RAJALAKSHMI ENGINEERING COLLEGE

RAJALAKSHMI NAGAR, THANDALAM – 602 105

RAJALAKSHMI
ENGINEERING COLLEGE

AI23331
FUNDAMENTALS OF MACHINE LEARNING LAB

Laboratory Observation Note Book

Name : Aswini G

Year / Branch / Section : 2nd Year/ AIML / A

Register No. : 231501028

Semester : 3rd Semester

Academic Year : 2024-2025

S.NO	EXPERIMENTS
1	A PYTHON PROGRAM TO IMPLEMENT UNIVARIATE, BIVARIATE AND MULTIVARIATE REGRESION
2	A PYTHON PROGRAM TO IMPLEMENT SIMPLE LINEARREGRESSION USING LEAST SQUARE METHOD
3	A PYTHON PROGRAM TO IMPLEMENT LOGISTIC MODEL
4	A PYTHON PROGRAM TO IMPLEMENT SINGLE LAYER PERCEPTRON
5	A PYTHON PROGRAM TO IMPLEMENT MULTI LAYER PERCEPTRON WITH BACK PROPOGATION
6	A PYTHON PROGRAM TO IMPLEMENT SVM CLASSIFIER MODEL
7	A PYTHON PROGRAM TO IMPLEMENT DECISION TREE
8	A PYTHON PROGRAM TO lMPLEMENT ADA BOOSTING
9	a. A PYTHON PROGRAM TO IMPLEMENT KNN MODEL b. A PYTHON PROGRAM TO IMPLEMENT K-MEANS MODEL
10	A PYTHON PROGRAM TO IMPLEMENT DIMENSIONALITY REDUCTION USING PCA

\

Ex No: 1
Date:

A PYTHON PROGRAM TO IMPLEMENT UNIVARIATE, BIVARIATE AND MULTIVARIATE REGRESION

Aim:

To implement a python program using univariate, bivariate and multivariate regression features for a given iris dataset.

Algorithm:

Step l: Import necessary libraries:

- pandas for data manipulation, numpy for numerical operations, and matplotlib.pyplot for plotting.

Step 2: Read the dataset:

- Use the pandas 'read_csv' function to read the dataset.
- Store the dataset in a variable (e.g., 'data').

Step 3: Prepare the data:

- Extract the independent variable(s) (X) and dependent variable (y) from the dataset.
- Reshape X and y to be 2D arrays if needed.

Step 4:Univariate Regression:

- For univariate regression, use only one independent variable.
- Fit a linear regression model to the data using numpy's polyfit function or skleam's LinearRegression class.
- Make predictions using the model.
- Calculate the R-squared value to evaluate the model's performance.

Step 5: Bivariate Regression:

- For bivariate regression, use two independent variables.
- Fit a linear regression model to the data using numpy's's'polyfif function or sklearn's 'LinearRegression' class.
- Make predictions using the model.
- Calculate the R-squared value to evaluate the model's performance.

Step 6: Multivariate Regression:

- For multivariate regression, use more than two independent variables.
- Fit a linear regression model to the data using skleam's 'LinearRegression' class.
- Make predictions using the model.
- Calculate the R-squared value to evaluate the model's perfotmance.

Step 7: Plot the results:

- For univariate regression, plot the original data points (X, y) as a scatter plot and the regression line as a line plot.
- For bivariate regression, plot the original data points (Xl, X2, y) as a 3D scatter plot and the regression plane.
- For multivariate regression, plot the predicted values against the actual values.

Step 8: Display the results:

- Print the coefficients (slope) and intercept for each regression model.
- Print the R-squared value for each regression model.

Step 9: Complete the program:

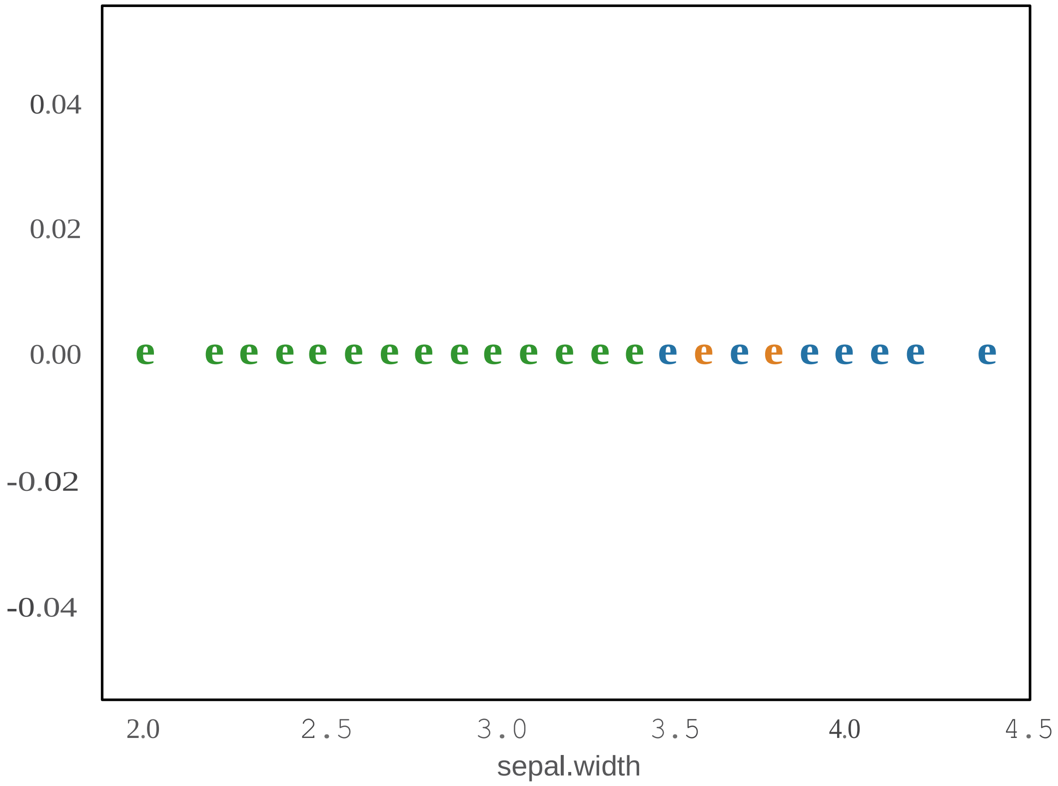
- Combine all the steps into a Python program.
- Run the program to perform univariate, bivariate, and multivariate regression on the dataset.

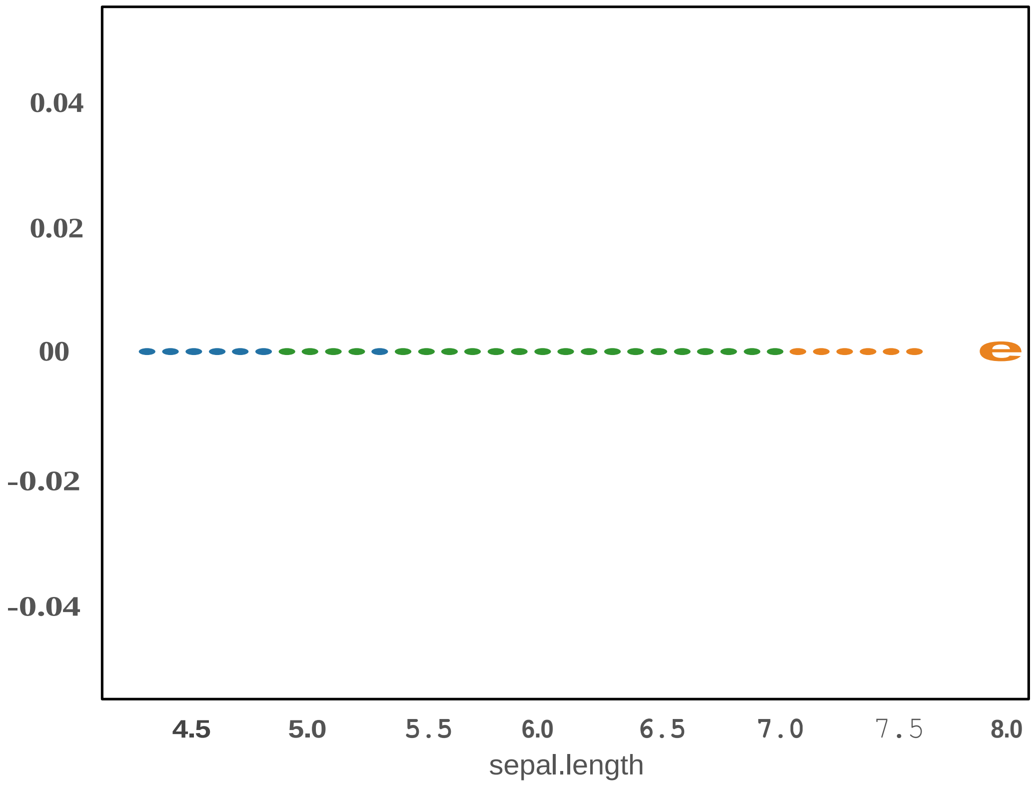
PROGRAM:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
df = pd.read_csv('../input/iris-dataset/iris.csv')
df.head(150)
df.shape
(150,5)
#univariate for sepal width
df.loc[df['variety']=='Setosa']
df_Setosa=df.loc[df['variety']=='Setosa']
df_Virginica=df.loc[df['variety']=='Virginica'J
df_Versicolor=df.loc[df['variety']=='Versicolor']
plt.scatter(df_Setosa['sepal.width'J,np.zeros_like(df_Setosa['sepal.width']))
plt.scatter(df_Virginica['sepal.width'J,np.zeros_like(df_Virginica['sepal.width']
))
plt.scatter(df_Versicolor['sepal.width'J,np.zeros_like(df_Versicolor['sepal.width
']))
plt.xlabel('sepal.width')
plt.show()

#univariate for sepal length
df.loc[df['variety']=='Setosa']
df_Setosa=df.loc[df['variety']=='Setosa']
df_Virginica=df.loc[df['variety']=='Virginica']
df_Versicolor=df.loc[df['variety']=='Versicolor']
plt.scatter(df_Setosa['sepal.length'],np.zeros_like(df_Setosa['sepal.length']))
plt.scatter(df_Virginica['sepal.length'],np.zeros_like(df_Virginica['sepal.length
']))
plt.scatter(df_Versicolor['sepal.length'],np.zeros_like(df_Versicolor['sepal.leng
th']))
plt.xlabel('sepal.length')
plt.show()

