PANDIT DEENDAYAL ENERGY UNIVERSITY

SCHOOL OF TECHNOLOGY

Pattern Recognition Lab

20CP412P

LAB MANUAL

B.Tech. (Computer Science and Engineering)

Semester 7

Submitted To:

Dr. Tanmay Bhowmik

Submitted By:

HARSH SHAH

21BCP359

G11 Batch

PRACTICAL 1

Name:	Harsh Shah	**Semester:**	VII	**Division:**	6
Roll No.:	21BCP359	**Date:**	23-07-24	**Batch:**	G11
Aim:	Calculate the possible eigen values for the given matrix.				

The Eigen Values are: 3, $\frac{7+\sqrt{41}}{2}$, $\frac{7-\sqrt{41}}{2}$

Q Calculate Possible Eigen Value for matrix A.

21BCP359
Harsh Shah

Given Matrix :

$$A = \begin{bmatrix} 3, 0, 0 \\ 0, 4, 5 \\ 0, 2, 3 \end{bmatrix}$$

$$\text{Identity Matrix } (I) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\text{calculating } A - \lambda I = \begin{bmatrix} 3-\lambda & 0 & 0 \\ 0 & 4-\lambda & 5 \\ 0 & 2 & 3-\lambda \end{bmatrix}$$

To calculate eigenvalues $|A - \lambda I| = 0$

$\Rightarrow (3-\lambda)\left[(4-\lambda)(3-\lambda) - (2)(5)\right] + 0 + 0 = 0$

$\Rightarrow (3-\lambda)\left[12 - 7\lambda + \lambda^2 - 10\right] = 0$

$= (\lambda - 3)(\lambda^2 - 7\lambda + 2) = 0$

$\lambda - 3 = 0$

$\boxed{\lambda = 3}$ — ①

$\lambda^2 - 7\lambda + 2 = 0$

$$\lambda = \frac{7 \pm \sqrt{(-7)^2 - 4(1)(2)}}{2}$$

$$\lambda = \frac{7 \pm \sqrt{41}}{2}$$

Thus Eigen values are : $3, \frac{7+\sqrt{41}}{2}, \frac{7-\sqrt{41}}{2}$

PRACTICAL 2

Name:	Harsh Shah	**Semester:**	VII	**Division:**	6
Roll No.:	21BCP359	**Date:**	30-07-24	**Batch:**	G11
Aim:	Extracting Region features and Boundary features from Images				

Program

```
import requests
from PIL import Image
import numpy as np
import cv2
from io import BytesIO

# List of image URLs
image_urls = [
    "https://images.pexels.com/photos/56866/garden-rose-red-pink-56866.jpeg",
    "https://cdn.pixabay.com/photo/2015/10/09/00/55/lotus-978659_640.jpg",
    "https://s28151.pcdn.co/wp-content/uploads/sites/2/2022/03/Coyote-animal-sentience-research.jpg",
    "https://i.natgeofe.com/k/9acd2bad-fb0e-43a8-935d-ec0aefc60c2f/monarch-butterfly-grass_3x2.jpg",
    "https://image.shutterstock.com/image-photo/green-leaves-philodendron-plant-nature-260nw-
2477697533.jpg"
]

# Download images
images = []
for url in image_urls:
    response = requests.get(url)
    img = Image.open(BytesIO(response.content))
    images.append(img)

# Resize images to 256x256 pixels
resized_images = [img.resize((256, 256)) for img in images]

# Convert images to grayscale
gray_images = [cv2.cvtColor(np.array(img), cv2.COLOR_RGB2GRAY) for img in resized_images]

# Extract boundary features using Canny edge detection
boundary_features = [cv2.Canny(img, 100, 200) for img in gray_images]

# Extract region features (using image moments)
region_features = [cv2.moments(img) for img in gray_images]

# Convert region features to a feature vector
feature_vectors = []
for moments in region_features:
    if moments["m00"] != 0:
        cx = int(moments["m10"] / moments["m00"])
        cy = int(moments["m01"] / moments["m00"])
```

```
    else:
        cx, cy = 0, 0
    feature_vectors.append([cx, cy])

# Display results
print("Boundary Features (Canny edges):")
for i, bf in enumerate(boundary_features):
    print(f"Image {i+1}:")
    print(bf)

print("\nRegion Features (Centroid coordinates):")
for i, fv in enumerate(feature_vectors):
    print(f"Image {i+1}: Centroid = {fv}")

print("\nFeature Vectore:")
print(feature_vectors)
```

Output:

```
Region Features (Centroid coordinates):
Image 1: Centroid = [117, 139]
Image 2: Centroid = [120, 124]
Image 3: Centroid = [129, 132]
Image 4: Centroid = [131, 122]
Image 5: Centroid = [130, 131]

Feature Vectore:
[[117, 139], [120, 124], [129, 132], [131, 122], [130, 131]]
```

PRACTICAL 3

Name:	Harsh Shah	**Semester:**	VII	**Division:**	6
Roll No.:	21BCP359	**Date:**	06-08-24	**Batch:**	G11
Aim:	Understanding Pre-Processing in Datasets.				

Question 1

Dataset: diabetes.csv

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler, Binarizer, StandardScaler

df = pd.read_csv('diabetes.csv')

# Dataset without label/class
df1 = df.drop(['Outcome'], axis=1)

# Scaling
min_max_scaler = MinMaxScaler(feature_range=(0,1))
scaled_features = min_max_scaler.fit_transform(df1)
scaled_df = pd.DataFrame(scaled_features, columns=df1.columns)
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	0.352941	0.743719	0.590164	0.353535	0.000000	0.500745	0.234415	0.483333
1	0.058824	0.427136	0.540984	0.292929	0.000000	0.396423	0.116567	0.166667
2	0.470588	0.919598	0.524590	0.000000	0.000000	0.347243	0.253629	0.183333
3	0.058824	0.447236	0.540984	0.232323	0.111111	0.418778	0.038002	0.000000
4	0.000000	0.688442	0.327869	0.353535	0.198582	0.642325	0.943638	0.200000
...	...	...	...	...	...	...	...	...
763	0.588235	0.507538	0.622951	0.484848	0.212766	0.490313	0.039710	0.700000
764	0.117647	0.613065	0.573770	0.272727	0.000000	0.548435	0.111870	0.100000
765	0.294118	0.608040	0.590164	0.232323	0.132388	0.390462	0.071307	0.150000
766	0.058824	0.633166	0.491803	0.000000	0.000000	0.448584	0.115713	0.433333
767	0.058824	0.467337	0.573770	0.313131	0.000000	0.453055	0.101196	0.033333

Figure 1: Scaled df

 # Binarization

binarizer = Binarizer(*threshold*=0.0)

binarized_data = binarizer.fit_transform(scaled_df)

binarized_df = pd.DataFrame(binarized_data, *columns*=scaled_df.columns)

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	1.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0
1	1.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0
2	1.0	1.0	1.0	0.0	0.0	1.0	1.0	1.0
3	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0
4	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0

Figure 2: Binarized df.head()

 # Standardization

scaler = StandardScaler()

standardized_data = scaler.fit_transform(binarized_df)

standardized_df = pd.DataFrame(standardized_data, *columns*=binarized_df.columns)

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	0.411035	0.080951	0.218515	0.647760	-1.026390	0.120545	0.036108	0.298934
1	0.411035	0.080951	0.218515	0.647760	-1.026390	0.120545	0.036108	0.298934
2	0.411035	0.080951	0.218515	-1.543781	-1.026390	0.120545	0.036108	0.298934
3	0.411035	0.080951	0.218515	0.647760	0.974289	0.120545	0.036108	-3.345217
4	-2.432883	0.080951	0.218515	0.647760	0.974289	0.120545	0.036108	0.298934

Figure 3: Standardized df.head()

Question 2

Dataset: spam.csv

import re

import nltk

import pandas as pd

from nltk.corpus import stopwords

nltk.download("stopwords")

df = pd.read_csv("spam.csv", *encoding*="latin-1")

	v1	v2
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

Figure 4: df.head()

```
# Remove Puntuation and Stopwords
def remove_punctuations(text):
    return re.sub(r"[^\w\s]", "", text)


def remove_stopwords(text):
    stop_words = set(stopwords.words("english"))
    return " ".join([word for word in text.split() if word.lower() not in stop_words])


df["v2"] = df["v2"].apply(remove_punctuations)
df["v2"] = df["v2"].apply(remove_stopwords)
```

	v1	v2
0	ham	Go jurong point crazy Available bugis n great ...
1	ham	Ok lar Joking wif u oni
2	spam	Free entry 2 wkly comp win FA Cup final tkts 2...
3	ham	U dun say early hor U c already say
4	ham	Nah dont think goes usf lives around though

Figure 5: df.head()

PRACTICAL 4

Name:	Harsh Shah	**Semester:**	VII	**Division:**	6
Roll No.:	21BCP359	**Date:**	13-08-24	**Batch:**	G11
Aim:	Understanding Feature Extraction in Datasets.				

