

|  |  |
| --- | --- |
| **EXP NO: 1** | **A PYTHON PROGRAM TO IMPLEMENT UNIVARIATE, BIVARIATE AND MULTIVARIATE REGRESION** |
| **DATE: 24/1/25** |

**AIM:**

To implement a python program using univariate, bivariate and multivariate regression

features for a given Housing dataset.

**ALGORITHM:**

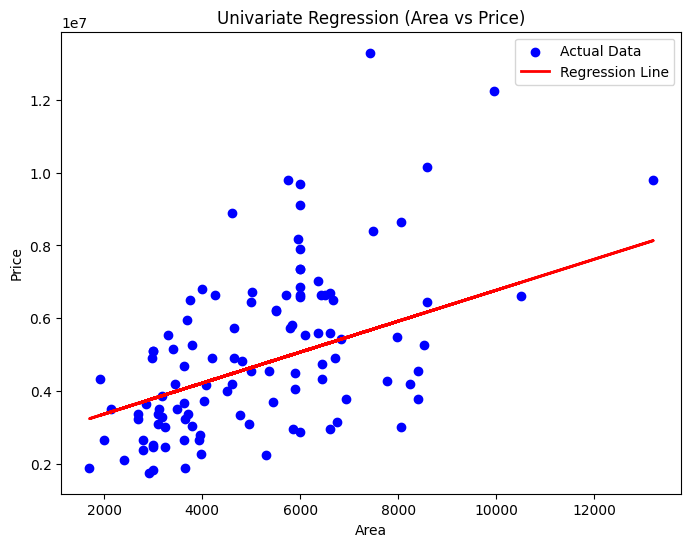
**Step 1:** Load and preview the dataset  
**Step 2:** Handle missing values  
**Step 3:** Univariate regression (1 feature → price)  
**Step 4:** Plot for univariate regression  
**Step 5:** Bivariate regression (2 features → price)  
**Step 6:** Plot for bivariate regression  
**Step 7:** Multivariate regression (multiple features → price)  
**Step 8:** Train the model  
**Step 9:** Make predictions  
**Step 10:** Evaluate performance (R² score)

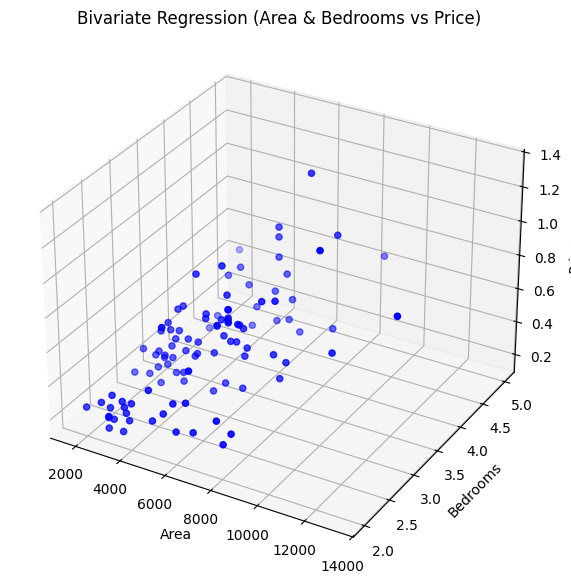
**SOURCE CODE:**

import pandas as pdimport numpy as npimport matplotlib.pyplot as pltimport seaborn as snsfrom sklearn.model\_selection import train\_test\_splitfrom sklearn.linear\_model import LinearRegressionfrom sklearn.preprocessing import LabelEncoderfrom mpl\_toolkits.mplot3d import Axes3Dfrom sklearn.metrics import mean\_squared\_error, r2\_score# Step 1: Load datasetfile\_path = "/content/Housing.csv"

df = pd.read\_csv(file\_path)# Step 2: Preprocess data (convert categorical variables)le = LabelEncoder()df['mainroad'] = le.fit\_transform(df['mainroad'])df['guestroom'] = le.fit\_transform(df['guestroom'])df['basement'] = le.fit\_transform(df['basement'])df['hotwaterheating'] = le.fit\_transform(df['hotwaterheating'])df['airconditioning'] = le.fit\_transform(df['airconditioning'])df['prefarea'] = le.fit\_transform(df['prefarea'])df['furnishingstatus'] = le.fit\_transform(df['furnishingstatus'])# Step 3: Univariate Regression (Price vs Area)X\_uni = df[['area']]y = df['price']X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_uni, y, test\_size=0.2, random\_state=42)model\_uni = LinearRegression()model\_uni.fit(X\_train, y\_train)y\_pred\_uni = model\_uni.predict(X\_test)# Plot Univariate Regressionplt.figure(figsize=(8,6))plt.scatter(X\_test, y\_test, color='blue', label='Actual Data')plt.plot(X\_test, y\_pred\_uni, color='red', linewidth=2, label='Regression Line')plt.xlabel('Area')plt.ylabel('Price')plt.title('Univariate Regression (Area vs Price)')plt.legend()plt.show()# Step 4: Bivariate Regression (Price vs Area & Bedrooms)X\_bi = df[['area', 'bedrooms']]X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_bi, y, test\_size=0.2, random\_state=42)model\_bi = LinearRegression()model\_bi.fit(X\_train, y\_train)y\_pred\_bi = model\_bi.predict(X\_test)# Plot Bivariate Regression in 3Dfig = plt.figure(figsize=(10,7))ax = fig.add\_subplot(111, projection='3d')ax.scatter(X\_test['area'], X\_test['bedrooms'], y\_test, color='blue', label='Actual Data')ax.set\_xlabel('Area')ax.set\_ylabel('Bedrooms')ax.set\_zlabel('Price')ax.set\_title('Bivariate Regression (Area & Bedrooms vs Price)')plt.show()# Step 5: Multivariate Regression (Using all features)X\_multi = df.drop(columns=['price'])X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_multi, y, test\_size=0.2, random\_state=42)model\_multi = LinearRegression()model\_multi.fit(X\_train, y\_train)y\_pred\_multi = model\_multi.predict(X\_test)# Model Evaluationmse = mean\_squared\_error(y\_test, y\_pred\_multi)r2 = r2\_score(y\_test, y\_pred\_multi)print(f"Multivariate Regression R² Score: {r2:.4f}")print(f"Multivariate Regression MSE: {mse:.2f}")# Residual Plotresiduals = y\_test - y\_pred\_multiplt.figure(figsize=(8,6))sns.histplot(residuals, kde=True, color='purple')plt.xlabel('Residuals')plt.ylabel('Frequency')plt.title('Residual Distribution - Multivariate Regression')plt.show()