#univariate for petal width
df.loc[df['variety']=='Setosa']
df_Setosa=df.loc[df['variety']=='Setosa']
df_Virginica=df.loc[df['variety']=='Virginica'J
df_Versicolor=df.loc[df['variety']=='Versicolor']
```

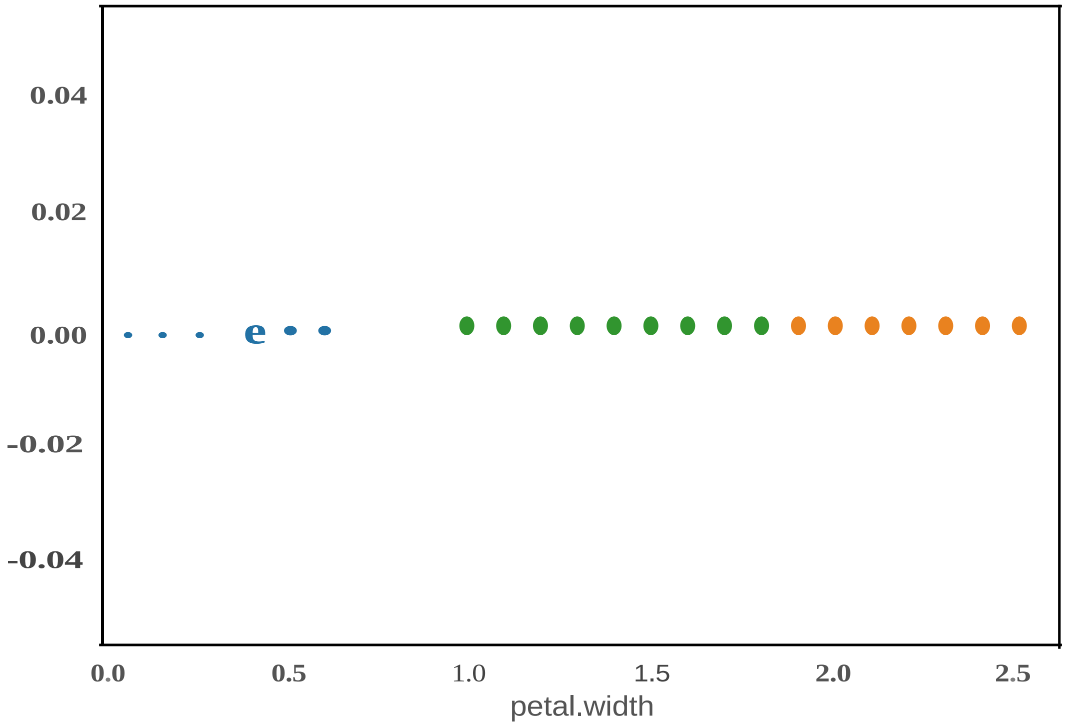
```
plt.scatter(df_Setosa['petal.width'],np.zeros_like(df_Setosa['petal.width']))
plt.scatter(df_Virginica['petal.width'],np.zeros_like(df_Virginica['petal.width']
))
plt.scatter(df_Versicolor['petal.width'],np.zeros_like(df_Versicolor['petal.width
']))
plt.xlabel('petal.width')
plt.show()

#univariate for petal length
df.loc[df['variety']=='Setosa']
df_Setosa=df.loc[df['variety']=='Setosa']
df_Virginica=df.loc[df['variety']=='Virginica']
df_Versicolor=df.loc[df['variety']=='Versicolor']
plt.scatter(df_Setosa['petal.length'],np.zeros_like(df_Setosa['petal.length']))
plt.scatter(df_Virginica['petal.length'],np.zeros_like(df_Virginica['petal.length
']))
plt.scatter(df_Versicolor['petal.length'],np.zeros_like(df_Versicolor['petal.leng
th']))
plt.xlabel('petal.length')
plt.show()

#bivariate sepal.width vs petal.width
sns.FacetGrid(df,hue='variety',size=S).map(plt.scatter,"sepal.width'',"petal.width
").add_legend();
plt.show()

#bivariate sepal.length vs petal.length
sns.FacetGrid(df,hue='variety",size=S).map(plt.scatter,"sepal.length","petal.leng
th").add_legend();
plt.show()


#multivariate all the features
sns.pairplot(df,hue="variety",size=2)
```

	sepal.length	sepal.width	petal.length	petal.width	variety
0	5.1	3.5	1.4	0.2	Setosa
1	4.9	3.0	1,4	0.2	Setosa
2	4.7	3.2	1.3	0.2	Setosa
3	4.6	3.1	1.5	0.2	Setosa
4	5.0	3.6	1.4	0.2	setosa

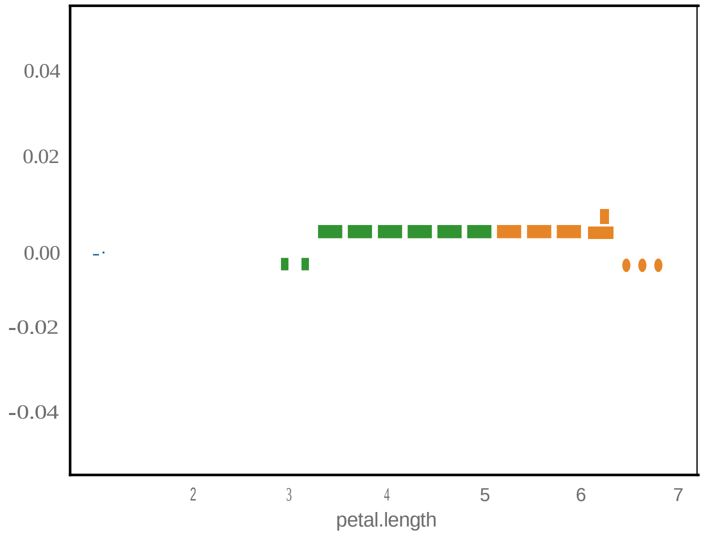


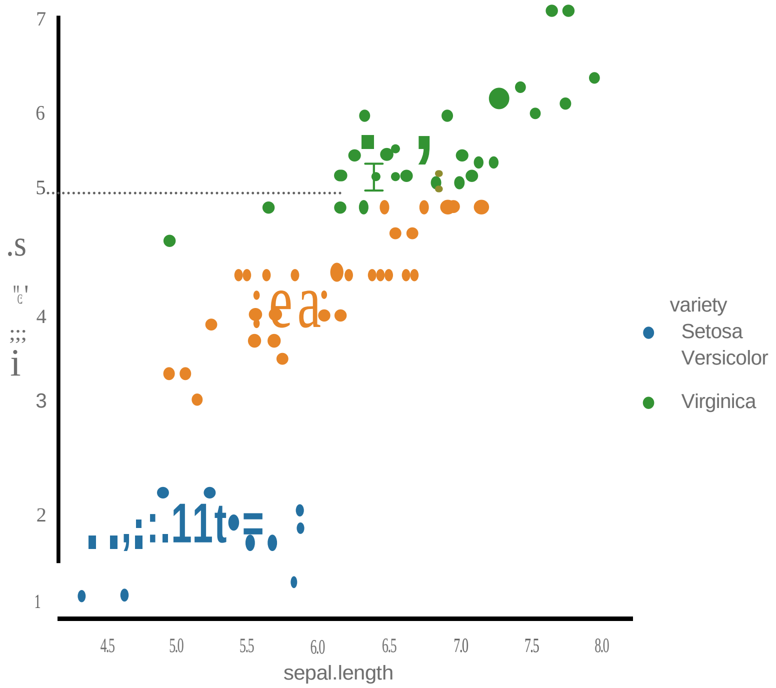


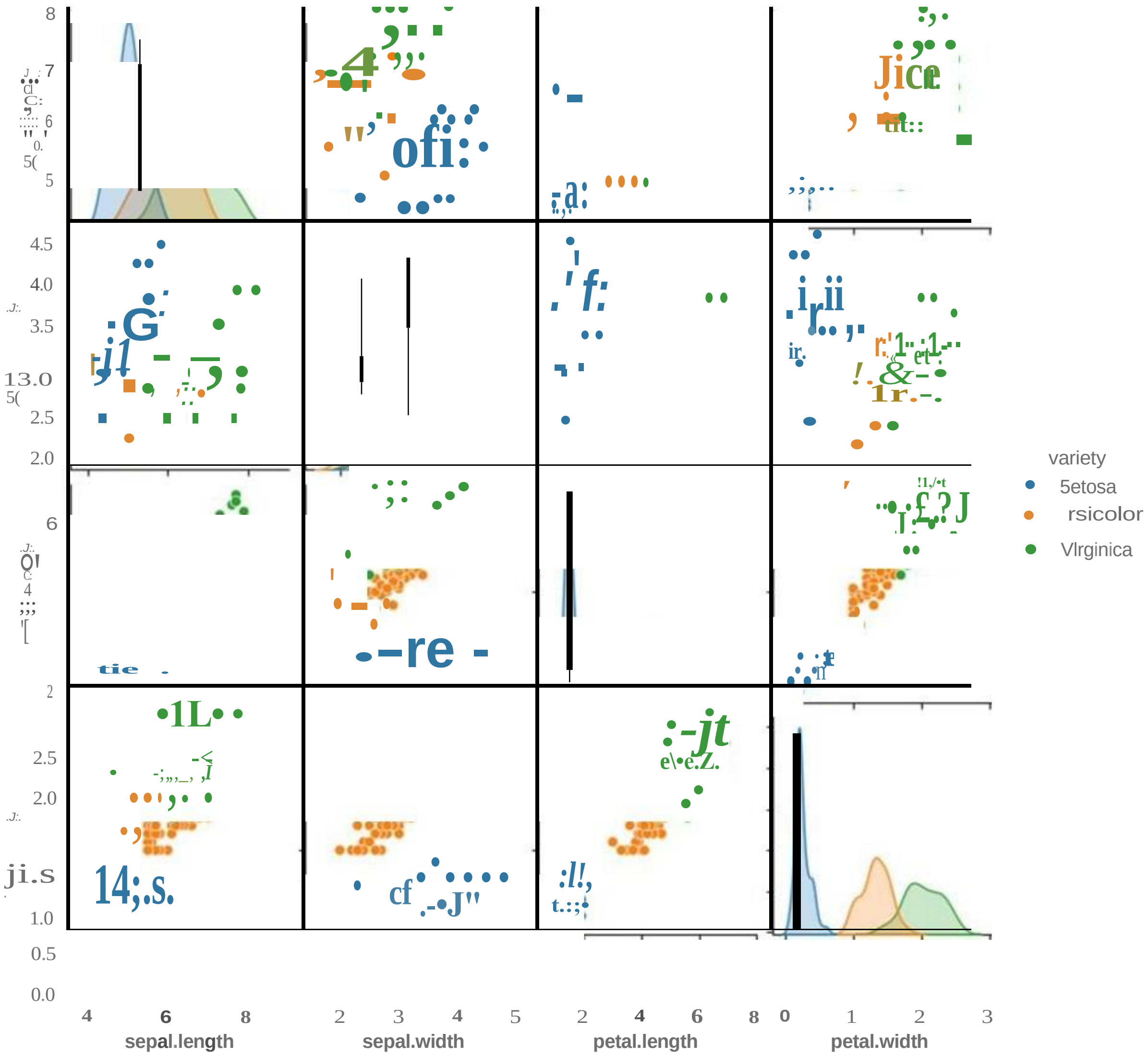
0.04
0.02
00
-0.02
-0.04
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0
sepal.length



0.04
0.02
0.00
-0.02
-0.04
0.0
0.5
1.0
1.5
2.0
2.5
petal.width


0.04
0.02
0.00
-0.02
-0.04
2
3
4
5
6
7
petal.length


7
6
5
4
3
2
1
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0
sepal.length
variety
Setosa
Versicolor
Virginica



RESULT:-

The Python program to implement **Univariate, Bivariate, and Multivariate Analysis** has been executed successfully. The data distributions, relationships, and variations across dimensions have been verified and analyzed, confirming accurate computation and interpretation.

Ex No: 2
Date:

A PYTHON PROGRAM TO IMPLEMENT SIMPLE LINEAR REGRESSION USING LEAST SQUARE METHOD

Aim:

To implement a python program for constructing a simple linear regression using least square method.

Algorithm:

Step 1: Import necessary libraries:

- pandas for data manipulation and matplotlib.pyplot for plotting.

Step 2: Read the dataset:

- Use the pandas 'read_csv' function to read the dataset (e.g., headbrain.csv).
- Store the dataset in a variable (e.g., 'data').

Step 3: Prepare the data:

- Extract the independent variable (X) and dependent variable (y) from the dataset.
- Reshape X and y to be 2D arrays if needed.

Step 4: Calculate the mean:

- Calculate the mean of X and y.

Step 5: Calculate the coefficients:

- Calculate the slope (m) using the formula:

  m = I::'--1(X; X)(y;-y) / I:1'...1(X;-X)2

- Calculate the intercept (b) using the formula:*b* = ӥ *mX*

Step 6: Make predictions:

- Use the calculated slope and intercept to make predictions for each X value:

  fj=mx+b

Step 7: Plot the regression line:

- Plot the original data points (X, y) as a scatter plot.
- Plot the regression line (X, predicted_y) as a line plot.

Step 8: Calculate the R-squared value:

- Calculate the total sum of squares (TSS) using the formula: $TSS-$ I:? 1(y-, YI^2
- RSS - ._,n (· - --)^2 Calculate the residual sum of squares (RSS) using the formula: Y,
- Calculate the R-squared value using the formula: $R^2 = 1 -$:ff

Step 9: Display the results:

- Print the slope, intercept, and R-squared value.

Step 10: Complete the program:

- Combine all the steps into a Python program.
- Run the program to perform simple linear regression on the dataset.

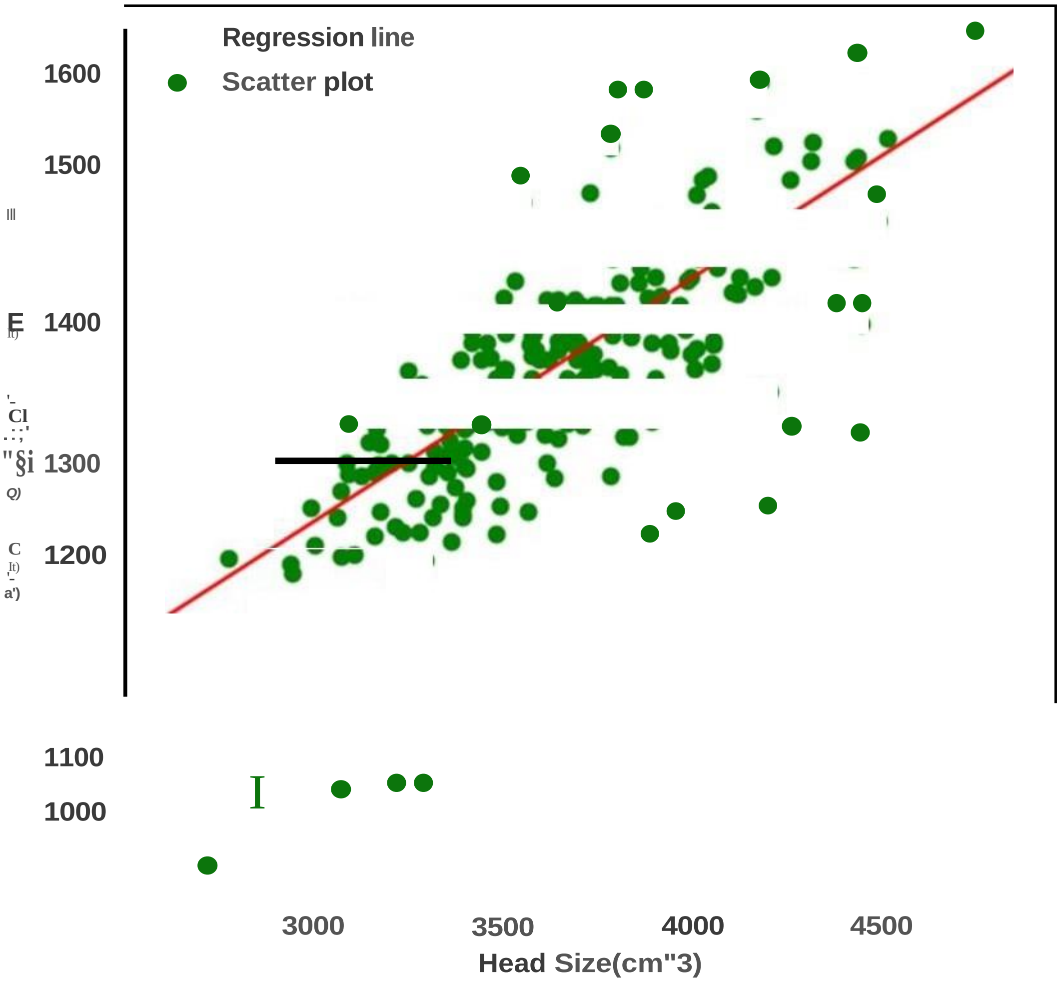
PROGRAM:

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
data= pd.read_csv('headbrain.csv')
x, y = np.array(list(data['Head Size(cmA3)'])), np.array(list(data['Brain
Weight(grams)']))
print(x[:5], y[:5])
[4512 3738 4261 3777 4177][1530 1297 1335 1282 1590]
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)
 return lambda x : m*x+c
lin = get_line(x, y)
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='Scatter plot')
plt.xlabel('Head Size(cmA3)')
plt.ylabel('Brain Weight(grams)')
plt.legend()
plt.show()

def  get_error(line_fuc, x, y):
 y_m = np.mean(y)
 y_pred = np.array([line_fuc(_) for   in x])
 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)

In-built Package
from sklearn.linear model import LinearRegression
x = x.reshape((len(x),1))
reg=LinearRegression()
reg=reg.fit(x, y)
print(reg.score(x, y))
```