Question 1

Dataset: iris.csv

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler

from sklearn.decomposition PCA

df = pd.read_csv('./Iris.csv')

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

 # Splitting Features and Target

X = df.drop(['Species'], axis=1)

y = df['Species']

```
X.head()
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
0	1	5.1	3.5	1.4	0.2
1	2	4.9	3.0	1.4	0.2
2	3	4.7	3.2	1.3	0.2
3	4	4.6	3.1	1.5	0.2
4	5	5.0	3.6	1.4	0.2

```
y.head()
```

```
0    Iris-setosa
1    Iris-setosa
2    Iris-setosa
3    Iris-setosa
4    Iris-setosa
Name: Species, dtype: object
```

 # Standard Scaler

scaler = StandardScaler()

X_standardized = scaler.fit_transform(X)

X_standardized_df = pd.DataFrame(X_standardized, columns=X.columns)

```
X_standardized_df.head()
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
0	-1.720542	-0.900681	1.032057	-1.341272	-1.312977
1	-1.697448	-1.143017	-0.124958	-1.341272	-1.312977
2	-1.674353	-1.385353	0.337848	-1.398138	-1.312977
3	-1.651258	-1.506521	0.106445	-1.284407	-1.312977
4	-1.628164	-1.021849	1.263460	-1.341272	-1.312977

```
X_standardized_df.describe()
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	1.500000e+02	1.500000e+02	1.500000e+02	1.500000e+02
mean	0.000000	-4.736952e-16	-6.631732e-16	3.315866e-16	-2.842171e-16
std	1.003350	1.003350e+00	1.003350e+00	1.003350e+00	1.003350e+00
min	-1.720542	-1.870024e+00	-2.438987e+00	-1.568735e+00	-1.444450e+00
25%	-0.860271	-9.006812e-01	-5.877635e-01	-1.227541e+00	-1.181504e+00
50%	0.000000	-5.250608e-02	-1.249576e-01	3.362659e-01	1.332259e-01
75%	0.860271	6.745011e-01	5.692513e-01	7.627586e-01	7.905908e-01
max	1.720542	2.492019e+00	3.114684e+00	1.786341e+00	1.710902e+00

 # Principle Component Analysis

pca = PCA(n_components=2)

principal_components = pca.fit_transform(X_standardized)

principal_df = pd.DataFrame(principal_components, columns=['PC1', 'PC2'])

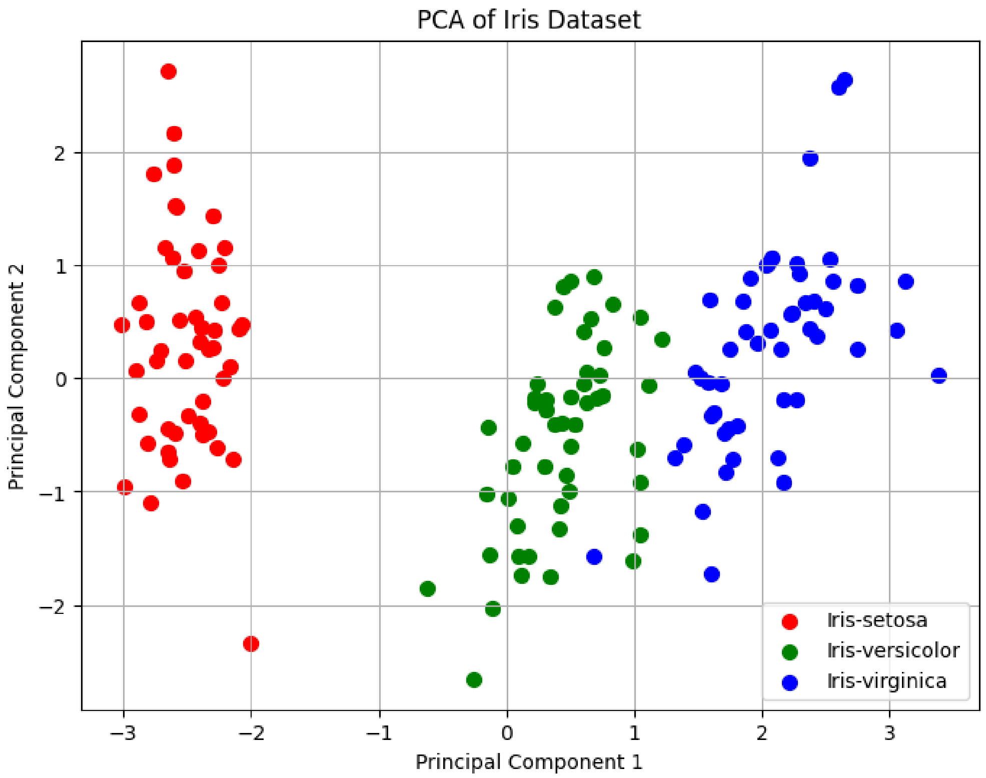
final_df = pd.concat([principal_df, y], axis=1)

```
final_df.head()
```

	PC1	PC2	Species
0	-2.816339	0.506051	Iris-setosa
1	-2.645527	-0.651799	Iris-setosa
2	-2.879481	-0.321036	Iris-setosa
3	-2.810934	-0.577363	Iris-setosa
4	-2.879884	0.670468	Iris-setosa

 # Plot

```
plt.figure(figsize=(8, 6))
colors = ['red', 'green', 'blue']
species_names = ['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']
for species, color in zip(species_names, colors):
    indices_to_keep = final_df['Species'] == species
    plt.scatter(final_df.loc[indices_to_keep, 'PC1'],
                final_df.loc[indices_to_keep, 'PC2'],
                c=color, s=50, label=species)

# Add labels and title
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('PCA of Iris Dataset')
plt.legend()
plt.grid()
```



Question 2

Dataset: wine.csv

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.decomposition import PCA

from sklearn.preprocessing import StandardScaler

df = pd.read_csv('./wine_data.csv')

```
df.head()
```

	class_label	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	color_intensity	hue	OD315_of_d
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64	1.04	
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	1.04	

X = df.drop(['class_label'], axis=1)

y = df['class_label']

 # Standardization

scaler = StandardScaler()

X_standardized = scaler.fit_transform(X)

X_standardized_df = pd.DataFrame(X_standardized, columns=X.columns)

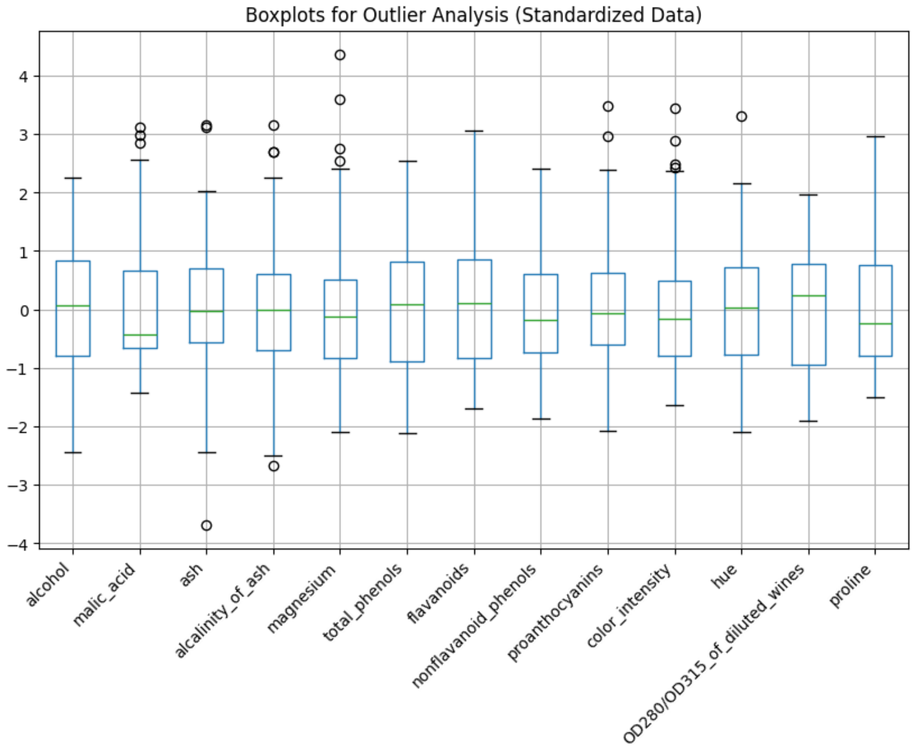
plt.figure(figsize=(10, 6))

X_standardized_df.boxplot()

plt.xticks(rotation=45, ha='right')

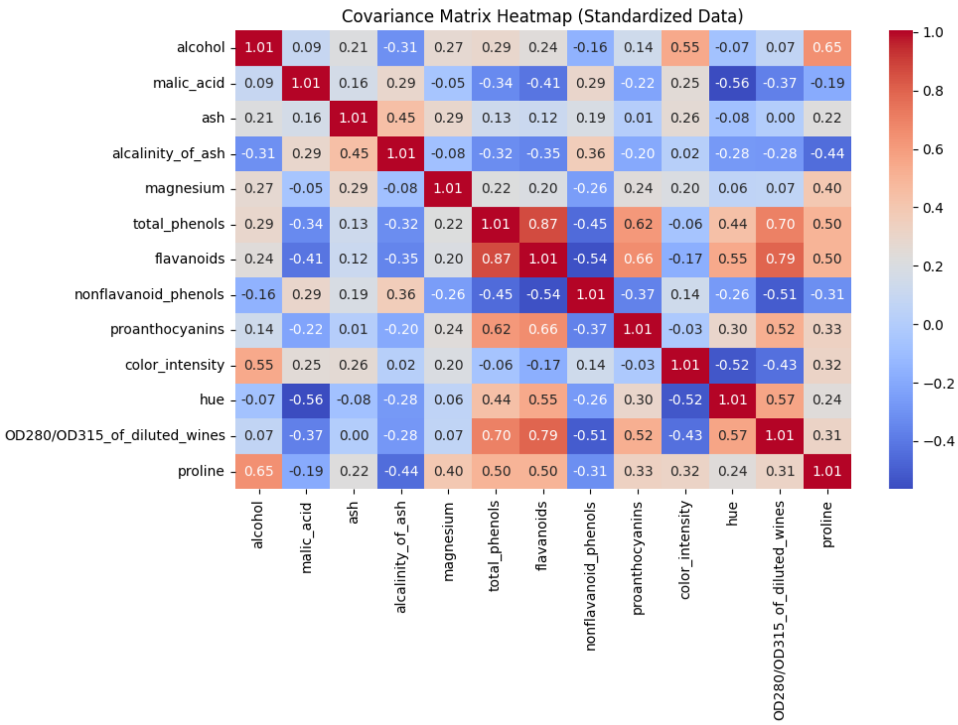
plt.title('Boxplots for Outlier Analysis (Standardized Data)')

plt.show()



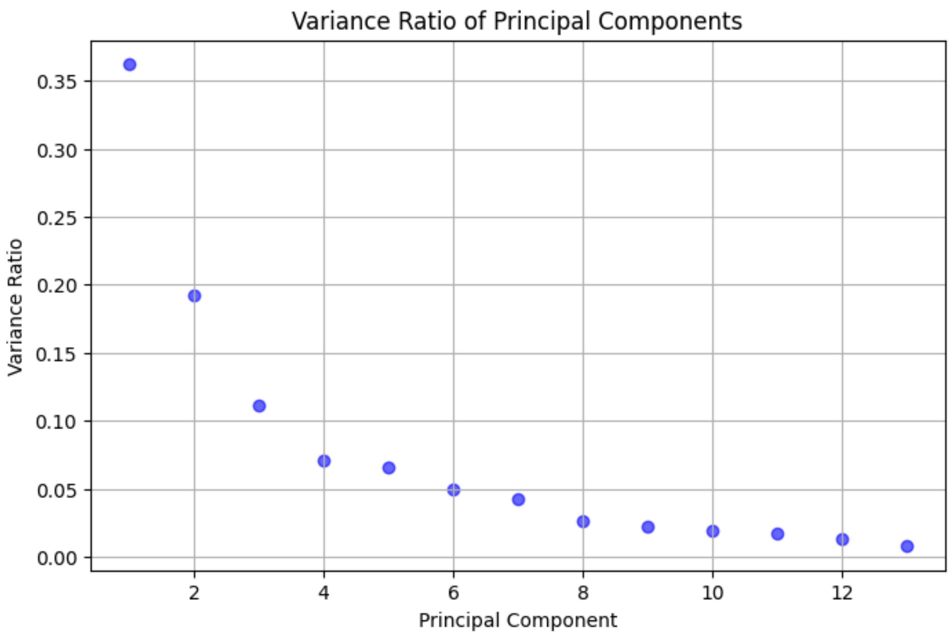
 # Covariance Matrix

