|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 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:

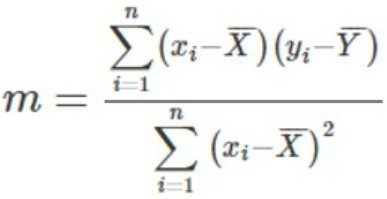
To implement univariate Linear Regression to fit a straight line using least squares.

## 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.



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



1. 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)):

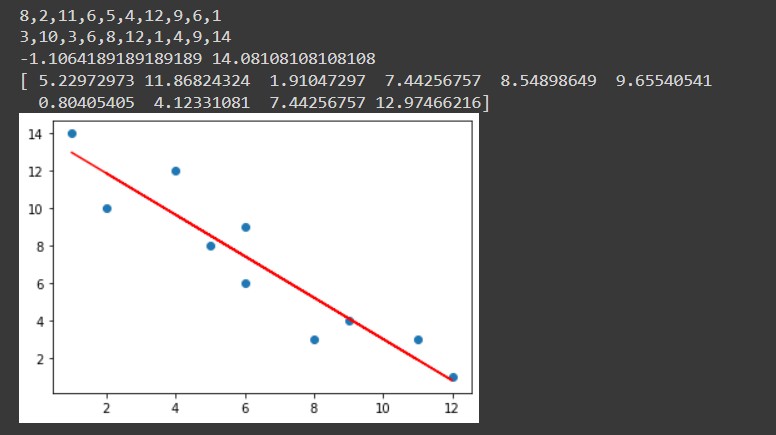
num+=((x[i]-x\_mean)\*(y[i]-y\_mean)) denom+=(x[i]-x\_mean)\*\*2

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:



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")

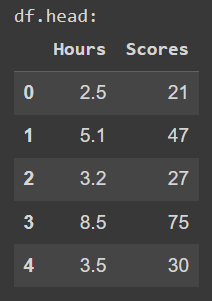
plt.plot(X\_test,regressor.predict(X\_test),color="pink") plt.title("Hours vs Scores (Training Set)") plt.xlabel("Hours")

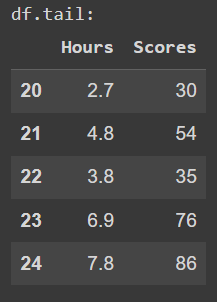
plt.ylabel("Scores") plt.show()

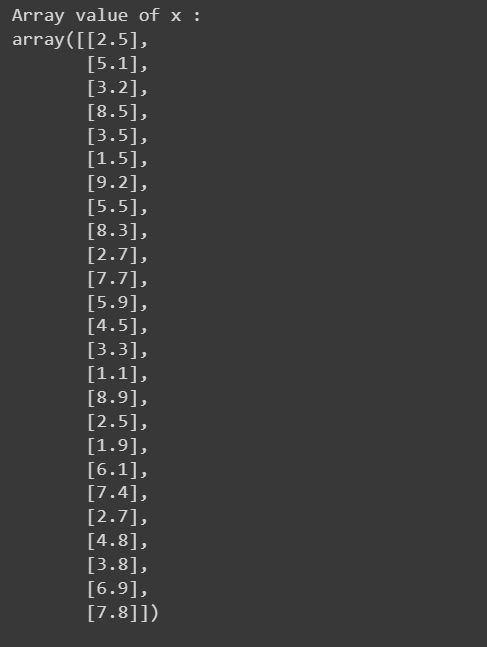
print('Values of MSE,MAE,RMSE:') mse=mean\_squared\_error(Y\_test,Y\_pred) print('MSE = ',mse) mae=mean\_absolute\_error(Y\_test,Y\_pred) print('MAE = ',mae)

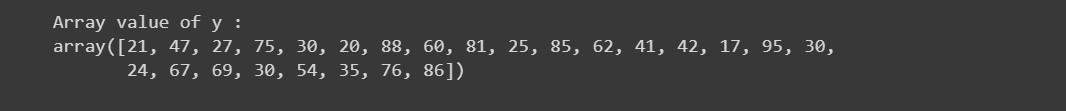
rmse=np.sqrt(mse) print("RMSE = " ,rmse)

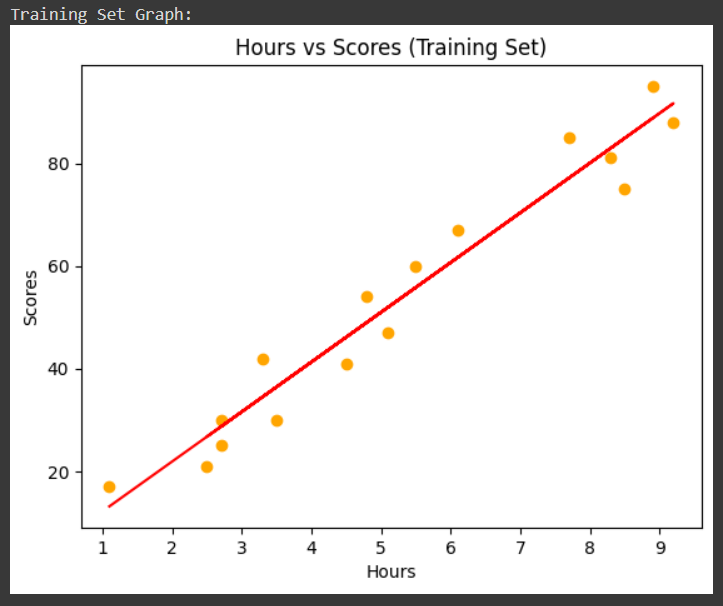
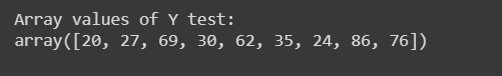
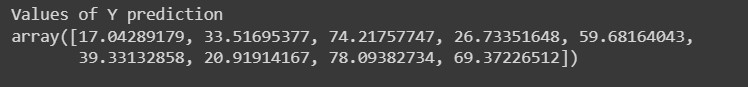
## Output:

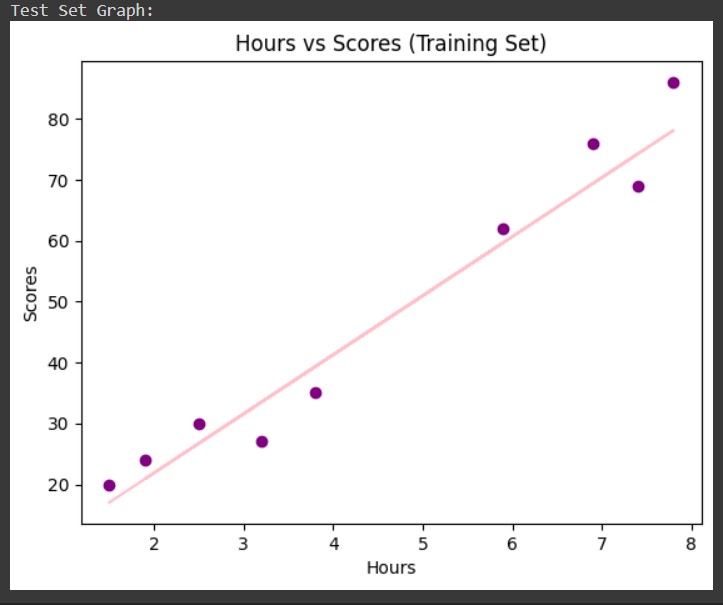


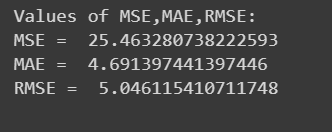












Result:

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):

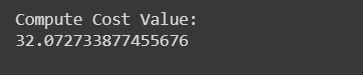
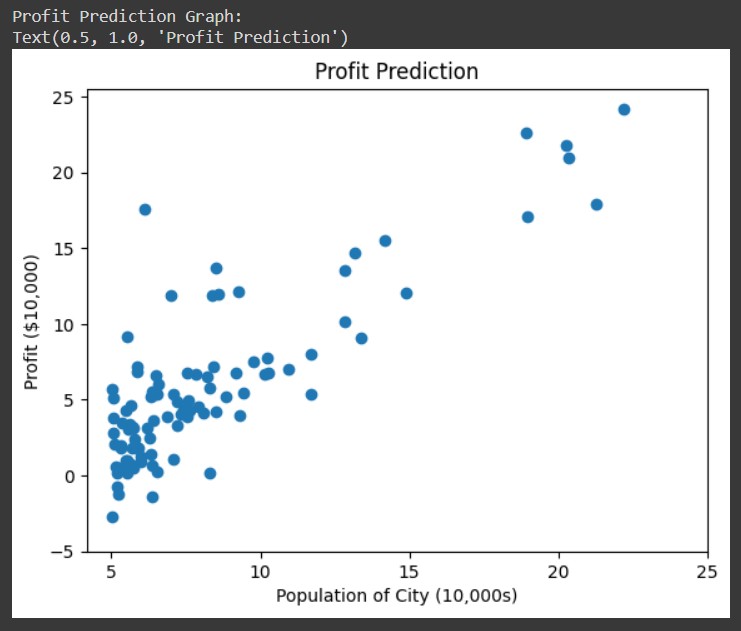
predictions=np.dot(theta.transpose(),x) return predictions[0]

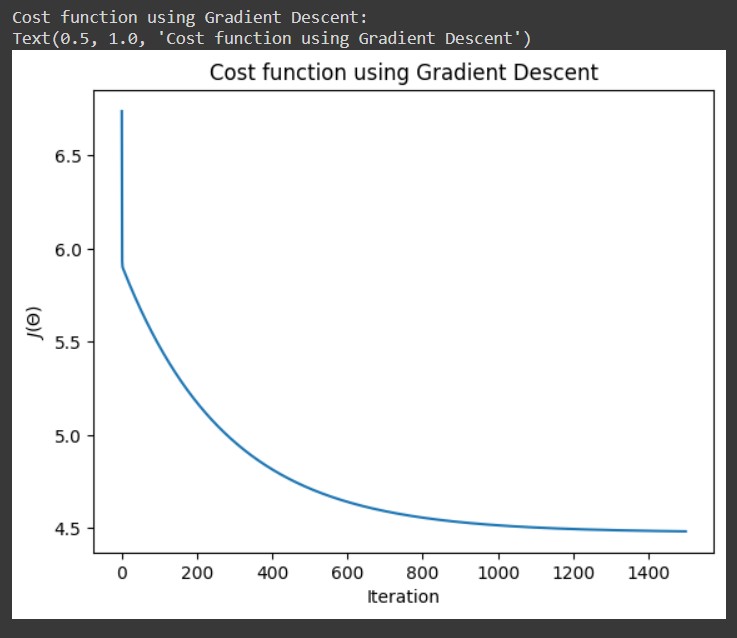
print("Profit for the Population 35,000:") predict1=predict(np.array([1,3.5]),theta)\*1000

