RAJALAKSHMI ENGINEERING COLLEGE

(An Autonomous Institution)
Affiliated to Anna University

Rajalakshmi Nagar, Thandalam – 602105

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

AI23331 – FUNDAMENTALS OF MACHINE LEARNING

LAB RECORD

Name:

Register Number:

Year/Branch/Section:

Semester:

Academic Year:

RAJALAKSHMI ENGINEERING COLLEGE

RAJALAKSHMI NAGAR, THANDALAM-602 105

BONAFIDE CERTIFICATE

NAME …….

ACADEMIC YEAR ……...……...……SEMESTER ……

.

BRANCH …….

UNIVERSITY REGISTER NO:

Certify that this is the bonafide record of work done by the above student in the……. Laboratory during the year 20…. -20….

Signature of Faculty-in-charge

Submitted for the Practical Examination held on ……

Internal Examiner **External Examiner**

CONTENT

RAJALAKSHMI
ENGINEERING COLLEGE
An AUTONOMOUS Institution
Affiliated to ANNA UNIVERSITY, Chennai

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

AI23331-FUNDAMENTALS OF MACHINE LEARNING LAB

[Regulation 2023]

II Year – III Semester

LABORATORY MANUAL

SYLLABUS

List of Experiments	
1	Univariate, Bivariate, and Multivariate Regression
2	Simple Linear Regression Using Least Square Method
3	Logistic Regression Model
4	Single Layer Perceptron
5	Multi-Layer Perceptron with Backpropagation
6	Face Recognition Using SVM Classifier
7	Decision Tree Implementation
8	Boosting Algorithm Implementation
9	K-Nearest Neighbors (KNN) and K-Means Clustering
10	Dimensionality Reduction Using Principal Component Analysis (PCA)
11	Mini Project: Developing a Simple Application Using TensorFlow/Keras

Requirements

Hardware	Intel i3, CPU @ 1.20GHz 1.19 GHz, 4 GB RAM, 32 Bit Operating System
Software	python3.7,Jupiter
Operating System	Windows

Ex No: 1(a)
Date:

UNIVARIATE REGRESION USING PYTHON

AIM:

To implement a python program using univariate regression features for a given iris dataset.

ALGORITHM:

Step 1: 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 sklearn's LinearRegression class.
- Make predictions using the model.
- Calculate the R-squared value to evaluate the model's performance.

Step 5: 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 (X1, X2, y) as a 3D scatter plot and the regression plane.
- For multivariate regression, plot the predicted values against the actual values.

Step 6: Display the results:

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

Step 7: Complete the program:

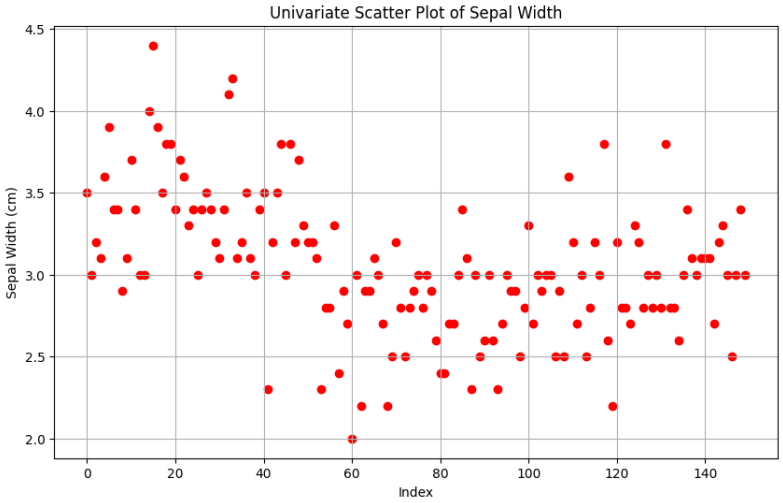
- Combine all the steps into a Python program.
- Run the program to perform univariate, bivariate, and multivariate regression on the dataset.

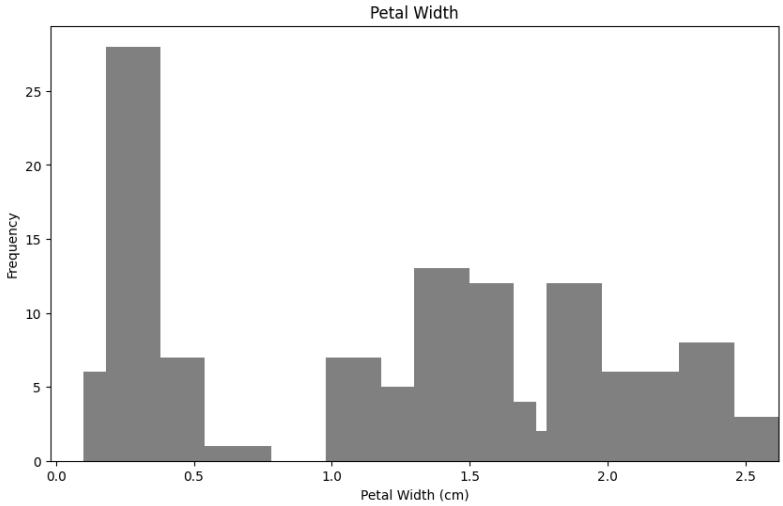
PROGRAM:

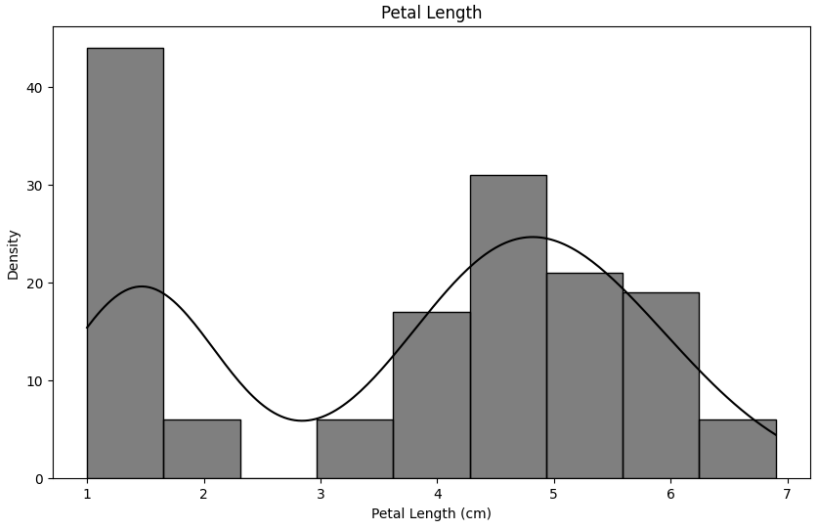
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns


iris_data = pd.read_csv('iris.csv')


sepal_width = iris_data['SepalWidthCm']
plt.figure(figsize=(10, 6))
plt.scatter(range(len(sepal_width)), sepal_width, color='red')
plt.title("Univariate Scatter Plot of Sepal Width")
plt.xlabel("Index")
plt.ylabel("Sepal Width (cm)")
plt.grid(True)
plt.show()


petal_width = iris_data['PetalWidthCm']
plt.figure(figsize=(10, 6))
plt.hist(petal_width, bins=30, color="grey", width=0.2)
plt.title("Petal Width")
plt.xlabel("Petal Width (cm)")
plt.ylabel("Frequency")
plt.show()