```
cov_matrix_standardized = pd.DataFrame(X_standardized, columns=X.columns).cov()
plt.figure(figsize=(10, 6))
sns.heatmap(cov_matrix_standardized, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Covariance Matrix Heatmap (Standardized Data)')
plt.show()
```



 # PCA without specifying components

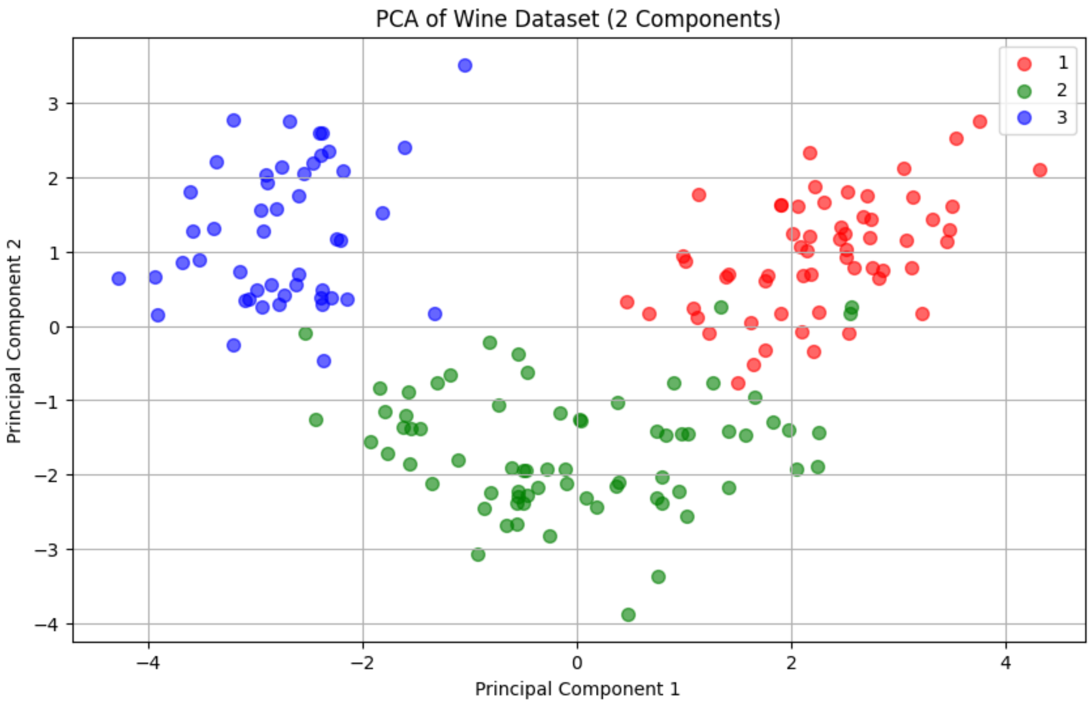
```
pca = PCA(n_components=None)
pca.fit(X_standardized)


plt.figure(figsize=(8, 5))
plt.scatter(range(1, len(pca.explained_variance_ratio_) + 1), pca.explained_variance_ratio_,
label='Variance Ratio', color='blue', alpha=0.6)
# plt.plot(range(1, len(pca.explained_variance_ratio_) + 1), pca.explained_variance_ratio_)
plt.xlabel('Principal Component')
plt.ylabel('Variance Ratio')
plt.title('Variance Ratio of Principal Components')
plt.grid()
plt.show()
```



 # PCA with 2 components

```
pca_2d = PCA(n_components=2)
principal_components = pca_2d.fit_transform(X_standardized)


principal_df = pd.DataFrame(data=principal_components, columns=['PC1', 'PC2'])
final_df = pd.concat([principal_df, y.reset_index(drop=True)], axis=1)
```

```
plt.figure(figsize=(10, 6))


colors = ['red', 'green', 'blue']
for label, color in zip(df['class_label'].unique(), colors):
    indices_to_keep = final_df['class_label'] == label
    plt.scatter(final_df.loc[indices_to_keep, 'PC1'],
                final_df.loc[indices_to_keep, 'PC2'],
                c=color, s=50, label=label, alpha=0.6)


plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('PCA of Wine Dataset (2 Components)')
plt.legend()
plt.grid()
plt.show()
```



PRACTICAL 5

Name:	Harsh Shah	**Semester:**	VII	**Division:**	6
Roll No.:	21BCP359	**Date:**	20-08-24	**Batch:**	G11
Aim:	Understanding Linear Discriminant projection in Datasets.				

Compute the Linear Discriminant projection for the following two dimensional dataset.

- **Samples for class ω1: X1= (x1, x2) = {(6, 4), (4, 5), (3, 4), (5, 7), (6, 6)}**
- **Sample for class ω2: X2= (x1, x2) = {(11, 12), (7, 9), (10, 7), (10, 9), (12, 10)}**

Linear Discriminant Projection

Linear Discriminant Projection (LDP) refers to the process of projecting data onto a lower-dimensional space in a way that maximizes the separation between different classes. It is a key part of **Linear Discriminant Analysis (LDA)**, a method used in statistics, pattern recognition, and machine learning for dimensionality reduction and classification.

Steps:

1. Define the samples for each class
2. Compute the mean vectors
3. Compute the within-class scatter matrix SW for both classes
4. Compute the between-class scatter matrix SB
5. Compute the eigenvalues and eigenvectors of SW^{-1} * SB
    a. First, compute the inverse of SW
    b. Then, compute the matrix SW^{-1} * SB
    c. Compute the eigenvalues and eigenvectors
    d. Find the eigenvector corresponding to the largest eigenvalue

Code

```
import numpy as np

X1 = np.array([[6, 4], [4, 5], [3, 4], [5, 7], [6, 6]])  # Class ω1
X2 = np.array([[11, 12], [7, 9], [10, 7], [10, 9], [12, 10]])  # Class ω2

# Step 1: Compute the mean vectors
mu1 = np.mean(X1, axis=0)
mu2 = np.mean(X2, axis=0)

# Step 2: Compute the within-class scatter matrices
S_W1 = np.dot((X1 - mu1).T, (X1 - mu1)) / (len(X1) - 1)
S_W2 = np.dot((X2 - mu2).T, (X2 - mu2)) / (len(X2) - 1)
S_W = S_W1 + S_W2

# Step 3: Compute the between-class scatter matrix
mu_diff = (mu2 - mu1).reshape(2, 1)
S_B = np.dot(mu_diff, mu_diff.T)
```

```
# Step 4: Compute the projection vector (eigenvector)
eigvals, eigvecs = np.linalg.eig(np.linalg.inv(S_W).dot(S_B))

# Sort eigenvectors by eigenvalues in descending order
eigvecs = eigvecs[:, np.argsort(-eigvals)]
w = eigvecs[:, 0]  # Projection vector (corresponding to the largest eigenvalue)

# Output the results
print("Mean vector of class ω1:", mu1)
print("Mean vector of class ω2:", mu2)
print("Within-class scatter matrix S_W:\n", S_W)
print("Between-class scatter matrix S_B:\n", S_B)
print("Projection vector w:", w)

# Project the samples onto the new axis
Y1 = np.dot(X1, w)
Y2 = np.dot(X2, w)

print("Projected samples for class ω1:", Y1)
print("Projected samples for class ω2:", Y2)
```

Output

```
Mean vector of class ω1: [4.8 5.2]
Mean vector of class ω2: [10.   9.4]
Within-class scatter matrix S_W:
 [[5.2 1.8]
 [1.8 5. ]]
Between-class scatter matrix S_B:
 [[27.04 21.84]
 [21.84 17.64]]
Projection vector w: [0.82816079 0.5604906 ]
```