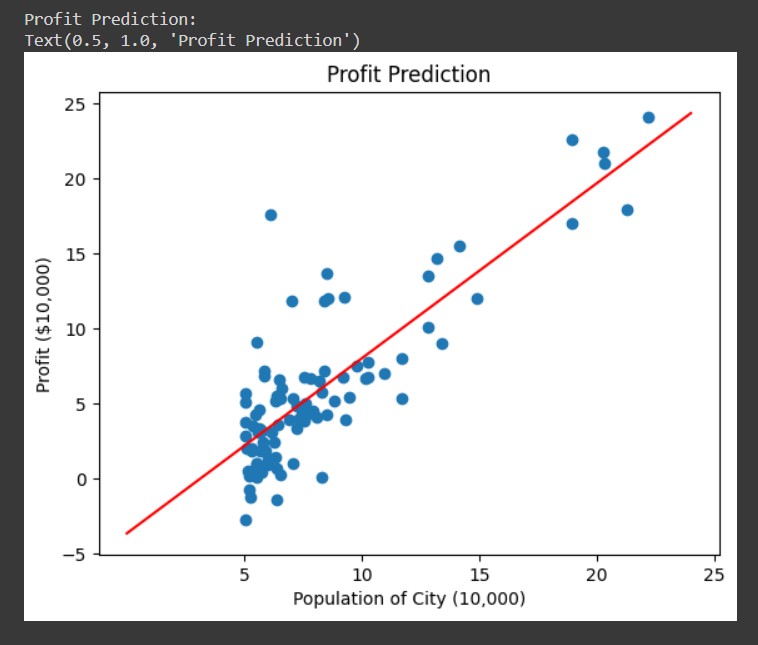
print("For population = 35,000 we predict a profit of $"+str(round(predict1,0))) print("Profit for the Population 70,000:") predict2=predict(np.array([1,7]),theta)\*1000

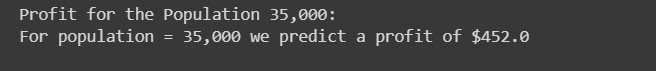
print("For population = 70,000 we predict a profit of $"+str(round(predict2,0)))

## Output:









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.
3. Split data into train and test data.
4. Import logistic regression and fit the training data. 6.Predict y value.
5. 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

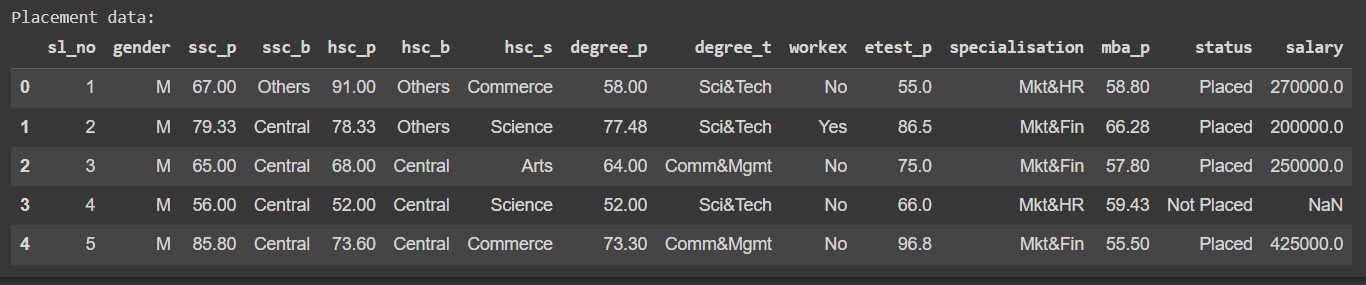
from sklearn.metrics import confusion\_matrix confusion=(y\_test,y\_pred)