OUTPUT:

[4512 3738 426137774177] [1530 1297 1335 1282 1590]

3633.9915611814345 I282.873417721519
0.2634293394893993325.5734210494428



0.639311719957

RESULT:-

The Python program for **Simple Linear Regression using the Least Squares Method** has been implemented and executed successfully. The results have been verified, and the model's parameters, along with the goodness-of-fit, have been analyzed, confirming accuracy in predictions and error minimization.

Ex no: 3
Date:

A PYTHON PROGRAM TO IMPLEMENT LOGISTIC MODEL

Aim:

Io implement python program for the logistic model using suv car dataset.

Algorithm:

Step 1: Import Necessary Libraries:

- pandas for data manipulation
- sklearn.model_selection for train-test split
- skleam.preprocessing for data preprocessing
- sklearn.linear_model for logistic regression
- matplotlib.pyplot for plotting

Step 2: Read the Dataset:

- Use pandas to read the suv_cars.csv dataset into a DataFrame.

Step 3: Preprocess the Data:

- Select the relevant columns for the analysis (e.g., 'Age', 'EstimatedSalary', 'Purchased').
- Encode categorical variables if necessary (e.g., usmg Labe!Encoder or OneHotEncoder).
- Split the data into features (X) and target variable (y).

Step 4: Split the Data:

- Split the dataset into training and testing sets using train_test_split.

Step 5: Feature Scaling:

- Standardize the features using StandardScaler to ensure they have the same scale.

Step 6: Create and Train the Model:

- Create a logistic regression model using LogisticRegression from skleam.Iinear_model.
- Train the model on the training data using the fit method.

- o Create a function named "Sigmoid()" which will define the sigmoid values using the
- o formula (1/1+e-z) and return the computed value.
- o Create a function named "initialize()" which will initialize the values with zeroes and assign the value to "weights" variable, initializes with ones and assigns the value to variable "x" and returns both "x" and "weights".
- o Create a function named "fit" which will be used to plot the graph according to the training data.
- o Create a predict function that will predict values according to the training model created using the fit function.
- o Invoke the standardize() function for "x-train" and "x-test"

Step 7: Make Predictions:

- Use the trained model to make predictions on the test data using the predict method.
  - o Use the "predict()" function to predict the values of the testing data and assign the value to "y_pred" variable.
  - o Use the "predict()" function to predict the values of the training data and assign the value to "y_trainn" variable.
  - o Compute fl_ score for both the training and testing data and assign the values to "fl_score_tr" and "fl_score_te" respectively

Step 8: Evaluate the Model:

- Calculate the accuracy of the model on the test data using the score method.
  (Accuracy= (tp+tn)/(tp+tn+fp+fn)).
- Generate a confusion matrix and classification report to further evaluate the model's performance.

Step 9: Visualize the Results:

- Plot the decision boundary of the logistic regression model (optional).

PROGRAM:

```
import pandas as pd
import numpy as np
from numpy import log,dot,exp,shape
from sklearn.metrics import confusion_matrix
data= pd.read_csv('../input/suvcars/suv_data.csv')
print(data.head())


x = data.iloc[:, [2, 3]].values
y = data.iloc[:, 4].values
In-built Function
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test=train_test_split(x,y,test_size=0.10,
random_state=0)
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x_train=sc.fit_transform(x_train)
x_test=sc.transform(x_test)
print (x_train[0:10,:])

from sklearn.linear model import LogisticRegression
classifier=LogisticRegression(random_state=0)
classifier.fit(x_train,y_train)
LogisticRegression  (random_state=0)
y_pred =  classifier.predict(x_test)
print(y_pred)


from sklearn.metrics import confusion_matrix
cm= confusion_matrix(y_test, y_pred)
print ("Confusion Matrix :  \n", cm)

from sklearn.metrics import accuracy_score
print ("Accuracy :  ", accuracy_score(y_test, y_pred))

User Defined function
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test=train_test_split(x,y,test_size=0.10,
random_state=0)
def Std(input_data):
mean0 = np.mean(input_data[:, 0])
sd0 = np.std(input_data[:, 0])
meanl = np.mean(input_data[:, 1])
```

```
sd1 = np.std(input_data[:, 1])
return lambda x:((x[0]-mean0)/sd0, (x[1]-mean1)/sd1)
my_std = Std(x)
my_std(x_train[0])

def standardize(X_tr):
for i in range(shape(X_tr)[1]):
X_tr[:,i] = (X_tr[:,i] - np.mean(X_tr[:,i]))/np.std(X_tr[:,i])
def F1_score(y,y_hat):
tp,tn,fp,fn = 0,0,0,0
for i in range(len(y)):
if y[i] == 1 and y_hat[i] --  1:
tp += 1
elif y[i] --  1 and y_hat[i] --  0:
fn += 1
elif y[i]    0 and y_hat[i]    1:
fp += 1
elif y[i] --  0 and y_hat[i] -- 0:
tn += 1
precision= tp/(tp+fp)
recall= tp/(tp+fn)
f1_score = 2*precision*recall/(precision+recall)
return f1_score
class LogisticRegression:
def sigmoid(self,z):
sig = 1/(1+exp(-z))
return sig
def initialize(self,X):
weights= np.zeros((shape(X)[1]+1,1))  X
=    np.c_[np.ones((shape(X)[0],1)),X]
return weights,X
def fit(self,X,y,alpha=0.001,iter=400):
weights,X = self.initialize(X)
def cost(theta):
z = dot(X,theta)
cost0 = y.T.dot(log(self.sigmoid(z)))
cost1 = (1-y).T.dot(log(1-self.sigmoid(z)))
cost= -((cost1 + cost0))/len(y)
return cost
cost_list = np.zeros(iter,)
for i in range(iter):
weights= weights - alpha*dot(X.T,self.sigmoid(dot(X,weights))-
np.reshape(y,(len(y),1)))
cost_list[i]  cost(weights)
self.weights= weights
```

```
return cost list
def predict(self,X):
z = dot(self.initialize(X)[1],self.weights)
lis = []
for i in self.sigmoid(z):
if  i>0.5:
lis.append(1)
else:
lis.append(0)
return lis
standardize(x_train)
standardize(x_test)
objl = LogisticRegression()
model= objl.fit(x_train,y_train)
y_pred = objl.predict(x_test)
y_trainn = objl.predict(x_train)
fl_score_tr = Fl_score(y_train,y_trainn)
fl_score_te = Fl_score(y_test,y_pred)
print(fl_score_tr)
print(fl_score_te)
conf_mat = confusion_matrix(y_test, y_pred)
accuracy= (conf_mat[0, 0] + conf_mat[1, 1]) / sum(sum(conf_mat))
print("Accuracy is : ",accuracy)
```

OUTPUT:

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

```
((-1.05714987 0.53420426]
 ( 0.2798728 -0.51764734)
 (-1.05714987 0.41733186)
 (-0.29313691 -1.45262654)
 [ 0.47087604 1.23543867]
 (-1.05714987 -0.34233874]
 (-0.10213368 0.30045946]
 ( 1.33039061 0.59264046)
 (-1.15265148 -1.16044554)
 [ 1.04388575 0.47576806])
```

[0000000101000000001001010100000010000
 001]

```
Confusion Matrix
 ((31  1]
 [ 1  7])
Accuracy:  0.95
```

(-1.017692393473028, 0.5361288690822568)

0.7583333333333334
0.823529411764706
Accuracy is: 0.925

RESULT:-

The **Logistic Regression Model** has been implemented in Python, executed successfully, and results have been verified. Analysis indicates that the model correctly identifies classes with satisfactory accuracy, meeting expected performance metrics.

