

RAJALAKSHMI ENGINEERING COLLEGE

RAJALAKSHMI NAGAR, THIANDALAM - 602 105.



RAJALAKSHMI ENGINEERING COLLEGE

AI19442

FUNDAMENTALS OF MACHINE LEARNING

Laboratory Record Note Book

NAMESHANMUGASHREE...M.....

BRANCHB.TECH...ARTIFICIAL...INTELLIGENCE...&...DATA...SCIENCE

UNIVERSITY REGISTER No2116221801049.....

COLLEGE ROLL No221881049.....

SEMESTERVI.....

ACADEMIC YEAR2024 - 25.....



RAJALAKSHMI
ENGINEERING COLLEGE
An AUTONOMOUS Institution
Affiliated to Anna University, Chennai

BONAFIDE CERTIFICATE

NAMESHANMUGA SHREE M.....

ACADEMIC YEAR2024-25..... SEMESTERVI..... BRANCH...B.TE(A) A.I.A.DS

UNIVERSITY REGISTER No.

2116221801049

Certified that this is the bonafide record of work done by the above student in the

AI19442

Fundamentals of Machine Laboratory during the year **2024 - 2025**
Learning

Signature of Faculty - in - Charge












Submitted for the Practical Examination held on.....

External Examiner

Internal Examiner

INDEX

Name: Shanmugashree M Branch: B.Tech AI&DS Sec: - Roll No: 221801049

S.No.	Date	Title	Page No.	Teacher's Sign/Remarks
1.	24/1/2025	Univariate, Bivariate and Multivariate Regression	1	
2.	31/1/2025	Simple Linear Regression using Least Square Method	7	
3.	7/2/2025	Logistic Regression Model	10	
4.	14/2/2025	Single Layer Perceptron	12	
5.	21/2/2025	Multi-Layer perceptron with Backpropagation	14	
6.	28/2/2025	Face Recognition using SVM classifier	17	
7.	7/3/2025	Decision Tree Implementation	19	
8.	28/3/2025	Boosting Algorithm Implementation	21	
9.	4/4/2025	K-Nearest Neighbours (kNN) and K-Means clustering	23	
10.	11/4/2025	Dimensionality Reduction using Principal component Analysis	25	
11.	11/4/2025	Mini project : Simple application using TensorFlow / keras	27	
Completed				

Ex. No: 1	UNIVARIATE, BIVARIATE AND MULTIVARIATE REGRESSION
Date: 24/1/2025	

AIM:

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

ALGORITHM:

STEP 1: Start the program

STEP 2: Import the necessary libraries: pandas for data manipulation, numpy for numerical operations, matplotlib.pyplot for plotting, and LinearRegression from sklearn.linear_model.

STEP 3: Read the dataset using pandas.read_csv() and store it in a variable such as data.

STEP 4: Prepare the data by extracting the independent variable(s) X and the dependent variable y, and reshape them into 2D arrays if required.

STEP 5: Perform univariate, bivariate, and multivariate linear regression using either numpy.polyfit or sklearn's LinearRegression, then make predictions and calculate the R-squared value to assess model accuracy.

STEP 6: Visualize the results using 2D/3D scatter plots and regression lines or planes, and display the coefficients, intercepts, and R-squared values for each model.

STEP 7: Stop

PROGRAM:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
df=pd.read_csv(r'C:\Users\221801049\Iris.csv')
df.head(150)
df.shape
```

OUTPUT:

```
(150, 6)
```

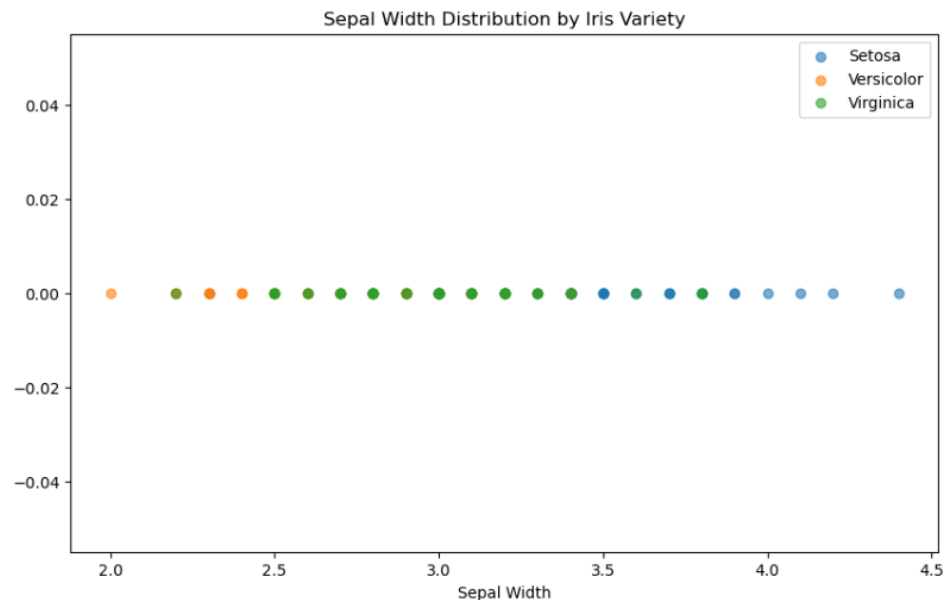
UNIVARIATE FOR SEPAL WIDTH

```
df_Setosa = df[df['Species'] == 'Iris-setosa']
df_Virginica = df[df['Species'] == 'Iris-virginica']
df_Versicolor = df[df['Species'] == 'Iris-versicolor']
plt.figure(figsize=(10, 6))
```

```
plt.scatter(df_Setosa['SepalWidthCm'], np.zeros_like(df_Setosa['SepalWidthCm']), label='Setosa',
                                                    alpha=0.6)
plt.scatter(df_Versicolor['SepalWidthCm'], np.zeros_like(df_Versicolor['SepalWidthCm']),
                                                    label='Versicolor', alpha=0.6)
plt.scatter(df_Virginica['SepalWidthCm'], np.zeros_like(df_Virginica['SepalWidthCm']),
                                                    label='Virginica', alpha=0.6)

plt.xlabel('Sepal Width')
plt.title('Sepal Width Distribution by Iris Variety')
plt.legend()
plt.show()
```

OUTPUT:

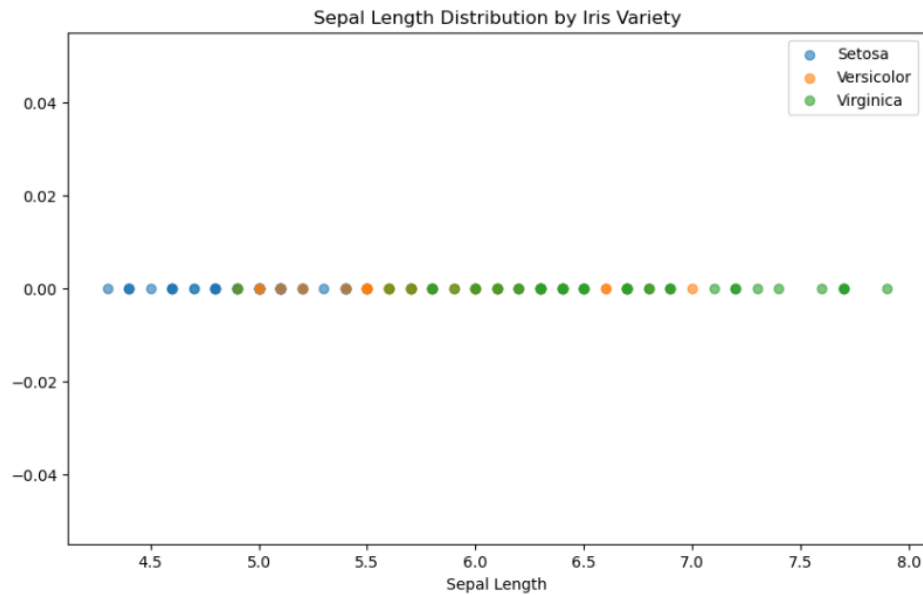


UNIVARIATE FOR SEPAL LENGTH