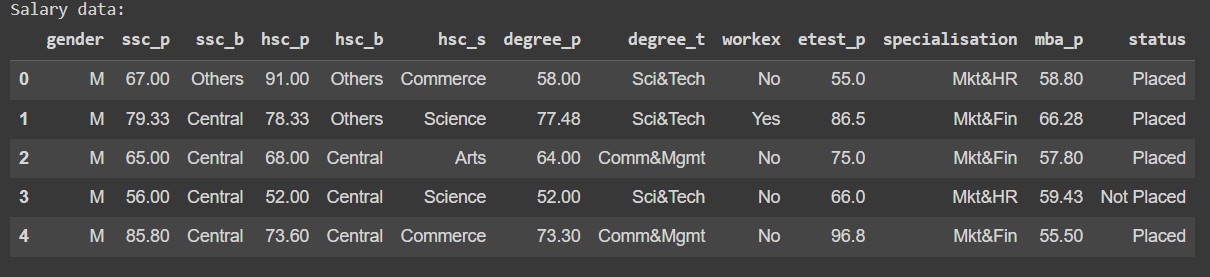
print("Confusion array:") confusion

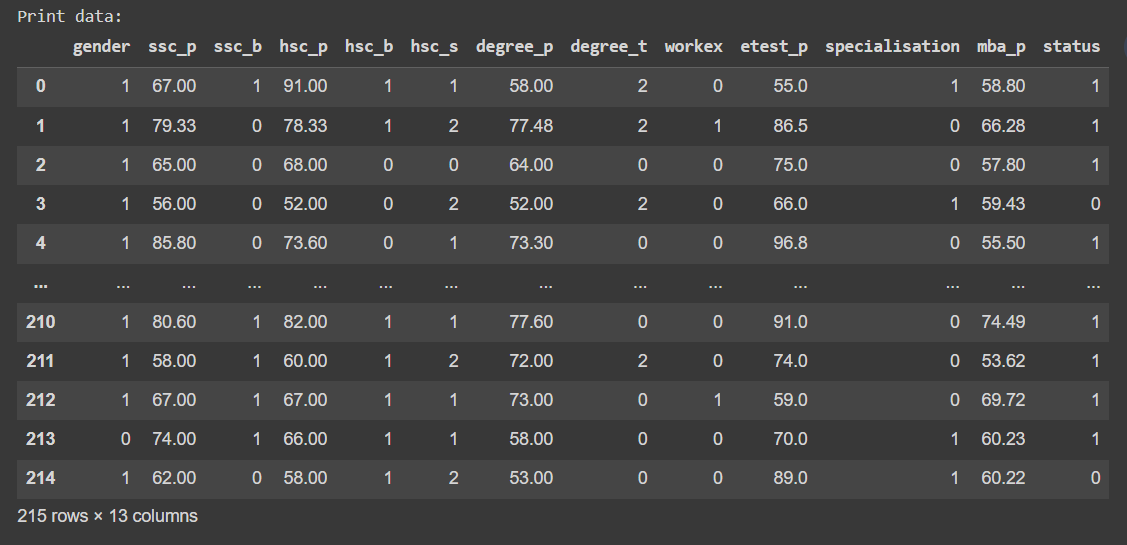
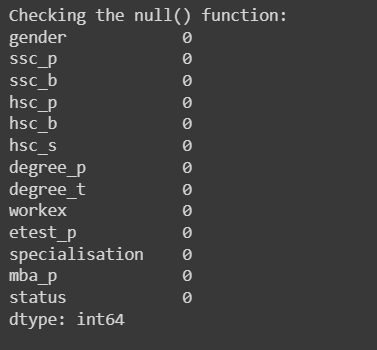
from sklearn.metrics import classification\_report classification\_report1=classification\_report(y\_test,y\_pred) print("Classification report:")

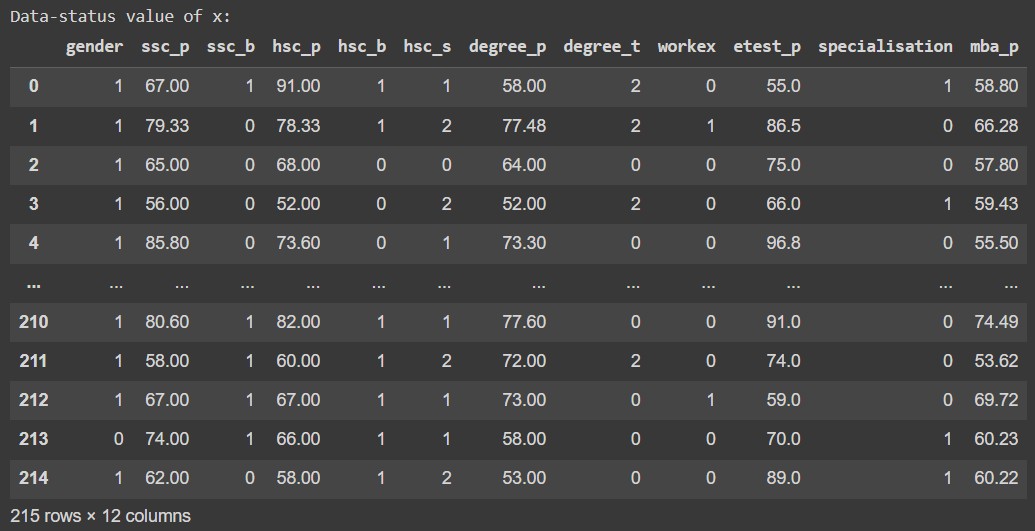
print(classification\_report1) print("Prediction of LR:") lr.predict([[1,80,1,90,1,1,90,1,0,85,1,85]])

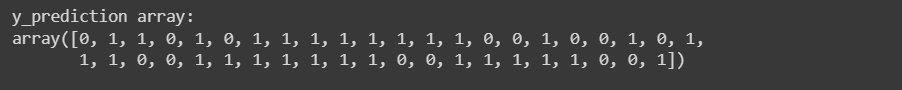
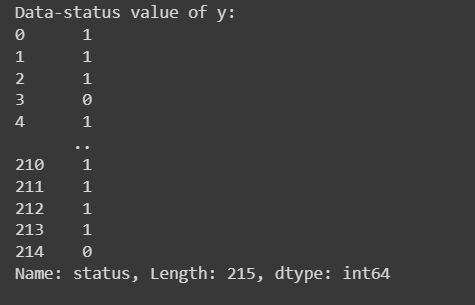
## Output:

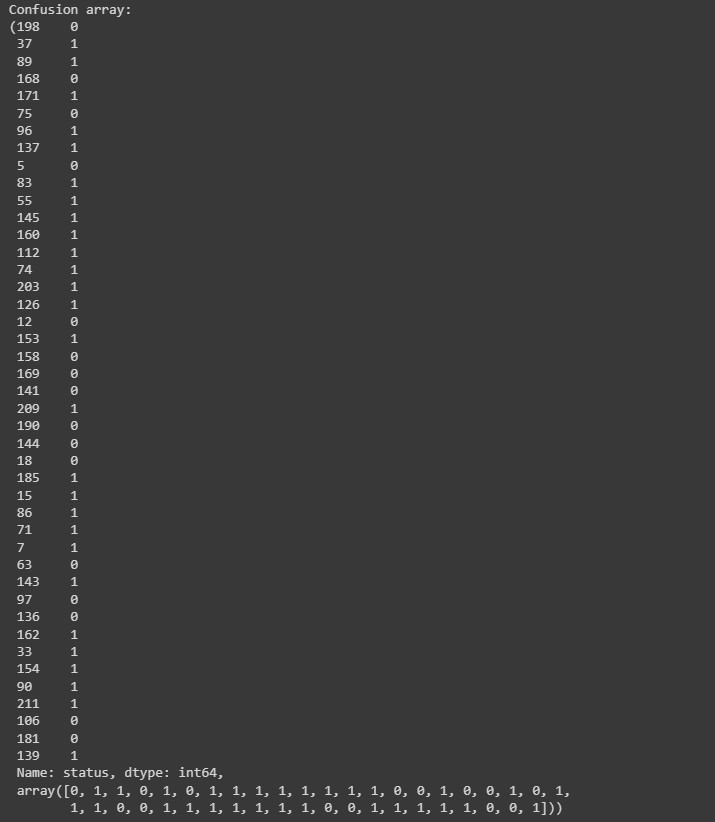


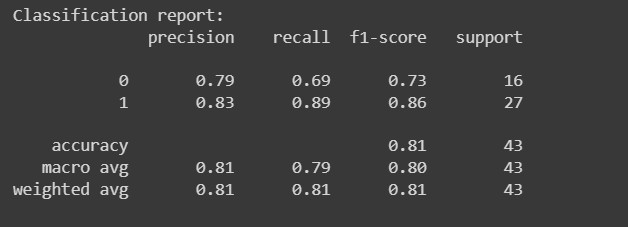


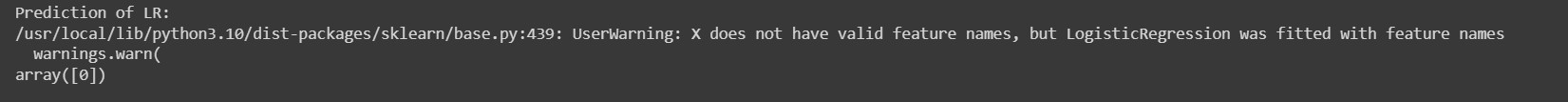












Result:

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.

* 1. Import linear regression from sklearn. 4.Predict the values of array.

1. 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)

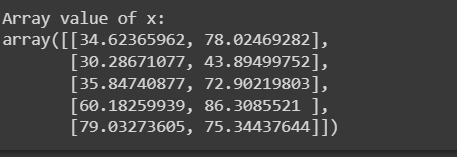
print("probability value:") prob=sigmoid(np.dot(np.array([1,45,85]),res.x)) print(prob)

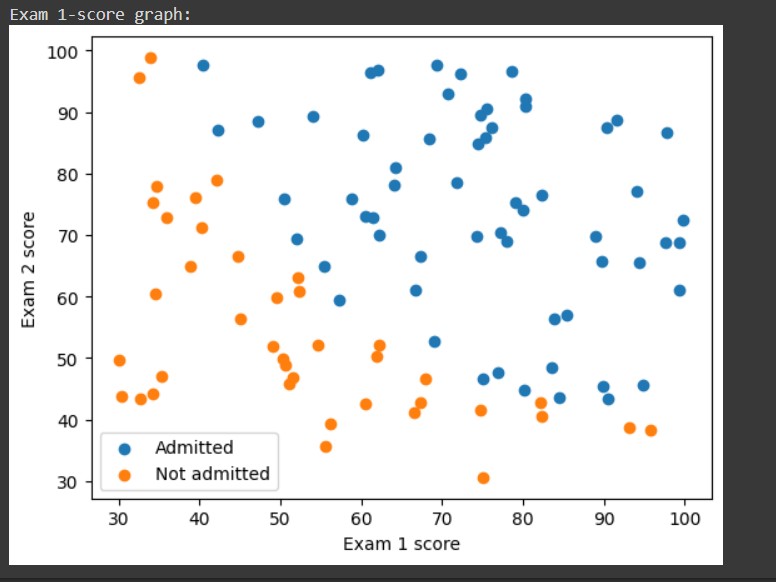
def predict(theta, x):

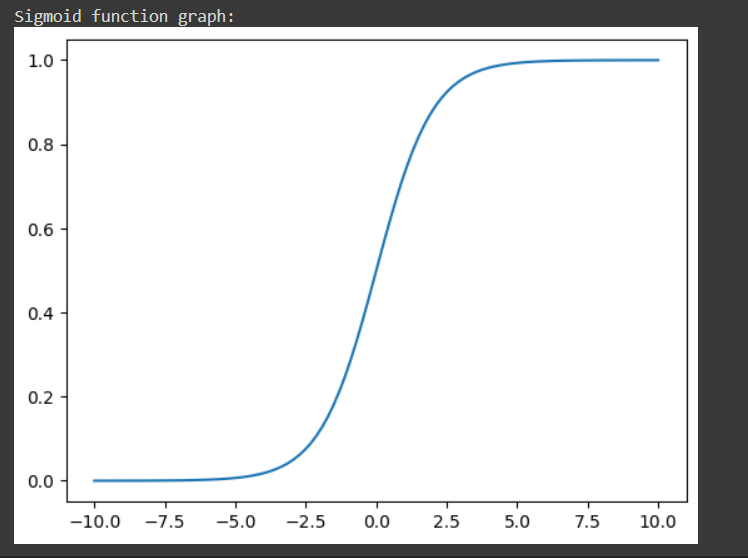
x\_train = np.hstack((np.ones((x.shape[0], 1)), x)) prob = sigmoid(np.dot(x\_train, theta))