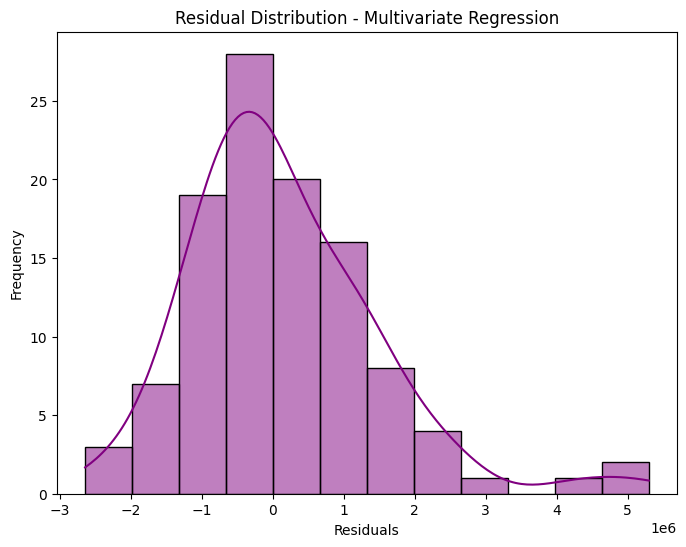
**OUTPUT:**





Multivariate Regression R² Score: 0.6495

Multivariate Regression MSE: 1771751116594.04



**RESULT:**

Thus, the python program to implement univariate, bivariate and multivariate regression

features for the given housing dataset is analyzed and the features are plotted using scatter plot.

|  |  |
| --- | --- |
| **EXP NO:** **2** | **A PYTHON PROGRAM TO IMPLEMENT SIMPLE LINEAR REGRESSION USING LEAST SQUARE METHOD** |
| **DATE:** **31/1/25** |

**AIM:**

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

square method.

**ALGORITHM:**

**Step 1:** Import necessary libraries (numpy, matplotlib, pandas).  
**Step 2:** Read the dataset (headbrain.csv) and explore data using .head(), .info(), and .describe().  
**Step 3:** Extract Head Size as X (independent variable) and Brain Weight as y (dependent variable).  
**Step 4:** Compute the mean of X and y to prepare for coefficient calculations.  
**Step 5:** Calculate slope (b1) and intercept (b0) using the Least Squares formula.  
**Step 6:** Generate predictions (y\_pred) using the linear equation y\_pred = b0 + b1 \* X.  
**Step 7:** Plot the regression line over the actual data points (X, y).  
**Step 8:** Plot residuals (differences between actual and predicted values) to analyze model fit.  
**Step 9:** Compute the R-squared value, which indicates the proportion of variance explained by the model.  
**Step 10:** Display results (Intercept, Slope, and R² Score) to evaluate model performance.

**SOURCE CODE:**

import numpy as npimport matplotlib.pyplot as pltimport pandas as pd# Step 1: Import necessary libraries# Step 2: Read the datasetfile\_path = "/content/headbrain.csv"data = pd.read\_csv(file\_path)data.head()data.info()data.describe()# Step 3: Prepare the dataX = data['Head Size(cm^3)'].valuesy = data['Brain Weight(grams)'].values# Step 4: Calculate the meanmean\_x, mean\_y = np.mean(X), np.mean(y)# Step 5: Calculate the coefficientsb1 = np.sum((X - mean\_x) \* (y - mean\_y)) / np.sum((X - mean\_x) \*\* 2)b0 = mean\_y - b1 \* mean\_x# Step 6: Make predictionsy\_pred = b0 + b1 \* X# Step 7: Plot the regression lineplt.figure(figsize=(8, 6))plt.scatter(X, y, color='blue', label='Actual data', alpha=0.6)plt.plot(X, y\_pred, color='red', label='Regression line', linewidth=2)plt.xlabel('Head Size (cm³)')plt.ylabel('Brain Weight (grams)')plt.legend()plt.title('Linear Regression using Least Squares')plt.show()# Step 8: Plot the residualsresiduals = y - y\_predplt.figure(figsize=(8, 6))plt.scatter(X, residuals, color='purple', alpha=0.6)

plt.axhline(y=0, color='black', linestyle='--', linewidth=1)plt.xlabel('Head Size (cm³)')plt.ylabel('Residuals')plt.title('Residual Plot')plt.show()# Step 9: Calculate the R-squared valueTSS = np.sum((y - mean\_y) \*\* 2)RSS = np.sum((y - y\_pred) \*\* 2)R2 = 1 - (RSS / TSS)# Step 10: Display the resultsprint(f"Intercept: {b0:.2f}")print(f"Slope: {b1:.2f}")print(f"R-squared Value: {R2:.4f}")

**OUTPUT:**

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 237 entries, 0 to 236

Data columns (total 4 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

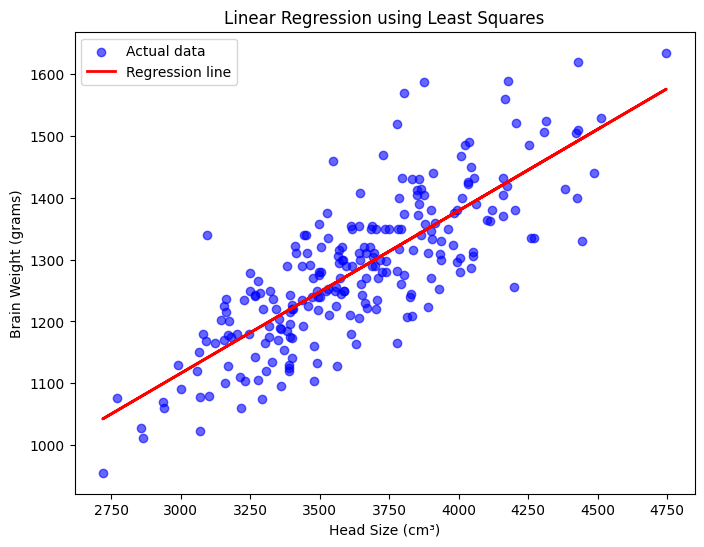
0 Gender 237 non-null int64

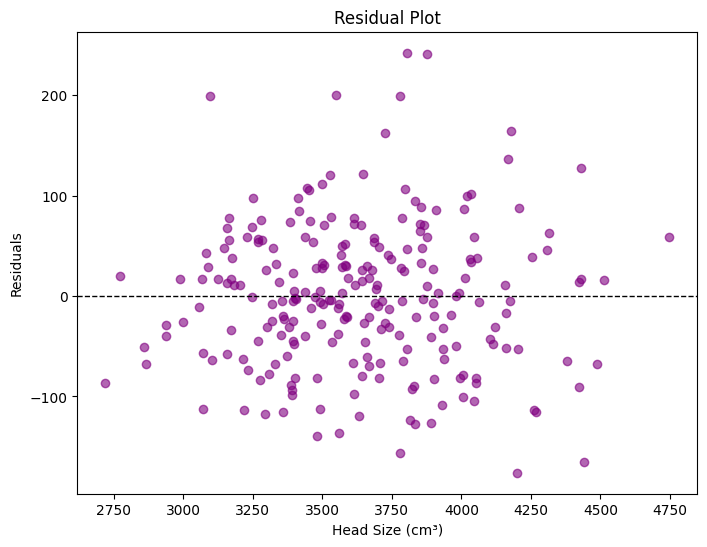
1 Age Range 237 non-null int64

2 Head Size(cm^3) 237 non-null int64

3 Brain Weight(grams) 237 non-null int64

dtypes: int64(4)





Intercept: 325.57

Slope: 0.26

R-squared Value: 0.6393

**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.

|  |  |
| --- | --- |
| **EXP NO:** 3 | **A PYTHON PROGRAM TO IMPLEMENT LOGISTIC MODEL** |
| **DATE:** 07**/2/25** |

**AIM:**

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

**ALGORITHM:**

**Step 1:** Import required libraries (numpy, matplotlib, pandas, sklearn).  
**Step 2:** Load the dataset (suv\_data.csv) into a pandas DataFrame.  
**Step 3:** Extract Age and Estimated Salary as X (features) and Purchased as y (target variable).  
**Step 4:** Split the data into training (80%) and testing (20%) sets using train\_test\_split().  
**Step 5:** Apply feature scaling (StandardScaler) to normalize X\_train and X\_test for better performance.  
**Step 6:** Train the Logistic Regression model using LogisticRegression().fit(X\_train, y\_train).  
**Step 7:** Make predictions (y\_pred) on X\_test using model.predict().  
**Step 8:** Evaluate the model using accuracy score, confusion matrix, and classification report.  
**Step 9:** Plot actual data using a scatter plot (Age vs. Estimated Salary, colored by y\_test).  
**Step 10:** Plot predicted data using a scatter plot (Age vs. Estimated Salary, colored by y\_pred).

