SNO	DATE	EXPERIMENT	PAGE NO.	MARKS	SIGN
1.		Implementation of Univariate Linear Regression to fit a straight line using least squares.			
2.		Implementation of Simple Linear Regression Model for Predicting the Marks Scored			
3.		Implementation of Simple Linear Regression Model Using Gradient descent			
4.		Implementation of Logistic Regression Model to Predict the Placement Status of Student			
5.		Implementation of Logistic Regression Using Gradient Descent			
6.		Implementation of Decision Tree Classifier Model for Predicting Employee Churn			
7.		Implementation of Decision Tree Regressor Model for Predicting the Salary of the Employee			
8.		Implementation of K Means Clustering for Customer Segmentation			
9.		Implementation of SVM For Spam Mail Detection			

Implementation of Univariate Linear Regression

AIM:

To implement univariate Linear Regression to fit a straight line using least squares.

Equipments Required:

1. Hardware – PCs
2. Anaconda – Python 3.7 Installation / Jupyter notebook

Algorithm

1. Get the independent variable X and dependent variable Y.
2. Calculate the mean of the X -values and the mean of the Y -values.
3. Find the slope m of the line of best fit using the formula.

$$m = \frac{\sum_{i=1}^{n}(x_i-\overline{X})(y_i-\overline{Y})}{\sum_{i=1}^{n}(x_i-\overline{X})^2}$$

4. Compute the y -intercept of the line by using the formula:

$$b = \overline{Y} - m\overline{X}$$

5. Use the slope m and the y -intercept to form the equation of the line. 6. Obtain the straight line equation Y=mX+b and plot the scatterplot.

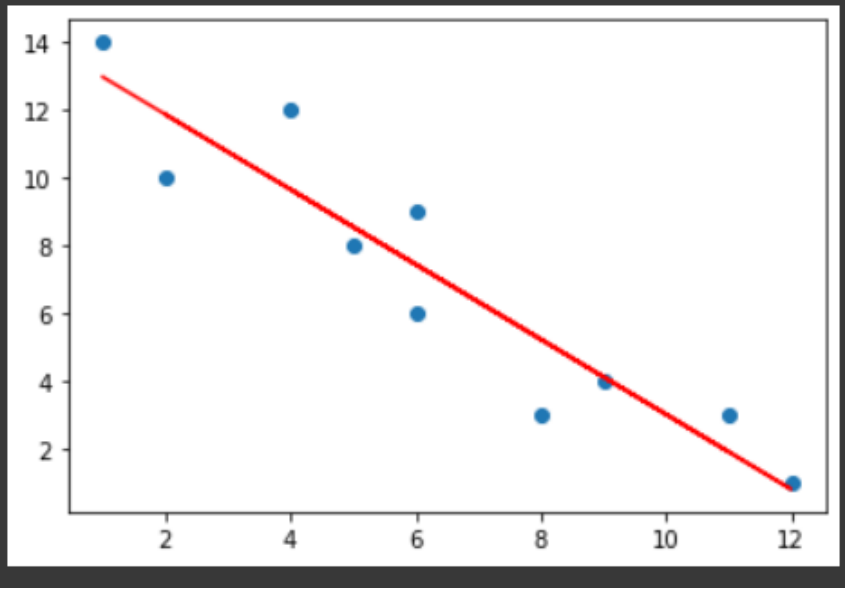
Program:

```
/*
Program to implement univariate Linear Regression to fit a straight line using least
squares.
Developed by: Dhanush.G.R.
RegisterNumber:  212221040038

import numpy as np
import matplotlib.pyplot as plt
#Getting the values of x&y
x=np.array(eval(input()))
y=np.array(eval(input()))
#Mean
x_mean=np.mean(x)
```

```
y_mean=np.mean(y)
num,denom=0,0
for i in range(len(x)):
  num+=((x[i]-x_mean)*(y[i]-y_mean))
  denom+=(x[i]-x_mean)**2
m=num/denom
b=y_mean-m*x_mean
print(m,b)
y_predicted=m*x+b
print(y_predicted)
plt.scatter(x,y)
plt.plot(x,y_predicted,color='red')
plt.show()
*/
```

Output:

```
8,2,11,6,5,4,12,9,6,1
3,10,3,6,8,12,1,4,9,14
-1.1064189189189189 14.08108108108108
[ 5.22972973 11.86824324  1.91047297  7.44256757  8.54898649  9.65540541
  0.80405405  4.12331081  7.44256757 12.97466216]
```



Result:

Thus the univariate Linear Regression was implemented to fit a straight line using least squares using python programming.

Implementation-of-Simple-Linear-Regression-Model-for-Predicting-the-Marks-Scored

AIM:

To write a program to predict the marks scored by a student using the simple linear regression model.

Equipments Required:

1. Hardware – PCs
2. Anaconda – Python 3.7 Installation / Jupyter notebook

Algorithm

1. Use the standard libraries in python for Gradient Design.
2. Set variables for assigning dataset values.
3. Import linear regression from sklearn.
4. Assign the points for representing the graph.
5. Predict the regression for marks by using the representation of the graph.
6. Compare the graphs and hence we obtained the linear regression for the given data.

Program:

```
/*
Program to implement the simple linear regression model for predicting the marks scored.
Developed by: Dhanush.G.R.
RegisterNumber:  212221040038
*/
```

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.metrics import mean_absolute_error,mean_squared_error

df=pd.read_csv('/content/student_scores.csv')

print('df.head:')

#displaying the content in datafile
```

```
df.head()
print("df.tail:")
df.tail()
print("Array value of x : ")
X=df.iloc[:,:-1].values
X
print("Array value of y : ")
Y=df.iloc[:,1].values
Y
#splitting train and test data
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=1/3,random_state=0)
from sklearn.linear_model import LinearRegression
regressor=LinearRegression()
regressor.fit(X_train,Y_train)
Y_pred=regressor.predict(X_test)
#displaying predicted values
print("Predicted Values:")
Y_pred
#displaying actual values
print("Actual Values:")
Y_test
#graph plot for training data
print("Graph plot for training data:")
plt.scatter(X_train,Y_train,color="orange")
plt.plot(X_train,regressor.predict(X_train),color="red")
plt.title("Hours vs Scores (Training Set)")
plt.xlabel("Hours")
```

```
plt.ylabel("Scores")
plt.show()
#graph plot for training data
print("Graph plot for training data:")
plt.scatter(X_test,Y_test,color="purple")
plt.plot(X_test,regressor.predict(X_test),color="pink")
plt.title("Hours vs Scores (Training Set)")
plt.xlabel("Hours")
plt.ylabel("Scores")
plt.show()
print('Values of MSE,MAE,RMSE:')
mse=mean_squared_error(Y_test,Y_pred)
print('MSE = ',mse)
mae=mean_absolute_error(Y_test,Y_pred)
print('MAE = ',mae)
rmse=np.sqrt(mse)
print("RMSE = " ,rmse)
```

Output:

```
df.head:
```

	Hours	Scores
0	2.5	21
1	5.1	47
2	3.2	27
3	8.5	75
4	3.5	30

df.tail:

	Hours	Scores
20	2.7	30
21	4.8	54
22	3.8	35
23	6.9	76
24	7.8	86

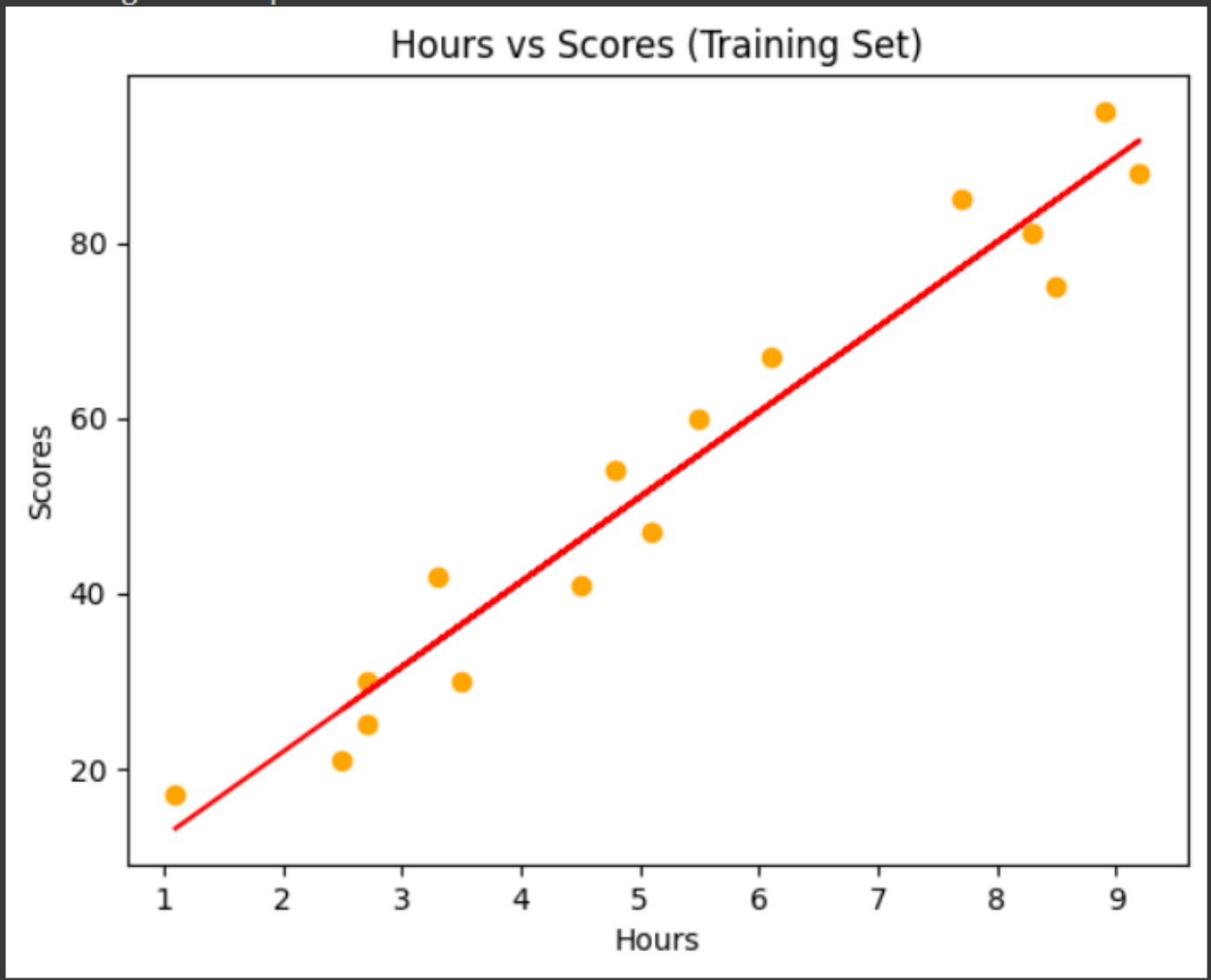
```
Array value of x :
array([[2.5],
       [5.1],
       [3.2],
       [8.5],
       [3.5],
       [1.5],
       [9.2],
       [5.5],
       [8.3],
       [2.7],
       [7.7],
       [5.9],
       [4.5],
       [3.3],
       [1.1],
       [8.9],
       [2.5],
       [1.9],
       [6.1],
       [7.4],
       [2.7],
       [4.8],
       [3.8],
       [6.9],
       [7.8]])
```

```
Array value of y :
array([21, 47, 27, 75, 30, 20, 88, 60, 81, 25, 85, 62, 41, 42, 17, 95, 30,
       24, 67, 69, 30, 54, 35, 76, 86])
```

```
Values of Y prediction
array([17.04289179, 33.51695377, 74.21757747, 26.73351648, 59.68164043,
       39.33132858, 20.91914167, 78.09382734, 69.37226512])
```

```
Array values of Y test:
array([20, 27, 69, 30, 62, 35, 24, 86, 76])
```

Training Set Graph:



Test Set Graph:


```
Values of MSE,MAE,RMSE:
MSE =  25.463280738222593
MAE =  4.691397441397446
RMSE =  5.046115410711748
```

Result:

Thus the program to implement the simple linear regression model for predicting the marks scored is written and verified using python programming.

Implementation-of-Linear-Regression-Using-Gradient-Descent

AIM:

To write a program to predict the profit of a city using the linear regression model with gradient descent.

Equipments Required:

1. Hardware – PCs
2. Anaconda – Python 3.7 Installation / Jupyter notebook

Algorithm

1. Import the standard python libraries for Gradient design.
2. Introduce the variables needed to execute the function.
3. Use function for the representation of the graph.
4. Using for loop apply the concept using the formulae.
5. Execute the program and plot the graph.
6. Predict and execute the values for the given conditions.

Program:

```
/*
Program to implement the linear regression using gradient descent.
Developed by: Dhanush.G.R.
RegisterNumber:  212221040038
*/
```

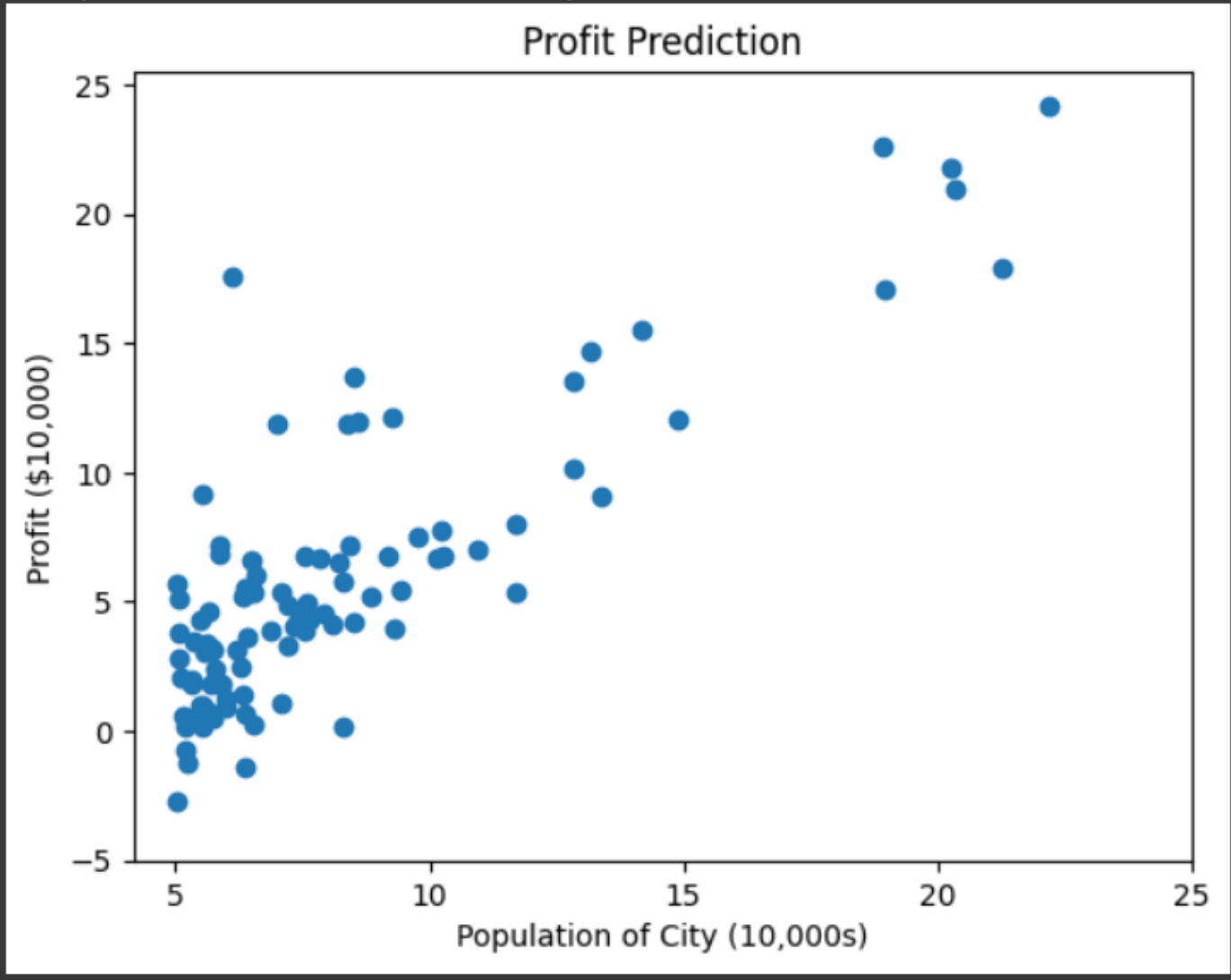
```
import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

data=pd.read_csv("/content/ex1.txt",header=None)

print("Profit Prediction Graph:")

plt.scatter(data[0],data[1])

plt.xticks(np.arange(5,30,step=5))
```

```
plt.yticks(np.arange(-5,30,step=5))
plt.xlabel("Population of City (10,000s)")
plt.ylabel("Profit ($10,000)")
plt.title("Profit Prediction") def computeCost(X,y,theta):
"""
Take in numpy array X,y,theta and generate the cost function in a linear regression model
"""
m=len(y)
h=X.dot(theta)
square_err=(h-y)**2
return 1/(2*m) * np.sum(square_err)
data_n=data.values
m=data_n[:,0].size
X=np.append(np.ones((m,1)),data_n[:,0].reshape(m,1),axis=1)
y=data_n[:,1].reshape(m,1)
theta=np.zeros((2,1))
print("Compute Cost Value:")
computeCost(X,y,theta)#call the function
def gradientDescent(X,y,theta,alpha,num_iters):
m=len(y)
J_history=[]
for i in range(num_iters):
predictions=X.dot(theta)
error=np.dot(X.transpose(),(predictions -y))
descent=alpha * 1/m * error
theta-=descent
J_history.append(computeCost(X,y,theta))
return theta,J_history
```