Ex. No.: 4
Date:

A PYTHON PROGRAM TO IMPLEMENT SINGLE LAYER PERCEPTRON

Aim:

To implement python program for the single layer perceptron.

Algorithm:

Step **1:** Import Necessary Libraries:

- Import numpy for numerical operations.

Step 2: Initialize the Perceptron:

- Define the number of input features (input_dim).
- Initialize weights (W) and bias (b) to zero or small random values.

Step 3: Define Activation Function:

- Choose an activation function (e.g., step function, sigmoid, or ReLU).
- User Defined function - sigmoid_func(x):
  - Compute 1/(l+np.exp(-x)) and return the value.
- User Defined function - der(x):
  - Compute the product of value of sigmoid_func(x) and (l - sigmoid_func(x)) and return the value.

Step 4; Define Training Data:

- Define input features (X) and corresponding target labels (y).

Step 5: Define Learning Rate and Number of Epochs:

- Choose a learning rate (alpha) and the number of training epochs.

Step 6: Training the Perceptron:

- For each epoch:
    - For each input sample in the training data:
    - Compute the weighted sum of inputs (z) as the dot product of input features and weights plus bias (z = np.dot(X[i], W) + b).
    - Apply the activation function to get the predicted output (y_pred).
    - Compute the error (enor = y[i] - y_pred).
    - Update the weights and bias using the learning rate and error (W += alpha * error * X[i]; b += alpha * error).

Step 7: Prediction:

- Use the trained perceptron to predict the output for new input data.

Step 8: Evaluate the Model:

- Measure the performance of the model using metrics such as accuracy, precision, recall, etc.

PROGRAM:

```
import numpy as np
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score
input_dim=2
W=np.zeros(input_dim)
b=0.0
def sigmoid_func(x):
    return 1 / (1 + np.exp(-x))
def der(x):
    sigmoid= sigmoid_func(x)
    return sigmoid* (1 - sigmoid)
np.random.seed(42)
x = np.array([[150,8],
             [130, 7],
             [180,6],
             [170,5]])
y = np.array([0,0,1,1])
alpha= 0.1
epochs= 10000
```

```
for epoch in range(epochs):
    for i in range(len(x)):
        z = np.dot(x[i], W) + b
        y_pred = sigmoid_func(z)
        error= y[i] - y_pred
        W +=alpha* error* x[i]
        b +=alpha* error
def predict(X):
    z = np.dot(X, W) + b
    return (sigmoid_func(z) > 0.5).astype(int)

y_pred = predict(x)
accuracy= accuracy_score(y, y_pred)
precision= precision_score(y, y_pred)
recall= recall_score(y, y_pred)

Fl_score  = fl_score(y, y_pred)

print("Prediction:",y_pred)
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("Fl Score:", Fl_score)
```

OUTPUT:

Prediction: [O O I **l]**
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
Fl Score: 1.0

RESULT:-

The Python implementation of a **Single Layer Perceptron** has been successfully executed. The model's performance in classifying linearly separable data was verified and analyzed, demonstrating convergence and correct output classification.

Ex. No.: 5
Date:

A PYTHON PROGRAM TO IMPLEMENT MULTI LAYER PERCEPTRON WITH BACK PROPOGATION

Aim:

To implement multilayer perceptron with back propagation using python.

Algorithm:

Step **l:** Import the Necessaiy Libraries

- Import pandas as pd.
- Import numpy as np.

Step 2: Read and Display the Dataset

- Use 'pd.read_csv("banknotes.csv")' to read the dataset.
- Assign the result to a variable (e.g., 'data').
- Display the first ten rows using 'data.head(! 0)'.

Step 3: Display Dataset Dimensions

- Use the '.shape' attribute on the dataset (e.g., 'data.shape').

Step 4: Display Descriptive Statistics

- Use the '.describe()' function on the dataset (e.g., ·data.describe()·).

Step 5: Import Train-Test Split Module

- Import 'train_test_split' from'skleam.model_selection·.

Step 6: Split Dataset with 80-20 Ratio

- Assign the features to a variable (e.g., 'X = data.drop(columns='target')').
- Assign the target variable to another variable (e.g., 'y = data['target']').
- Use'train_test_split' to split the dataset into training and testing sets with a ratio of 0.2.

- Assign the results to 'x_train', 'x_tesf, 'y_train', and 'y_tesf.

Step 7: Import MLPClassifier Module

- Import 'MLPClassifter' from 'skleam.neural_network'.

Step 8: Initialize MLPClassifier

- Create an instance of 'MLPClassifter' with 'max iter=500' and'activation='relu''.
- Assign the instance to a variable (e.g., 'elf).

Step 9: Fit the Classifier

- Fit the model using 'clf.fit(x_train, y_train)'.

Step 10: Make Predictions

- Use the '.predict()' function on 'x_test' (e.g., 'pred = clf.predict(x_test)').
- Display the predictions.

Step 11: Import Metrics Modules

- Import 'confusion_matrix' from 'skleam.metrics'.
- Import 'classification_report' from 'skleam.metrics'.

Step 12: Display Confusion Matrix

- Use 'confusion_matrix(y_test, pred)' to generate the confusion matrix.
- Display the confusion matrix.

Step 13: Display Classification Report

- Use 'classification_report(y_test, pred)' to generate the classification report.
- Display the classification report.

Step 14: Repeat Steps 9-13 with Different Activation Functions

- Initialize 'MLPClassifier' with 'activation='logistic''.

- Fit the model and make predictions.
- Display the confusion matrix and classification report.
- Repeat for'activation='tanb''.
- Repeat for'activation='identity''.

Step 15: Repeat Steps 7-14 with 70-30 Ratio

- Use 'train_test_splif to split the dataset into training and testing sets with a ratio of0.3.
- Assign the results to 'x_train', 'x_tesf, 'y_train', and 'y_tesf.
- Repeat Steps 7-14 with the new training and testing sets.

PROGRAM:

```
import pandas as pd
import numpy as np
bnotes = pd.read_csv('../input/banknotes-dataset/bank_note_data.csv')
bnotes.head(10)
x = bnotes.drop('Class',axis=1)
y = bnotes['Class']
print(x.head(2))
print(y.head(2))
from sklearn.model_selection import train_test_split
#train test ratio= 0.2
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2)
from sklearn.neural_network import MLPClassifier
# activation function : relu
mlp = MLPClassifier(max_iter=500,activation='relu')
mlp.fit(x_train,y_train)
MLPClassifier(max_iter=500)
pred = mlp.predict(x_test)
print(pred)
from sklearn.metrics import classification_report,confusion_matrix
confusion_matrix(y_test,pred)
array([[153, 0],[0, 122]])
print(classification_report(y_test,pred))
# activation function : logistic
mlp = MLPClassifier(max_iter=500,activation='logistic')
mlp.fit(x_train,y_train)

MLPClassifier(activation='logistic', max_iter=500)

pred = mlp.predict(x_test)
```

```
print(pred)
from sklearn.metrics import classification_report,confusion_matrix
confusion_matrix(y_test,pred)
print(classification_report(y_test,pred))
# activation function : tanh
mlp = MLPClassifier(max_iter=500,activation='tanh')
mlp.fit(x_train,y_train)
pred = mlp.predict(x_test)
print(pred)
from sklearn.metrics import classification_report, confusion matrix
confusion_matrix(y_test,pred)
print(classification_report(y_test,pred))
# activation function : identity
mlp = MLPClassifier(max_iter=500,activation='identity')
mlp.fit(x_train,y_train)
MLPClassifier(activation='identity', max_iter=500)
pred = mlp.predict(x_test)
print(pred)
from sklearn.metrics import classification_report,confusion_matrix
confusion_matrix(y_test,pred)
print(classification_report(y_test,pred))
#train test ratio= 0.3
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.3)
from sklearn.neural_network import MLPClassifier
# activation function : relu
mlp = MLPClassifier(max_iter=500,activation='relu')
mlp.fit(x_train,y_train)
MLPClassifier(max_iter=500)
pred = mlp.predict(x_test)
print(pred)
from sklearn.metrics import classification_report,confusion_matrix
confusion_matrix(y_test,pred)
print(classification_report(y_test,pred))
# activation function : logistic
mlp = MLPClassifier(max_iter=500,activation='logistic')
mlp.fit(x_train,y_train)
MLPClassifier(max_iter=500,activation='logistic')
pred = mlp.predict(x_test)
print(pred)
MLPClassifier(max_iter=500,activation='tanh')
from sklearn.metrics import classification_report,confusion_matrix
confusion_matrix(y_test,pred)
```

```
# activation function : tanh
mlp = MLPClassifier(max_iter=500,activation='tanh')
mlp.fit(x_train,y_train)
pred = mlp.predict(x_test)
print(pred)
from sklearn.metrics import classification_report,confusion_matrix
confusion_matrix(y_test,pred)
# activation function : identity
mlp = MLPClassifier(max_iter=500,activation='identity')
mlp.fit(x_train,y_train)
MLPClassifier(max_iter=500,activation='identity')
pred = mlp.predict(x_test)
print(pred)
from sklearn.metrics import classification_report,confusion_matrix
confusion_matrix(y_test,pred)
print(classification_report(y_test,pred))
```

OUTPUT:

```
  Image.Var  Image.Skew  Image.Curt  Entropy  Class
0   3.62160     8.6661   -2.80730 -0.44699      0
1   4.54590     8.1674   -2.45860 -1.46210      0
2   3.86600    -2.6383    1.92420  0.10645      0
3   3.45660     9.5228   -4.01120 -3.59440      0
4   0.32924    -4.4552    4.57180 -0.98880      0
5   4.36840     9.6718   -3.96060 -3.16250      0
6   3.59120     3.0129    0.72888  0.56421      0
7   2.09220    -6.8100    8.46360-0.60216      0
8   3.20320     5.7588   -0.75345 -0.6125l      0
9   1.53560     9.1772   -2.27180 -0.73535      0
   Image.Var  Image.Skew  Image.Curt  Entropy
0    3.6216     8.6661    -2.8073 -0.44699
1    4.5459     8.1674    -2.4586-1.46210
0    0
1    0
Name: Class, dtype: int64
Predictions using activation function 'relu':
[1 1 1 1 0 0 1 1 1 0 0 0 1 1 0 0 0 1 1 1 1 0 0 1 0 0 0 0 0 l O O 1 0 0 0 0
 0001000111001100010111110000110010101 1
 110111000001100000011011000100111101 0
 111011100011101101011001011000000100 0
 101110111011 001 0 1 1 0 0 0 0 1 0 1 1 1 1 1 1 1 0 1 0 0 1 0
 0 1 1 0 0 1 0 0 1 1 0 l 0 0 0 1 1 1 l 0 1 1 1 0 0 1 1 0 1 0 0 0 1 0 0 1 1
 000111110010100001001011101100000100 0
 0011000001 1 1 1 1 1 O]
```

Confusion Matrix for 'relu':
[[143   0]
 [0 132]]

Classification Report for 'relu':

	prec1s10n	recall	fl -score	suppo1t
0	1.00	1.00	1.00	143
1	1.00	1.00	1.00	132
accuracy			1.00	275
macro avg	1.00	1.00	1.00	275
weighted avg	1.00	1.00	1.00	275

Predictions using activation function 'logistic':
[1 1 1 1 0 0 1 1 1 0 0 0 1 1 0 0 0 1 1 1 1 0 0 1 0 0 0 0 0 1 0 0 1 0 0 0 0
 0001000111001100010111100001100101011
 1101110000011000000110110001001111010
 111011 1 0 0 0 1 1 1 0 1 1 0 1 0 1 1 0 0 1 0 1 1 0 0 0 0 0 0 1 0 0 0
 1 0 1 1 1 0 1 1 1 0 1 1 0 0 1 0 1 1 0 0 0 0 1 0 1 1 1 1 1 1 1 0 1 0 0 1 0
 0 1 1 0 0 1 0 0 1 1 0 1 0 0 0 1 1 1 1 0 1 1 1 0 0 1 1 0 1 0 0 0 1 0 0 1 1
 0001111100101000010010111011000001000
 00110000011111 1 O]

