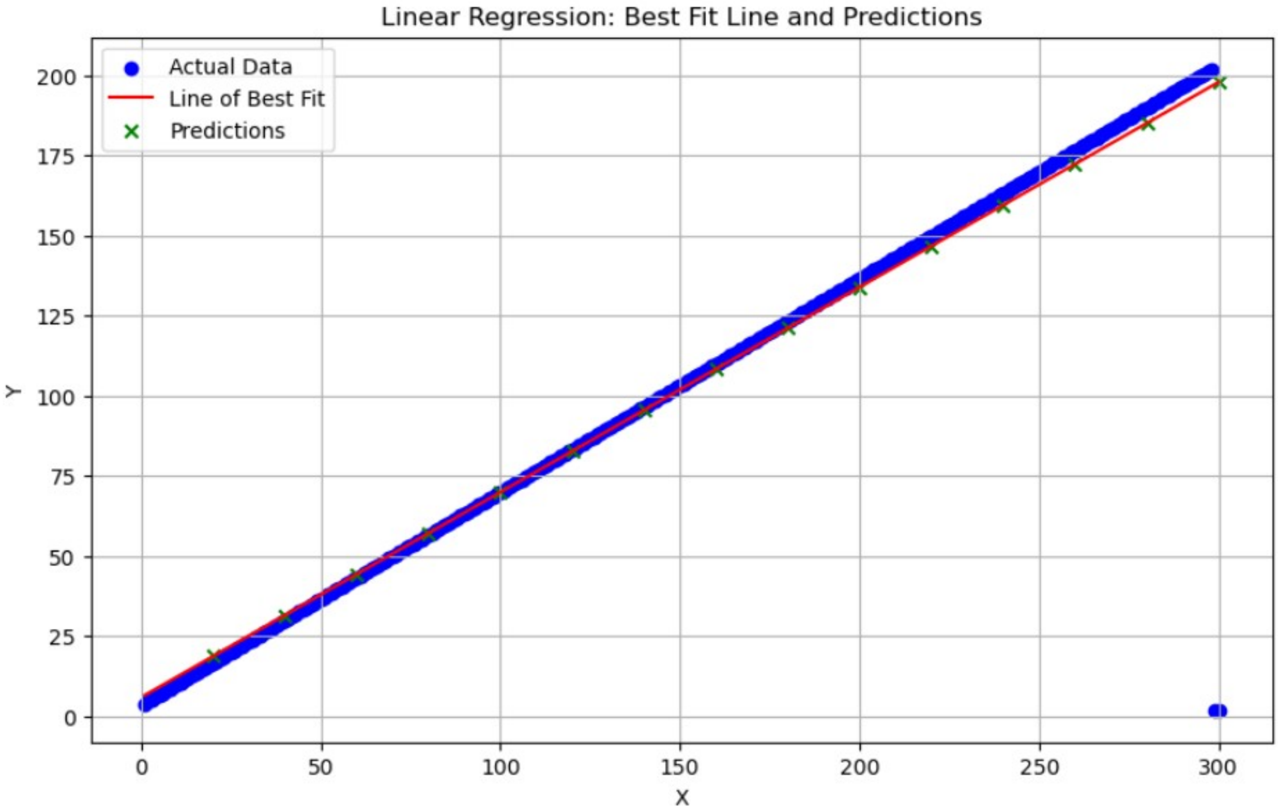
Nithin S
221IT085

IT464 Lab Assignment 1

Q1. Find the line of best fit (linear regression) for the following data set and plot it. https://www.kaggle.com/datasets/tanuprabhu/linear-regression-dataset Find the predictions to x=[20,40,60,...,280,300] and Compute the least squared error (LSE). Print/tabulate ” x, predictions and LSE”.

Code

```python
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
import matplotlib.pyplot as plt

df = pd.read_csv('LR.csv')
X = df['X'].values.reshape(-1, 1)
y = df['Y'].values


model = LinearRegression()
model.fit(X, y)


x_pred = np.arange(20, 301, 20).reshape(-1, 1)
y_pred = model.predict(x_pred)

y_pred_original = model.predict(X)

# Calculate the total Least Squared Error (LSE) for the original data
lse_total = np.sum((y - y_pred_original) ** 2)

# Use the mean of the actual y values as the "actual" values for the predicted dataset
y_mean = np.mean(y)
y_actual_pred = np.full_like(y_pred, y_mean)


lse_individual_pred = (y_actual_pred - y_pred) ** 2


r2 = r2_score(y, y_pred_original)
accuracy_percentage = r2 * 100

results_df = pd.DataFrame({
    'X': x_pred.flatten(),
    'Predicted_Y': y_pred,
    'Actual_Y': y_actual_pred,
    'Individual_LSE': lse_individual_pred
})

plt.figure(figsize=(10, 6))
plt.scatter(X, y, color='blue', label='Actual Data')
plt.plot(X, model.predict(X), color='red', label='Line of Best Fit')
plt.scatter(x_pred, y_pred, color='green', marker='x', label='Predictions')
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Linear Regression: Best Fit Line and Predictions')
plt.legend()
plt.grid(True)
plt.show()

# Print the regression equation, LSE, and R-squared score
print(f"\nRegression Equation: y = {model.coef_[0]:.4f}x + {model.intercept_:.4f}")
print(f"Total Least Squared Error (LSE): {lse_total:.4f}")
print(f"R-squared Score (R²): {r2:.4f}")
print(f"Accuracy (R² in percentage): {accuracy_percentage:.2f}%")

# Display results
print("\nPredictions and LSE for Predicted Data:")
print(results_df.to_string(index=False))
```



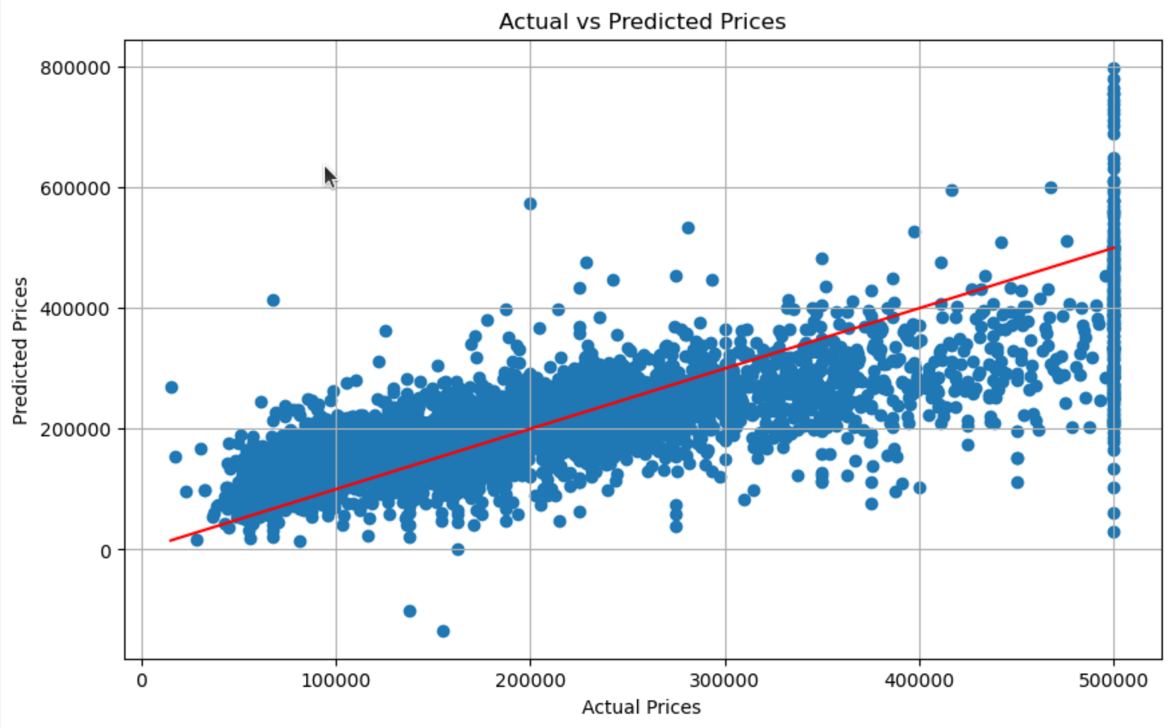

```
Regression Equation: y = 0.6400x + 5.8888
Total Least Squared Error (LSE): 78668.9421
R-squared Score (R²): 0.9214
Accuracy (R² in percentage): 92.14%

Predictions and LSE for Predicted Data:
  X  Predicted_Y  Individual_LSE
 20    18.689697     6976.568967
 40    31.490595     5002.020007
 60    44.291493     3355.197016
 80    57.092391     2036.099993
100    69.893289     1044.728939
120    82.694186      381.083853
140    95.495084       45.164735
160   108.295982       36.971586
180   121.096880      356.504405
200   133.897778     1003.763193
220   146.698675     1978.747949
240   159.499573     3281.458673
260   172.300471     4911.895366
280   185.101369     6870.058027
300   197.902267     9155.946656
```

Q2. Find the line of best fit (multiple linear regression - MLR) for the California Housing
Dataset and plot it.
https://www.geeksforgeeks.org/dataset-for-linear-regression/
Note: Exclude “longitude, latitude and ocean proximity” parameters/variables.

Code

```
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import numpy as np

housing_data = pd.read_csv('housing.csv')
housing_data = housing_data.drop(['longitude', 'latitude', 'ocean_proximity'], axis=1)

X = housing_data.drop('median_house_value', axis=1)
y = housing_data['median_house_value']

imputer = SimpleImputer(strategy='median')
X_imputed = imputer.fit_transform(X)

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

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

mlr_model = LinearRegression()
mlr_model.fit(X_train_scaled, y_train)

print("Coefficients:", mlr_model.coef_)
print("Intercept:", mlr_model.intercept_)
y_pred = mlr_model.predict(X_test_scaled)
plt.scatter(y_test, y_pred)
plt.xlabel('Actual Prices')
plt.ylabel('Predicted Prices')
plt.title('Actual vs Predicted Prices')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red')  # Line of best fit
plt.show()

import pandas as pd


housing_test_data= pd.read_csv("housing2.csv", skiprows=2)
housing_test_data = housing_test_data.drop(housing_test_data.columns[0], axis=1)
housing_test_data= housing_test_data.drop(index=housing_test_data.index[0]).reset_index(drop=True)

housing_test_data.columns
housing_test_data

expected_columns = ['housing_median_age','total_rooms', 'total_bedrooms', 'population', 'households', 'median_income']
housing_test_data = housing_test_data[expected_columns]
housing_test_data.dropna()
len(housing_test_data)
housing_test_data_imputed = imputer.transform(housing_test_data)
housing_test_data_scaled = scaler.transform(housing_test_data_imputed)
len(housing_test_data_scaled)
housing_test_predictions = mlr_model.predict(housing_test_data_scaled)
housing_test_data['Predicted_Price'] = housing_test_predictions
housing_test_data
```

```
Coefficients: [ 23906.19520346 -43497.74884467  42713.1788278  -40536.87160442
  48392.8781107   91325.09220035]
Intercept: 207194.6937378882
```




```
Mean Absolute Error (MAE): 56713.67
Mean Squared Error (MSE): 5968852333.91
Root Mean Squared Error (RMSE): 77258.35
R-squared Score (R²): 0.5445

Test Data with Predictions:
```

[11]:

	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	Predicted_Price
0	20.0	4113.0	302.0	999.0	302.0	6.7905	266969.176243
1	10.0	7837.0	645.0	2307.0	265.0	2.0460	-70398.852854
2	63.0	8677.0	619.0	1659.0	227.0	8.5273	339800.487261
3	46.0	3754.0	984.0	607.0	334.0	6.6504	404318.088162
4	10.0	4301.0	578.0	653.0	391.0	5.5973	238791.306407
5	23.0	4052.0	458.0	1458.0	285.0	6.7502	269326.887482
6	9.0	9692.0	791.0	1472.0	369.0	5.3007	104554.690633
7	42.0	4602.0	916.0	657.0	353.0	7.8508	431036.493171
8	5.0	7083.0	101.0	1289.0	383.0	8.4030	235899.608621
9	33.0	5450.0	404.0	980.0	481.0	3.7130	151106.685909

Q3. Perform MLR and Logistic regression on the following data to predict heart disease. https://www.kaggle.com/datasets/dileep070/heart-disease-prediction-using-logistic-regres sion Predict heart disease for the "heart2" test data.

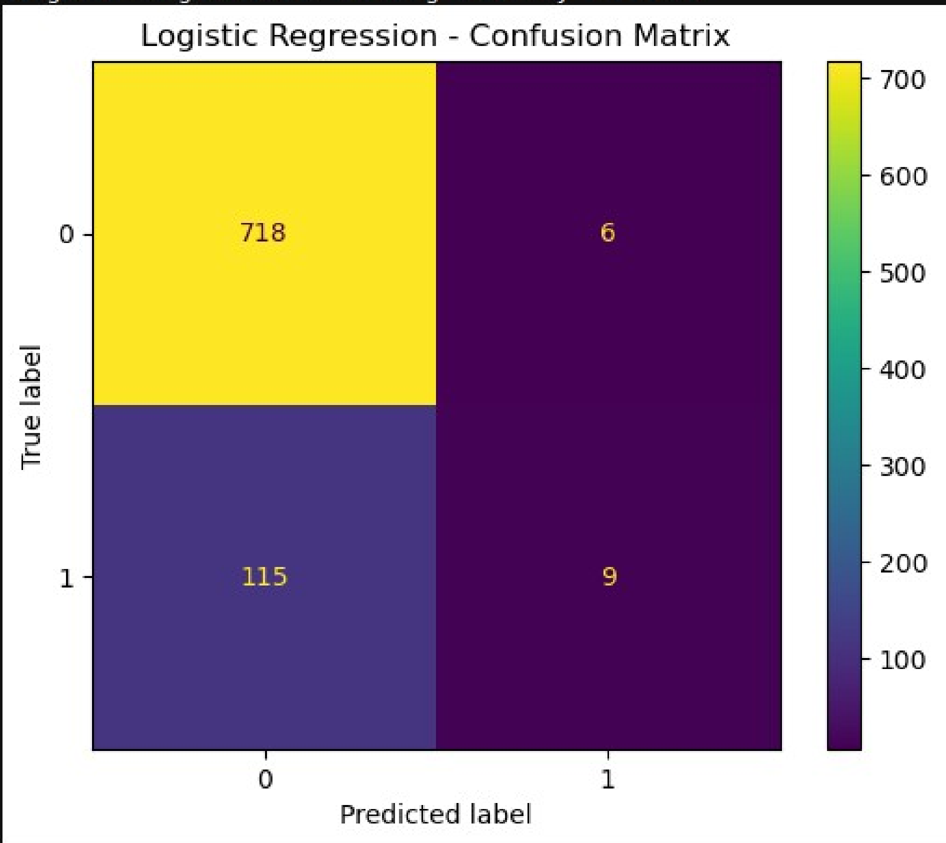
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split
from sklearn.metrics import (ConfusionMatrixDisplay, classification_report, accuracy_score,
                             roc_curve, auc, mean_squared_error, r2_score)
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import LogisticRegression, LinearRegression
from sklearn.preprocessing import MinMaxScaler

df = pd.read_csv('heart.csv')
df.fillna(df.mean(), inplace=True)
X = df.drop(columns=['TenYearCHD'])
target = df['TenYearCHD']

X_train, X_test, y_train, y_test = train_test_split(X, target, test_size=0.2, random_state=42)
print("X_train shape:", X_train.shape)
print("y_train shape:", y_train.shape)
print("X_test shape:", X_test.shape)
print("y_test shape:", y_test.shape)


print("\n==== Logistic Regression ====")
LR_model = make_pipeline(
    SimpleImputer(strategy='mean'),
    MinMaxScaler(),
    LogisticRegression()
)
LR_model.fit(X_train, y_train)


y_training_pred = LR_model.predict(X_train)
y_testing_pred = LR_model.predict(X_test)


training_acc = accuracy_score(y_train, y_training_pred)
testing_acc = accuracy_score(y_test, y_testing_pred)
print(f"Logistic Regression - Training accuracy : {training_acc:.4f}")
print(f"Logistic Regression - Testing accuracy : {testing_acc:.4f}")
```

```python
ConfusionMatrixDisplay.from_estimator(LR_model, X_test, y_test)
plt.title("Logistic Regression - Confusion Matrix")
plt.show()


print("Logistic Regression - Classification Report:\n", classification_report(y_test, y_testing_pred))


y_test_prob = LR_model.predict_proba(X_test)[:, 1]
fpr, tpr, _ = roc_curve(y_test, y_test_prob)
roc_auc = auc(fpr, tpr)

plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (area = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Logistic Regression - ROC Curve')
plt.legend(loc="lower right")
plt.show()


print("\n==== Multiple Linear Regression ====")
MLR_model = make_pipeline(
    SimpleImputer(strategy='mean'),
    MinMaxScaler(),
    LinearRegression()
)
MLR_model.fit(X_train, y_train)


y_train_pred_mlr = MLR_model.predict(X_train)
y_test_pred_mlr = MLR_model.predict(X_test)


train_mse = mean_squared_error(y_train, y_train_pred_mlr)
test_mse = mean_squared_error(y_test, y_test_pred_mlr)
train_r2 = r2_score(y_train, y_train_pred_mlr)
test_r2 = r2_score(y_test, y_test_pred_mlr)
print(f"Multiple Linear Regression - Training MSE: {train_mse:.4f}")
print(f"Multiple Linear Regression - Testing MSE: {test_mse:.4f}")
```

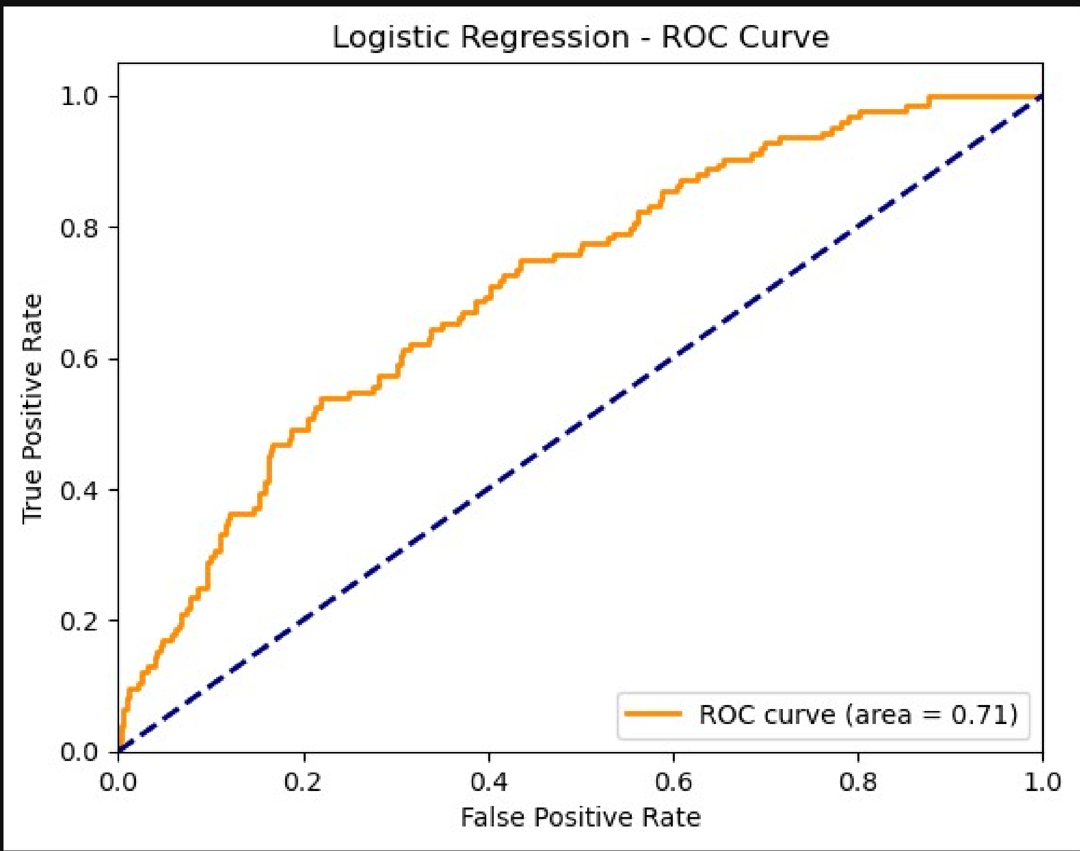
```
print(f"Multiple Linear Regression - Testing MSE: {test_mse:.4f}")
print(f"Multiple Linear Regression - Training R^2 Score: {train_r2:.4f}")
print(f"Multiple Linear Regression - Testing R^2 Score: {test_r2:.4f}")


y_train_pred_mlr_rounded = np.round(y_train_pred_mlr)
y_test_pred_mlr_rounded = np.round(y_test_pred_mlr)


mlr_training_acc = accuracy_score(y_train, y_train_pred_mlr_rounded)
mlr_testing_acc = accuracy_score(y_test, y_test_pred_mlr_rounded)
print(f"Multiple Linear Regression (Rounded) - Training accuracy: {mlr_training_acc:.4f}")
print(f"Multiple Linear Regression (Rounded) - Testing accuracy: {mlr_testing_acc:.4f}")

ConfusionMatrixDisplay.from_predictions(y_test, y_test_pred_mlr_rounded)
plt.title("Multiple Linear Regression (Rounded) - Confusion Matrix")
plt.show()


print("Multiple Linear Regression (Rounded) - Classification Report:\n",
      classification_report(y_test, y_test_pred_mlr_rounded))


fpr_mlr, tpr_mlr, _ = roc_curve(y_test, y_test_pred_mlr_rounded)
roc_auc_mlr = auc(fpr_mlr, tpr_mlr)

plt.figure()
plt.plot(fpr_mlr, tpr_mlr, color='darkorange', lw=2, label=f'ROC curve (area = {roc_auc_mlr:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Multiple Linear Regression (Rounded) - ROC Curve')
plt.legend(loc="lower right")
plt.show()


predict_df = pd.read_csv("heart2.csv", skiprows=2)
predict_df = predict_df.iloc[:, 1:]
predict_df = predict_df.iloc[1:].reset_index(drop=True)
predict_df = predict_df[X_train.columns]

predictions_LR = LR_model.predict(predict_df)
```

```
predict_df = pd.read_csv("heart2.csv", skiprows=2)
predict_df = predict_df.iloc[:, 1:]
predict_df = predict_df.iloc[1:].reset_index(drop=True)
predict_df = predict_df[X_train.columns]

predictions_LR = LR_model.predict(predict_df)
predictions_MLR_rounded = np.round(MLR_model.predict(predict_df))

predict_df['TenYearCHD_Logistic'] = predictions_LR
predict_df['TenYearCHD_Linear'] = predictions_MLR_rounded

print("\nPredictions (Logistic Regression):", predictions_LR)
print("Predictions (Multiple Linear Regression Rounded):", predictions_MLR_rounded)
```

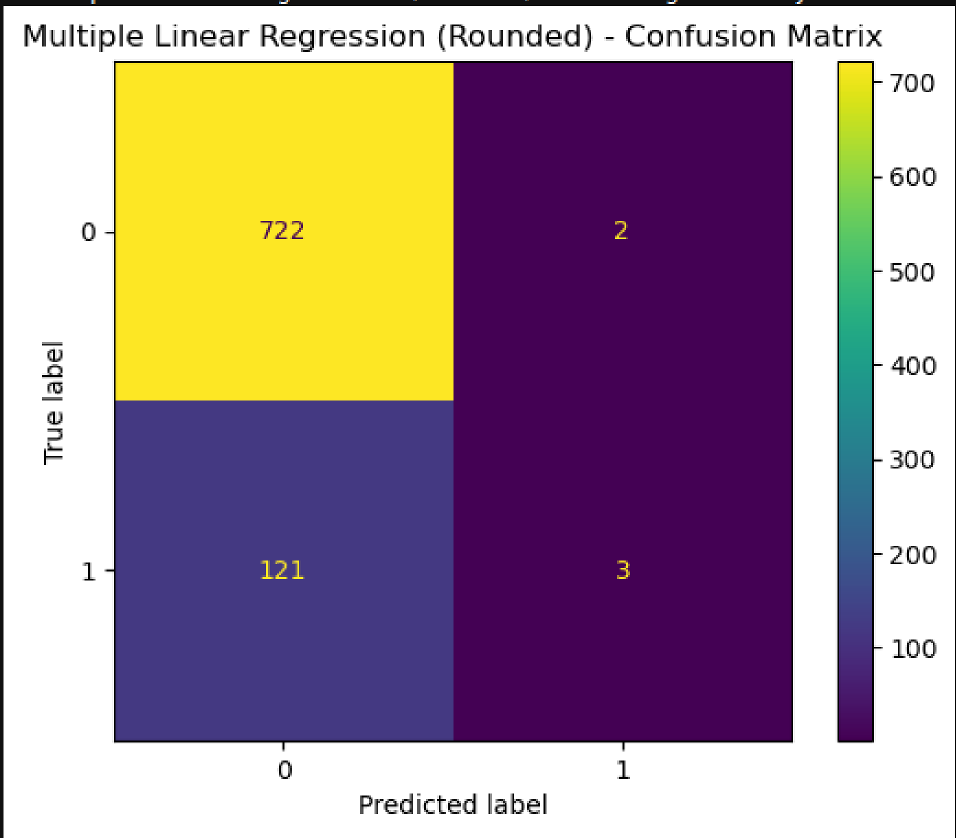
```
X_train shape: (3390, 15)
y_train shape: (3390,)
X_test shape: (848, 15)
y_test shape: (848,)

==== Logistic Regression ====
Logistic Regression - Training accuracy : 0.8546
Logistic Regression - Testing accuracy : 0.8573
```



```
Logistic Regression - Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.99      0.92       724
           1       0.60      0.07      0.13       124

    accuracy                           0.86       848
   macro avg       0.73      0.53      0.53       848
weighted avg       0.82      0.86      0.81       848
```

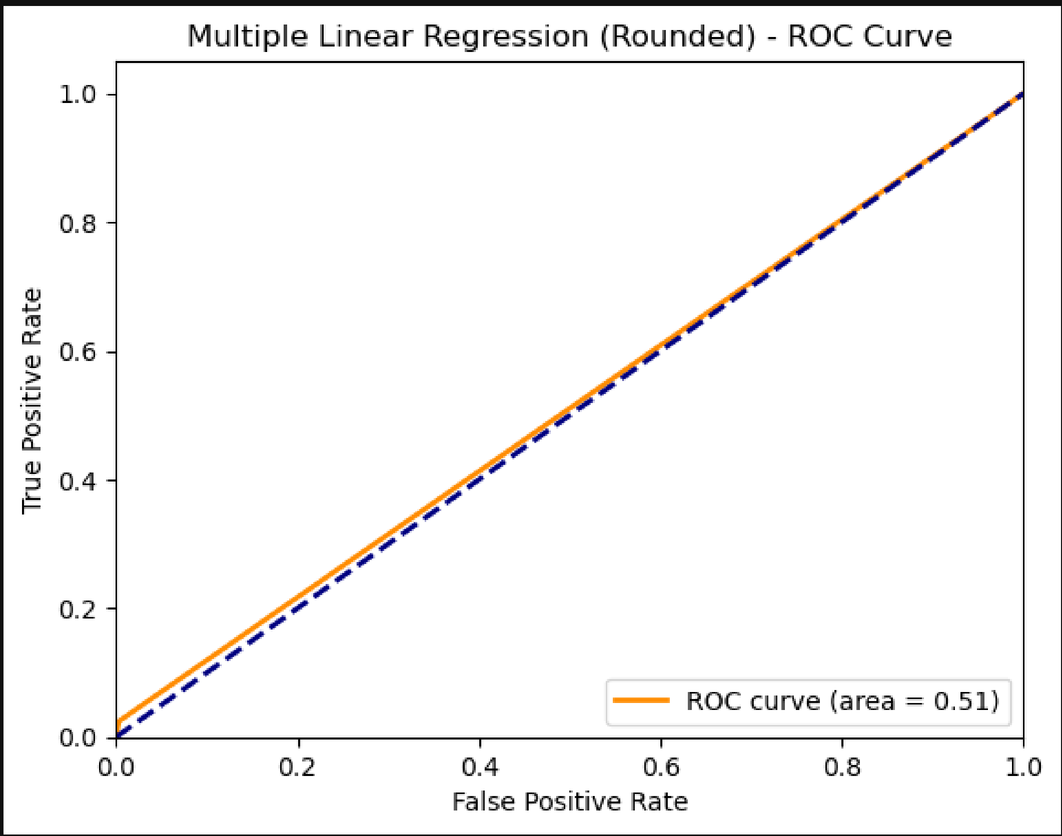


```
==== Multiple Linear Regression ====
Multiple Linear Regression - Training MSE: 0.1167
Multiple Linear Regression - Testing MSE: 0.1160
Multiple Linear Regression - Training R^2 Score: 0.1017
Multiple Linear Regression - Testing R^2 Score: 0.0708
Multiple Linear Regression (Rounded) - Training accuracy: 0.8513
Multiple Linear Regression (Rounded) - Testing accuracy: 0.8550
```




```
Multiple Linear Regression (Rounded) - Classification Report:
              precision    recall  f1-score   support

           0       0.86      1.00      0.92       724
           1       0.60      0.02      0.05       124

    accuracy                           0.85       848
   macro avg       0.73      0.51      0.48       848
weighted avg       0.82      0.85      0.79       848
```

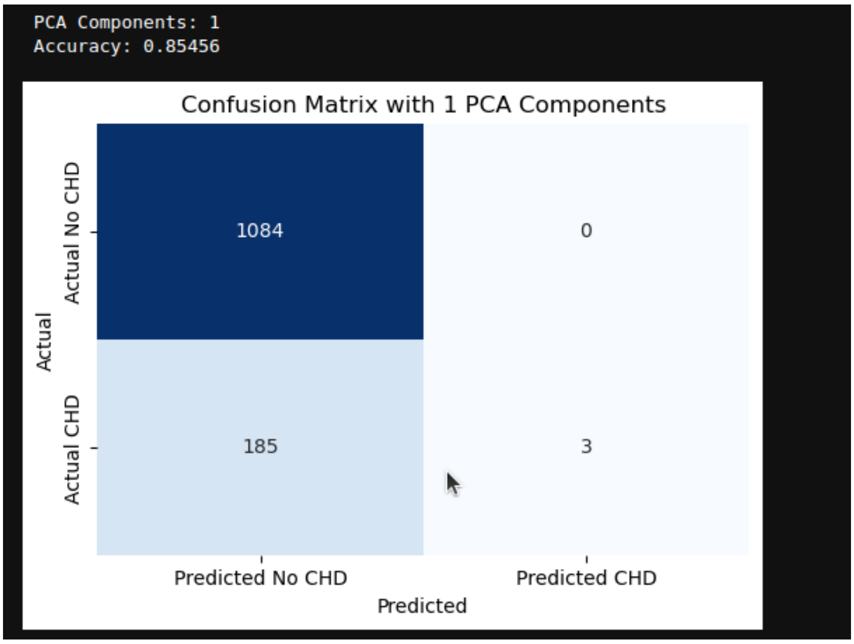
```
Multiple Linear Regression (Rounded) - Classification Report:
              precision    recall  f1-score   support

           0       0.86      1.00      0.92       724
           1       0.60      0.02      0.05       124

    accuracy                           0.85       848
   macro avg       0.73      0.51      0.48       848
weighted avg       0.82      0.85      0.79       848
```

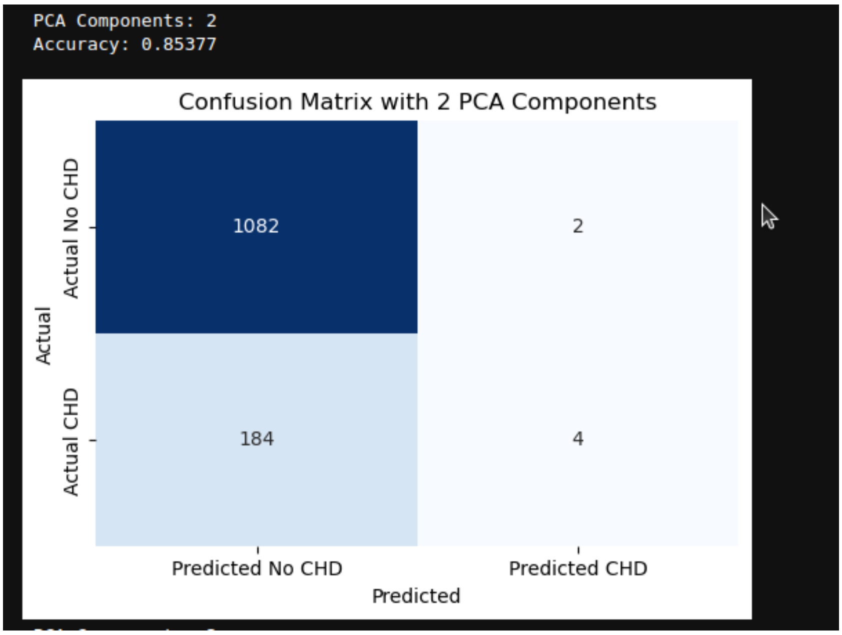


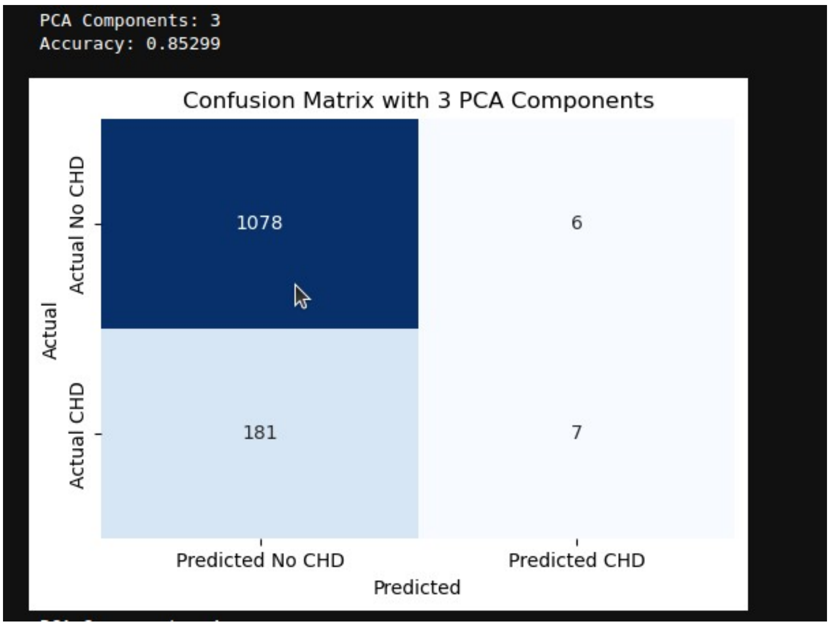

```
Predictions (Logistic Regression): [1 0 1 0 0 1 1 0 1 0]
Predictions (Multiple Linear Regression Rounded): [1. 0. 1. 1. 0. 1. 1. 0. 1. 0.]
```

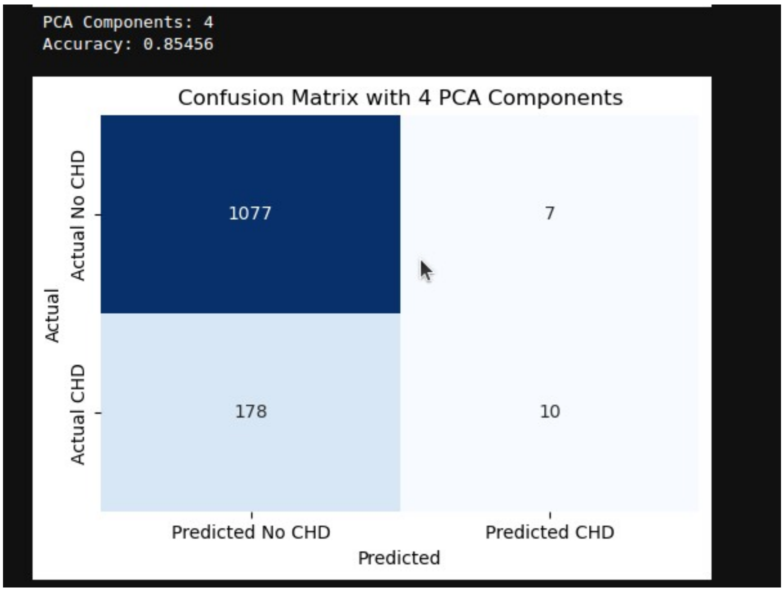
Q4. Compute PCA components for the heart disease data. Predict heart disease with the PCA features (consider #PCA features = [1,2,3,4,5]) and evaluate the performance in terms of confusion matrix. Note down your observations

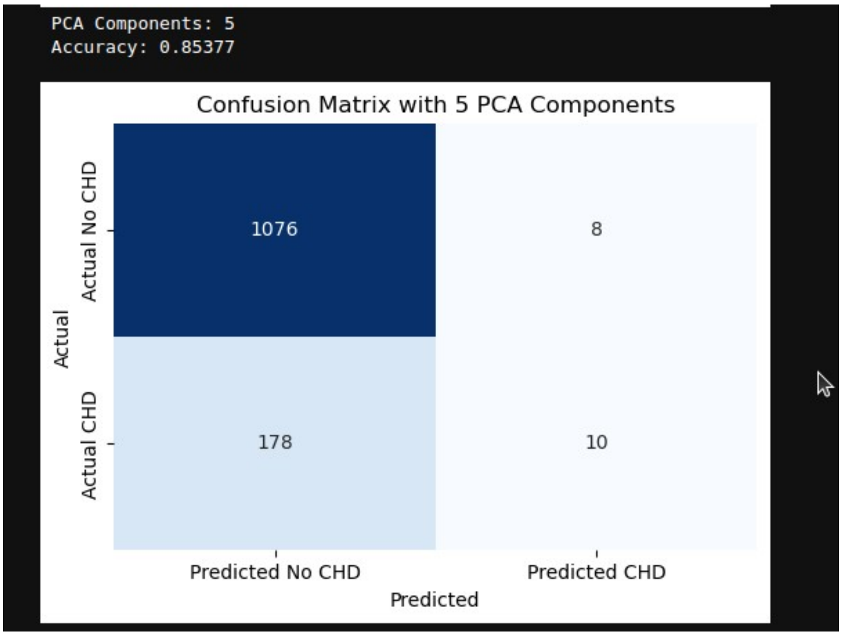
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import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

df = pd.read_csv('heart.csv')
df.fillna(df.mean(), inplace=True)
X = df.drop(columns=['TenYearCHD'], axis=1)
y = df['TenYearCHD']
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
accuracies = []
conf_matrices = []
for n_components in range(1, 8):
    pca = PCA(n_components=n_components)
    X_pca = pca.fit_transform(X_scaled)
    X_train, X_test, y_train, y_test = train_test_split(X_pca, y, test_size=0.3, random_state=42)
    model = LogisticRegression()
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    conf_matrix = confusion_matrix(y_test, y_pred)
    accuracy = accuracy_score(y_test, y_pred)
    conf_matrices.append(conf_matrix)
    accuracies.append(accuracy)
    print(f"PCA Components: {n_components}")
    print("Confusion Matrix:")
    print(conf_matrix)
    print(f"Accuracy: {accuracy:.5f}\n")

plt.plot(range(1, 8), accuracies, marker='o')
plt.xlabel('Number of PCA Components')
plt.ylabel('Accuracy')
plt.title('Accuracy vs. Number of PCA Components')
plt.grid(True)
plt.show()

predict_df = pd.read_csv("heart2.csv", skiprows=2)
predict_df = predict_df.iloc[:, 1:]
predict_df = predict_df.iloc[1:].reset_index(drop=True)
predict_df_scaled = scaler.transform(predict_df)

for n_components in range(1, 8):
    pca = PCA(n_components=n_components)
    predict_df_pca = pca.fit_transform(predict_df_scaled)
    X_pca = PCA(n_components=n_components).fit_transform(X_scaled)
    X_train, X_test, y_train, y_test = train_test_split(X_pca, y, test_size=0.3, random_state=42)
    model = LogisticRegression()
    model.fit(X_train, y_train)
    predictions = model.predict(predict_df_pca)
    print(f"Predictions with {n_components} PCA Components:")
    print(predictions)
```

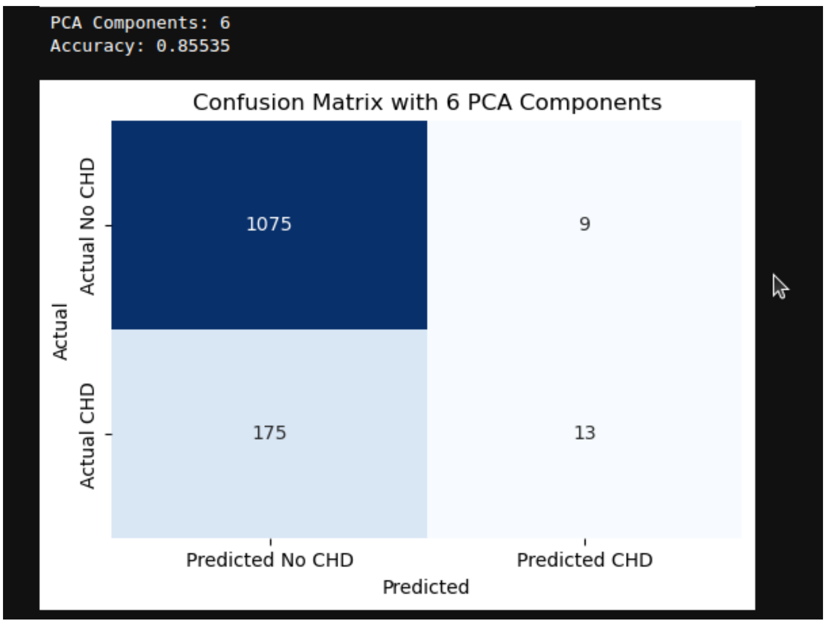

PCA Components: 1
Accuracy: 0.85456
Confusion Matrix with 1 PCA Components
Actual
Actual No CHD
Actual CHD
1084
0
185
3
Predicted No CHD
Predicted CHD
Predicted

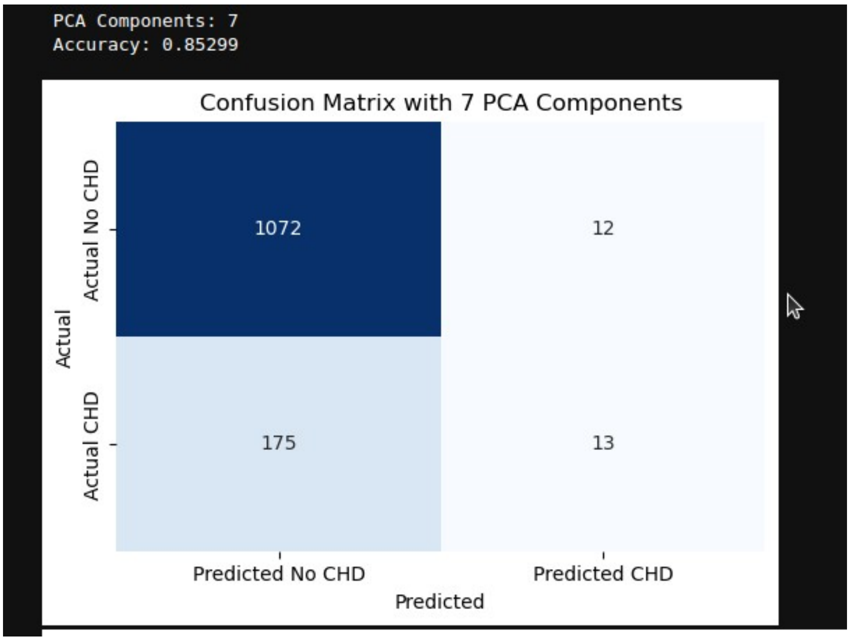

PCA Components: 2
Accuracy: 0.85377
Confusion Matrix with 2 PCA Components
Actual
Actual No CHD
Actual CHD
1082
2
184
4
Predicted No CHD
Predicted CHD
Predicted

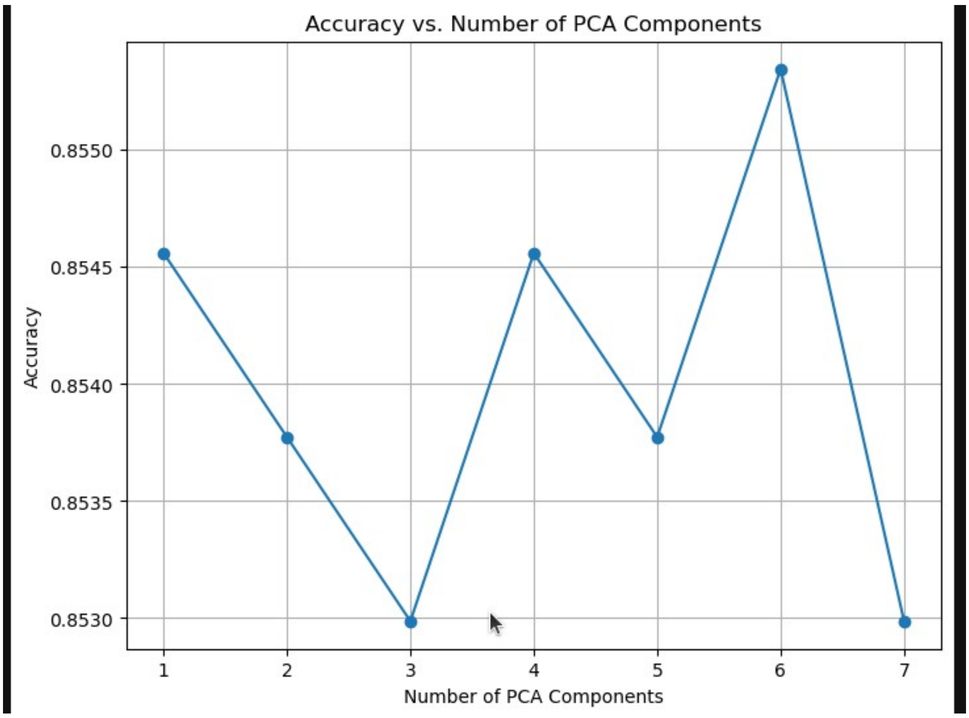

PCA Components: 3
Accuracy: 0.85299
Confusion Matrix with 3 PCA Components
Actual
Actual No CHD
Actual CHD
1078
6
181
7
Predicted No CHD
Predicted CHD
Predicted


PCA Components: 4
Accuracy: 0.85456
Confusion Matrix with 4 PCA Components
Actual
Actual No CHD
Actual CHD
1077
7
178
10
Predicted No CHD
Predicted CHD
Predicted


PCA Components: 5
Accuracy: 0.85377
Confusion Matrix with 5 PCA Components
Actual
Actual No CHD
Actual CHD
1076
8
178
10
Predicted No CHD
Predicted CHD
Predicted


PCA Components: 6
Accuracy: 0.85535
Confusion Matrix with 6 PCA Components
Actual
Actual No CHD
Actual CHD
1075
9
175
13
Predicted No CHD
Predicted CHD
Predicted


PCA Components: 7
Accuracy: 0.85299
Confusion Matrix with 7 PCA Components
Actual No CHD
Actual CHD
Actual
1072
12
175
13
Predicted No CHD
Predicted CHD
Predicted


Accuracy vs. Number of PCA Components
Accuracy
0.8550
0.8545
0.8540
0.8535
0.8530
1
2
3
4
5
6
7
Number of PCA Components

```
Predictions with 1 PCA Components:
[0 0 1 0 0 1 1 0 0 0]
Predictions with 2 PCA Components:
[0 0 1 0 0 1 1 0 0 0]
Predictions with 3 PCA Components:
[0 0 1 0 0 1 1 0 0 0]
Predictions with 4 PCA Components:
[0 0 1 0 0 1 1 0 0 0]
Predictions with 5 PCA Components:
[0 0 1 0 0 1 1 0 0 0]
Predictions with 6 PCA Components:
[0 0 1 0 0 1 1 0 0 0]
Predictions with 7 PCA Components:
[0 0 1 0 0 1 1 0 0 0]
```

Observations

PCA is used to reduce the dimensionality of the dataset while retaining as much variance as possible. This can help in improving the performance and reducing the complexity of the model.

Each principal component will explain a certain amount of the variance in the dataset. The first few components usually capture the majority of the variance.

As we increase the number of PCA components, we observe changes in the accuracy of the logistic regression model. Initially, with fewer components, the model doesn't not perform well due to loss of information. As we add more components, the performance will improve up to a certain point and then plateau or even decrease due to overfitting.

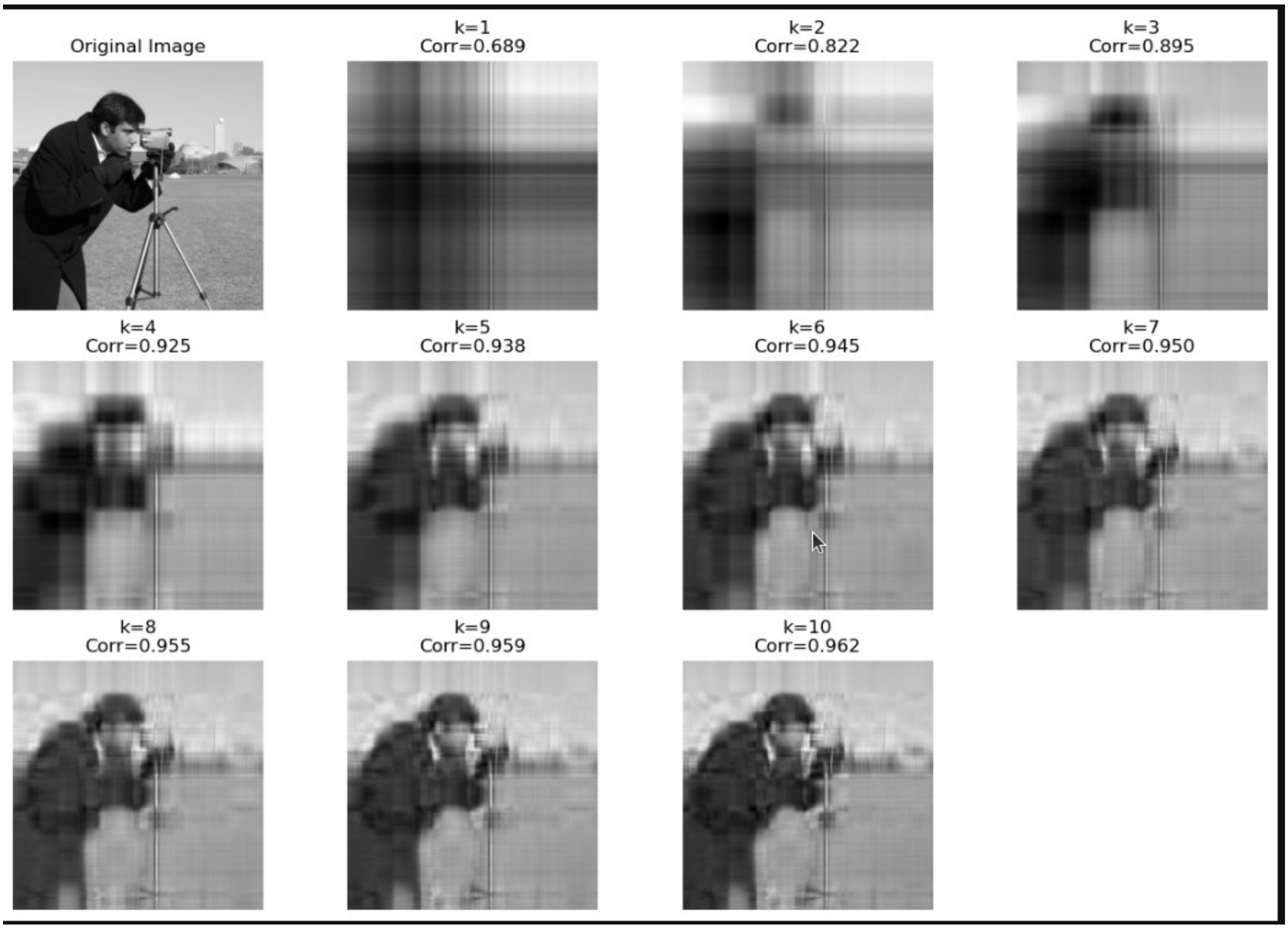
The confusion matrices shows how well the model is performing in terms of true positives, true negatives, false positives, and false negatives. We observe that with more PCA components, the confusion matrix becomes more balanced, indicating better model performance.

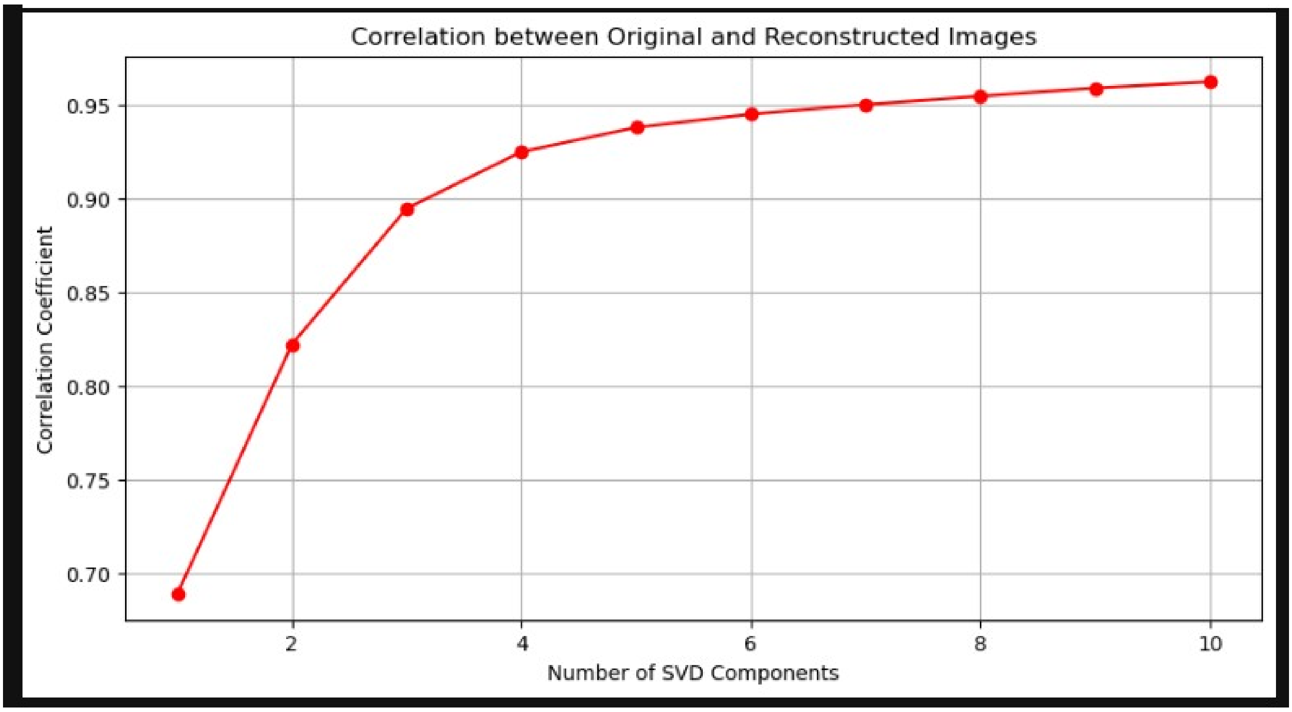
Plotting the accuracy against the number of PCA components will help we visualize the optimal number of components. We find that after a certain number of components, the accuracy does not improve significantly.

The predictions on the new dataset using different numbers of PCA components shows how well the model generalizes to unseen data. We observe differences in predictions with different numbers of components.

Q5. Load Cameraman image from python libraries and reduce the dimensionality using
SVD, Check its visual appearances (original versus new image) for the different numbers of SVD components. Additionally, find the correlation between the original and
reconstructed images from the different numbers of SVD components (say 1,2,...,9,10)

```
import numpy as np
from skimage import data
import matplotlib.pyplot as plt
from numpy.linalg import svd

image = data.camera()
U, s, Vt = svd(image, full_matrices=False)
def reconstruct_image(U, s, Vt, k):
    return np.matrix(U[:, :k]) * np.diag(s[:k]) * np.matrix(Vt[:k, :])
plt.figure(figsize=(15, 10))
plt.subplot(3, 4, 1)
plt.imshow(image, cmap='gray')
plt.title('Original Image')
plt.axis('off')
correlations = []
for i, k in enumerate([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]):
    reconstructed = reconstruct_image(U, s, Vt, k)
    correlation = np.corrcoef(image.flatten(), np.array(reconstructed).flatten())[0,1]
    correlations.append(correlation)
    plt.subplot(3, 4, i+2)
    plt.imshow(reconstructed, cmap='gray')
    plt.title(f'k={k}\nCorr={correlation:.3f}')
    plt.axis('off')
plt.figure(figsize=(10, 5))
plt.plot(range(1, 11), correlations, 'ro-')
plt.xlabel('Number of SVD Components')
plt.ylabel('Correlation Coefficient')
plt.title('Correlation between Original and Reconstructed Images')
plt.grid(True)
plt.show()
print("\nCorrelation coefficients:")
for k, corr in enumerate(correlations, 1):
    print(f"Components: {k}, Correlation: {corr:.4f}")
```


Original Image
k=1
Corr=0.689
k=2
Corr=0.822
k=3
Corr=0.895
k=4
Corr=0.925
k=5
Corr=0.938
k=6
Corr=0.945
k=7
Corr=0.950
k=8
Corr=0.955
k=9
Corr=0.959
k=10
Corr=0.962



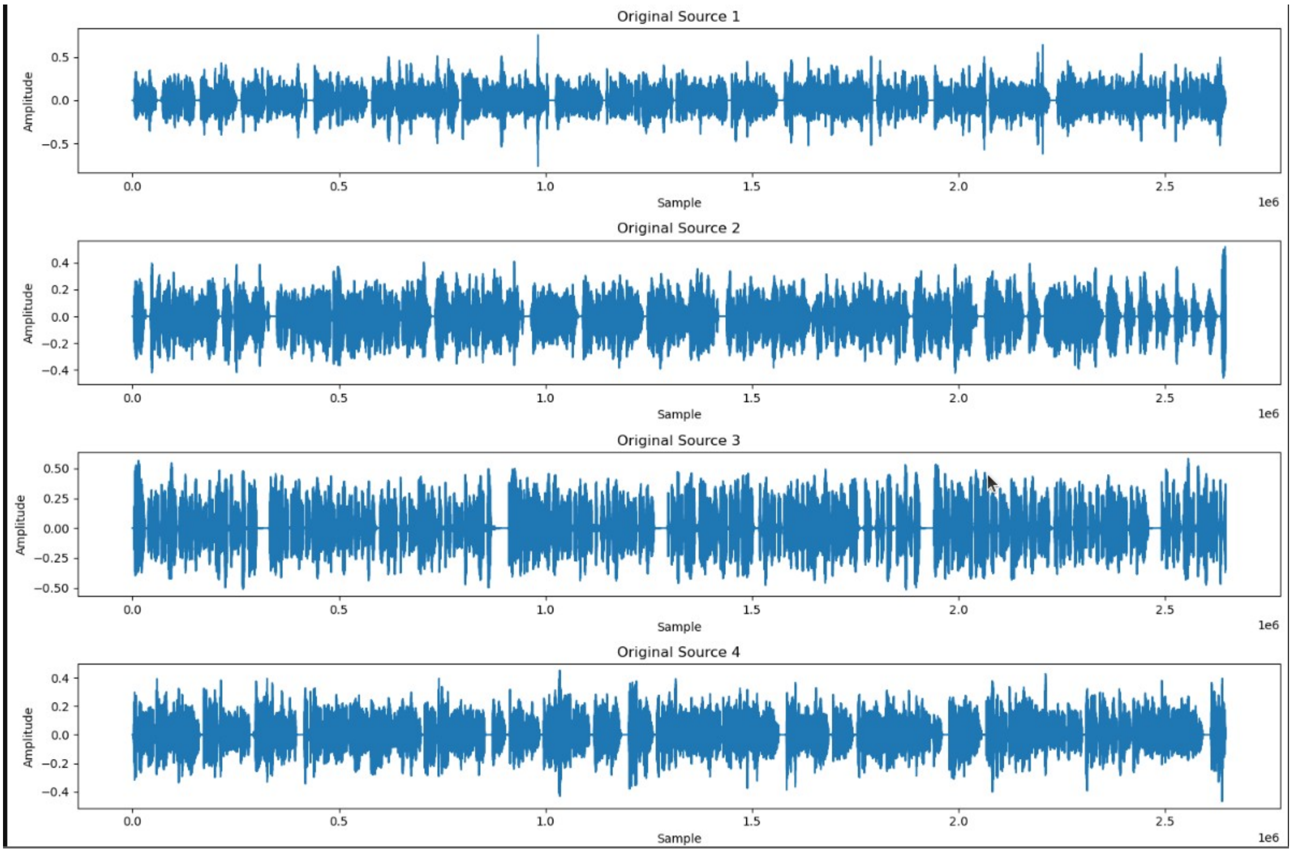

```
Correlation coefficients:
Components: 1, Correlation: 0.6888
Components: 2, Correlation: 0.8221
Components: 3, Correlation: 0.8946
Components: 4, Correlation: 0.9248
Components: 5, Correlation: 0.9378
Components: 6, Correlation: 0.9449
Components: 7, Correlation: 0.9501
Components: 8, Correlation: 0.9545
Components: 9, Correlation: 0.9588
Components: 10, Correlation: 0.9622
```

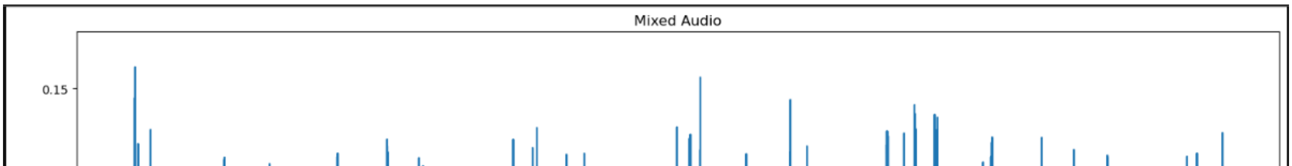
Q6. Load Cocktail Party Problem dataset from kaggle to perform ICA on separating the audios of different speakers. Test PCA and compare its performance with ICA’s in source separation problem.
https://www.kaggle.com/datasets/anashamoutni/cocktail-party-problem-cities-of-the-us

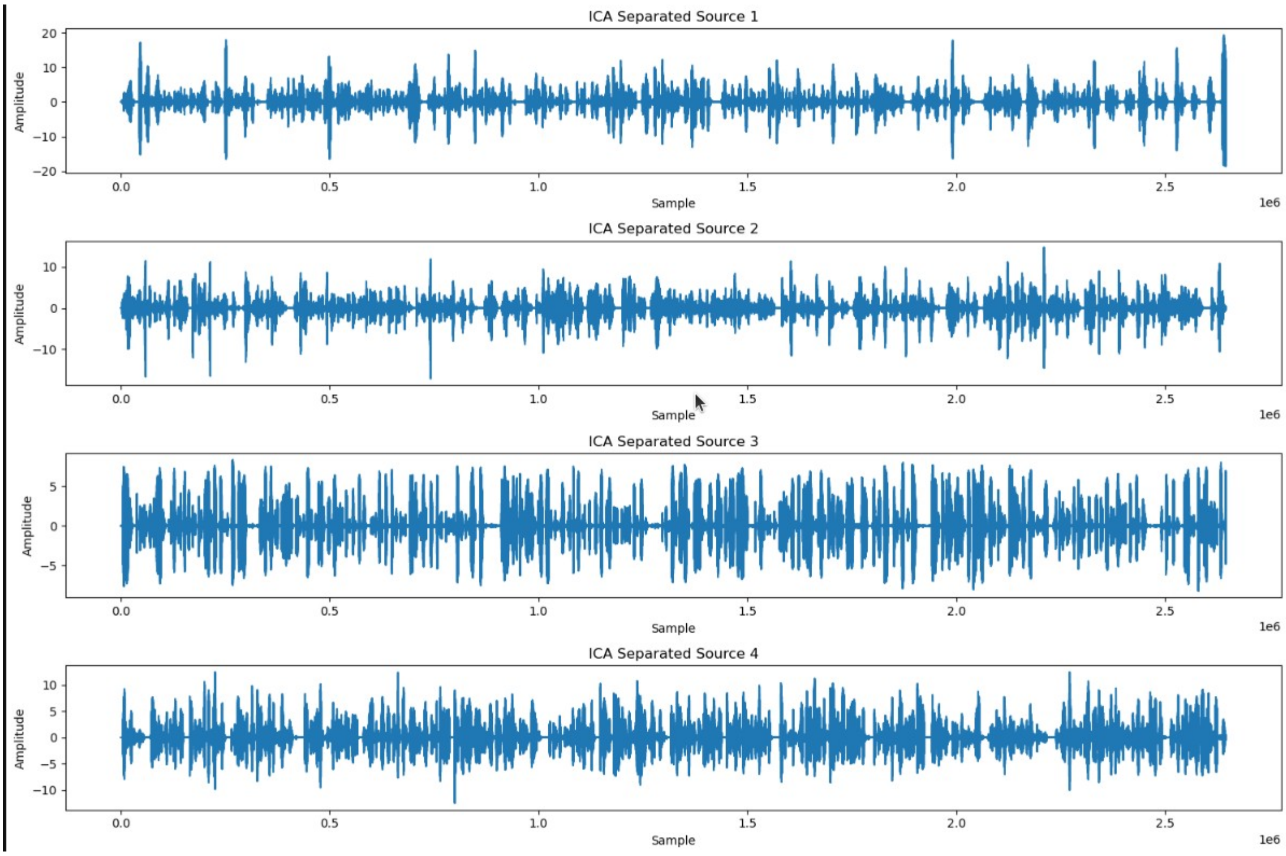
```python
import numpy as np
import matplotlib.pyplot as plt
import librosa
from sklearn.decomposition import FastICA, PCA
from sklearn.preprocessing import StandardScaler
from scipy.signal import butter, filtfilt
from IPython.display import Audio
from IPython.display import Audio, display

original_paths = [
    "Washinton - American Woman.mp3",
    "New York - American Man.mp3",
    "Miami - Australian Woman.mp3",
    "Chicago - British Woman.mp3"
]

original_audios = [librosa.load(path, sr=44100)[0][:44100 * 60] for path in original_paths]

for i, audio in enumerate(original_audios):
    plt.subplot(4, 1, i + 1)
    plt.plot(audio)
    plt.title(f'Original Audio {i+1}')
plt.tight_layout()
plt.show()

for audio in original_audios:
    display(Audio(audio, rate=44100))

mixed_path = "Audio mix.mp3"
mixed_audio, _ = librosa.load(mixed_path, sr=44100)
mixed_audio = mixed_audio[:44100 * 60]

plt.plot(mixed_audio)
plt.title('Mixed Audio')
plt.show()

display(Audio(mixed_audio, rate=44100))



def play_audio_in_jupyter(audio_results):
    """
    Play audio results specifically in Jupyter Notebook
    """
    print("Original Sources:")
    for i, source in enumerate(audio_results['original'], 1):
        print(f"Original Source {i}")
        display(source)

    print("\nMixed Audio:")
    display(audio_results['mixed'])

    print("\nICA Separated Sources:")
    for i, source in enumerate(audio_results['ica_separated'], 1):
        print(f"ICA Separated Source {i}")
        display(source)

    print("\nPCA Separated Sources:")
    for i, source in enumerate(audio_results['pca_separated'], 1):
        print(f"PCA Separated Source {i}")
        display(source)
```

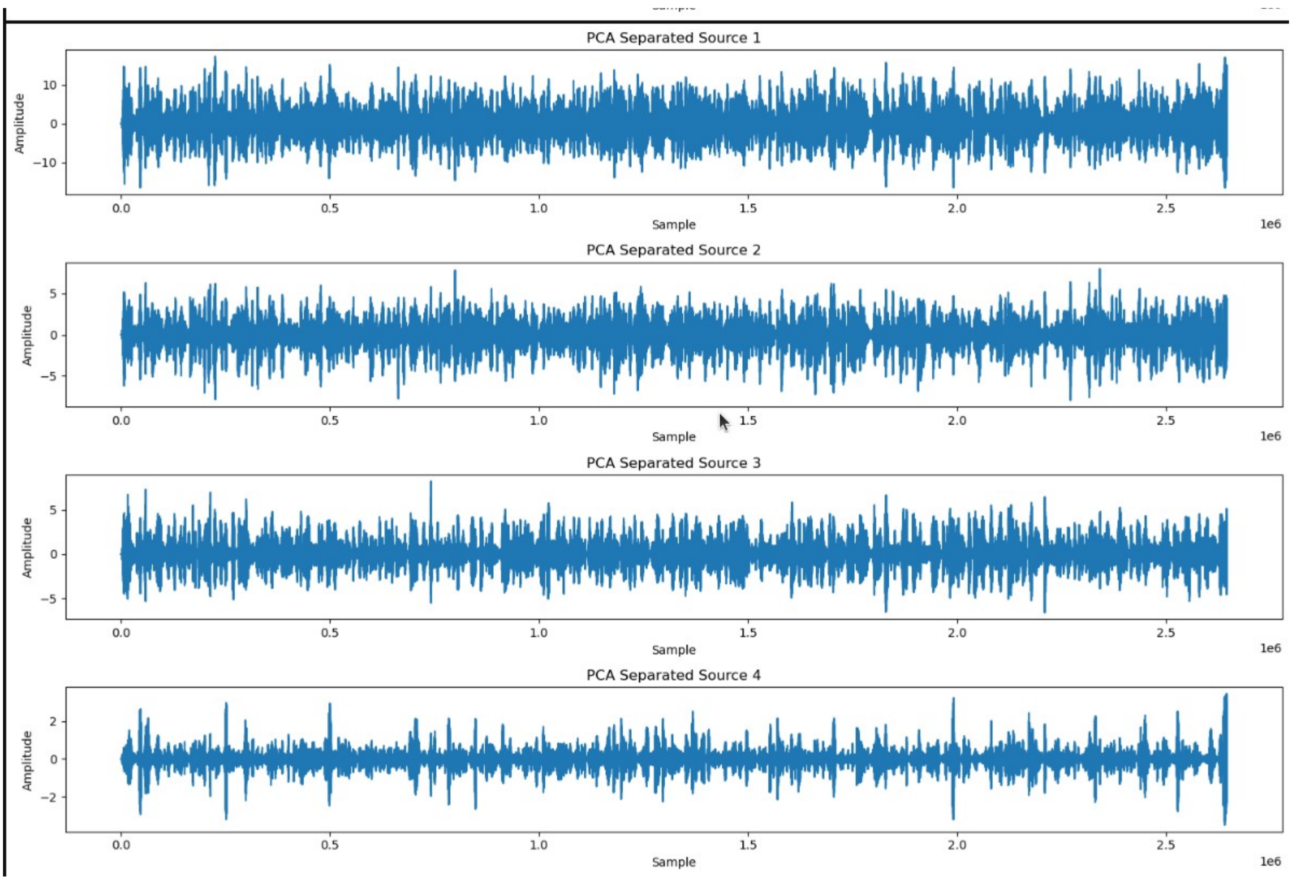
```python
def load_and_preprocess_audio(file_paths, duration=60, sr=44100):
    audios = []
    for path in file_paths:
        audio, _ = librosa.load(path, sr=sr)
        # Trim to specified duration
        audio = audio[:sr * duration]
        # Pad if audio is shorter than desired length
        if len(audio) < sr * duration:
            audio = np.pad(audio, (0, sr * duration - len(audio)))
        audios.append(audio)
    return np.array(audios)

def apply_bandpass_filter(audio, lowcut=500, highcut=3000, sr=44100, order=5):
    nyquist = 0.5 * sr
    low = lowcut / nyquist
    high = highcut / nyquist
    b, a = butter(order, [low, high], btype='band')
    return filtfilt(b, a, audio)

def separate_sources(mixed_signals, n_components, method='ica'):
    X = mixed_signals.T
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    if method.lower() == 'ica':
        separator = FastICA(n_components=n_components, random_state=42)
    else:  # PCA
        separator = PCA(n_components=n_components)

    # Perform separation
    separated = separator.fit_transform(X_scaled)

    return separated.T

def plot_waveforms(signals, titles, figsize=(15, 10)):
    n_signals = len(signals)
    plt.figure(figsize=figsize)

    for i, (signal, title) in enumerate(zip(signals, titles)):
        plt.subplot(n_signals, 1, i + 1)
        plt.plot(signal)
        plt.title(title)
        plt.xlabel('Sample')
        plt.ylabel('Amplitude')

    plt.tight_layout()
    plt.show()
```

```python
# Main processing pipeline
def process_audio_separation(original_paths, mixed_path, duration=60, sr=44100):
    original_sources = load_and_preprocess_audio(original_paths, duration, sr)

    mixed_audio, _ = librosa.load(mixed_path, sr=sr)
    mixed_audio = mixed_audio[:sr * duration]

    n_sources = len(original_paths)
    mixing_matrix = np.random.rand(n_sources, n_sources)
    artificial_mixtures = np.dot(mixing_matrix, original_sources)

    all_mixtures = np.vstack([mixed_audio, artificial_mixtures])

    filtered_mixtures = np.array([apply_bandpass_filter(mix) for mix in all_mixtures])

    separated_ica = separate_sources(filtered_mixtures, n_components=n_sources, method='ica')
    separated_pca = separate_sources(filtered_mixtures, n_components=n_sources, method='pca')

    plot_waveforms(original_sources,
                   [f'Original Source {i+1}' for i in range(n_sources)])

    plot_waveforms([mixed_audio], ['Mixed Audio'])

    plot_waveforms(separated_ica,
                   [f'ICA Separated Source {i+1}' for i in range(n_sources)])

    plot_waveforms(separated_pca,
                   [f'PCA Separated Source {i+1}' for i in range(n_sources)])

    return {
        'original': [Audio(source, rate=sr) for source in original_sources],
        'mixed': Audio(mixed_audio, rate=sr),
        'ica_separated': [Audio(source, rate=sr) for source in separated_ica],
        'pca_separated': [Audio(source, rate=sr) for source in separated_pca]
    }

results = process_audio_separation(original_paths, mixed_path)

play_audio_in_jupyter(results)
```


Original Source 1
Amplitude
Sample
1e6
Original Source 2
Amplitude
Sample
1e6
Original Source 3
Amplitude
Sample
1e6
Original Source 4
Amplitude
Sample
1e6


Mixed Audio
Amplitude
Sample
1e6


ICA Separated Source 1
Amplitude
Sample
1e6
ICA Separated Source 2
Amplitude
Sample
1e6
ICA Separated Source 3
Amplitude
Sample
1e6
ICA Separated Source 4
Amplitude
Sample
1e6


PCA Separated Source 1
Amplitude
Sample
1e6
PCA Separated Source 2
Amplitude
Sample
1e6
PCA Separated Source 3
Amplitude
Sample
1e6
PCA Separated Source 4
Amplitude
Sample
1e6

```
Original Sources:
Original Source 1
0:00 / 1:00
Original Source 2
0:00 / 1:00
Original Source 3
0:00 / 1:00
Original Source 4
0:00 / 1:00

Mixed Audio:
0:00 / 1:00

ICA Separated Sources:
ICA Separated Source 1
0:00 / 1:00
ICA Separated Source 2
0:00 / 1:00
ICA Separated Source 3
0:00 / 1:00
ICA Separated Source 4
0:00 / 1:00

PCA Separated Sources:
PCA Separated Source 1
0:00 / 1:00
PCA Separated Source 2
0:00 / 1:00
PCA Separated Source 3
0:00 / 1:00
PCA Separated Source 4
0:00 / 1:00
```

ICA is more effective for source separation in audio signals because it tries to find components that are statistically independent, which aligns well with the nature of mixed audio sources. PCA, on the other hand, focuses on capturing the maximum variance, which might not necessarily correspond to independent sources.