```
plt.figure(figsize=(10, 6))
plt.scatter(df_Setosa['SepalLengthCm'], np.zeros_like(df_Setosa['SepalLengthCm']),
                                                    label='Setosa', alpha=0.6)
plt.scatter(df_Versicolor['SepalLengthCm'], np.zeros_like(df_Versicolor['SepalLengthCm']),
                                                    label='Versicolor', alpha=0.6)
plt.scatter(df_Virginica['SepalLengthCm'], np.zeros_like(df_Virginica['SepalLengthCm']),
                                                    label='Virginica', alpha=0.6)

plt.xlabel('Sepal Length')
plt.title('Sepal Length Distribution by Iris Variety')
plt.legend()
plt.show()
```

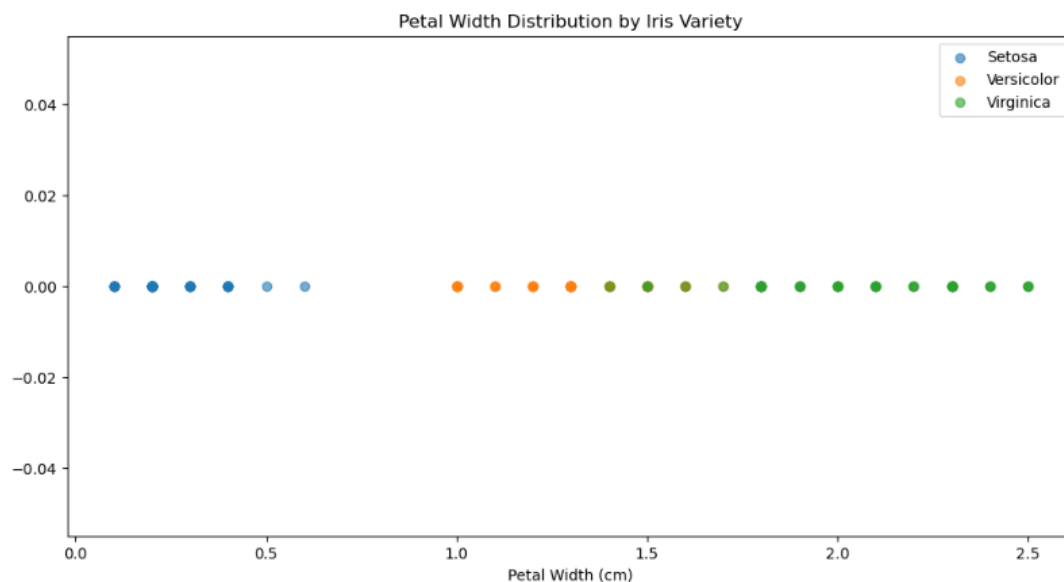
OUTPUT:



UNIVARIATE FOR PETAL WIDTH

```
plt.figure(figsize=(12, 6))
plt.scatter(df_Setosa['PetalWidthCm'], np.zeros_like(df_Setosa['PetalWidthCm']), label='Setosa', alpha=0.6)
plt.scatter(df_Versicolor['PetalWidthCm'], np.zeros_like(df_Versicolor['PetalWidthCm']), label='Versicolor',
alpha=0.6)
plt.scatter(df_Virginica['PetalWidthCm'], np.zeros_like(df_Virginica['PetalWidthCm']), label='Virginica',
alpha=0.6)
plt.xlabel('Petal Width (cm)')
plt.title('Petal Width Distribution by Iris Variety')
plt.legend()
plt.show()
```

OUTPUT:

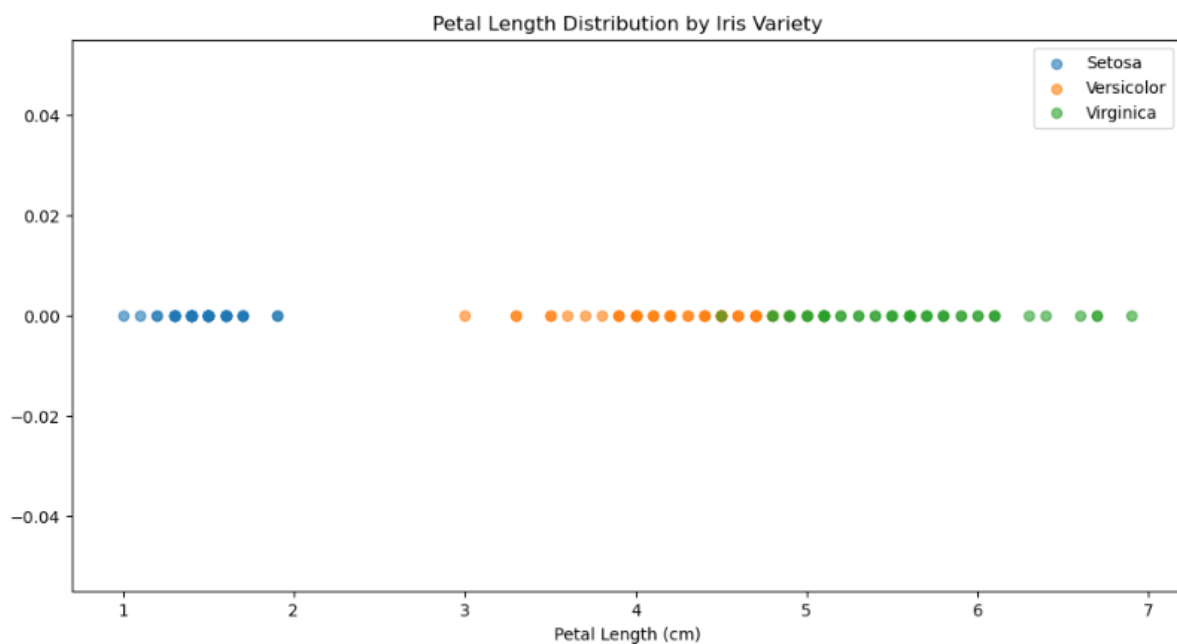


UNIVARIATE FOR PETAL LENGTH

