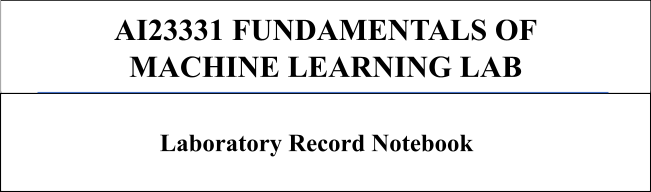
RAJALAKSHMI ENGINEERING COLLEGE

# RAJALAKSHMI NAGAR,THANDALAM–602105



NAME: LOKAA.V

ROLL NO.: 2116231501085

CLASS & SEC: AIML-B

SEMESTER: 3

YEAR: 2024-25

**INDEX PAGE**

|  |  |  |
| --- | --- | --- |
| SL.NO | DATE | NAMEOFTHEEXPERIMENT |
| 01 | 23.08.2024 | A PYTHON PROGRAM TO IMPLEMENT UNIVARIATE REGRESSION,BIVARIATE REGRESSION AND MULTIVARIATE REGRESSION. |
| 02 | 30.08.2024 | A PYTHON PROGRAM TO IMPLEMENT SIMPLE LINEAR REGRESSION USING LEAST SQUARE METHOD |
| 03 | 06.09.2024 | A PYTHON PROGRAM TO IMPLEMENT LOGISTIC MODEL |
| 04 | 13.09.2024 | A PYTHON PROGRAM TO IMPLEMENT SINGLE LAYER PERCEPTRON |
| 05 | 20.09.2024 | A PYTHON PROGRAM TO IMPLEMENT MULTILAYER PERCEPTRON WITH BACKPROPAGATION |
| 06 | 27.09.2024 | A PYTHON PROGRAM TO DO FACE RECOGNITION USING SVM CLASSIFIER |
| 07 | 04.10.2024 | A PYTHON PROGRAM TO IMPLEMENT DECISIONTREE |
| 08 | 18.10.2024 | A PYTHON PROGRAM TO IMPLEMENT DECISION TREE |
| 09A | 25.10.2024 | A PYTHON PROGRAM TO IMPLEMENT KNN MODEL. |
| 09B | 25.10.2024 | A PYTHON PROGRAM TO IMPLEMENT K-MEANS MODEL |
| 10 | 04.11.2024 | A PYTHON PROGRAM TO IMPLEMENT DIMENSIONALITY REDUCTION -PCA. |

**EXPT NO: 1 Apythonprogramtoimplementunivariateregression DATE: 23.08.2024 bivariate regression and multivariate regression.**

# AIM:

Towriteapythonprogramtoimplementunivariateregression,bivariate regression and multivariate regression.

# PROCEDURE:

Implementingunivariate,bivariate,andmultivariateregressionusingtheIris dataset involve the following steps:

**Step1:ImportNecessaryLibraries**

First,importthelibrariesthatareessentialfordatamanipulation,visualization,and model building.

import numpy as np importpandasaspd

import seaborn as sns

import matplotlib.pyplot as plt

fromsklearn.model\_selectionimporttrain\_test\_split from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

**Step2:LoadtheIrisDataset**

TheIrisdatasetcanbeloadedanddisplaythefirstfewrowsofthedataset.

3 231501082

# Load the Iris dataset

iris = sns.load\_dataset('iris')

# Display the first few rows of the dataset

print(iris.head())

# OUTPUT:

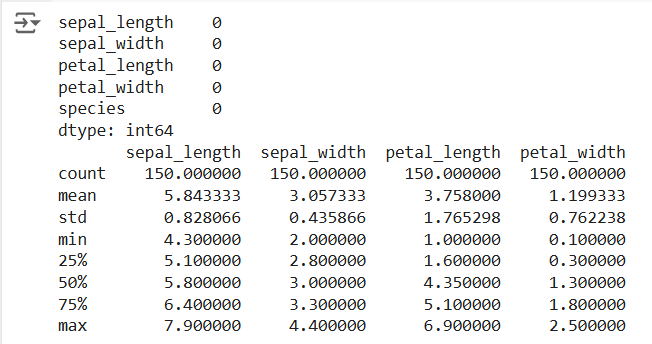
**Step3:DataPreprocessing**

Ensurethedataiscleanandreadyformodeling.SincetheIrisdatasetisclean, minimal preprocessing is needed.

#Checkformissingvalues print(iris.isnull().sum())

#Displaythebasicstatisticaldetails print(iris.describe())

# OUTPUT:



**Step4:UnivariateRegression**

Univariateregressioninvolvespredictingonevariablebasedonasinglepredictor.

* 1. **:SelecttheFeatures**

Chooseonefeature(e.g.,sepal\_length)andonetargetvariable(e.g.,sepal\_width).

X\_uni=iris[['sepal\_length']] y\_uni = iris['sepal\_width']

* 1. **:SplittheData**

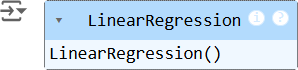
Splitthedataintotrainingandtestingsets.

Fitthelinearregressionmodelonthetrainingdata.

X\_uni\_train,X\_uni\_test,y\_uni\_train,y\_uni\_test=train\_test\_split(X\_uni, y\_uni,

test\_size=0.2, random\_state=42)

* 1. **:Trainthemodel**



uni\_model = LinearRegression() uni\_model.fit(X\_uni\_train,y\_uni\_train)

* 1. **:Make Predictions**

Usethemodeltomakepredictionsonthetestdata.

y\_uni\_pred = uni\_model.predict(X\_uni\_test)

* 1. **:EvaluatetheModel**

EvaluatethemodelperformanceusingmetricslikeMeanSquaredError(MSE) and R-squared.

print(f'UnivariateMSE:{mean\_squared\_error(y\_uni\_test,y\_uni\_pred)}') print(f'Univariate R-squared: {r2\_score(y\_uni\_test, y\_uni\_pred)}')

# OUTPUT:

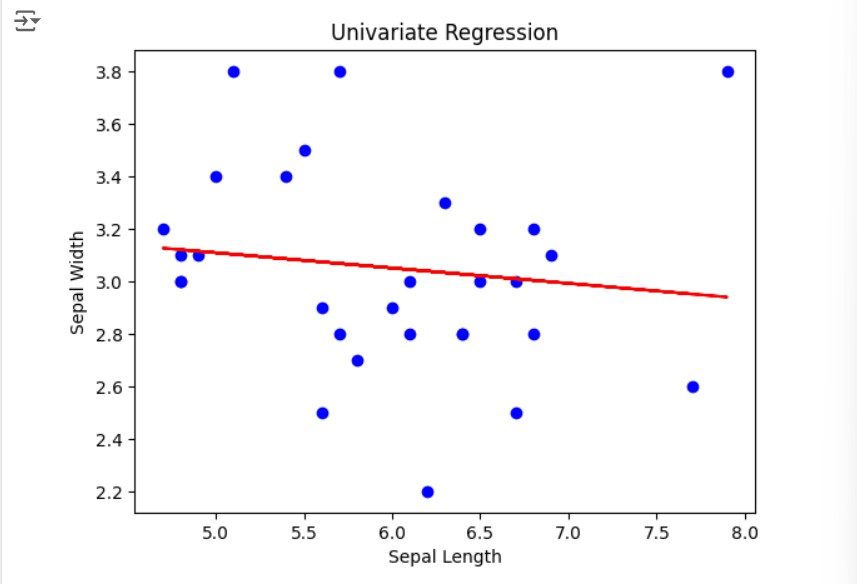
* 1. **:VisualizetheResults**

Visualizetherelationshipbetweenthepredictorandthetargetvariable.

plt.scatter(X\_uni\_test,y\_uni\_test,color='blue') plt.plot(X\_uni\_test, y\_uni\_pred, color='red') plt.xlabel('Sepal Length')

plt.ylabel('Sepal Width') plt.title('UnivariateRegression') plt.show()

# OUTPUT:



**Step5:BivariateRegression**

Bivariateregressioninvolvespredictingonevariablebasedontwopredictors.

