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

BONAFIDE CERTIFICATE

NAME ......Harsavardhini R......

ACADEMIC YEAR ..2024 - 2025.... SEMESTER ....VI.............. BRANCH ....AI & DS........

UNIVERSITY REGISTER No. 2116221801016

Certified that this is the bonafide record of work done by the above student in the AI19442 Fundamentals of Machine Learning Laboratory during the year 2024 - 2025

Signature of Faculty - in - Charge

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

External Examiner

Internal Examiner

INDEX

Name: Harshwardhini R Branch: AI&DS Sec: Roll No: 221801016

Completed

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 (R^2 score)

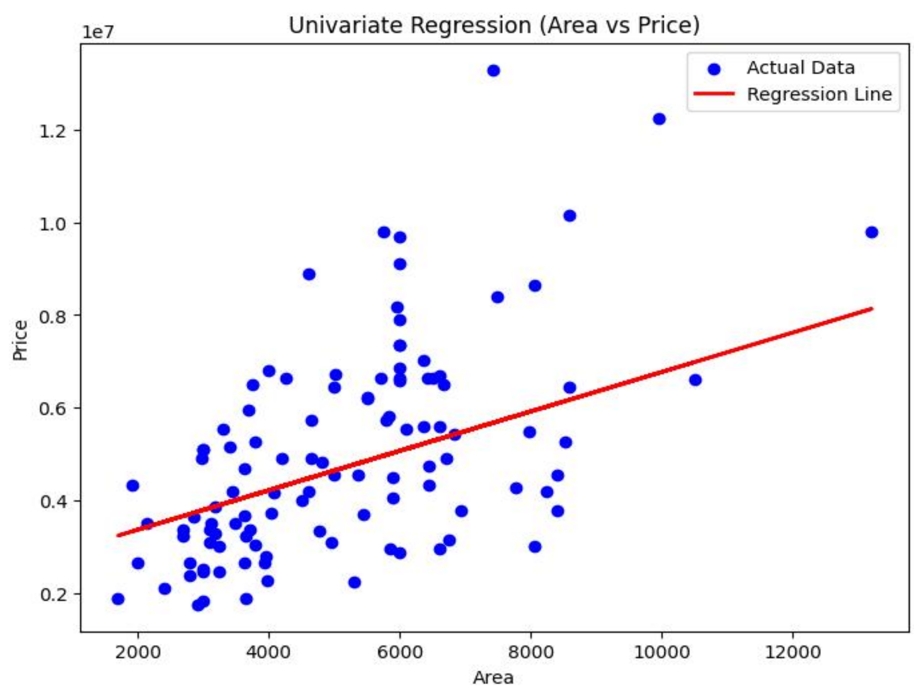
SOURCE CODE:

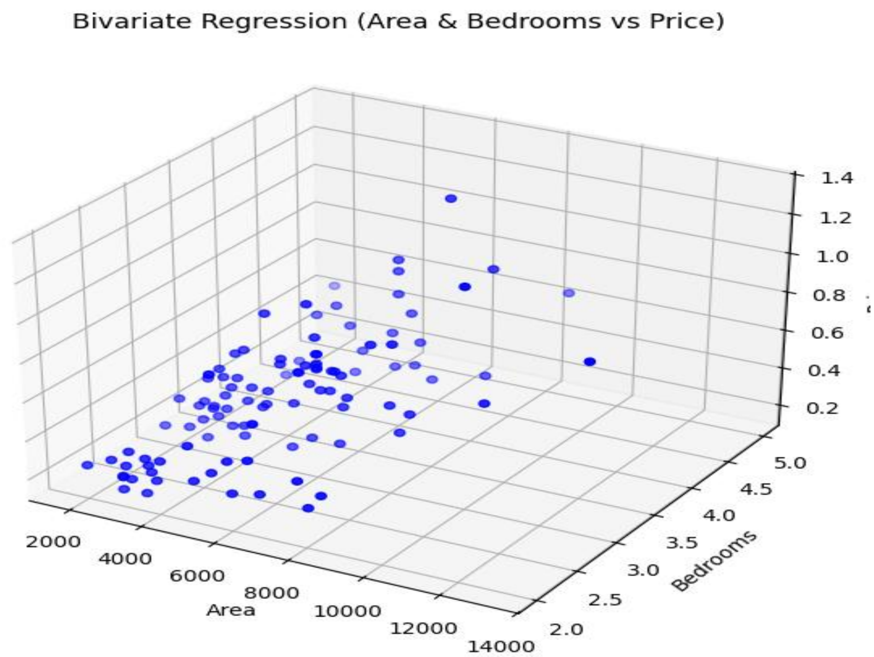
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import LabelEncoder
from mpl_toolkits.mplot3d import Axes3D
from sklearn.metrics import mean_squared_error, r2_score

