100)

y=1/(1+np.exp(-x))#sigmoid

plt.plot(x,y)

Out[34]:

[<matplotlib.lines.Line2D at 0x177a543ea60>]



In [35]:

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 [36]:

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 [37]:

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

**KNN- classification.**

In [1]:

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

## Data Preprocessing[¶](#Data-Preprocessing)

In [2]:

credit\_df = pd.read\_csv('CreditRisk.csv')

credit\_df.head()

Out[2]:

|  | **Loan\_ID** | **Gender** | **Married** | **Dependents** | **Education** | **Self\_Employed** | **ApplicantIncome** | **CoapplicantIncome** | **LoanAmount** | **Loan\_Amount\_Term** | **Credit\_History** | **Property\_Area** | **Loan\_Status** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | LP001002 | Male | No | 0 | Graduate | No | 5849 | 0.0 | 0 | 360.0 | 1.0 | Urban | 1 |
| **1** | LP001003 | Male | Yes | 1 | Graduate | No | 4583 | 1508.0 | 128 | 360.0 | 1.0 | Rural | 0 |
| **2** | LP001005 | Male | Yes | 0 | Graduate | Yes | 3000 | 0.0 | 66 | 360.0 | 1.0 | Urban | 1 |
| **3** | LP001006 | Male | Yes | 0 | Not Graduate | No | 2583 | 2358.0 | 120 | 360.0 | 1.0 | Urban | 1 |
| **4** | LP001008 | Male | No | 0 | Graduate | No | 6000 | 0.0 | 141 | 360.0 | 1.0 | Urban | 1 |

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.Loan\_Status.value\_counts()

Out[4]:

1 422

0 192

Name: Loan\_Status, dtype: int64

In [5]:

credit\_df.groupby(['Education', 'Loan\_Status']).Education.count()

Out[5]:

Education Loan\_Status

Graduate 0 140

1 340

Not Graduate 0 52

1 82

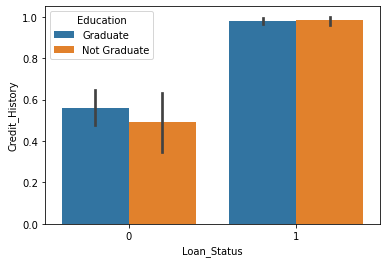
Name: Education, dtype: int64

In [6]:

sns.barplot(y='Credit\_History', x='Loan\_Status', hue='Education',data = credit\_df)

Out[6]:

<AxesSubplot:xlabel='Loan\_Status', ylabel='Credit\_History'>



In [7]:

100 \* credit\_df.isnull().sum() / credit\_df.shape[0]

Out[7]:

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 [8]:

DF = credit\_df.drop('Loan\_ID', axis = 1)

In [9]:

object\_columns = DF.select\_dtypes(include =['object']).columns

numeric\_columns = DF.select\_dtypes(exclude =['object']).columns

In [10]:

for column in object\_columns:

majority = DF[column].value\_counts().iloc[0]

DF[column].fillna(majority, inplace=True)

In [11]:

for column in numeric\_columns:

mean = DF[column].mean()

DF[column].fillna(mean, inplace=True)

In [12]:

DF.head()

Out[12]:

|  | **Gender** | **Married** | **Dependents** | **Education** | **Self\_Employed** | **ApplicantIncome** | **CoapplicantIncome** | **LoanAmount** | **Loan\_Amount\_Term** | **Credit\_History** | **Property\_Area** | **Loan\_Status** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | Male | No | 0 | Graduate | No | 5849 | 0.0 | 0 | 360.0 | 1.0 | Urban | 1 |
| **1** | Male | Yes | 1 | Graduate | No | 4583 | 1508.0 | 128 | 360.0 | 1.0 | Rural | 0 |
| **2** | Male | Yes | 0 | Graduate | Yes | 3000 | 0.0 | 66 | 360.0 | 1.0 | Urban | 1 |
| **3** | Male | Yes | 0 | Not Graduate | No | 2583 | 2358.0 | 120 | 360.0 | 1.0 | Urban | 1 |
| **4** | Male | No | 0 | Graduate | No | 6000 | 0.0 | 141 | 360.0 | 1.0 | Urban | 1 |