```
print("h(x) value:")
theta,J_history=gradientDescent(X,y,theta,0.01,1500)
print("h(x) ="+str(round(theta[0,0],2))+" + "+str(round(theta[1,0],2))+"x1")
print("Cost function using Gradient Descent:")
plt.plot(J_history)
plt.xlabel("Iteration")
plt.ylabel("$J(\Theta)$")
plt.title("Cost function using Gradient Descent")
print("Profit Prediction:")
plt.scatter(data[0],data[1])
x_value=[x for x in range(25)]
y_value=[y*theta[1]+theta[0] for y in x_value]
plt.plot(x_value,y_value,color="r")
plt.xticks(np.arange(5,30,step=5))
plt.yticks(np.arange(-5,30,step=5))
plt.xlabel("Population of City (10,000)")
plt.ylabel("Profit ($10,000)")
plt.title("Profit Prediction")
def predict(x,theta):
predictions=np.dot(theta.transpose(),x)
return predictions[0]
print("Profit for the Population 35,000:")
predict1=predict(np.array([1,3.5]),theta)*1000
print("For population = 35,000 we predict a profit of $"+str(round(predict1,0)))
print("Profit for the Population 70,000:")
predict2=predict(np.array([1,7]),theta)*1000
print("For population = 70,000 we predict a profit of $"+str(round(predict2,0)))
```

Output:

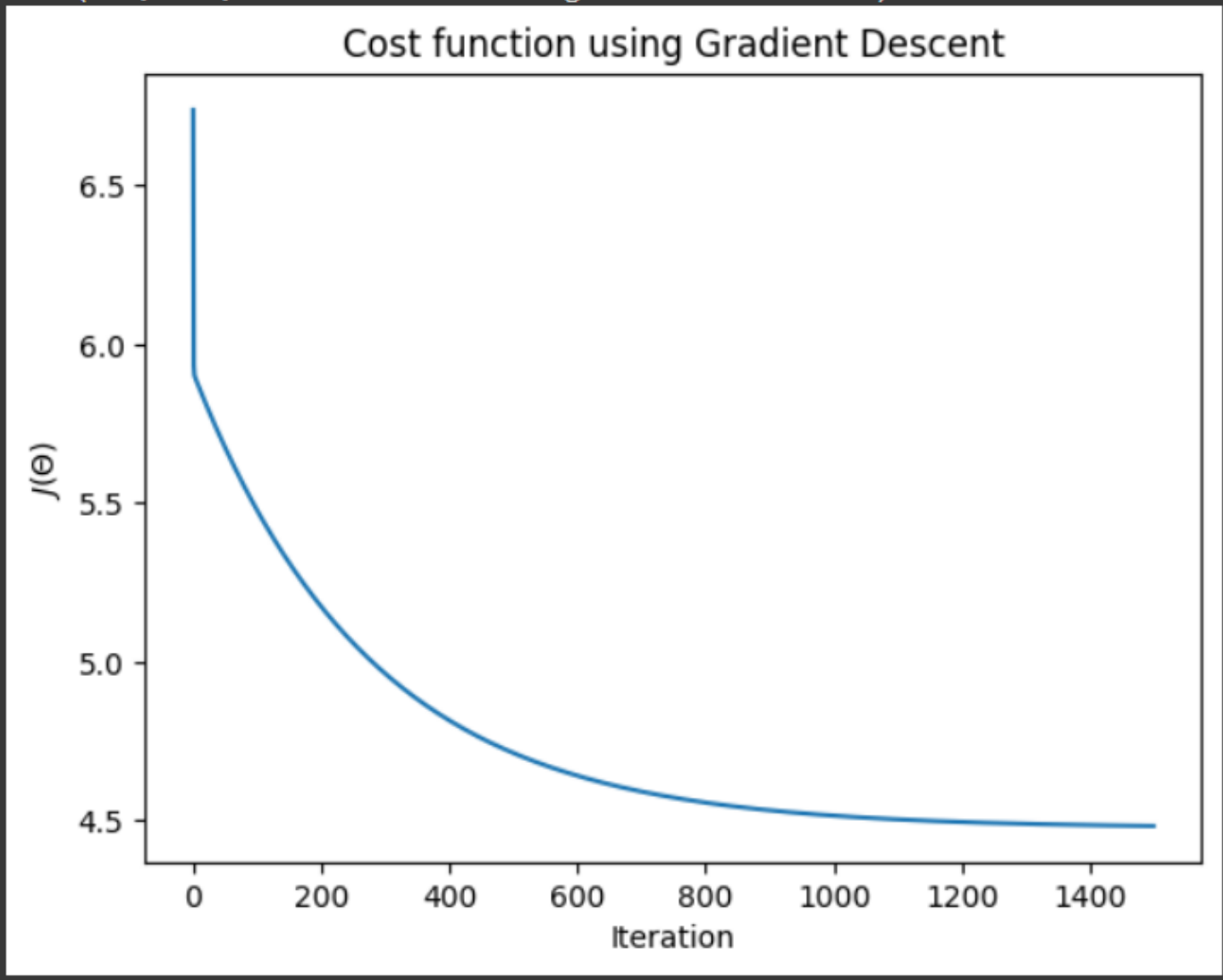
```
Profit Prediction Graph:
Text(0.5, 1.0, 'Profit Prediction')
```



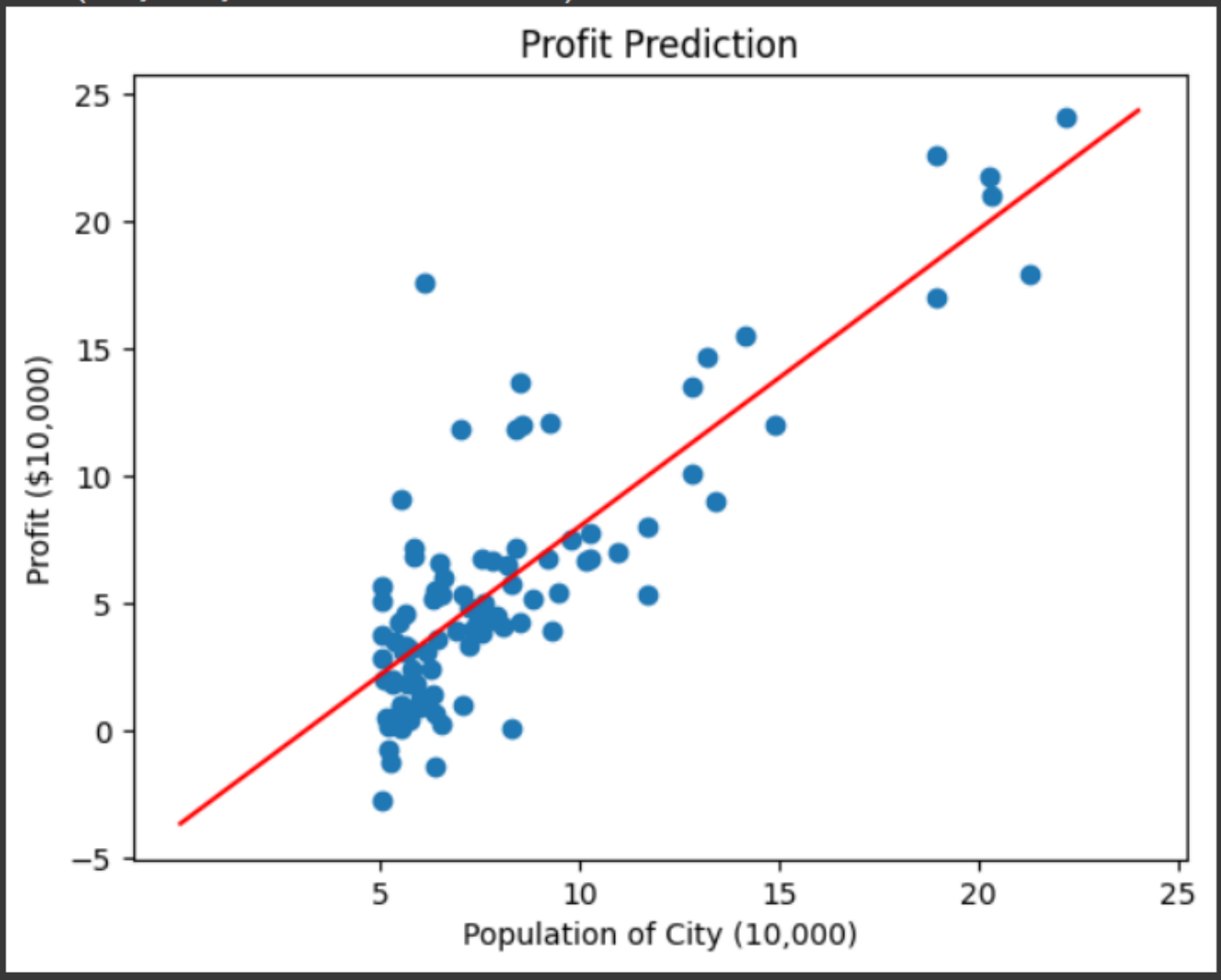

```
Compute Cost Value:
32.072733877455676
```

```
  h(x) value:
  h(x) =-3.63 + 1.17x1
```

```
Cost function using Gradient Descent:
Text(0.5, 1.0, 'Cost function using Gradient Descent')
```



```
Profit Prediction:
Text(0.5, 1.0, 'Profit Prediction')
```




```
Profit for the Population 35,000:
For population = 35,000 we predict a profit of $452.0
```