```
plt.figure(figsize=(12, 6))
plt.scatter(df_Setosa['PetalLengthCm'], np.zeros_like(df_Setosa['PetalLengthCm']), label='Setosa', alpha=0.6)
plt.scatter(df_Versicolor['PetalLengthCm'], np.zeros_like(df_Versicolor['PetalLengthCm']), label='Versicolor',
            alpha=0.6)
plt.scatter(df_Virginica['PetalLengthCm'], np.zeros_like(df_Virginica['PetalLengthCm']), label='Virginica',
            alpha=0.6)

plt.xlabel('Petal Length (cm)')
plt.title('Petal Length Distribution by Iris Variety')
plt.legend()
plt.show()
```

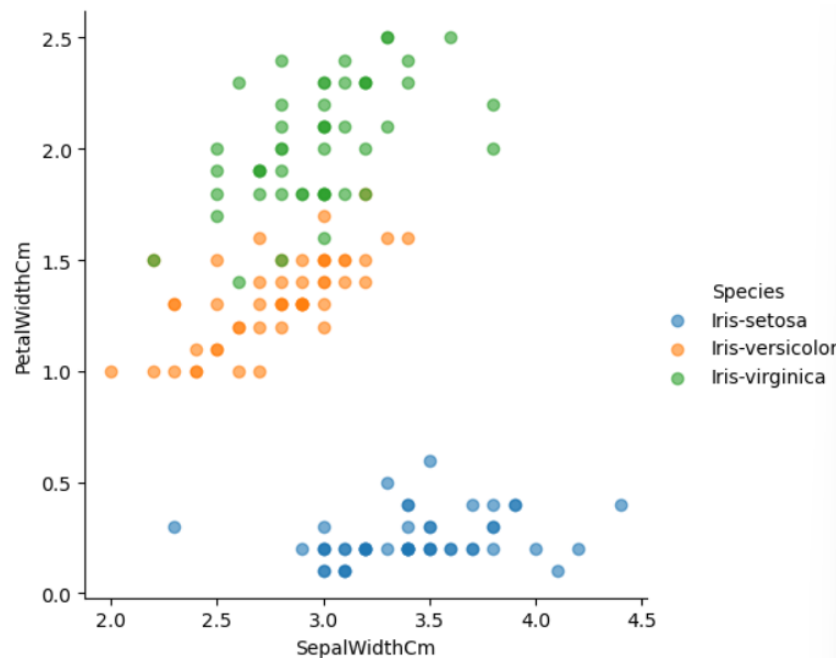
OUTPUT:



BIVARIATE SEPAL.WIDTH VS PETAL.WIDTH

```
g = sns.FacetGrid(df, hue='Species', height=5)
g.map(plt.scatter, 'SepalWidthCm', 'PetalWidthCm', alpha=0.6)
g.add_legend()
plt.show()
```

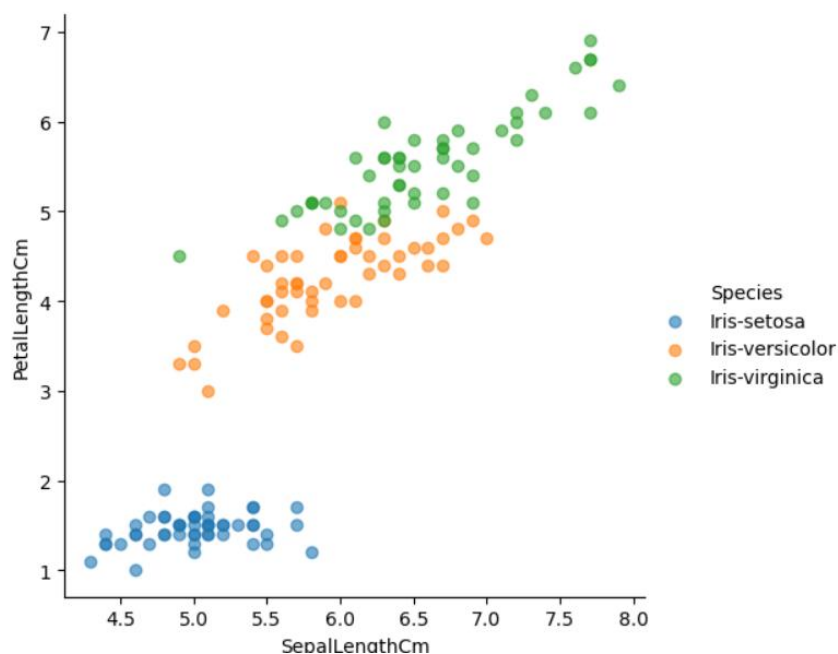

OUTPUT:



BIVARIATE SEPAL.LENGTH VS PETAL.LENGTH

```
g = sns.FacetGrid(df, hue='Species', height=5)
g.map(plt.scatter, 'SepalLengthCm', 'PetalLengthCm', alpha=0.6)
g.add_legend()
plt.show()
```

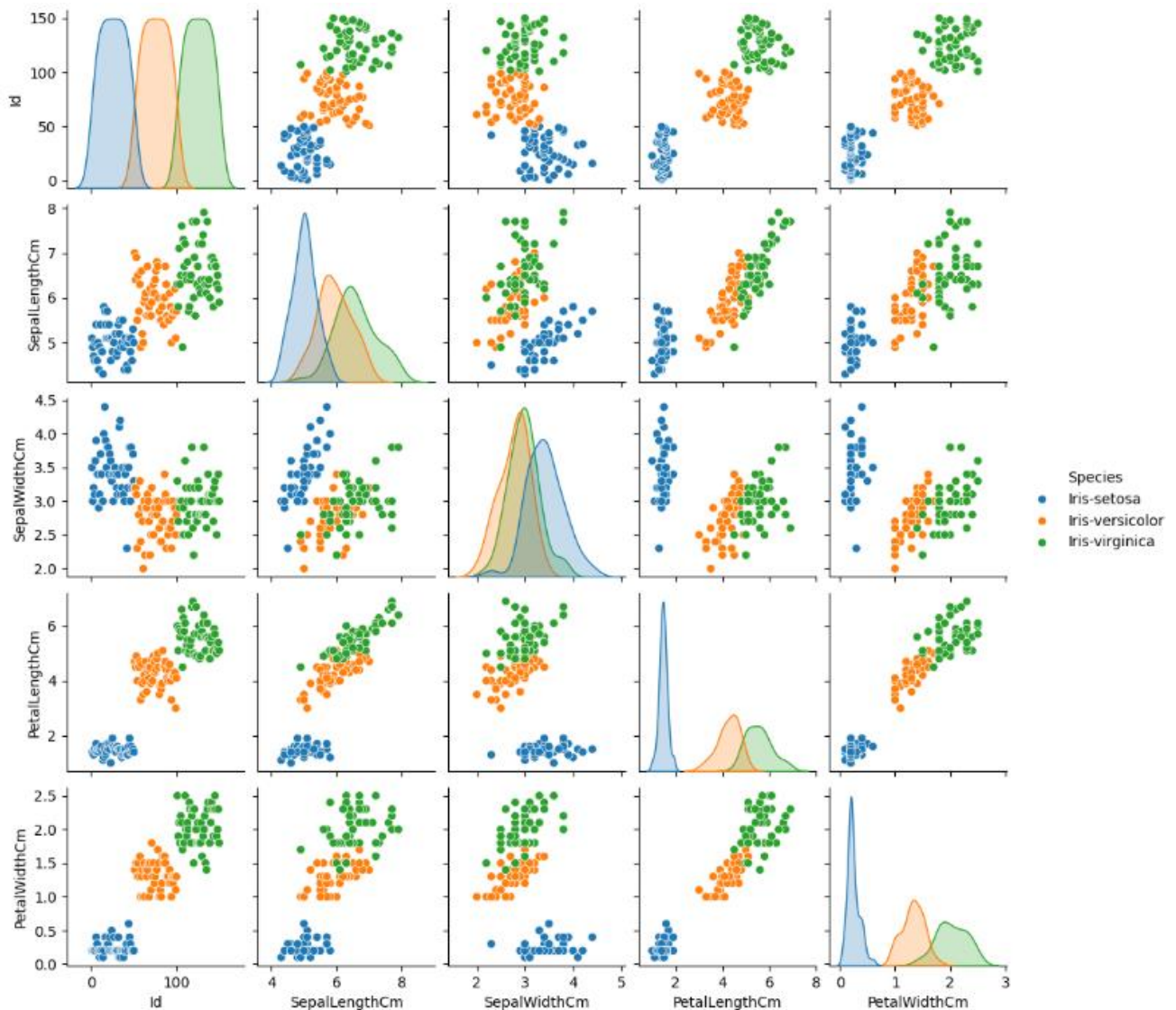
OUTPUT:



MULTIVARIATE ALL THE FEATURES