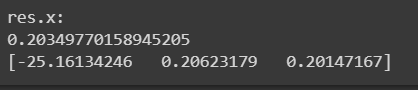
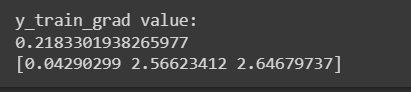
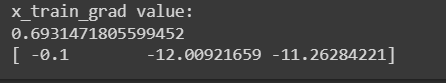
return (prob >= 0.5).astype(int) print("Prediction value of mean:") np.mean(predict(res.x, x) == y)

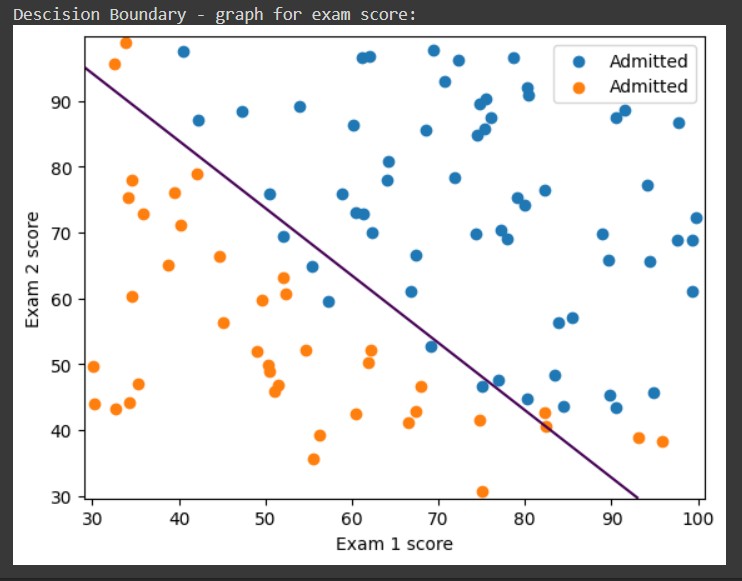
## Output:

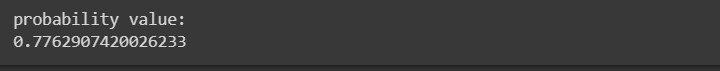












Result:

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()

1. Visualize data value count.
2. Import sklearn from LabelEncoder.
3. Split data into training and testing.
4. 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

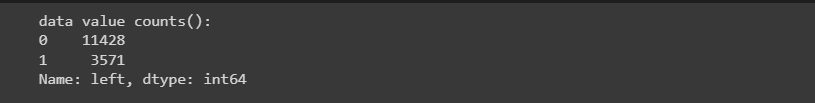
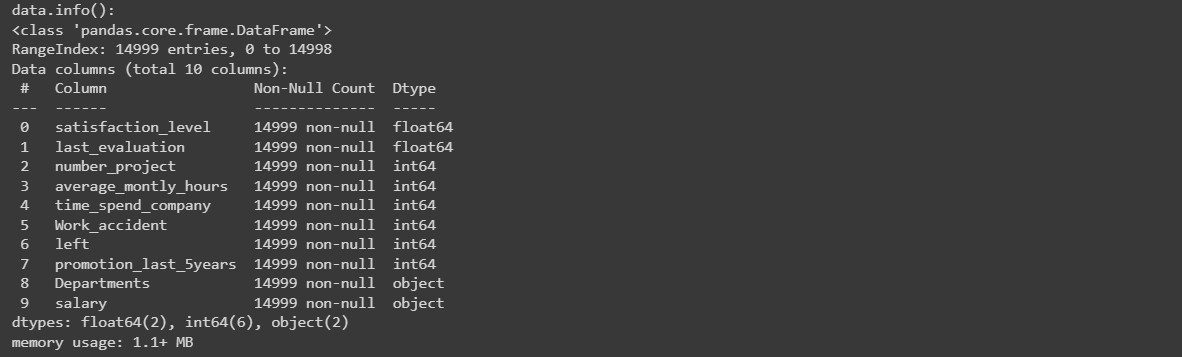
x\_train,x\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=0.2,random\_state=100) from sklearn.tree import DecisionTreeClassifier dt=DecisionTreeClassifier(criterion="entropy")

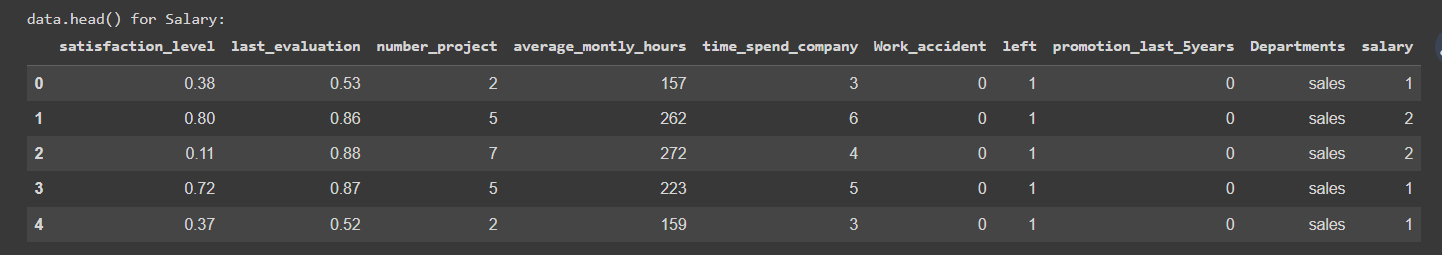
dt.fit(x\_train,y\_train) y\_pred=dt.predict(x\_test) print("Accuracy value:") from sklearn import metrics