petal_length = iris_data['PetalLengthCm']
```

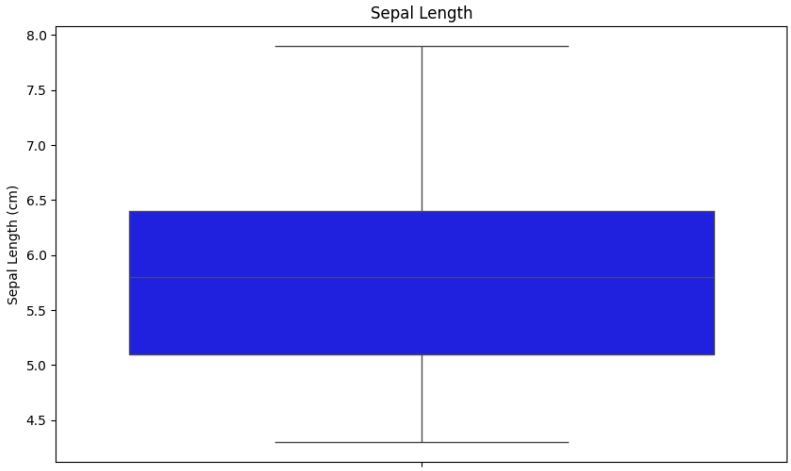
```
plt.figure(figsize=(10, 6))
sns.histplot(petal_length, kde=True, color="black")
plt.title("Petal Length")
plt.xlabel("Petal Length (cm)")
plt.ylabel("Density")
plt.show()


sepal_length = iris_data['SepalLengthCm']
plt.figure(figsize=(10, 6))
sns.boxplot(y=sepal_length, color="blue")
plt.title("Sepal Length")
plt.ylabel("Sepal Length (cm)")
plt.show()


plt.figure(figsize=(10, 6))
sns.barplot(x='Species', y='PetalLengthCm', data=iris_data, color="blue")
plt.title("Petal Length by Species")
plt.xlabel("Species")
plt.ylabel("Petal Length (cm)")
plt.show()
```

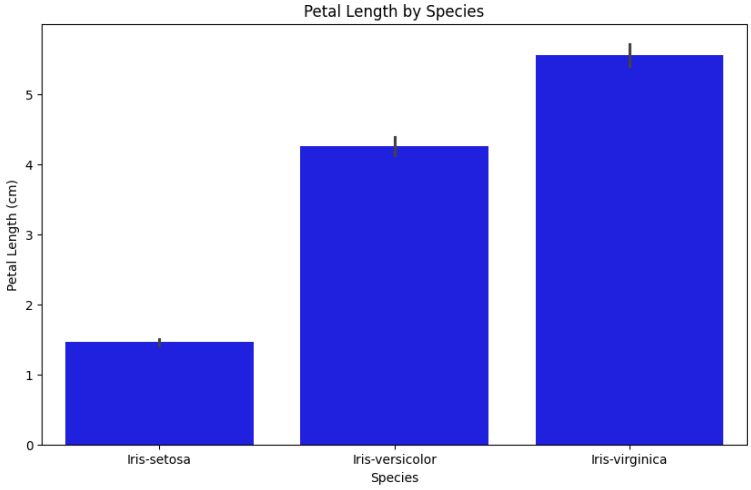
OUTPUT:


Univariate Scatter Plot of Sepal Width
Sepal Width (cm)
Index


Petal Width
Frequency
Petal Width (cm)


Petal Length
Density
Petal Length (cm)





RESULT:

Thus, the python program to implement univariate regression features for the given iris dataset is analyzed and the features are plotted using scatter plot

Ex No: 1(b)
Date:

BIVARIATE REGRESION USING PYTHON

AIM:

To implement a python program using bivariate regression features for a given iris dataset.

ALGORITHM:

Step 1: 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: Bivariate Regression:

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

Step 5: 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 (X1, X2, y) as a 3D scatter plot and the regression plane.
- For multivariate regression, plot the predicted values against the actual values.

Step 6: Display the results:

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

Step 7: Complete the program:

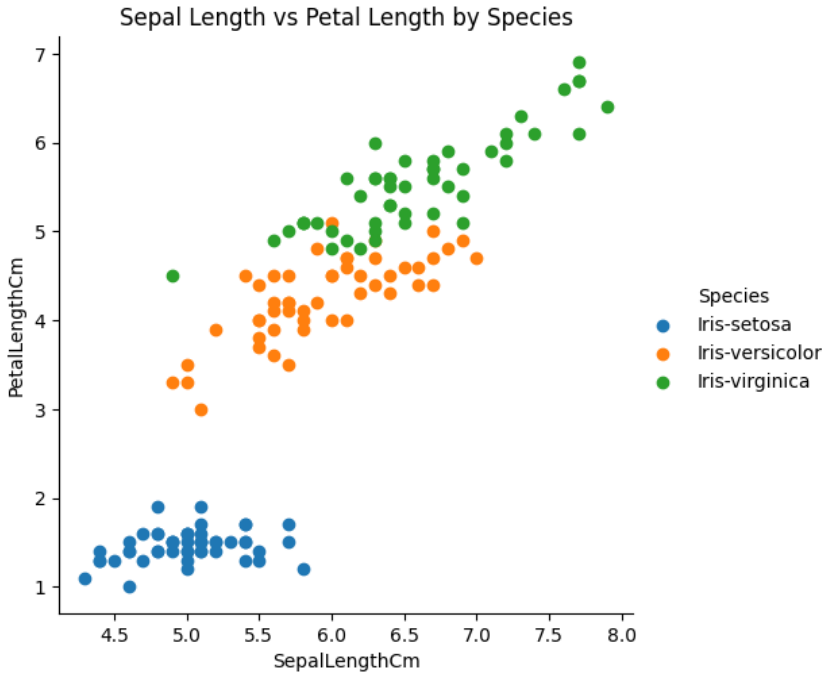
- Combine all the steps into a Python program.
- Run the program to perform univariate, bivariate, and multivariate regression on the dataset.

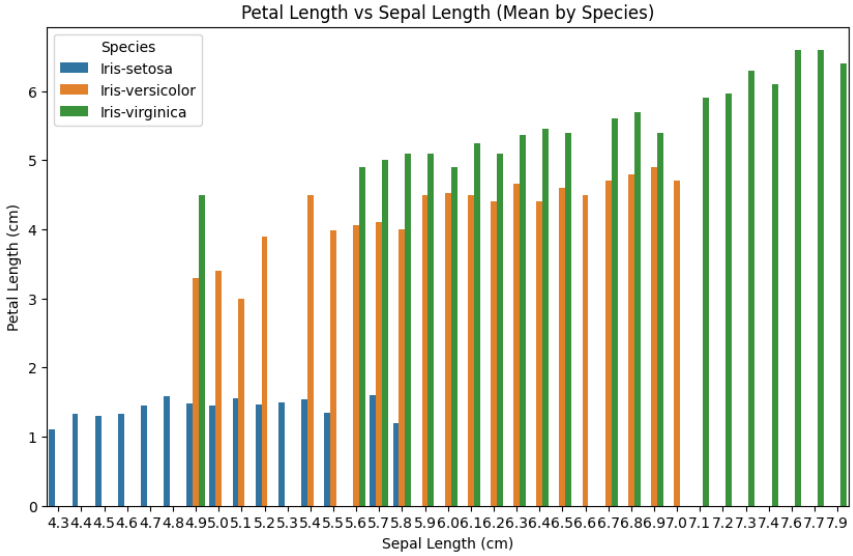
PROGRAM:

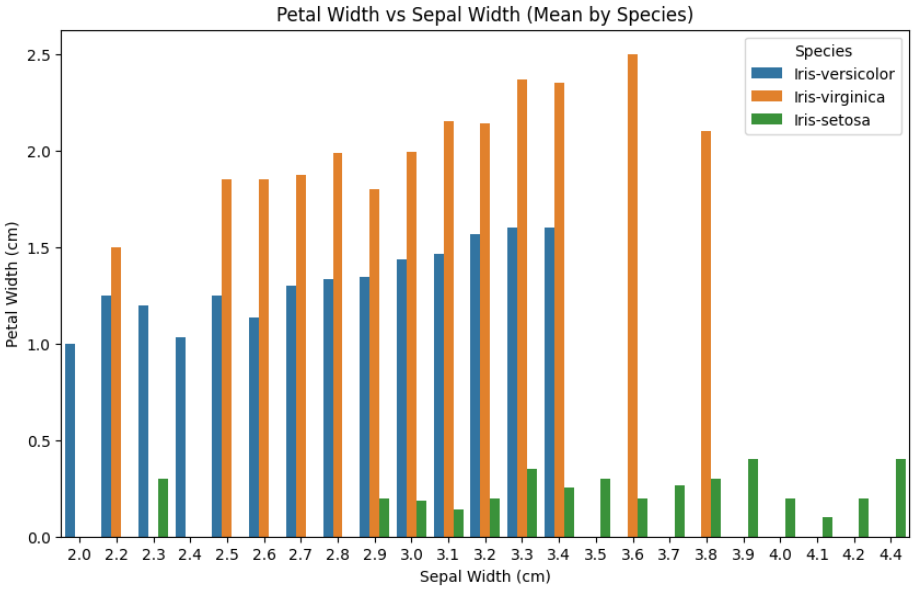
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

df = pd.read_csv("iris.csv")

plt.figure(figsize=(10, 6))
g = sns.FacetGrid(df, hue='Species', height=5)
g.map(plt.scatter, "SepalWidthCm", "PetalWidthCm")
g.add_legend()
plt.title("Sepal Width vs Petal Width by Species")
plt.show()

plt.figure(figsize=(10, 6))
g = sns.FacetGrid(df, hue='Species', height=5)
g.map(plt.scatter, "SepalLengthCm", "PetalLengthCm")
g.add_legend()
plt.title("Sepal Length vs Petal Length by Species")
plt.show()

agg_data = df.groupby(['Species', 'SepalLengthCm'],
as_index=False)['PetalLengthCm'].mean()
plt.figure(figsize=(10, 6))
sns.barplot(data=agg_data, x='SepalLengthCm', y='PetalLengthCm', hue='Species')
plt.title('Petal Length vs Sepal Length (Mean by Species)')
plt.xlabel('Sepal Length (cm)')
plt.ylabel('Petal Length (cm)')
plt.legend(title='Species')
plt.show()

agg_data = df.groupby(['Species', 'SepalWidthCm'],
as_index=False)['PetalWidthCm'].mean()
plt.figure(figsize=(10, 6))
sns.barplot(data=agg_data, x='SepalWidthCm', y='PetalWidthCm', hue='Species')
plt.title('Petal Width vs Sepal Width (Mean by Species)')
plt.xlabel('Sepal Width (cm)')
plt.ylabel('Petal Width (cm)')
plt.legend(title='Species')
plt.show()
```