**SOURCE CODE:**

import numpy as npimport matplotlib.pyplot as pltimport pandas as pdfrom sklearn.model\_selection import train\_test\_splitfrom sklearn.preprocessing import StandardScalerfrom sklearn.linear\_model import LogisticRegressionfrom sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report# Step 2: Read the datasetfile\_path = "/content/suv\_data.csv"data = pd.read\_csv(file\_path)# Step 3: Prepare the dataX = data[['Age', 'EstimatedSalary']].values # Independent variablesy = data['Purchased'].values # Dependent variable# Step 4: Split the dataset into training and testing setsX\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=0)# Step 5: Feature scalingscaler = StandardScaler()X\_train = scaler.fit\_transform(X\_train)X\_test = scaler.transform(X\_test)# Step 6: Train the logistic regression modelmodel = LogisticRegression()model.fit(X\_train, y\_train)# Step 7: Make predictionsy\_pred = model.predict(X\_test)# Step 8: Evaluate the modelaccuracy = accuracy\_score(y\_test, y\_pred)conf\_matrix = confusion\_matrix(y\_test, y\_pred)report = classification\_report(y\_test, y\_pred)print(f"Accuracy: {accuracy:.4f}")print("Confusion Matrix:")print(conf\_matrix)print("Classification Report:")print(report)# Step 9: Simple plots# Scatter plot of actual dataplt.figure(figsize=(8, 6))plt.scatter(X\_test[:, 0], X\_test[:, 1], c=y\_test, cmap='coolwarm', edgecolors='k')plt.xlabel('Age')plt.ylabel('Estimated Salary')plt.title('Actual Data Distribution')plt.show()# Scatter plot of predictionsplt.figure(figsize=(8, 6))plt.scatter(X\_test[:, 0], X\_test[:, 1], c=y\_pred, cmap='coolwarm', edgecolors='k')plt.xlabel('Age')plt.ylabel('Estimated Salary')plt.title('Predicted Data Distribution')plt.show()

**OUTPUT:**

Accuracy: 0.9250

Confusion Matrix:

[[57 1]

[ 5 17]]

Classification Report:

precision recall f1-score support

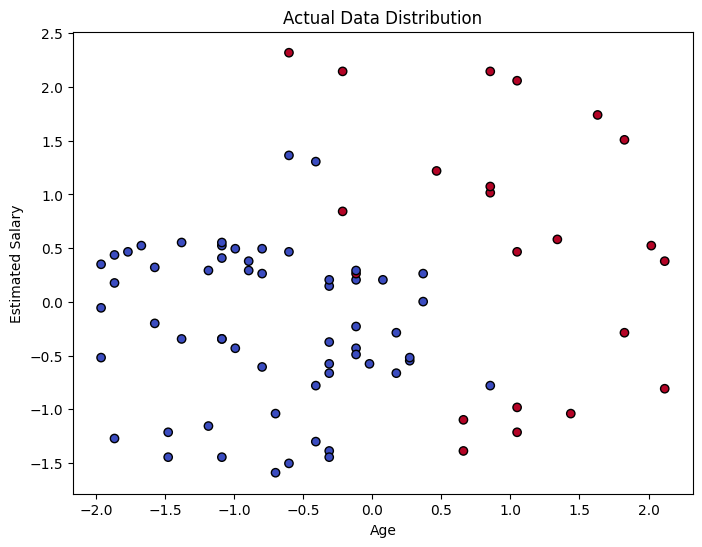
0 0.92 0.98 0.95 58

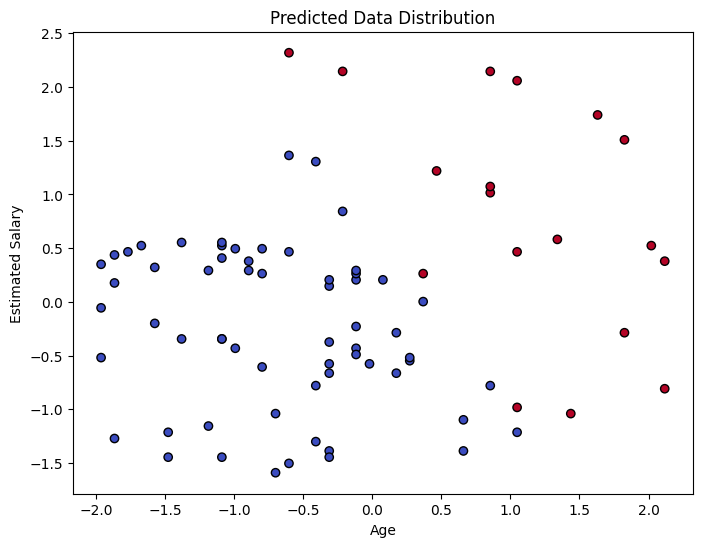
1 0.94 0.77 0.85 22

accuracy 0.93 80

macro avg 0.93 0.88 0.90 80

weighted avg 0.93 0.93 0.92 80

****

****

**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.

|  |  |
| --- | --- |
| **EXP NO:** 4 | **A PYTHON PROGRAM TO IMPLEMENT**  **SINGLE LAYER PERCEPTRON** |
| **DATE:** 1**4/2/25** |

**AIM:**

To implement python program for the single layer perceptron.

**ALGORITHM:**

**Step 1:** Initialize the input data (X) and corresponding labels (y).  
**Step 2:** Initialize weights and bias randomly.  
**Step 3:** Define an activation function (e.g., step function).  
**Step 4:** Set the learning rate (e.g., 0.1).  
**Step 5:** Compute the weighted sum of inputs (X) and weights (W).  
**Step 6:** Apply the activation function to get the output.  
**Step 7:** Calculate the error (difference between expected and predicted output).  
**Step 8:** Update weights and bias using the Perceptron Learning Rule.  
**Step 9:** Repeat steps 5–8 for multiple epochs to train the model.  
**Step 10:** Test the perceptron on new inputs and print predictions.