```
Projected samples for class ω1: [7.21092712 6.11509614 4.72644475 8.06423812 8.33190831]
Projected samples for class ω2: [15.83565584 10.84154089 12.20504206 13.32602325 15.54283543]
```

PRACTICAL 6

Name:	Harsh Shah	**Semester:**	VII	**Division:**	6
Roll No.:	21BCP359	**Date:**	03-09-24	**Batch:**	G11
Aim:	Understanding Principal Component Analysis in Datasets.				

1. **Using sklearn library import the digits datasets (i.e. through load_digits()). It is a 8 x 8 pixel images dataset and they are 64-dimensional. In order to retrieve some of the intuition behind the relationship between these points, make use of PCA to reduce the dimension to a manageable number of dimensions (i.e., 2). After reducing the dimension plot them using scatter plots with cmap.**

2. **How many number of components will be ideal is an important part of using PCA. Using this data plot the cumulative explained variance ratio as a function of the number of components.**

3. **Try to reconstruct the data using the largest subset of principal components. The idea behind this is that any components with variance much larger than the effect of the noise should be relatively unaffected by the noise. Add some random noise to the dataset and replot it.**

Code

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

# Load the digits dataset
digits = load_digits()
X = digits.data  # Feature matrix
y = digits.target  # Labels

# Standardize the data (PCA works better on standardized data)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply PCA to reduce the dimensionality to 2 dimensions
pca_2d = PCA(n_components=2)
X_pca_2d = pca_2d.fit_transform(X_scaled)
```

```
# Scatter plot of the 2D PCA-reduced data
plt.figure(figsize=(8, 6))
scatter = plt.scatter(X_pca_2d[:, 0], X_pca_2d[:, 1], c=y, cmap='Spectral', edgecolor='k', s=60)
plt.colorbar(scatter)
plt.xlabel('First Principal Component')
plt.ylabel('Second Principal Component')
plt.title('2D PCA of Digits Dataset')
plt.show()

# Plot the cumulative explained variance as a function of the number of components
pca_full = PCA().fit(X_scaled)
plt.figure(figsize=(8, 6))
plt.plot(np.cumsum(pca_full.explained_variance_ratio_), marker='o')
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Explained Variance Ratio')
plt.title('Cumulative Explained Variance Ratio by PCA Components')
plt.grid(True)
plt.show()

# Reconstruct the data using the largest subset of principal components
n_components = 30  # Select the top 30 components
pca_reconstruct = PCA(n_components=n_components)
X_pca_reduced = pca_reconstruct.fit_transform(X_scaled)
X_reconstructed = pca_reconstruct.inverse_transform(X_pca_reduced)

# Add noise to the dataset
noise_factor = 0.5
X_noisy = X_scaled + noise_factor * np.random.normal(size=X_scaled.shape)

# Apply PCA again to the noisy data
X_pca_noisy = pca_2d.fit_transform(X_noisy)
```

 # Scatter plot of noisy PCA-reduced data

plt.figure(figsize=(8, 6))

scatter_noisy = plt.scatter(X_pca_noisy[:, 0], X_pca_noisy[:, 1], c=y, cmap='Spectral', edgecolor='k', s=60)

plt.colorbar(scatter_noisy)

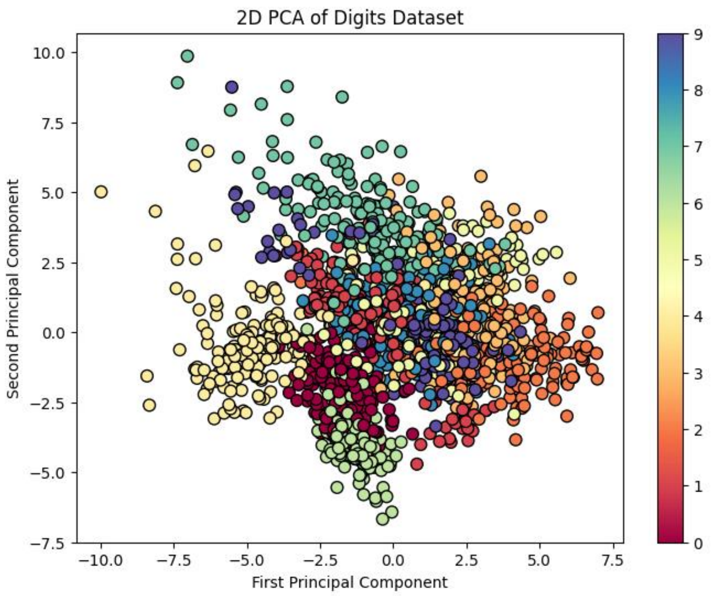
plt.xlabel('First Principal Component (Noisy)')

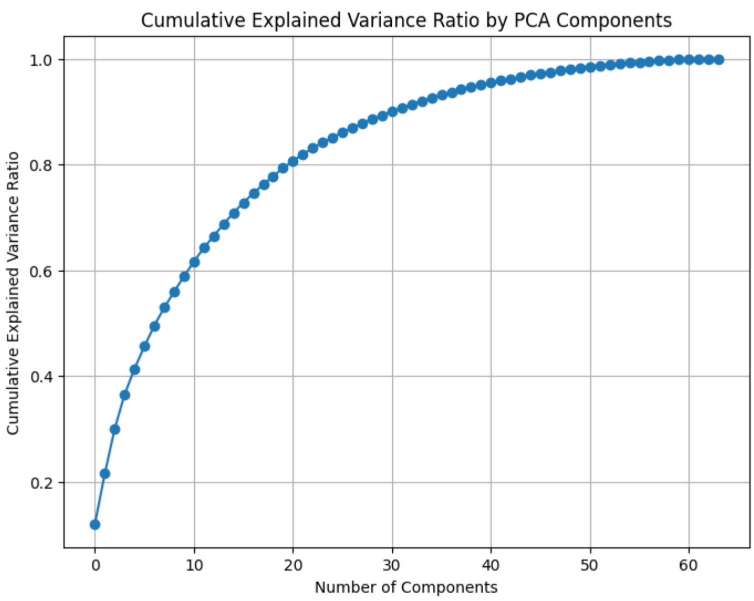
plt.ylabel('Second Principal Component (Noisy)')

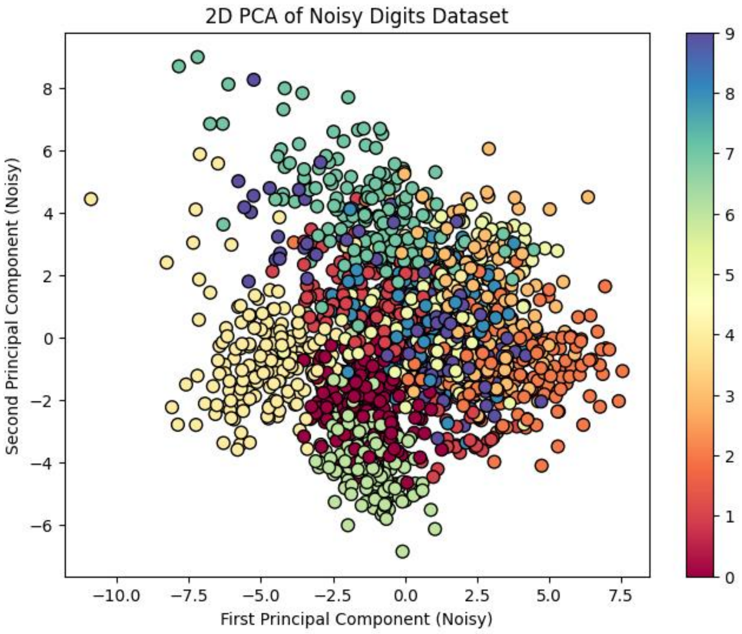
plt.title('2D PCA of Noisy Digits Dataset')

plt.show()

Output






2D PCA of Noisy Digits Dataset
Second Principal Component (Noisy)
First Principal Component (Noisy)
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
8
6
4
2
0
−2
−4
−6
9
8
7
6
5
4
3
2
1
0

PRACTICAL 7

Name:	Harsh Shah	**Semester:**	VII	**Division:**	6
Roll No.:	21BCP359	**Date:**	10-09-24	**Batch:**	G11
Aim:	Understanding Jaccard Similarity.				

Jaccard Similarity

Jaccard Similarity is a measure of similarity between two asymmetric binary vectors or we can say a way to find the similarity between two sets. It is a common proximity measurement used to compute the similarity of two items, such as two text documents. The index ranges from 0 to 1. Range closer to 1 means more similarity in two sets of data.

$$J(A,B) = \frac{| A \cap B |}{| A \cup B |}$$

Code

```
def jaccard_similarity(list1, list2):
    intersection = 0
    union = 0

    for a, b in zip(list1, list2):
        if a == 1 or b == 1:
            union += 1
        if a == 1 and b == 1:
            intersection += 1

    if union == 0:
        return 0
    return intersection / union


C1 = [0, 1, 0, 0, 0, 1, 0, 0, 1]
C2 = [0, 0, 1, 0, 0, 0, 0, 0, 1]
C3 = [1, 1, 0, 0, 0, 1, 0, 0, 0]

similarity_C1_C2 = jaccard_similarity(C1, C2)
similarity_C1_C3 = jaccard_similarity(C1, C3)
similarity_C2_C3 = jaccard_similarity(C2, C3)

print(f"Similarity - Customer C1 and C2 is {similarity_C1_C2}")
print(f"Similarity - Customer C1 and C3 is {similarity_C1_C3}")
print(f"Similarity - Customer C2 and C3 is {similarity_C2_C3}")


def jaccard_similarity_sets(set1, set2):
    intersection = len(set(set1).intersection(set2))
```

```
    union = len(set(set1).union(set2))
    return intersection / union

S1 = [0, 2, 5, 7, 9]
S2 = [0, 1, 2, 4, 5, 6, 8]

similarity_S1_S2 = jaccard_similarity_sets(S1, S2)

print(f"Similarity between Set S1 and S2 is {similarity_S1_S2}")
```

