Practical No: 01

Aim: Logical programming using prolog-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 9.2.3)
File  Edit  Settings  Run  Debug  Help
For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?-
% d:/AI MI Lab (Sandesh)/water_jug.pl compiled 0.00 sec, 4 clauses
?- water_jug(3,2).
false.

?- water_jug(4,3).
false.

?- water_jug(1,3).

4L:1 & 3L:0 (Action: Empty 3L jug.)
4L:0 & 3L:1 (Action: Pour water from 4L jug to 3L jug.)
4L:4 & 3L:1 (Action: Fill 4L jug.)
4L:2 & 3L:3 (Action: Pour water from 4L to 3L jug untill 3L jug is full.)
4L:2 & 3L:0 (Action: Empty 3L jug.)
4L:2 & 3L:0 (Action: Goal State reached...)
true .

?- water_jug(2,3).

4L:2 & 3L:0 (Action: Empty 3L jug.)
4L:2 & 3L:0 (Action: Goal State reached...)
true .

?- water_jug(2,0).

4L:2 & 3L:0 (Action: Goal State reached...)
true
```

Practical No:2

Aim: Introduction to Basic Library in Python

Output:

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

In [3]:
```
list = [1,2,3,4]
array = np.array(list)
print(array)
```

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

In [4]:
```
list = [ [1,2,3,4], [5,6,7,8] ]
array = np.array(list)
print(array)
```

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

In [5]:
```
toprice = np.array( [5,8,3,6] )
print(toprice - 2)
```

```
[3 6 1 4]
```

In [11]:
```
dataf = pd.DataFrame([
    ['Sandesh', "THANE", 22],
    ['Manoj', 'SANPADA', 28],
    ['Pratik', 'BHIWANDI', 22],
    ['Aniket', 'THANE', 22],
    ['Abhijeet', 'KALYAN', 21],
],
columns = ['name', 'address', 'age'])

print(dataf)
```

```
       name   address  age
0   Sandesh     THANE   22
1     Manoj   SANPADA   28
2    Pratik  BHIWANDI   22
3    Aniket     THANE   22
4  Abhijeet    KALYAN   21
```

In [12]:
```
dataf.set_index('name')
```

Out[12]:

name	address	age
Sandesh	THANE	22
Manoj	SANPADA	28
Pratik	BHIWANDI	22
Aniket	THANE	22
Abhijeet	KALYAN	21

Practical No:3

Aim: Exploratory data analysis using python

Output:

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

```
In [5]: data1 = pd.read_csv("E:\AI ML\mtcars - mtcars.csv")
```

```
In [6]: print(data1)
```

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

```
    gear  carb
0      4     4
1      4     4
2      4     1
3      3     1
4      3     2
5      3     1
6      3     4
7      4     2
8      4     2
9      4     4
10     4     4
11     3     3
12     3     3
13     3     3
14     3     4
15     3     4
16     3     4
17     4     1
18     4     2
19     4     1
20     3     1
21     3     2
22     3     2
23     3     4
24     3     2
25     4     1
26     5     2
27     5     2
28     5     4
29     5     6
30     5     8
31     4     2
```

In [7]:
```
data1.head()
```

Out[7]:

	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 [8]:
```
data1.tail()
```

Out[8]:

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

In [9]:
```
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 [10]:
```
data1.isnull()
```

Out[10]:

	model	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	False	False	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	False	False
5	False	False	False	False	False	False	False	False	False	False	False	False
6	False	False	False	False	False	False	False	False	False	False	False	False
7	False	False	False	False	False	False	False	False	False	False	False	False
8	False	False	False	False	False	False	False	False	False	False	False	False
9	False	False	False	False	False	False	False	False	False	False	False	False
10	False	False	False	False	False	False	False	False	False	False	False	False
11	False	False	False	False	False	False	False	False	False	False	False	False
12	False	False	False	False	False	False	False	False	False	False	False	False
13	False	False	False	False	False	False	False	False	False	False	False	False
14	False	False	False	False	False	False	False	False	False	False	False	False
15	False	False	False	False	False	False	False	False	False	False	False	False
16	False	False	False	False	False	False	False	False	False	False	False	False
17	False	False	False	False	False	False	False	False	False	False	False	False
18	False	False	False	False	False	False	False	False	False	False	False	False
19	False	False	False	False	False	False	False	False	False	False	False	False
20	False	False	False	False	False	False	False	False	False	False	False	False
21	False	False	False	False	False	False	False	False	False	False	False	False
22	False	False	False	False	False	False	False	False	False	False	False	False
23	False	False	False	False	False	False	False	False	False	False	False	False
24	False	False	False	False	False	False	False	False	False	False	False	False
25	False	False	False	False	False	False	False	False	False	False	False	False
26	False	False	False	False	False	False	False	False	False	False	False	False
27	False	False	False	False	False	False	False	False	False	False	False	False
28	False	False	False	False	False	False	False	False	False	False	False	False
29	False	False	False	False	False	False	False	False	False	False	False	False
30	False	False	False	False	False	False	False	False	False	False	False	False
31	False	False	False	False	False	False	False	False	False	False	False	False

In [11]:
```
data1.isnull().sum()
```

Out[11]:
```
model    0
mpg      0
cyl      0
disp     0
hp       0
drat     0
wt       0
qsec     0
vs       0
am       0
gear     0
carb     0
dtype: int64
```

In [12]:
```
data1.describe()
```

Out[12]:

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
count	32.000000	32.000000	32.000000	32.000000	32.000000	32.000000	32.000000	32.000000	32.000000	32.000000	32.0000
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 [13]:
```
data1.size
```

Out[13]: 384

In [14]:
```
data1.shape
```

Out[14]: (32, 12)

In [15]:
```
data1.ndim
```

Out[15]: 2

In [16]:
```
data1.at[4,'model']
```

Out[16]: 'Hornet Sportabout'

In [17]:
```
data1.at[4,'disp']
```

Out[17]: 360.0

In [18]:
```
data1.iat[4,5]
```

Out[18]: 3.15

In [19]:
```
data1.loc[:,'model']
```

Out[19]:
```
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 [20]:
```
data1.loc[2,'model']
```

Out[20]: 'Datsun 710'

In [21]:
```
data1.iloc[0:3,0:2]
```

Out[21]:

	model	mpg
0	Mazda RX4	21.0
1	Mazda RX4 Wag	21.0
2	Datsun 710	22.8

In [22]:
```
data1.iloc[22:32,:]
```

Out[22]:

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

```
In [23]: data1.dtypes
```

```
Out[23]: 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 [24]: data1['model'].dtype
```

```
Out[24]: dtype('O')
```

```
In [25]: data1['wt'].dtype
```

```
Out[25]: dtype('float64')
```

```
In [26]: data1.axes
```

```
Out[26]: [RangeIndex(start=0, stop=32, step=1),
          Index(['model', 'mpg', 'cyl', 'disp', 'hp', 'drat', 'wt', 'qsec', 'vs', 'am',
                 'gear', 'carb'],
                dtype='object')]
```