**SOURCE CODE:**

import numpy as np# Step 1: Initialize input features (X) and target labels (y)X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]]) # Inputsy = np.array([0, 0, 0, 1]) # AND logic gate output# Step 2: Initialize weights and biasweights = np.random.rand(2)bias = np.random.rand(1)learning\_rate = 0.1# Step 3: Define activation function (step function)def step\_function(x): return 1 if x >= 0 else 0# Step 4: Train the perceptron using the Perceptron Learning Algorithmepochs = 10for epoch in range(epochs): for i in range(len(X)): # Step 5: Compute weighted sum weighted\_sum = np.dot(X[i], weights) + bias # Step 6: Apply activation function y\_pred = step\_function(weighted\_sum) # Step 7: Compute error error = y[i] - y\_pred # Step 8: Update weights and bias weights += learning\_rate \* error \* X[i] bias += learning\_rate \* error# Step 9: Make predictionsfor i in range(len(X)): output = step\_function(np.dot(X[i], weights) + bias) print(f"Input: {X[i]}, Predicted Output: {output}")# Step 10: Final weights and biasprint("Final Weights:", weights)print("Final Bias:", bias)

**OUTPUT:**

Input: [0 0], Predicted Output: 0

Input: [0 1], Predicted Output: 0

Input: [1 0], Predicted Output: 0

Input: [1 1], Predicted Output: 1

Final Weights: [0.23942754 0.09998966]

Final Bias: [-0.33008925]

**RESULT:**

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

|  |  |
| --- | --- |
| **EXP NO:** 5 | **A PYTHON PROGRAM TO IMPLEMENT MULTI LAYER PERCEPTRON WITH BACK PROPOGATION** |
| **DATE:** **21/2/25** |

**AIM:**

To implement multilayer perceptron with back propagation using python.

**ALGORITHM:**

**Step 1:** Load the dataset from file (CSV or other formats).  
**Step 2:** Preprocess the dataset (Handle missing values if any).  
**Step 3:** Split the dataset into training and testing sets.  
**Step 4:** Normalize the features using StandardScaler().  
**Step 5:** Define and train the MLP model with one hidden layer.  
**Step 6:** Make predictions on the test set.  
**Step 7:** Evaluate the model using accuracy and confusion matrix.  
**Step 8:** Test the model with a new sample.  
**Step 9:** Retrieve final weights and biases of the model.  
**Step 10:** Visualize the classification results.

**SOURCE CODE:**

import numpy as npimport pandas as pdimport matplotlib.pyplot as pltfrom sklearn.model\_selection import train\_test\_splitfrom sklearn.preprocessing import StandardScalerfrom sklearn.neural\_network import MLPClassifierfrom sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report# Step 1: Load the dataset from filefile\_path = "/content/BankNote\_Authentication.csv" # Replace with your file pathdata = pd.read\_csv(file\_path)# Step 2: Preprocess the dataset (Check for missing values)print(data.info())print(data.describe())# Step 3: Prepare the data (Assuming last column is 'Class' and rest are features)X = data.iloc[:, :-1].values # Features (all columns except last)y = data.iloc[:, -1].values # Target (last column)# Step 4: Split dataset into training (80%) and testing (20%)X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)# Step 5: Normalize the datasetscaler = StandardScaler()X\_train = scaler.fit\_transform(X\_train)X\_test = scaler.transform(X\_test)# Step 6: Define the MLP model (1 hidden layer with 10 neurons)mlp = MLPClassifier(hidden\_layer\_sizes=(10,), activation='relu', solver='adam', max\_iter=1000, random\_state=42)# Step 7: Train the modelmlp.fit(X\_train, y\_train)# Step 8: Make predictionsy\_pred = mlp.predict(X\_test)# Step 9: Evaluate the modelaccuracy = accuracy\_score(y\_test, y\_pred)conf\_matrix = confusion\_matrix(y\_test, y\_pred)report = classification\_report(y\_test, y\_pred)print(f"Model Accuracy: {accuracy:.2%}")print("Confusion Matrix:")print(conf\_matrix)print("Classification Report:")print(report)# Step 10: Test the model with a new samplenew\_sample = [[2.5, -1.2, 3.1, -0.8]] # Replace with actual feature valuesnew\_sample\_scaled = scaler.transform(new\_sample)prediction = mlp.predict(new\_sample\_scaled)print(f"Predicted Class: {'Forged' if prediction[0] == 1 else 'Genuine'}")

**OUTPUT:**

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 1372 entries, 0 to 1371

Data columns (total 5 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 variance 1372 non-null float64

1 skewness 1372 non-null float64

2 curtosis 1372 non-null float64

3 entropy 1372 non-null float64

4 class 1372 non-null int64

dtypes: float64(4), int64(1)

memory usage: 53.7 KB

None

variance skewness curtosis entropy class

count 1372.000000 1372.000000 1372.000000 1372.000000 1372.000000

mean 0.433735 1.922353 1.397627 -1.191657 0.444606

std 2.842763 5.869047 4.310030 2.101013 0.497103

min -7.042100 -13.773100 -5.286100 -8.548200 0.000000

25% -1.773000 -1.708200 -1.574975 -2.413450 0.000000

50% 0.496180 2.319650 0.616630 -0.586650 0.000000

75% 2.821475 6.814625 3.179250 0.394810 1.000000

max 6.824800 12.951600 17.927400 2.449500 1.000000

Model Accuracy: 99.64%

Confusion Matrix:

[[147 1]

[ 0 127]]

Classification Report:

precision recall f1-score support

0 1.00 0.99 1.00 148

1 0.99 1.00 1.00 127

accuracy 1.00 275

macro avg 1.00 1.00 1.00 275

weighted avg 1.00 1.00 1.00 275

Predicted Class: Genuine

**RESULT:**

The MLP with backpropagation was successfully implemented on banknotes.csv, and results were analyzed using various activation functions (relu, logistic, tanh, identity) with training-testing splits of 0.2 and 0.3.

|  |  |
| --- | --- |
| **EXP NO:** 6 | **A PYTHON PROGRAM TO IMPLEMENT FACE RECOGNITION USING SVM CLASSIFIER MODEL** |
| **DATE:** **28/2/25** |

**AIM:**

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

**ALGORITHM:**

**Step 1:** Load the Labeled Faces in the Wild (LFW) dataset.  
**Step 2:** Extract face images (grayscale) and corresponding labels (person names).  
**Step 3:** Flatten 2D face images into 1D feature vectors for processing.  
**Step 4:** Normalize the feature vectors using StandardScaler to improve model performance.  
**Step 5:** Split the dataset into training (80%) and testing (20%) sets.  
**Step 6:** Apply PCA (Principal Component Analysis) to reduce dimensionality to 150 components.  
**Step 7:** Train an SVM (Support Vector Machine) classifier with a linear kernel on the PCA-transformed data.  
**Step 8:** Predict labels for the test set using the trained SVM model.  
**Step 9:** Evaluate model performance using accuracy score and confusion matrix.  
**Step 10:** Display sample predictions with actual vs. predicted labels using matplotlib.