# Step 1: Load dataset
file_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 Regression
plt.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 3D
fig = 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()
```

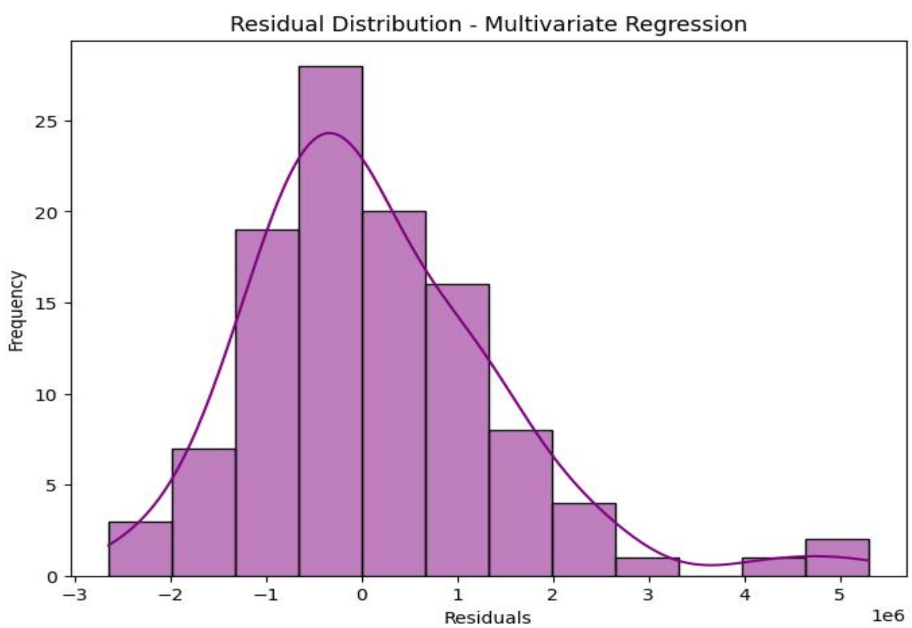
```
# Step 5: Multivariate Regression (Using all features)
X_multi = df.drop(columns=['price'])
X_train, X_test, y_train, y_test = train_test_split(X_multi, y, test_size=0.2, random_state=42)
model_multi = LinearRegression()
model_multi.fit(X_train, y_train)
y_pred_multi = model_multi.predict(X_test)

# Model Evaluation
mse = 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 Plot
residuals = y_test - y_pred_multi
plt.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 R^2 Score) to evaluate model performance.

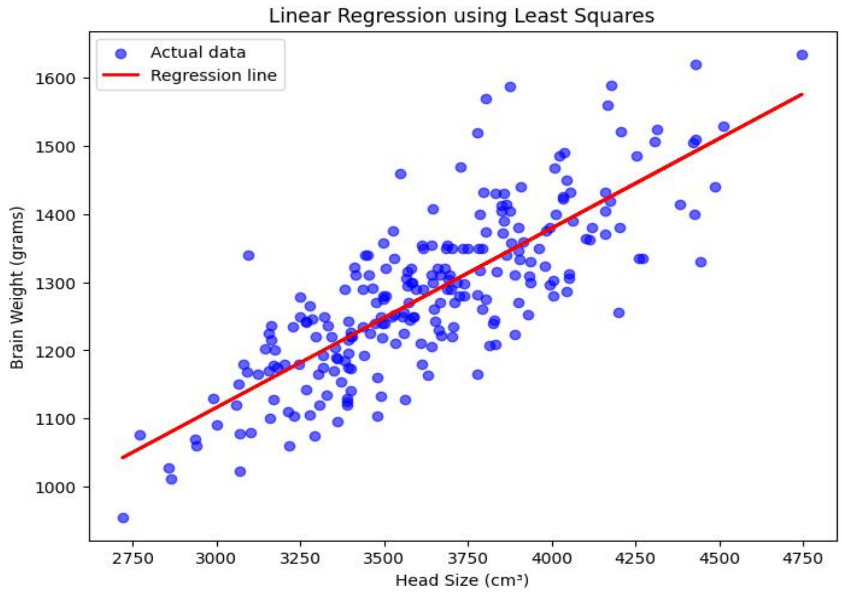
SOURCE CODE:

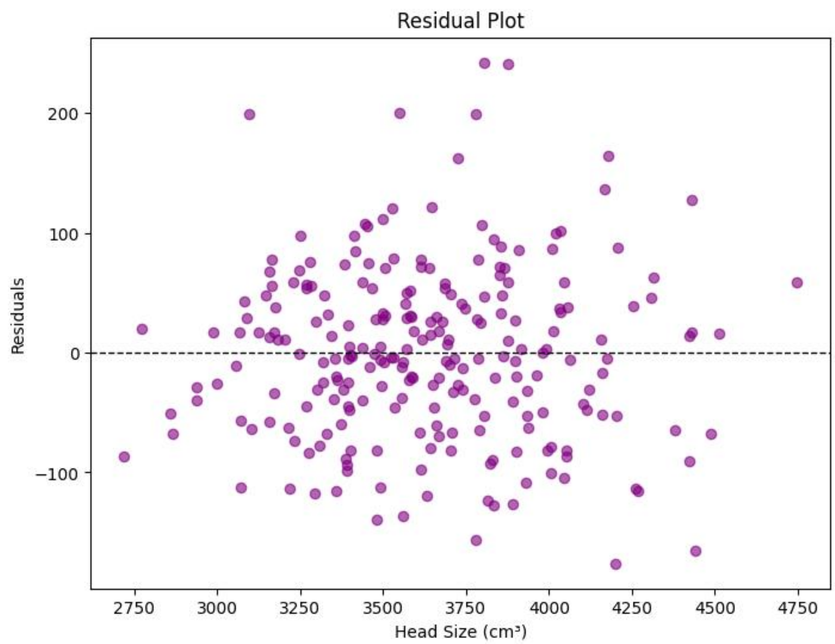
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Step 1: Import necessary libraries