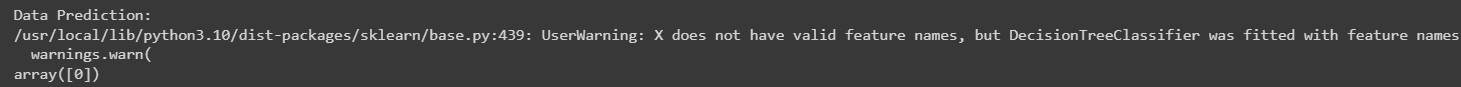
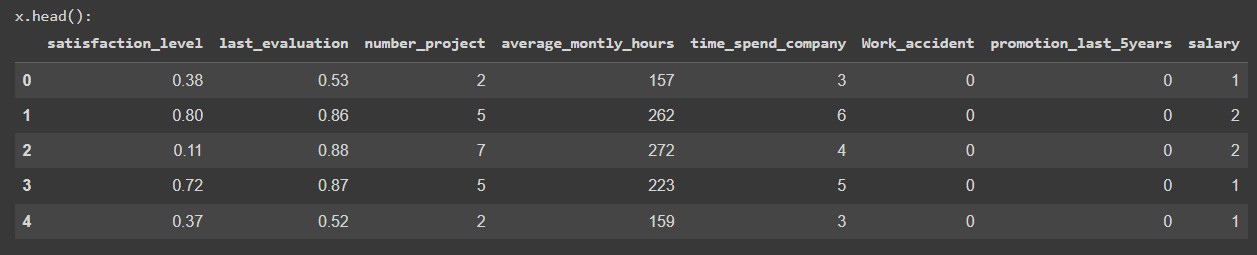
accuracy=metrics.accuracy\_score(y\_test,y\_pred) accuracy

print("Data Prediction:") dt.predict([[0.5,0.8,9,260,6,0,1,2]])

Output:







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

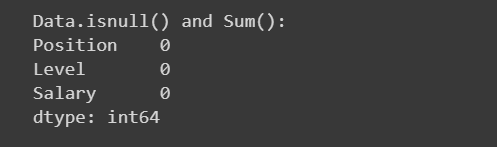
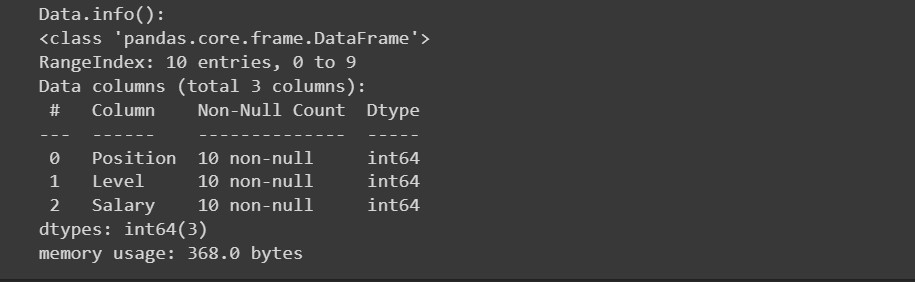
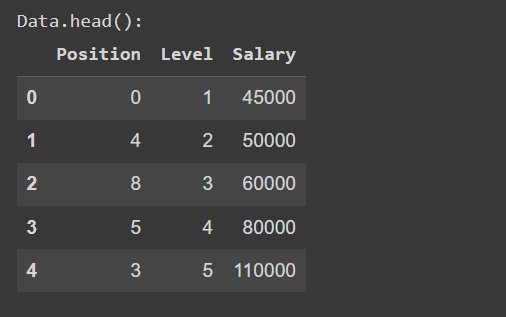
r2=metrics.r2\_score(y\_test,y\_pred) print("r2 Value:")

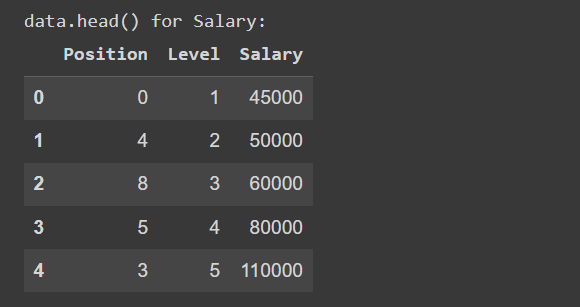
r2

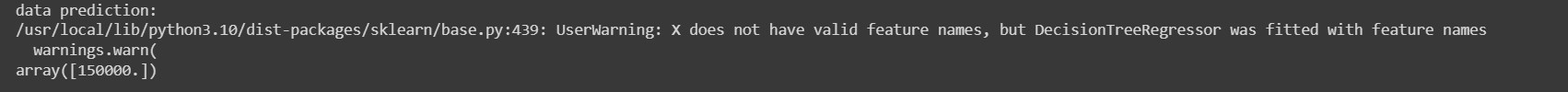
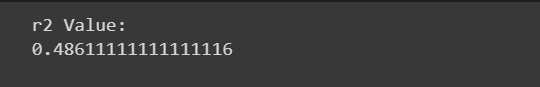
print("data prediction:")