In [13]:

DF[object\_columns].Property\_Area

Out[13]:

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 [14]:

DF[object\_columns].Property\_Area.head()

Out[14]:

0 Urban

1 Rural

2 Urban

3 Urban

4 Urban Name: Property\_Area, dtype: object

In [15]:

DF\_dummy = pd.get\_dummies(DF , columns = object\_columns)

In [16]:

DF\_dummy.shape

Out[16]:

(614, 25)

In [17]:

DF\_dummy.head()

Out[17]:

|  | **ApplicantIncome** | **CoapplicantIncome** | **LoanAmount** | **Loan\_Amount\_Term** | **Credit\_History** | **Loan\_Status** | **Gender\_489** | **Gender\_Female** | **Gender\_Male** | **Married\_398** | **...** | **Dependents\_2** | **Dependents\_3+** | **Education\_Graduate** | **Education\_Not Graduate** | **Self\_Employed\_500** | **Self\_Employed\_No** | **Self\_Employed\_Yes** | **Property\_Area\_Rural** | **Property\_Area\_Semiurban** | **Property\_Area\_Urban** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 5849 | 0.0 | 0 | 360.0 | 1.0 | 1 | 0 | 0 | 1 | 0 | ... | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| **1** | 4583 | 1508.0 | 128 | 360.0 | 1.0 | 0 | 0 | 0 | 1 | 0 | ... | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| **2** | 3000 | 0.0 | 66 | 360.0 | 1.0 | 1 | 0 | 0 | 1 | 0 | ... | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| **3** | 2583 | 2358.0 | 120 | 360.0 | 1.0 | 1 | 0 | 0 | 1 | 0 | ... | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 |
| **4** | 6000 | 0.0 | 141 | 360.0 | 1.0 | 1 | 0 | 0 | 1 | 0 | ... | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |

5 rows × 25 columns

In [18]:

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 [25]:

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 [26]:

train\_x.shape, test\_x.shape

Out[26]:

((429, 24), (185, 24))

## KNN Model[¶](#KNN-Model)

In [27]:

knn\_model = KNN(n\_neighbors=7)

In [28]:

knn\_model.fit(train\_x, train\_y)

Out[28]:

KNeighborsClassifier(n\_neighbors=7)

In [29]:

train\_y\_hat = knn\_model.predict(train\_x)

test\_y\_hat = knn\_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 0.70 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 No.4**

**Aim:** Implementation of dimensionality reduction techniques: Features Extraction and Selection, Normalization, Transformation, Principal Components Analysis.

**Code:**

**Practical No.5**

**Aim:** Implementation of K-Means and K-medoid clustering algorithm.

**Code:**

**K-Means**

In [3]:

import pandas as pd

from sklearn.cluster import KMeans

import matplotlib.pyplot as plt

In [4]:

df = pd.read\_csv('driver-data.csv')

df

Out[4]:

|  | **id** | **mean\_dist\_day** | **mean\_over\_speed\_perc** |
| --- | --- | --- | --- |
| **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 |

In [5]:

df.head()

Out[5]:

|  | **id** | **mean\_dist\_day** | **mean\_over\_speed\_perc** |
| --- | --- | --- | --- |
| **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]:

|  | **id** | **mean\_dist\_day** | **mean\_over\_speed\_perc** |
| --- | --- | --- | --- |
| **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 [8]:

df.shape

Out[8]:

(4000, 3)

In [30]:

plt.plot(df.mean\_dist\_day, df.mean\_over\_speed\_perc, 'o')

plt.xlabel('Mean Distance per day')

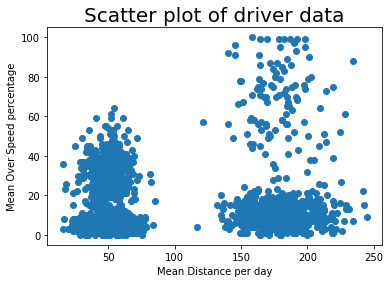
plt.ylabel('Mean Over Speed percentage')

plt.title('Scatter plot of driver data', fontsize=20)

plt.show

Out[30]:

<function matplotlib.pyplot.show(close=None, block=None)>



In [20]:

data = df.drop(['id'], axis = 1)

cluster\_model = KMeans(n\_clusters = 2)

cluster\_model.fit(data)