* 1. **:SelecttheFeatures**

Choosetwofeatures(e.g.,sepal\_length,petal\_length)andonetargetvariable(e.g., sepal\_width).

X\_bi = iris[['sepal\_length', 'petal\_length']]

y\_bi= iris['sepal\_width']

* 1. **:SplittheData**

Splitthedataintotrainingandtestingsets.

X\_bi\_train, X\_bi\_test, y\_bi\_train, y\_bi\_test = train\_test\_split(X\_bi, y\_bi,

test\_size=0.2, random\_state=42)

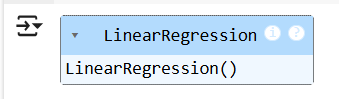
* 1. **:TraintheModel**

Fitthelinearregressionmodelonthetrainingdata.

bi\_model = LinearRegression()

bi\_model.fit(X\_bi\_train, y\_bi\_train)

# OUTPUT:



* 1. **:Make Predictions**

Usethemodeltomakepredictionsonthetestdata.

y\_bi\_pred = bi\_model.predict(X\_bi\_test)

* 1. **:EvaluatetheModel**

EvaluatethemodelperformanceusingmetricslikeMSEandR-squared.

print(f'BivariateMSE:{mean\_squared\_error(y\_bi\_test,y\_bi\_pred)}') print(f'Bivariate R-squared: {r2\_score(y\_bi\_test, y\_bi\_pred)}')

OUTPUT:



* 1. **:VisualizetheResults**

Sincevisualizingin3Dischallenging,wecanplottherelationshipsbetweenthe target and each predictor separately.

#SepalLengthvsSepalWidth plt.subplot(1, 2, 1)

plt.scatter(X\_bi\_test['sepal\_length'],y\_bi\_test,color='blue') plt.plot(X\_bi\_test['sepal\_length'], y\_bi\_pred, color='red') plt.xlabel('Sepal Length')

plt.ylabel('Sepal Width')

#PetalLengthvsSepalWidth plt.subplot(1, 2, 2)

plt.scatter(X\_bi\_test['petal\_length'],y\_bi\_test,color='blue') plt.plot(X\_bi\_test['petal\_length'], y\_bi\_pred, color='red') plt.xlabel('Petal Length')

plt.ylabel('Sepal Width')

plt.suptitle('BivariateRegression') plt.show()

# OUTPUT:

**Step6:MultivariateRegression**

Multivariateregressioninvolvespredictingonevariablebasedonmultiple predictors.

* 1. **:SelecttheFeatures**

Choosemultiplefeatures(e.g.,sepal\_length,petal\_length,petal\_width)andone target variable (e.g., sepal\_width).

X\_multi = iris[['sepal\_length', 'petal\_length', 'petal\_width']]

y\_multi= iris['sepal\_width']

* 1. **:SplittheData**

Splitthedataintotrainingandtestingsets.

X\_multi\_train,X\_multi\_test,y\_multi\_train,y\_multi\_test= train\_test\_split(X\_multi,

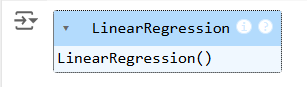
y\_multi, test\_size=0.2, random\_state=42)

* 1. **:TraintheModel**

Fitthelinearregressionmodelonthetrainingdata.

multi\_model = LinearRegression() multi\_model.fit(X\_multi\_train,y\_multi\_train)

# OUTPUT:



* 1. **:Make Predictions**

Usethemodeltomakepredictionsonthetestdata.

y\_multi\_pred = multi\_model.predict(X\_multi\_test)

* 1. **:EvaluatetheModel**

EvaluatethemodelperformanceusingmetricslikeMSEandR-squared.

print(f'MultivariateMSE:{mean\_squared\_error(y\_multi\_test,y\_multi\_pred)}') print(f'Multivariate R-squared: {r2\_score(y\_multi\_test, y\_multi\_pred)}')

# OUTPUT:

**Step7:Visualizethemultivariateregression**

plt.figure(figsize=(15,4)) plt.subplot(1, 2, 1)

plt.scatter(X\_multi\_test['sepal\_length'],y\_multi\_test,color='blue') plt.plot(X\_multi\_test['sepal\_length'], y\_multi\_pred, color='red') plt.xlabel('sepal\_length')

plt.ylabel('sepal\_width')

plt.title('MultivariateRegression-1') plt.show()

plt.figure(figsize=(15,4)) plt.subplot(1, 2, 1)

plt.scatter(X\_multi\_test['petal\_length'],y\_multi\_test,color='blue') plt.plot(X\_multi\_test['petal\_length'], y\_multi\_pred, color='red') plt.xlabel('petal\_length')

plt.ylabel('sepal\_width')

plt.title('MultivariateRegression-2') plt.show()

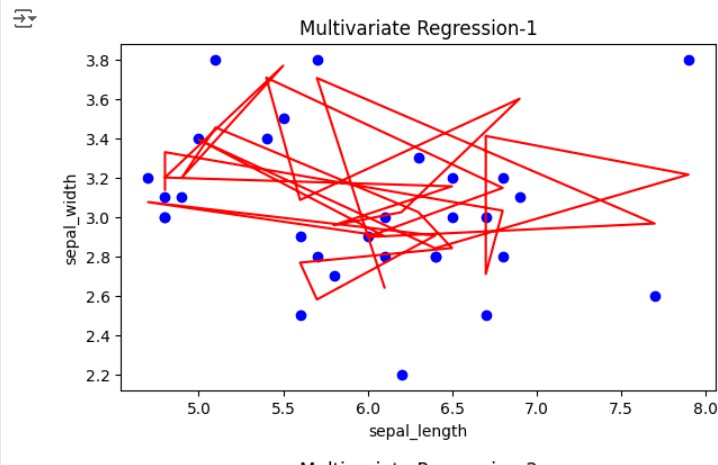
plt.figure(figsize=(15,4)) plt.subplot(1, 2, 2 )

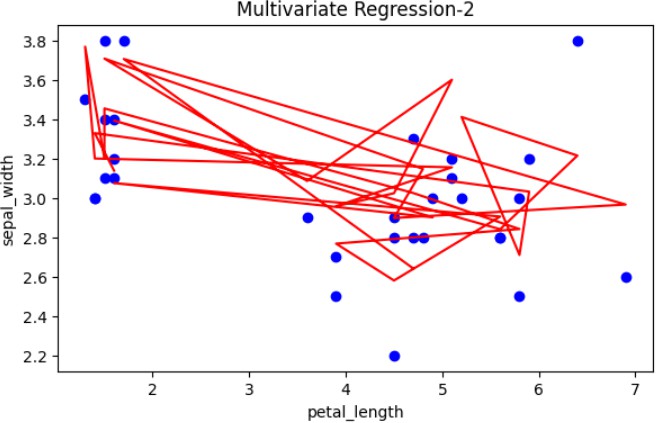
plt.scatter(X\_multi\_test['petal\_length'],y\_multi\_test,color='blue') plt.plot(X\_multi\_test['petal\_length'], y\_multi\_pred, color='red') plt.xlabel('petal\_length')

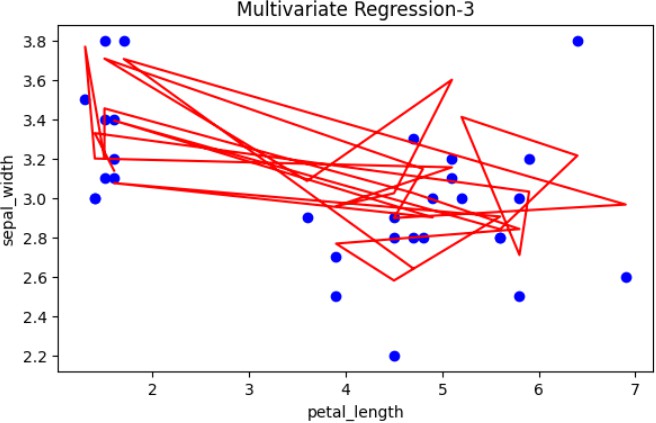
plt.ylabel('sepal\_width')

plt.title('MultivariateRegression-3') plt.show()

# OUTPUT:





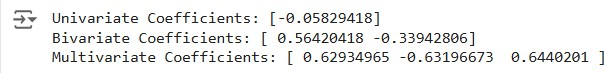


**Step8:InterprettheResults**

Afterimplementingandevaluatingthemodels,interpretthecoefficientsto understand the influence of each predictor on the target variable.

print('UnivariateCoefficients:',uni\_model.coef\_) print('Bivariate Coefficients:', bi\_model.coef\_)

print('Multivariate Coefficients:', multi\_model.coef\_)

**OUTPUT :**

# RESULT:

This step-by-step process will help us to implement univariate, bivariate, and multivariateregressionmodelsusingtheIrisdatasetandanalyzetheirperformance.