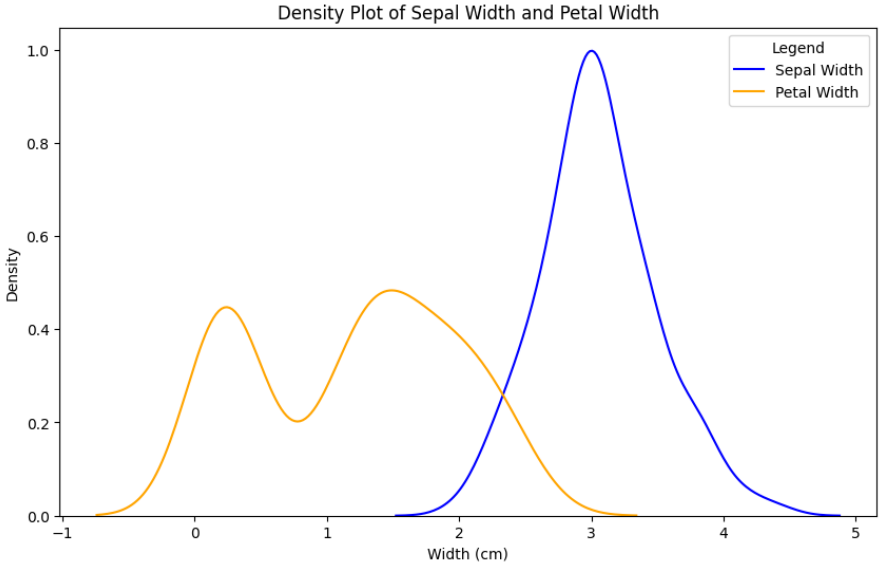
```
plt.figure(figsize=(10, 6))
sns.kdeplot(df['SepalWidthCm'], label='Sepal Width', color='blue', fill=False)
sns.kdeplot(df['PetalWidthCm'], label='Petal Width', color='orange', fill=False)
plt.title('Density Plot of Sepal Width and Petal Width')
plt.xlabel('Width (cm)')
plt.ylabel('Density')
plt.legend(title='Legend')
plt.show()

plt.figure(figsize=(10, 6))
sns.kdeplot(df['SepalLengthCm'], label='Sepal Length', color='blue', fill=False)
sns.kdeplot(df['PetalLengthCm'], label='Petal Length', color='orange', fill=False)
plt.title('Density Plot of Sepal Length and Petal Length')
plt.xlabel('Length (cm)')
plt.ylabel('Density')
plt.legend(title='Legend')
plt.show()

plt.figure(figsize=(10, 6))
sns.boxplot(data=df[['SepalWidthCm', 'PetalWidthCm']])
plt.title('Box Plot of Sepal Width and Petal Width')
plt.xlabel('Features')
plt.ylabel('Width (cm)')
plt.xticks([0, 1], ['Sepal Width', 'Petal Width'])
plt.show()

plt.figure(figsize=(10, 6))
sns.boxplot(data=df[['SepalLengthCm', 'PetalLengthCm']])
plt.title('Box Plot of Sepal Length and Petal Length')
plt.xlabel('Features')
plt.ylabel('Length (cm)')
plt.xticks([0, 1], ['Sepal Length', 'Petal Length'])
plt.show()
```

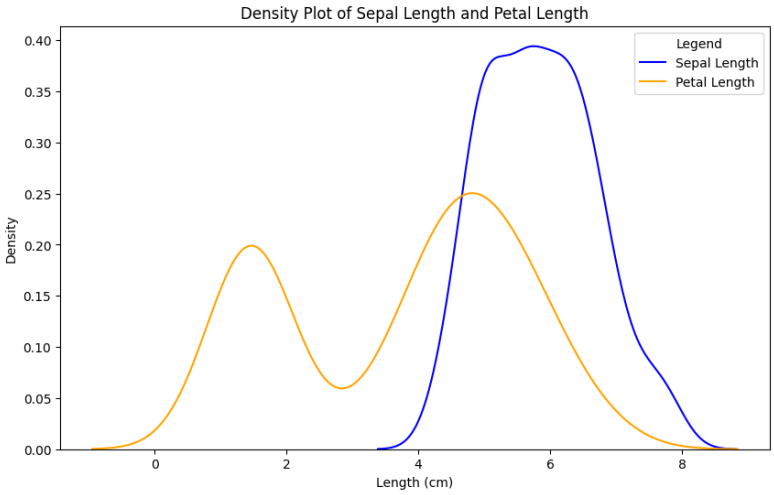
OUTPUT:

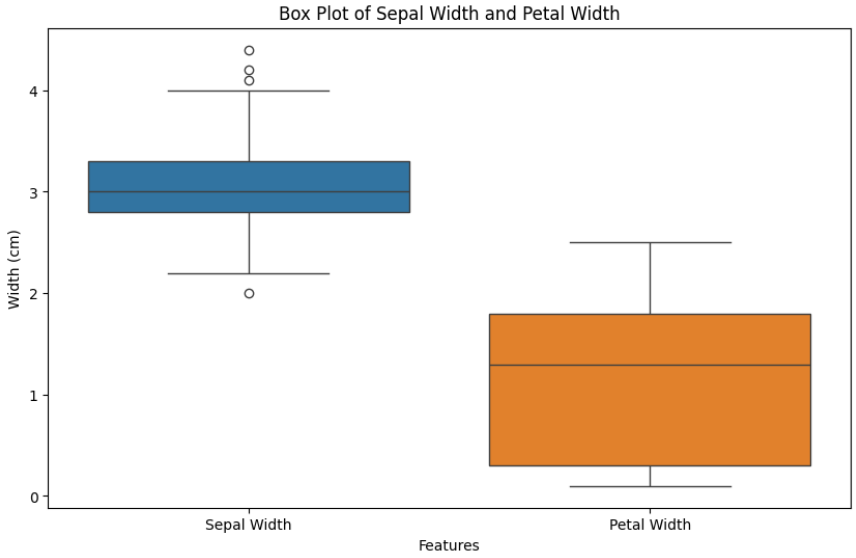

Sepal Length vs Petal Length by Species
PetalLengthCm
SepalLengthCm
Species
Iris-setosa
Iris-versicolor
Iris-virginica

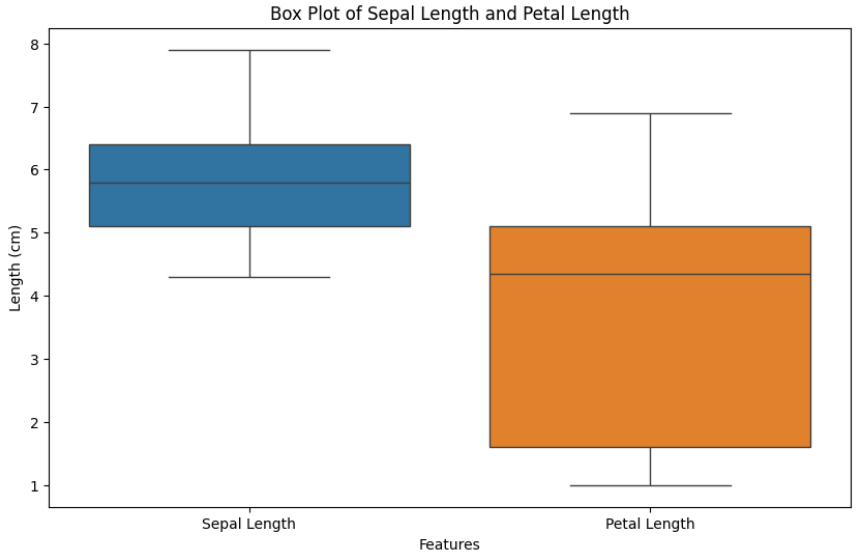

Petal Length vs Sepal Length (Mean by Species)
Species
Iris-setosa
Iris-versicolor
Iris-virginica
Petal Length (cm)
Sepal Length (cm)


Petal Width vs Sepal Width (Mean by Species)
Species
Iris-versicolor
Iris-virginica
Iris-setosa
Petal Width (cm)
Sepal Width (cm)


Density Plot of Sepal Width and Petal Width
Legend
Sepal Width
Petal Width
Density
Width (cm)


Density Plot of Sepal Length and Petal Length
Legend
Sepal Length
Petal Length
Density
Length (cm)





RESULT:

Thus, the python program to implement bivariate regression features for the given iris dataset is analyzed and the features are plotted using scatter plot

Ex No: 1(c)
Date:

MULTIVARIATE REGRESION USING PYTHON

AIM:

To implement a python program using multivariate regression features for a given iris dataset.

ALGORITHM:

Step 1: 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: Multivariate Regression:

- For multivariate regression, use more than two independent variables.
- Fit a linear regression model to the data using sklearn's `LinearRegression` class.
- Make predictions using the model.

Calculate the R-squared value to evaluate the model's performance

.

Step 5: 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 (X1, X2, y) as a 3D scatter plot and the regression plane.
- For multivariate regression, plot the predicted values against the actual values.

Step 6: Display the results:

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

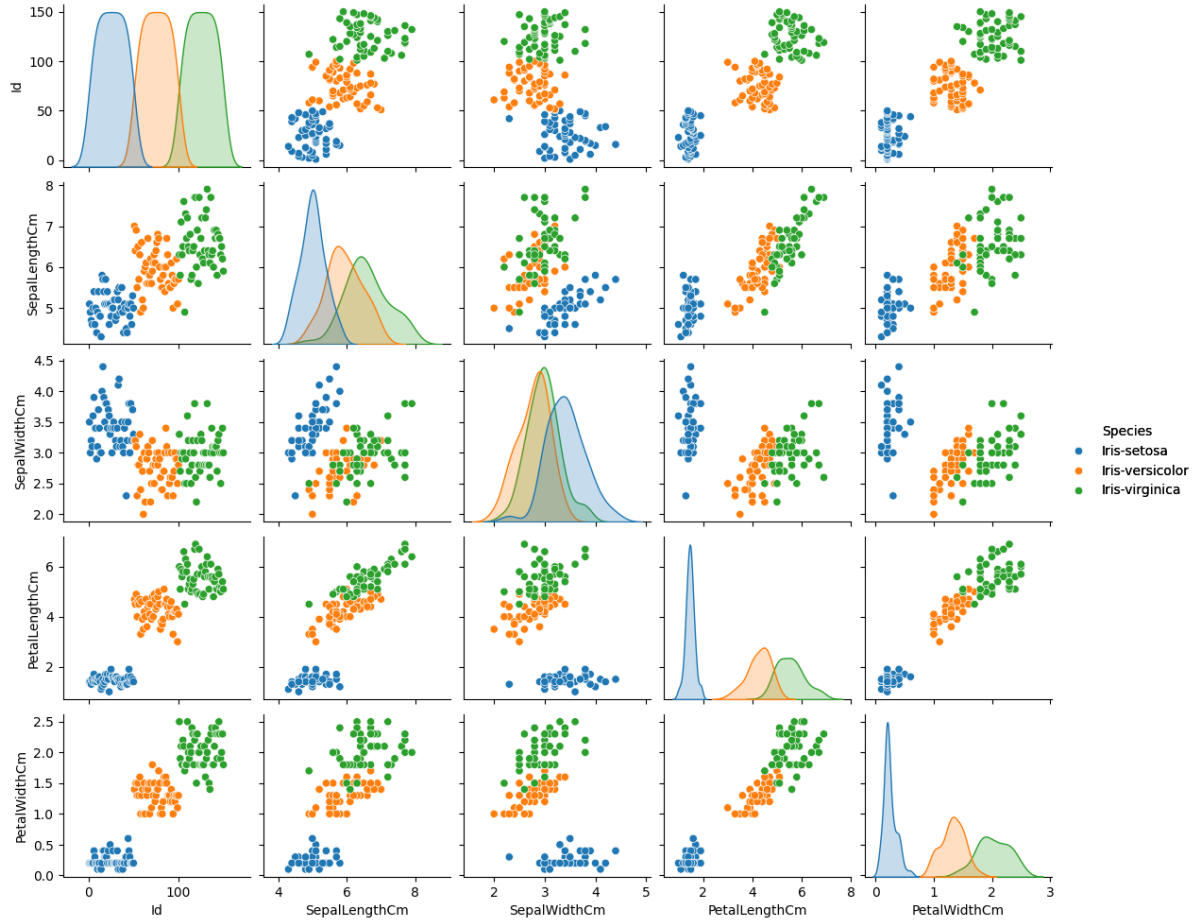
Step 7: Complete the program:

- 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

df = pd.read_csv("iris.csv")
g=sns.pairplot(df,hue="Species",size=2)
g.add_legend()
plt.show()
```

OUTPUT:



RESULT:

Thus, the python program to implement multivariate regression features for the given iris dataset is analyzed and the features are plotted using scatter plot

Ex No: 2(a)

Date:

SIMPLE LINEAR REGRESSION USING LEAST SQUARE METHOD (MODEL)

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:

$$m = \frac{\sum_{i=1}^{n}(X_i-\bar{X})(y_i-\bar{y})}{\sum_{i=1}^{n}(X_i-\bar{X})^2}$$

- Calculate the intercept (b) using the formula: $b = \bar{y} - m\bar{X}$

Step 6: Make predictions:

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

$$\hat{y} = mx + b$$

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: $TSS = \sum_{i=1}^{n}(y_i - \bar{y})^2$
- Calculate the residual sum of squares (RSS) using the formula: $RSS = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$
- Calculate the R-squared value using the formula: $R^2 = 1 - \frac{RSS}{TSS}$

Step 9: Display the results:

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

Step 10: Complete the program:

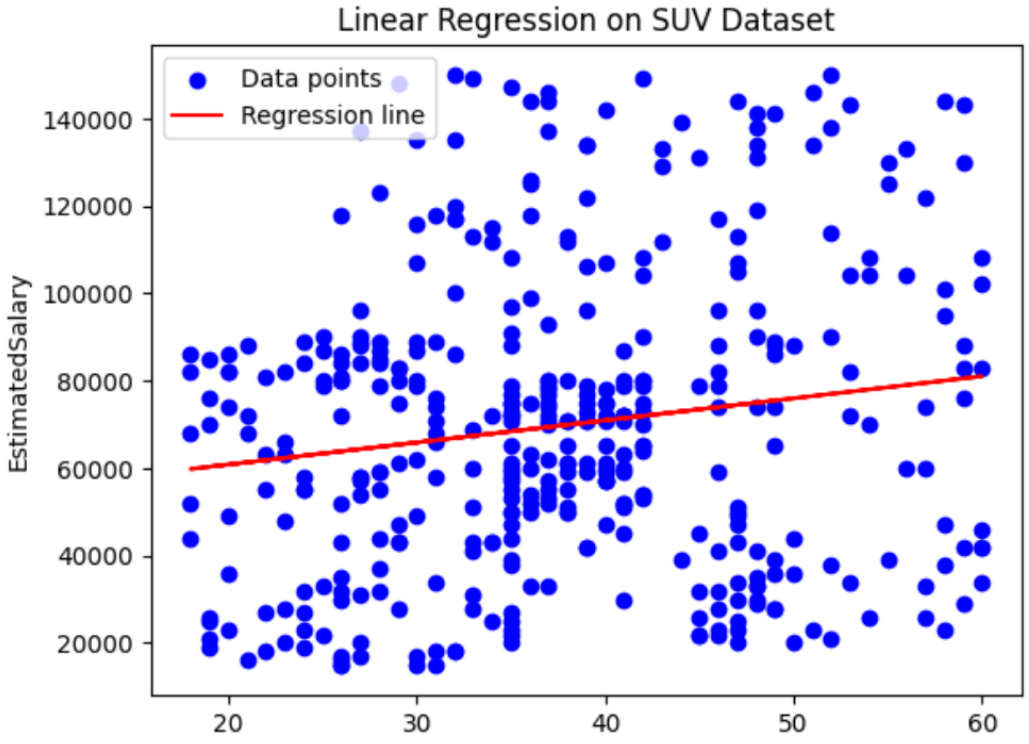
- Combine all the steps into a Python program.
- Run the program to perform simple linear regression on the dataset.

PROGRAM:

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
file_path = 'suvdata.csv'  # Adjust the file name and path as necessary
data = pd.read_csv(file_path)
print("First few rows of the dataset:")
print(data.head())
x_column = 'Age'  # Independent variable
y_column = 'EstimatedSalary'  # Dependent variable
X = data[[x_column]].values  # Note the double brackets to keep X as a 2D array
y = data[y_column].values
model = LinearRegression()
model.fit(X, y)
m = model.coef_[0]
c = model.intercept_
print("Slope (m):", m)
print("Intercept (c):", c)
y_pred = model.predict(X)
plt.scatter(X, y, color="blue", label="Data points")
plt.plot(X, y_pred, color="red", label="Regression line")
plt.xlabel(x_column)
plt.ylabel(y_column)
plt.title("Linear Regression on SUV Dataset")
plt.legend()
plt.show()
```

OUTPUT:

```
Slope (m): 504.9324471182231
Intercept (c): 50729.26870376331
```



RESULT:

Thus, the python program to implement simple linear regression using least square method for the given head brain dataset is analyzed and the linear regression line is constructed successfully

Ex No: 2(b)
Date:

SIMPLE LINEAR REGRESSION USING LEAST SQUARE METHOD (MATHEMATICAL)

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:

$$m = \frac{\sum_{i=1}^{n}(X_i-\bar{X})(y_i-\bar{y})}{\sum_{i=1}^{n}(X_i-\bar{X})^2}$$

- Calculate the intercept (b) using the formula: $b = \bar{y} - m\bar{X}$

Step 6: Make predictions:

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

$$\hat{y} = mx + b$$

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: $TSS = \sum_{i=1}^{n}(y_i - \bar{y})^2$
- Calculate the residual sum of squares (RSS) using the formula: $RSS = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$
- Calculate the R-squared value using the formula: $R^2 = 1 - \frac{RSS}{TSS}$

Step 9: Display the results:

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

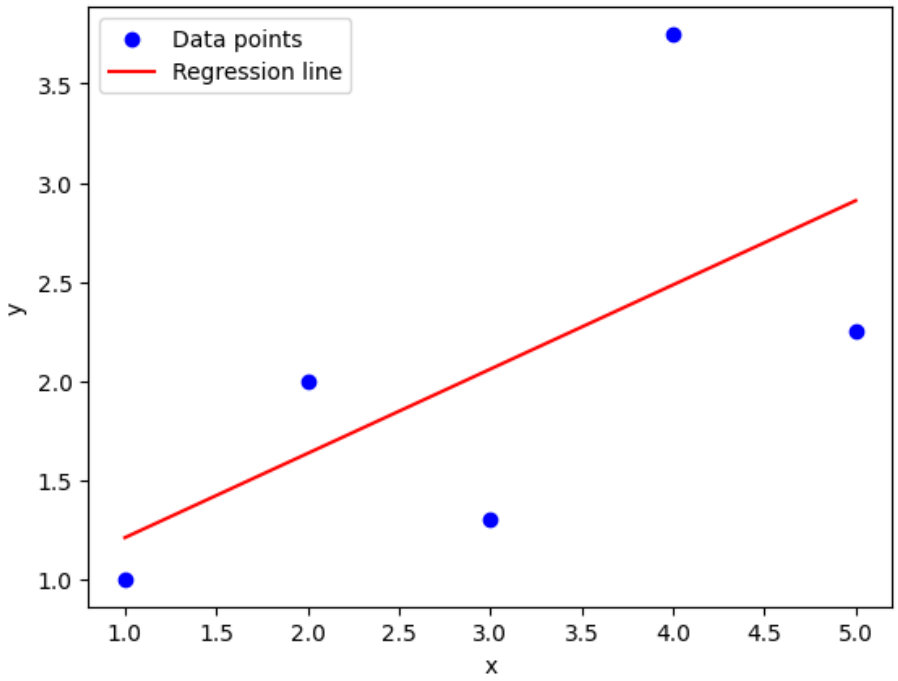
Step 10: Complete the program:

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

PROGRAM:

```
import numpy as np
import matplotlib.pyplot as plt
x = np.array([1, 2, 3, 4, 5])  # Independent variable
y = np.array([1, 2, 1.3, 3.75, 2.25])  # Dependent variable
x_mean = np.mean(x)
y_mean = np.mean(y)
numerator = np.sum((x - x_mean) * (y - y_mean))
denominator = np.sum((x - x_mean) ** 2)
m = numerator / denominator
c = y_mean - (m * x_mean)
print("Slope (m):", m)
print("Intercept (c):", c)
y_pred = m * x + c
plt.scatter(x, y, color="blue", label="Data points")
plt.plot(x, y_pred, color="red", label="Regression line")
plt.xlabel("x")
plt.ylabel("y")
plt.legend()
plt.show()
```

OUTPUT:



RESULT:

Thus, the python program to implement simple linear regression using least square method for the given head brain dataset is analyzed and the linear regression line is constructed successfully

Ex no: 3

Date:

LOGISTIC MODEL USING PYTHON

AIM:

To 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
- sklearn.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., using LabelEncoder 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 sklearn.linear_model.
- Train the model on the training data using the fit method.
    - Create a function named “Sigmoid ()” which will define the sigmoid values using the
    - formula (1/1+e-z) and return the computed value.
    - 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”.
    - Create a function named “fit” which will be used to plot the graph according to the training data.
    - Create a predict function that will predict values according to the training model created using the fit function.
    - 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.
    - Use the “predict()” function to predict the values of the testing data and assign the value to “y_pred” variable.
    - Use the “predict()” function to predict the values of the training data and assign the value to “y_trainn” variable.
    - Compute f1_score for both the training and testing data and assign the values to “f1_score_tr” and “f1_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.

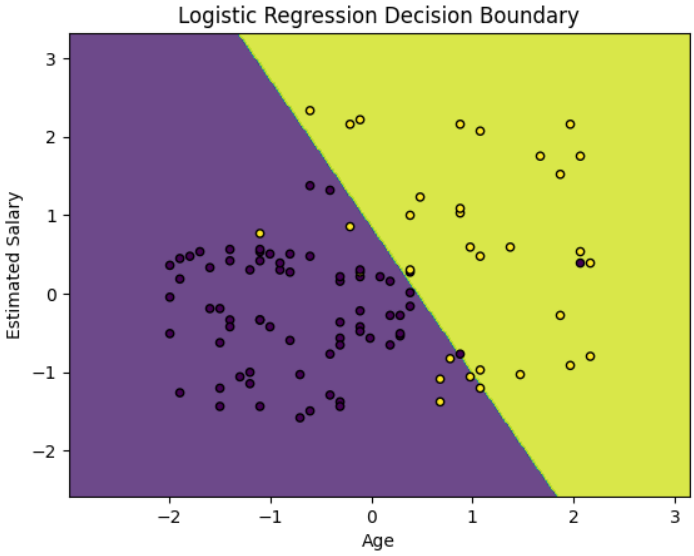
Step 9: Visualize the Results:

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

PROGRAM:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
df = pd.read_csv('suvdata.csv')
X = df[['Age', 'EstimatedSalary']].values
y = df['Purchased'].values
scaler = StandardScaler()
X = scaler.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)
model = LogisticRegression().fit(X_train, y_train)

accuracy = accuracy_score(y_test, model.predict(X_test))
print("Model Accuracy:", accuracy)
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1), np.arange(y_min, y_max, 0.1))
Z = model.predict(np.c_[xx.ravel(), yy.ravel()]).reshape(xx.shape)
plt.contourf(xx, yy, Z, alpha=0.3, cmap=plt.cm.Paired)
plt.scatter(X[:, 0], X[:, 1], c=y, edgecolor='k', cmap=plt.cm.Paired, s=30)
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.title('Logistic Regression Decision Boundary')
plt.show()
```

OUTPUT:



RESULT:-

Thus, the python program to implement logistic regression for the given suv_cars dataset is analyzed and the logistic regression model is classifies successfully. The performance of the developed model is measured using F1-score and Accuracy

Ex. No: 4

Date:

SINGLE LAYER PERCEPTRON USING PYTHON

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/(1+np.exp(-x)) and return the value.
- User Defined function - der(x):
    - Compute the product of value of sigmoid_func(x) and (1 - 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 (error = 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 pandas as pd
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
df = pd.read_csv('suvdata.csv')  # Update the path as necessary
X = df.drop(columns=['Purchased'])  # Drop the target column
y = df['Purchased']                 # This should be the target variable (0 or 1)
print(df.info())
preprocessor = ColumnTransformer(
    transformers=[
        ('num', SimpleImputer(strategy='mean'), ['Age', 'EstimatedSalary']),  # Replace
with your numeric columns
        ('cat', OneHotEncoder(), ['Gender'])  # Replace 'Gender' with your categorical
columns
    ],
    remainder='passthrough'  # Keep other columns unchanged
)
pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', MLPClassifier(hidden_layer_sizes=(10, 10), max_iter=1000,
random_state=42))
])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)
```

```
pipeline.fit(X_train, y_train)
y_pred = pipeline.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
```

OUTPUT:

```
[[6.62916366]
 [6.62916441]]
[-10.23197316]
[0.02652435]
[0.95375065]
[3.59993686e-05]
[0.02652437]
```

RESULT:-

Thus, the python program to implement Single Layer Perceptron has been executed successfully.

Ex. No: 5

Date:

MULTI LAYER PERCEPTRON WITH BACK PROPOGATION USING PYTHON

AIM:

To implement multilayer perceptron with back propagation using python.

ALGORITHM:

Step 1: Import the Necessary 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(10)`.

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 `sklearn.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_test`, `y_train`, and `y_test`.

Step 7: Import MLPClassifier Module

- Import `MLPClassifier` from `sklearn.neural_network`.

Step 8: Initialize MLPClassifier

- Create an instance of `MLPClassifier` with `max_iter=500` and `activation='relu'`.
- Assign the instance to a variable (e.g., `clf`).

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 `sklearn.metrics`.
- Import `classification_report` from `sklearn.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='tanh'`.
- Repeat for `activation='identity'`.

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

- Use `train_test_split` to split the dataset into training and testing sets with a ratio of 0.3.
- Assign the results to `x_train`, `x_test`, `y_train`, and `y_test`.
- Repeat Steps 7-14 with the new training and testing sets.

PROGRAM:

```
import pandas as pd
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
df = pd.read_csv('suvdata.csv')
X = df.drop(columns=['Purchased'])
y = df['Purchased'
preprocessor = ColumnTransformer(
    transformers=[
        ('cat', OneHotEncoder(), ['Gender']),
    ],
    remainder='passthrough'
)
pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', MLPClassifier(hidden_layer_sizes=(10, 10), max_iter=1000,
random_state=42))
])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)
pipeline.fit(X_train, y_train)
y_pred = pipeline.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
```

OUTPUT:

```
Training MLPClassifier with activation function: relu
Confusion Matrix:
[[147   0]
 [  0 128]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       147
           1       1.00      1.00      1.00       128

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


Training MLPClassifier with activation function: logistic
Confusion Matrix:
[[147   0]
 [  0 128]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       147
           1       1.00      1.00      1.00       128
```

RESULT:

Thus the python program to implement multi layer perceptron with back propagation on the given dataset(banknotes.csv) has been executed successfully and it's results have been analyzed successfully for different activation functions(relu,logistic,tanh,identity) with two different training-testing ratios(0.2 and 0.3)

Ex no: 6

Date:

SVM CLASSIFIER MODEL USING PYTHON

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 sklearn.
4. Import matplotlib.pyplot as plt.
5. Import seaborn as sns.
6. Set the font_scale attribute to 1.2 in seaborn.