Output

```
Similarity - Customer C1 and C2 is 0.25
Similarity - Customer C1 and C3 is 0.5
Similarity - Customer C2 and C3 is 0.0
Similarity between Set S1 and S2 is 0.3333333333333333
```

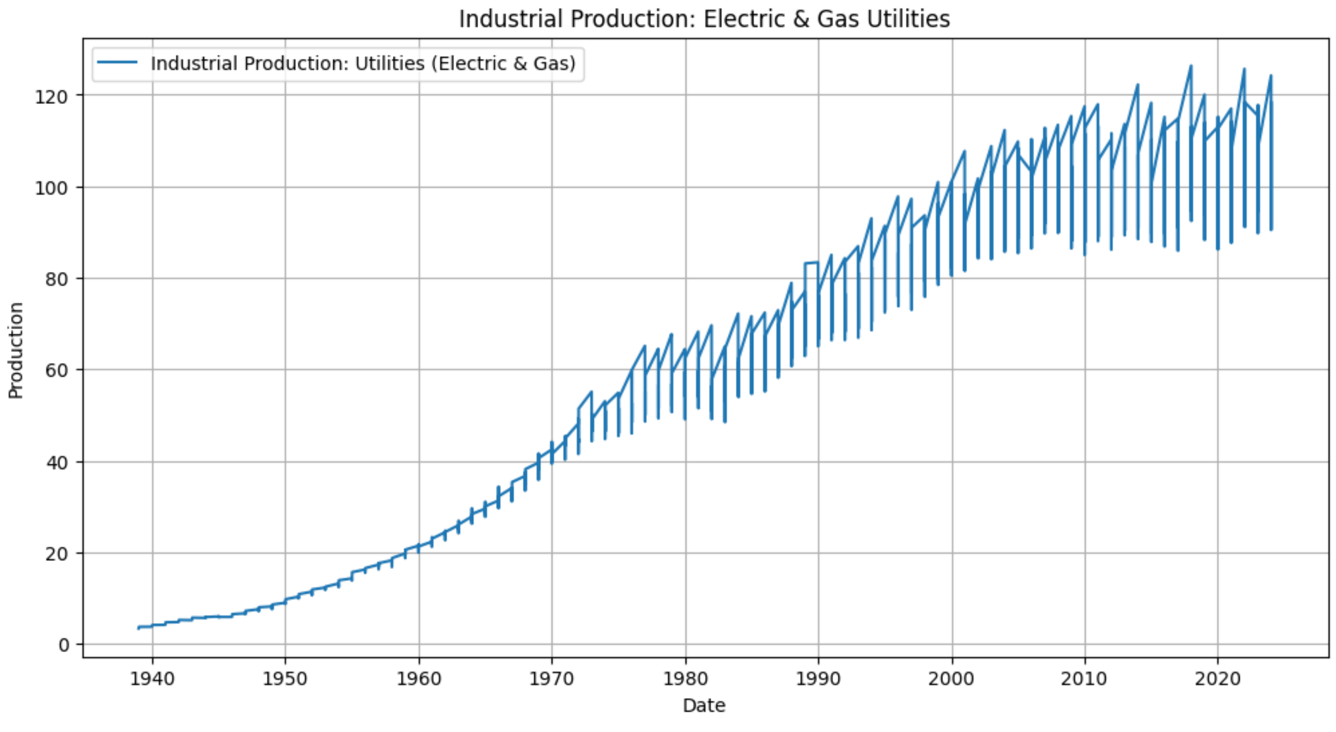
PRACTICAL 8

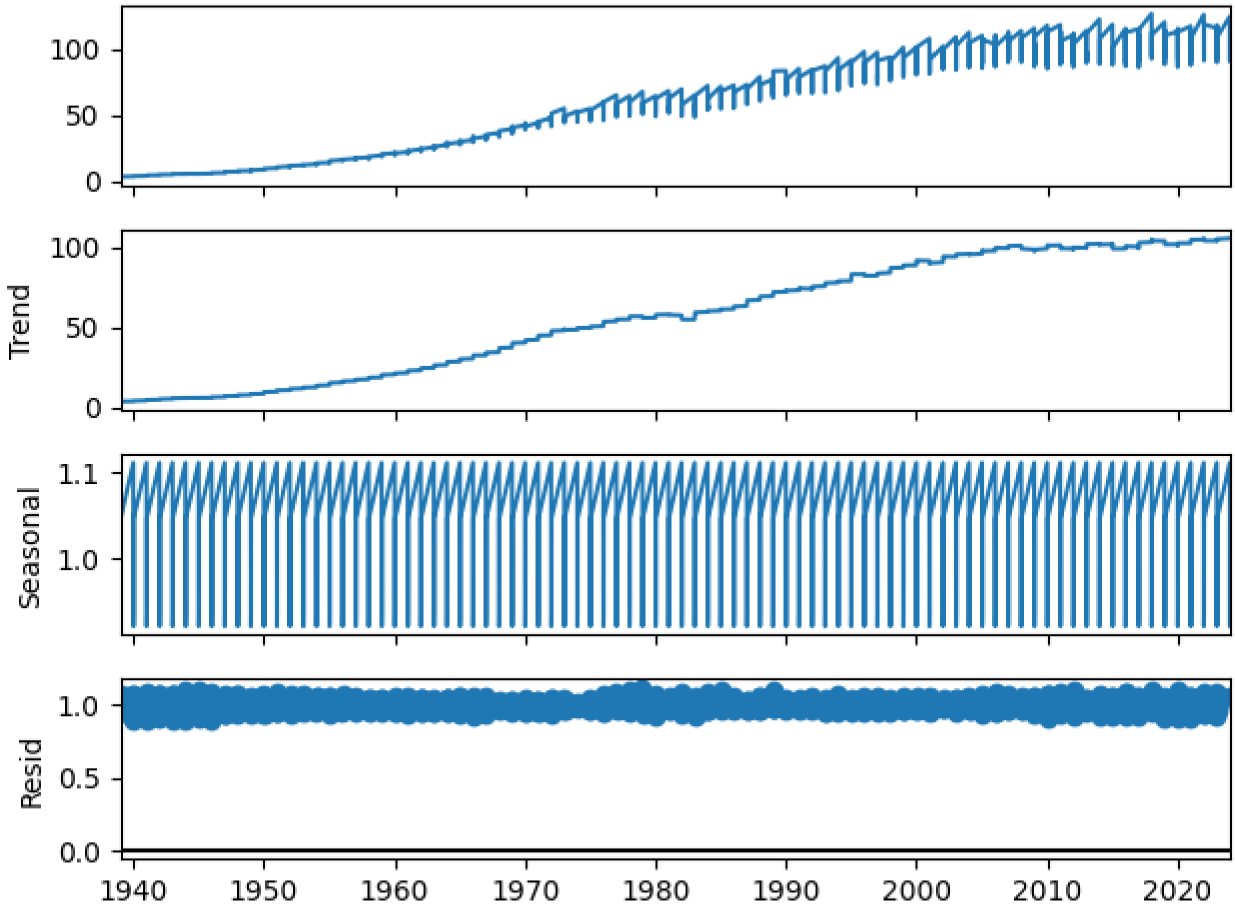
Name:	Harsh Shah	Semester:	VII	Division:	6
Roll No.:	21BCP359	Date:	10-09-24	Batch:	G11
Aim:	Feature Selection in Dataset.				