**EXPT NO : 2 ApythonprogramtoimplementSimplelinear DATE: 30.08.2024 Regression using Least Square Method**

# AIM:

TowriteapythonprogramtoimplementSimplelinearregressionusingLeast Square Method.

# PROCEDURE:

ImplementingSimplelinearregressionusingLeastSquaremethodusingthe headbrain dataset involve the following steps:

**Step1:ImportNecessaryLibraries**

First,importthelibrariesthatareessentialfordatamanipulation,visualization,and model building.

import pandas as pd

importmatplotlib.pyplotasplt import numpy as np

**Step2:LoadtheIrisDataset**

TheHeadBraindatasetcanbeloaded.

data= pd.read\_csv('/content/headbrain.csv')

**Step3:DataPreprocessing**

Ensurethedataiscleanandreadyformodeling.SincetheIrisdatasetisclean, minimal preprocessing is needed.

x,y=np.array(list(data['HeadSize(cm^3)'])),np.array(list(data['Brain Weight(grams)']))

print(x[:5],y[:5])

# OUTPUT:



**Step4:ComputetheLeastSquaresSolution**

Applytheleastsquaresformulatofindtheregressioncoefficients.

def get\_line(x,y):

x\_m,y\_m=np.mean(x),np.mean(y) print(x\_m,y\_m)

x\_d,y\_d=x-x\_m,y-y\_m

m=np.sum(x\_d\*y\_d)/np.sum(x\_d\*\*2) c = y\_m - (m\*x\_m)

print(m, c)

returnlambdax:m\*x+c lin=get\_line(x,y)

# OUTPUT:



**Step5:Make Predictions**

Usethemodeltomakepredictionsbasedontheindependentvariable.

def get\_error(line\_fuc, x, y):

y\_m = np.mean(y)

y\_pred=np.array([line\_fuc(\_)for\_inx]) ss\_t = np.sum((y-y\_m)\*\*2)

ss\_r=np.sum((y-y\_pred)\*\*2) return 1-(ss\_r/ss\_t)

get\_error(lin, x, y)

fromsklearn.linear\_modelimportLinearRegression x = x.reshape((len(x),1))

reg=LinearRegression() reg=reg.fit(x, y)

print(reg.score(x, y))

# OUTPUT:





**Step6:VisualizetheResults**

Plottheoriginaldatapointsandthefittedregressionline.

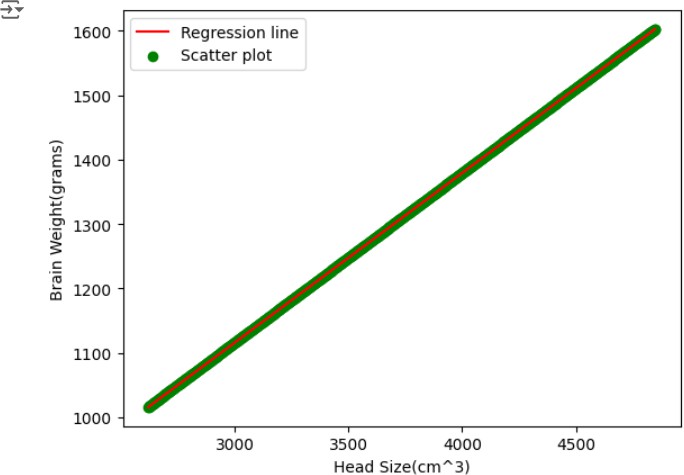
x=np.linspace(np.min(x)-100,np.max(x)+100,1000) y=np.array([lin(x)for x in x])

plt.plot(x, y, color='red', label='Regression line') plt.scatter(x,y,color='green',label='Scatterplot') plt.xlabel('Head Size(cm^3)')

plt.ylabel('BrainWeight(grams)') plt.legend()

plt.show()

# OUTPUT:



**RESULT:**

Thisstep-by-stepprocesswillhelpustoimplementleastsquareregressionmodels using the HeadBrain dataset and analyze their performance.

**EXPT NO : 3 ApythonprogramtoimplementLogisticModel DATE: 06.09.2024**

# AIM:

TowriteapythonprogramtoimplementaLogisticModel.

**PROCEDURE:**

ImplementingLogisticmethodusingtheirisdatasetinvolvethefollowingsteps:

**Step1:ImportNecessaryLibraries**

First,importthelibrariesthatareessentialfordatamanipulation,visualization,and model building.

#Step1:ImportNecessaryLibraries import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

fromsklearn.model\_selectionimporttrain\_test\_split from sklearn.linear\_model import LogisticRegression

fromsklearn.metricsimportaccuracy\_score,confusion\_matrix, classification\_report

**Step2:LoadtheIrisDataset**

Theirisdatasetcanbeloaded.

# Step 2: Load the Dataset

#Forthisexample,we'lluseabuilt-indatasetfromsklearn.Youcan replace it with your dataset.

from sklearn.datasets import load\_iris

#Loadtheirisdataset data = load\_iris()

X = data.data

y = (data.target == 0).astype(int) #Forbinaryclassification(classifying Iris-setosa)

**Step3:DataPreprocessing**

Ensurethedataiscleanandreadyformodeling.SincetheIrisdatasetisclean, minimal preprocessing is needed.

# Step 3: Prepare the Data

# Split the dataset into training and testing sets

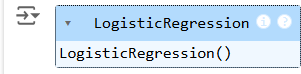
X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,test\_size=0.2, random\_state=42)

**Step4:TrainaModel**

# Step 4: Create and Train the Model

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

# OUTPUT:



**Step5:Make Predictions**

Usethemodeltomakepredictionsbasedontheindependentvariable.

# Step 5: Make Predictions

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

**Step6:EvaluatetheModel**

Evaluatethemodelperformance.

# Step 6: Evaluate the Model

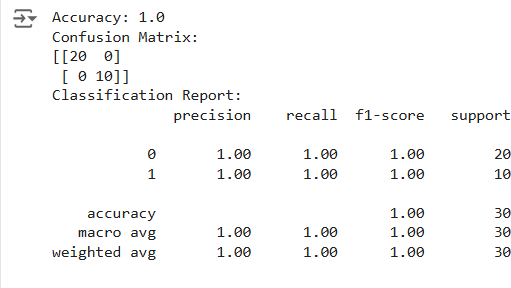
accuracy = accuracy\_score(y\_test, y\_pred) conf\_matrix = confusion\_matrix(y\_test, y\_pred) class\_report=classification\_report(y\_test,y\_pred) # Print evaluation metrics

print(f"Accuracy:{accuracy}") print("Confusion Matrix:")

print(conf\_matrix)

print("ClassificationReport:") print(class\_report)

# OUTPUT:



**Step7:VisualizetheResults**

Plottheoriginaldatapointsandthefittedregressionline.