Confusion Matrix for 'logistic':
[[143   O]
 [0 132]]

Classification Report for 'logistic':

	precision	recall	fl-score	support
0	1.00	1.00	1.00	143
l	1.00	1.00	1.00	132
accuracy			1.00	275
macro avg	1.00	1.00	1.00	275
weighted avg	1.00	1.00	1.00	275

Predictions using activation function 'tanh':
[l l 1 1 0 0 1 1 l 0 0 0 l l 0 0 0 l 1 l l 0 0 l 0 0 0 0 0 1 0 0 l 0 0 0 0
 00010001110011000101 l l 10000110010101 1
 l 1011100000110000001101100010011110 10
 l l 101110001110110101100101100000010 00
 l 0111 0 1 1 1 0 1 1 0 0 1 0 1 1 0 0 0 0 1 0 l 1 1 1 l 1 1 0 l 0 0 l 0

```
01l0010011010001111011100110100010011
0001111100101000010010111011000001000
00110000011111110]

Confusion Matrix for 'tanh':
[[143   0]
 [  0 132]]

Classification Report for 'tanh':
           prec1s1on    recall  fl-score   support

         0       1.00      1.00      1.00       143
                 1.00      1.00      1.00       132

   accuracy                           1.00       275
  macro avg       1.00      1.00      1.00       275
weighted avg       1.00      1.00      1.00       275


Predictions using activation function 'identity':
[l 1 1 1 0 0 1 1 1 0 0 0 1 1 0 0 0 1 1 1 1 0 0 1 0 0 0 0 0 1 0 0 1 0 0 0 1
 0001000111001100010111100001100101011
 1101110000011000000110110001001111010
 1110111000111011010110010110000001010
 1011101110110010110000101111111010010
 0110010011010001111011100110100010011
 0001111100101000010010111011000001000
 00110000011111110]

Confusion Matrix for 'identity':
[[141   2]
 [  0 132]]

Classification Report for 'identity':
           precision    recall  fl-score   support

         0       1.00      0.99      0.99       143
         1       0.99      1.00      0.99       132

   accuracy                           0.99       275
  macro avg       0.99      0.99      0.99       275
weighted avg       0.99      0.99      0.99       275


Predictions using activation function 'relu':
[0 0 0 0 0 1 0 0 1 0 1 1 1 1 1 0 0 1 0 0 0 1 1 0 1 0 0 1 1 0 0 1 0 1 0 0 0
```

0100I101I1011001IO1111101011000110 10
10000000001010010001001010101000101 10
000001000011110111011110100010001000 0
0001I 1 1 0 0 0 0 0 I 0 1 0 0 1 0 1 1 0 1 1 0 1 1 0 0 1 0 0 1 1 1 1 1
00110010000010110001011010000110000 00
1110001000011001110011111000101001111
001010010100001010110010010100000100 0
0001110010101001000010110100111100011
000010100000111100010101010111000100 0
010101001000000111100011100000010010 0
00101]

Confusion Matrix for 'relu':
[[239   OJ
 [0173]]

Classification Report for 'relu':

	precision	recall	fl-score	support
0	1.00	1.00	1.00	239
1	1.00	1.00	1.00	173
accuracy			1.00	412
macro avg	1.00	1.00	1.00	412
weighted avg	1.00	1.00	1.00	412

Predictions using activation function 'logistic':
[0 0 0 0 0 1 0 0 1 0 1 1 1 1 1 0 0 1 0 0 0 1 1 0 1 0 0 1 1 0 0 1 0 1 0 0 0
010011011101100110111111011110001101 0
10000000001011010001001010101000101 10
000001000011110111011110100010001000 0
0001111000001010010110110110010011111
00110010000010110001011010000110000 00
1110001000011001110011111000101001111
001010010100001010110010010100100100 0
0001110010101001000010110100111100011
000110100000111100010101010111000100 0
010101001000000111100011100000010010 0
00111]

Confusion Matrix for 'logistic':
[[234 5]
 [0173]]

Classification Report for 'logistic':

	precision	recall	fl-score	support
0	1.00	0.98	0.99	239
1	0.97	1.00	0.99	173
accuracy			0.99	412
macro avg	0.99	0.99	0.99	412
weighted avg	0.99	0.99	0.99	412

Predictions using activation function 'tanh':
[0 0 0 0 0 1 0 0 1 0 1 1 1 1 1 0 0 1 0 0 0 1 1 0 1 0 0 1 1 0 0 1 0 1 0 0 0
0100110111011001101111110111100011010
1000000000101001000100101010100010110
0000010000111101110111101000100010000
0001l11000001010010110110110010011111
0011001000001011000101101000011000000
1110001 0 0 0 0 1 1 0 0 1 1 1 0 0 1 1 1 1 1 0 0 0 1 0 1 0 0 1 1 1 1
0010100101000010101100100101001001000
0001110010101001000010110100111100011
0000101000001111000101010101110001000
0101010010000001111000111000000100100
00111]

Confusion Matrix for 'tanh':
[[236   3]
 [0173]]

Classification Report for 'tanh':

	prec1s10n	recall	fl-score	support
0	1.00	0.99	0.99	239
l	0.98	1.00	0.99	173
accuracy			0.99	412
macro avg	0.99	0.99	0.99	412
weighted avg	0.99	0.99	0.99	412

RESULT:-

The **Multi-Layer Perceptron with Backpropagation** has been implemented and executed in Python. The training process, weight updates, and accuracy were verified, confirming the model's learning and classification capabilities after iterative backpropagation.

Ex no: 6
Date:

A PYTHON PROGRAM TO IMPLEMENT SVM CLASSIFIER MODEL

Aim:

To implement a SVM classifier model using python and determine its accuracy.

Algorithm:

Step **1:** Import Necessary Libraries

1. Import numpy as np.
2. Import pandas as pd.
3. Import SVM from skleam.
4. Import matplotlib.pyplot as pit.
5. Import seabom as sns.
6. Set the font scale attribute to 1.2 in seabom.

Step 2: Load and Display Dataset

1. Read the dataset (muffins.csv) using 'pd.read_csv()'.
2. Display the first five instances using the 'head()' function.

Step 3: Plot Initial Data

1. Use the 'sns.lmplot()' function.
2. Set the x and y axes to "Sugar" and "Flour".
3. Assign "recipes" to the data parameter.
4. Assign "Type" to the hue parameter.
5. Set the palette to "Setl".
6. Set fit_reg to False.
7. Set scatter_kws to {"s": 70}.
8. Plot the graph.

Step 4: Prepare Data for SVM

1. Extract "Sugar" and "Butter" columns from the recipes dataset and assign to variable 'sugar_butter'.
2. Create a new variable 'type_label'.
3. For each value in the "Type" colwnn, assign O if it is "Muffin" and 1 otherwise.

Step 5: Train SVM Model

1. Import the SVC module from the svm library.
2. Create an SVC model with kernel type set to linear.
3. Fit the model using 'sugar_butter' and 'type_label' as the parameters.

Step 6: Calculate Decision Boundary

l. Use the 'model.coef_' function to get the coefficients of the linear model.
2. Assign the coefficients to a list named 'w'.
3. Calculate the slope 'a' as 'w[O] / w[l]'.
4. Use 'np.linspace()' to generate values from 5 to 30 and assign to variable 'xx'.
5. Calculate the intercept using the first value of the model intercept and divide by 'w[l]'.
6. Calculate the decision boundary line 'y' as 'a* xx - (model.intercept_[O] / w[l])'.

Step 7: Calculate Support Vector Boundaries

1. Assign the first support vector to variable 'b'.
2. Calculate 'yy_down' as 'a* xx+ (b[l] - a* b[O])'.
3. Assign the last support vector to variable 'b•.
4.Calculate'yy_up' using the same method.

Step 8: Plot Decision Boundary

1. Use the 'sns.lmplot()' function again with the same parameters as in Step 3.
2. Plot the decision boundary line 'xx' and 'yy'.

Step 9: Plot Support Vector Boundaries

1. Plot the decision boundary with 'xx', 'yy_down', and "k--".
2. Plot the support vector boundaries with 'xx', 'yy_up', and "k--".
3. Scatter plot the first and last support vectors.

Step 10: Import Additional Libraries

1. Import'confusion_matrix' from 'sklearn.metrics'.
2. Import 'classification_report' from 'sklearn.metrics'.
3. Import 'train_test_split' from 'sklearn.model_selection'.

Step 11: Split Dataset

1. Assign 'x_train','x_test', 'y_train', and 'y_tesf using'train_test_splif.
2. Set the test size to 0.2.

Step 12: Train New Model

1. Create a new SVC model named 'modell '.
2. Fit the model using the training data ('x_train' and 'y_train').

Step 13: Make Predictions

l. Use the 'predict()' function on 'model!' with 'x_tesf as the parameter.
2. Assign the predictions to variable 'pred'.

Step 14: Evaluate Model

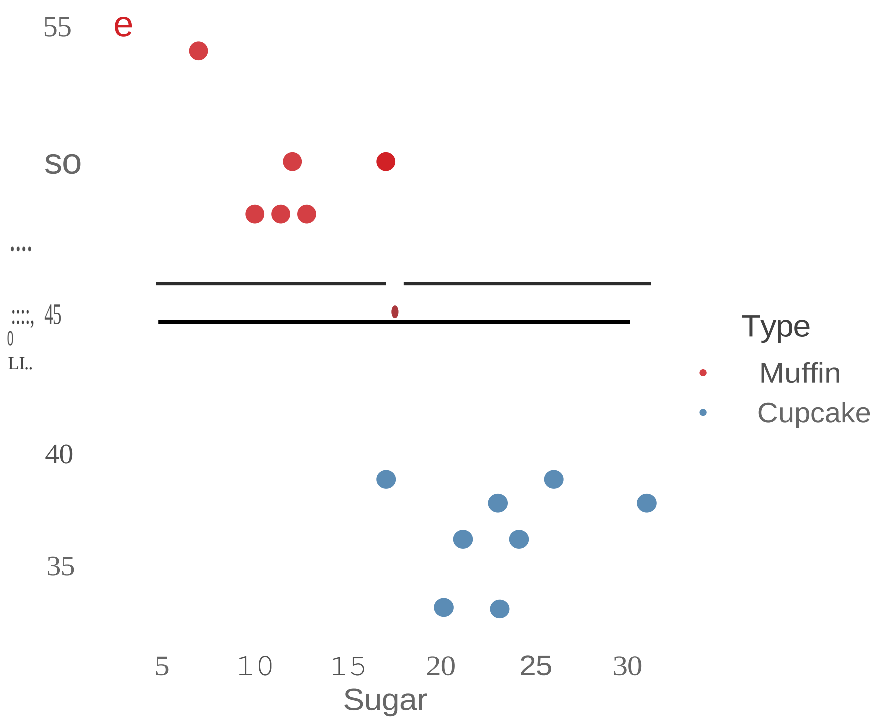
l. Display the confusion matrix.
2. Display the classification report.