Out[20]:

KMeans(n\_clusters=2)

In [21]:

df['labels'] = cluster\_model.labels\_

In [23]:

df.head()

Out[23]:

|  | **id** | **mean\_dist\_day** | **mean\_over\_speed\_perc** | **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 [24]:

df.labels.unique()

Out[24]:

array([0, 1])

In [33]:

df['labels'].value\_counts()

Out[33]:

0 3200

1 800

Name: labels, dtype: int64

In [43]:

for label in df.labels.unique():

plt.plot(df.loc[df.labels == label, 'mean\_dist\_day'],

df.loc[df.labels == label, 'mean\_over\_speed\_perc'], 'o',label = label)

plt.xlabel('Mean Distance per day')

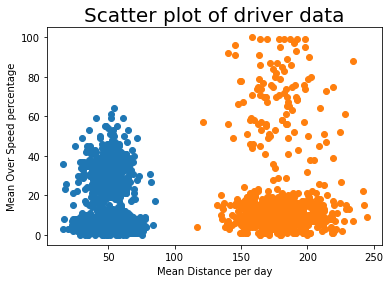
plt.ylabel('Mean Over Speed percentage')

plt.title('Scatter plot of driver data', fontsize=20)

plt.show

Out[43]:

<function matplotlib.pyplot.show(close=None, block=None)>



In [45]:

cluster\_model.cluster\_centers\_

Out[45]:

array([[ 50.04763438, 8.82875 ],

[180.017075 , 18.29 ]])

In [47]:

error = []

for k in range(1,11):

cluster\_model = KMeans(k)

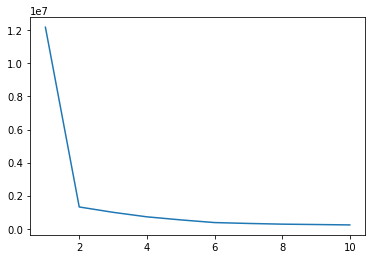
cluster\_model.fit(data)

error.append(cluster\_model.inertia\_)

In [48]:

plt.plot(range(1,11), error)

plt.show()



**Practical No.6**

**Aim:** Implementation of Classifying data using Support Vector Machines (SVMs).

**Code:**

In [3]:

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

In [4]:

credit\_df = pd.read\_csv('CreditRisk.csv')

credit\_df.head()

Out[4]:

|  | **Loan\_ID** | **Gender** | **Married** | **Dependents** | **Education** | **Self\_Employed** | **ApplicantIncome** | **CoapplicantIncome** | **LoanAmount** | **Loan\_Amount\_Term** | **Credit\_History** | **Property\_Area** | **Loan\_Status** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | LP001002 | Male | No | 0 | Graduate | No | 5849 | 0.0 | 0 | 360.0 | 1.0 | Urban | 1 |
| **1** | LP001003 | Male | Yes | 1 | Graduate | No | 4583 | 1508.0 | 128 | 360.0 | 1.0 | Rural | 0 |
| **2** | LP001005 | Male | Yes | 0 | Graduate | Yes | 3000 | 0.0 | 66 | 360.0 | 1.0 | Urban | 1 |
| **3** | LP001006 | Male | Yes | 0 | Not Graduate | No | 2583 | 2358.0 | 120 | 360.0 | 1.0 | Urban | 1 |
| **4** | LP001008 | Male | No | 0 | Graduate | No | 6000 | 0.0 | 141 | 360.0 | 1.0 | Urban | 1 |

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 [6]:

credit\_df.describe()

Out[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 |

In [7]:

credit\_df.Loan\_Status.value\_counts()

Out[7]:

1 422

0 192

Name: Loan\_Status, dtype: int64

In [8]:

credit\_df.groupby(['Education', 'Loan\_Status']).Education.count()

Out[8]:

Education Loan\_Status

Graduate 0 140

1 340

Not Graduate 0 52

1 82

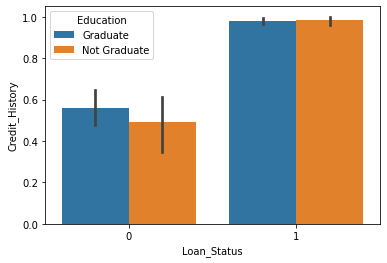
Name: Education, dtype: int64

In [9]:

sns.barplot(y='Credit\_History', x='Loan\_Status', hue='Education',data = credit\_df)