# Step 7: Visualize Results (Optional)

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

sigmoid\_values = 1 / (1 + np.exp(-x\_values))

#Plotthesigmoidfunction plt.figure(figsize=(10, 5))

plt.plot(x\_values,sigmoid\_values,label='SigmoidFunction',color='blue') plt.title('Sigmoid Function')

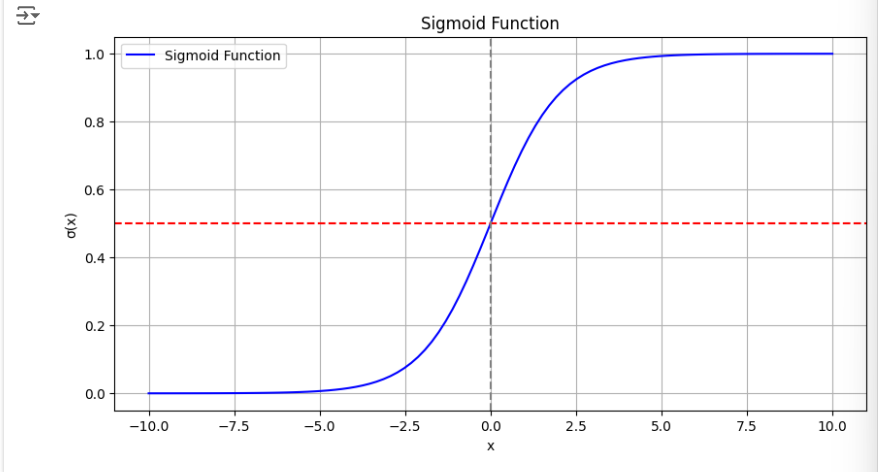
plt.xlabel('x')

plt.ylabel('σ(x)') plt.grid()

plt.axhline(0.5, color='red', linestyle='--') #Lineaty=0.5 plt.axvline(0, color='gray', linestyle='--') # Line at x=0 plt.legend()

plt.show()

# OUTPUT:



**RESULT:**

Thisstep-by-stepprocesswillhelpustoimplementLogisticmodelsusingtheIris dataset and analyze their performance.

**EXPT NO : 4 ApythonprogramtoimplementSingleLayer DATE: 13.09.2024 Perceptron**

# AIM:

TowriteapythonprogramtoimplementSinglelayerperceptron.

# PROCEDURE:

ImplementingSinglelayerperceptronmethodusingtheKerasdatasetinvolvethe following steps:

**Step1:ImportNecessaryLibraries**

First,importthelibrariesthatareessentialfordatamanipulation,visualization,and model building.

import numpy as np importpandasaspd

from tensorflow import keras

import matplotlib.pyplot as plt

**Step2:LoadtheKerasDataset**

TheKerasdatasetcanbeloaded.

(X\_train,y\_train),(X\_test,y\_test)=keras.datasets.mnist.load\_data()

**Step3:DataPreprocessing**

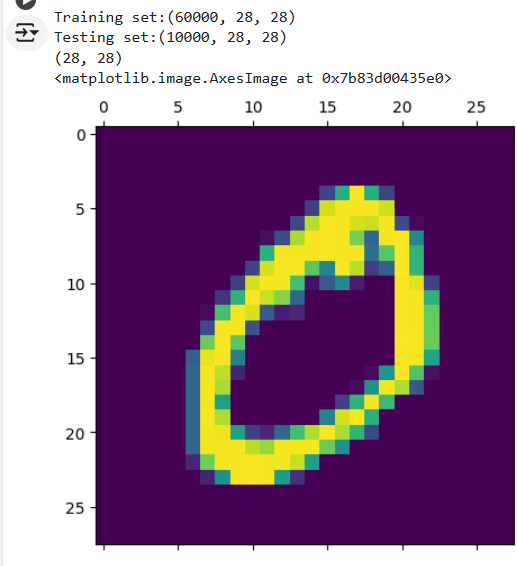
Ensurethedataiscleanandreadyformodeling.SincetheIrisdatasetisclean, minimal preprocessing is needed.

print(f"Trainingset:{X\_train.shape}") print(f"Testing set:{X\_test.shape}")

print(X\_train[1].shape)

plt.matshow(X\_train[1])

# OUTPUT:



**Step4:TrainaModel**

#Normalizingthedataset x\_train=X\_train/255 x\_test=X\_test/255

#Flattingthedatasetinordertocomputeformodelbuilding x\_train\_flatten=x\_train.reshape(len(x\_train),28\*28) x\_test\_flatten=x\_test.reshape(len(x\_test),28\*28) x\_train\_flatten.shape

**Step5:Make Predictions**

Usethemodeltomakepredictionsbasedontheindependentvariable.

model=keras.Sequential([

keras.layers.Dense(10,input\_shape=(784,),

activation='sigmoid')

])

model.compile(

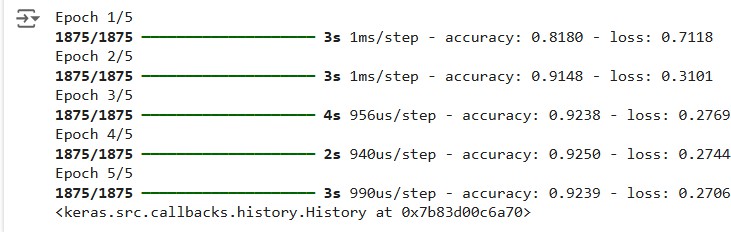
optimizer='adam',

loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

model.fit(x\_train\_flatten,y\_train,epochs=5

)

# OUTPUT:



**Step6:EvaluatetheModel**

Evaluatethemodelperformance.

model.evaluate(x\_test\_flatten,y\_test)

# OUTPUT:



**RESULT:**

Thisstep-by-stepprocesswillhelpustoimplementSingleLayerPerceptronmodels using the Keras dataset and analyze their performance.

**EXPT NO : 5 ApythonprogramtoimplementMultiLayer DATE: 20.09.2024 Perceptron With Backpropagation**

# AIM:

TowriteapythonprogramtoimplementMultilayerperceptronwith backpropagation .

# PROCEDURE:

ImplementingMultilayerperceptronwithbackpropagationusingtheKerasdataset involve the following steps:

**Step1:ImportNecessaryLibraries**

First,importthelibrariesthatareessentialfordatamanipulation,visualization,and model building.

# importing modules

importtensorflowastf import numpy as np

fromtensorflow.keras.modelsimportSequential from tensorflow.keras.layers import Flatten

from tensorflow.keras.layers import Dense

fromtensorflow.keras.layersimportActivation import matplotlib.pyplot as plt

**Step2:LoadtheKerasDataset**

TheKerasdatasetcanbeloaded.

(x\_train, y\_train), (x\_test, y\_test) = tf.keras.datasets.mnist.load\_data()

# OUTPUT:



**Step3:DataPreprocessing**

Ensurethedataiscleanandreadyformodeling.SincetheIrisdatasetisclean, minimal preprocessing is needed.

#Casttherecordsintofloatvalues x\_train = x\_train.astype('float32') x\_test = x\_test.astype('float32')

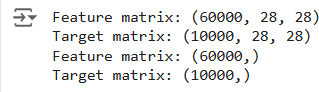
#normalizeimagepixelvaluesbydividing # by 255

gray\_scale = 255 x\_train/=gray\_scale x\_test /= gray\_scale

print("Featurematrix:",x\_train.shape) print("Target matrix:", x\_test.shape)

print("Featurematrix:",y\_train.shape) print("Target matrix:", y\_test.shape)

# OUTPUT:



**Step4:TrainaModel**

model = Sequential([

#reshape28row\*28columndatato28\*28rows Flatten(input\_shape=(28, 28)),

# dense layer 1

Dense(256, activation='sigmoid'),

# dense layer 2

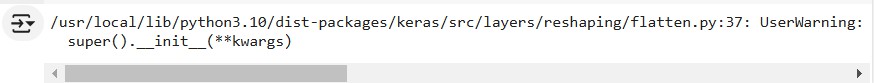
Dense(128, activation='sigmoid'),

# output layer

Dense(10, activation='sigmoid'),

**])**

# OUTPUT:



**Step5:Make Predictions**

Usethemodeltomakepredictionsbasedontheindependentvariable.