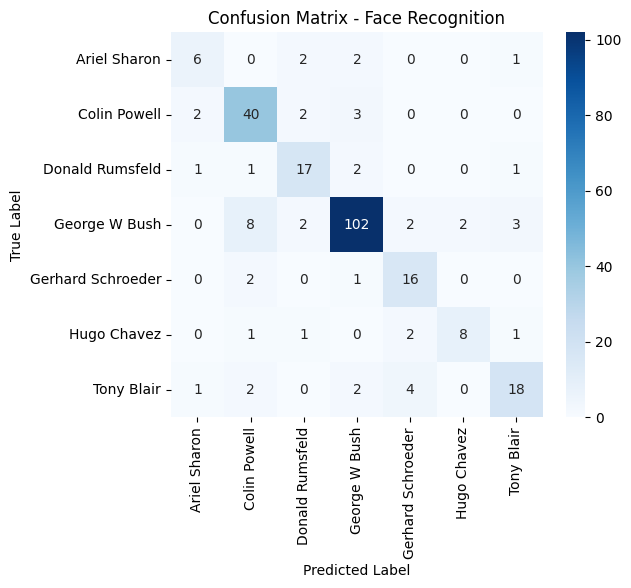
**SOURCE CODE:**

import numpy as npimport matplotlib.pyplot as pltimport seaborn as snsfrom sklearn.datasets import fetch\_lfw\_peoplefrom sklearn.model\_selection import train\_test\_splitfrom sklearn.svm import SVCfrom sklearn.decomposition import PCAfrom sklearn.preprocessing import StandardScalerfrom sklearn.metrics import accuracy\_score, confusion\_matrix# Load the Labeled Faces in the Wild (LFW) datasetlfw\_people = fetch\_lfw\_people(min\_faces\_per\_person=70, resize=0.4)X = lfw\_people.images # Face images (Gray-scale)y = lfw\_people.target # Person labelstarget\_names = lfw\_people.target\_names # Names of people# Flatten images for SVM input (Convert 2D images to 1D feature vectors)n\_samples, h, w = X.shapeX = X.reshape(n\_samples, h \* w)# Normalize datascaler = StandardScaler()X = scaler.fit\_transform(X)# Split data (80% training, 20% testing)X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)# Apply PCA (Principal Component Analysis) for dimensionality reductionn\_components = 150 # Reduce features to 150 dimensionspca = PCA(n\_components=n\_components, whiten=True)X\_train\_pca = pca.fit\_transform(X\_train)X\_test\_pca = pca.transform(X\_test)# Train SVM classifiersvm\_classifier = SVC(kernel="linear", class\_weight="balanced", probability=True)svm\_classifier.fit(X\_train\_pca, y\_train)# Test the modely\_pred = svm\_classifier.predict(X\_test\_pca)# Calculate accuracyaccuracy = accuracy\_score(y\_test, y\_pred)print(f"Face Recognition Model Accuracy: {accuracy \* 100:.2f}%")# Display Confusion Matrixconf\_matrix = confusion\_matrix(y\_test, y\_pred)plt.figure(figsize=(6, 5))sns.heatmap(conf\_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=target\_names, yticklabels=target\_names)plt.xlabel("Predicted Label")plt.ylabel("True Label")plt.title("Confusion Matrix - Face Recognition")plt.show()# Test with a sample imagesample\_idx = 5 # Choose any index from test setplt.imshow(lfw\_people.images[sample\_idx], cmap="gray")

plt.title(f"Actual: {target\_names[y\_test[sample\_idx]]} \nPredicted: {target\_names[y\_pred[sample\_idx]]}")plt.axis("off")plt.show()

**OUTPUT:**

Face Recognition Model Accuracy: 80.23%

****

****

**RESULT:**

Thus the python program to implement face recognition using SVM classifier model has been executed successfully and the classified output has been analyzed for the given dataset(fetch\_lfw\_people).

|  |  |
| --- | --- |
| **EXP NO:** 7 | **A PYTHON PROGRAM TO IMPLEMENT DECISION TREE** |
| **DATE:** 07**/3/25** |

**AIM:**

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

**ALGORITHM:**

**Step 1:** Import necessary libraries (numpy, matplotlib, sklearn).  
**Step 2:** Load the Iris dataset using load\_iris() function.  
**Step 3:** Extract features (X) and labels (y) from the dataset.  
**Step 4:** Split the dataset into training (80%) and testing (20%) sets using train\_test\_split().  
**Step 5:** Initialize the Decision Tree Classifier with a gini criterion and a maximum depth of 3.  
**Step 6:** Train the Decision Tree model on the training dataset using clf.fit(X\_train, y\_train).  
**Step 7:** Predict the class labels for the test dataset using clf.predict(X\_test).  
**Step 8:** Evaluate the model's accuracy using accuracy\_score().  
**Step 9:** Print the model's accuracy as a percentage (accuracy \* 100).  
**Step 10:** Visualize the trained Decision Tree using plot\_tree().

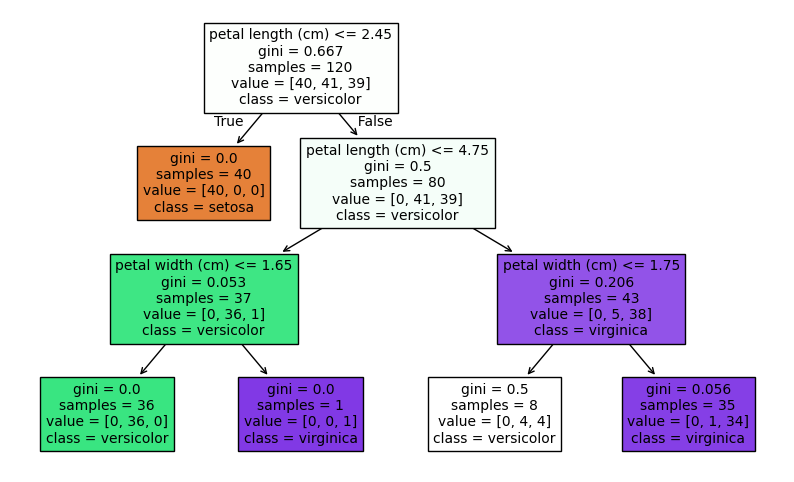
**SOURCE CODE:**

import numpy as npimport matplotlib.pyplot as pltfrom sklearn.datasets import load\_irisfrom sklearn.tree import DecisionTreeClassifier, plot\_treefrom sklearn.model\_selection import train\_test\_splitfrom sklearn.metrics import accuracy\_score# Load datasetiris = load\_iris()X, y = iris.data, iris.target # Features & Labels# Split dataset (80% training, 20% testing)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)# Create Decision Tree modelclf = DecisionTreeClassifier(criterion="gini", max\_depth=3, random\_state=42)# Train the modelclf.fit(X\_train, y\_train)# Predict on test datay\_pred = clf.predict(X\_test)# Evaluate model accuracyaccuracy = accuracy\_score(y\_test, y\_pred)print(f"Model Accuracy: {accuracy \* 100:.2f}%")# Visualize the Decision Treeplt.figure(figsize=(10, 6))plot\_tree(clf, feature\_names=iris.feature\_names, class\_names=iris.target\_names, filled=True)plt.show()

**OUTPUT:**

Model Accuracy: 100.00%



**RESULT:**

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

|  |  |
| --- | --- |
| **EXP NO:** 8 | **A PYTHON PROGRAM TO IMPLEMENT BOOSTING** |
| **DATE:** **28/3/25** |

**AIM:**

To implement a python program using the ada boosting model and gradient boosting model.