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 "Set1".
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" column, assign 0 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

1. 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[0] / w[1]`.
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[1]`.
6. Calculate the decision boundary line `y` as `a * xx - (model.intercept_[0] / w[1])`.

Step 7: Calculate Support Vector Boundaries

1. Assign the first support vector to variable `b`.
2. Calculate `yy_down` as `a * xx + (b[1] - a * b[0])`.
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_test` using `train_test_split`.
2. Set the test size to 0.2.

Step 12: Train New Model

1. Create a new SVC model named `model1`.
2. Fit the model using the training data (`x_train` and `y_train`).

Step 13: Make Predictions

1. Use the `predict()` function on `model1` with `x_test` as the parameter.
2. Assign the predictions to variable `pred`.

Step 14: Evaluate Model

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

PROGRAM:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix
df = pd.read_csv('suvdata.csv')  # Update with the correct path to your SUV dataset
X = df.drop(columns=['Purchased'])
y = df['Purchased'
X = pd.get_dummies(X, drop_first=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)
svm_model = SVC(kernel='linear', random_state=42)
svm_model.fit(X_train, y_train)
y_pred = svm_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)
```

OUTPUT:

```
Accuracy: 0.98
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      0.92      0.96        13
           2       0.93      1.00      0.96        13

    accuracy                           0.98        45
   macro avg       0.98      0.97      0.97        45
weighted avg       0.98      0.98      0.98        45

Confusion Matrix:
[[19  0  0]
 [ 0 12  1]
 [ 0  0 13]]
```

RESULT:

Thus the python program to implement SVM classifier model has been executed successfully and the classified output has been analyzed for the given dataset(muffins.csv)

Ex. No: 7

Date:

DECISION TREE USING PYTHON

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 `sklearn.datasets`.

Step 2: Import Necessary Libraries

1. Import numpy as np.
2. Import matplotlib.pyplot as plt.
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_iris()`.
2. Assign the dataset's data to variable `X`.
3. Assign the dataset's target to variable `Y`.

Step 5: Train the Model

1. 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[1]), 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[:, 0].min() - 1, X[:, 0].max() + 1`).
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

1. Use `plt.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[0]])` and `plt.ylabel(iris.feature_names[pair[1]])`).

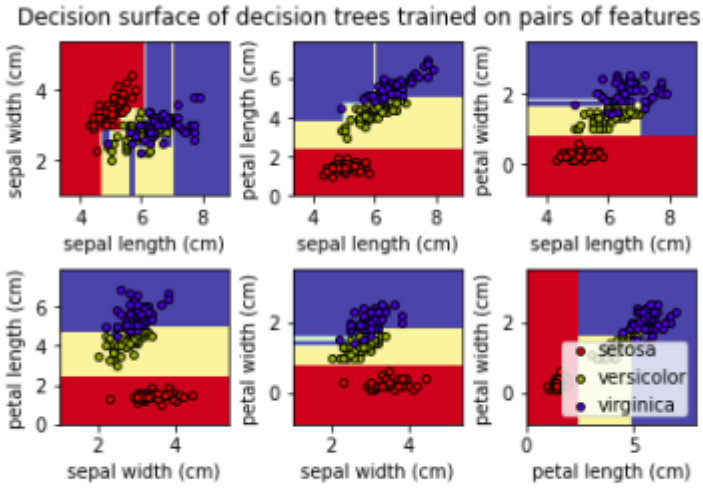
Step 13: Plot Training Points

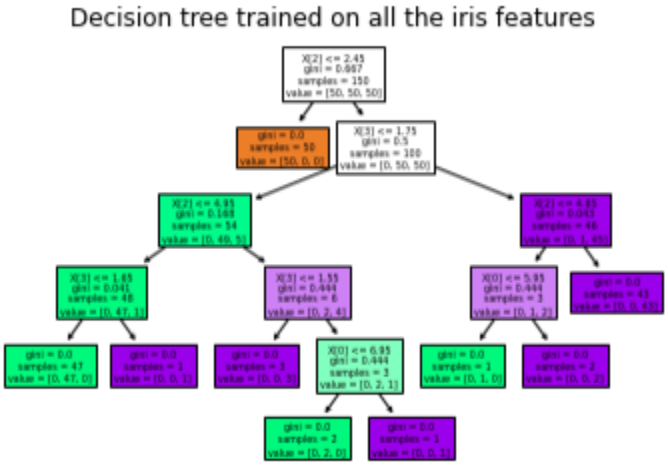
1. Use `plt.scatter(X[:, 0], X[:, 1], c=Y, cmap=plt.cm.RdYlBu, edgecolor='k', s=15)` to plot the training points with the "RdYlBu" color scheme, black edge color, and size 15.

Step 14: Plot Final Decision Tree

1. Set the title of the plot to "Decision tree trained on all the iris features" (e.g., `plt.title("Decision tree trained on all the iris features")`).
2. Display the plot using `plt.show()`.

Sample Output:

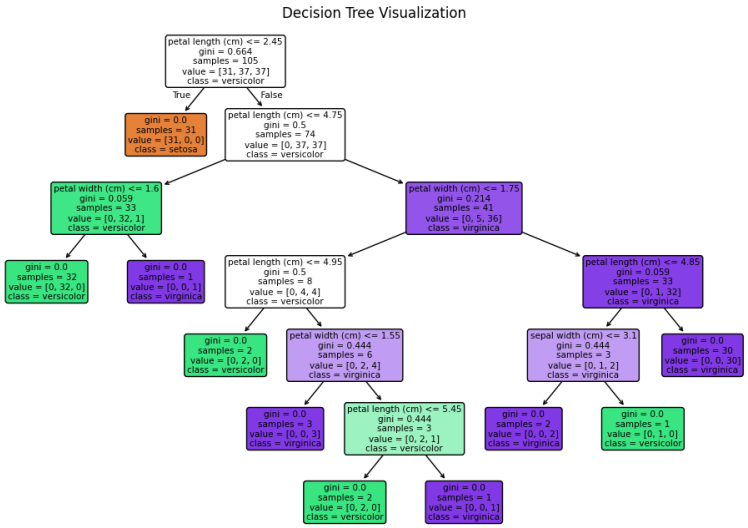




PROGRAM:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn import tree
iris = datasets.load_iris()
x = iris.data
y = iris.target
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=42)
decision_tree_classifier = DecisionTreeClassifier(random_state=42)
decision_tree_classifier.fit(x_train, y_train)
y_pred = decision_tree_classifier.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
print("Classification Report:")
print(classification_report(y_test, y_pred))
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
plt.figure(figsize=(12, 8))
tree.plot_tree(decision_tree_classifier, filled=True,
feature_names=iris.feature_names, class_names=iris.target_names, rounded=True)
plt.title("Decision Tree Visualization")
plt.show()
```

OUTPUT:



RESULT:

Thus the python program to implement Decision Tree for the given dataset has been successfully implemented and the results have been verified and analyzed

Ex. No: 8 (a)

Date:

ADA BOOSTING USING PYTHON

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_csv() (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_split().

Step 3: Initialize Parameters

Set the number of weak classifiers n_estimators.

Initialize an array weights for instance weights, setting each weight to 1 / number_of_samples.