model.compile(optimizer='adam',

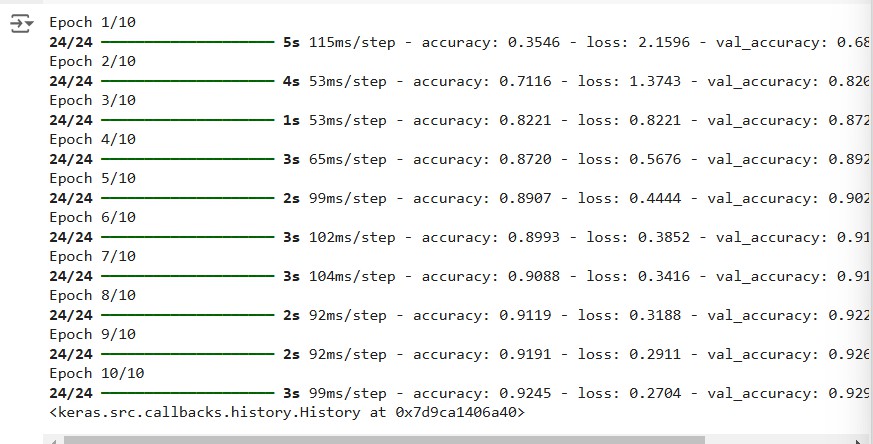
loss='sparse\_categorical\_crossentropy',

metrics=['accuracy']) model.fit(x\_train,y\_train,epochs=10,

batch\_size=2000,

validation\_split=0.2)

# OUTPUT:



**Step6:EvaluatetheModel**

Evaluatethemodelperformance.

results=model.evaluate(x\_test,y\_test,verbose=0) print('test loss, test acc:', results)

fig,ax=plt.subplots(10,10) k = 0

for i in range(10):

for j in range(10): ax[i][j].imshow(x\_train[k].reshape(28,28),

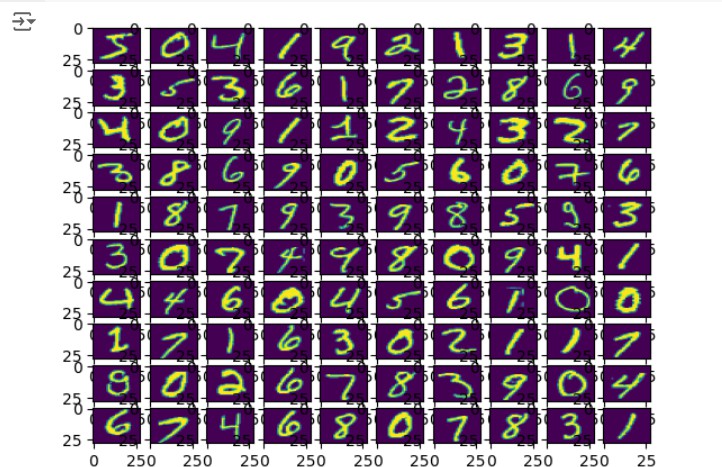
aspect='auto')

k += 1

plt.show()

# OUTPUT:





**RESULT:**

Thisstep-by-stepprocesswillhelpustoimplementMultiLayerPerceptronwith BackpropagationmodelsusingtheKerasdatasetandanalyzetheirperformance.

**EXPT NO: 6 Apythonprogramtodofacerecognitionusing DATE: 27.09.2024 SVM Classifier**

# AIM:

TowriteapythonprogramtoimplementfacerecognitionusingtheSVM Classifier

**PROCEDURE:**

ImplementingfacerecognitionusingtheSVMClassifierusingthecatanddog dataset involve the following steps:

**Step1:ImportNecessaryLibraries**

First,importthelibrariesthatareessentialfordatamanipulation,visualization,and model building.

importpandasaspd import imageio

import os

fromskimage.transformimportresize from skimage.io import imread

import numpy as np

importmatplotlib.pyplotasplt from sklearn import svm

from sklearn.model\_selection import GridSearchCV

fromsklearn.model\_selectionimporttrain\_test\_split from sklearn.metrics import accuracy\_score

from sklearn.metrics import classification\_report

**Step2:LoadtheDogandcatDataset**

Thedogandcatdatasetcanbeloaded.

Categories=['cats','dogs'] flat\_data\_arr=[]#inputarray target\_arr=[] #output array datadir='/content/images'

#pathwhichcontainsallthecategoriesofimages for i in Categories:

print(f'loading...category:{i}') path=os.path.join(datadir,i)

for img in os.listdir(path): img\_array=imread(os.path.join(path,img)) img\_resized=resize(img\_array,(150,150,3)) flat\_data\_arr.append(img\_resized.flatten()) target\_arr.append(Categories.index(i))

print(f'loadedcategory:{i}successfully') flat\_data=np.array(flat\_data\_arr) target=np.array(target\_arr)

#dataframe df=pd.DataFrame(flat\_data) df['Target']=target df.shape

# OUTPUT:



**Step3:Separateinputfeaturesandtargets.**

#input data x=df.iloc[:,:-1] #output data y=df.iloc[:,-1]

**Step4:Separatetheinputfeaturesandtarget**

# Splitting the data into training and testing sets x\_train,x\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=0.20, random\_state=77, stratify=y)

**Step5:Buildandtrainthemodel**

#DefiningtheparametersgridforGridSearchCV param\_grid={'C':[0.1,1,10,100],

'gamma':[0.0001,0.001,0.1,1],

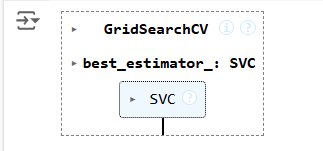
'kernel':['rbf','poly']}

#Creatingasupportvectorclassifier svc=svm.SVC(probability=True)

#CreatingamodelusingGridSearchCVwiththeparametersgrid model=GridSearchCV(svc,param\_grid)

#Trainingthemodelusingthetrainingdata model.fit(x\_train,y\_train)

# OUTPUT:



**Step6:Model evaluation**

#Testingthemodelusingthetestingdata y\_pred = model.predict(x\_test)

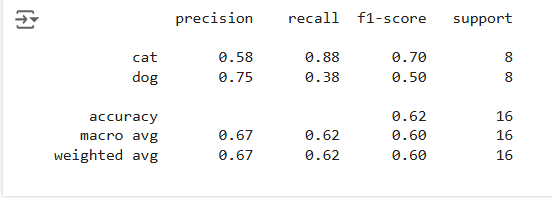
# Calculating the accuracy of the model accuracy=accuracy\_score(y\_pred,y\_test)

# Print the accuracy of the model

print(f"The model is {accuracy\*100}% accurate")

print(classification\_report(y\_test, y\_pred, target\_names=['cat', 'dog']))

# OUTPUT:



**Step7: Prediction**

path='/content/cat.83.jpg'img=imread(path) plt.imshow(img)

plt.show() img\_resize=resize(img,(150,150,3)) l=[img\_resize.flatten()] probability=model.predict\_proba(l)

for ind,val in enumerate(Categories):

print(f'{val}= {probability[0][ind]\*100}%')

print("The predicted image is : "+Categories[model.predict(l)[0]])

# OUTPUT:

cats = 52.70216647851706%

dogs = 47.29783352148294%

The predicted image is : cat

# RESULT:

ThustheprocesshelpsustoimplementthefacerecognitionusingSVMClassifier using python program.

# EX.NO: 8 APYTHONPROGRAMTOIMPLEMENT DATE : 18.10.2024 ADA BOOSTING

**AIM:**

TowriteapythonprogramtoimplementADABoosting.