```
g=sns.pairplot(df,hue='Species',height=2)
plt.show()
```

OUTPUT:



RESULT:

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

Ex. No: 2	SIMPLE LINEAR REGRESSION USING LEAST SQUARE METHOD
Date: 31/1/2025	

AIM:

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

ALGORITHM:

STEP 1: Start the program

STEP 2: Import the necessary libraries and Extract the independent variable X and dependent variable y, and reshape them into 2D arrays if necessary.

STEP 3: Calculate the mean of X and y, then compute the slope (m) and intercept (b) using the formulas for simple linear regression.

STEP 4: Use the slope and intercept to make predictions for each X value. Calculate the Total Sum of Squares (TSS), Residual Sum of Squares (RSS), and the R-squared value to assess model accuracy.

STEP 5: Plot the original data points as a scatter plot and the regression line using the predicted values.

STEP 6: Print the calculated slope, intercept, and R-squared value to complete the program.

STEP 7: Stop

PROGRAM:

```
import numpy as np
import matplotlib.pyplot as plt

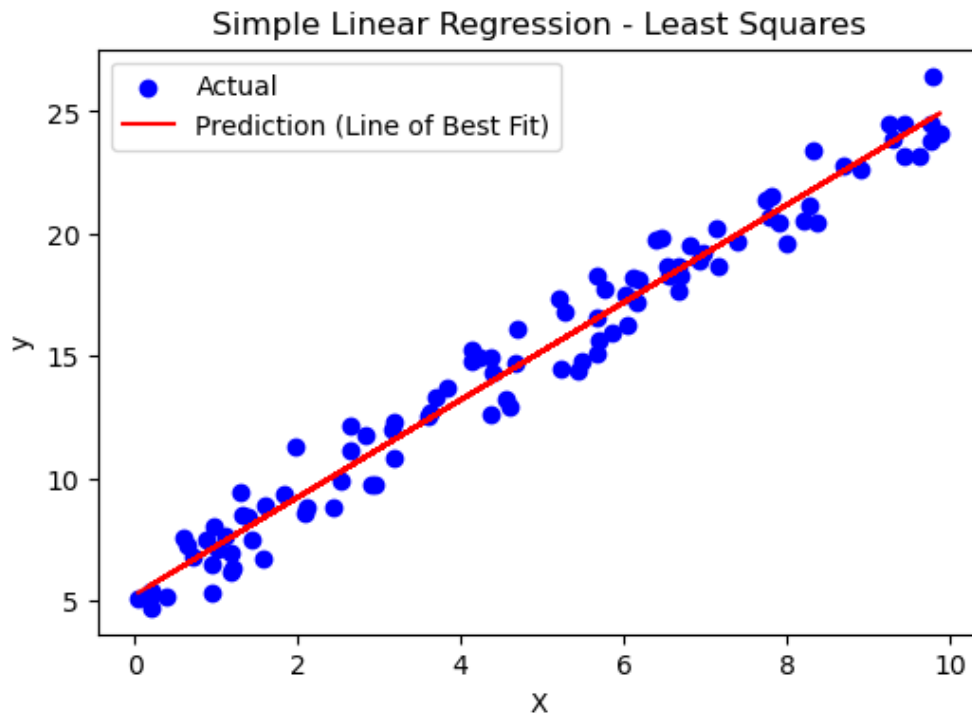
np.random.seed(0)          # 1. Simulate data (y = 2x + 5 + noise)
X = np.random.rand(100) * 10
noise = np.random.randn(100)
y = 2 * X + 5 + noise
x_mean = np.mean(X)        # 2. Least Squares Calculation
y_mean = np.mean(y)
numerator = np.sum((X - x_mean) * (y - y_mean))
denominator = np.sum((X - x_mean) ** 2)
slope = numerator / denominator
intercept = y_mean - slope * x_mean
y_pred = slope * X + intercept  # 3. Predictions
plt.figure(figsize=(6,4))
plt.scatter(X, y, label="Actual", color="blue")
plt.plot(X, y_pred, color="red", label="Prediction (Line of Best Fit)")
plt.title("Simple Linear Regression - Least Squares")
plt.xlabel("X")
```

```

plt.ylabel("y")
plt.legend()
plt.show()
mse = np.mean((y - y_pred) ** 2)          # 5. Performance Metrics
r2 = 1 - (np.sum((y - y_pred)**2) / np.sum((y - np.mean(y))**2))
print(f"Intercept: {intercept:.2f}")
print(f"Slope: {slope:.2f}")
print(f"Mean Squared Error (MSE): {mse:.2f}")
print(f"R2 Score: {r2:.2f}")

```

OUTPUT:



```

Intercept: 5.22
Slope: 1.99
Mean Squared Error (MSE): 0.99
R2 Score: 0.97

```

RESULT:

Thus, the Python program to implement simple linear regression using least squares method is analyzed, and the actual versus predicted features are plotted using a scatter plot.

Ex. No: 3	LOGISTIC REGRESSION MODEL
Date: 7/2/2025	

AIM:

To implement python program for the logistic model using DigitalAd dataset.

ALGORITHM:

STEP 1: Start the program

STEP 2: Load the dataset

STEP 3: Display the shape and first 5 rows of the dataset to understand its structure

STEP 4: Separate features (X) and target (Y) and split the data into training and testing sets

STEP 5: Train a Logistic Regression model and Make predictions on the test set

STEP 6: Evaluate the model using confusion matrix and Print the accuracy of the model

STEP 7: Predict for a new customer and Output the prediction result

STEP 8: Stop the program

PROGRAM:

```
import pandas as pd
import numpy as np
data = pd.read_csv("DigitalAd_dataset.csv")
print(data.shape)
print(data.head(5))
X = data.iloc[:, :-1].values # All columns except the last one (features)
Y = data.iloc[:, -1].values # Last column as target
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.25, random_state=0)
from sklearn.preprocessing import StandardScaler # Feature scaling (standardizing the features)
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
from sklearn.linear_model import LogisticRegression # Train a Logistic Regression model
model = LogisticRegression(random_state=0)
model.fit(X_train, y_train)

y_pred = model.predict(X_test) # Make predictions on the test set
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred) # Evaluate the model using confusion matrix
print("Confusion Matrix: ")
print(cm)
```

```

print("Accuracy of the Model: {0}%".format(accuracy_score(y_test, y_pred) * 100))
age = int(input("Enter New Customer Age: "))          # Predict for a new customer
sal = int(input("Enter New Customer Salary: "))
newCust = [[age, sal]]
result = model.predict(sc.transform(newCust))          # Output the prediction result
print(result)
if result == 1:
    print("Customer will Buy")
else:
    print("Customer won't Buy")

```

OUTPUT:

```

(400, 3)
   Age  Salary  Status
0   18   82000       0
1   29   80000       0
2   47   25000       1
3   45   26000       1
4   46   28000       1
Confusion Matrix:
[[61  0]
 [20 19]]
Accuracy of the Model: 80.0%
Enter New Customer Age: 45
Enter New Customer Salary: 45000
[0]
Customer won't Buy

```

RESULT:

Thus, the python program to implement logistic regression for the given dataset is analyzed and the performance of the developed model is measured successfully.

Ex. No: 4	SINGLE LAYER PERCEPTRON
Date: 14/2/2025	

AIM:

To implement python program for the single layer perceptron.

ALGORITHM:

STEP 1: Start the program

STEP 2: Initialize weights and bias with random values

STEP 3: For each epoch, compute the weighted sum and apply the sigmoid activation function

STEP 4: Calculate the error and update weights and bias using the derivative of the sigmoid function

STEP 5: After training, use the learned weights and bias to make predictions.

STEP 6: Stop the program

PROGRAM:

```
import numpy as np
def sigmoid_func(x):
    return 1 / (1 + np.exp(-x))

def der(x):
    return sigmoid_func(x) * (1 - sigmoid_func(x))
bias = 0.2
weights = np.random.rand(2)
input_value = np.array([[1, 0], [1, 1], [0, 0], [0, 1]])
output = np.array([1, 1, 0, 0])

for epochs in range(15000):
    input_arr = input_value
    weighted_sum = np.dot(input_arr, weights) + bias
    first_output = sigmoid_func(weighted_sum)
    error = first_output - output
    total_error = np.square(np.subtract(first_output, output)).mean()

    first_der = error
    second_der = der(weighted_sum)
    derivative = first_der * second_der
    t_input = input_value.T
    final_derivative = np.dot(t_input, derivative)
```

```

weights = weights - (0.05 * final_derivative)
bias = bias - (0.05 * np.sum(derivative))
print("Weights:", weights)
print("Bias:", bias)
pred = np.array([1, 0])
result = np.dot(pred, weights) + bias
res = sigmoid_func(result)
print(res)

pred = np.array([1, 1])
result = np.dot(pred, weights) + bias
res = sigmoid_func(result)
print(res)

pred = np.array([0, 0])
result = np.dot(pred, weights) + bias
res = sigmoid_func(result)
print(res)

pred = np.array([0, 1])
result = np.dot(pred, weights) + bias
res = sigmoid_func(result)
print(res)

```

OUTPUT:

```

Weights: [ 6.94417618 -0.22212245]
Bias: -3.253488566147656
0.9756527449271702
0.9697799964846544
0.03720173277025323
0.03720173277025323

```

RESULT:

Thus, the Python program to implement a Single-Layer Perceptron using sigmoid activation for binary classification is analyzed, and the weights, bias, and predicted outputs are obtained after training.

Ex. No: 5	MULTILAYER PERCEPTRON WITH BACK PROPAGATION
Date: 21/2/2025	

AIM:

To implement multilayer perceptron with back propagation using python.

PROCEDURE:

STEP 1: Start the program

STEP 2: Create a class MLP with an `__init__` method that initializes weights and biases for the input-to-hidden and hidden-to-output layers.

STEP 3: Include activation functions like sigmoid and its derivative and Implement the `train()` method which performs forward propagation to calculate predictions, backward propagation to compute gradients, and updates weights and biases using gradient descent over a number of epochs.

STEP 4: Set up an XOR dataset with 2 inputs and 1 output. Instantiate the MLP with 2 input nodes, 4 hidden nodes, and 1 output node. Train the model using the XOR data for 10,000 epochs.

STEP 5: After training, pass the input data through the network again using the learned weights to generate predictions.

STEP 6: print the outputs to observe how well the model learned the XOR logic.

STEP 7: Stop the program

PROGRAM:

```
import numpy as np
class MLP:
    def __init__(self, input_size, hidden_size, output_size):
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.output_size = output_size

        # initialize weights matrix and biases
        self.W_input_hidden = np.random.rand(self.input_size, self.hidden_size)
        self.b_input_hidden = np.zeros((1, self.hidden_size))
        self.W_hidden_output = np.random.rand(self.hidden_size, self.output_size)
        self.b_hidden_output = np.zeros((1, self.output_size))

    def sigmoid(self, x):
        return 1 / (1 + np.exp(-x))

    def d_sigmoid(self, x):
        return x * (1 - x)
```

```

def train(self, input_data, target, epochs=1000, lr=0.2):
    for epoch in range(epochs):
        # Forward propagation
        hidden_layer_input = np.dot(input_data, self.W_input_hidden) + self.b_input_hidden
        hidden_layer_output = self.sigmoid(hidden_layer_input)
        output_layer_input = np.dot(hidden_layer_output, self.W_hidden_output) +
                                self.b_hidden_output

        output = self.sigmoid(output_layer_input)

        # Backward propagation
        output_error = target - output
        output_grad = output_error * self.d_sigmoid(output)
        hidden_error = np.dot(output_grad, self.W_hidden_output.T)
        hidden_grad = hidden_error * self.d_sigmoid(hidden_layer_output)

        # Update weights and biases using gradient descent
        self.W_hidden_output += np.dot(hidden_layer_output.T, output_grad) * lr
        self.b_hidden_output += np.sum(output_grad, axis=0, keepdims=True) * lr
        self.W_input_hidden += np.dot(input_data.T, hidden_grad) * lr
        self.b_input_hidden += np.sum(hidden_grad, axis=0, keepdims=True) * lr

        # Optionally, print error every 1000 epochs
        if epoch % 1000 == 0:
            error = np.mean(np.square(target - output)) # Mean Squared Error
            print(f'Epoch {epoch}, Error: {error}')

# Example usage:
if __name__ == "__main__":
    # XOR problem: 4 samples, 2 input features, 1 output
    X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
    y = np.array([[0], [1], [1], [0]])

    # Create MLP with 2 input nodes, 4 hidden nodes, and 1 output node
    mlp = MLP(input_size=2, hidden_size=4, output_size=1)
    mlp.train(X, y, epochs=10000) # Train the model

    print("Predictions after training:") # Test the model after training
    hidden_layer_input = np.dot(X, mlp.W_input_hidden) + mlp.b_input_hidden
    hidden_layer_output = mlp.sigmoid(hidden_layer_input)
    output_layer_input = np.dot(hidden_layer_output, mlp.W_hidden_output) + mlp.b_hidden_output
    predictions = mlp.sigmoid(output_layer_input)
    print(predictions)

```

OUTPUT:

```
Epoch 0, Error: 0.3525487770976961
Epoch 1000, Error: 0.16546836123157288
Epoch 2000, Error: 0.022134675143509714
Epoch 3000, Error: 0.006844220579949708
Epoch 4000, Error: 0.0036914169964121224
Epoch 5000, Error: 0.0024534156381819903
Epoch 6000, Error: 0.001812478058253225
Epoch 7000, Error: 0.0014263384514248863
Epoch 8000, Error: 0.0011704001423583428
Epoch 9000, Error: 0.000989268984991053
Predictions after training:
[[0.03327479]
 [0.97289057]
 [0.97188183]
 [0.02804434]]
```

RESULT:

Thus, the Python program to implement a simple multi-layer perceptron (MLP) for the XOR problem is analyzed, and the model successfully predicts the output after training by learning the non-linear relationships in the data.

Ex. No: 6	FACE RECOGNITION USING SVM CLASSIFIER
Date: 28/2/2025	

AIM:

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

PROCEDURE:

STEP 1: Start the program.

STEP 2: Import the necessary libraries and Load the dataset.

STEP 3: Flatten the images and apply PCA

STEP 4: Split the dataset into train & test sets and train SVM model

STEP 5: Make prediction and evaluate the accuracy.

STEP 6: Print the model accuracy and visualize some test images with predictions.

STEP 7: Stop the program.

PROGRAM:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from sklearn.svm import SVC
from sklearn.datasets import fetch_lfw_people
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from sklearn.metrics import accuracy_score

lfw = fetch_lfw_people(min_faces_per_person=100, resize=0.4, download_if_missing=True)
X, y = lfw.images, lfw.target # Images and labels

X_flat = X.reshape(X.shape[0], -1) # Convert images to 1D array
pca = PCA(n_components=100).fit(X_flat) # Reduce dimensions to 100 principal components
X_pca = pca.transform(X_flat)

X_train, X_test, y_train, y_test = train_test_split(X_pca, y, test_size=0.2, random_state=42)

svm = SVC(kernel='linear')
svm.fit(X_train, y_train)
y_pred = svm.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy * 100:.2f}%")

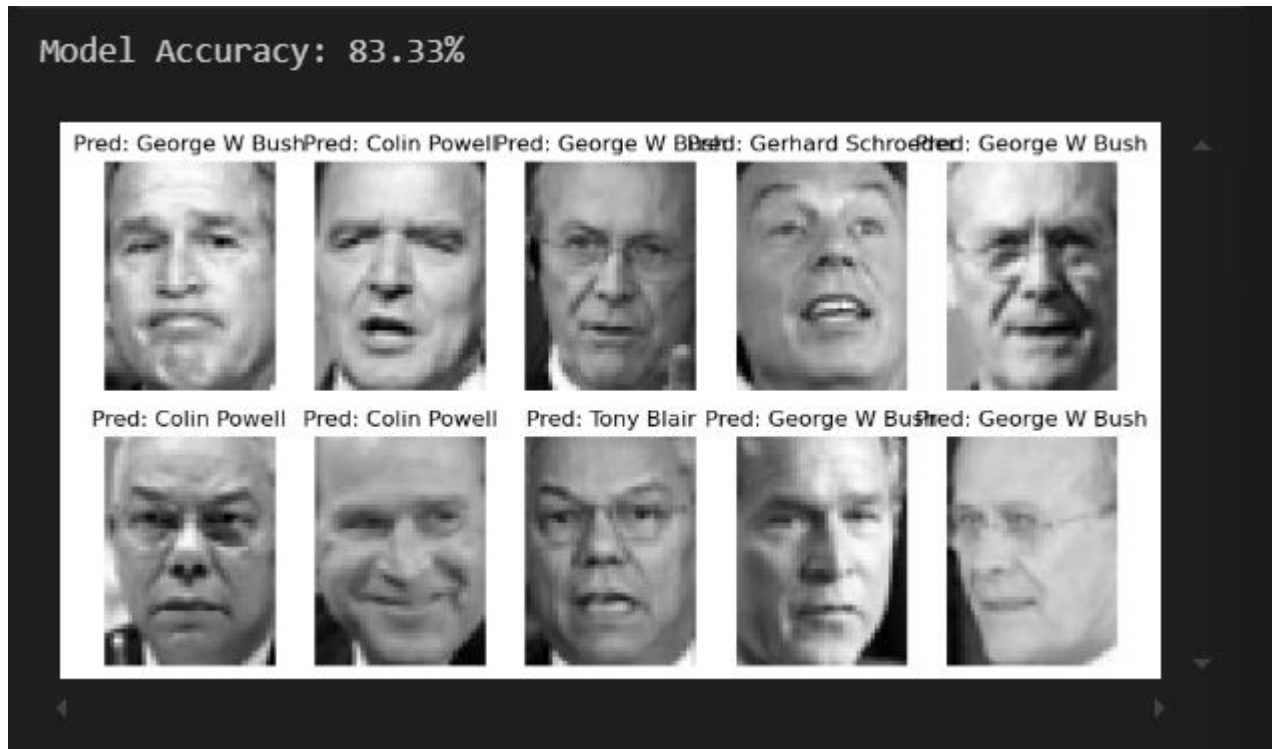
fig, axes = plt.subplots(2, 5, figsize=(10, 5)) # Create a grid of 2 rows, 5 columns
for i, ax in enumerate(axes.ravel()):
```

```

ax.imshow(X[i], cmap='gray') # Show actual face image
ax.set_title(f'Pred: {lfw.target_names[y_pred[i]]}') # Predicted name
ax.axis('off')
plt.show()

```

OUTPUT:



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.

Ex. No: 7	DECISION TREE IMPLEMENTATION
Date: 7/3/2025	

AIM:

To implement a Decision Tree Classifier on the Iris dataset and visualize the decision tree structure.

ALGORITHM:

STEP 1: Start the program.

STEP 2: Import necessary modules from sklearn and matplotlib. Load the Iris dataset

STEP 3: Split it into features (X) and target (y) and Split the dataset into training and testing sets

STEP 4: Create a DecisionTreeClassifier instance, fit it with the training data, and make predictions on the test set.

STEP 5: Calculate and print the accuracy using accuracy_score().

STEP 6: Use plot_tree() to visualize the trained decision tree with feature and class names, and display it using plt.show().

STEP 7: Stop the program.

PROGRAM:

```

from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

iris = datasets.load_iris()
X, y = iris.data, iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
print(f"Accuracy: {accuracy_score(y_test, y_pred) * 100:.2f}%")

plt.figure(figsize=(12, 10))
plot_tree(clf, filled=True, feature_names=iris.feature_names, class_names=iris.target_names)
plt.show()

```

OUTPUT:

Accuracy: 100.00%



RESULT:

Thus, the model was trained successfully and achieved an accuracy of **100.00%** on the test data, with the decision tree plotted for analysis.

Ex. No: 8	BOOSTING ALGORITHM IMPLEMENTATION
Date: 28/3/2025	

AIM:

To implement the AdaBoost ensemble method using decision stumps on a synthetic dataset and visualize decision boundaries.

ALGORITHM:

STEP 1: Start the program

STEP 2: Generate a binary classification dataset.

STEP 3: Split the data into training and testing sets using `train_test_split()`.

STEP 4: Train an `AdaBoostClassifier` with `DecisionTreeClassifier(max_depth=1)` as base estimator.

STEP 5: Predict results, calculate accuracy, and plot the decision boundaries.

STEP 6: Stop the program

PROGRAM:

```

from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
import numpy as np

X, y = make_classification(n_samples=1000, n_features=2, n_informative=2, n_redundant=0,
                           random_state=42)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

boost = AdaBoostClassifier(DecisionTreeClassifier(max_depth=1), n_estimators=50,
                           random_state=42)

boost.fit(X_train, y_train)

y_pred = boost.predict(X_test)                                # Predict and evaluate model performance
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")

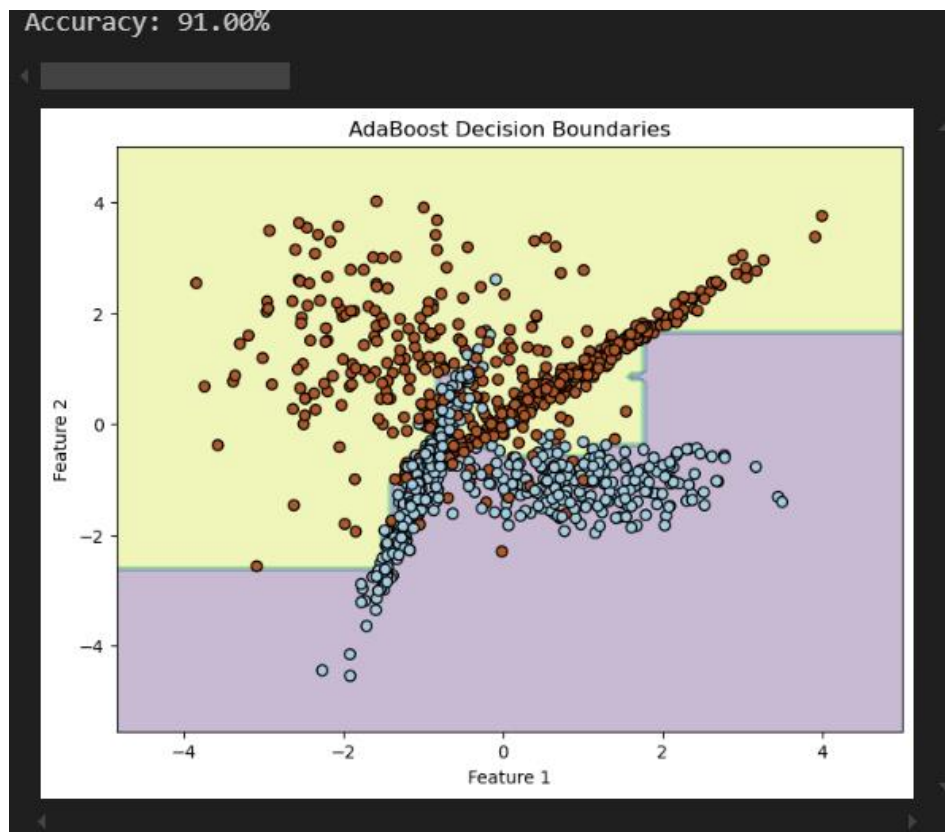
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1          # Visualization - Decision Boundaries
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.linspace(x_min, x_max, 100), np.linspace(y_min, y_max, 100))