```
Profit for the Population 70,000:
For population = 70,000 we predict a profit of $4534.0
```

Result:

Thus the program to implement the linear regression using gradient descent is written and verified using python programming.

Implementation-of-Logistic-Regression-Model-to-Predict-the-Placement-Status-of-Student

AIM:

To write a program to implement the the Logistic Regression Model to Predict the Placement Status of Student.

Equipments Required:

1. Hardware – PCs
2. Anaconda – Python 3.7 Installation / Jupyter notebook

Algorithm

1. Import dataset.

2.Check for null and duplicate values.

3.Assign x and y values.

4.Split data into train and test data.

5.Import logistic regression and fit the training data.

6.Predict y value.

7.Calculate accuracy and confusion matrix.

Program:

```
/*
Program to implement the the Logistic Regression Model to Predict the Placement
Status of Student.
Developed by: DHANUSH.G.R.
RegisterNumber:212221040038
*/
```

```
import pandas as pd

data=pd.read_csv('/content/Placement_Data.csv')

print("Placement data:")

data.head()
```

```
data1=data.copy()
data1=data1.drop(["sl_no","salary"],axis=1)#removes the specified row or coloumn
print("Salary data:")
data1.head()
print("Checking the null() function:")
data1.isnull().sum()
print ("Data Duplicate:")
data1.duplicated().sum()
print("Print data:")
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
data1["gender"]=le.fit_transform(data1["gender"])
data1["ssc_b"]=le.fit_transform(data1["ssc_b"])
data1["hsc_b"]=le.fit_transform(data1["hsc_b"])
data1["hsc_s"]=le.fit_transform(data1["hsc_s"])
data1["degree_t"]=le.fit_transform(data1["degree_t"])
data1["workex"]=le.fit_transform(data1["workex"])
data1["specialisation"]=le.fit_transform(data1["specialisation"])
data1["status"]=le.fit_transform(data1["status"])
data1
print("Data-status value of x:")
x=data1.iloc[:,:-1]
x
print("Data-status value of y:")
y=data1["status"]
y
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)
```

```
print ("y_prediction array:")

from sklearn.linear_model import LogisticRegression

lr=LogisticRegression(solver="liblinear")#a library for large

lr.fit(x_train,y_train)

y_pred=lr.predict(x_test)

y_pred

from sklearn.metrics import accuracy_score

accuracy=accuracy_score(y_test,y_pred)

print("Accuracy value:")

accuracy

from sklearn.metrics import confusion_matrix

confusion=(y_test,y_pred)

print("Confusion array:")

confusion

from sklearn.metrics import classification_report

classification_report1=classification_report(y_test,y_pred)

print("Classification report:")

print(classification_report1)

print("Prediction of LR:")

lr.predict([[1,80,1,90,1,1,90,1,0,85,1,85]])
```

Output:

Placement data:

	sl_no	gender	ssc_p	ssc_b	hsc_p	hsc_b	hsc_s	degree_p	degree_t	workex	etest_p	specialisation	mba_p	status	salary
0	1	M	67.00	Others	91.00	Others	Commerce	58.00	Sci&Tech	No	55.0	Mkt&HR	58.80	Placed	270000.0
1	2	M	79.33	Central	78.33	Others	Science	77.48	Sci&Tech	Yes	86.5	Mkt&Fin	66.28	Placed	200000.0
2	3	M	65.00	Central	68.00	Central	Arts	64.00	Comm&Mgmt	No	75.0	Mkt&Fin	57.80	Placed	250000.0
3	4	M	56.00	Central	52.00	Central	Science	52.00	Sci&Tech	No	66.0	Mkt&HR	59.43	Not Placed	NaN
4	5	M	85.80	Central	73.60	Central	Commerce	73.30	Comm&Mgmt	No	96.8	Mkt&Fin	55.50	Placed	425000.0

Salary data:

	gender	ssc_p	ssc_b	hsc_p	hsc_b	hsc_s	degree_p	degree_t	workex	etest_p	specialisation	mba_p	status
0	M	67.00	Others	91.00	Others	Commerce	58.00	Sci&Tech	No	55.0	Mkt&HR	58.80	Placed
1	M	79.33	Central	78.33	Others	Science	77.48	Sci&Tech	Yes	86.5	Mkt&Fin	66.28	Placed
2	M	65.00	Central	68.00	Central	Arts	64.00	Comm&Mgmt	No	75.0	Mkt&Fin	57.80	Placed
3	M	56.00	Central	52.00	Central	Science	52.00	Sci&Tech	No	66.0	Mkt&HR	59.43	Not Placed
4	M	85.80	Central	73.60	Central	Commerce	73.30	Comm&Mgmt	No	96.8	Mkt&Fin	55.50	Placed

Checking the null() function:

```
gender            0
ssc_p             0
ssc_b             0
hsc_p             0
hsc_b             0
hsc_s             0
degree_p          0
degree_t          0
workex            0
etest_p           0
specialisation    0
mba_p             0
status            0
dtype: int64
```

Data Duplicate:

```
0
```

Print data:

	gender	ssc_p	ssc_b	hsc_p	hsc_b	hsc_s	degree_p	degree_t	workex	etest_p	specialisation	mba_p	status
0	1	67.00	1	91.00	1	1	58.00	2	0	55.0	1	58.80	1
1	1	79.33	0	78.33	1	2	77.48	2	1	86.5	0	66.28	1
2	1	65.00	0	68.00	0	0	64.00	0	0	75.0	0	57.80	1
3	1	56.00	0	52.00	0	2	52.00	2	0	66.0	1	59.43	0
4	1	85.80	0	73.60	0	1	73.30	0	0	96.8	0	55.50	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...
210	1	80.60	1	82.00	1	1	77.60	0	0	91.0	0	74.49	1
211	1	58.00	1	60.00	1	2	72.00	2	0	74.0	0	53.62	1
212	1	67.00	1	67.00	1	1	73.00	0	1	59.0	0	69.72	1
213	0	74.00	1	66.00	1	1	58.00	0	0	70.0	1	60.23	1
214	1	62.00	0	58.00	1	2	53.00	0	0	89.0	1	60.22	0

215 rows × 13 columns

Data-status value of x:

	gender	ssc_p	ssc_b	hsc_p	hsc_b	hsc_s	degree_p	degree_t	workex	etest_p	specialisation	mba_p
0	1	67.00	1	91.00	1	1	58.00	2	0	55.0	1	58.80
1	1	79.33	0	78.33	1	2	77.48	2	1	86.5	0	66.28
2	1	65.00	0	68.00	0	0	64.00	0	0	75.0	0	57.80
3	1	56.00	0	52.00	0	2	52.00	2	0	66.0	1	59.43
4	1	85.80	0	73.60	0	1	73.30	0	0	96.8	0	55.50
...	...	...	...	...	...	...	...	...	...	...	...	...
210	1	80.60	1	82.00	1	1	77.60	0	0	91.0	0	74.49
211	1	58.00	1	60.00	1	2	72.00	2	0	74.0	0	53.62
212	1	67.00	1	67.00	1	1	73.00	0	1	59.0	0	69.72
213	0	74.00	1	66.00	1	1	58.00	0	0	70.0	1	60.23
214	1	62.00	0	58.00	1	2	53.00	0	0	89.0	1	60.22

215 rows × 12 columns

```
Data-status value of y:
0      1
1      1
2      1
3      0
4      1
      ..
210    1
211    1
212    1
213    1
214    0
Name: status, Length: 215, dtype: int64
```

```
y_prediction array:
array([0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1,
       1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1])
```

```
Accuracy value:
0.813953488372093
```

```
Confusion array:
(198    0
 37     1
 89     1
 168    0
 171    1
 75     0
 96     1
 137    1
 5      0
 83     1
 55     1
 145    1
 160    1
 112    1
 74     1
 203    1
 126    1
 12     0
 153    1
 158    0
 169    0
 141    0
 209    1
 190    0
 144    0
 18     0
 185    1
 15     1
 86     1
 71     1
 7      1
 63     0
 143    1
 97     0
 136    0
 162    1
 33     1
 154    1
 90     1
 211    1
 106    0
 181    0
 139    1
 Name: status, dtype: int64,
 array([0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1,
        1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1]))
```

```
Classification report:
              precision    recall  f1-score   support

           0       0.79      0.69      0.73        16
           1       0.83      0.89      0.86        27

    accuracy                           0.81        43
   macro avg       0.81      0.79      0.80        43
weighted avg       0.81      0.81      0.81        43
```

```
Prediction of LR:
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but LogisticRegression was fitted with feature names
  warnings.warn(
array([0])
```

Result:

Thus the program to implement the the Logistic Regression Model to Predict the Placement Status of Student is written and verified using python programming.

Implementation-of-Logistic-Regression-Using-Gradient-Descent

AIM:

To write a program to implement the the Logistic Regression Using Gradient Descent.

Equipments Required:

1. Hardware – PCs
2. Anaconda – Python 3.7 Installation / Jupyter notebook

Algorithm

1. Use the standard libraries in python for finding linear regression.

2.Set variables for assigning dataset values.

3.Import linear regression from sklearn.

4.Predict the values of array.

5.Calculate the accuracy, confusion and classification report by importing the required modules from sklearn.

Program:

```
/*
Program to implement the the Logistic Regression Using Gradient Descent.
Developed by: DHANUSH GR
RegisterNumber:  212221040038
*/
```

```
import numpy as np

import matplotlib.pyplot as plt

from scipy import optimize

data=np.loadtxt("/content/ex2data1 (1).txt",delimiter=',')

x=data[:,[0,1]]

y=data[:,2]
```

```
print("Array value of x:")
x[:5]
print("Array value of y:")
y[:5]
print("Exam 1-score graph:")
plt.figure()
plt.scatter(x[y==1][:,0],x[y==1][:,1],label="Admitted")
plt.scatter(x[y==0][:,0],x[y==0][:,1],label="Not admitted")
plt.xlabel("Exam 1 score")
plt.ylabel("Exam 2 score")
plt.legend()
plt.show()
def sigmoid(z):
return 1/(1+np.exp(-z))
print("Sigmoid function graph: ")
plt.plot()
x_plot=np.linspace(-10,10,100)
plt.plot(x_plot,sigmoid(x_plot))
plt.show()
def costFunction(theta,x,y):
h=sigmoid(np.dot(x,theta))
J=-(np.dot(y,np.log(h))+np.dot(1-y,np.log(1-h)))/x.shape[0]
grad=np.dot(x.T,h-y)/x.shape[0]
return J,grad
x_train=np.hstack((np.ones((x.shape[0],1)),x))
theta=np.array([0,0,0])
J,grad=costFunction(theta,x_train,y)
print("x_train_grad value:")
```

```
print(J)
print(grad)
x_train=np.hstack((np.ones((x.shape[0],1)),x))
theta=np.array([-24,0.2,0.2])
J,grad=costFunction(theta,x_train,y)
print("y_train_grad value:")
print(J)
print(grad)
def cost(theta,x,y):
h=sigmoid(np.dot(x,theta))
J=-(np.dot(y,np.log(h))+np.dot(1-y,np.log(1-h)))/x.shape[0]
return J
def gradient(theta,x,y):
h=sigmoid(np.dot(x,theta))
grad=np.dot(x.T,h-y)/x.shape[0]
return grad
x_train=np.hstack((np.ones((x.shape[0],1)),x))
theta=np.array([0,0,0])
res=optimize.minimize(fun=cost,x0=theta,args=(x_train,y),method='Newton-CG',jac=gradient)
print("res.x:")
print(res.fun)
print(res.x)
def plotDecisionBoundary(theta,x,y):
x_min,x_max=x[:,0].min()-1,x[:,0].max()+1
y_min,y_max=x[:,1].min()-1,x[:,1].max()+1
xx,yy=np.meshgrid(np.arange(x_min,x_max,0.1),np.arange(y_min,y_max,0.1))
x_plot=np.c_[xx.ravel(),yy.ravel()]
x_plot=np.hstack((np.ones((x_plot.shape[0],1)),x_plot))
```

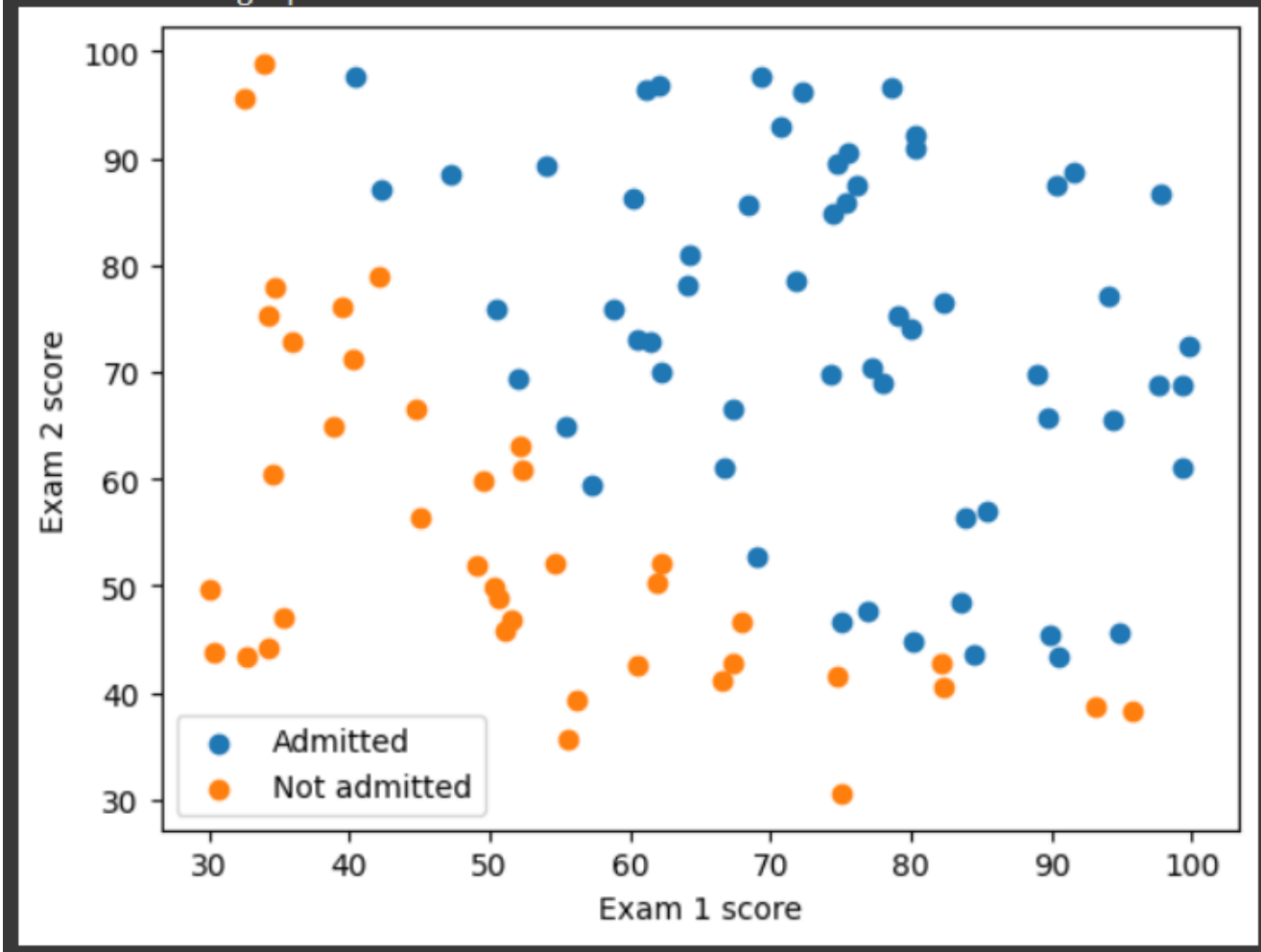
```
y_plot=np.dot(x_plot,theta).reshape(xx.shape)
plt.figure()
plt.scatter(x[y==1][:,0],x[y==1][:,1],label="Admitted")
plt.scatter(x[y==0][:,0],x[y==0][:,1],label="Admitted")
plt.contour(xx,yy,y_plot,levels=[0])
plt.xlabel("Exam 1 score")
plt.ylabel("Exam 2 score")
plt.legend()
plt.show()
print("Descision Boundary - graph for exam score:")
plotDecisionBoundary(res.x,x,y)
print("probability value:")
prob=sigmoid(np.dot(np.array([1,45,85]),res.x))
print(prob)
def predict(theta, x):
x_train = np.hstack((np.ones((x.shape[0], 1)), x))
prob = sigmoid(np.dot(x_train, theta))
return (prob >= 0.5).astype(int)
print("Prediction value of mean:")
np.mean(predict(res.x, x) == y)
```

Output:

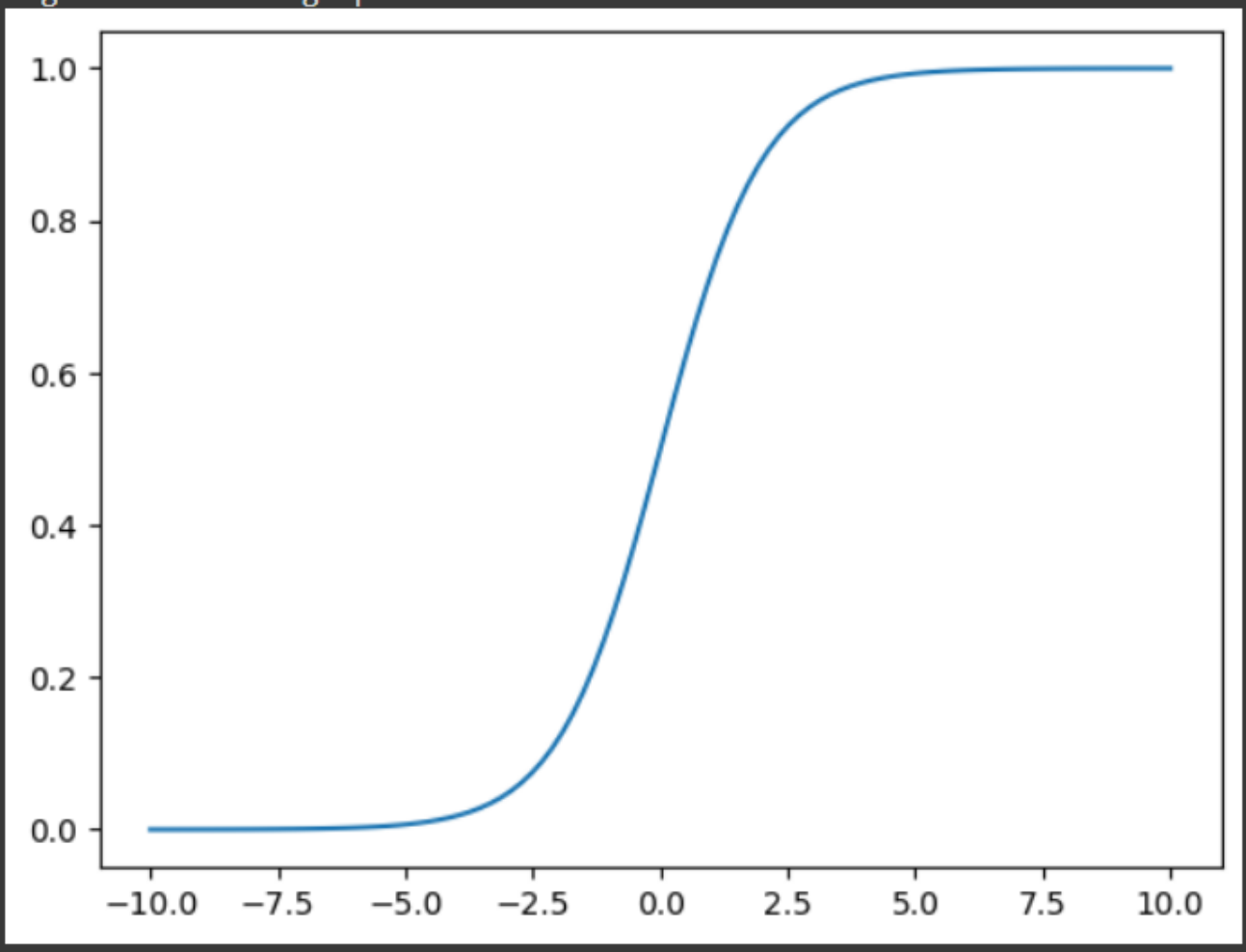
```
Array value of x:
array([[34.62365962, 78.02469282],
       [30.28671077, 43.89499752],
       [35.84740877, 72.90219803],
       [60.18259939, 86.3085521 ],
       [79.03273605, 75.34437644]])
```