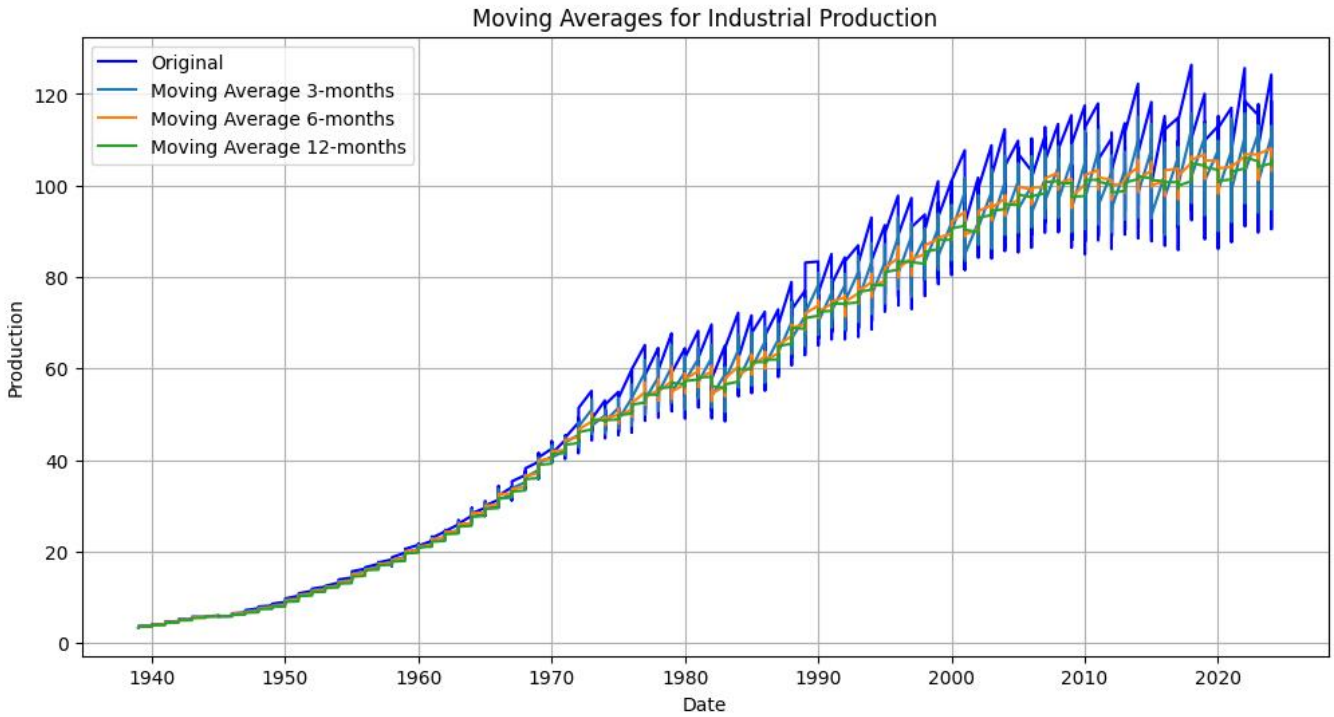
Code

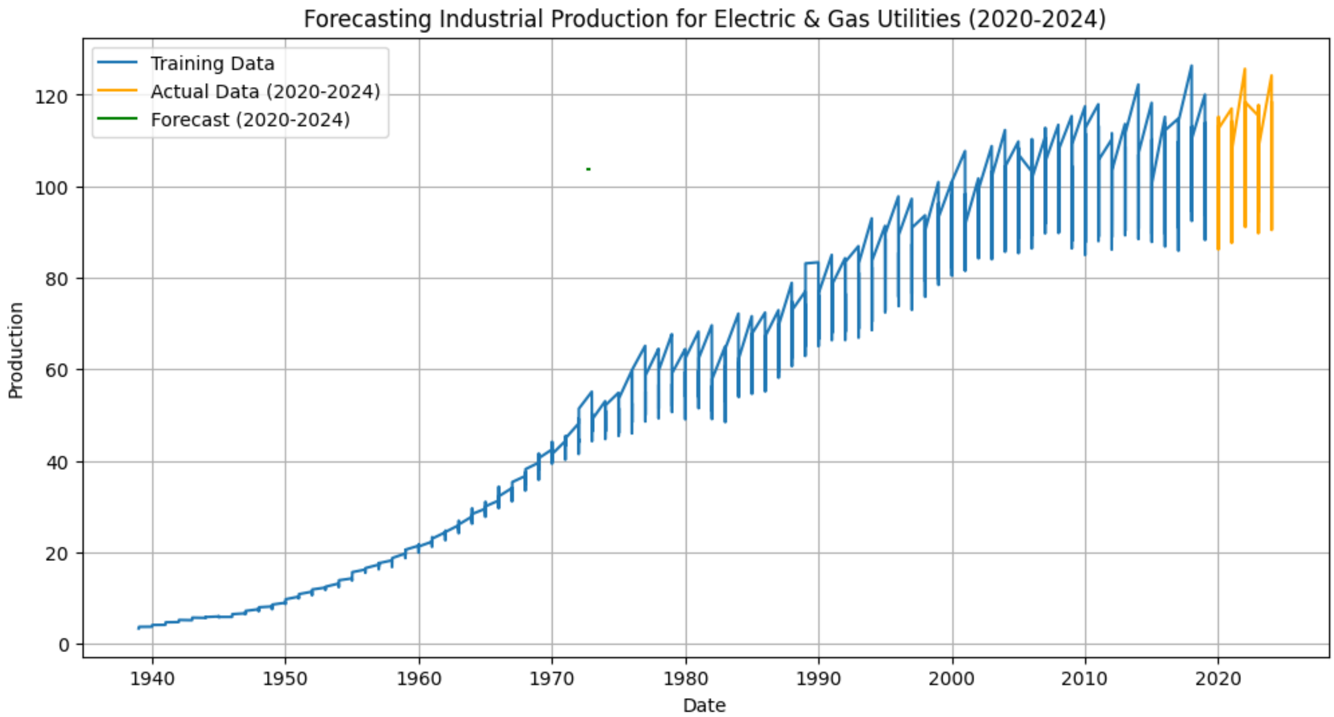
```
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.api import SimpleExpSmoothing
from sklearn.metrics import mean_squared_error

# Step 1: Load the data
data = pd.read_csv("IPG2211A2N.csv", index_col="DATE", parse_dates=True)

# Step 2: Plot the raw data
plt.figure(figsize=(12, 6))
plt.plot(data, label="Industrial Production: Utilities (Electric & Gas)")
plt.title("Industrial Production: Electric & Gas Utilities")
plt.xlabel("Date")
plt.ylabel("Production")
plt.legend()
plt.grid(True)
plt.show()

# Step 3: Trend and Seasonal Variation (Seasonal Decomposition)
decompose_result = seasonal_decompose(
    data, model="multiplicative", period=12
)  # Assuming monthly data
decompose_result.plot()
plt.show()

# Step 4: Moving Averages
def plot_moving_average(data, window_sizes):
    plt.figure(figsize=(12, 6))
    plt.plot(data, label="Original", color="blue")

    for window in window_sizes:
        data[f"MA_{window}"] = data["IPG2211A2N"].rolling(window=window).mean()
        plt.plot(data[f"MA_{window}"], label=f"Moving Average {window}-months")

    plt.title("Moving Averages for Industrial Production")
    plt.xlabel("Date")
    plt.ylabel("Production")
    plt.legend()
    plt.grid(True)
    plt.show()
```

```
# Moving averages for 3, 6, and 12 months
plot_moving_average(data.copy(), window_sizes=[3, 6, 12])

# Step 5: Time Series Forecasting
# Using Simple Exponential Smoothing to predict for 2020-2024

# Split data into training and testing
train = data[:"2019"]
test = data["2020":]

# Fit the model on training data
model = SimpleExpSmoothing(train).fit(smoothing_level=0.2, optimized=True)

# Forecast for 2020-2024
forecast = model.forecast(steps=len(test))

# Plot the forecasted data
plt.figure(figsize=(12, 6))
plt.plot(train, label="Training Data")
plt.plot(test, label="Actual Data (2020-2024)", color="orange")
plt.plot(forecast, label="Forecast (2020-2024)", color="green")
plt.title("Forecasting Industrial Production for Electric & Gas Utilities (2020-2024)")
plt.xlabel("Date")
plt.ylabel("Production")
plt.legend()
plt.grid(True)
plt.show()

# Step 6: Analysis
print(f"Mean Squared Error: {mean_squared_error(test, forecast)}")
```




Trend
Seasonal
Resid
1940
1950
1960
1970
1980
1990
2000
2010
2020


Moving Averages for Industrial Production
Original
Moving Average 3-months
Moving Average 6-months
Moving Average 12-months
Production
Date




```
Mean Squared Error: 107.6594220615686
```

PRACTICAL 9

Name:	Harsh Shah	**Semester:**	VII	**Division:**	6
Roll No.:	21BCP359	**Date:**	01-10-24	**Batch:**	G11

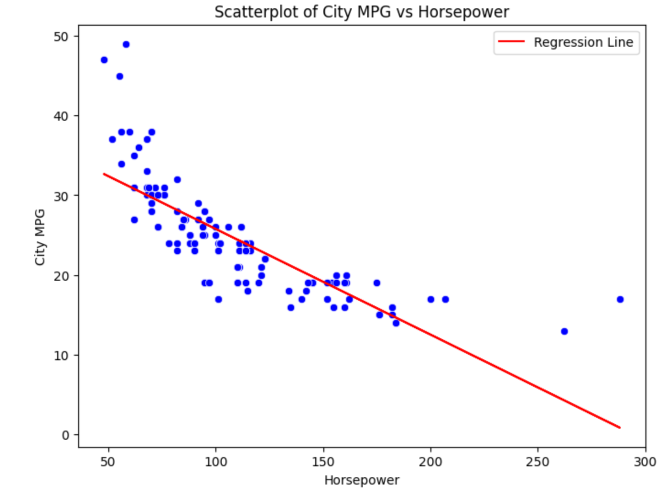
Code

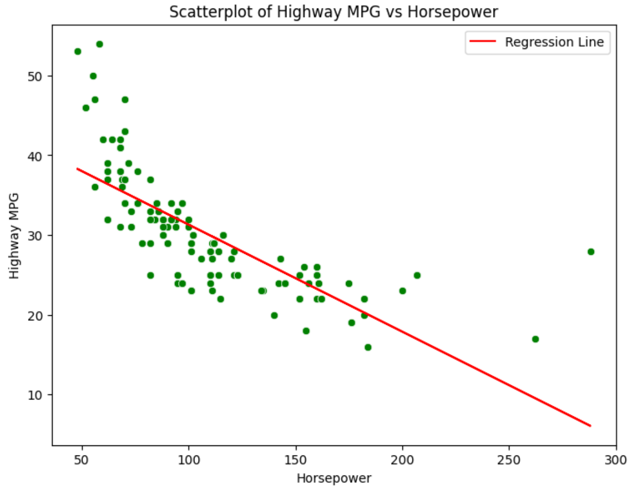
```
import pandas as pd
import statsmodels.api as sm

df = pd.read_csv('CarPrice_Assignment.csv')

X = df['horsepower']
y = df['citympg']

X = sm.add_constant(X)

# Create the regression model
model_citympg = sm.OLS(y, X).fit()

print(model_citympg.summary())

X_highway = df['horsepower']
y_highway = df['highwaympg']

X_highway = sm.add_constant(X_highway)

# Create the regression model
model_highway = sm.OLS(y_highway, X_highway).fit()

print(model_highway.summary())


import matplotlib.pyplot as plt
import seaborn as sns

# Scatterplot for citympg vs. horsepower
plt.figure(figsize=(8, 6))
sns.scatterplot(x=df['horsepower'], y=df['citympg'], color='blue')
plt.plot(df['horsepower'], model_citympg.predict(sm.add_constant(df['horsepower'])), color='red', label='Regression Line')
plt.title('Scatterplot of City MPG vs Horsepower')
plt.xlabel('Horsepower')
plt.ylabel('City MPG')
plt.legend()
plt.show()

# Scatterplot for highwaympg vs. horsepower
plt.figure(figsize=(8, 6))
sns.scatterplot(x=df['horsepower'], y=df['highwaympg'], color='green')
```