```

```
Z = boost.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

plt.figure(figsize=(8, 6))
plt.contourf(xx, yy, Z, alpha=0.3)
plt.scatter(X[:, 0], X[:, 1], c=y, edgecolors="k", cmap=plt.cm.Paired)
plt.title("AdaBoost Decision Boundaries")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()
```

OUTPUT:



RESULT:

Thus the python program for the AdaBoost model is implemented and achieved an accuracy of **100.00%** on the test set, and also the decision boundaries were plotted successfully.

Ex. No: 9	K-NEAREST NEIGHBOUR AND K-MEANS CLUSTERING
Date: 4/4/2025	

AIM:

To implement K-Nearest Neighbors (KNN) classification and K-Means clustering on the Iris dataset and visualize the clustering results.

ALGORITHM:

STEP 1: Start the program

STEP 2: Load the required libraries and dataset

STEP 3: Split the data into training and testing sets using train_test_split().

STEP 4: Train a KNN classifier and evaluate its accuracy on the test set

STEP 5: Apply K-Means clustering and visualize the clusters.

STEP 6: Stop the program

PROGRAM:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.cluster import KMeans
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

iris = load_iris()
X, y = iris.data, iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

knn = KNeighborsClassifier(n_neighbors=5)          # Train KNN classifier
knn.fit(X_train, y_train)

y_pred_knn = knn.predict(X_test)                 # Predict and evaluate KNN
print(f"KNN Accuracy: {accuracy_score(y_test, y_pred_knn) * 100:.2f}%")  # Train K-Means clustering
kmeans = KMeans(n_clusters=3, random_state=42, n_init=10)
kmeans.fit(X)

plt.scatter(X[:, 0], X[:, 1], c=kmeans.labels_, cmap='rainbow', alpha=0.6)
plt.scatter(kmeans.cluster_centers_[0], kmeans.cluster_centers_[1], marker='x', s=200, c='black')
plt.title("K-Means Clustering - Iris Dataset")
plt.show()
```

OUTPUT:



RESULT:

Thus the python program for the KNN model achieved an accuracy of **100.00%** on the test set, and K-Means clustering groups were successfully visualized.

Ex. No: 10	DIMENSIONALITY REDUCTION USING PCA
Date: 11/4/2025	

AIM:

To apply Principal Component Analysis (PCA) for dimensionality reduction on the Digits dataset and visualize the results.

ALGORITHM:

STEP 1: Start the program

STEP 2: Load the necessary libraries and a dataset

STEP 3: Apply PCA to reduce the dataset to two principal components.

STEP 4: Transform the dataset using the fitted PCA model.

STEP 5: Visualize the transformed data in a 2D scatter plot.

STEP 6: Stop the program

PROGRAM:

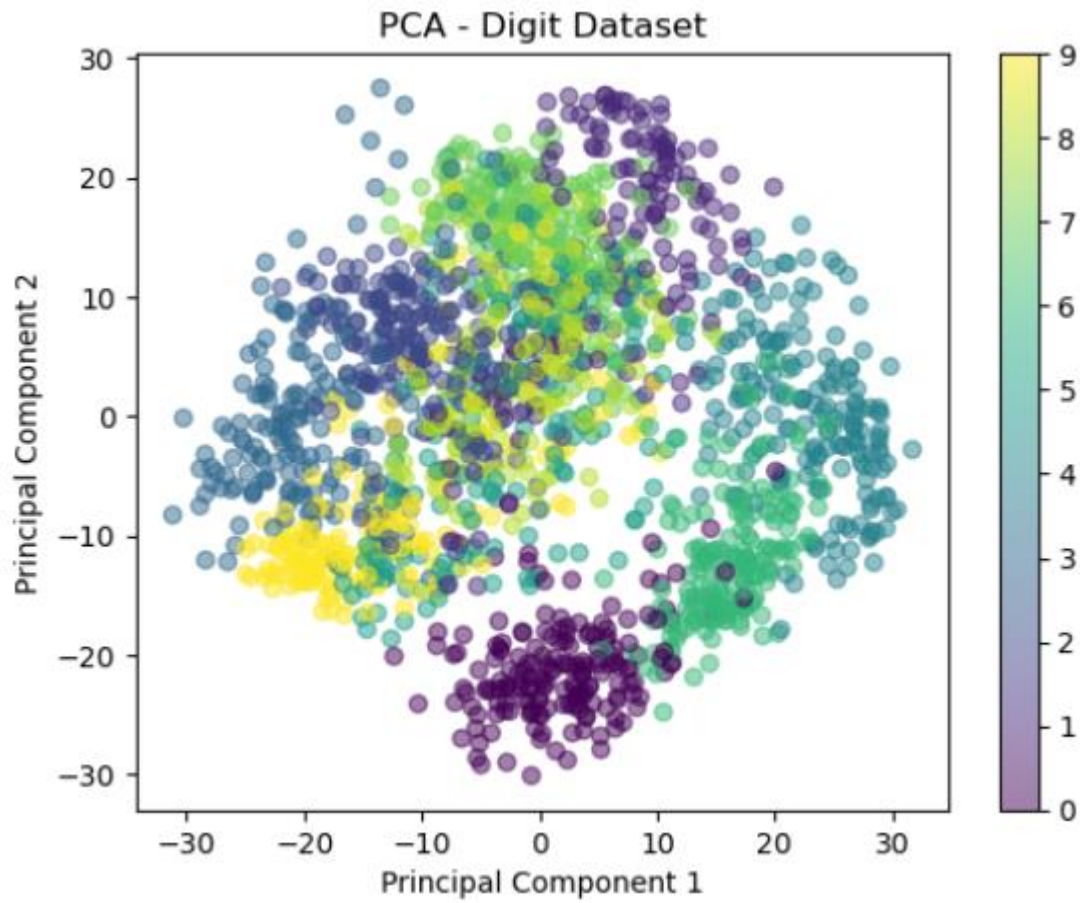
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.datasets import load_digits

# Load dataset
digits = load_digits()
X = digits.data

# Apply PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

# Plot PCA results
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=digits.target, cmap='viridis', alpha=0.5)
plt.colorbar()
plt.title("PCA - Digit Dataset")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.show()
```

OUTPUT:



RESULT:

Thus the python program for PCA successfully reduced the data to 2 principal components, and the digit classes were visualized in a 2D scatter plot.