Out[9]:

<AxesSubplot:xlabel='Loan\_Status', ylabel='Credit\_History'>



In [10]:

100 \* credit\_df.isnull().sum() / credit\_df.shape[0]

Out[10]:

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 [11]:

DF = credit\_df.drop('Loan\_ID', axis = 1)

In [12]:

object\_columns = DF.select\_dtypes(include =['object']).columns

numeric\_columns = DF.select\_dtypes(exclude =['object']).columns

In [13]:

for column in object\_columns:

majority = DF[column].value\_counts().iloc[0]

DF[column].fillna(majority, inplace=True)

In [14]:

for column in numeric\_columns:

mean = DF[column].mean()

DF[column].fillna(mean, inplace=True)

In [15]:

DF.head()

Out[15]:

|  | **Gender** | **Married** | **Dependents** | **Education** | **Self\_Employed** | **ApplicantIncome** | **CoapplicantIncome** | **LoanAmount** | **Loan\_Amount\_Term** | **Credit\_History** | **Property\_Area** | **Loan\_Status** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | Male | No | 0 | Graduate | No | 5849 | 0.0 | 0 | 360.0 | 1.0 | Urban | 1 |
| **1** | Male | Yes | 1 | Graduate | No | 4583 | 1508.0 | 128 | 360.0 | 1.0 | Rural | 0 |
| **2** | Male | Yes | 0 | Graduate | Yes | 3000 | 0.0 | 66 | 360.0 | 1.0 | Urban | 1 |
| **3** | Male | Yes | 0 | Not Graduate | No | 2583 | 2358.0 | 120 | 360.0 | 1.0 | Urban | 1 |
| **4** | Male | No | 0 | Graduate | No | 6000 | 0.0 | 141 | 360.0 | 1.0 | Urban | 1 |

In [16]:

DF[object\_columns].Property\_Area

Out[16]:

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 [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)

In [19]:

DF\_dummy.shape

Out[19]:

(614, 25)

In [20]:

DF\_dummy.head()

Out[20]:

|  | **ApplicantIncome** | **CoapplicantIncome** | **LoanAmount** | **Loan\_Amount\_Term** | **Credit\_History** | **Loan\_Status** | **Gender\_489** | **Gender\_Female** | **Gender\_Male** | **Married\_398** | **...** | **Dependents\_2** | **Dependents\_3+** | **Education\_Graduate** | **Education\_Not Graduate** | **Self\_Employed\_500** | **Self\_Employed\_No** | **Self\_Employed\_Yes** | **Property\_Area\_Rural** | **Property\_Area\_Semiurban** | **Property\_Area\_Urban** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 5849 | 0.0 | 0 | 360.0 | 1.0 | 1 | 0 | 0 | 1 | 0 | ... | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| **1** | 4583 | 1508.0 | 128 | 360.0 | 1.0 | 0 | 0 | 0 | 1 | 0 | ... | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| **2** | 3000 | 0.0 | 66 | 360.0 | 1.0 | 1 | 0 | 0 | 1 | 0 | ... | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| **3** | 2583 | 2358.0 | 120 | 360.0 | 1.0 | 1 | 0 | 0 | 1 | 0 | ... | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 |
| **4** | 6000 | 0.0 | 141 | 360.0 | 1.0 | 1 | 0 | 0 | 1 | 0 | ... | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |

5 rows × 25 columns

In [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

In [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)

In [23]:

train\_x.shape, test\_x.shape

Out[23]:

((429, 24), (185, 24))

## SVM Model

## [¶](#SVM-Model)

In [24]:

svm\_model = SVC(kernel='rbf', gamma=0.00001, C=1000)

In [25]:

svm\_model.fit(train\_x, train\_y)

Out[25]:

SVC(C=1000, gamma=1e-05)

In [26]:

train\_y\_hat = svm\_model.predict(train\_x)

test\_y\_hat = svm\_model.predict(test\_x)