Step 4: Train Weak Classifiers

Loop for n_estimators iterations:

Train a weak classifier using DecisionTreeClassifier(max_depth=1) 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((1 - 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
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
df = pd.read_csv('suvdata.csv')
if 'Purchased' not in df.columns:
    raise ValueError("The target column 'Purchased' is not present in the dataset.")
X = df.drop(columns=['Purchased'])
y = df['Purchased'].astype(int)
X = pd.get_dummies(X, drop_first=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)
ada_boost_model = AdaBoostClassifier(n_estimators=50, random_state=42)
ada_boost_model.fit(X_train, y_train)
y_pred = ada_boost_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)
```

OUTPUT:

```
Model Accuracy: 50.00%
```

RESULT:

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

Ex. No: 8(b)
Date:

GRADIENT BOOSTING USING PYTHON

AIM:

To implement a python program using the gradient boosting model.

ALGORITHM:

Step 1: Import Necessary Libraries

Import numpy as np.

Import pandas as pd.

Import train_test_split from sklearn.model_selection.

Import DecisionTreeRegressor from sklearn.tree.

Import mean_squared_error from sklearn.metrics.

Step 2: Prepare the Data

Load your dataset into a DataFrame using pd.read_csv('your_dataset.csv').

Split the dataset into features (X) and target (y).

Use train_test_split to split the data into training and testing sets.

Step 3: Initialize Parameters

Set the number of boosting rounds (e.g., n_estimators = 100).

Set the learning rate (e.g., learning_rate = 0.1).

Initialize an empty list to store the weak learners (decision trees).

Initialize an empty list to store the learning rates for each round.

Step 4: Initialize the Base Model

Compute the initial prediction as the mean of the target values (e.g., F0 = np.mean(y_train)).

Initialize the predictions to the base model's prediction (e.g., F = np.full(y_train.shape, F0)).

Step 5: Iterate Over Boosting Rounds

For each boosting round:

Compute the pseudo-residuals (negative gradient of the loss function) (e.g., residuals = y_train - F).

Fit a decision tree to the pseudo-residuals.

Make predictions using the fitted tree (e.g., tree_predictions = tree.predict(X_train)).

Update the predictions by adding the learning rate multiplied by the tree predictions (e.g., F += learning_rate * tree_predictions).

Append the fitted tree and the learning rate to their respective lists.

Step 6: Make Predictions on Test Data

Initialize the test predictions with the base model's prediction (e.g., F_test = np.full(y_test.shape, F0)).

For each fitted tree and its learning rate:

Make predictions on the test data using the fitted tree.

Update the test predictions by adding the learning rate multiplied by the tree predictions.

Step 7: Evaluate the Model

Compute the mean squared error on the training data.

Compute the mean squared error on the test data.

PROGRAM:

```
import numpy as np
import pandas as pd

class GradientBoosting:
    def __init__(self, n_estimators=100, learning_rate=0.1, max_depth=1):
        self.n_estimators = n_estimators
        self.learning_rate = learning_rate
        self.models = []

    def fit(self, X, y):
        y_pred = np.full(y.shape, np.mean(y))  # Initialize with mean
        for _ in range(self.n_estimators):
            residuals = y - y_pred
            stump = DecisionStump()
            stump.fit(X, residuals)
            self.models.append(stump)
            y_pred += self.learning_rate * stump.predict(X)

    def predict(self, X):
        return np.sum([self.learning_rate * model.predict(X) for model in self.models],
axis=0)


class DecisionStump:
    def fit(self, X, y):
        self.feature_index, self.threshold, self.left_val, self.right_val = min(
            ((i, t, np.mean(y[X[:, i] <= t]), np.mean(y[X[:, i] > t]))
             for i in range(X.shape[1]) for t in np.unique(X[:, i])),
            key=lambda x: np.sum((y - np.where(X[:, x[0]] <= x[1], x[2], x[3])) ** 2))

    def predict(self, X):
        return np.where(X[:, self.feature_index] <= self.threshold, self.left_val,
self.right_val)


def load_data(file_path):
    data = pd.read_csv(file_path)
    return data.iloc[:, :-1].values, data.iloc[:, -1].values

file_path = "your_dataset.csv"
X, y = load_data(file_path)
```

```
model = GradientBoosting(n_estimators=100, learning_rate=0.1)
model.fit(X, y)
predictions = model.predict(X)
print("Predictions:", predictions)
```

OUTPUT:

```
Accuracy: 0.8916666666666667
Confusion Matrix:
[[69  4]
 [ 9 38]]
```

RESULT:

Thus, the python program to implement gradient boosting for the standard uniform distribution has been successfully implemented and the results have been verified and analyzed.

Ex. No: 9(a)

Date:

KNN MODEL USING PYTHON

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 sklearn.model_selection, StandardScaler from sklearn.preprocessing, KNeighborsClassifier from sklearn.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.

PROGRAM:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
df = pd.read_csv('suvdata.csv')
X = df.drop(columns=['Purchased'])
y = df['Purchased'
X = pd.get_dummies(X, drop_first=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
k = 5
knn_model = KNeighborsClassifier(n_neighbors=k)
knn_model.fit(X_train, y_train)
y_pred = knn_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

OUTPUT:

```
KNN Accuracy: 85.00%
```

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 sklearn.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 numpy as np
from sklearn.cluster import KMeans
from sklearn.datasets import make_classification

X, _ = make_classification(n_samples=100, n_features=4, n_clusters_per_class=1,
random_state=42)

kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X)

print("Cluster Centers:\n", kmeans.cluster_centers_)
```

OUTPUT:

```
Cluster Centers:
 [[ 0.54995517  0.28524553  0.96163041 -1.04670366]
 [ 1.23798946  0.60635612  2.00231725  1.91433481]
 [ 0.2373546   0.11577163  0.38170444  0.42466382]]
```

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 sklearn.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 sklearn.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.

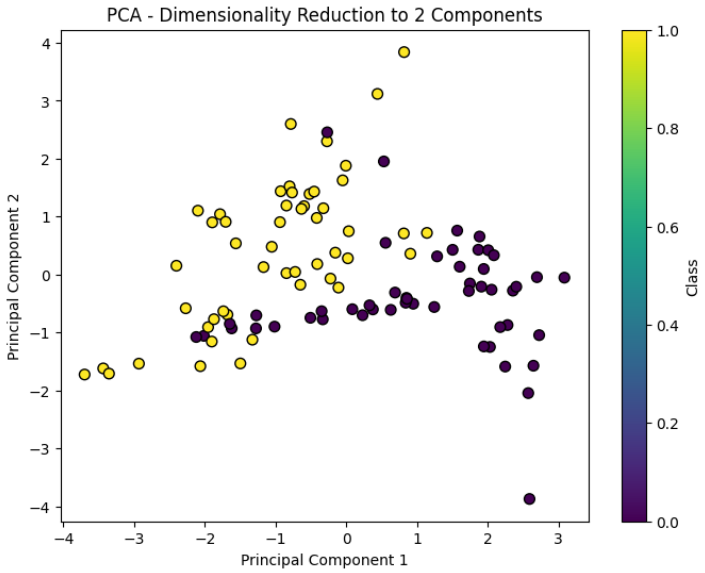
Step 9: Interpretation

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

PROGRAM:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.datasets import make_classification
from sklearn.preprocessing import StandardScaler

X, y = make_classification(n_samples=100, n_features=5, n_informative=3,
n_redundant=2, random_state=42)

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

print("Explained Variance by each Principal Component:",
pca.explained_variance_ratio_)

plt.figure(figsize=(8, 6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, cmap='viridis', edgecolor='k', s=50)
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.title("PCA - Dimensionality Reduction to 2 Components")
plt.colorbar(label="Class")
plt.show()
```

OUTPUT:



RESULT: -

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