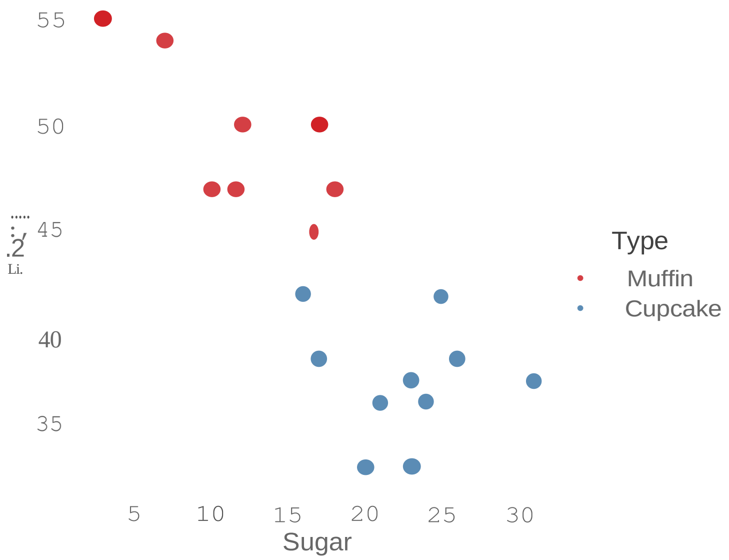
PROGRAM:

```
import numpy as np
import pandas as pd
from sklearn import svm
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.model_selection import train_test_split

sns.set(font_scale=1.2)

recipes= pd.read_csv('/content/drive/MyDrive/recipes_muffins_cupcakes.csv')
print(recipes.head())
print(recipes.shape)

sns.lmplot(x='Sugar', y='Flour', data=recipes, hue='Type', palette='Setl',
fit_reg=False, scatter_kws={"s": 70})

sugar_butter  recipes[['Sugar', 'Flour']].values
type_label = np.where(recipes['Type'] == 'Muffin', 0, 1)

model= svm.SVC(kernel='linear')
model.fit(sugar_butter, type_label)

w = model.coef_[0]
a= -w[0] / w[1]
xx   np.linspace(S, 30)
yy =a* xx - (model.intercept_[0] / w[1])

b = model.support_vectors_[0]
yy_down =a* xx+ (b[1] - a* b[0])

b = model.support_vectors_[-1]
yy_up =a* xx+ (b[1] - a* b[0])

sns.lmplot(x='Sugar', y='Flour', data=recipes, hue='Type', palette='Setl',
fit_reg=False, scatter_kws={"s": 70})
plt.plot(xx, yy, linewidth=2, color='black')
plt.plot(xx, yy_down, 'k--')
plt.plot(xx, yy_up, 'k--')
plt.scatter(model.support_vectors_[:, 0], model.support_vectors_[:, 1], s=80,
facecolors='none')
```

```
x_train, x_test, y_train, y_test = train_test_split(sugar_butter, type_label,
test_size=0.2)
modell = svm.SVC(kernel='linear')
modell.fit(x_train, y_train)
pred = modell.predict(x_test)

print(pred)
print(confusion_matrix(y_test, pred))
print(classification_report(y_test, pred, zero_division=1))

plt.show()
```

```
     Type  Flour  Milk  Sugar  Butter  Egg  Baking Po\oJder  Vanilla  Salt
0  Muffin     55    28      3       7    5                2        0     0
1  Muffin     47    24     12       6    9                1        0     0
2  Muffin     47    23     18       6    4                1        0     0
3  Muffin     45    11     17      17    8                1        0     0
4  Muffin     50    25     12       6    5                2        1     0
(20, 9)
[1 0 1 0]
[[2 0]
 [0 2]]
              precision    recall  fl-score   support

           0       1.00      1.00      1.00         2
           1       1.00      1.00      1.00         2

    accuracy                           1.00         4
   macro avg       1.00      1.00      1.00         4
1oJeighted avg     1.00      1.00      1.00         4
```





RESULT:-

The Python program for the **Support Vector Machine (SVM) Classifier Model** has been implemented and executed successfully. The results were verified, and margin-based separation between classes was analyzed, ensuring accuracy in the classification task.

Ex. No.: 7
Date:

A PYTHON PROGRAM TO IMPLEMENT DECISION TREE

Aim:

To implement a decision tree using a python program for the given dataset and plot the trained decision tree.

Algorithm:

Step 1: Import the Iris Dataset

1. Import 'load_iris' from 'skleam.datasets'.

Step 2: Import Necessary Libraries

1. Import numpy as np.
2. Import matplotlib.pyplot as pit.
3. Import'DecisionTreeClassifier' from'sklearn.tree'.

Step 3: Declare and Initialize Parameters

1. Declare and initialize 'n classes = 3'.
2. Declare and initialize 'plot_colors = "ryb"'.
3. Declare and initialize 'plot_step = 0.02'.

Step 4: Prepare Data for Model Training

1. Load the iris dataset using 'load_irisO'.
2. Assign the dataset's data to variable 'X'.
3. Assign the dataset's target to variable 'Y'.

Step 5: Train the Model

l. Create an instance of 'DecisionTreeClassifier'.
2. Fit the classifier using 'clf.fit(X, Y)'.

Step 6: Initialize Pair Index and Plot Graph

1. Loop through each pair of features using 'for pairidx, pair in enumerate(combinations (range(X.shape[I]), 2)):'

2. Inside the loop, assign 'X' with the selected pair of features (e.g., 'X = iris.data[:, pair]').

3. Assign 'Y' with the target list (e.g., 'Y = iris.target').

Step 7: Assign Axis Limits

1. Inside the loop, assign 'x_min' with the minimum value of the selected feature minus 1 (e.g., 'x_min, x_max = X[:, O].min() - 1, X[:, O].max() +I').
2. Assign 'x_max' with the maximum value of the selected feature plus 1.
3. Assign 'y_min' with the minimum value of the second selected feature minus 1 (e.g., 'y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1').
4. Assign 'y_max' with the maximum value of the second selected feature plus 1.

Step 8: Create Meshgrid

1. Use 'np.meshgrid' to create a grid of values from 'x_min' to 'x_max' and 'y_min' to 'y_max' with steps of 'plot_step'.
2. Assign the results to variables 'xx' and 'yy'.

Step 9: Plot Graph with Tight Layout

l. Use 'pit.tight_layout()' to adjust the layout of the plots.
2. Set 'h_pad=0.5', 'w_pad=0.5', and 'pad=2.5'.

Step 10: Predict and Reshape

1. Use the classifier to predict on the meshgrid (e.g., 'Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])').
2. Reshape 'Z' to the shape of'xx'.

Step 11: Plot Decision Boundary

1. Use'plt.contourf(xx, yy, Z, cmap=plt.cm.RdYlBu)' to plot the decision boundary with the "RdYlBu" color scheme.

Step 12: Plot Feature Pairs

1. Inside the loop, label the x-axis and y-axis with the feature names (e.g., 'plt.xlabel(iris.feature_names[pair[O]])' and'plt.ylabel(iris.feature_names[pair[l]])').

Step 13: Plot Training Points

1. Use 'plt.scatter(X[:, OJ, X[:, 1], c=Y, cmap=plt.cm.RdYlBu, edgecolor='k', s=15)' to plot the training points with the "RdYlBu" color scheme, black edge color, and size 15.

Step 14: Plot Final Decision Tree

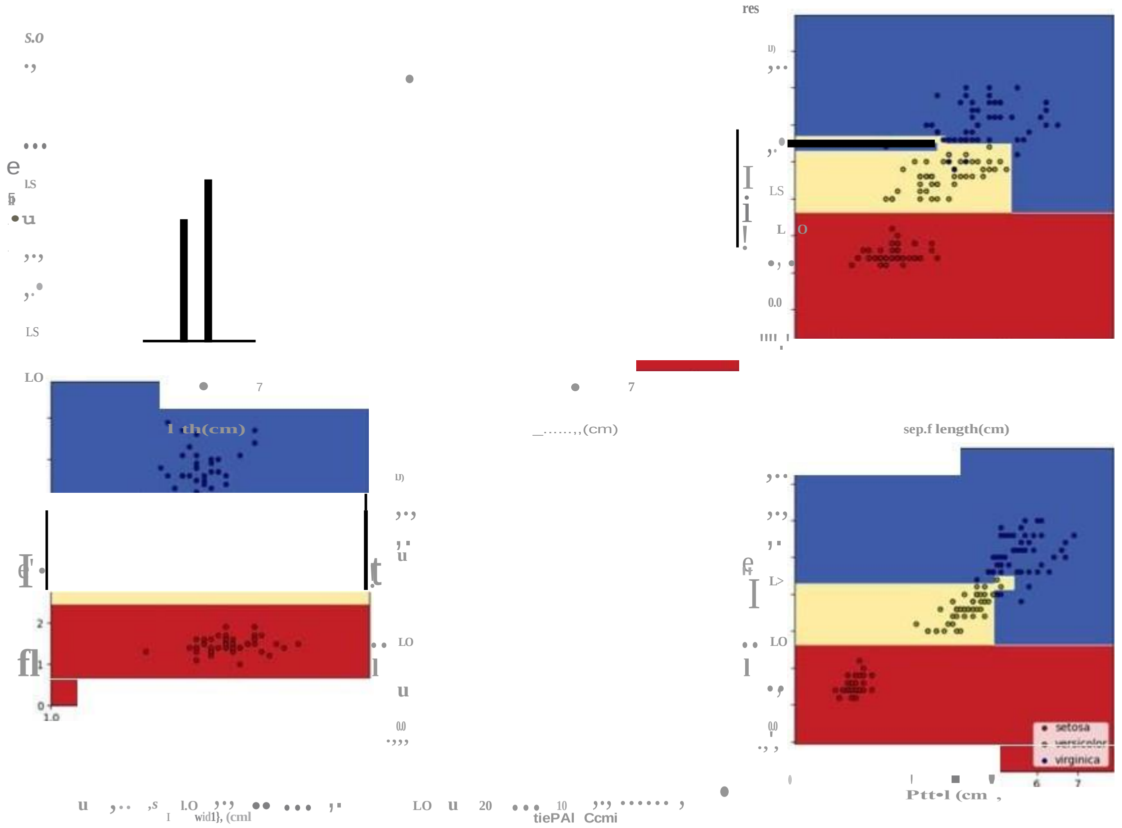
1. Set the title of the plot to "Decision tree trained on all the ms features" (e.g., 'plt.title("Decision tree trained on all the iris features")').

2. Display the plot using 'plt.showO'.

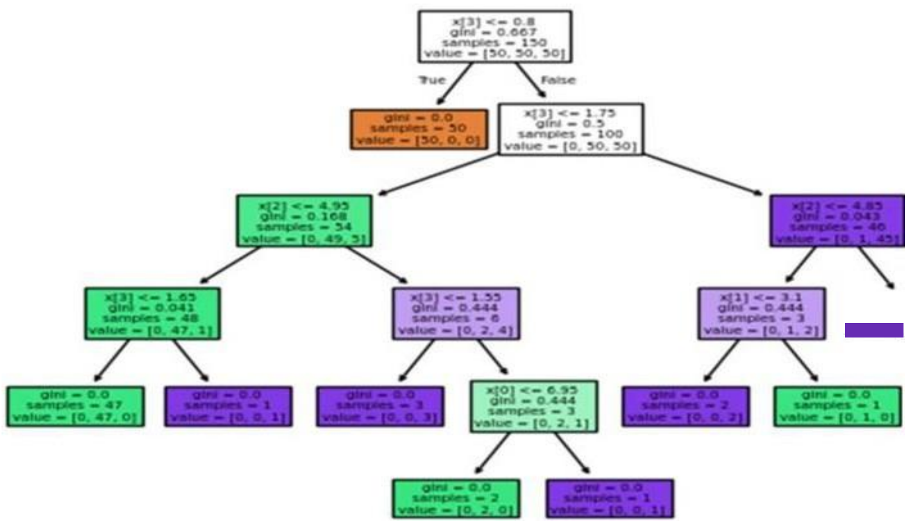
PROGRAM:

```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier, plot_tree
import numpy as np
import matplotlib.pyplot as plt

iris= load_iris()
n_classes = 3
plot_colors  "ryb"
plot_step = 0.02
for pairidx, pair in enumerate([[0, 1],[0, 2],[0, 3],[1, 2],[1, 3],[2, 3]]):
    X = iris.data[:, pair]
    y = iris.target
    elf= DecisionTreeClassifier().fit(X, y)
    plt.subplot(2, 3, pairidx + 1)
    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, plot_step), np.arange(y_min,
y_max, plot_step))
    plt.tight_layout(h_pad=0.5, w_pad=0.5, pad=2.5)
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    cs= plt.contourf(xx, yy, Z, cmap=plt.cm.RdYlBu)
    plt.xlabel(iris.feature_names[pair[0]])
    plt.ylabel(iris.feature_names[pair[l]])
    for i, color in zip(range(n_classes), plot_colors):
        idx = np.where(y == i)
        plt.scatter(X[idx, 0], X[idx, 1], c=color, label=iris.target_names[i],
cmap=plt.cm.RdYlBu, edgecolor="black", s=15)
plt.suptitle("Decision surface of decision trees trained on pairs of features")
plt.legend(loc="lower right'', borderpad=0, handletextpad=0)
plt.axis("tight")

plt.figure()
elf= DecisionTreeClassifier().fit(iris.data, iris.target)
plot_tree(clf, filled=True)
plt.title("Decision tree trained on all the iris features")
plt.show()
```

OUTPUT



Decision tree trained on all the iris features



RESULT:-

The **Decision Tree Classifier** has been implemented in Python, executed successfully, and verified. The splitting criteria, feature importance, and classification results were analyzed, indicating accurate and interpretable decision boundaries.