In [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

In [28]:

confusion\_matrix(test\_y, test\_y\_hat)

Out[28]:

array([[12, 53],

[21, 99]], dtype=int64)

In [29]:

confusion\_matrix(train\_y, train\_y\_hat)

Out[29]:

array([[121, 6],

[ 7, 295]], dtype=int64)

**Practical No.7**

**Aim:** Implementation of Bagging Algorithm: Decision Tree, Random Forest.

**Code:**

**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('CreditRisk.csv')

credit\_df.head()

Out[2]:

|  | **Loan\_ID** | **Gender** | **Married** | **Dependents** | **Education** | **Self\_Employed** | **ApplicantIncome** | **CoapplicantIncome** | **LoanAmount** | **Loan\_Amount\_Term** | **Credit\_History** | **Property\_Area** | **Loan\_Status** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | LP001002 | Male | No | 0 | Graduate | No | 5849 | 0.0 | 0 | 360.0 | 1.0 | Urban | 1 |
| **1** | LP001003 | Male | Yes | 1 | Graduate | No | 4583 | 1508.0 | 128 | 360.0 | 1.0 | Rural | 0 |
| **2** | LP001005 | Male | Yes | 0 | Graduate | Yes | 3000 | 0.0 | 66 | 360.0 | 1.0 | Urban | 1 |
| **3** | LP001006 | Male | Yes | 0 | Not Graduate | No | 2583 | 2358.0 | 120 | 360.0 | 1.0 | Urban | 1 |
| **4** | LP001008 | Male | No | 0 | Graduate | No | 6000 | 0.0 | 141 | 360.0 | 1.0 | Urban | 1 |

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

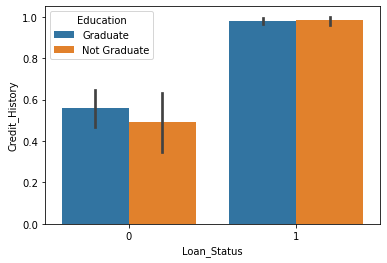
Name: Education, dtype: int64

In [7]:

sns.barplot(y='Credit\_History', x='Loan\_Status', hue='Education',data = credit\_df)

Out[7]:

<AxesSubplot:xlabel='Loan\_Status', ylabel='Credit\_History'>

****

## [¶](#Filling-Missing-Values)

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\_Area** | **Loan\_Status** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | Male | No | 0 | Graduate | No | 5849 | 0.0 | 0 | 360.0 | 1.0 | Urban | 1 |
| **1** | Male | Yes | 1 | Graduate | No | 4583 | 1508.0 | 128 | 360.0 | 1.0 | Rural | 0 |
| **2** | Male | Yes | 0 | Graduate | Yes | 3000 | 0.0 | 66 | 360.0 | 1.0 | Urban | 1 |
| **3** | Male | Yes | 0 | Not Graduate | No | 2583 | 2358.0 | 120 | 360.0 | 1.0 | Urban | 1 |
| **4** | Male | No | 0 | Graduate | No | 6000 | 0.0 | 141 | 360.0 | 1.0 | Urban | 1 |

In [14]:

credit\_df.head()

Out[14]:

|  | **Loan\_ID** | **Gender** | **Married** | **Dependents** | **Education** | **Self\_Employed** | **ApplicantIncome** | **CoapplicantIncome** | **LoanAmount** | **Loan\_Amount\_Term** | **Credit\_History** | **Property\_Area** | **Loan\_Status** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | LP001002 | Male | No | 0 | Graduate | No | 5849 | 0.0 | 0 | 360.0 | 1.0 | Urban | 1 |
| **1** | LP001003 | Male | Yes | 1 | Graduate | No | 4583 | 1508.0 | 128 | 360.0 | 1.0 | Rural | 0 |
| **2** | LP001005 | Male | Yes | 0 | Graduate | Yes | 3000 | 0.0 | 66 | 360.0 | 1.0 | Urban | 1 |
| **3** | LP001006 | Male | Yes | 0 | Not Graduate | No | 2583 | 2358.0 | 120 | 360.0 | 1.0 | Urban | 1 |
| **4** | LP001008 | Male | No | 0 | Graduate | No | 6000 | 0.0 | 141 | 360.0 | 1.0 | Urban | 1 |