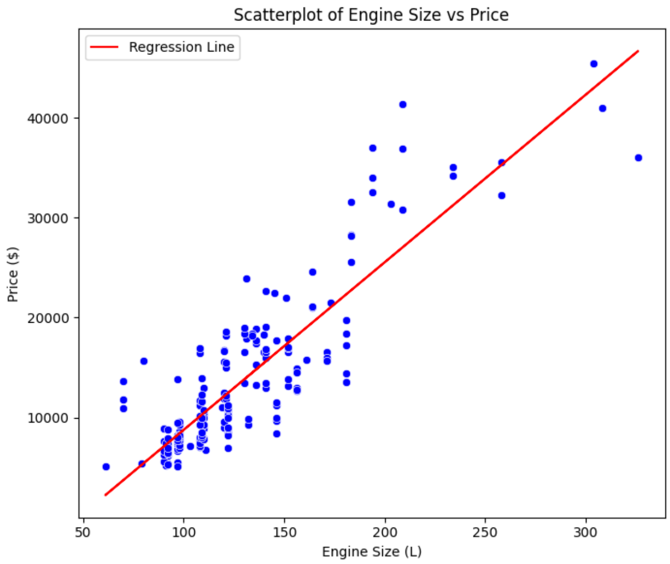
```
plt.plot(df['horsepower'], model_highway.predict(sm.add_constant(df['horsepower'])), color='red', label='Regression Line')
plt.title('Scatterplot of Highway MPG vs Horsepower')
plt.xlabel('Horsepower')
plt.ylabel('Highway MPG')
plt.legend()
plt.show()


# Regression Model 1: price as dependent and citympg as independent variable
X_citympg = sm.add_constant(df['citympg'])
model_price_citympg = sm.OLS(df['price'], X_citympg).fit()

# Display model statistics for price vs. citympg
print("Model 1: price vs citympg")
print(model_price_citympg.summary())

# Regression Model 2: price as dependent and highwaympg as independent variable
X_highwaympg = sm.add_constant(df['highwaympg'])  # Add constant term
model_price_highwaympg = sm.OLS(df['price'], X_highwaympg).fit()

# Display model statistics for price vs. highwaympg
print("\nModel 2: price vs highwaympg")
print(model_price_highwaympg.summary())



# Model 1: Regression of 'price' on 'enginesize'
X_enginesize = df[['enginesize']]
y_price = df['price']

# Add a constant (intercept)
X_enginesize = sm.add_constant(X_enginesize)

# Create and fit the regression model
model_enginesize = sm.OLS(y_price, X_enginesize).fit()

# Output the model summary
print(model_enginesize.summary())


# Model 2: Regression of 'price' on 'curbweight'
X_curbweight = df[['curbweight']]  # Independent variable
y_price = df['price']  # Dependent variable

# Add a constant (intercept)
X_curbweight = sm.add_constant(X_curbweight)

# Create and fit the regression model
model_curbweight = sm.OLS(y_price, X_curbweight).fit()

# Output the model summary
```

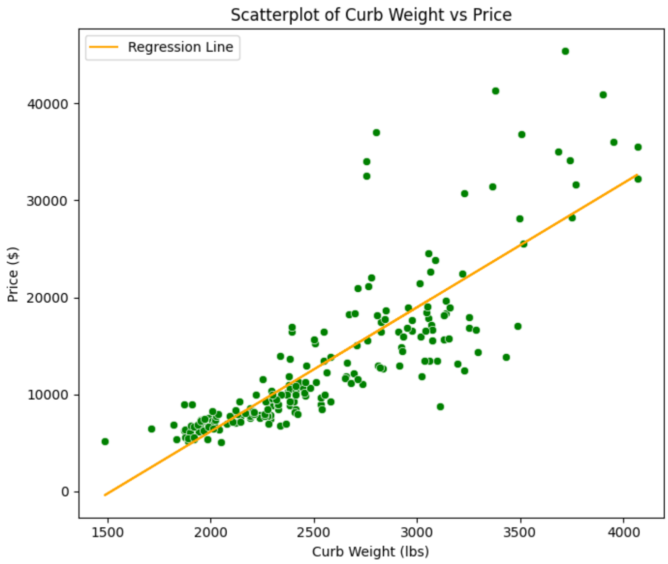
```
print(model_curbweight.summary())


# Scatterplot for Engine Size vs. Price
plt.figure(figsize=(14, 6))

plt.subplot(1, 2, 1)
sns.scatterplot(x=df['enginesize'], y=df['price'], color='blue')
plt.plot(df['enginesize'], model_enginesize.predict(sm.add_constant(df['enginesize'])), color='red', label='Regression Line')
plt.title('Scatterplot of Engine Size vs Price')
plt.xlabel('Engine Size (L)')
plt.ylabel('Price ($)')
plt.legend()

# Scatterplot for Curb Weight vs. Price
plt.subplot(1, 2, 2)
sns.scatterplot(x=df['curbweight'], y=df['price'], color='green')
plt.plot(df['curbweight'], model_curbweight.predict(sm.add_constant(df['curbweight'])), color='orange', label='Regression Line')
plt.title('Scatterplot of Curb Weight vs Price')
plt.xlabel('Curb Weight (lbs)')
plt.ylabel('Price ($)')
plt.legend()

# Show the plots
plt.tight_layout()
plt.show()


import pandas as pd
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor

df = pd.read_csv('CarPrice_Assignment.csv')

# Select numeric variables except 'citympg' and 'highwaympg'
numeric_df = df.drop(['price', 'citympg', 'highwaympg'], axis=1).select_dtypes(include=[float, int])

independent_vars = sm.add_constant(numeric_df)

# Variance Inflation Factor (VIF) calculation
vif_data = pd.DataFrame()
vif_data['Feature'] = independent_vars.columns
vif_data['VIF'] = [variance_inflation_factor(independent_vars.values, i) for i in range(independent_vars.shape[1])]

print(vif_data)
```

Output


Scatterplot of City MPG vs Horsepower
Regression Line
City MPG
0
10
20
30
40
50
50
100
150
200
250
300
Horsepower


Scatterplot of Highway MPG vs Horsepower
Regression Line
Highway MPG
10
20
30
40
50
50
100
150
200
250
300
Horsepower


Scatterplot of Engine Size vs Price
Regression Line
Price ($)
10000
20000
30000
40000
50
100
150
200
250
300
Engine Size (L)


Scatterplot of Curb Weight vs Price
Regression Line
Price ($)
0
10000
20000
30000
40000
1500
2000
2500
3000
3500
4000
Curb Weight (lbs)

PRACTICAL 10

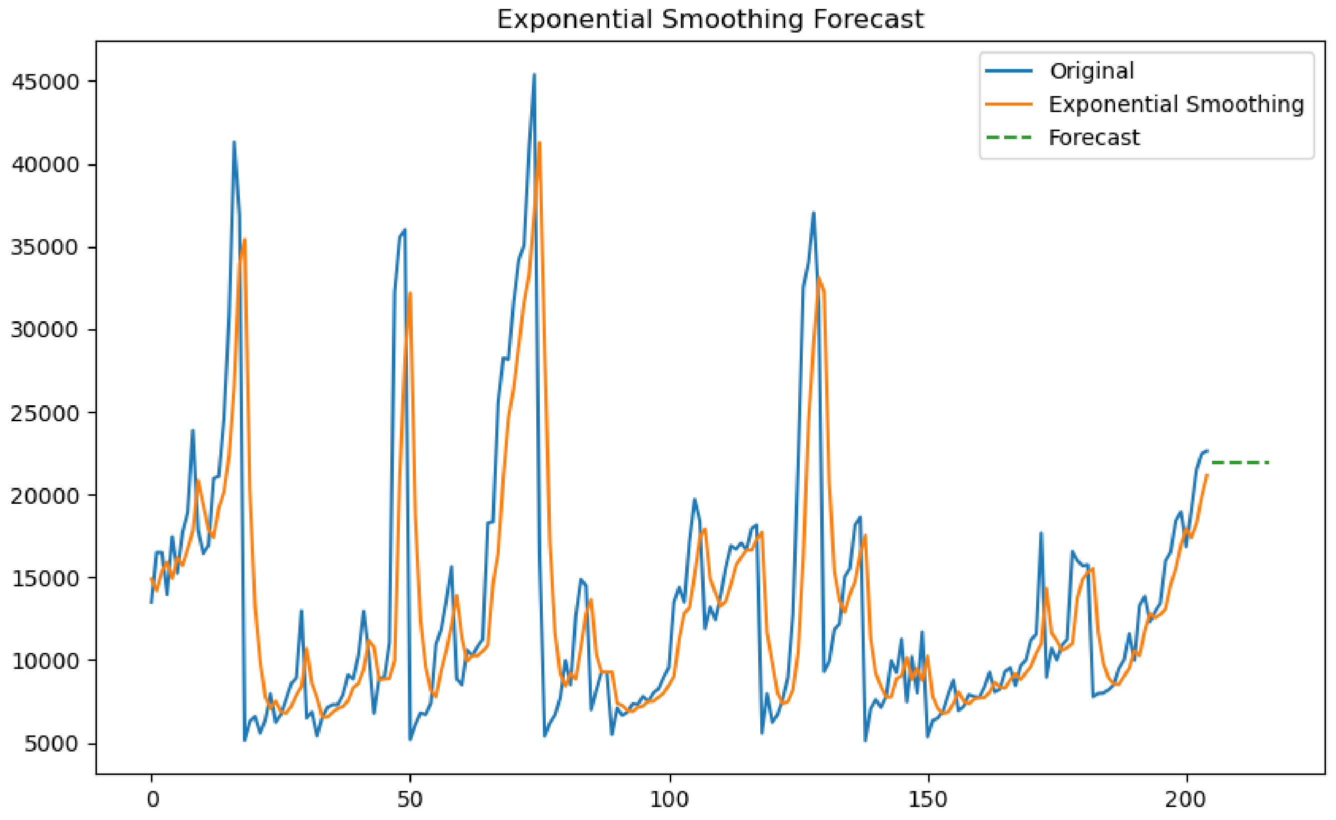
Name:	Harsh Shah	**Semester:**	VII	**Division:**	6
Roll No.:	21BCP359	**Date:**	08-10-24	**Batch:**	G11