**(1) ADA BOOSTING**

**ALGORITHM:**

**Step 1:** Import necessary libraries (numpy, matplotlib, sklearn).  
**Step 2:** Load the Iris dataset and extract features (X) and labels (y).  
**Step 3:** Split the dataset into training (80%) and testing (20%) sets using train\_test\_split().  
**Step 4:** Initialize the AdaBoost Classifier with a Decision Tree (max\_depth=1) as the base estimator.  
**Step 5:** Train the AdaBoost model on the training dataset and make predictions on the test dataset.  
**Step 6:** Evaluate the model's accuracy and plot feature importance using a bar chart.

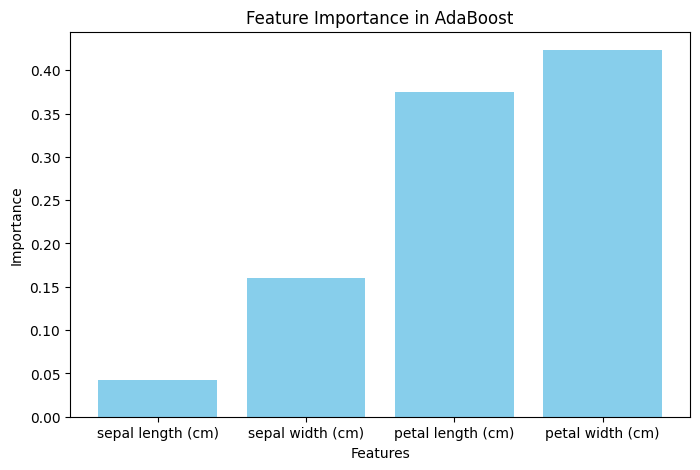
**SOURCE CODE:**

import numpy as npimport matplotlib.pyplot as pltfrom sklearn.datasets import load\_irisfrom sklearn.model\_selection import train\_test\_splitfrom sklearn.ensemble import AdaBoostClassifierfrom sklearn.tree import DecisionTreeClassifierfrom sklearn.metrics import accuracy\_score# Load datasetiris = load\_iris()X, y = iris.data, iris.target# Split datasetX\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)# Create AdaBoost model with Decision Tree as base estimator

boosting\_model = AdaBoostClassifier( estimator=DecisionTreeClassifier(max\_depth=1), n\_estimators=50, learning\_rate=1.0, random\_state=42)# Train the modelboosting\_model.fit(X\_train, y\_train)# Predict on test datay\_pred = boosting\_model.predict(X\_test)# Evaluate model accuracyaccuracy = accuracy\_score(y\_test, y\_pred)print(f"Model Accuracy: {accuracy \*100 :.2f}%")# Plot feature importanceplt.figure(figsize=(8, 5))plt.bar(iris.feature\_names, boosting\_model.feature\_importances\_, color='skyblue')plt.xlabel("Features")plt.ylabel("Importance")plt.title("Feature Importance in AdaBoost")plt.show()

**OUTPUT:**

Model Accuracy: 93.33%



**II) GRADIENT BOOSTING**

**ALGORITHM:**

**Step 1:** Import required libraries (sklearn, numpy, matplotlib).  
**Step 2:** Load the Iris dataset and extract features (X) and labels (y).  
**Step 3:** Split the dataset into training (80%) and testing (20%) sets using train\_test\_split().  
**Step 4:** Initialize the Gradient Boosting Classifier with 100 estimators, a learning rate of 0.1, and a max depth of 3.  
**Step 5:** Train the Gradient Boosting model on the training dataset and predict labels for the test dataset.  
**Step 6:** Evaluate the model's accuracy and plot the training loss curve to visualize model performance.

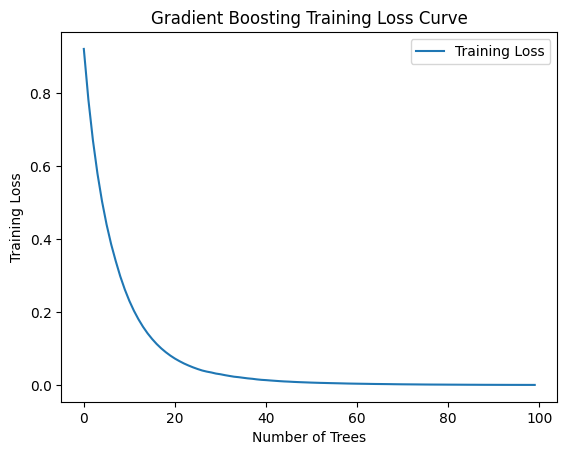
**SOURCE CODE:**

from sklearn.ensemble import GradientBoostingClassifierfrom sklearn.model\_selection import train\_test\_splitfrom sklearn.datasets import load\_irisfrom sklearn.metrics import accuracy\_score# Load datasetdata = load\_iris()X, y = data.data, data.target# Split into training and testing setsX\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)# Create Gradient Boosting modelgb\_clf = GradientBoostingClassifier(n\_estimators=100, learning\_rate=0.1, max\_depth=3, random\_state=42)# Train the modelgb\_clf.fit(X\_train, y\_train)# Predict on test datay\_pred = gb\_clf.predict(X\_test)# Calculate accuracyaccuracy = accuracy\_score(y\_test, y\_pred)print(f"Model Accuracy: {accuracy \* 100:.2f}%")# Plot the training loss curve

plt.plot(np.arange(len(gb\_clf.train\_score\_)), gb\_clf.train\_score\_, label="Training Loss")plt.xlabel("Number of Trees")plt.ylabel("Training Loss")plt.title("Gradient Boosting Training Loss Curve")plt.legend()plt.show()

**OUTPUT:**

Model Accuracy: 100.00%

****

**RESULT:**

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

|  |  |
| --- | --- |
| **EXP NO:** 9 | **A PYTHON PROGRAM TO IMPLEMENT KNN AND KMEANS MODEL** |
| **DATE:** **4/4/25** |

**AIM:**

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

**(I) KNN MODEL**

**ALGORITHM:**

**Step 1:** Import necessary libraries (numpy, matplotlib, sklearn).  
**Step 2:** Load the Breast Cancer dataset and extract features (X) and labels (y).  
**Step 3:** Split the dataset into training (80%) and testing (20%) sets using train\_test\_split().  
**Step 4:** Initialize the K-Nearest Neighbors (KNN) classifier with k=5 and train it using the training dataset.  
**Step 5:** Predict the labels for the test dataset and compute the model’s accuracy score.  
**Step 6:** Plot the accuracy vs. k-values to visualize model performance for different k.