# PROCEDURE:

ImplementingADABoostingusingthedatasetinvolvethefollowingsteps:

**Step1:ImportNecessaryLibraries**

First,importthelibrariesthatareessentialfordatamanipulation,visualization,and model building.

import numpy as np importpandasaspd

from sklearn.tree import DecisionTreeClassifier

frommlxtend.plottingimportplot\_decision\_regions import seaborn as sns

from sklearn.metrics import accuracy\_score

**Step2:Loadandpreparedata**

**sns.scatterplot(x=df['X1'], y=df['X2'], hue=df['label'])**

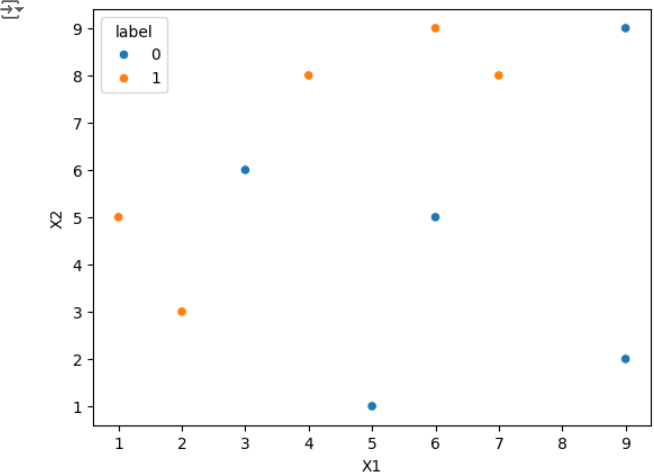
**df['weights'] = 1 / df.shape[0]**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **df = pd.DataFrame()** | | | | | | | | |
| **df['X1'] = [1, 2,** | **3,** | **4,** | **5,** | **6,** | **6,** | **7,** | **9,** | **9]** |
| **df['X2'] = [5, 3,** | **6,** | **8,** | **1,** | **9,** | **5,** | **8,** | **9,** | **2]** |
| **df['label'] = [1,** | **1,** | **0,** | **1,** | **0,** | **1,** | **0,** | **1,** | **0, 0]** |

x = df.iloc[:, 0:2].values

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

# OUTPUT:



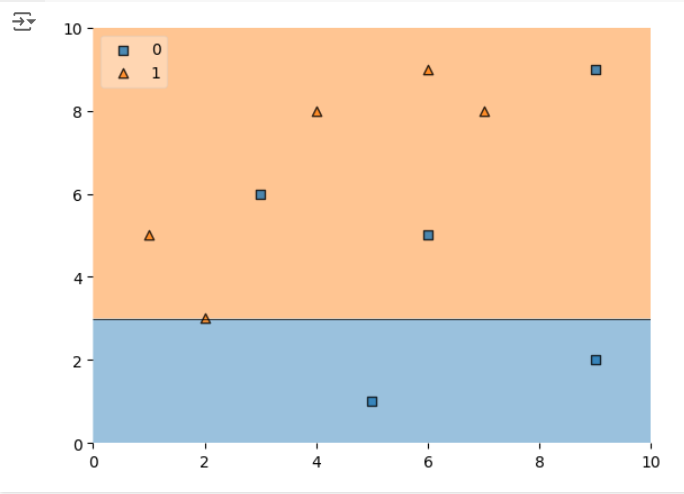
**Step3:Trainthe1stmodel**

# Step 2: Train 1st Model

dt1=DecisionTreeClassifier(max\_depth=1) dt1.fit(x, y)

plot\_decision\_regions(x,y,clf=dt1,legend=2) df['y\_pred'] = dt1.predict(x)

# OUTPUT:



**Step4:Calculatemodelweight**

# Step 4: Update Weights

defupdate\_row\_weights(row,alpha=0.423): if row['label'] == row['y\_pred']:

returnrow['weights']\*np.exp(-alpha) else:

return row['weights'] \* np.exp(alpha)

df['updated\_weights']= df.apply(update\_row\_weights, axis=1)

df['normalized\_weights']=df['updated\_weights']/ df['updated\_weights'].sum()

df['cumsum\_upper'] = np.cumsum(df['normalized\_weights'])

df['cumsum\_lower'] = df['cumsum\_upper'] - df['normalized\_weights']

**Step5:Createnewdataset**

# Step 5: Create New Dataset

defcreate\_new\_dataset(df): indices = []

foriinrange(df.shape[0]): a = np.random.random()

for index, row in df.iterrows():

ifrow['cumsum\_upper']>aanda>row['cumsum\_lower']: indices.append(index)

return indices

index\_values = create\_new\_dataset(df)

second\_df = df.iloc[index\_values, [0, 1, 2, 3]]

**Step6:Train2ndmodel**

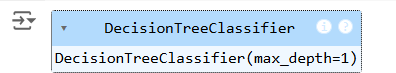
# Step 6: Train 2nd Model

dt2=DecisionTreeClassifier(max\_depth=1) x = second\_df.iloc[:, 0:2].values

y = second\_df.iloc[:, 2].values

dt2.fit(x, y)

# OUTPUT:



**Step7:Plotdecisiontreeandcalculatemodelweightsfor2ndmodel**

# Plot the decision tree for the second model plot\_decision\_regions(x,y,clf=dt2,legend=2) second\_df['y\_pred'] = dt2.predict(x)

#Step7:CalculateModelWeightfor2ndModel alpha2 = calculate\_model\_weight(0.1)

print(f"Alpha2: {alpha2}")

**Step8:updateweightsfor2ndmodel**

# Step 8: Update Weights for 2nd Model

defupdate\_row\_weights(row,alpha=1.09): if row['label'] == row['y\_pred']:

returnrow['weights']\*np.exp(-alpha) else:

return row['weights'] \* np.exp(alpha)

second\_df['updated\_weights']= second\_df.apply(update\_row\_weights, axis=1)

second\_df['nomalized\_weights']=second\_df['updated\_weights']/ second\_df['updated\_weights'].sum()

second\_df['cumsum\_upper'] = np.cumsum(second\_df['nomalized\_weights'])

second\_df['cumsum\_lower']=second\_df['cumsum\_upper']- second\_df['nomalized\_weights']

**Step9:Calculatealphafor3rdmodel**

#Step9:CalculateAlphafor3rdModel alpha3 = calculate\_model\_weight(0.7)

print(f"Alpha3: {alpha3}")

# Step 10: Accuracy Calculation y\_true = second\_df['label'].values y\_pred=second\_df['y\_pred'].values

#CalculateaccuracyfortheAdaBoostmodel accuracy = accuracy\_score(y\_true, y\_pred)

print(f"Accuracy of the AdaBoost model: {accuracy:.4f}")

# OUTPUT:

**ALPHA3:-0.4236489301936017**

**AccuracyoftheAdaBoostingmodel:0.80000**

# RESULT:

ThusthepythonprogramtoimplementAdaboostinghasbeenexecuted successfully and the results have been verified.

**EXPT NO: 9A Apythonprogramtoimplement DATE: 25.10.2024 KNN MODEL .**

# AIM:

TowriteapythonprogramtoimplementKNNModel.

# PROCEDURE:

ImplementingKNNModelusingthemall\_customerdatasetinvolvethefollowing steps:

**Step1:ImportNecessaryLibraries**

First,importthelibrariesthatareessentialfordatamanipulation,visualization,and model building.

import numpy as np

importmatplotlib.pyplotasplt import pandas as pd

fromsklearn.model\_selectionimporttrain\_test\_split from sklearn.preprocessing import StandardScaler

from sklearn.neighbors import KNeighborsClassifier

fromsklearn.metricsimportclassification\_report,confusion\_matrix from sklearn.cluster import KMeans

**Step2:LoadtheDataset**

Themall\_customerdatasetcanbeloadedanddisplaythefirstfewrowsofthedataset.