# Step 2: Read the dataset
file_path = "/content/headbrain.csv"
data = pd.read_csv(file_path)

data.head()
data.info()
data.describe()

# Step 3: Prepare the data
X = data['Head Size(cm^3)'].values
y = data['Brain Weight(grams)'].values

# Step 4: Calculate the mean
mean_x, mean_y = np.mean(X), np.mean(y)

# Step 5: Calculate the coefficients
b1 = np.sum((X - mean_x) * (y - mean_y)) / np.sum((X - mean_x) ** 2)
b0 = mean_y - b1 * mean_x

# Step 6: Make predictions
y_pred = b0 + b1 * X

# Step 7: Plot the regression line
plt.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 residuals
residuals = y - y_pred
plt.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 value
TSS = np.sum((y - mean_y) ** 2)
RSS = np.sum((y - y_pred) ** 2)
R2 = 1 - (RSS / TSS)

# Step 10: Display the results
print(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`).

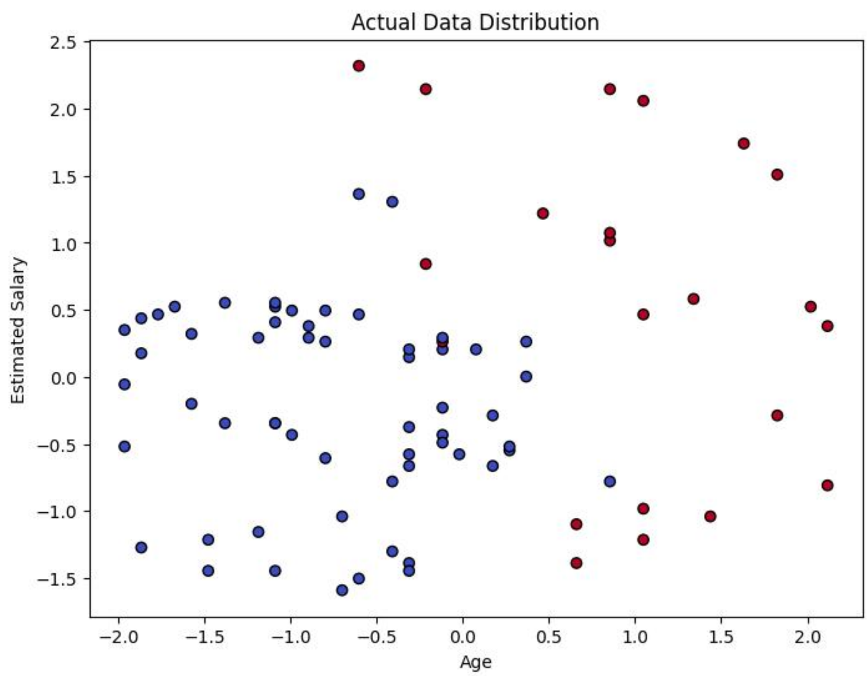
SOURCE CODE:

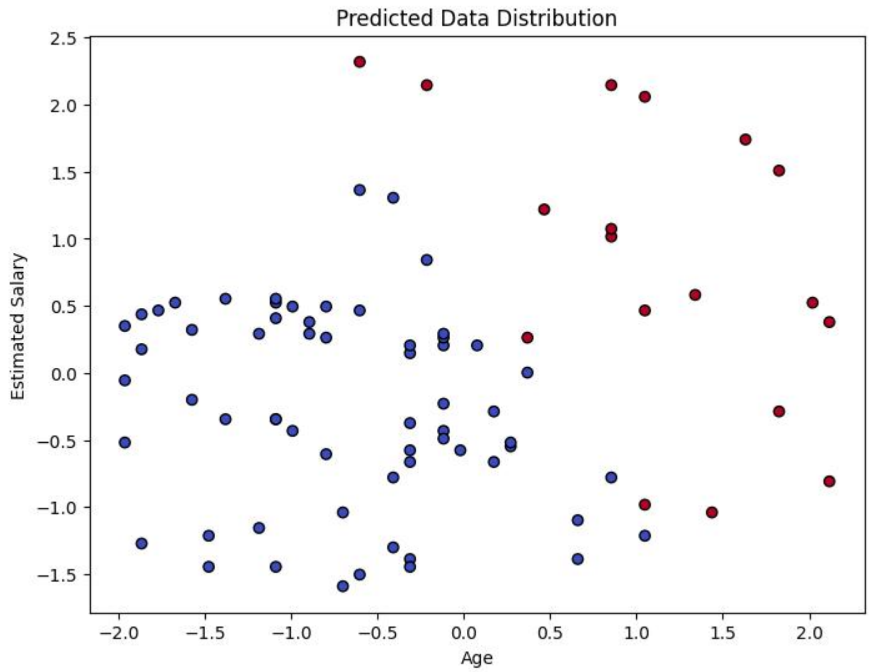
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Step 2: Read the dataset
file_path = "/content/suv_data.csv"
data = pd.read_csv(file_path)
```

```
# Step 3: Prepare the data
X = data[['Age', 'EstimatedSalary']].values  # Independent variables
y = data['Purchased'].values  # Dependent variable

# Step 4: Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

# Step 5: Feature scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Step 6: Train the logistic regression model
model = LogisticRegression()
model.fit(X_train, y_train)

# Step 7: Make predictions
y_pred = model.predict(X_test)

# Step 8: Evaluate the model
accuracy = 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 data
plt.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 predictions
plt.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: 14/2/25	

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]])  # Inputs
y = np.array([0, 0, 0, 1])  # AND logic gate output

# Step 2: Initialize weights and bias
weights = 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 Algorithm
epochs = 10
for 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 predictions
for 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 bias
print("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 np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Step 1: Load the dataset from file
file_path = "/content/BankNote_Authentication.csv"  # Replace with your file path
data = 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 dataset
scaler = 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 model
mlp.fit(X_train, y_train)

# Step 8: Make predictions
y_pred = mlp.predict(X_test)

# Step 9: Evaluate the model
accuracy = 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 sample
new_sample = [[2.5, -1.2, 3.1, -0.8]]  # Replace with actual feature values
new_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`.

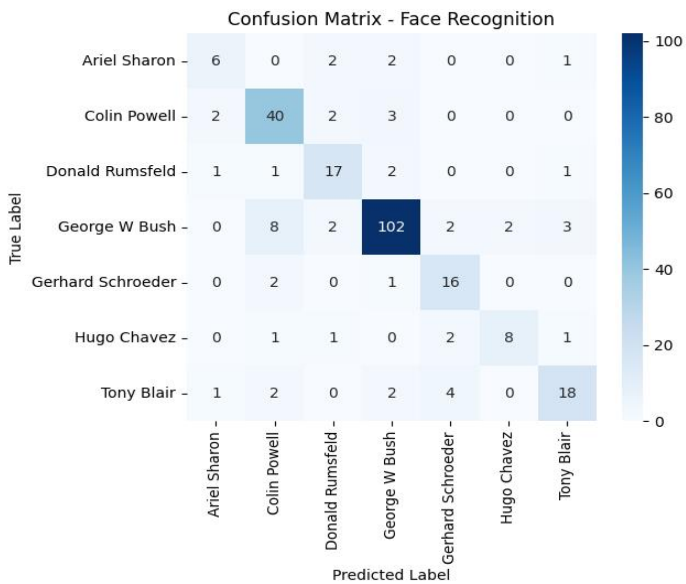
SOURCE CODE:

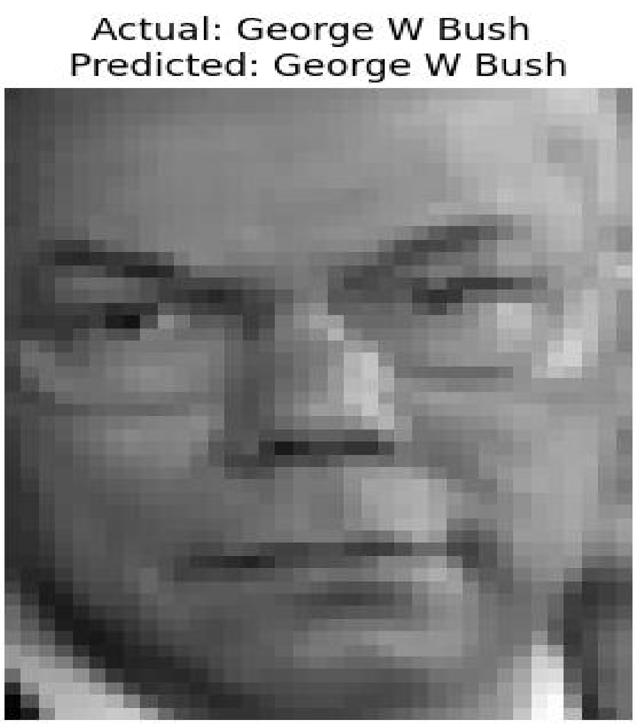
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import fetch_lfw_people
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, confusion_matrix

# Load the Labeled Faces in the Wild (LFW) dataset
```

```
lfw_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 labels
target_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.shape
X = X.reshape(n_samples, h * w)
# Normalize data
scaler = 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 reduction
n_components = 150  # Reduce features to 150 dimensions
pca = PCA(n_components=n_components, whiten=True)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)
# Train SVM classifier
svm_classifier = SVC(kernel="linear", class_weight="balanced", probability=True)
svm_classifier.fit(X_train_pca, y_train)
# Test the model
y_pred = svm_classifier.predict(X_test_pca)
# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Face Recognition Model Accuracy: {accuracy * 100:.2f}%")

# Display Confusion Matrix
conf_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 image
sample_idx = 5  # Choose any index from test set
plt.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()`.

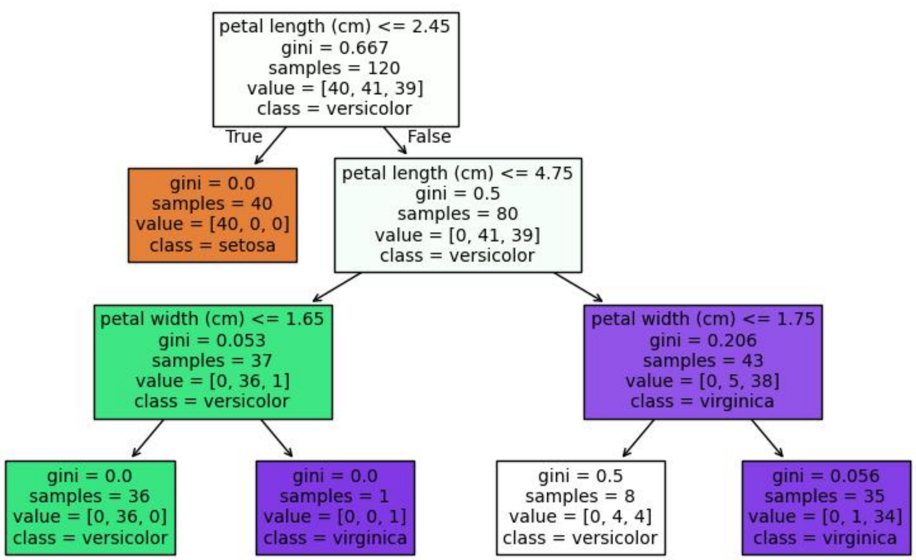
SOURCE CODE:

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

# Load dataset
iris = 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 model
clf = DecisionTreeClassifier(criterion="gini", max_depth=3, random_state=42)

# Train the model
clf.fit(X_train, y_train)

# Predict on test data
y_pred = clf.predict(X_test)

# Evaluate model accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy * 100:.2f}%")

# Visualize the Decision Tree
plt.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.

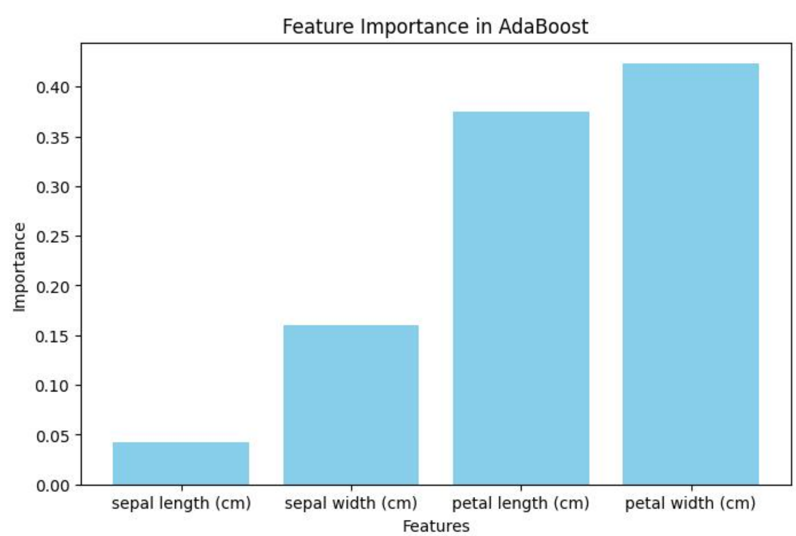
SOURCE CODE:

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

# Load dataset
iris = load_iris()
X, y = iris.data, iris.target

# Split dataset
X_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 model
boosting_model.fit(X_train, y_train)

# Predict on test data
y_pred = boosting_model.predict(X_test)

# Evaluate model accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy *100 :.2f}%")

# Plot feature importance
plt.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.

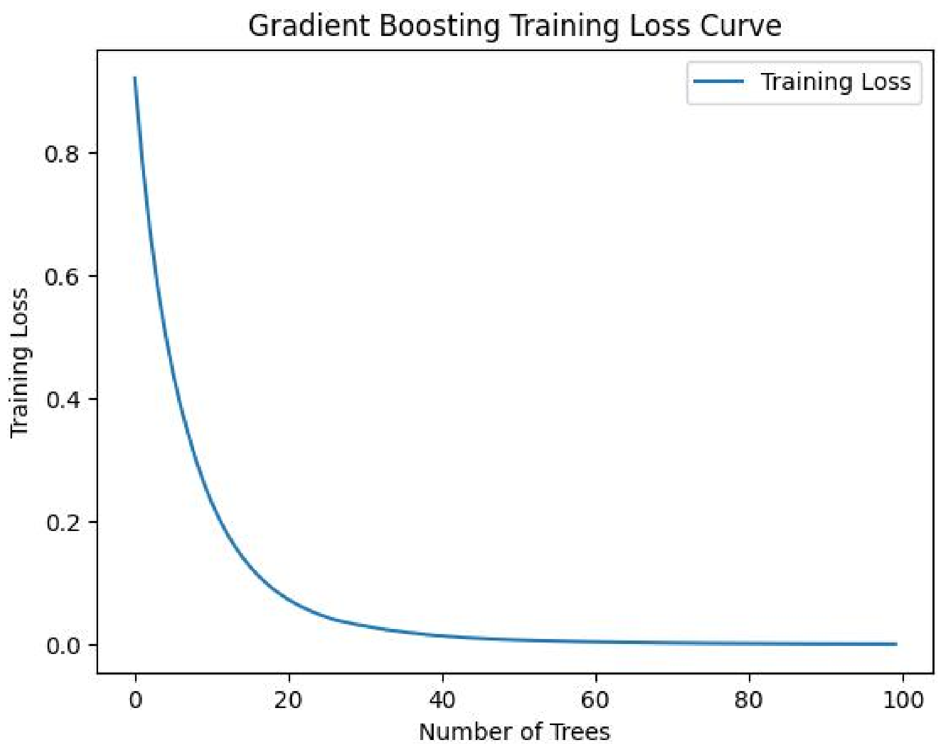
SOURCE CODE:

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
from sklearn.metrics import accuracy_score

# Load dataset
data = load_iris()
X, y = data.data, data.target

# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create Gradient Boosting model
gb_clf = GradientBoostingClassifier(n_estimators=100, learning_rate=0.1, max_depth=3,
random_state=42)

# Train the model
gb_clf.fit(X_train, y_train)

# Predict on test data
y_pred = gb_clf.predict(X_test)

# Calculate accuracy
accuracy = 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.

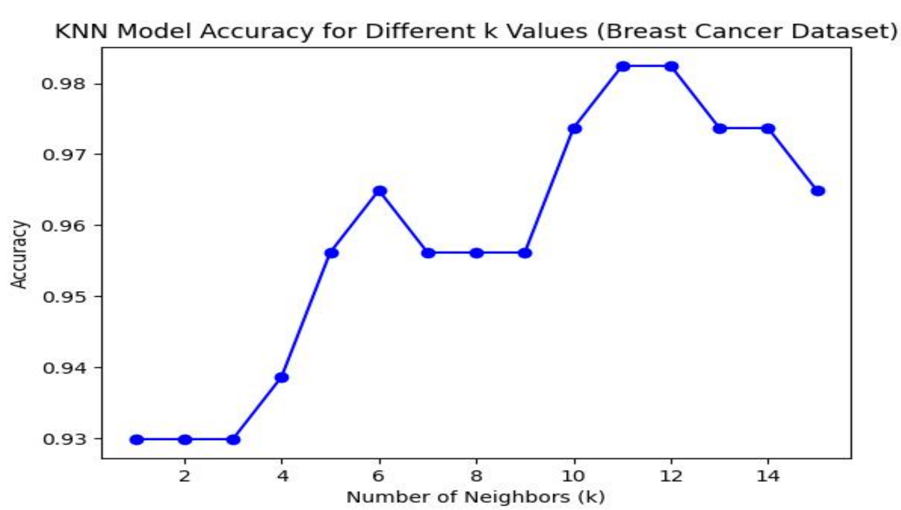
SOURCE CODE:

```
# Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.datasets import load_breast_cancer
from sklearn.metrics import accuracy_score

# Load the Breast Cancer dataset
cancer = load_breast_cancer()
X, y = cancer.data, cancer.target  # Features and labels

# Split the data into training (80%) and testing (20%) sets
X_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=5
knn = KNeighborsClassifier(n_neighbors=5)
```

```
knn.fit(X_train, y_train)

# Predict on the test set
y_pred = knn.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy:.2%}")  # Accuracy in percentage format

# Plot accuracy for different values of k
k_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.

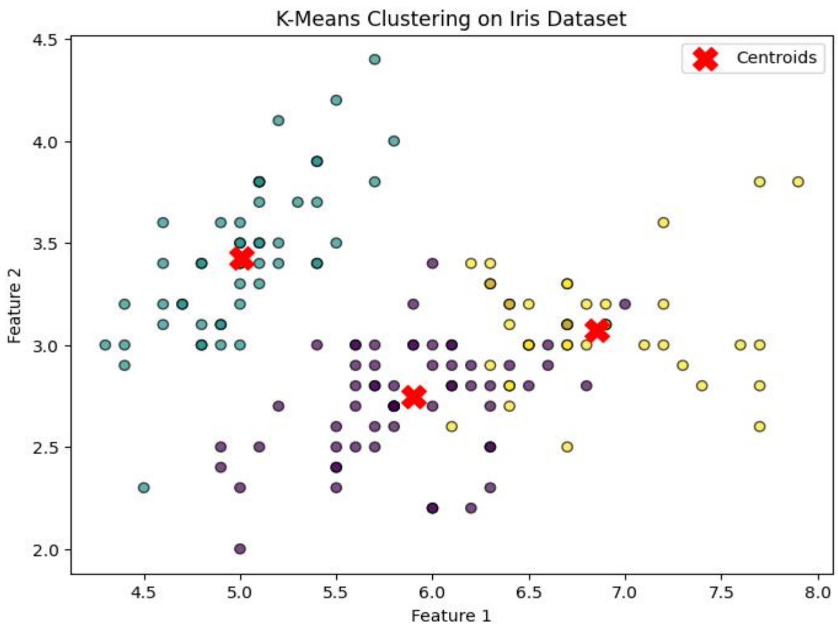
Step 6: Display the clustering results and analyze the Silhouette Score for quality assessment.

SOURCE CODE:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

# Load the Iris dataset
iris = datasets.load_iris()
X = iris.data  # Features (4D)
y_true = iris.target  # True labels (for reference)

# Apply K-Means Clustering
kmeans = 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 clusters
plt.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.

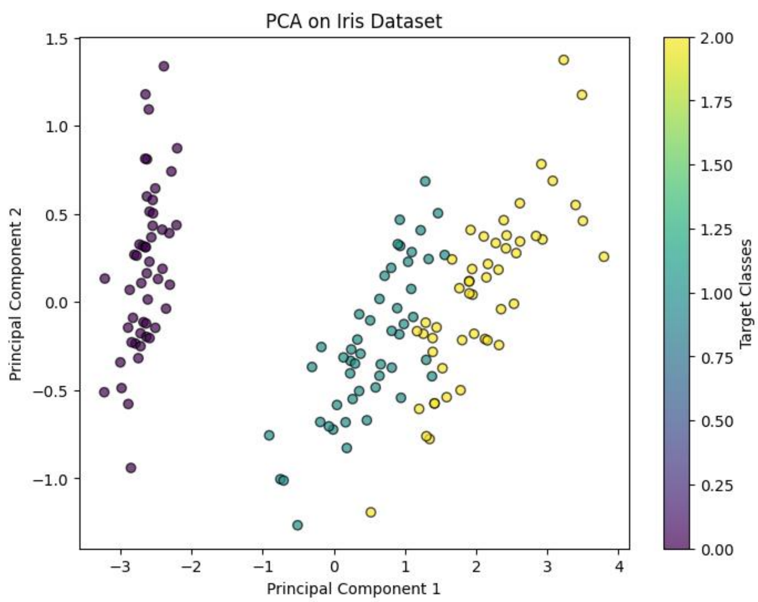
SOURCE CODE:

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

# Load the Iris dataset
iris = datasets.load_iris()
X = iris.data  # Features (4D)
y = iris.target  # Labels (0,1,2)

# Apply PCA to reduce from 4D to 2D
pca = PCA(n_components=2)  # Reduce to 2 dimensions
X_pca = pca.fit_transform(X)

# Print explained variance ratio
explained_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 data
plt.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 (pandas, tensorflow, sklearn, numpy).

Step 2: Load the emoji sentiment dataset using pandas.read_csv().

Step 3: Encode the text labels (sentiments) into numeric values using LabelEncoder.

Step 4: Split the dataset into training and testing sets using train_test_split().

Step 5: Create a Tokenizer to convert text sentences into sequences of integers.

Step 6: Fit the tokenizer on the training data and transform both training and testing texts into padded sequences.

Step 7: Build a Sequential neural network model.

Step 8: Compile the model using the Adam optimizer and sparse categorical crossentropy loss.

Step 9: Train the model on the training data for 50 epochs using model.fit(), with validation on the test set, and save the best model using ModelCheckpoint().

Step 10: Save the trained model using model.save().

Step 11: Evaluate the model on the test data using model.evaluate().

Step 12: Load the saved model using load_model() if needed for future predictions.

Step 13: Make predictions on new input texts using the trained model and interpret the predicted labels.

SOURCE CODE:

```
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
```

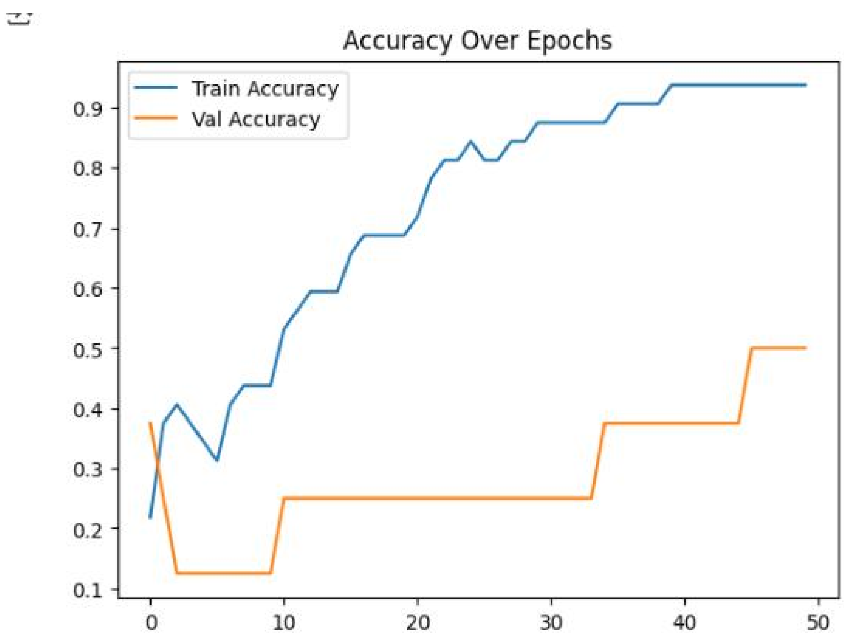
```
from tensorflow.keras.layers import Embedding, GlobalAveragePooling1D, Dense
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

# Load dataset (Use the dataset you created or the one you've downloaded)
df = pd.read_csv('/mnt/data/emoji_sentiment_dataset.csv')  # Adjust the path if needed

# Checking the first few rows of the dataset to verify the structure
print(df.head())

# Label Encoding for Sentiments (assuming 'Happy', 'Sad', etc. in the 'label' column)
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
df['label_encoded'] = label_encoder.fit_transform(df['label'])

# Tokenizer for text data
tokenizer = Tokenizer(num_words=1000, oov_token="<OOV>")
tokenizer.fit_on_texts(df['text'])

# Convert text to sequences
sequences = tokenizer.texts_to_sequences(df['text'])
padded_sequences = pad_sequences(sequences, padding='post')

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(padded_sequences, df['label_encoded'],
test_size=0.2, random_state=42)

# Load GloVe embeddings (Make sure you have this file in your directory)
embeddings_index = {}
with open('glove.6B.100d.txt', 'r', encoding='utf-8') as f:
    for line in f:
        values = line.split()
        word = values[0]
        coefs = np.asarray(values[1:], dtype='float32')
        embeddings_index[word] = coefs

# Prepare embedding matrix
embedding_dim = 100  # GloVe 100-dimensional vectors
embedding_matrix = np.zeros((len(tokenizer.word_index) + 1, embedding_dim))
```

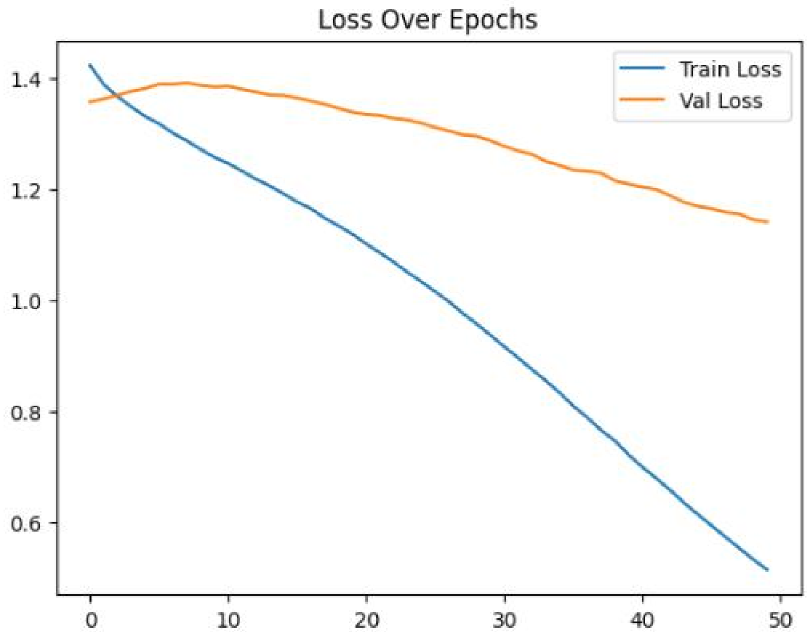
```
for word, i in tokenizer.word_index.items():
    if i < 1000:  # Ensure the word index is within the vocab size
        embedding_vector = embeddings_index.get(word)
        if embedding_vector is not None:
            embedding_matrix[i] = embedding_vector

# Build the model with GloVe embeddings
model = Sequential([
    Embedding(input_dim=len(tokenizer.word_index) + 1,  # Vocabulary size
              output_dim=embedding_dim,
              weights=[embedding_matrix],  # Pre-trained GloVe embeddings
              input_length=padded_sequences.shape[1],
              trainable=False),  # Keep embeddings frozen (you can set to True if you want to fine-
tune)
    GlobalAveragePooling1D(),
    Dense(64, activation='relu'),  # Added more neurons
    Dense(32, activation='relu'),  # Added another layer for complexity
    Dense(len(label_encoder.classes_), activation='softmax')  # Adjust the output layer for your
sentiment classes
])

# Compile the model
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# Train the model
history = model.fit(X_train, y_train, epochs=50, batch_size=16, validation_data=(X_test, y_test))

# Plot training & validation accuracy/loss curves
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Val Accuracy')
plt.legend(loc='best')
plt.title('Accuracy Over Epochs')
plt.show()

plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Val Loss')
plt.legend(loc='best')
plt.title('Loss Over Epochs')
plt.show()
```

```
# Evaluate the model on the test set
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {accuracy * 100:.2f}%")

# Test predictions on new sample texts
sample_texts = ["I am so happy today!", "I feel really sad.", "This is amazing!", "I am very
angry."]
sample_seq = tokenizer.texts_to_sequences(sample_texts)
sample_pad = pad_sequences(sample_seq, padding='post', maxlen=padded_sequences.shape[1])

predictions = model.predict(sample_pad)
predicted_labels = label_encoder.inverse_transform(predictions.argmax(axis=1))

for text, sentiment in zip(sample_texts, predicted_labels):
   print(f"Text: {text} => Predicted Sentiment: {sentiment}")
x_train, x_test = x_train/255.0, x_test/255.0  # Normalize

# Build model
model = tf.keras.Sequential([
   tf.keras.layers.Flatten(input_shape=(28, 28)),
   tf.keras.layers.Dense(128, activation='relu'),
   tf.keras.layers.Dense(10)
])

# Compile and train
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
model.fit(x_train, y_train, epochs=5)

# Evaluate
model.evaluate(x_test, y_test)
predictions = model(x_test)

# Visualize the first 5 test images and their predicted labels
for i in range(5):
   plt.imshow(x_test[i], cmap=plt.cm.binary)
   plt.title(f"Predicted: {np.argmax(predictions[i])} | True: {y_test[i]}")
   plt.show()
```

```
# Evaluate the model on the test set
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {accuracy * 100:.2f}%")

# Test predictions on new sample texts
sample_texts = ["I am so happy today!", "I feel really sad.", "This is amazing!", "I am very
angry."]
sample_seq = tokenizer.texts_to_sequences(sample_texts)
sample_pad = pad_sequences(sample_seq, padding='post', maxlen=padded_sequences.shape[1])

predictions = model.predict(sample_pad)
predicted_labels = label_encoder.inverse_transform(predictions.argmax(axis=1))

for text, sentiment in zip(sample_texts, predicted_labels):
    print(f"Text: {text} => Predicted Sentiment: {sentiment}")
```

OUTPUT:






```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 69ms/step - accuracy: 0.5000 - loss: 1.1404
Test Accuracy: 50.00%
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 388ms/step
Text: I am so happy today! => Predicted Sentiment: Happy
Text: I feel really sad. => Predicted Sentiment: Sad
Text: This is amazing! => Predicted Sentiment: Happy
Text: I am very angry. => Predicted Sentiment: Angry
```

RESULT:

Thus a simple application using tensorflow/keras is developed.