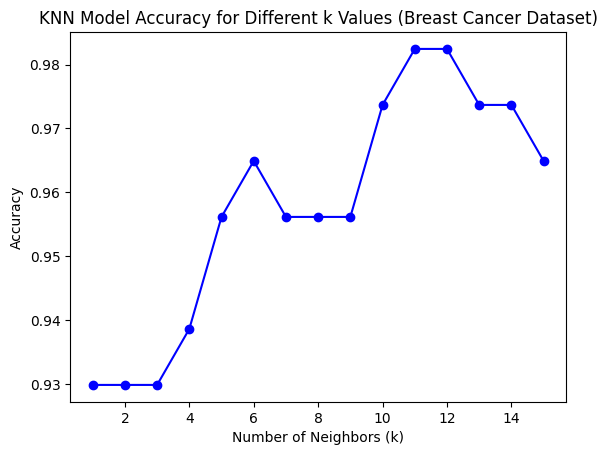
**SOURCE CODE:**

# Import necessary librariesimport numpy as npimport matplotlib.pyplot as pltfrom sklearn.model\_selection import train\_test\_splitfrom sklearn.neighbors import KNeighborsClassifierfrom sklearn.datasets import load\_breast\_cancerfrom sklearn.metrics import accuracy\_score# Load the Breast Cancer datasetcancer = load\_breast\_cancer()X, y = cancer.data, cancer.target # Features and labels# Split the data into training (80%) and testing (20%) setsX\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)# Create and train the KNN model with k=5knn = KNeighborsClassifier(n\_neighbors=5)

knn.fit(X\_train, y\_train)# Predict on the test sety\_pred = knn.predict(X\_test)# Calculate accuracyaccuracy = accuracy\_score(y\_test, y\_pred)print(f"Model Accuracy: {accuracy:.2%}") # Accuracy in percentage format# Plot accuracy for different values of kk\_values = range(1, 16)accuracy\_scores = []for k in k\_values: knn = KNeighborsClassifier(n\_neighbors=k) knn.fit(X\_train, y\_train) y\_pred = knn.predict(X\_test) accuracy\_scores.append(accuracy\_score(y\_test, y\_pred))plt.plot(k\_values, accuracy\_scores, marker='o', linestyle='-', color='b')plt.xlabel('Number of Neighbors (k)')plt.ylabel('Accuracy')plt.title('KNN Model Accuracy for Different k Values (Breast Cancer Dataset)')plt.show()

**OUTPUT:**

Model Accuracy: 95.61%

****

**(I) KMEANS MODEL**

**ALGORITHM:**

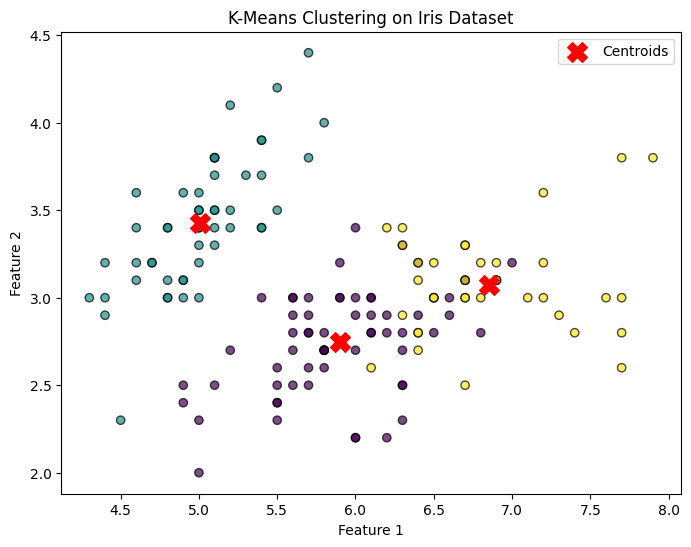
**Step 1:** Import necessary libraries (numpy, matplotlib, sklearn).  
**Step 2:** Load the Iris dataset and extract features (X).  
**Step 3:** Apply K-Means clustering with n\_clusters=3 and fit the model.  
**Step 4:** Predict cluster labels and compute the Silhouette Score to evaluate clustering performance.  
**Step 5:** Plot the clusters using the first two features and mark cluster centroids.  
**Step 6:** Display the clustering results and analyze the Silhouette Score for quality assessment.

**SOURCE CODE:**

import numpy as npimport matplotlib.pyplot as pltfrom sklearn import datasetsfrom sklearn.cluster import KMeansfrom sklearn.metrics import silhouette\_score# Load the Iris datasetiris = datasets.load\_iris()X = iris.data # Features (4D)y\_true = iris.target # True labels (for reference)# Apply K-Means Clusteringkmeans = KMeans(n\_clusters=3, random\_state=42, n\_init=10)y\_kmeans = kmeans.fit\_predict(X)# Calculate Silhouette Score (higher is better)sil\_score = silhouette\_score(X, y\_kmeans)print(f"Silhouette Score: {sil\_score:.4f}")# Plot clustersplt.figure(figsize=(8,6))plt.scatter(X[:, 0], X[:, 1], c=y\_kmeans, cmap='viridis', edgecolors='k', alpha=0.7)plt.scatter(kmeans.cluster\_centers\_[:, 0], kmeans.cluster\_centers\_[:, 1], s=200, c='red', marker='X', label="Centroids")plt.xlabel("Feature 1")plt.ylabel("Feature 2")plt.title("K-Means Clustering on Iris Dataset")plt.legend()plt.show()

**OUTPUT:**

Silhouette Score: 0.5528

****

**RESULT:**

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

|  |  |
| --- | --- |
| **EXP NO:** 10 | **PYTHON PROGRAM FOR SIMPLE LINEAR**  **REGRESSION** |
| **DATE:** 11/4/25 |

**AIM:**

To implement Dimensionality Reduction using PCA in a python program.

**ALGORITHM:**

**Step 1:** Import required libraries (numpy, matplotlib, sklearn).  
**Step 2:** Load the Iris dataset and extract features (X) and labels (y).  
**Step 3:** Apply PCA to reduce 4D features to 2D (n\_components=2).  
**Step 4:** Compute and print the explained variance ratio for both principal components.  
**Step 5:** Plot the transformed 2D data, color-coded by target class (y).  
**Step 6:** Display the scatter plot with labeled axes and a color bar for class identification.

**SOURCE CODE:**

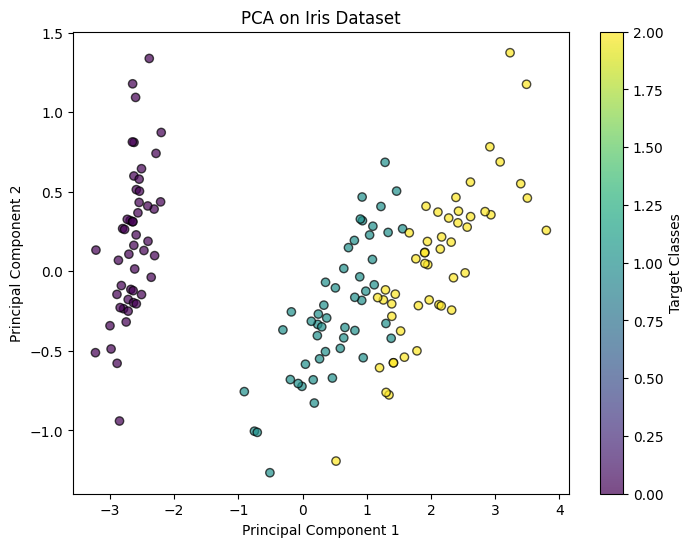
import numpy as npimport matplotlib.pyplot as pltfrom sklearn import datasetsfrom sklearn.decomposition import PCA# Load the Iris datasetiris = datasets.load\_iris()X = iris.data # Features (4D)y = iris.target # Labels (0,1,2)# Apply PCA to reduce from 4D to 2Dpca = PCA(n\_components=2) # Reduce to 2 dimensionsX\_pca = pca.fit\_transform(X)# Print explained variance ratioexplained\_variance = pca.explained\_variance\_ratio\_print(f"Explained Variance by Component 1: {explained\_variance[0]\*100:.2f}%")print(f"Explained Variance by Component 2: {explained\_variance[1]\*100:.2f}%")print(f"Total Variance Retained: {sum(explained\_variance)\*100:.2f}%")# Plot the reduced 2D dataplt.figure(figsize=(8,6))plt.scatter(X\_pca[:, 0], X\_pca[:, 1], c=y, cmap='viridis', edgecolors='k', alpha=0.7)plt.xlabel("Principal Component 1")plt.ylabel("Principal Component 2")plt.title("PCA on Iris Dataset")plt.colorbar(label="Target Classes")plt.show()