```
Array value of y:
array([0., 0., 0., 1., 1.])
```

Exam 1-score graph:



Sigmoid function graph:

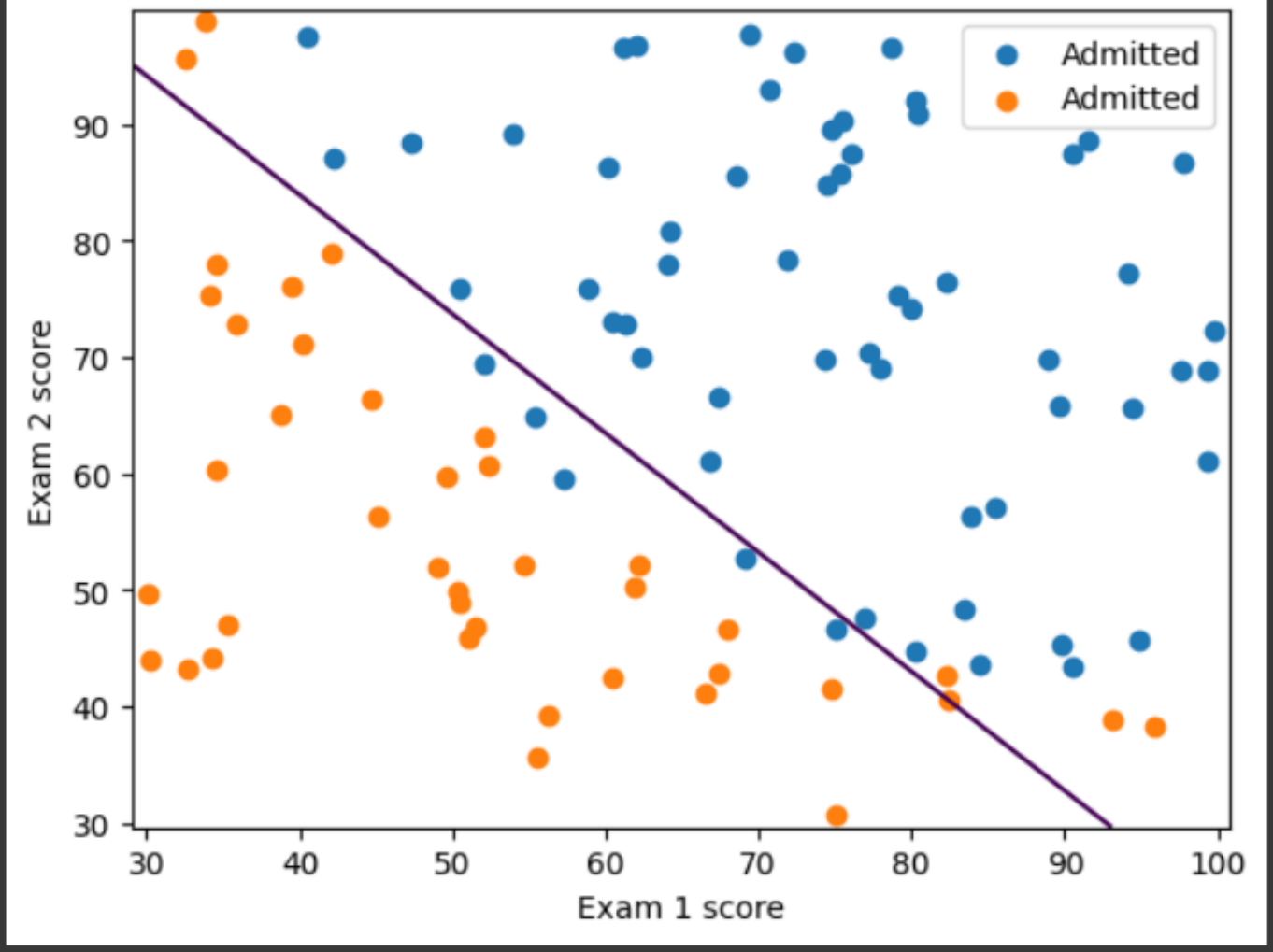



```
x_train_grad value:
0.6931471805599452
[ -0.1        -12.00921659 -11.26284221]
```

```
y_train_grad value:
0.2183301938265977
[0.04290299 2.56623412 2.64679737]
```

```
res.x:
0.20349770158945205
[-25.16134246   0.20623179   0.20147167]
```

```
Descision Boundary - graph for exam score:
```




```
probability value:
0.7762907420026233
```

```
Prediction value of mean:
0.89
```

Result:

Thus the program to implement the the Logistic Regression Using Gradient Descent is written and verified using python programming.

Implementation-of-Decision-Tree-Classifier-Model-for-Predicting-Employee-Churn

AIM:

To write a program to implement the Decision Tree Classifier Model for Predicting Employee Churn.

Equipments Required:

1. Hardware – PCs
2. Anaconda – Python 3.7 Installation / Jupyter notebook

Algorithm

1. Import standard libraries in python for finding Decision tree classsifier model for predicting employee churn.

2.Initialize and print the Data.head(),data.info(),data.isnull().sum()

3. Visualize data value count.

4. Import sklearn from LabelEncoder.

5. Split data into training and testing.

6. Calculate the accuracy, data prediction by importing the required modules from sklearn

Program:

```
/*
Program to implement the Decision Tree Classifier Model for Predicting Employee Churn.
Developed by: G.R.DHANUSH
RegisterNumber:  212221040038
*/

import pandas as pd

data=pd.read_csv("/content/Employee.csv")

print("data.head():")

data.head()

print("data.info():")

data.info()

print("isnull() and sum():")

data.isnull().sum()

print("data value counts():")

data["left"].value_counts()

from sklearn.preprocessing import LabelEncoder

le=LabelEncoder()

print("data.head() for Salary:")

data["salary"]=le.fit_transform(data["salary"])

data.head()

print("x.head():")

x=data[["satisfaction_level","last_evaluation","number_project","average_montly_hours","time_spend_company","Work_accident","promotion_last_5years

x.head()

y=data["left"]

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=100)

from sklearn.tree import DecisionTreeClassifier

dt=DecisionTreeClassifier(criterion="entropy")

dt.fit(x_train,y_train)

y_pred=dt.predict(x_test)

print("Accuracy value:")

from sklearn import metrics

accuracy=metrics.accuracy_score(y_test,y_pred)

accuracy

print("Data Prediction:")

dt.predict([[0.5,0.8,9,260,6,0,1,2]])
```

Output:

data.head():

	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	Work_accident	left	promotion_last_5years	Departments	salary
0	0.38	0.53	2	157	3	0	1	0	sales	low
1	0.80	0.86	5	262	6	0	1	0	sales	medium
2	0.11	0.88	7	272	4	0	1	0	sales	medium
3	0.72	0.87	5	223	5	0	1	0	sales	low
4	0.37	0.52	2	159	3	0	1	0	sales	low