```
In [27]: data1.columns
```

```
Out[27]: Index(['model', 'mpg', 'cyl', 'disp', 'hp', 'drat', 'wt', 'qsec', 'vs', 'am',
                'gear', 'carb'],
               dtype='object')
```

```
In [28]: data1['mpg'].mean()
```

```
Out[28]: 20.090625000000003
```

```
In [29]: data1['mpg'].median()
```

```
Out[29]: 19.2
```

```
In [30]: data1['mpg'].std()
```

```
Out[30]: 6.026948052089105
```

```
In [31]: data1['mpg'].describe()
```

```
Out[31]: count    32.000000
         mean     20.090625
         std       6.026948
         min      10.400000
         25%      15.425000
         50%      19.200000
         75%      22.800000
         max      33.900000
         Name: mpg, dtype: float64
```

Practical No:4

Aim: Implementation Of KNN Alogrithm.

Output:

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

In [4]:
```
credit_df = pd.read_csv('E:\AI MI Lab (Sandesh)\CreditRisk.csv')
```

In [5]:
```
credit_df.shape
```

Out[5]: (614, 13)

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

Out[6]:

	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 [7]:
```
credit_df.head(7)
```

Out[7]:

	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
5	LP001011	Male	Yes	2	Graduate	Yes	5417	4196.0	267	360.0	1.0
6	LP001013	Male	Yes	0	Not Graduate	No	2333	1516.0	95	360.0	1.0

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

Out[8]:

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

In [9]:
```
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 [10]:
```
credit_df.describe()
```

Out[10]:

	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 [11]:
```
credit_df.Loan_Status.value_counts()
```

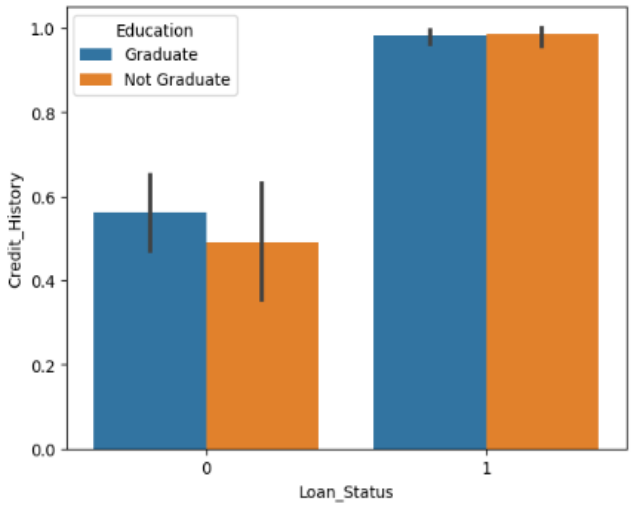
Out[11]:
```
Loan_Status
1    422
0    192
Name: count, dtype: int64
```

In [12]:
```
credit_df.groupby(['Education','Loan_Status']).Education.count()
```

Out[12]:
```
Education     Loan_Status
Graduate      0              140
              1              340
Not Graduate  0               52
              1               82
Name: Education, dtype: int64
```

In [13]:
```
sns.barplot(y='Credit_History', x='Loan_Status' , hue='Education', data= credit_df )
```

Out[13]:
```
<Axes: xlabel='Loan_Status', ylabel='Credit_History'>
```



In [14]:
```
100 * credit_df.isnull().sum() / credit_df.shape[0]
```

Out[14]:
```
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 [15]:
```
credit_df.isnull().sum()
```

Out[15]:
```
Loan_ID               0
Gender               13
Married               3
Dependents           15
Education             0
Self_Employed        32
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount            0
Loan_Amount_Term     14
Credit_History       50
Property_Area         0
Loan_Status           0
dtype: int64
```

In [16]:
```
# Segregation
object_columns = credit_df.select_dtypes(include=['object']).columns
numeric_columns = credit_df.select_dtypes(exclude=['object']).columns
```

In [17]:
```
for column in object_columns:
    majority = credit_df[column].value_counts().iloc[0]
    credit_df[column].fillna(majority, inplace=True)
```

In [18]:
```
for column in numeric_columns:
    mean = credit_df[column].mean()
    credit_df[column].fillna(mean, inplace=True)
```

In [19]:
```
# remove unwanted columns
credit_df.drop('Loan_ID', axis=1, inplace=True)
```

In [20]:
```
credit_df.head()
```

Out[20]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_A
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 [21]:
```
credit_df
```

Out[21]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_
0	Male	No	0	Graduate	No	5849	0.0	0	360.0	1.0	U
1	Male	Yes	1	Graduate	No	4583	1508.0	128	360.0	1.0	
2	Male	Yes	0	Graduate	Yes	3000	0.0	66	360.0	1.0	U
3	Male	Yes	0	Not Graduate	No	2583	2358.0	120	360.0	1.0	U
4	Male	No	0	Graduate	No	6000	0.0	141	360.0	1.0	U
...	...	...	...	...	...	...	...	...	...	...	
609	Female	No	0	Graduate	No	2900	0.0	71	360.0	1.0	
610	Male	Yes	3+	Graduate	No	4106	0.0	40	180.0	1.0	
611	Male	Yes	1	Graduate	No	8072	240.0	253	360.0	1.0	U
612	Male	Yes	2	Graduate	No	7583	0.0	187	360.0	1.0	U
613	Female	No	0	Graduate	Yes	4583	0.0	133	360.0	0.0	Semi

614 rows × 12 columns

In [22]:
```
#impute => replacing the value (filling null values)
```

In [23]:
```
# Categorical Columns
```

In [24]:
```
object_columns = credit_df.select_dtypes(include=['object']).columns
```

In [25]:
```
credit_df.head()
```

Out[25]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_A
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 [26]:
```
credit_df[object_columns].Property_Area
```

Out[26]:
```
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 [27]:
```
credit_df[object_columns].Property_Area.head()
```

Out[27]:
```
0    Urban
1    Rural
2    Urban
3    Urban
4    Urban
Name: Property_Area, dtype: object
```

In [28]:
```
credit_df_dummy = pd.get_dummies(credit_df, columns = object_columns)
```

In [29]:
```
credit_df_dummy.shape
```

Out[29]:
```
(614, 25)
```

In [30]:
```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

In [31]:
```
X = credit_df_dummy.drop('Loan_Status',axis=1)
y = credit_df_dummy.Loan_Status
train_x, test_x, train_y, test_y = train_test_split(X,y,test_size=0.3,random_state=42)
```

In [32]:
```
# KNN Model
```

In [33]:
```
knn_model = KNeighborsClassifier(n_neighbors=7)
```

In [34]:
```
knn_model.fit(train_x, train_y)
```

Out[34]:
```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=7)
```

In [35]:
```
train_y_hat = knn_model.predict(train_x)
test_y_hat = knn_model.predict(test_x)