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

## Output:







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()

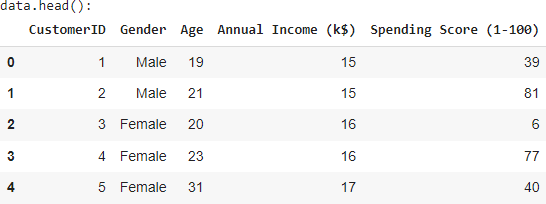
from sklearn.cluster import KMeans wcss = []

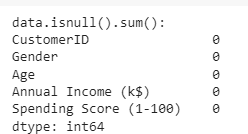
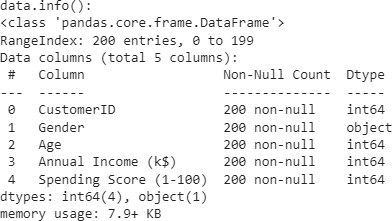
for i in range (1,11):

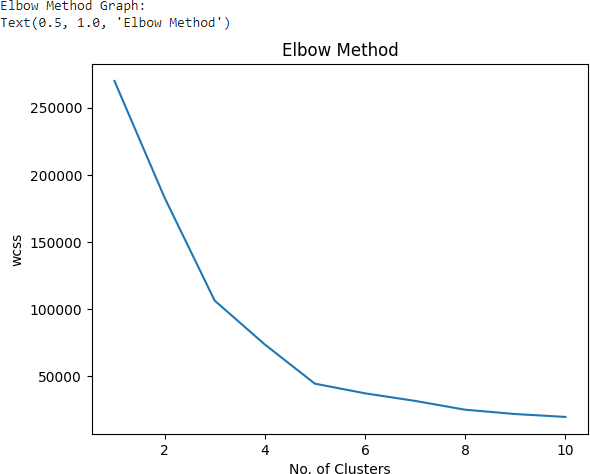
kmeans = KMeans(n\_clusters = i,init = "k-means++") kmeans.fit(data.iloc[:,3:]) wcss.append(kmeans.inertia\_)

|  |  |  |
| --- | --- | --- |
| 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  100)"],c="yellow",label="cluster4") | Score | (1- |
| plt.legend() |  |  |
| plt.title("Customer Segments") |  |  |

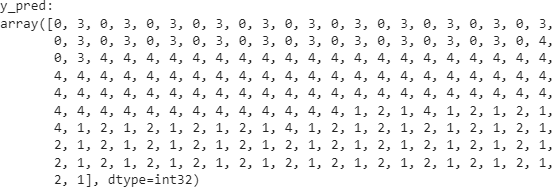
## Output:

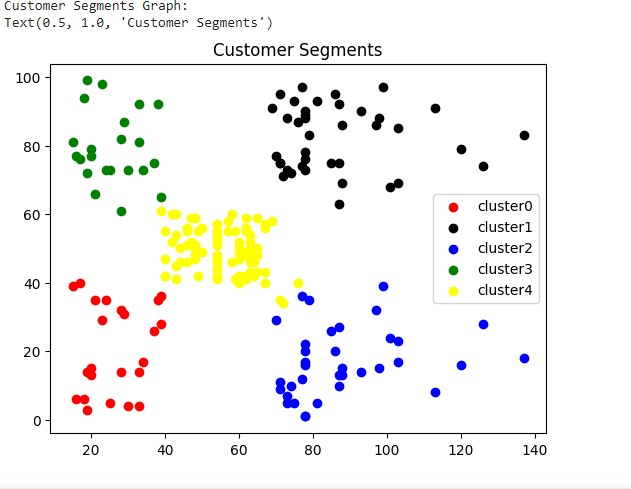












Result:

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()

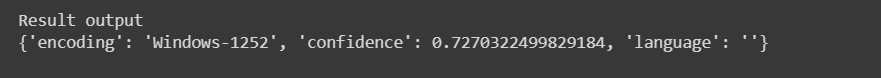
x\_train=cv.fit\_transform(x\_train) x\_test=cv.transform(x\_test)

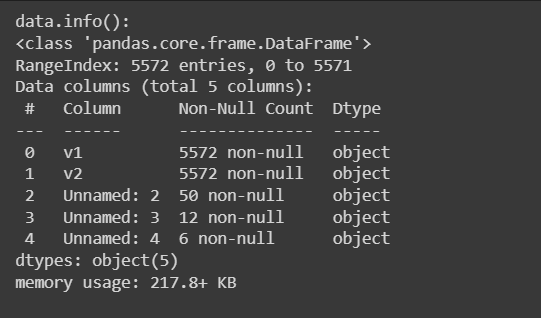
from sklearn.svm import SVC svc=SVC() svc.fit(x\_train,y\_train)

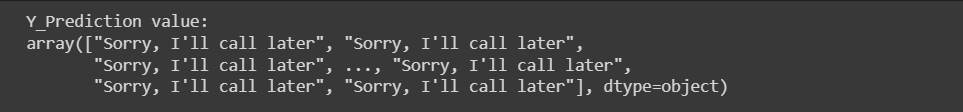
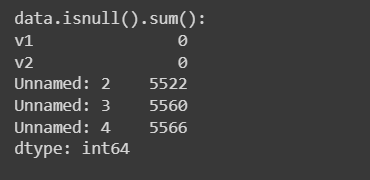
y\_pred=svc.predict(x\_test) y\_pred

from sklearn import metrics accuracy=metrics.accuracy\_score(y\_test,y\_pred) accuracy

### Output:







Result:

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