**OUTPUT:**

Explained Variance by Component 1: 92.46%

Explained Variance by Component 2: 5.31%

Total Variance Retained: 97.77%

****

**RESULT:**

Thus Dimensionality Reduction has been implemented using PCA in a python program

successfully and the results have been analyzed

|  |  |
| --- | --- |
| **EXP NO:** 11 | **DEVELOP A SIMPLE APPLICATION USING TENSORFLOW/KERAS** |
| **DATE:** 11/4/25 |

**AIM:**

To develop a simple application using tensorflow/keras.

**ALGORITHM:**

**Step 1:** Import necessary libraries: numpy, tensorflow.keras, and random.

**Step 2:** Set seeds for reproducibility using np.random.seed(42) and tf.random.set\_seed(42).

**Step 3:** Generate 1000 samples, create one-hot encoded inputs and normalized outputs.  
**Step 4:** Convert X and y lists to NumPy arrays.  
**Step 5:** Build a Sequential model with two hidden layers (ReLU) and one output layer (linear).

**Step 6:** Compile the model with Adam optimizer and mean squared error loss.  
**Step 7:** Train the model on generated data for 30 epochs with batch size 32.  
**Step 8:** Define a function to predict and display the model’s guess for a given number.  
**Step 9:** Test the function on numbers [5, 25, 50, 75, 95].

**SOURCE CODE:**

import numpy as np

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

import random

# Set seed for reproducibility

np.random.seed(42)

tf.random.set\_seed(42)

# Generate training data

X = []

y = []

for \_ in range(1000):

number = random.randint(1, 100)

one\_hot\_input = [0] \* 100

one\_hot\_input[number - 1] = 1

X.append(one\_hot\_input)

y.append([number / 100]) # Normalize output between 0 and 1

X = np.array(X)

y = np.array(y)

# Build the model

model = Sequential([

Dense(128, activation='relu', input\_shape=(100,)),

Dense(64, activation='relu'),

Dense(1, activation='linear') # Output a single number (normalized)

])

# Compile the model

model.compile(optimizer='adam', loss='mse')

# Train the model

model.fit(X, y, epochs=30, batch\_size=32, verbose=1)

# Test the model with a few numbers

def guess\_number(num):

input\_data = np.zeros((1, 100))

input\_data[0][num - 1] = 1

prediction = model.predict(input\_data)[0][0] \* 100

print(f"Actual Number: {num}")

print(f"Model's Guess: {round(prediction, 2)}")

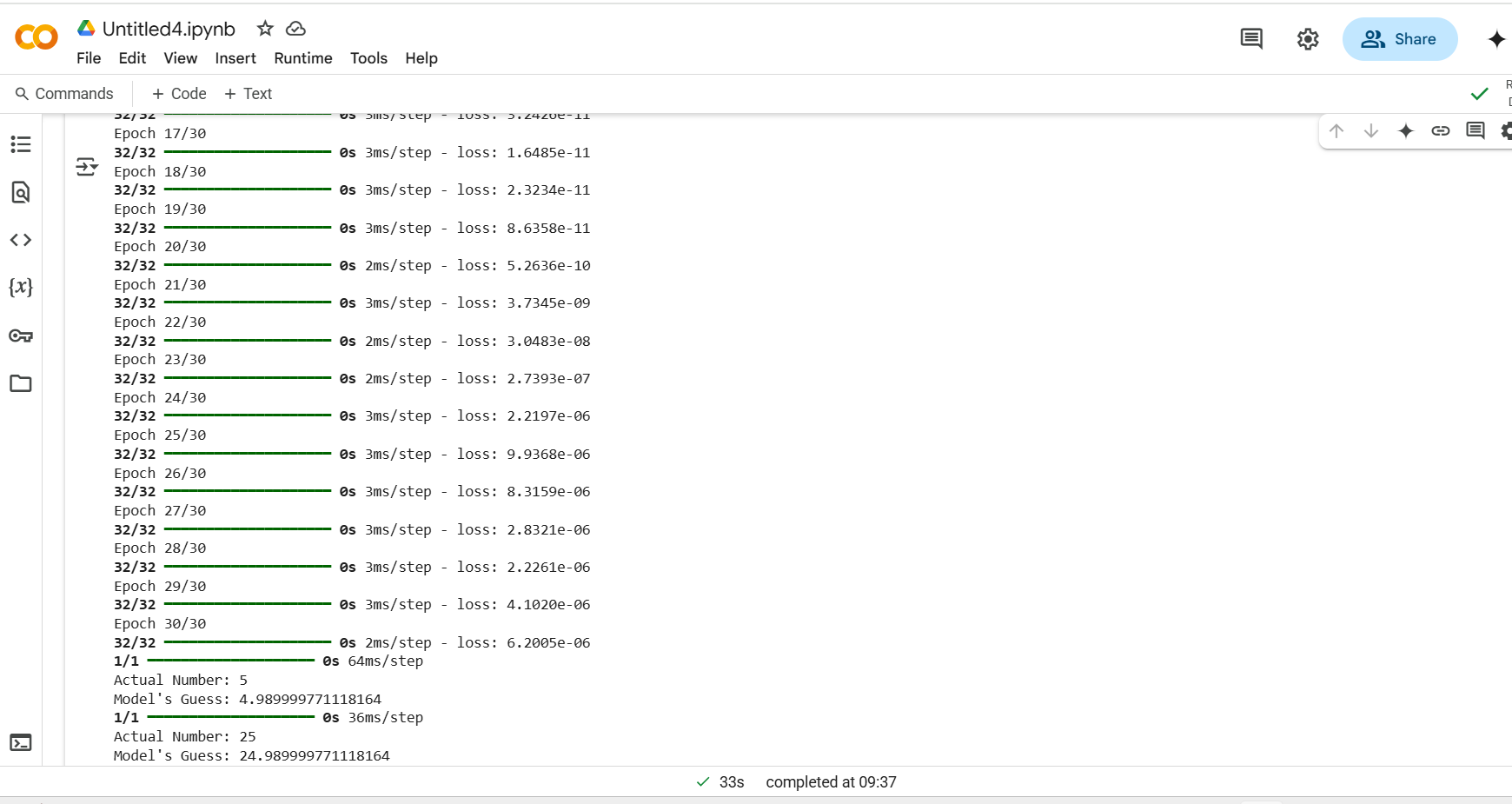
# Try out the model

for num in [5, 25, 50, 75, 95]:

guess\_number(num)

**OUTPUT:**

****

****

**RESULT:**

Thus a simple application using tensorflow/keras is developed.