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
```

Practical No: 5

Aim: Program to Implement Decision Tree

Output:

In [31]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [32]:
```
credit_df = pd.read_csv('E:\AI ML Lab (Sandesh)\CreditRisk.csv')
```

In [33]:
```
credit_df.shape
```

Out[33]: (614, 13)

In [34]:
```
credit_df.head()
```

Out[34]:

	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 [35]:
```
credit_df.tail()
```

Out[35]:

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

In [36]:
```
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 [37]:
```
credit_df.describe()
```

Out[37]:

	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 [38]:
```
credit_df.Loan_Status.value_counts()
```

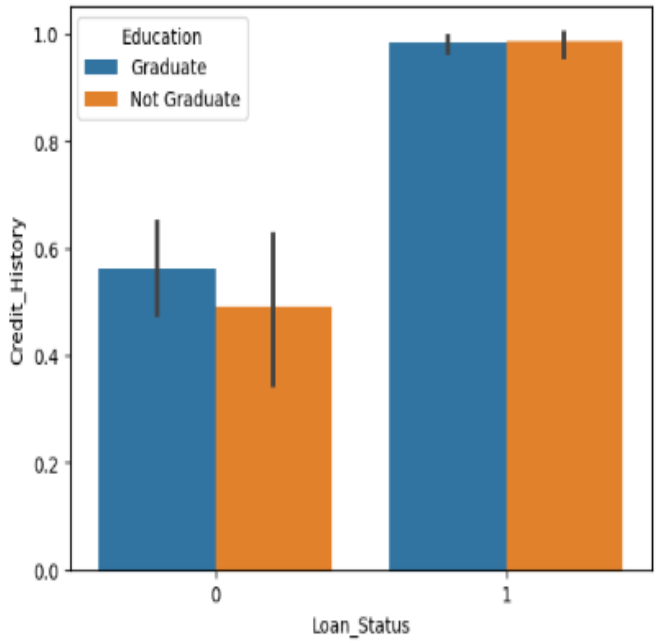
Out[38]:
```
Loan_Status
1    422
0    192
Name: count, dtype: int64
```

In [39]:
```
credit_df.groupby(['Education','Loan_Status']).Education.count()
```

Out[39]:
```
Education     Loan_Status
Graduate      0              140
              1              340
Not Graduate  0               52
              1               82
Name: Education, dtype: int64
```

In [40]:
```
sns.barplot(y='Credit_History', x = 'Loan_Status', hue ='Education', data = credit_df)
```

Out[40]:
```
<Axes: xlabel='Loan_Status', ylabel='Credit_History'>
```



In [41]:
```
# Fill Null Values

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

Out[41]:
```
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 [42]:
```
object_columns = credit_df.select_dtypes(include=['object']).columns
numeric_columns = credit_df.select_dtypes(exclude = ['object']).columns
```

In [43]:
```
for column in object_columns:
    majority = credit_df[column].value_counts().iloc[0]
    credit_df[column].fillna(majority, inplace=True)
```

In [44]:
```
for column in numeric_columns:
    mean = credit_df[column].mean()
    credit_df[column].fillna(mean, inplace=True)
```

In [45]:
```
#impute
```

In [46]:
```
credit_df.head()
```

Out[46]:

	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 [47]:
```
credit_df.drop('Loan_ID', axis=1, inplace=True)
```

In [48]:
```
object_columns = credit_df.select_dtypes(include=['object']).columns
```

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

Out[49]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_A
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 [50]:
```
# Categorical Columns
```

In [51]:
```
credit_df[object_columns].Property_Area
```

Out[51]:
```
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 [52]:
```
credit_df_dummy = pd.get_dummies(credit_df, columns=object_columns)
```

In [53]:
```
credit_df_dummy.shape
```

Out[53]:
```
(614, 25)
```

In [54]:
```
# Model
```

In [55]:
```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

In [56]:
```
X = credit_df_dummy.drop('Loan_Status', axis=1)
y = credit_df_dummy.Loan_Status
train_x, test_x, train_y, test_y = train_test_split(X, y, test_size = 0.3, random_state = 42)
```

In [57]:
```
train_x.shape, test_x.shape
```

Out[57]:
```
((429, 24), (185, 24))
```

In [58]:
```
dt_model = DecisionTreeClassifier(max_depth=2, min_samples_leaf=2, min_samples_split=2)
```

In [59]:
```
dt_model.fit(train_x, train_y)
```

Out[59]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier(max_depth=2, min_samples_leaf=2)
```

In [60]:
```
train_y_hat = dt_model.predict(train_x)
test_y_hat = dt_model.predict(test_x)
```

In [61]:
```
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       0.92      0.43      0.59       127
           1       0.80      0.98      0.89       302

    accuracy                           0.82       429
   macro avg       0.86      0.71      0.74       429
weighted avg       0.84      0.82      0.80       429

-------------------- test --------------------
              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 [62]:
```
confusion_matrix(test_y, test_y_hat)
```

Out[62]:
```
array([[ 27,  38],
       [  2, 118]], dtype=int64)
```

Practical No: 6

Aim: Implementation Of Support Vector Machine.

Output:

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

In [2]:
```
credit_df = pd.read_csv('E:\AI MI Lab (Sandesh)\CreditRisk.csv')
```

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

Out[3]: (614, 13)

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

Out[4]:

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

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

Out[5]:

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

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

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

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

Out[7]:

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

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

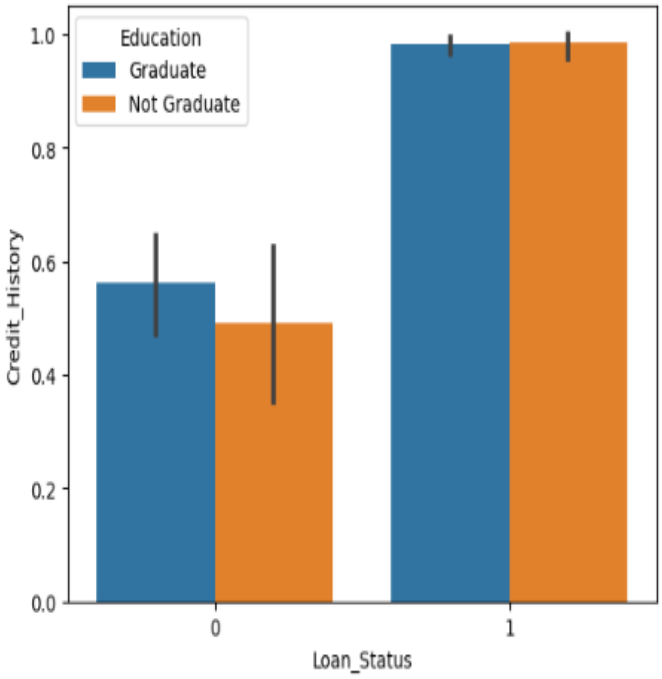
Out[8]:
```
Loan_Status
1    422
0    192
Name: count, dtype: int64
```