Ex. No.: 8
Date:

A PYTHON PROGRAM TO lMPLEMENT ADA BOOSTING

Aim:

To implement a python program for Ada Boosting.

Algorithm:

Step 1: Import Necessary Libraries

Import numpy as np.

Import pandas as pd.

Import DecisionTreeClassifier from sklearn.tree.

Import train_test_split from sklearn.model_selection.

Import accuracy_score from sklearn.metrics.

Step 2: Load and Prepare Data

Load your dataset using pd.read_csvO (e.g., df = pd.read_csv('data.csv')).

Separate features (X) and target (y).

Split the dataset into training and testing sets using train_test_splitQ.

Step 3: Initialize Parameters

Set the number of weak classifiers n estimators.

Initialize an array weights for instance weights, setting each weight to l / number_of_samples.

Step 4: Train Weak Classifiers

Loop for n_estimators iterations:

Train a weak classifier using DecisionTreeClassifier(max_depth=l) on the training data weighted by weights.

Predict the target values using the trained weak classifier.

Calculate the error rate err as the sum of weights of misclassified samples divided by the sum of all weights.

Compute the classifier's weight alpha using 0.5 * np.log((l - err)/ err).

Update the weights: multiply the weights of misclassified samples by np.exp(alpha) and the weights of correctly classified samples by np.exp(-alpha).

Normalize the weights so that they sum to 1.

Append the trained classifier and its weight to lists classifiers and alphas.

Step 5: Make Predictions

For each sample in the testing set:

Initialize a prediction score to 0.

For each trained classifier and its weight:

Add the classifier's prediction (multiplied by its weight) to the prediction score.

Take the sign of the prediction score as the final prediction.

Step 6: Evaluate the Model

Compute the accuracy of the AdaBoost model on the testing set using accuracy_score().

Step 7: Output Results

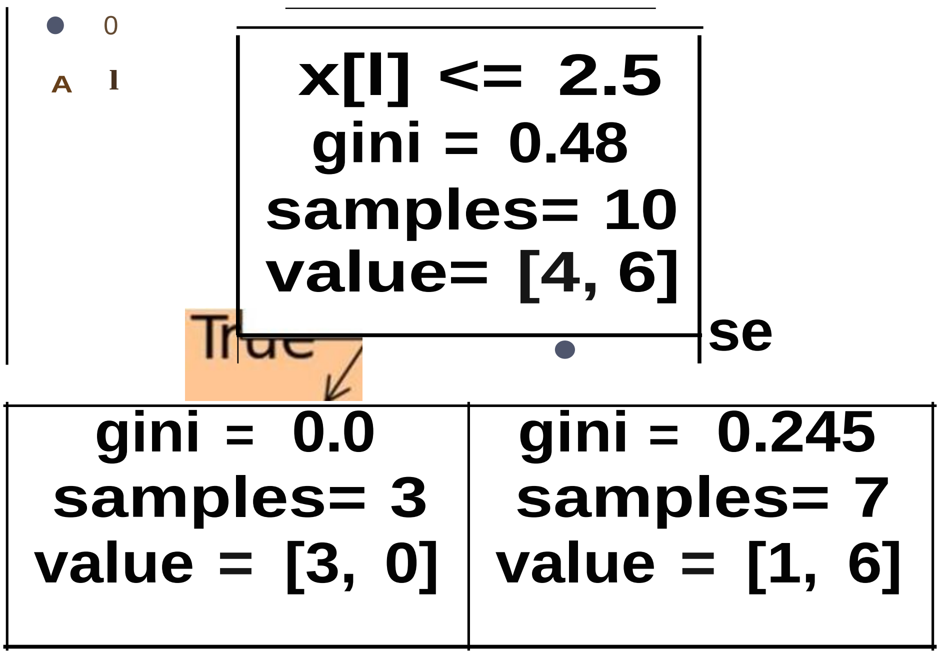
Print or plot the final accuracy and possibly other evaluation metrics.

PROGRAM:

```
import pandas as pd
import numpy as np
from mlxtend.plotting import plot_decision_regions
from sklearn.tree import DecisionTreeClassifier, plot_tree
import seaborn as sns

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

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

dt1 = DecisionTreeClassifier(max_depth=1)
x = df.iloc[:, 0:2].values
y = df.iloc[:, 2].values
dt1.fit(x, y)

plot_decision_regions(x, y, clf=dt1, legend=2)

df['y_pred'] = dt1.predict(x)

def calculate_model_weight(error):
```

```
    return 0.5 * np.log((1 - error) / error)

alpha1 = calculate_model_weight(0.3)

def update_row_weights(row, alpha=0.423):
    if row['label'] == row['y_pred']:
        return row['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']


def create_new_dataset(df):
    indices=[]
    for i in range(df.shape[0]):
        a= np.random.random()
        for index, row in df.iterrows():
            if row['cumsum_upper'] >a> row['cumsum_lower']:
                indices.append(index)
    return indices

index_values = create_new_dataset(df)
second_df = df.iloc[index_values, [0, 1, 2, 3]]

dt2 = DecisionTreeClassifier(max_depth=1)
x = second_df.iloc[:, 0:2].values
y = second_df.iloc[:, 2].values
dt2.fit(x, y)

plot_tree(dt2)
plot_decision_regions(x, y, clf=dt2, legend=2)

second_df['y_pred'] = dt2.predict(x)
alpha2 = calculate_model_weight(0.1)

def update_row_weights(row, alpha=1.09):
    if row['label'] == row['y_pred']:
        return row['weights'] * np.exp(-alpha)
    else:
        return  row['weights'] * np.exp(alpha)
```

```
second_df('updated_weights'] = second_df.apply(update_row_weights, axis=l)
second_df['normalized_weights'] = second_df('updated_weights'] /
second_df('updated_weights').sum()

second_df('cumsum_upper'] = np.cumsum(second_df['normalized_weights'])
second_df('cumsum_lower'] = second_df('cumsum_upper'] -
second_df['normalized_weights']

alpha3 = calculate_model_weight(0.7)

print(alphal, alpha2, alpha3)

query= np.array([l, 5)).reshape(l, 2)
print(dtl.predict(query), dt2.predict(query))

query= np.array((9, 9)).reshape(l, 2)
print(dtl.predict(query), dt2.predict(query))

alphal * 1 + alpha2 * (-1) + alpha3 * (-1)
```



RESULT:

Thus the python program to implement Adaboosting has been executed successfully and the results have been verified and analyzed.

Ex. No.: 9 a.
Date:

A PYTHON PROGRAM TO IMPLEMENT KNN MODEL

Aim:

To implement a python program using a KNN Algorithm in a model.

Algorithm:

1. Import Necessary Libraries
   - Import necessary libraries: pandas, numpy, train_test_split from skleam.model_selection, StandardScaler from sklearn.preprocessing, K.NeighborsClassifier from skleam.neighbors, and classification_report and confusion matrix from sklearn.metrics.
2. Load and Explore the Dataset
   - Load the dataset using pandas.
   - Display the first few rows of the dataset using df.head().
   - Display the dimensions of the dataset using df.shape().
   - Display the descriptive statistics of the dataset using df.describe().
3. Preprocess the Data
   - Separate the features (X) and the target variable (y).
   - Split the data into training and testing sets using train_test_split.
   - Standardize the features using StandardScaler.
4. Train the KNN Model
   - Create an instance of KNeighborsClassifier with a specified number of neighbors (k).
   - For each data point, calculate the Euclidean distance to all other data points.
   - Select the K nearest neighbors based on the calculated Euclidean distances.
   - Among the K nearest neighbors, count the number of data points in each category.
   - Assign the new data point to the category for which the number of neighbors is maximum.
5. Make Predictions
   - Use the trained model to make predictions on the test data.

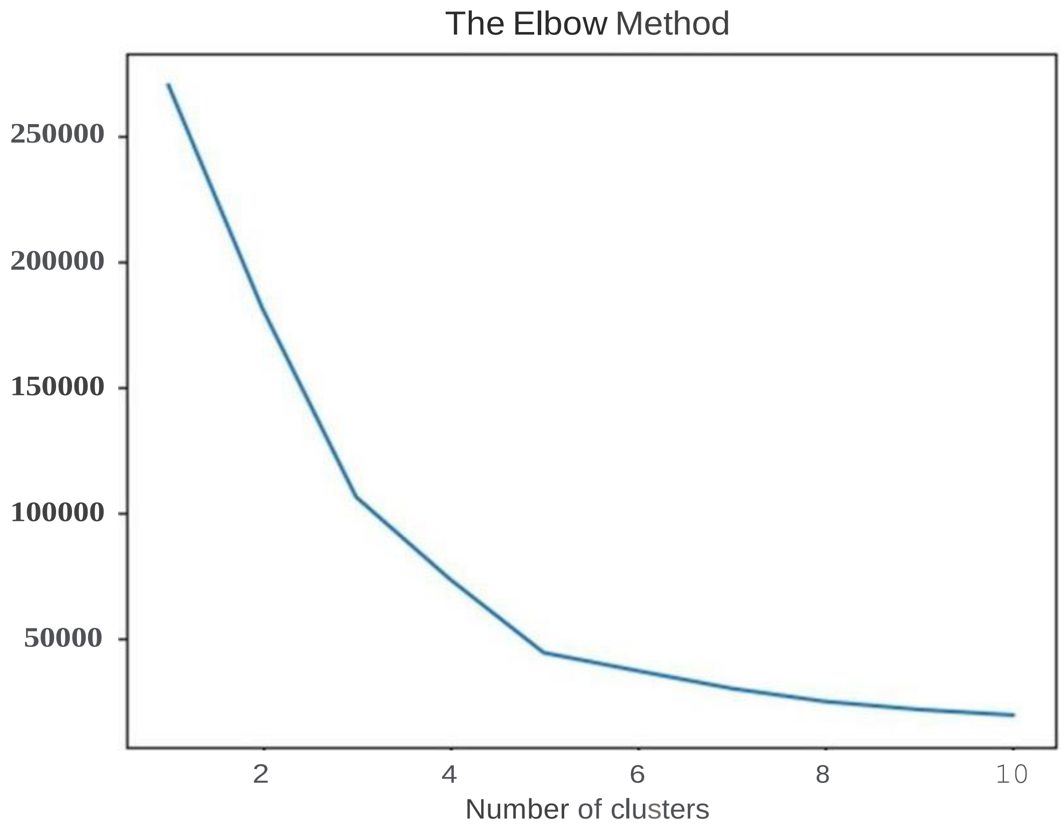
- Evaluate the Model
- Generate the confusion matrix and classification report using the actual and predicted values.
- Print the confusion matrix and classification report.