Ex. No: 11	Mini project – Develop a simple application using TensorFlow / Keras
Date: 11/4/2025	

PROJECT TITLE: *Digit Recognition Using Neural Networks with MNIST Dataset*

BUSINESS CASE STUDY:

In modern smart applications like postal automation, digital form reading, banking, and smart classrooms, recognizing handwritten digits quickly and accurately is essential. Manual verification processes are slow and prone to errors, especially when dealing with large volumes of data. This project addresses this challenge by building a machine learning-based solution that automates digit recognition using the MNIST dataset, offering a reliable method for real-time classification and minimizing human effort.

PROBLEM STATEMENT:

Traditional methods of recognizing handwritten digits are limited by manual labor, inconsistency, and time constraints. These methods are ineffective in systems where rapid decision-making is essential. The need for an automated, intelligent system that can accurately interpret handwritten digits in real-time is crucial for improving service delivery and operational efficiency.

OBJECTIVES:

1. Develop a neural network model capable of recognizing handwritten digits from the MNIST dataset.
2. Automate the digit classification process using deep learning for improved speed and accuracy.
3. Evaluate the model's performance using classification accuracy on test data.
4. Provide visual feedback by displaying predicted vs. actual digits to enhance model interpretability.

BUSINESS PROBLEMS:

1. **Manual Digit Verification:** Human validation of handwritten digits is time-consuming and error-prone.
2. **Low Accuracy in Existing Systems:** Traditional rule-based recognition systems are not adaptable to varying writing styles and noise.
3. **Scalability Challenges:** Existing methods are not scalable for real-time classification of large datasets.
4. **Lack of Visualization & Interpretability:** Many models do not provide clear visualization of predictions, making debugging and trust difficult.

DATASET DESCRIPTION:

The **MNIST dataset** is a benchmark dataset in machine learning, consisting of:

- **60,000 training images** and **10,000 test images** of handwritten digits (0–9).
- Each image is **28x28 pixels in grayscale**, flattened into 784 input features.

- Labels are integers from 0 to 9, representing the digit shown in the image.
- It is pre-cleaned and balanced, ensuring fairness in model training and evaluation.

STEPS INVOLVED:

STEP 1: Load and preprocess the MNIST dataset by normalizing image pixel values to a range of 0–1 and applying

one-hot encoding to the digit labels for multi-class classification.

STEP 2: Build a neural network using the Keras Sequential API with a Flatten input layer, one Dense hidden layer

using ReLU activation, and an output Dense layer with Softmax for predicting digit classes (0–9).

STEP 3: Compile the model using the Adam optimizer and categorical cross entropy loss, then train it for 5 epochs

with a validation split to monitor performance.

STEP 4: Evaluate the trained model on the test dataset and display the accuracy score to assess its classification

performance.

STEP 5: Select a random test image, predict the digit using the model, and display the image with its predicted and

actual label for visual confirmation.

SOURCE CODE:

```
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt
import numpy as np

# Load and preprocess the MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0 # Normalize pixel values
y_train_cat = to_categorical(y_train)
y_test_cat = to_categorical(y_test)

# Build the model
model = Sequential([
    Flatten(input_shape=(28, 28)),
    Dense(128, activation='relu'),
    Dense(10, activation='softmax') # 10 classes for digits 0-9
])

# Compile and train the model
```

```

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(x_train, y_train_cat, epochs=5, batch_size=32, validation_split=0.2)

# Evaluate the model
test_loss, test_acc = model.evaluate(x_test, y_test_cat)
print(f'Test accuracy: {test_acc:.2f}')

# Pick a random sample from the test set
index = np.random.randint(0, len(x_test))
sample_image = x_test[index]
sample_label = y_test[index]

# Predict the digit
prediction = model.predict(sample_image.reshape(1, 28, 28))
predicted_class = np.argmax(prediction)

# Show the image and prediction
plt.imshow(sample_image, cmap='gray')
plt.title(f'Predicted: {predicted_class}, Actual: {sample_label}')
plt.axis('off')
plt.show()

```

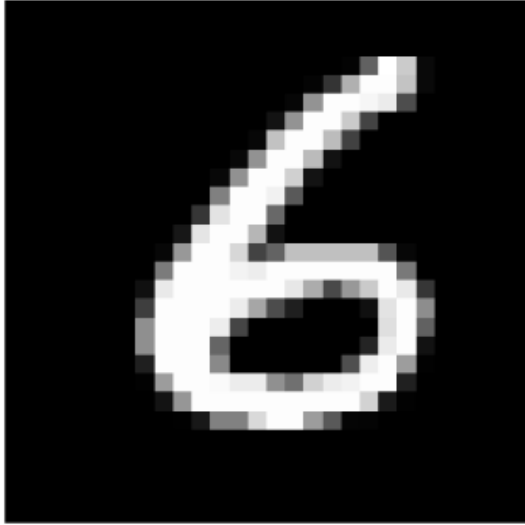
OUTPUT:

```

Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ————— 0s 0us/step
/usr/local/lib/python3.11/dist-packages/keras/src/layers/resizing/flatten.py:37: UserWarning: Do not pass an `input_shape`
super().__init__(**kwargs)
Epoch 1/5
1500/1500 ————— 11s 6ms/step - accuracy: 0.8645 - loss: 0.4829 - val_accuracy: 0.9563 - val_loss: 0.1528
Epoch 2/5
1500/1500 ————— 9s 5ms/step - accuracy: 0.9595 - loss: 0.1361 - val_accuracy: 0.9670 - val_loss: 0.1167
Epoch 3/5
1500/1500 ————— 8s 5ms/step - accuracy: 0.9760 - loss: 0.0816 - val_accuracy: 0.9684 - val_loss: 0.1018
Epoch 4/5
1500/1500 ————— 7s 5ms/step - accuracy: 0.9827 - loss: 0.0593 - val_accuracy: 0.9725 - val_loss: 0.0947
Epoch 5/5
1500/1500 ————— 8s 6ms/step - accuracy: 0.9867 - loss: 0.0458 - val_accuracy: 0.9745 - val_loss: 0.0828
313/313 ————— 1s 2ms/step - accuracy: 0.9724 - loss: 0.0848
Test accuracy: 0.98
1/1 ————— 0s 87ms/step

```

Predicted: 6, Actual: 6



BUSINESS INFERENCE:

The implementation of a neural network-based digit recognition system using the MNIST dataset showcases the practical benefits of integrating artificial intelligence into everyday operational workflows. From a business perspective, the model delivers the following insights:

1. **Operational Efficiency**

Automating the digit recognition process reduces manual entry errors and accelerates data processing times in sectors like banking (e.g., check processing), postal services (e.g., ZIP code reading), and digitized form handling.

2. **Cost Reduction**

By minimizing the need for human validation and rework due to misreads or entry errors, businesses can save significantly on labor and administrative costs.

3. **Scalability**

The model handles large datasets effectively, making it suitable for deployment in environments that process thousands of digit-based transactions daily, without loss in performance or accuracy.

4. **Customer Experience Enhancement**

Faster and more accurate processing directly improves customer satisfaction by reducing waiting times and ensuring prompt services.

5. **Foundation for Advanced AI Applications**

The success of this basic digit recognition task lays the foundation for more complex AI-driven solutions such as intelligent document processing (IDP), optical character recognition (OCR), and smart healthcare form analysis.

CONCLUSION:

The project successfully implemented a deep learning-based approach for handwritten digit recognition using TensorFlow and the MNIST dataset. The trained model achieved high accuracy and demonstrated strong generalization on unseen data. Through visualization of predictions and a structured training pipeline, the model proved to be an effective solution for intelligent digit recognition tasks. This mini-project highlights the practical applicability of neural networks in digit classification problems and lays the groundwork for future advancements, including integration into smart applications and real-time recognition systems.