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

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

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

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



In [11]:
```
# Fill Null Values

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

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

In [12]:
```
object_columns = credit_df.select_dtypes(include=['object']).columns
numeric_columns = credit_df.select_dtypes(exclude = ['object']).columns
```

In [13]:
```
# credit_df.columns[credit_df.dtypes == object]
```

In [14]:
```
for column in object_columns:
    majority = credit_df[column].value_counts().iloc[0]
    credit_df[column].fillna(majority, inplace=True)
```

In [15]:
```
for column in numeric_columns:
    mean = credit_df[column].mean()
    credit_df[column].fillna(mean, inplace=True)
```

In [16]:
```
#impute
```

In [17]:
```
credit_df.head()
```

Out[17]:

	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 [18]:
```
credit_df.drop('Loan_ID', axis=1, inplace=True)
```

In [19]:
```
object_columns = credit_df.select_dtypes(include=['object']).columns
```

In [20]:
```
credit_df.head()
```

Out[20]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_A
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 [21]:
```
# Categorical Columns
```

In [22]:
```
credit_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]:
```
credit_df_dummy = pd.get_dummies(credit_df, columns=object_columns)
```

In [24]:
```
# Sklearn - LabelEncoding
# Sklearn - LabelBinarize
# Sklearn - OneHotEncoding
```

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

Out[25]: (614, 25)

In [26]:
```
# Model
```

In [27]:
```
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

In [28]:
```
X = credit_df_dummy.drop('Loan_Status', axis=1)
y = credit_df_dummy.Loan_Status
train_x, test_x, train_y, test_y = train_test_split(X, y, test_size = 0.3, random_state = 42)
```

In [29]:
```
train_x.shape, test_x.shape
```

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

In [30]:
```
# SVM
```

In [31]:
```
svm_model = SVC(kernel='rbf', gamma=0.00001, C=1000)
```

In [32]:
```
svm_model.fit(train_x, train_y)
```

Out[32]:
```
SVC
SVC(C=1000, gamma=1e-05)
```

In [33]:
```
train_y_hat = svm_model.predict(train_x)
test_y_hat = svm_model.predict(test_x)
```

In [34]:
```
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 [35]:
```
confusion_matrix(test_y, test_y_hat)
```

Out[35]:
```
array([[12, 53],
       [21, 99]], dtype=int64)
```

Practical No:7

Aim: Program to Implement Linear Regression

Output:

In [27]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [28]:
```
data = pd.read_csv('D:/AI MI Lab (Sandesh)/Salary_Data.csv')
```

In [29]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30 entries, 0 to 29
Data columns (total 2 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   YearsExperience  30 non-null     float64
 1   Salary           30 non-null     int64
dtypes: float64(1), int64(1)
memory usage: 608.0 bytes
```

In [30]:
```
data.head()
```

Out[30]:

	YearsExperience	Salary
0	1.1	39343
1	1.3	46205
2	1.5	37731
3	2.0	43525
4	2.2	39891

In [35]:
```
data.describe()
```

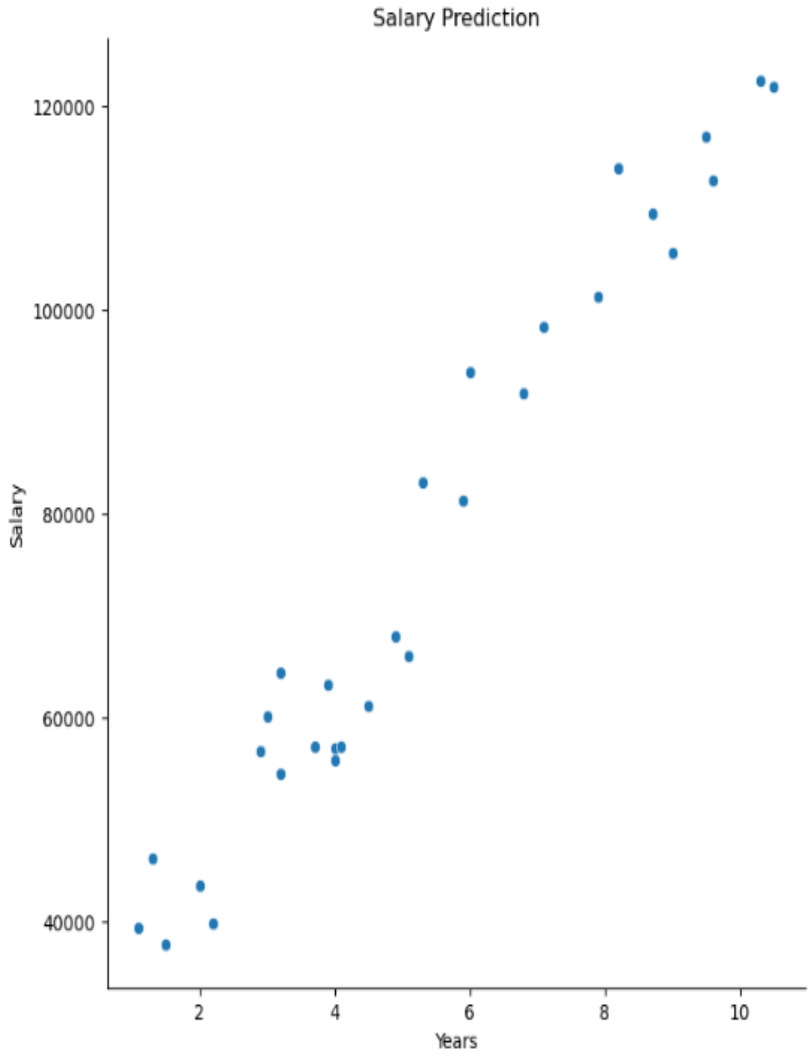
Out[35]:

	YearsExperience	Salary
count	30.000000	30.000000
mean	5.313333	76003.000000
std	2.837888	27414.429785
min	1.100000	37731.000000
25%	3.200000	56720.750000
50%	4.700000	65237.000000
75%	7.700000	100544.750000
max	10.500000	122391.000000

In [48]:
```
plt.figure(figsize=(12,6))
sns.pairplot(data,x_vars=['YearsExperience'],y_vars=['Salary'],size=7,kind='scatter')
plt.xlabel('Years')
plt.ylabel('Salary')
plt.title('Salary Prediction')
plt.show()
```

```
C:\Users\Lab2_45\anaconda3\lib\site-packages\seaborn\axisgrid.py:2095: UserWarning: The `size` parameter has been renamed to `h
eight`; please update your code.
  warnings.warn(msg, UserWarning)
```