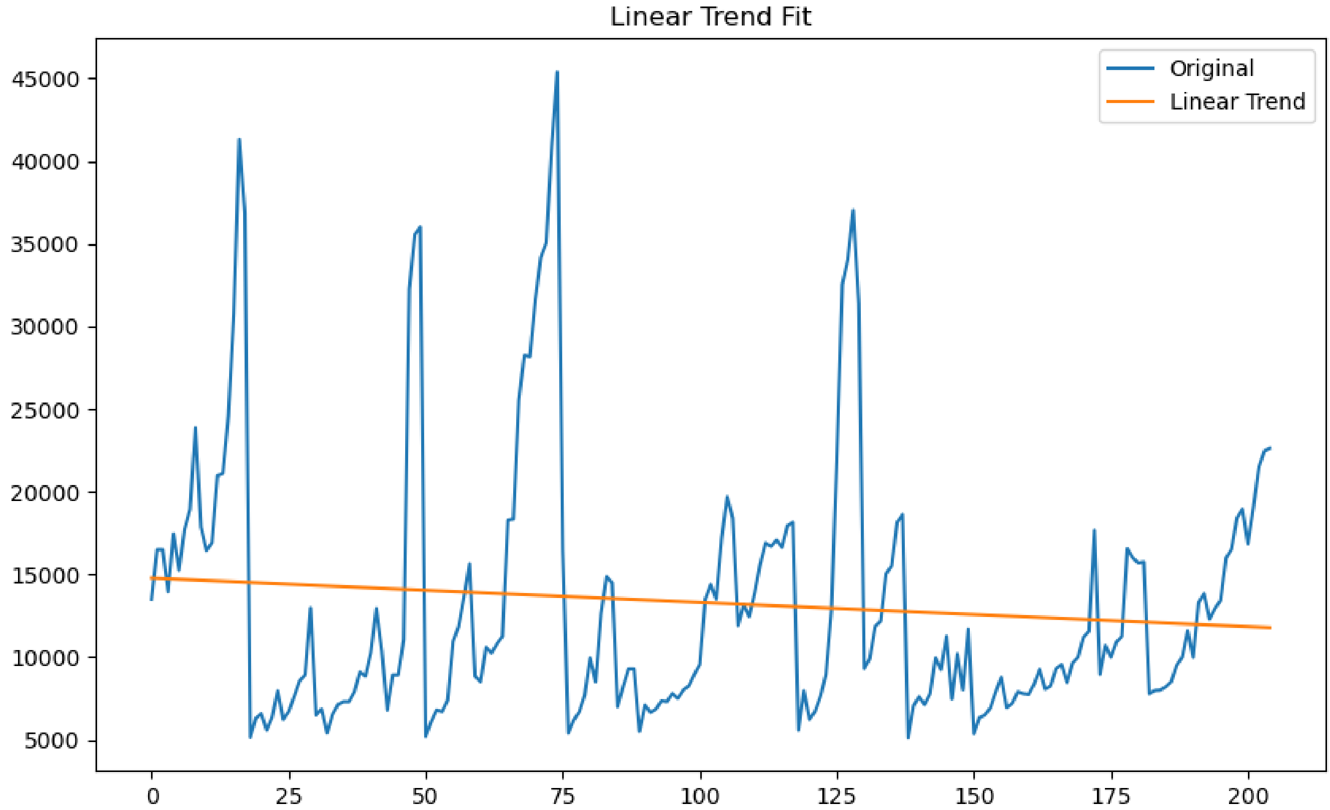
Code

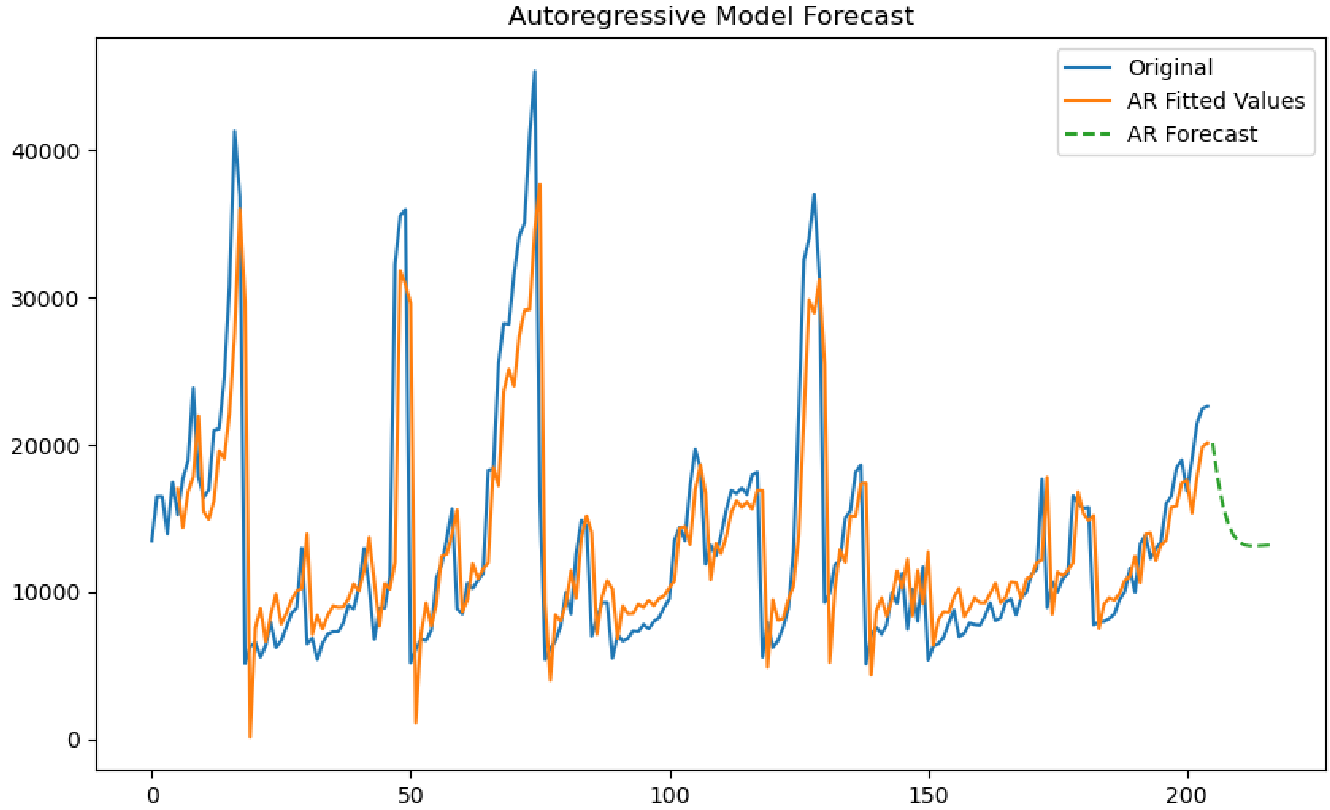
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.holtwinters import ExponentialSmoothing
from statsmodels.tsa.ar_model import AutoReg
from sklearn.linear_model import LinearRegression

data = pd.read_csv('CarPrice_Assignment.csv')

# Assume data is sequential (e.g., monthly observations)
data['time_index'] = np.arange(len(data))

# Use the 'price' column as the time series target
data['price'] = pd.to_numeric(data['price'], errors='coerce')
data.dropna(subset=['price'], inplace=True)

# Exponential Smoothing Model
exp_model = ExponentialSmoothing(data['price'], seasonal=None, trend=None, damped_trend=False).fit(smoothing_level=0.5)

# Predict future values
exp_forecast = exp_model.forecast(steps=12)

# Plotting
plt.figure(figsize=(10, 6))
plt.plot(data['time_index'], data['price'], label='Original')
plt.plot(data['time_index'], exp_model.fittedvalues, label='Exponential Smoothing')
plt.plot(range(len(data), len(data) + 12), exp_forecast, label='Forecast', linestyle='--')
plt.legend()
plt.title('Exponential Smoothing Forecast')
plt.show()

# Linear Trend Model
# Fit a linear regression model
linear_model = LinearRegression()
linear_model.fit(data[['time_index']], data['price'])

# Predict values using the model
data['linear_trend'] = linear_model.predict(data[['time_index']])

# Plotting
```

```
plt.figure(figsize=(10, 6))
plt.plot(data['time_index'], data['price'], label='Original')
plt.plot(data['time_index'], data['linear_trend'], label='Linear Trend')
plt.legend()
plt.title('Linear Trend Fit')
plt.show()

# Autoregressive Model (AR)
# Fit the AR model with a specified lag
ar_model = AutoReg(data['price'], lags=5).fit()

# Predict future values using the AR model
ar_forecast = ar_model.predict(start=len(data), end=len(data) + 11)

# Plotting
plt.figure(figsize=(10, 6))
plt.plot(data['time_index'], data['price'], label='Original')
plt.plot(ar_model.fittedvalues.index, ar_model.fittedvalues, label='AR Fitted Values')
plt.plot(range(len(data), len(data) + 12), ar_forecast, label='AR Forecast', linestyle='--')
plt.legend()
plt.title('Autoregressive Model Forecast')
plt.show()
```

Output




Linear Trend Fit
Original
Linear Trend
45000
40000
35000
30000
25000
20000
15000
10000
5000
0
25
50
75
100
125
150
175
200


Autoregressive Model Forecast
Original
AR Fitted Values
AR Forecast
40000
30000
20000
10000
0
0
50
100
150
200