# Load the dataset

dataset = pd.read\_csv('/content/Mall\_Customers.csv')

#Displaythefirstfewrowsofthedataset print(dataset.head())

# Display the dimensions of the dataset print(f"Datasetshape:{dataset.shape}")

#Displaydescriptivestatisticsofthedataset print(dataset.describe())

**Step3:Separatethefeatures(x)andtargetvariable (y)**

# Separate the features (X) and the target variable (y)

X = dataset.iloc[:, [3, 4]].values #Weuse'AnnualIncome'and'Spending Score'

#Standardizethefeatures scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

**Step4:Visualizingtheclusterofcustomer**

#ApplyKMeansclusteringusingtheElbowMethodtofindtheoptimalnumber of clusters

wcss = [] #Within-clustersumofsquares for i in range(1, 11):

kmeans=KMeans(n\_clusters=i,init='k-means++',max\_iter=300,n\_init=10, random\_state=0)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **kmeans.fit(X\_scaled) wcss.append(kmeans.inertia\_)**  **# Plot the Elbow Method graph plt.plot(range(1, 11), wcss) plt.title('The Elbow Method') plt.xlabel('Numberofclusters') plt.ylabel('WCSS')**  **plt.show()**  **#Fromtheplot,wecanobservethattheoptimalnumberofclustersis5 (elbow point)**  **kmeans=KMeans(n\_clusters=5,init='k-means++',max\_iter=300,n\_init=10, random\_state=0)**  **y\_kmeans = kmeans.fit\_predict(X\_scaled)**  **# Visualizing the clusters of customers** | | | | | | | | |
| **plt.scatter(X\_scaled[y\_kmeans c='red', label='Cluster 1')** | **==** | **0,** | **0],** | **X\_scaled[y\_kmeans** | **==** | **0,** | **1],** | **s=100,** |
| **plt.scatter(X\_scaled[y\_kmeans c='blue', label='Cluster 2')** | **==** | **1,** | **0],** | **X\_scaled[y\_kmeans** | **==** | **1,** | **1],** | **s=100,** |
| **plt.scatter(X\_scaled[y\_kmeans c='green', label='Cluster 3')** | **==** | **2,** | **0],** | **X\_scaled[y\_kmeans** | **==** | **2,** | **1],** | **s=100,** |
| **plt.scatter(X\_scaled[y\_kmeans c='cyan', label='Cluster 4')** | **==** | **3,** | **0],** | **X\_scaled[y\_kmeans** | **==** | **3,** | **1],** | **s=100,** |
| **plt.scatter(X\_scaled[y\_kmeans** | **==** | **4,** | **0],** | **X\_scaled[y\_kmeans** | **==** | **4,** | **1],** | **s=100,** |
| **c='magenta', label='Cluster 5')** | | | | | | | | |

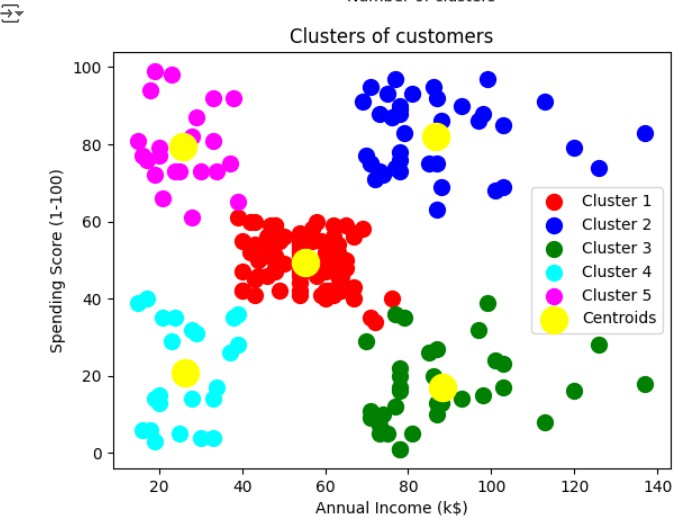
# Plot the centroids

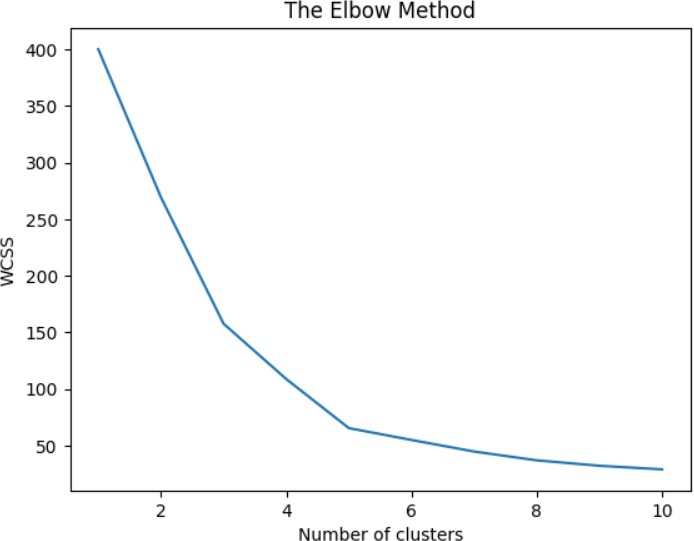
plt.scatter(kmeans.cluster\_centers\_[:,0],kmeans.cluster\_centers\_[:,1], s=300, c='yellow', label='Centroids')

plt.title('Clusters of customers') plt.xlabel('Annual Income (k$)') plt.ylabel('SpendingScore(1-100)') plt.legend()

plt.show()

# OUTPUT:





**RESULT:**

ThusthepythonprogramtoimplementKNNmodelhasbeensuccessfully implemented and the results have been verified.

**EXPT NO: 9B Apythonprogramtoimplement DATE: 25.10.2024 K-Means Model**

# AIM:

TowriteapythonprogramtoimplementtheK-meansModel.

# PROCEDURE:

ImplementingK-meansModelusingthemall\_customerdatasetinvolvethe following steps:

**Step1:ImportNecessaryLibraries**

First,importthelibrariesthatareessentialfordatamanipulation,visualization,and model building.

import numpy as np importpandasaspd

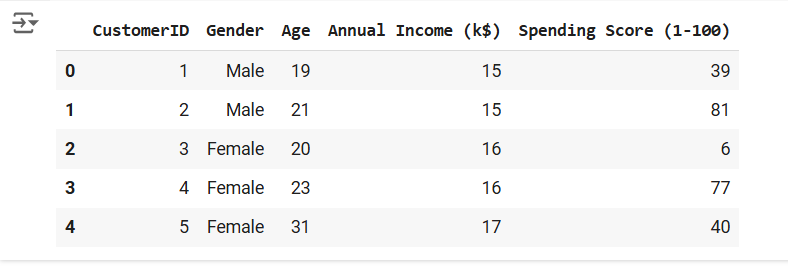
from math import sqrt

**Step2:loadthe Dataset**

data = pd.read\_csv('/content/Mall\_Customers.csv')

data.head(5)

# OUTPUT:

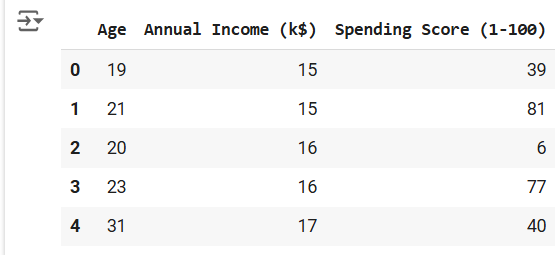


**Step3:Preprocessthedata**

req\_data = data[['Age', 'Annual Income (k$)', 'Spending Score (1-100)']]

req\_data.head(5)

# OUTPUT:



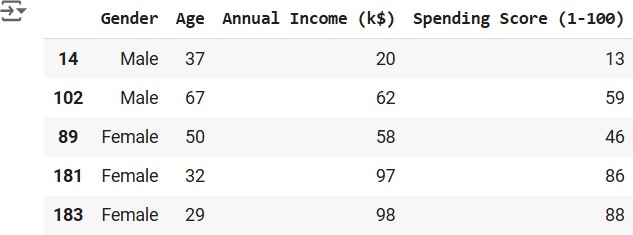
**Step4:Assignthedatapointstoclusters**

shuffle\_index = np.random.permutation(req\_data.shape[0]) #Shufflethe dataset rows

req\_data = req\_data.iloc[shuffle\_index]

req\_data.head(5)