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** | **...** | **Dependents\_2** | **Dependents\_3+** | **Education\_Graduate** | **Education\_Not Graduate** | **Self\_Employed\_500** | **Self\_Employed\_No** | **Self\_Employed\_Yes** | **Property\_Area\_Rural** | **Property\_Area\_Semiurban** | **Property\_Area\_Urban** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 5849 | 0.0 | 0 | 360.0 | 1.0 | 1 | 0 | 0 | 1 | 0 | ... | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| **1** | 4583 | 1508.0 | 128 | 360.0 | 1.0 | 0 | 0 | 0 | 1 | 0 | ... | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| **2** | 3000 | 0.0 | 66 | 360.0 | 1.0 | 1 | 0 | 0 | 1 | 0 | ... | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| **3** | 2583 | 2358.0 | 120 | 360.0 | 1.0 | 1 | 0 | 0 | 1 | 0 | ... | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 |
| **4** | 6000 | 0.0 | 141 | 360.0 | 1.0 | 1 | 0 | 0 | 1 | 0 | ... | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |

5 rows × 25 columns

## MODEL Construction[¶](#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))

[¶](#Decision-Tree)

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 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 0.53 0.48 0.50 65

1 0.73 0.78 0.75 120

accuracy 0.67 185

macro avg 0.63 0.63 0.63 185

weighted avg 0.66 0.67 0.67 185

In [27]:

confusion\_matrix(train\_y, train\_y\_hat)

Out[27]:

array([[127, 0],

[ 1, 301]], dtype=int64)

In [28]:

confusion\_matrix(test\_y, test\_y\_hat)

Out[28]:

array([[31, 34],

[27, 93]], dtype=int64)

**Random Forest**

In [1]:

import pandas as pd

from sklearn.datasets import load\_digits

digits = load\_digits()

In [3]:

import matplotlib.pyplot as plt

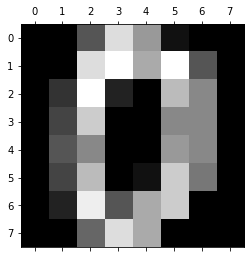
In [4]:

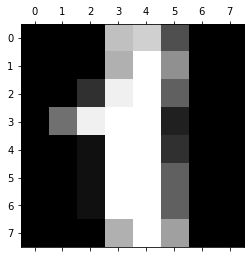
plt.gray()

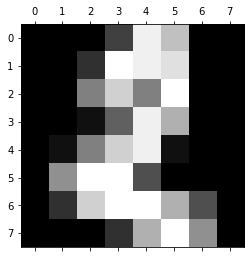
for i in range(4):

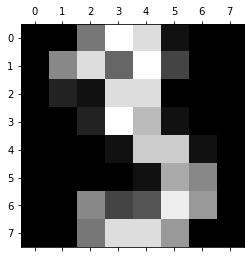
plt.matshow(digits.images[i])

<Figure size 432x288 with 0 Axes>

****

****

****

****

In [5]:

df = pd.DataFrame(digits.data)

df.head()

Out[5]:

|  | **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 [6]:

df['target'] = digits.target

In [7]:

df[0:12]

Out[7]:

|  | **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 [8]:

X = df.drop('target',axis='columns')

y = df.target

In [9]:

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 [10]:

from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(n\_estimators=40)

model.fit(X\_train, y\_train)

Out[10]:

RandomForestClassifier(n\_estimators=40)

In [11]:

model.score(X\_test, y\_test)

Out[11]:

0.9694444444444444

In [12]:

y\_predicted = model.predict(X\_test)

In [14]:

from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(y\_test, y\_predicted)

cm

Out[14]:

array([[40, 0, 0, 0, 0, 0, 0, 0, 0, 0],

[ 0, 45, 0, 0, 0, 0, 0, 0, 0, 0],

[ 0, 0, 36, 0, 0, 0, 0, 0, 0, 0],

[ 0, 0, 0, 34, 0, 0, 0, 0, 1, 0],

[ 0, 0, 0, 0, 34, 0, 0, 1, 0, 1],

[ 0, 0, 0, 0, 1, 28, 0, 0, 0, 0],

[ 0, 1, 0, 0, 0, 0, 39, 0, 0, 0],

[ 0, 0, 0, 0, 0, 0, 0, 32, 0, 2],

[ 0, 0, 0, 0, 0, 0, 0, 0, 27, 1],

[ 0, 0, 0, 0, 0, 0, 0, 1, 2, 34]], dtype=int64)