```
<Figure size 1200x600 with 0 Axes>
```




```
In [49]: X = data['YearsExperience']
         X.head()
```

```
Out[49]: 0    1.1
         1    1.3
         2    1.5
         3    2.0
         4    2.2
         Name: YearsExperience, dtype: float64
```

```
In [50]: y = data['Salary']
         y.head()
```

```
Out[50]: 0    39343
         1    46205
         2    37731
         3    43525
         4    39891
         Name: Salary, dtype: int64
```

In [51]:
```
# Split the data for train and test
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,train_size=.7,random_state=42)
X_train = X_train[:,np.newaxis]
X_test = X_test[:,np.newaxis]
```

```
C:\Users\Lab2_45\AppData\Local\Temp\ipykernel_8488\1607020857.py:4: FutureWarning: Support for multi-dimensional indexing (e.g.
`obj[:, None]`) is deprecated and will be removed in a future version.  Convert to a numpy array before indexing instead.
  X_train = X_train[:,np.newaxis]
C:\Users\Lab2_45\AppData\Local\Temp\ipykernel_8488\1607020857.py:5: FutureWarning: Support for multi-dimensional indexing (e.g.
`obj[:, None]`) is deprecated and will be removed in a future version.  Convert to a numpy array before indexing instead.
  X_test = X_test[:,np.newaxis]
```

In [52]:
```
X_test
```

Out[52]:
```
array([[ 9.6],
       [ 4.9],
       [ 8.2],
       [ 5.3],
       [ 3.2],
       [ 3.7],
       [10.3],
       [ 8.7],
       [ 4. ]])
```

In [53]:
```
X_train
```

Out[53]:
```
array([[ 1.1],
       [ 2.2],
       [ 5.1],
       [ 2.9],
       [ 4.1],
       [ 4. ],
       [ 7.9],
       [ 1.3],
       [ 1.5],
       [ 9. ],
       [ 2. ],
       [ 7.1],
       [ 9.5],
       [ 5.9],
       [10.5],
       [ 6.8],
       [ 3.2],
       [ 3.9],
       [ 4.5],
       [ 6. ],
       [ 3. ]])
```

In [54]:
```
#Importing Linear Regression model from scikit learn
from sklearn.linear_model import LinearRegression
```

In [55]:
```
# Fitting the model
lr = LinearRegression()
lr.fit(X_train,y_train)

# Predicting the Salary for the Test values
y_pred = lr.predict(X_test)
```

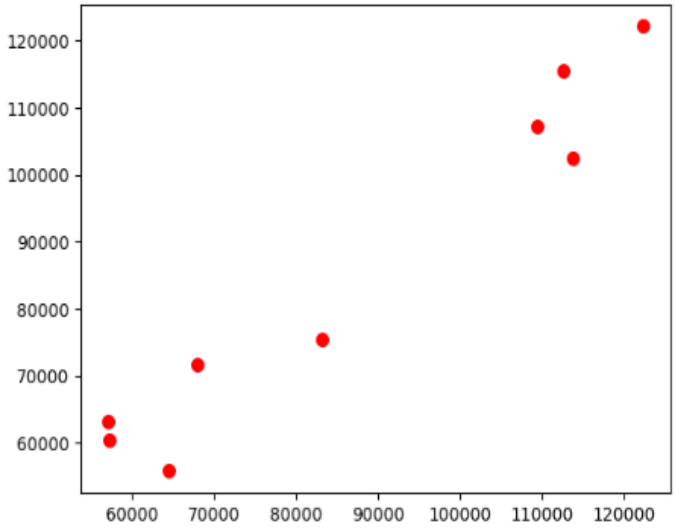
In [56]:
```
#Importing metrics for the evaluation of the model
from sklearn.metrics import r2_score,mean_squared_error

#calculate Mean square error
mse = mean_squared_error(y_test,y_pred)

#Calculate R square vale
rsq = r2_score(y_test,y_pred)
print('mean sqaured error :', mse)
print('r sqaure :', rsq)
```

```
mean sqaured error : 37784662.46621314
r sqaure : 0.9414466227178214
```

In [57]:
```
#Just plot actual and predicted values for more insights plt.figure(figsize=(12,6))
plt.scatter(y_test,y_pred,color='r',linewidth=2)
plt.show()
```



In [47]:
```
# Intecept and coeff of the line
print('Intercept of the model:',lr.intercept_)
print('Coefficient of the line:',lr.coef_)
```

```
Intercept of the model: 25918.438334893202
Coefficient of the line: [9339.08172382]
```

Practical No:8

Aim: Program to Implement Logistic Regression

Output:

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('E:\AI MI Lab (Sandesh)\CreditRisk.csv')
```

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

Out[3]:
```
(614, 13)
```

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

Out[4]:

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

In [5]:
```
credit_df.head(8)
```

Out[5]:

	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
5	LP001011	Male	Yes	2	Graduate	Yes	5417	4196.0	267	360.0	1.0
6	LP001013	Male	Yes	0	Not Graduate	No	2333	1516.0	95	360.0	1.0
7	LP001014	Male	Yes	3+	Graduate	No	3036	2504.0	158	360.0	0.0

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

Out[6]:

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

In [7]:
```
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 [8]:
```
credit_df.describe()
```

Out[8]:

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

In [9]:
```
credit_df.Loan_Status.value_counts()
```

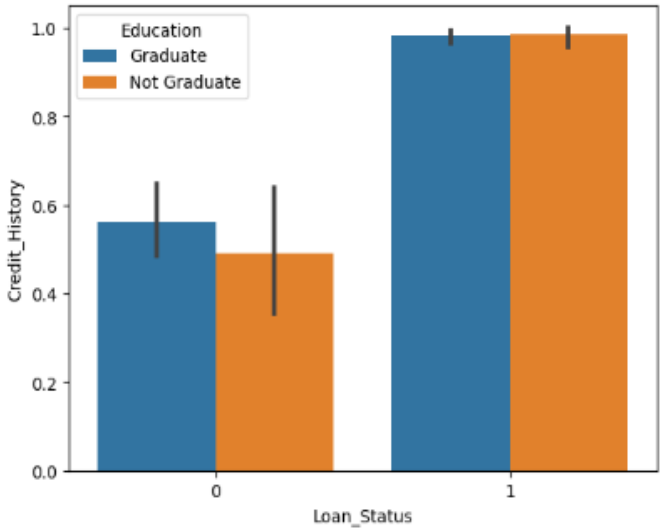
Out[9]:
```
Loan_Status
1    422
0    192
Name: count, dtype: int64
```

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

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

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

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



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

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

In [13]:
```
DF = credit_df.drop(credit_df.columns[0],axis=1)
```

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

Out[14]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_A
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 [15]:
```
object_columns = DF.select_dtypes(include=['object']).columns
numeric_columns = DF.select_dtypes(exclude=['object']).columns
```

In [16]:
```
for column in object_columns:
    majority = DF[column].value_counts().iloc[0]
    DF[column].fillna(majority, inplace=True)
```

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

In [18]:
```
#Impute
```

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

Out[19]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_A
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 [20]:
```
DF
```

Out[20]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_
0	Male	No	0	Graduate	No	5849	0.0	0	360.0	1.0	U
1	Male	Yes	1	Graduate	No	4583	1508.0	128	360.0	1.0	
2	Male	Yes	0	Graduate	Yes	3000	0.0	66	360.0	1.0	U
3	Male	Yes	0	Not Graduate	No	2583	2358.0	120	360.0	1.0	U
4	Male	No	0	Graduate	No	6000	0.0	141	360.0	1.0	U
...	...	...	...	...	...	...	...	...	...	...	
609	Female	No	0	Graduate	No	2900	0.0	71	360.0	1.0	
610	Male	Yes	3+	Graduate	No	4106	0.0	40	180.0	1.0	
611	Male	Yes	1	Graduate	No	8072	240.0	253	360.0	1.0	U
612	Male	Yes	2	Graduate	No	7583	0.0	187	360.0	1.0	U
613	Female	No	0	Graduate	Yes	4583	0.0	133	360.0	0.0	Semi

614 rows × 12 columns

In [21]:
```
DF[object_columns].Property_Area
```

Out[21]:
```
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 [22]:
```
DF[object_columns].Property_Area.head()
```

Out[22]:
```
0    Urban
1    Rural
2    Urban
3    Urban
4    Urban
Name: Property_Area, dtype: object
```

In [23]:
```
DF_dummy = pd.get_dummies(DF, columns=object_columns)
```

In [24]:
```
DF_dummy.head()
```

Out[24]:

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

5 rows × 25 columns

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

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

In [26]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

In [27]:
```
X = DF_dummy.drop('Loan_Status', axis=1)
y = DF_dummy.Loan_Status
train_x, test_x, train_y, test_y = train_test_split(X,y,test_size=0.3, random_state=42)
```

In [28]:
```
train_x.shape, test_x.shape
```

Out[28]:
```
((429, 24), (185, 24))
```

In [29]:
```
model = LogisticRegression()
```

In [30]:
```
model.fit(train_x, train_y)
```

```
C:\Users\sande\anaconda3\Lib\site-packages\sklearn\linear_model\_logistic.py:458: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Out[30]:
```
▾ LogisticRegression
LogisticRegression()
```

In [31]:
```
train_y_hat = model.predict(train_x)
test_y_hat = model.predict(test_x)
```

In [32]:
```
print('train accurracy', accuracy_score(train_y, train_y_hat))
print('test accurracy', accuracy_score(test_y, test_y_hat))
```

```
train accurracy 0.8205128205128205
test accurracy 0.7837837837837838
```

In [33]:
```
print(confusion_matrix(train_y, train_y_hat))
```

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

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

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

In [35]:
```
test_y.value_counts()
```

Out[35]:
```
Loan_Status
1    120
0     65
Name: count, dtype: int64
```

In [36]:
```
pd.Series(test_y_hat).value_counts()
```

Out[36]:
```
1    156
0     29
Name: count, dtype: int64
```

In [37]:
```
print(classification_report(test_y, test_y_hat))
```

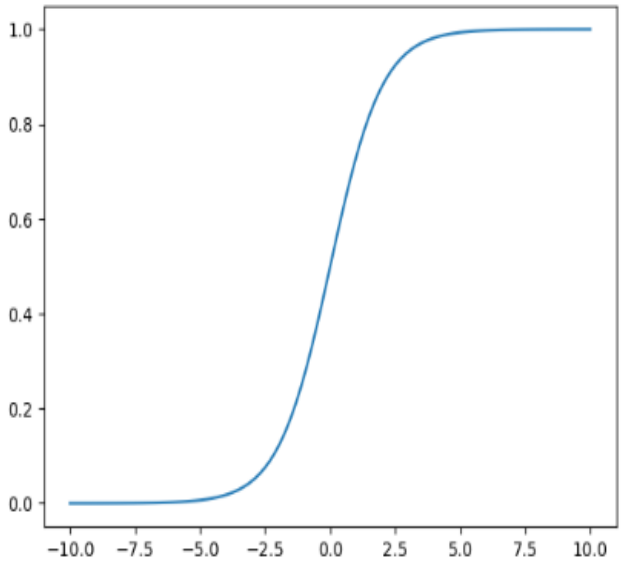
```
              precision    recall  f1-score   support

           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 [38]:
```
x = np.linspace(-10,10,100)
y= 1 / (1 + np.exp(-x))  #Sigmoid
plt.plot(x, y)
```

Out[38]:
```
[<matplotlib.lines.Line2D at 0x24968702510>]
```



Practical No: 9

Aim: Program to Implement AdaBoost

Output:

In [1]:
```
from sklearn.ensemble import AdaBoostClassifier
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn import metrics
```

In [2]:
```
iris = datasets.load_iris()
X = iris.data
y = iris.target
```

In [3]:
```
# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
# 80% training and 20% test
```

In [4]:
```
# Create adaboost classifer object
AdaModel = AdaBoostClassifier(n_estimators=100, learning_rate=1)

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

In [5]:
```
#Predict the response for test dataset
y_pred = model.predict(X_test)
```

In [6]:
```
# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.9333333333333333
```

In [7]:
```
# Import Support Vector Classifier
from sklearn.svm import SVC

#Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics
svc=SVC(probability=True, kernel='linear')
```

In [8]:
```
# Create adaboost classifer object
abc =AdaBoostClassifier(n_estimators=50, base_estimator=svc,learning_rate=1)
```

In [9]:
```
# Train Adaboost Classifer
model = abc.fit(X_train, y_train)
```

```
C:\Users\sande\anaconda3\Lib\site-packages\sklearn\ensemble\_base.py:166: FutureWarning: `base_estimator` was renamed to `estim
ator` in version 1.2 and will be removed in 1.4.
  warnings.warn(
```

In [10]:
```
#Predict the response for test dataset
y_pred = model.predict(X_test)
```

In [12]:
```
# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.9333333333333333
```

Practical No:10

Aim: Aim: Program to Implement Random Forest Regression

Output:

```
In [27]: import numpy as np
         import pandas as pd
         import matplotlib as mpl
         import matplotlib.pyplot as plt
```

```
In [28]: train = pd.read_csv("D:\AI ML\Titanic-Dataset.csv")
```

```
In [29]: print(train.shape)

         (891, 12)
```

```
In [30]: #checking for missing data
         NAs = pd.concat([train.isnull().sum()], axis=1, keys=["Train"])
         NAs[NAs.sum(axis=1) > 0]
```

Out[30]:

	Train
Age	177
Cabin	687
Embarked	2

```
In [31]: train. pop("Cabin")
         train. pop ("Name")
         train.pop("Ticket")
```

```
Out[31]: 0            A/5 21171
         1             PC 17599
         2     STON/O2. 3101282
         3               113803
         4               373450
                   ...
         886             211536
         887             112053
         888         W./C. 6607
         889             111369
         890             370376
         Name: Ticket, Length: 891, dtype: object
```

```
In [32]: # Filling missing Age values with mean
         train["Age"] = train["Age"].fillna(train["Age"].mean())
```

```
In [33]: # Filling missing Embarked values with most common value
         train["Embarked"] = train["Embarked"].fillna(train["Embarked"].mode()[0])
```

```
In [34]: train["Pclass"] = train["Pclass"].apply(str)
```

In [35]:
```
# Getting Dummies from all other categorical vars
for col in train.dtypes[train.dtypes == "object"].index:
    for_dummy = train.pop(col)
    train = pd.concat([train, pd.get_dummies(for_dummy, prefix=col)], axis=1)
train. head()
```

Out[35]:

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

In [36]:
```
labels = train.pop("Survived")
```

In [37]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(train, labels, test_size=0.25)
```

In [38]:
```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators=100)
rf.fit(x_train, y_train)
```

Out[38]:
```
▾ RandomForestClassifier
RandomForestClassifier()
```

In [39]:
```
y_pred = rf.predict(x_test)
```

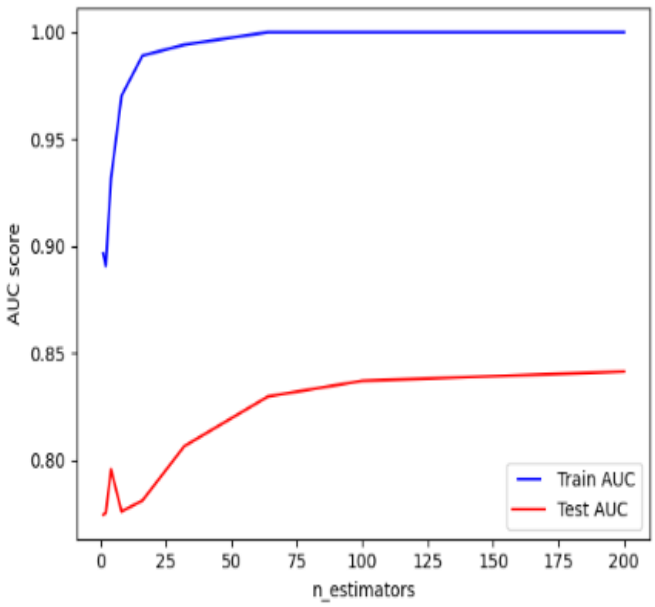
In [40]:
```
from sklearn.metrics import roc_curve, auc
false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, y_pred)
roc_auc = auc(false_positive_rate, true_positive_rate)
roc_auc
```

Out[40]: 0.8238414530639959

In [41]:
```
n_estimators = [1, 2, 4, 8, 16, 32, 64, 100, 200]
train_results = []
test_results = []
```

In [42]:
```
for estimator in n_estimators:
    rf = RandomForestClassifier(n_estimators=estimator, n_jobs =- 1)
    rf.fit(x_train, y_train)
    train_pred = rf.predict(x_train)
    false_positive_rate, true_positive_rate, thresholds = roc_curve(y_train,train_pred)
    roc_auc = auc(false_positive_rate, true_positive_rate)
    train_results.append(roc_auc)
    y_pred = rf.predict(x_test)
    false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, y_pred)
    roc_auc = auc(false_positive_rate, true_positive_rate)
    test_results.append(roc_auc)
```

In [43]:
```
from matplotlib. legend_handler import HandlerLine2D
line1, = plt.plot(n_estimators, train_results, "b", label="Train AUC")
line2, = plt.plot(n_estimators, test_results, "r", label="Test AUC")
plt. legend(handler_map={line1: HandlerLine2D(numpoints=2)})
plt.ylabel("AUC score")
plt.xlabel("n_estimators")
plt. show()
```



In [44]:
```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators=200)
rf.fit(x_train, y_train)
```

Out[44]:
```
RandomForestClassifier
RandomForestClassifier(n_estimators=200)
```

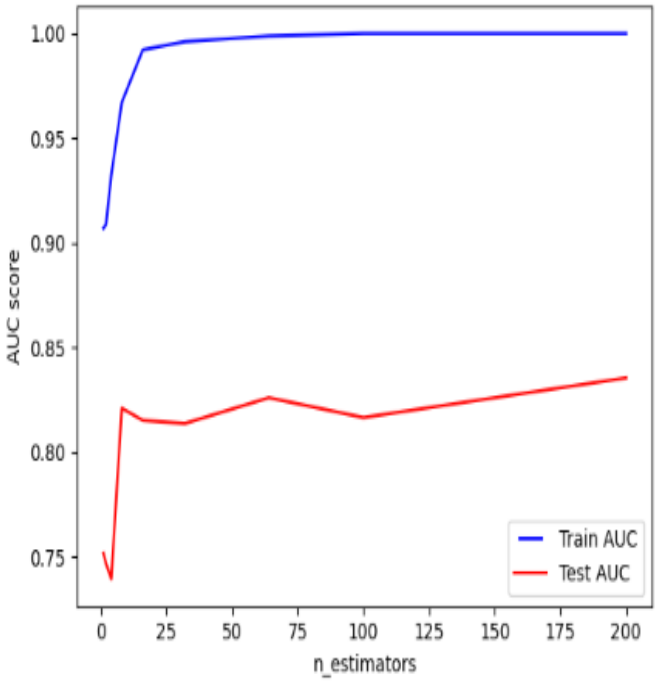
In [45]:
```
y_pred = rf.predict(x_test)
from sklearn.metrics import roc_curve, auc
false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, y_pred)
roc_auc = auc(false_positive_rate, true_positive_rate)
roc_auc
```

Out[45]: 0.8201918180274997

In [46]:
```
n_estimators = [1, 2, 4, 8, 16, 32, 64, 100, 200]
train_results = []
test_results = []
```

In [47]:
```
for estimator in n_estimators:
    rf = RandomForestClassifier(n_estimators=estimator, n_jobs =- 1)
    rf.fit(x_train, y_train)
    train_pred = rf.predict(x_train)
    false_positive_rate, true_positive_rate, thresholds = roc_curve(y_train,train_pred)
    roc_auc = auc(false_positive_rate, true_positive_rate)
    train_results.append(roc_auc)
    y_pred = rf.predict(x_test)
    false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, y_pred)
    roc_auc = auc(false_positive_rate, true_positive_rate)
    test_results.append(roc_auc)
```