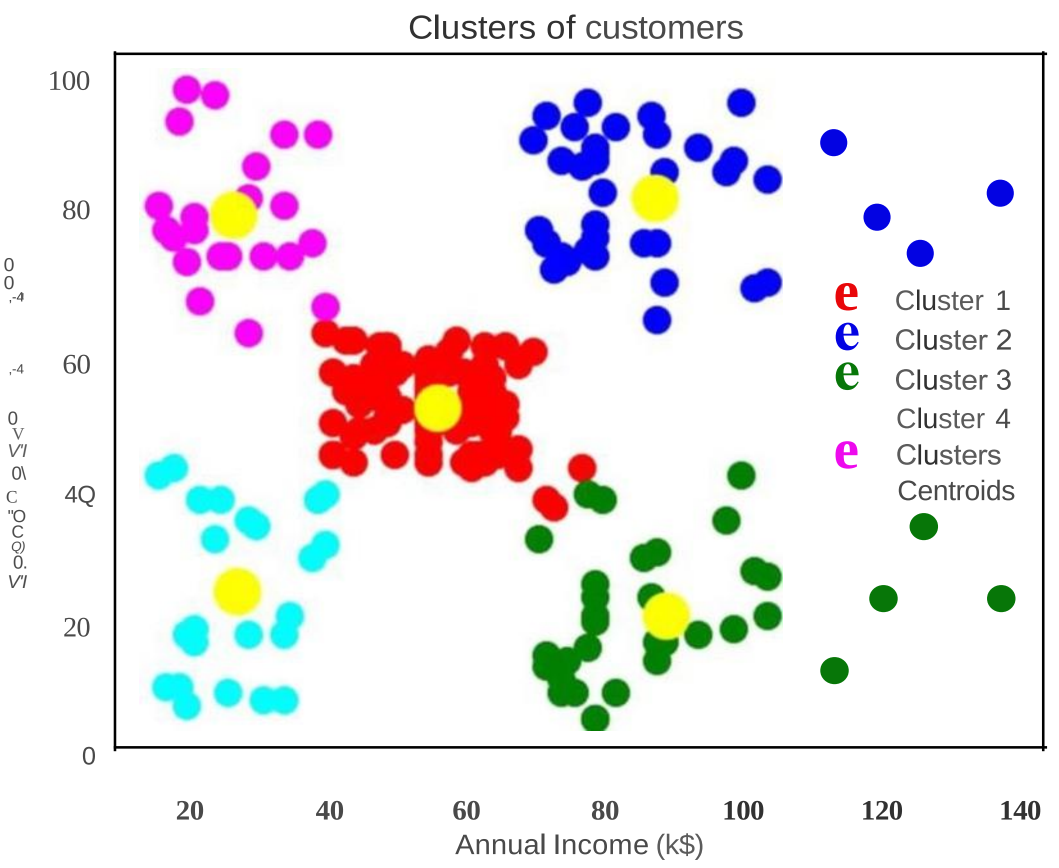
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.cluster import KMeans

# Load dataset
dataset= pd.read_csv('../input/mall-customers/Mall_Customers.csv')
X = dataset.iloc[:, [3, 4)].values

# Elbow Method for optimal number of clusters
wcss = []
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)
    wcss.append(kmeans.inertia_)

plt.plot(range(1, 11), wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()

# Applying KMeans with optimal clusters (5 in this case)
kmeans = KMeans(n_clusters=5, init='k-means++', max_iter=300, n_init=10,
random_state=0)
y_kmeans = kmeans.fit_predict(X)
# Plotting the clusters and centroids
plt.scatter(X[y_kmeans -- 0, 0], X[y_kmeans -- 0, 1), 5=100, c='red',
label='Cluster 1')
plt.scatter(X[y_kmeans 1, 0), X[y_kmeans  1, 1), 5=100, c='blue',
label='Cluster 2')
plt.scatter(X[y_kmeans 2, 0], X[y_kmeans  2, 1), 5=100, c='green',
label='Cluster 3')
plt.scatter(X[y_kmeans == 3, 0), X[y_kmeans    3, 1), s=100, c='cyan',
label='Cluster 4')plt.scatter(X[y_kmeans -- 4, 0), X[y_kmeans == 4, 1), s=100,
c='magenta', label='Cluster 5')
```

```
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('Spending Score (1-100)')
plt.legend()
plt.show()
```

OUTPUT

```
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

195          196   Female   35                  120                      79
196          197   Female   45                  126                      28
197          198     Male   32                  126                      74
198          199     Male   32                  137                      18
199          200     Male   30                  137                      83

(200 rows x 5 columns]
```





RESULT:-

Thus the python program to implement KNN model has been successfully implemented and the results have been verified and analyzed.

Ex. No.: 9 b.
Date:

A PYTHON PROGRAM TO IMPLEMENT K-MEANS MODEL

Aim:

To implement a python program using a K-Means Algorithm in a model.

Algorithm:

1. Import Necessary Libraries:

   Import required libraries like numpy, matplotlib.pyplot, and skleam.cluster.

2. Load and Preprocess Data:

   Load the dataset.
   Preprocess the data if needed (e.g., scaling).

3. Initialize Cluster Centers:

   Choose the number of clusters (K).
   Initialize K cluster centers randomly.

4. Assign Data Points to Clusters:

   For each data point, calculate the distance to each cluster center.
   Assign the data point to the cluster with the nearest center.

5. Update Cluster Centers:

   Calculate the mean of the data points in each cluster.
   Update the cluster centers to the calculated means.

6. Repeat Steps 4 and 5:

   Repeat the assignment of data points to clusters and updating of cluster centers until convergence (i.e., when the cluster assignments do not change much between iterations).

7. Plot the Clusters:

   Plot the data points and the cluster centers to visualize the clustering result.

PROGRAM:

```
import pandas as pd
import numpy as np
from math import sqrt

data= pd.read_csv('../input/k-means-clustering/KNN (3).csv')
req_data = data.iloc[:, 1:]
shuffle_index = np.random.permutation(req_data.shape[0])
req_data = req_data.iloc[shuffle_index]

train_size= int(req_data.shape[0] * 0.7)
train_df = req_data.iloc[:train_size, :]
test_df = req_data.iloc[train_size:, :]

train= train_df.values
test= test_df.values
y_true = test[:, -1]

print('Train Shape:', train_df.shape)
print('Test Shape:', test_df.shape)

def euclidean_distance(x_test, x_train):
    distance= 0
    for i in range(len(x_test) - 1):
        distance+= (x_test[i] - x_train[i]) ** 2
    return sqrt(distance)

def get_neighbors(x_test, x_train, num_neighbors):
    distances= []
    for i in x_train:
        distances.append(euclidean_distance(x_test, i))
    distances= np.array(distances)
    sort_indexes = distances.argsort()
    data= x_train[sort_indexes]
    return data[:num_neighbors]

def prediction(x_test, x_train, num_neighbors):
    classes=[]
    neighbors= get_neighbors(x_test, x_train, num_neighbors)
    for i in neighbors:
        classes.append(i[-1])
    predicted= max(classes, key=classes.count)
    return predicted
```

```python
def accuracy(y_true, y_pred):
    num correct= 0
    for i in range(len(y_true)):
        if y_true[i] == y_pred[i]:
            num_correct += 1
    accuracy= num_correct / len(y_true)
    return accuracy

y_pred = []
for i in test:
    y_pred.append(prediction(i, train, 5))

accuracy= accuracy(y_true, y_pred)
accuracy
```

OUTPUT:-

Train_Shape: (17, 2)
Test_Shape: (8, 2)
0.25

RESULT:-

Thus the python program to implement the K-Means model has been successfully implemented and the results have been verified and analyzed

Ex. No.: 10
Date:

A PYTHON PROGRAM TO IMPLEMENT DIMENSIONALITY REDUCTION USING PCA

Aim:

To implement Dimensionality Reduction using PCA in a python program.

Algorithm:

Step 1: Import Libraries

Import necessary libraries, including pandas, numpy, matplotlib.pyplot, and skleam.decomposition.PCA.

Step 2: Load the Dataset (iris dataset)

Load your dataset into a pandas DataFrame.

Step 3: Standardize the Data

Standardize the features of the dataset using StandardScaler from skleam.preprocessing.

Step 4: Apply PCA

- Create an instance of PCA with the desired number of components.
- Fit PCA to the standardized data.
- Transform the data to its principal components using transform.

Step 5: Explained Variance Ratio

- Calculate the explained variance ratio for each principal component.
- Plot a scree plot to visualize the explained variance ratio.

Step 6: Choose the Number of Components

Based on the scree plot, choose the number of principal components that explain a significant amount of variance.

Step 7: Apply PCA with Chosen Components

Apply PCA again with the chosen number of components.

Step 8: Visualize the Reduced Data

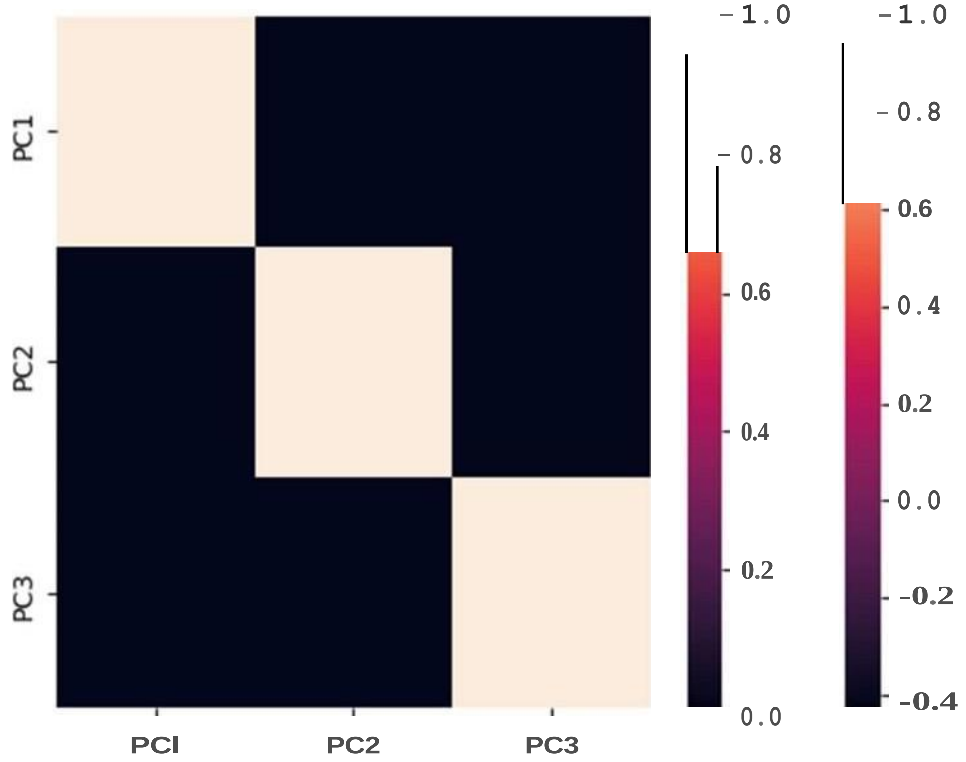
- Transform the original data to the reduced dimension using the fitted PCA.
- Visualize the reduced data using a scatter plot.

Step 9: Interpretation

Interpret the results, considering the trade-offs between dimensionality reduction and information loss.

PROGRAM:

```
from sklearn import datasets
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import seaborn as sns

iris= datasets.load_iris()
df = pd.DataFrame(iris['data'], columns=iris['feature_names'])
df.head()

scalar= StandardScaler()
scaled_data = pd.DataFrame(scalar.fit_transform(df)) # scaling the data
scaled_data

sns.heatmap(scaled_data.corr())

pea= PCA(n_components=3)
pca.fit(scaled_data)
data_pca = pca.transform(scaled_data)
data_pca = pd.DataFrame(data_pca, columns=['PC1', 'PC2', 'PC3'])
data_pca.head()
sns.heatmap(data_pca.corr())
```



RESULT:-

Thus Dimensionality Reduction has been implemented using PCA in a python program successfully and the results have been analyzed