```
data.info():
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14999 entries, 0 to 14998
Data columns (total 10 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   satisfaction_level     14999 non-null  float64
 1   last_evaluation        14999 non-null  float64
 2   number_project         14999 non-null  int64
 3   average_montly_hours   14999 non-null  int64
 4   time_spend_company     14999 non-null  int64
 5   Work_accident          14999 non-null  int64
 6   left                   14999 non-null  int64
 7   promotion_last_5years  14999 non-null  int64
 8   Departments            14999 non-null  object
 9   salary                 14999 non-null  object
dtypes: float64(2), int64(6), object(2)
memory usage: 1.1+ MB
```

```
isnull() and sum():
satisfaction_level       0
last_evaluation          0
number_project           0
average_montly_hours     0
time_spend_company       0
Work_accident            0
left                     0
promotion_last_5years    0
Departments              0
salary                   0
dtype: int64
```

```
data value counts():
0    11428
1     3571
Name: left, dtype: int64
```

data.head() for Salary:

	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	Work_accident	left	promotion_last_5years	Departments	salary
0	0.38	0.53	2	157	3	0	1	0	sales	1
1	0.80	0.86	5	262	6	0	1	0	sales	2
2	0.11	0.88	7	272	4	0	1	0	sales	2
3	0.72	0.87	5	223	5	0	1	0	sales	1
4	0.37	0.52	2	159	3	0	1	0	sales	1

x.head():

	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	Work_accident	promotion_last_5years	salary
0	0.38	0.53	2	157	3	0	0	1
1	0.80	0.86	5	262	6	0	0	2
2	0.11	0.88	7	272	4	0	0	2
3	0.72	0.87	5	223	5	0	0	1
4	0.37	0.52	2	159	3	0	0	1

```
Accuracy value:
0.9846666666666667
```

```
Data Prediction:
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but DecisionTreeClassifier was fitted with feature names
  warnings.warn(
array([0])
```

Result:

Thus the program to implement the Decision Tree Classifier Model for Predicting Employee Churn is written and verified using python programming.

Implementation-of-Decision-Tree-Regressor-Model-for-Predicting-the-Salary-of-the-Employee

AIM:

To write a program to implement the Decision Tree Regressor Model for Predicting the Salary of the Employee.

Equipments Required:

1. Hardware – PCs
2. Anaconda – Python 3.7 Installation / Jupyter notebook

Algorithm

1. Import standard libraries in python for finding Decision tree regressor model for predicting the salary of the employee.
2. Initialize and print the Data.head(),data.info(),data.isnull().sum()
3. Visualize data value count.
4. Import sklearn from LabelEncoder.
5. Split data into training and testing.
6. Calculate the MSE Value,r2 Value and data prediction by importing the required modules from sklearn

Program:

```
Program to implement the Decision Tree Regressor Model for Predicting the Salary of the Employee.
Developed by: DHANUSH.G.R.
RegisterNumber:  212221040038

import pandas as pd
data=pd.read_csv("/content/Salary.csv")

print("Data.head():")

data.head()
```

```
data.isnull().sum()

from sklearn.preprocessing import LabelEncoder

le=LabelEncoder()
print("data.head() for Salary:")
data["Position"]=le.fit_transform(data["Position"])

print("data.head() for Salary:")

data.head()

print("Data.info():")
data.info()
print("Data.isnull() and Sum():")

data.isnull().sum()

x=data[["Position","Level"]]

y=data[["Salary"]]


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=2)

from sklearn.tree import DecisionTreeRegressor

dt=DecisionTreeRegressor()

dt.fit(x_train,y_train)

y_pred=dt.predict(x_test)

print("MSE Value:")


from sklearn import metrics

mse=metrics.mean_squared_error(y_test,y_pred)

mse

r2=metrics.r2_score(y_test,y_pred)
print("r2 Value:")

r2

print("data prediction:")
```

```
dt.predict([[5,6]])
```

Output:

Data.head():

	Position	Level	Salary
0	0	1	45000
1	4	2	50000
2	8	3	60000
3	5	4	80000
4	3	5	110000

```
Data.info():
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 3 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Position  10 non-null     int64
 1   Level     10 non-null     int64
 2   Salary    10 non-null     int64
dtypes: int64(3)
memory usage: 368.0 bytes
```

```
Data.isnull() and Sum():
Position    0
Level       0
Salary      0
dtype: int64
```

data.head() for Salary:

	Position	Level	Salary
0	0	1	45000
1	4	2	50000
2	8	3	60000
3	5	4	80000
4	3	5	110000

```
MSE Value:
462500000.0
```

```
r2 Value:
0.486111111111111116
```

```
data prediction:
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but DecisionTreeRegressor was fitted with feature names
  warnings.warn(
array([150000.])
```

Result:

Thus the program to implement the Decision Tree Regressor Model for Predicting the Salary of the Employee is written and verified using python programming.

Implementation-of-K-Means-Clustering-for-Customer-Segmentation

AIM:

To write a program to implement the K Means Clustering for Customer Segmentation.

Equipments Required:

1. Hardware – PCs
2. Anaconda – Python 3.7 Installation / Jupyter notebook

Algorithm

1. Import standard libraries in python for finding Implementation-of-K-Means-Clustering-for-Customer-Segmentation.
2. Initialize and print the Data.head(),data.info(),data.isnull().sum()
3. Import sklearn.cluster import KMeans
4. calculate the value of KMeans Clusters.
5. plot the graph from Elbow method and find y_pred values .
6. plot the graph from Customer Segments Graph.

Program:

```
Program to implement the K Means Clustering for Customer Segmentation.
Developed by: DHANUSH.G.R.
RegisterNumber:  212221040038

import pandas as pd
import matplotlib.pyplot as plt
data = pd.read_csv("/content/Mall_Customers (1).csv")

print("data.head():")
data.head()

print("data.info():")
data.info()

print("data.isnull().sum():")
data.isnull().sum()

from sklearn.cluster import KMeans
wcss = []
```

```
for i in range (1,11):
    kmeans = KMeans(n_clusters = i,init = "k-means++")
    kmeans.fit(data.iloc[:,3:])
    wcss.append(kmeans.inertia_)

print("Elbow Method Graph:")
plt.plot(range(1,11),wcss)
plt.xlabel("No. of Clusters")
plt.ylabel("wcss")
plt.title("Elbow Method")

print("KMeans cluster value:")
km = KMeans(n_clusters = 5)
km.fit(data.iloc[:,3:])

print("y_pred:")
y_pred = km.predict(data.iloc[:,3:])
y_pred

print("Customer Segments Graph:")
data["cluster"] = y_pred
df0=data[data["cluster"]==0]
df1=data[data["cluster"]==1]
df2=data[data["cluster"]==2]
df3=data[data["cluster"]==3]
df4=data[data["cluster"]==4]
plt.scatter(df0["Annual Income (k$)"],df0["Spending Score (1-
100)"],c="red",label="cluster0")
plt.scatter(df1["Annual Income (k$)"],df1["Spending Score (1-
100)"],c="black",label="cluster1")
plt.scatter(df2["Annual Income (k$)"],df2["Spending Score (1-
100)"],c="blue",label="cluster2")
plt.scatter(df3["Annual Income (k$)"],df3["Spending Score (1-
100)"],c="green",label="cluster3")
plt.scatter(df4["Annual Income (k$)"],df4["Spending Score (1-
100)"],c="yellow",label="cluster4")
plt.legend()
plt.title("Customer Segments")
```

Output:

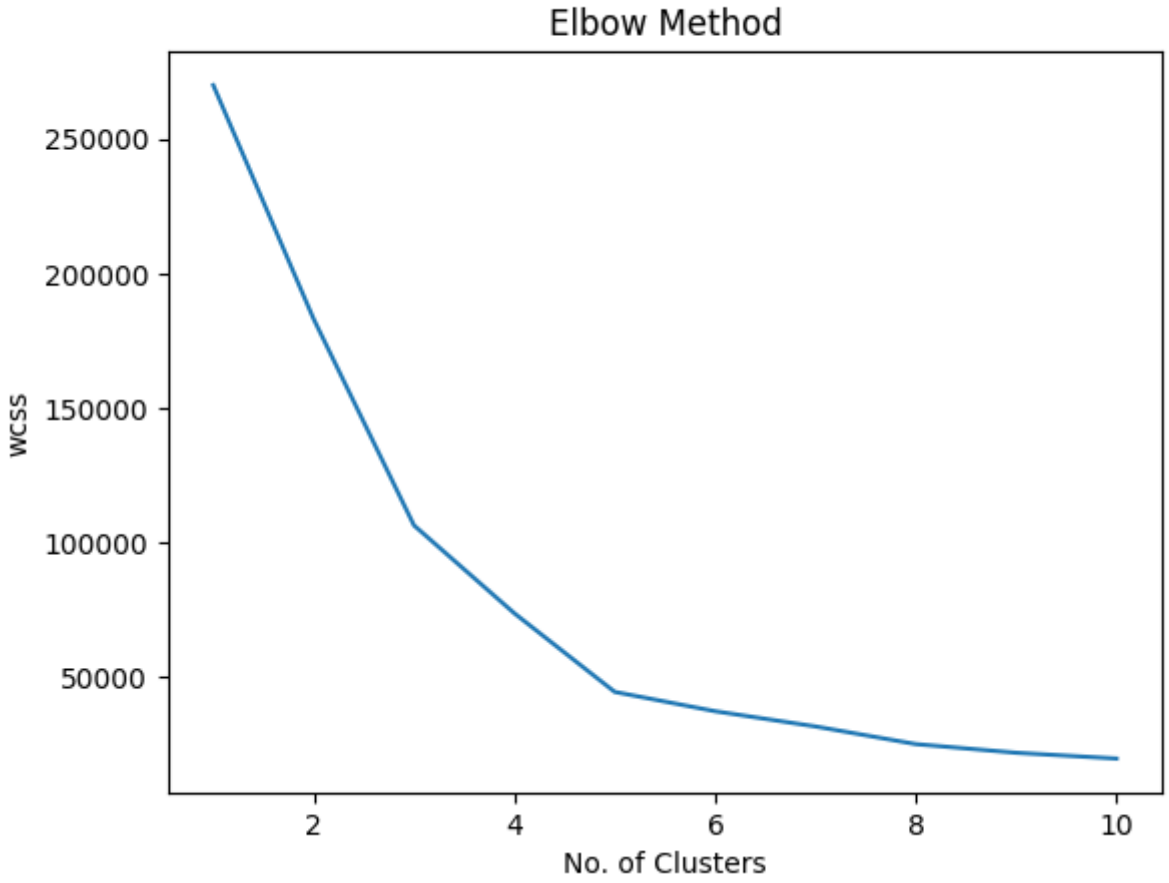
data.head():

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40