# OUTPUT:



**Step5:Updatetheclusterscenters**

train\_size = int(req\_data.shape[0]\*0.7) #Set70%ofthedatafortraining train\_df = req\_data.iloc[:train\_size,:]

test\_df = req\_data.iloc[train\_size:,:]

train = train\_df.values #Converttraindatatonumpyarray test = test\_df.values # Convert test data to numpy array y\_true = test[:,-1] # The target values for the test set

print('Train\_Shape:',train\_df.shape) print('Test\_Shape: ', test\_df.shape)

from math import sqrt

defeuclidean\_distance(x\_test,x\_train): distance = 0

for i in range(len(x\_test)): #Loopthroughallfeatures distance += (x\_test[i]-x\_train[i])\*\*2

return sqrt(distance)

defget\_neighbors(x\_test,x\_train,num\_neighbors): distances = []

data = []

for i in x\_train: distances.append(euclidean\_distance(x\_test,i)) data.append(i)

distances = np.array(distances)

data = np.array(data)

sort\_indexes = distances.argsort() #Sortdistancesinascendingorder data = data[sort\_indexes] # Sort the data based on sorted distances

return data[:num\_neighbors] #Returntheclosest'num\_neighbors'neighbors

defprediction(x\_test,x\_train,num\_neighbors): classes = []

neighbors=get\_neighbors(x\_test,x\_train,num\_neighbors) for i in neighbors:

classes.append(i[-1]) #The target value is the last column

predicted = max(classes, key=classes.count) #Returnthemostfrequent class (the majority vote)

return predicted

defpredict\_classifier(x\_test): classes = []

neighbors = get\_neighbors(x\_test, req\_data.values, 5) #Predictusing the top 5 neighbors

for i in neighbors: classes.append(i[-1])

predicted = max(classes, key=classes.count) #Returnthemajorityvote print(predicted)

return predicted

defaccuracy(y\_true,y\_pred): num\_correct = 0

for i in range(len(y\_true)):

if y\_true[i] == y\_pred[i]: #Comparetruevaluestopredictedvalues num\_correct += 1

accuracy = num\_correct / len(y\_true) #Calculateaccuracyastheratio of correct predictions

return accuracy

defaccuracy(y\_true,y\_pred): num\_correct = 0

for i in range(len(y\_true)):

ify\_true[i]==y\_pred[i]: num\_correct += 1

returnnum\_correct/len(y\_true) y\_pred = []

for i in test:

y\_pred.append(prediction(i, train, 5)) #Makepredictionsforeachtest instance

#Calculateandprinttheaccuracy acc = accuracy(y\_true, y\_pred)

print(f"Accuracy: {acc \* 1000:.2f}%")

# OUTPUT:



**RESULT:**

Thusthepythonprogramimplementingthek-meansmodelissuccessful.

**EXPT NO: 10 ApythonprogramtoimplementDimensionality DATE: 04.11.2024 Reduction -PCA.**

# AIM:

TowriteapythonprogramtoimplementDimensionalityReduction-PCA.

# PROCEDURE:

ImplementingDimensionalityreduction-pcausingtheIrisdatasetinvolvethe following steps:

**Step1:ImportNecessaryLibraries**

First,importthelibrariesthatareessentialfordatamanipulation,visualization,and model building.

#Importingnecessarylibraries from sklearn import datasets

import pandas as pd

fromsklearn.preprocessingimportStandardScaler from sklearn.decomposition import PCA

import seaborn as sns

import matplotlib.pyplot as plt

**Step2:LoadtheIrisDataset**

TheIrisdatasetcanbeloadedanddisplaythefirstfewrowsofthedataset

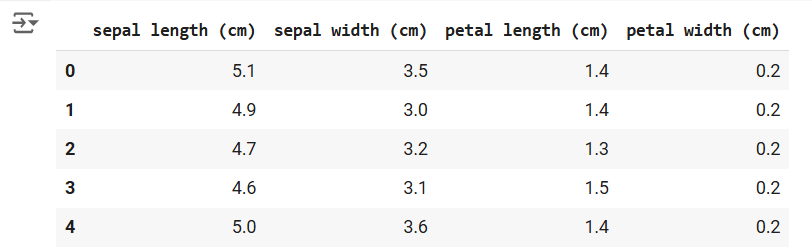
# Load the Iris dataset iris=datasets.load\_iris()

df = pd.DataFrame(iris['data'], columns=iris['feature\_names'])

# Display the first few rows of the dataset

df.head()

# OUTPUT:



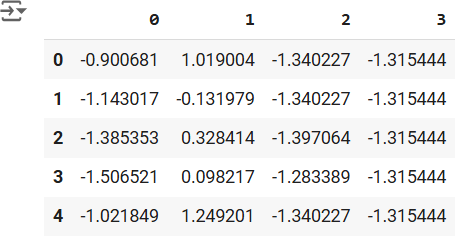
**Step3:Standardizethe data**

#StandardizethefeaturesusingStandardScaler scalar = StandardScaler()

scaled\_data = pd.DataFrame(scalar.fit\_transform(df)) #Scaling the data

#Displaythescaleddata(optional) scaled\_data.head()

# OUTPUT:



**Step4:ApplyPCA**

#ApplyPCAtoreducethedatato3components pca = PCA(n\_components=3)

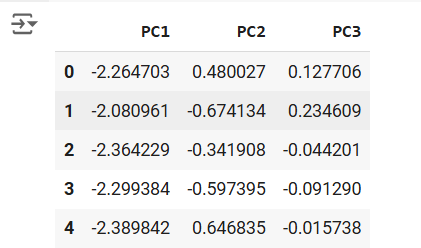
pca.fit(scaled\_data) #Fit PCA on scaled data

data\_pca = pca.transform(scaled\_data) #Transformthedatatoprincipal components

# Convert PCA data to a DataFrame for easier inspection

data\_pca=pd.DataFrame(data\_pca,columns=['PC1','PC2','PC3']) data\_pca.head()

# OUTPUT:



**Step5:ExplainedVarianceRatio**

#Calculatetheexplainedvarianceratioforeachprincipalcomponent explained\_variance = pca.explained\_variance\_ratio\_

print(f"ExplainedVariance Ratio: {explained\_variance}")

# This output shows how much variance each principal component explains.

# OUTPUT:



**Step6:Visualizethereduceddata.**

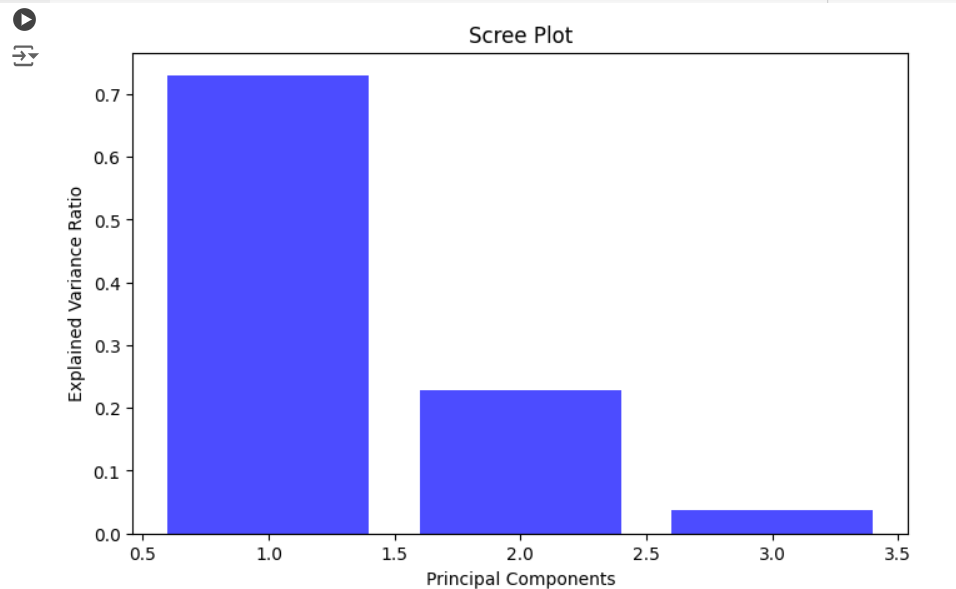
#Plottingtheexplainedvarianceratioasascreeplot plt.figure(figsize=(8, 5))

plt.bar(range(1,len(explained\_variance)+1),explained\_variance,alpha=0.7, color='blue')

plt.ylabel('ExplainedVarianceRatio') plt.xlabel('Principal Components') plt.title('Scree Plot')

plt.show()

# OUTPUT:



**RESULT:**

ThustheDimensionalityReductionhasbeenimplementedusingPCAinpython program Successfully.