In [17]:

import seaborn as sn

plt.figure(figsize=(10,7))

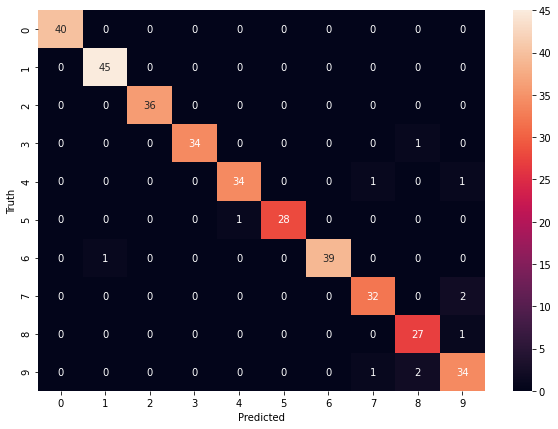
sn.heatmap(cm, annot=True)

plt.xlabel('Predicted')

plt.ylabel('Truth')

Out[17]:

Text(69.0, 0.5, 'Truth')

****

**Practical No. 8**

**Aim:** Implementation of Boosting Algorithms: AdaBoost, Stochastic Gradient Boosting, Voting Ensemble.

**Code:**

In [1]:

import pandas as pd

from sklearn.ensemble import AdaBoostClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn import metrics

In [2]:

LoanData = pd.read\_csv(r"C:\Users\Kartik\Downloads\income.csv")

LoanData.head()

Out[2]:

|  | **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 [3]:

X=LoanData.iloc[:,0:6]

y=LoanData.iloc[:,6]

In [4]:

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2)

In [5]:

AdaModel = AdaBoostClassifier(n\_estimators=100,

learning\_rate=1)

model = AdaModel.fit(X\_train, y\_train)

In [6]:

y\_pred = model.predict(X\_test)

In [7]:

print("Accuracy:",metrics.accuracy\_score(y\_test, y\_pred))

Accuracy: 0.8387757191114751

In [8]:

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 [9]:

boston = load\_boston()

X = pd.DataFrame(boston.data, columns=boston.feature\_names)

y = pd.Series(boston.target)

In [11]:

X.head()

Out[11]:

|  | **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 [12]:

y[1:10]

Out[12]:

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 [13]:

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2)

In [14]:

gradientregressor = GradientBoostingRegressor(max\_depth=2,n\_estimators=3,learning\_rate=0.5)

In [15]:

model = gradientregressor.fit(X\_train, y\_train)

In [16]:

y\_pred = model.predict(X\_test)

In [17]:

r2\_score(y\_pred,y\_test)

Out[17]:

0.5612358579108006

In [18]:

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

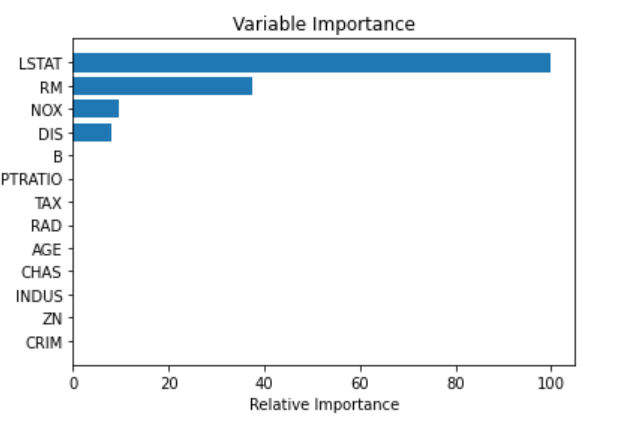
plt.barh(pos, feature\_importance[sorted\_idx], align='center')

plt.yticks(pos, boston.feature\_names[sorted\_idx])

plt.xlabel('Relative Importance')

plt.title('Variable Importance')

plt.show()



In [ ]: