Simple Linear Regression

```
import pandas as pd
import numpy as np
df  = pd.read_csv('salary_data.csv')
```

```
df
```

```
    YearsExperience    Salary
0               1.1   39343.0
1               1.3   46205.0
2               1.5   37731.0
3               2.0   43525.0
4               2.2   39891.0
5               2.9   56642.0
6               3.0   60150.0
7               3.2   54445.0
8               3.2   64445.0
9               3.7   57189.0
10              3.9   63218.0
11              4.0   55794.0
12              4.0   56957.0
13              4.1   57081.0
14              4.5   61111.0
15              4.9   67938.0
16              5.1   66029.0
17              5.3   83088.0
18              5.9   81363.0
19              6.0   93940.0
20              6.8   91738.0
21              7.1   98273.0
22              7.9  101302.0
23              8.2  113812.0
24              8.7  109431.0
25              9.0  105582.0
26              9.5  116969.0
27              9.6  112635.0
28             10.3  122391.0
29             10.5  121872.0
```

```
df.head()
```

```
   YearsExperience   Salary
0              1.1  39343.0
1              1.3  46205.0
2              1.5  37731.0
3              2.0  43525.0
4              2.2  39891.0
```

```
df.shape
```

```
(30, 2)
```

```python
df[['YearsExperience']] #independetn variable
```

```
    YearsExperience
0               1.1
1               1.3
2               1.5
3               2.0
4               2.2
5               2.9
6               3.0
7               3.2
8               3.2
9               3.7
10              3.9
11              4.0
12              4.0
13              4.1
14              4.5
15              4.9
16              5.1
17              5.3
18              5.9
19              6.0
20              6.8
21              7.1
22              7.9
23              8.2
24              8.7
25              9.0
26              9.5
27              9.6
28             10.3
29             10.5
```

```python
df[['Salary']].head()
```

```
    Salary
0  39343.0
1  46205.0
2  37731.0
3  43525.0
4  39891.0
```

```python
df.describe()
```

```
       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
```

```
df.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     float64
dtypes: float64(2)
memory usage: 608.0 bytes
```

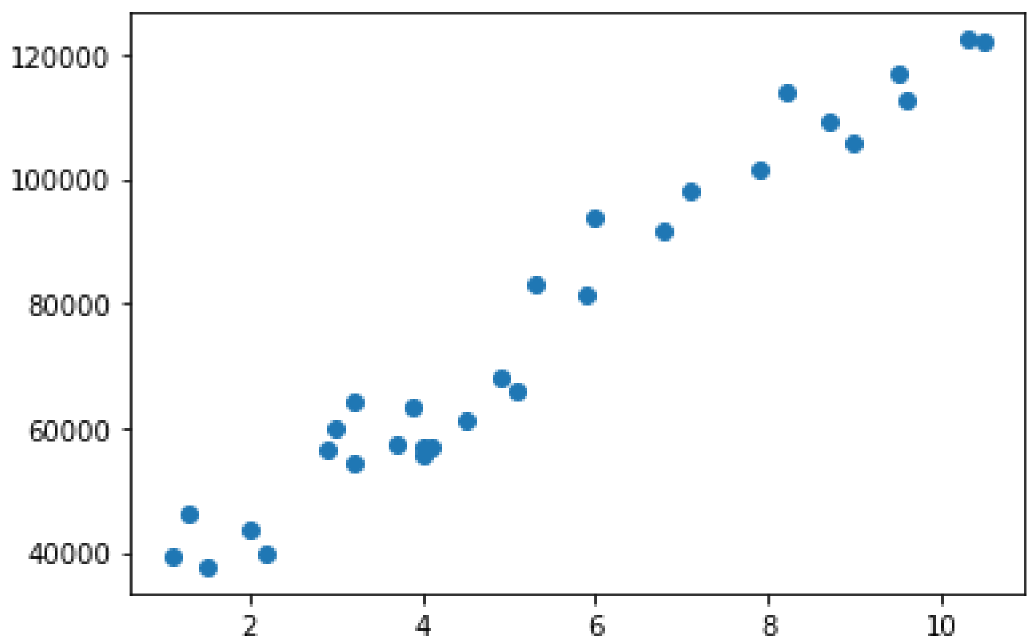
```
df.isnull().sum()
```

```
YearsExperience    0
Salary             0
dtype: int64
```

```
import matplotlib.pyplot as plt #data visualization
```

```
plt.scatter(df.YearsExperience,df.Salary) # scatter plot
```

```
<matplotlib.collections.PathCollection at 0x1ec388414c0>
```



Independent and Dependent variables

```
x = df.iloc[:,0:1]
x
```

```
    YearsExperience
0               1.1
1               1.3
2               1.5
3               2.0
4               2.2
5               2.9
6               3.0
7               3.2
8               3.2
9               3.7
10              3.9
11              4.0
12              4.0
13              4.1
14              4.5
15              4.9
16              5.1
17              5.3
18              5.9
19              6.0
20              6.8
21              7.1
22              7.9
23              8.2
24              8.7
25              9.0
26              9.5
27              9.6
28             10.3
29             10.5
```

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

```
     Salary
0   39343.0
1   46205.0
2   37731.0
3   43525.0
4   39891.0
5   56642.0
6   60150.0
7   54445.0
8   64445.0
9   57189.0
```

```
10   63218.0
11   55794.0
12   56957.0
13   57081.0
14   61111.0
15   67938.0
16   66029.0
17   83088.0
18   81363.0
19   93940.0
20   91738.0
21   98273.0
22  101302.0
23  113812.0
24  109431.0
25  105582.0
26  116969.0
27  112635.0
28  122391.0
29  121872.0
```

```
df.shape
```

```
(30, 2)
```

Train,Test & Split

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test =
train_test_split(x,y,test_size=0.2,random_state=0)
```

```
x_train.shape
```

```
(24, 1)
```

```
x_test.shape
```

```
(6, 1)
```

```
y_train.shape
```

```
(24, 1)
```

```
y_test.shape
```

```
(6, 1)
```

Model Building

```
from sklearn.linear_model import LinearRegression
sl = LinearRegression()
```

```python
sl.fit(x_train,y_train) # simple linearregression is created with training data
```

```
LinearRegression()
```

```python
x_test
```

```
    YearsExperience
2               1.5
28             10.3
13              4.1
10              3.9
26              9.5
24              8.7
```

```python
y_test
```

```
      Salary
2    37731.0
28  122391.0
13   57081.0
10   63218.0
26  116969.0
24  109431.0
```

```python
sl.predict(x_test)
```

```
array([[ 40748.96184072],
       [122699.62295594],
       [ 64961.65717022],
       [ 63099.14214487],
       [115249.56285456],
       [107799.50275317]])
```

```python
from sklearn.metrics import r2_score
r2_score(sl.predict(x_test),y_test)
```

```
0.986482673117654
```

```python
sl.predict([[9.3]]) # testing with unseen data
```

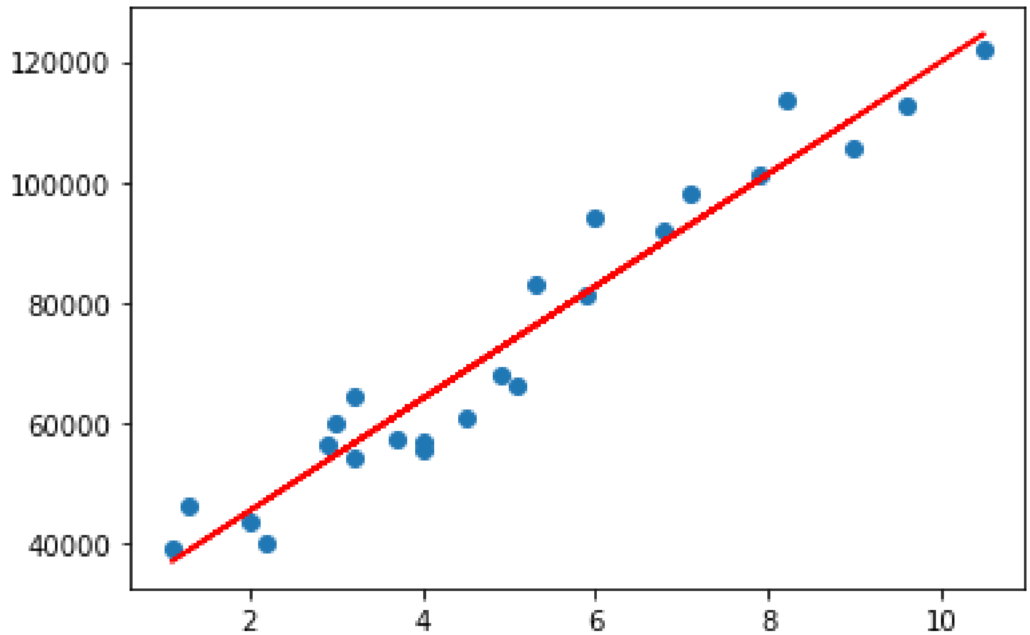
```
array([[113387.04782921]])
```

```python
sl.predict([[12.3]]) # 12.3 is experience, so we are getting salary
```

```
array([[141324.7732094]])
```

```python
plt.scatter(x_train,y_train)
plt.plot(x_train,sl.predict(x_train),'r')
```

```
[<matplotlib.lines.Line2D at 0x1ec38d1a3d0>]
```



3 Multiple Linear Regression

```
import numpy as np
import pandas as pd

df = pd.read_csv('50_Startups (4).csv')

df
```

```
     R&D Spend  Administration  Marketing Spend       State     Profit
0    165349.20       136897.80        471784.10    New York  192261.83
1    162597.70       151377.59        443898.53  California  191792.06
2    153441.51       101145.55        407934.54     Florida  191050.39
3    144372.41       118671.85        383199.62    New York  182901.99
4    142107.34        91391.77        366168.42     Florida  166187.94
..         ...             ...              ...         ...        ...
103  119943.24       156547.42        256512.92     Florida  132602.65
104  114523.61       122616.84        261776.23    New York  129917.04
105   78013.11       121597.55        264346.06  California  126992.93
106   94657.16       145077.58        282574.31    New York  125370.37
107   91749.16       114175.79        294919.57     Florida  124266.90

[108 rows x 5 columns]
```

```
df.head()
```

```
   R&D Spend  Administration  Marketing Spend       State     Profit
0  165349.20       136897.80        471784.10    New York  192261.83
1  162597.70       151377.59        443898.53  California  191792.06
2  153441.51       101145.55        407934.54     Florida  191050.39
```

```
3  144372.41       118671.85        383199.62    New York  182901.99
4  142107.34        91391.77        366168.42     Florida  166187.94
```

```
df.shape
```

```
(108, 5)
```

```
df.tail()
```

```
     R&D Spend  Administration  Marketing Spend       State     Profit
103  119943.24       156547.42        256512.92     Florida  132602.65
104  114523.61       122616.84        261776.23    New York  129917.04
105   78013.11       121597.55        264346.06  California  126992.93
106   94657.16       145077.58        282574.31    New York  125370.37
107   91749.16       114175.79        294919.57     Florida  124266.90
```

```
df.isnull().sum()
```

```
R&D Spend          0
Administration     0
Marketing Spend    0
State              0
Profit             0
dtype: int64
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 108 entries, 0 to 107
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   R&D Spend        108 non-null    float64
 1   Administration   108 non-null    float64
 2   Marketing Spend  108 non-null    float64
 3   State            108 non-null    object
 4   Profit           108 non-null    float64
dtypes: float64(4), object(1)
memory usage: 4.3+ KB
```

```
df.describe() # to get statical information
```

```
           R&D Spend  Administration  Marketing Spend         Profit
count     108.000000      108.000000       108.000000     108.000000
mean    74959.338704   121750.788889    214952.664722  113523.760000
std     44996.368152    27322.385654    117937.942120   38991.013654
min         0.000000    51283.140000         0.000000   14681.400000
25%     38558.510000   105077.645000    134050.070000   90708.190000
50%     75791.365000   122699.795000    239452.750000  109543.120000
75%    101913.080000   145077.580000    298664.470000  141585.520000
max    165349.200000   182645.560000    471784.100000  192261.830000
```

```
df.State.value_counts()
```

```
New York      39
California    36
Florida       33
Name: State, dtype: int64
```

```
df.columns
```

```
Index(['R&D Spend', 'Administration', 'Marketing Spend', 'State',
'Profit'], dtype='object')
```

Label Encoder

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
```

```
df.State = le.fit_transform(df.State)
df
```

```
     R&D Spend  Administration  Marketing Spend  State     Profit
0    165349.20       136897.80        471784.10      2  192261.83
1    162597.70       151377.59        443898.53      0  191792.06
2    153441.51       101145.55        407934.54      1  191050.39
3    144372.41       118671.85        383199.62      2  182901.99
4    142107.34        91391.77        366168.42      1  166187.94
..         ...             ...              ...    ...        ...
103  119943.24       156547.42        256512.92      1  132602.65
104  114523.61       122616.84        261776.23      2  129917.04
105   78013.11       121597.55        264346.06      0  126992.93
106   94657.16       145077.58        282574.31      2  125370.37
107   91749.16       114175.79        294919.57      1  124266.90

[108 rows x 5 columns]
```

```
df.State.value_counts() # encoded
```

```
2    39
0    36
1    33
Name: State, dtype: int64
```

```
df.head()
```

```
   R&D Spend  Administration  Marketing Spend  State     Profit
0  165349.20       136897.80        471784.10      2  192261.83
1  162597.70       151377.59        443898.53      0  191792.06
2  153441.51       101145.55        407934.54      1  191050.39
3  144372.41       118671.85        383199.62      2  182901.99
4  142107.34        91391.77        366168.42      1  166187.94
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 108 entries, 0 to 107
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   R&D Spend        108 non-null    float64
 1   Administration   108 non-null    float64
 2   Marketing Spend  108 non-null    float64
 3   State            108 non-null    int32  
 4   Profit           108 non-null    float64
dtypes: float64(4), int32(1)
memory usage: 3.9 KB
```

Independent and Dependent variables

```
x = df.iloc[:,:4]
x
```

```
     R&D Spend  Administration  Marketing Spend  State
0    165349.20       136897.80        471784.10      2
1    162597.70       151377.59        443898.53      0
2    153441.51       101145.55        407934.54      1
3    144372.41       118671.85        383199.62      2
4    142107.34        91391.77        366168.42      1
..         ...             ...              ...    ...
103  119943.24       156547.42        256512.92      1
104  114523.61       122616.84        261776.23      2
105   78013.11       121597.55        264346.06      0
106   94657.16       145077.58        282574.31      2
107   91749.16       114175.79        294919.57      1

[108 rows x 4 columns]
```

```
y = df.iloc[:,4:]

y
```

```
        Profit
0    192261.83
1    191792.06
2    191050.39
3    182901.99
4    166187.94
..         ...
103  132602.65
104  129917.04
105  126992.93
106  125370.37
107  124266.90
```

```
[108 rows x 1 columns]
```

Train,Test & Split

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size =
0.2,random_state=0)
```

```
df.shape
```

```
(108, 5)
```

```
x_train.shape#remaining all the variables
```

```
(86, 4)
```

```
y_train.shape #profit
```

```
(86, 1)
```

```
x_test.shape
```

```
(22, 4)
```

```
from sklearn.linear_model import LinearRegression
slr = LinearRegression()
slr.fit(x_train,y_train)
```

```
LinearRegression()
```

```
x_test.head()
```

```
    R&D Spend  Administration  Marketing Spend  State
84    1000.23       124153.04          1903.93      2
10  101913.08       110594.11        229160.95      1
75   28663.76       127056.21        201126.82      1
2   153441.51       101145.55        407934.54      1
24   77044.01        99281.34        140574.81      2
```

```
y_test.head()
```

```
       Profit
84   64926.08
10  146121.95
75   90708.19
2   191050.39
24  108552.04
```

```
x_test[0:5]
```

```
    R&D Spend  Administration  Marketing Spend  State
84    1000.23       124153.04          1903.93      2
```

```
10  101913.08       110594.11        229160.95      1
75   28663.76       127056.21        201126.82      1
2   153441.51       101145.55        407934.54      1
24   77044.01        99281.34        140574.81      2
```

```
mlr_pred = slr.predict(x_test)
mlr_pred[0:5]
```

```
array([[ 48379.24868384],
       [134848.91924675],
       [ 76483.10965219],
       [181561.78529195],
       [112966.00035119]])
```

```
slr.predict(x_test[0:5])
```

```
array([[ 48379.24868384],
       [134848.91924675],
       [ 76483.10965219],
       [181561.78529195],
       [112966.00035119]])
```

```
import numpy as np
from sklearn.metrics import r2_score
r2_score(slr.predict(x_test),y_test)
```

```
0.9314720388994493
```

```
slr.predict([[123120.54,12321.32,567800.34,0]])
```

```
array([[163972.11719139]])
```

Polynomial Regression

```
df = pd.read_csv('salaries_data.csv')
```

```
df
```

```
            Position  Level   Salary
0   Business Analyst      1    45000
1  Junior Consultant      2    50000
2  Senior Consultant      3    60000
3            Manager      4    80000
4    Country Manager      5   110000
5     Region Manager      6   150000
6            Partner      7   200000
7     Senior Partner      8   300000
8            C-level      9   500000
9                CEO     10  1000000
```

```
df.head()
```

```
            Position  Level   Salary
0   Business Analyst      1    45000
1  Junior Consultant      2    50000
2  Senior Consultant      3    60000
3            Manager      4    80000
4    Country Manager      5   110000
```

```
df.corr()
```

```
           Level    Salary
Level   1.000000  0.817949
Salary  0.817949  1.000000
```

Independent and Dependent variables

```
x = df.iloc[:,1:2]
```

```
x
```

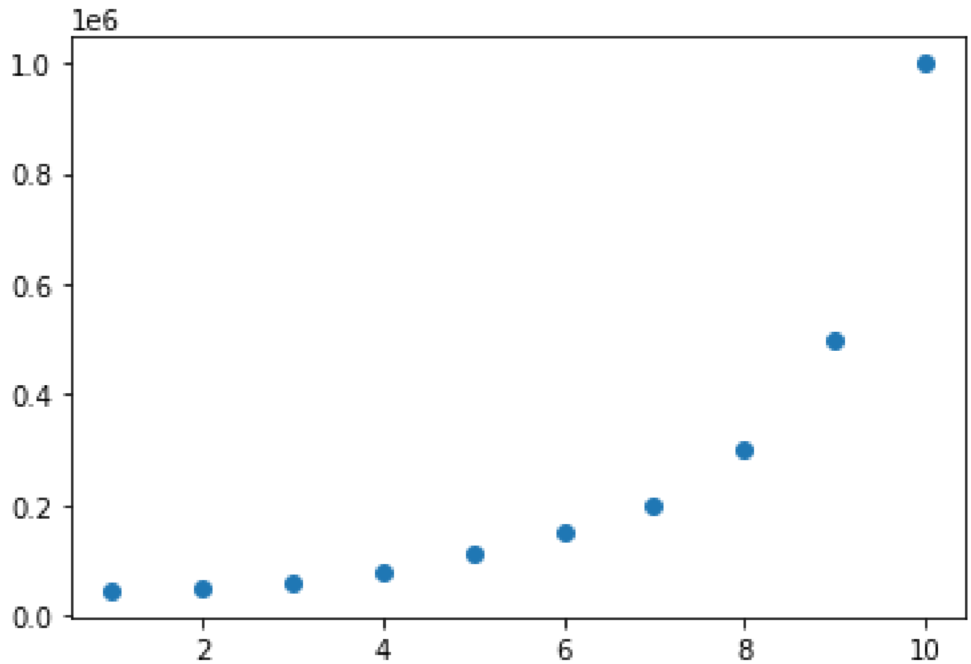
```
   Level
0      1
1      2
2      3
3      4
4      5
5      6
6      7
7      8
8      9
9     10
```

```
y = df.iloc[:,2:]
y
```

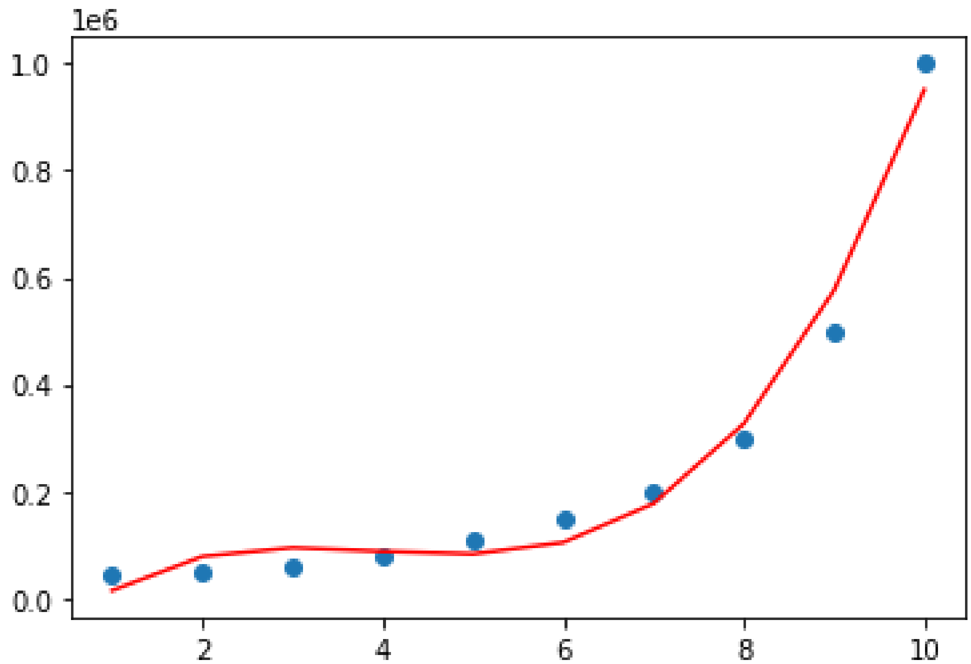
```
    Salary
0    45000
1    50000
2    60000
3    80000
4   110000
5   150000
6   200000
7   300000
8   500000
9  1000000
```

```
import matplotlib.pyplot as plt
plt.scatter(x,y)
```

```
<matplotlib.collections.PathCollection at 0x1ec38da1e80>
```




```
from sklearn.preprocessing import PolynomialFeatures
pr = PolynomialFeatures(degree = 3)

xp = pr.fit_transform(x)

xp

array([[   1.,    1.,    1.,    1.],
       [   1.,    2.,    4.,    8.],
       [   1.,    3.,    9.,   27.],
       [   1.,    4.,   16.,   64.],
       [   1.,    5.,   25.,  125.],
       [   1.,    6.,   36.,  216.],
       [   1.,    7.,   49.,  343.],
       [   1.,    8.,   64.,  512.],
       [   1.,    9.,   81.,  729.],
       [   1.,   10.,  100., 1000.]])

lr = LinearRegression()
lr.fit(xp,y)

LinearRegression()

plt.scatter(x,y)
plt.plot(x,lr.predict(xp),'r')

[<matplotlib.lines.Line2D at 0x1ec38e02fa0>]
```



Decision Tree Regression

```
# It build the regression in tree structure
# which has decision and leaf nodes
# it split the big thing into smaller ones
# applicable for both classification and regression problems
# it had root node and internal decision nodes
# and it has subtress and children nodes and we can also perform
cloning(removing of the unwanted nodes from the tree)
# we can best attribute

df = pd.read_csv('50_Startups (4).csv')

df.head()
```

	R&D Spend	Administration	Marketing Spend	State	Profit
0	165349.20	136897.80	471784.10	New York	192261.83
1	162597.70	151377.59	443898.53	California	191792.06
2	153441.51	101145.55	407934.54	Florida	191050.39
3	144372.41	118671.85	383199.62	New York	182901.99
4	142107.34	91391.77	366168.42	Florida	166187.94

Model Building

```
from sklearn.tree import DecisionTreeRegressor
dt = DecisionTreeRegressor()
dt.fit(x_train,y_train)

DecisionTreeRegressor()
```

```
dt.predict(x_test)
```

```
array([ 42559.73, 146121.95,  90708.19, 191050.39, 108552.04, 144259.4
,
       124266.9 , 155752.6 , 126992.93,  42559.73, 101004.64,
110352.25,
        42559.73, 111313.02,  89949.14, 134307.35, 134307.35,  96712.8
,
        49490.75, 129917.04, 132602.65, 152211.77])
```

```
y_test
```

```
        Profit
84    64926.08
10   146121.95
75    90708.19
2    191050.39
24   108552.04
100  144259.40
107  124266.90
7    155752.60
16   126992.93
86    42559.73
68   101004.64
22   110352.25
45    64926.08
60   111313.02
76    89949.14
52   134307.35
13   134307.35
73    96712.80
85    49490.75
54   129917.04
103  132602.65
8    152211.77
```

Random forest Regression

```
# if we have more decision tress in the dataset and most of them are
supporting categorical data
# assemble uses bagging and boosting
# bagging output is majority good
# boosting creates the accurate model
# overcome the over fit problem

from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor(n_estimators = 5,random_state = 0)
rf.fit(x_train,y_train)
```

```
C:\Users\bharg\AppData\Local\Temp/ipykernel_1992/921291185.py:3:
DataConversionWarning: A column-vector y was passed when a 1d array
```

```
was expected. Please change the shape of y to (n_samples,), for
example using ravel().
  rf.fit(x_train,y_train)
```

```
RandomForestRegressor(n_estimators=5, random_state=0)
```

```
rfr = rf.predict(x_test) # predicted data
rfr
```

```
array([ 36993.004, 145749.44 ,  84934.656, 187791.03 , 128036.856,
       144259.4  , 124812.106, 155044.434, 126992.93 ,  25832.732,
       101460.188, 110352.25 ,  36993.004, 115556.692,  92681.052,
       132844.466, 132844.466,  96680.32 ,  32803.542, 133513.412,
       136034.112, 131440.084])
```

```
y_test # actual data
```

	Profit
84	64926.08
10	146121.95
75	90708.19
2	191050.39
24	108552.04
100	144259.40
107	124266.90
7	155752.60
16	126992.93
86	42559.73
68	101004.64
22	110352.25
45	64926.08
60	111313.02
76	89949.14
52	134307.35
13	134307.35
73	96712.80
85	49490.75
54	129917.04
103	132602.65
8	152211.77

```
rfscore = r2_score(rfr,y_test)
```

```
rfscore
```

```
0.922343446522065
```

Classification Problems

```
# we have multi class classifier
#1.garbage (glass,metal,fiber,plastic)
```

```
#2.more than two we contain
# binary  classifier
#1.two possible outcomes(no/yes and reject/approve)
```

Decision tree classifier

```
df = pd.read_csv('loan_prediction.csv')

df.head()

    Loan_ID Gender Married Dependents     Education Self_Employed  \
0  LP001002   Male      No          0      Graduate            No
1  LP001003   Male     Yes          1      Graduate            No
2  LP001005   Male     Yes          0      Graduate           Yes
3  LP001006   Male     Yes          0  Not Graduate            No
4  LP001008   Male      No          0      Graduate            No

   ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
0             5849                0.0         NaN             360.0
1             4583             1508.0       128.0             360.0
2             3000                0.0        66.0             360.0
3             2583             2358.0       120.0             360.0
4             6000                0.0       141.0             360.0

   Credit_History Property_Area Loan_Status
0             1.0         Urban           Y
1             1.0         Rural           N
2             1.0         Urban           Y
3             1.0         Urban           Y
4             1.0         Urban           Y

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         592 non-null    float64
 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    object
dtypes: float64(4), int64(1), object(8)
memory usage: 62.5+ KB
```

```
df.isnull().sum()
```

```
Loan_ID               0
Gender               13
Married               3
Dependents           15
Education             0
Self_Employed        32
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount           22
Loan_Amount_Term     14
Credit_History       50
Property_Area         0
Loan_Status           0
dtype: int64
```

```
df['Gender'].isnull().sum()
```

```
13
```

```
df['Gender'].unique()
```

```
array(['Male', 'Female', nan], dtype=object)
```

```
df['Married'].unique()
```

```
array(['No', 'Yes', nan], dtype=object)
```

```
df['Dependents'].unique()
```

```
array(['0', '1', '2', '3+', nan], dtype=object)
```

```
df['Gender'] = df['Gender'].map({'Male':1,'Female':0})
df['Married'] = df['Married'].map({'Yes':1,'No':0})
```

```
#df['Gender'].unique()
df['Married'].unique()
```

```
array([ 0.,  1., nan])
```

Logistic Regression

```
#credit problems all are resolve by using the logistic regression
#ln(p/1-p)
```

```
import pandas as pd
import numpy as np #scientific calculations
df = pd.read_csv('Social_Network_Ads.csv')
```

```
df.head()
```

```
    User ID  Gender  Age  EstimatedSalary  Purchased
0  15624510    Male   19            19000          0
1  15810944    Male   35            20000          0
2  15668575  Female   26            43000          0
3  15603246  Female   27            57000          0
4  15804002    Male   19            76000          0
```

```
df.shape
```

```
(400, 5)
```

```
df.describe()
```

```
            User ID         Age  EstimatedSalary   Purchased
count  4.000000e+02  400.000000       400.000000  400.000000
mean   1.569154e+07   37.655000     69742.500000    0.357500
std    7.165832e+04   10.482877     34096.960282    0.479864
min    1.556669e+07   18.000000     15000.000000    0.000000
25%    1.562676e+07   29.750000     43000.000000    0.000000
50%    1.569434e+07   37.000000     70000.000000    0.000000
75%    1.575036e+07   46.000000     88000.000000    1.000000
max    1.581524e+07   60.000000    150000.000000    1.000000
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   User ID          400 non-null    int64 
 1   Gender           400 non-null    object
 2   Age              400 non-null    int64 
 3   EstimatedSalary  400 non-null    int64 
 4   Purchased        400 non-null    int64 
dtypes: int64(4), object(1)
memory usage: 15.8+ KB
```

```
df.isnull().sum()
```

```
User ID            0
Gender             0
Age                0
EstimatedSalary    0
Purchased          0
dtype: int64
```

```
df['Purchased'].unique()
```

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

Independent and Dependent Variables

```
x =df.iloc[:,[2,3]].values
```

```
x
```

```
array([[    19,  19000],
       [    35,  20000],
       [    26,  43000],
       [    27,  57000],
       [    19,  76000],
       [    27,  58000],
       [    27,  84000],
       [    32, 150000],
       [    25,  33000],
       [    35,  65000],
       [    26,  80000],
       [    26,  52000],
       [    20,  86000],
       [    32,  18000],
       [    18,  82000],
       [    29,  80000],
       [    47,  25000],
       [    45,  26000],
       [    46,  28000],
       [    48,  29000],
       [    45,  22000],
       [    47,  49000],
       [    48,  41000],
       [    45,  22000],
       [    46,  23000],
       [    47,  20000],
       [    49,  28000],
       [    47,  30000],
       [    29,  43000],
       [    31,  18000],
       [    31,  74000],
       [    27, 137000],
       [    21,  16000],
       [    28,  44000],
       [    27,  90000],
       [    35,  27000],
       [    33,  28000],
       [    30,  49000],
       [    26,  72000],
       [    27,  31000],
       [    27,  17000],
       [    33,  51000],
       [    35, 108000],
       [    30,  15000],
       [    28,  84000],
```

```
[    23,  20000],
[    25,  79000],
[    27,  54000],
[    30, 135000],
[    31,  89000],
[    24,  32000],
[    18,  44000],
[    29,  83000],
[    35,  23000],
[    27,  58000],
[    24,  55000],
[    23,  48000],
[    28,  79000],
[    22,  18000],
[    32, 117000],
[    27,  20000],
[    25,  87000],
[    23,  66000],
[    32, 120000],
[    59,  83000],
[    24,  58000],
[    24,  19000],
[    23,  82000],
[    22,  63000],
[    31,  68000],
[    25,  80000],
[    24,  27000],
[    20,  23000],
[    33, 113000],
[    32,  18000],
[    34, 112000],
[    18,  52000],
[    22,  27000],
[    28,  87000],
[    26,  17000],
[    30,  80000],
[    39,  42000],
[    20,  49000],
[    35,  88000],
[    30,  62000],
[    31, 118000],
[    24,  55000],
[    28,  85000],
[    26,  81000],
[    35,  50000],
[    22,  81000],
[    30, 116000],
[    26,  15000],
[    29,  28000],
[    29,  83000],
```

```
[    35,  44000],
[    35,  25000],
[    28, 123000],
[    35,  73000],
[    28,  37000],
[    27,  88000],
[    28,  59000],
[    32,  86000],
[    33, 149000],
[    19,  21000],
[    21,  72000],
[    26,  35000],
[    27,  89000],
[    26,  86000],
[    38,  80000],
[    39,  71000],
[    37,  71000],
[    38,  61000],
[    37,  55000],
[    42,  80000],
[    40,  57000],
[    35,  75000],
[    36,  52000],
[    40,  59000],
[    41,  59000],
[    36,  75000],
[    37,  72000],
[    40,  75000],
[    35,  53000],
[    41,  51000],
[    39,  61000],
[    42,  65000],
[    26,  32000],
[    30,  17000],
[    26,  84000],
[    31,  58000],
[    33,  31000],
[    30,  87000],
[    21,  68000],
[    28,  55000],
[    23,  63000],
[    20,  82000],
[    30, 107000],
[    28,  59000],
[    19,  25000],
[    19,  85000],
[    18,  68000],
[    35,  59000],
[    30,  89000],
[    34,  25000],
```

```
[    24,  89000],
[    27,  96000],
[    41,  30000],
[    29,  61000],
[    20,  74000],
[    26,  15000],
[    41,  45000],
[    31,  76000],
[    36,  50000],
[    40,  47000],
[    31,  15000],
[    46,  59000],
[    29,  75000],
[    26,  30000],
[    32, 135000],
[    32, 100000],
[    25,  90000],
[    37,  33000],
[    35,  38000],
[    33,  69000],
[    18,  86000],
[    22,  55000],
[    35,  71000],
[    29, 148000],
[    29,  47000],
[    21,  88000],
[    34, 115000],
[    26, 118000],
[    34,  43000],
[    34,  72000],
[    23,  28000],
[    35,  47000],
[    25,  22000],
[    24,  23000],
[    31,  34000],
[    26,  16000],
[    31,  71000],
[    32, 117000],
[    33,  43000],
[    33,  60000],
[    31,  66000],
[    20,  82000],
[    33,  41000],
[    35,  72000],
[    28,  32000],
[    24,  84000],
[    19,  26000],
[    29,  43000],
[    19,  70000],
[    28,  89000],
```

```
[    34,  43000],
[    30,  79000],
[    20,  36000],
[    26,  80000],
[    35,  22000],
[    35,  39000],
[    49,  74000],
[    39, 134000],
[    41,  71000],
[    58, 101000],
[    47,  47000],
[    55, 130000],
[    52, 114000],
[    40, 142000],
[    46,  22000],
[    48,  96000],
[    52, 150000],
[    59,  42000],
[    35,  58000],
[    47,  43000],
[    60, 108000],
[    49,  65000],
[    40,  78000],
[    46,  96000],
[    59, 143000],
[    41,  80000],
[    35,  91000],
[    37, 144000],
[    60, 102000],
[    35,  60000],
[    37,  53000],
[    36, 126000],
[    56, 133000],
[    40,  72000],
[    42,  80000],
[    35, 147000],
[    39,  42000],
[    40, 107000],
[    49,  86000],
[    38, 112000],
[    46,  79000],
[    40,  57000],
[    37,  80000],
[    46,  82000],
[    53, 143000],
[    42, 149000],
[    38,  59000],
[    50,  88000],
[    56, 104000],
[    41,  72000],
```

```
[    51, 146000],
[    35,  50000],
[    57, 122000],
[    41,  52000],
[    35,  97000],
[    44,  39000],
[    37,  52000],
[    48, 134000],
[    37, 146000],
[    50,  44000],
[    52,  90000],
[    41,  72000],
[    40,  57000],
[    58,  95000],
[    45, 131000],
[    35,  77000],
[    36, 144000],
[    55, 125000],
[    35,  72000],
[    48,  90000],
[    42, 108000],
[    40,  75000],
[    37,  74000],
[    47, 144000],
[    40,  61000],
[    43, 133000],
[    59,  76000],
[    60,  42000],
[    39, 106000],
[    57,  26000],
[    57,  74000],
[    38,  71000],
[    49,  88000],
[    52,  38000],
[    50,  36000],
[    59,  88000],
[    35,  61000],
[    37,  70000],
[    52,  21000],
[    48, 141000],
[    37,  93000],
[    37,  62000],
[    48, 138000],
[    41,  79000],
[    37,  78000],
[    39, 134000],
[    49,  89000],
[    55,  39000],
[    37,  77000],
[    35,  57000],
```

```
[    36,  63000],
[    42,  73000],
[    43, 112000],
[    45,  79000],
[    46, 117000],
[    58,  38000],
[    48,  74000],
[    37, 137000],
[    37,  79000],
[    40,  60000],
[    42,  54000],
[    51, 134000],
[    47, 113000],
[    36, 125000],
[    38,  50000],
[    42,  70000],
[    39,  96000],
[    38,  50000],
[    49, 141000],
[    39,  79000],
[    39,  75000],
[    54, 104000],
[    35,  55000],
[    45,  32000],
[    36,  60000],
[    52, 138000],
[    53,  82000],
[    41,  52000],
[    48,  30000],
[    48, 131000],
[    41,  60000],
[    41,  72000],
[    42,  75000],
[    36, 118000],
[    47, 107000],
[    38,  51000],
[    48, 119000],
[    42,  65000],
[    40,  65000],
[    57,  60000],
[    36,  54000],
[    58, 144000],
[    35,  79000],
[    38,  55000],
[    39, 122000],
[    53, 104000],
[    35,  75000],
[    38,  65000],
[    47,  51000],
[    47, 105000],
```

```
[    41,  63000],
[    53,  72000],
[    54, 108000],
[    39,  77000],
[    38,  61000],
[    38, 113000],
[    37,  75000],
[    42,  90000],
[    37,  57000],
[    36,  99000],
[    60,  34000],
[    54,  70000],
[    41,  72000],
[    40,  71000],
[    42,  54000],
[    43, 129000],
[    53,  34000],
[    47,  50000],
[    42,  79000],
[    42, 104000],
[    59,  29000],
[    58,  47000],
[    46,  88000],
[    38,  71000],
[    54,  26000],
[    60,  46000],
[    60,  83000],
[    39,  73000],
[    59, 130000],
[    37,  80000],
[    46,  32000],
[    46,  74000],
[    42,  53000],
[    41,  87000],
[    58,  23000],
[    42,  64000],
[    48,  33000],
[    44, 139000],
[    49,  28000],
[    57,  33000],
[    56,  60000],
[    49,  39000],
[    39,  71000],
[    47,  34000],
[    48,  35000],
[    48,  33000],
[    47,  23000],
[    45,  45000],
[    60,  42000],
[    39,  59000],
```

```
       [    46,  41000],
       [    51,  23000],
       [    50,  20000],
       [    36,  33000],
       [    49,  36000]], dtype=int64)
```

```
y  = df.iloc[:,4].values
```

```
y
```

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

```
df['Purchased'].value_counts()
```

```
0    257
1    143
Name: Purchased, dtype: int64
```

Train,Test and split

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size =
0.2,random_state = 0)
```

feature scaling

```
#cannot apply for regression problem, we can only implement for
classification problems
#X' = X-Xmin/Xmax-Xmin if num == den then x' = 1 it is a maximum case
in feature scaling
#Standardization is one of the scaling techinque
#x' = x-u/sigma
#nomalization is useful when the distribution is not in the form of
guassian distribution
1.x' = 0 min
2.x' = 1 max
3.x' =inbetween 0 and 1
#standardization is not useful when the distribution is not in the
form of guassian distribution
```

```
  File
"C:\Users\bharg\AppData\Local\Temp/ipykernel_1992/220819968.py", line
6
    1.x' = 0 min
      ^
SyntaxError: invalid syntax
```

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

```
x_train
```

```
x_test[0:5]
```

```
x_test = st.transform(x_test)
```

```
x_test[0:5]
```

Model Building

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr.fit(x_train,y_train)

LogisticRegression()

lrpred = lr.predict(x_test)

lrpred

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

y_test

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

Confusion Matrix

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(lrpred,y_test)

cm

array([[58, 22],
       [ 0,  0]], dtype=int64)
```

Accuracy score

```
from sklearn.metrics import accuracy_score
acc = accuracy_score(lrpred,y_test)

acc

0.725
```

KNN - K Nearest Neighbours

```
# In order to calculate the distance between two points we are using different methods
1.Eculidean Distance
2.Manhattan Distance
3.Minkowski Distance

step1:
    Choose the number of k of neighbours
step2:
    Take the k nearest neighbours of the new datapoint,according to the euclidan distance
step3:
    Among the k neighbours, count the number of datapoints in each category
step4:
    Assign the new data points to the category where you counted the most neighbors
Model is ready
```

```
from sklearn.neighbors import KNeighborsClassifier
knn =  KNeighborsClassifier(n_neighbors=5,metric='euclidean')
```

```
knn.fit(x_train,y_train)
```

```
knn
```

```
knnpred = knn.predict(x_test)
```

```
knnpred
```

```
cm = confusion_matrix(knnpred,y_test)
```

```
cm
```

```
from sklearn.metrics import accuracy_score
acc = accuracy_score(knnpred,y_test)
```

```
acc
```

```
knn.predict([[19,38900]])
```

Naive Bayes

```
# naive is specifies the occurence of one feature independent to the occurence of another feature
# bayes is used to determine the hypothesis on prior knowledge
```

```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(x_train,y_train)
```

```
GaussianNB()
```

```
gnbpred = gnb.predict(x_test)
```

```
gnbpred
```

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

```
y_test
```

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

```
cm = confusion_matrix(gnbpred,y_test)
```

```
cm
```

```
array([[56,  4],
       [ 2, 18]], dtype=int64)
```

```
gnbacc = accuracy_score(gnbpred,y_test)
```

```
gnbacc
```

```
0.925
```