In [48]:
```
from matplotlib. legend_handler import HandlerLine2D
line1, = plt.plot(n_estimators, train_results, "b", label="Train AUC")
line2, = plt.plot(n_estimators, test_results, "r", label="Test AUC")
plt. legend(handler_map={line1: HandlerLine2D(numpoints=2)})
plt.ylabel("AUC score")
plt.xlabel("n_estimators")
plt.show()
```


1.00
0.95
0.90
0.85
0.80
0.75
AUC score
0
25
50
75
100
125
150
175
200
n_estimators
Train AUC
Test AUC

Practical No: 11

Aim: Program to Implement K-Means Clustering Algorithm

Output:

In [2]:
```
import pandas as pd
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
```

In [3]:
```
df = pd.read_csv('E:\AI MI Lab (Sandesh)\driver-data.csv')
df.head()
```

Out[3]:

	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 [4]:
```
df.describe()
```

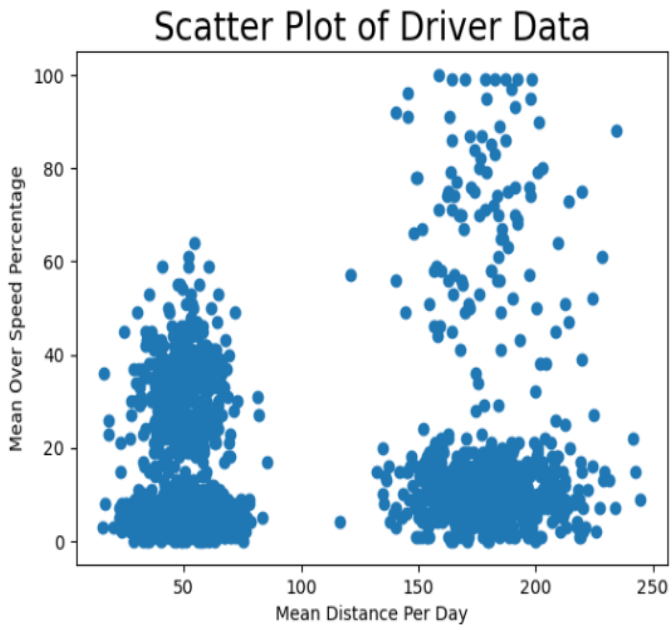
Out[4]:

	id	mean_dist_day	mean_over_speed_perc
count	4.000000e+03	4000.000000	4000.000000
mean	3.423312e+09	76.041522	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 [5]:
```
df.shape
```

Out[5]: (4000, 3)

In [6]:
```
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()
```



In [7]:
```
df.head()
```

Out[7]:

	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 [8]:
```
data = df.drop(['id'], axis=1)
cluster_model = KMeans(n_clusters=2, n_init='auto')
cluster_model.fit(data)
```

Out[8]:
```
▾ KMeans
KMeans(n_clusters=2, n_init='auto')
```

In [9]:
```
df['labels'] = cluster_model.labels_
```

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

Out[10]:

	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

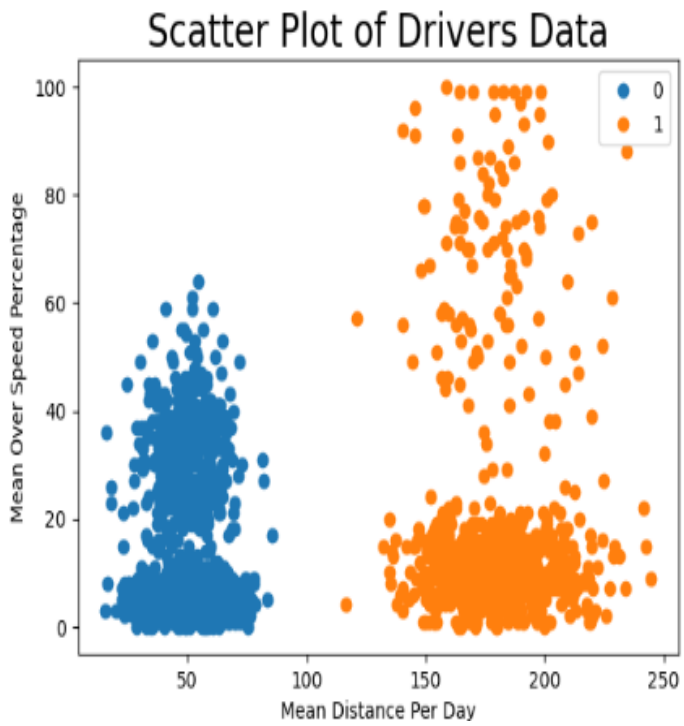
In [11]:
```
df.labels.unique()
```

Out[11]:
```
array([0, 1])
```

In [12]:
```
df['labels'].value_counts()
```

Out[12]:
```
labels
0    3200
1     800
Name: count, dtype: int64
```

In [13]:
```
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 Drivers Data', fontsize=20)
plt.legend()
plt.show()
```

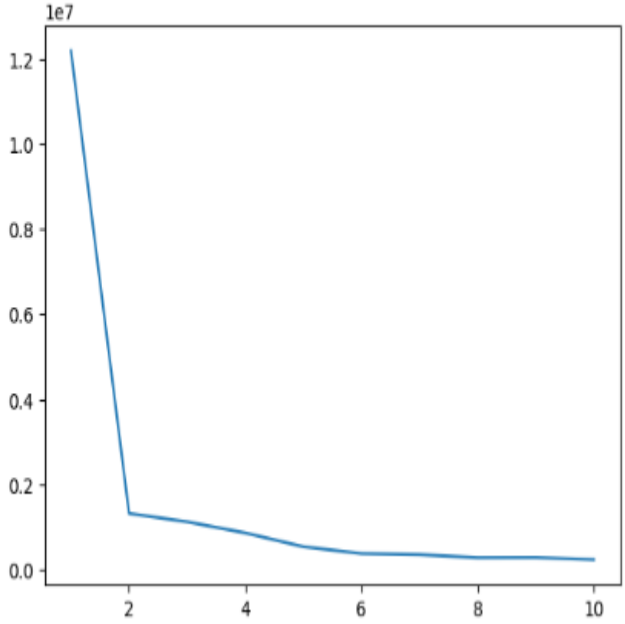


In [14]:
```
cluster_model.cluster_centers_
```

Out[14]:
```
array([[ 50.04763438,   8.82875   ],
       [180.017075  ,  18.29      ]])
```

In [15]:
```
error = []
for k in range(1,11):
    cluster_model = KMeans(k, n_init='auto')
    cluster_model.fit(data)
    error.append(cluster_model.inertia_)
```

In [16]:
```
plt.plot(range(1,11), error)
plt.show()
```



In [17]:
```
df[df.labels == 0]
```

Out[17]:

	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
...	...	...	...	...
3195	3423312761	47.56	5	0
3196	3423314346	39.39	8	0
3197	3423313451	46.85	7	0
3198	3423313552	37.68	7	0
3199	3423312998	50.56	5	0

3200 rows × 4 columns