```
data.info():
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 5 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   CustomerID              200 non-null    int64
 1   Gender                  200 non-null    object
 2   Age                     200 non-null    int64
 3   Annual Income (k$)      200 non-null    int64
 4   Spending Score (1-100)  200 non-null    int64
dtypes: int64(4), object(1)
memory usage: 7.9+ KB
```

```
data.isnull().sum():
CustomerID                0
Gender                    0
Age                       0
Annual Income (k$)        0
Spending Score (1-100)    0
dtype: int64
```

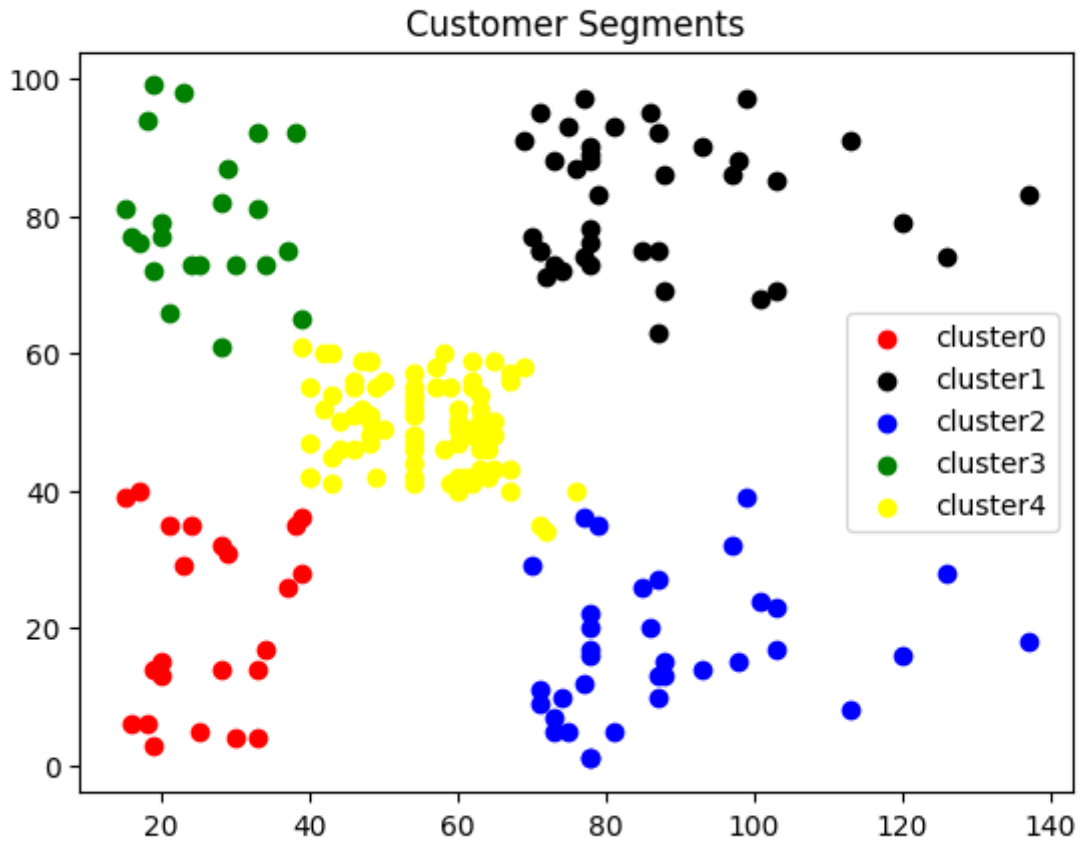
```
Elbow Method Graph:
Text(0.5, 1.0, 'Elbow Method')
```




```
KMeans
KMeans(n_clusters=5)
```

```
y_pred:
array([0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3,
       0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 4,
       0, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
       4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
       4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
       4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 1, 2, 1, 4, 1, 2, 1, 2, 1,
       4, 1, 2, 1, 2, 1, 2, 1, 2, 1, 4, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1,
       2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1,
       2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1,
       2, 1], dtype=int32)
```

```
Customer Segments Graph:
Text(0.5, 1.0, 'Customer Segments')
```



Result:

Thus the program to implement the K Means Clustering for Customer Segmentation is written and verified using python programming.

Implementation-of-SVM-For-Spam-Mail-Detection

AIM:

To write a program to implement the SVM For Spam Mail Detection.

Equipments Required:

1. Hardware – PCs
2. Anaconda – Python 3.7 Installation / Jupyter notebook

Algorithm:

1. Import the necessary packages.
2. Read the given csv file and display the few contents of the data.
3. Assign the features for x and y respectively.
4. Split the x and y sets into train and test sets.
5. Convert the Alphabetical data to numeric using CountVectorizer.
6. Predict the number of spam in the data using SVC (C-Support Vector Classification) method of SVM (Support vector machine) in sklearn library.
7. Find the accuracy of the model.

Program:

```
/*
Program to implement the SVM For Spam Mail Detection..
Developed by: DHANUSH GR
RegisterNumber: 212221040038
/*



import chardet
file='/content/spam.csv'
with open(file, 'rb') as rawdata:
  result = chardet.detect(rawdata.read(100000))
result

import pandas as pd
data=pd.read_csv("/content/spam.csv",encoding = 'Windows-1252')
```

```
data.head()

data.info()

data.isnull().sum()

x=data["v1"].values

y=data["v2"].values

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)

from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer()

x_train=cv.fit_transform(x_train)
x_test=cv.transform(x_test)

from sklearn.svm import SVC
svc=SVC()
svc.fit(x_train,y_train)

y_pred=svc.predict(x_test)
y_pred

from sklearn import metrics
accuracy=metrics.accuracy_score(y_test,y_pred)
accuracy
```

Output:

```
Result output
{'encoding': 'Windows-1252', 'confidence': 0.7270322499829184, 'language': ''}
```

data.head():

	v1	v2	Unnamed: 2	Unnamed: 3	Unnamed: 4
0	ham	Go until jurong point, crazy.. Available only ...	NaN	NaN	NaN
1	ham	Ok lar... Joking wif u oni...	NaN	NaN	NaN
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	NaN	NaN	NaN
3	ham	U dun say so early hor... U c already then say...	NaN	NaN	NaN
4	ham	Nah I don't think he goes to usf, he lives aro...	NaN	NaN	NaN

```
data.info():
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5572 entries, 0 to 5571
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   v1          5572 non-null   object
 1   v2          5572 non-null   object
 2   Unnamed: 2  50 non-null     object
 3   Unnamed: 3  12 non-null     object
 4   Unnamed: 4  6 non-null      object
dtypes: object(5)
memory usage: 217.8+ KB
```

```
data.isnull().sum():
v1              0
v2              0
Unnamed: 2    5522
Unnamed: 3    5560
Unnamed: 4    5566
dtype: int64
```

```
Y_Prediction value:
array(["Sorry, I'll call later", "Sorry, I'll call later",
       "Sorry, I'll call later", ..., "Sorry, I'll call later",
       "Sorry, I'll call later", "Sorry, I'll call later"], dtype=object)
```

```
 Accuracy value:
 0.003587443946188341
```

Result:

Thus the program to implement the SVM For Spam Mail Detection is written and verified using python programming.