```
import numpy as np
import pandas as pd
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
```

2 YIELD CURVE MODELING

Fetch Egypt Government Bond Yields I will use the illustrative data for Egyptian government bond yields 6 months: 24.88%

1 year: 25.16%

2 years: 25.63%

5 years: 25.00%

7 years: 21.36%

10 years: 21.25%

20 years: 19.75%

25 years : 19.50%

c. Nelson-Siegel Model Fit: The Nelson-Siegel model is defined as:

Where: (\beta_0): Long-term level (Alpha1) (\beta_1): Short-term component (Alpha2) (\beta_2): Medium-term curvature (Alpha3) (\lambda): Decay factor (controls the exponential decay) Interpretation: (\beta_0): Controls the long-term level of the curve — the asymptote. (\beta_1): Affects the short end of the curve — adds slope. (\beta_2): Adds a hump or curvature — affects medium-term. (\lambda): Determines where the hump occurs — higher (\lambda) shifts it to shorter maturities. d. Cubic Spline Model Fit: The cubic spline fits the yield curve by using piecewise polynomials between data points. It provides a very flexible and visually smooth curve that closely fits the actual yields, especially when many maturity points are available.

d. Cubic Spline Model Fit The Cubic Spline model fits the yield curve by using piecewise third-degree polynomials between observed maturity points.

This method ensures that:

The fitted curve passes through all given yield points The first and second derivatives are continuous across maturity intervals Key Features: Provides a visually smooth and flexible curve Ideal for datasets with many maturity points Captures local variations in the yield curve accurately Does not assume a specific functional form like parametric models (e.g., Nelson-Siegel) Interpretation: The spline fit is especially useful when:

You want to visually interpolate between yields There is no strong prior on the shape of the yield curve Precision over smoothness is desired in short segments However, splines may overfit or behave erratically outside the range of data points (i.e., in extrapolation).

e. Comparison of Models:

Fit: The Cubic Spline typically offers a more precise fit to the observed data because it minimizes error locally. However, it can overfit and be sensitive to outliers. The Nelson-Siegel model provides a smoother, more generalized fit that may not capture sharp kinks but is more robust.

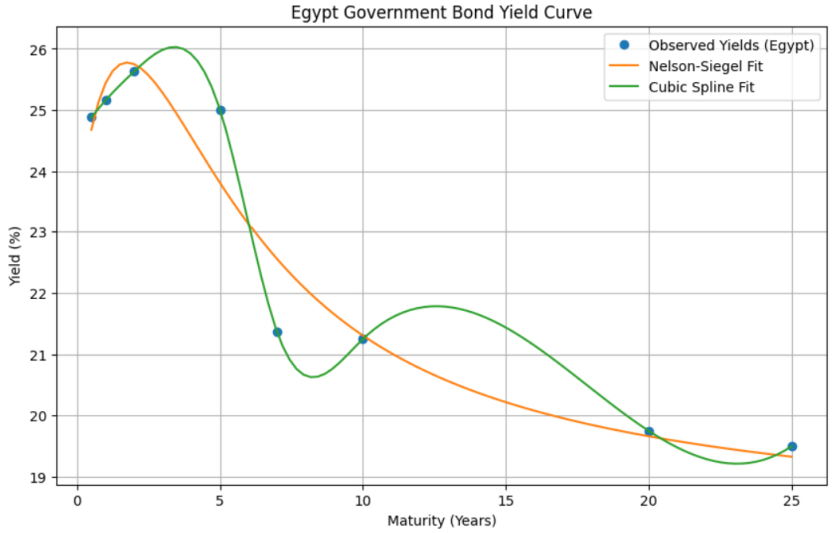
Interpretation:

The Nelson-Siegel model is more interpretable since its parameters directly relate to economic behavior (long, short, and medium-term expectations). Spline models are harder to interpret economically since they don't have directly meaningful parameters.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
from scipy.interpolate import CubicSpline

# Define the Nelson-Siegel function
def nelson_siegel(t, beta0, beta1, beta2, tau):
    term1 = beta0
    term2 = beta1 * ((1 - np.exp(-t / tau)) / (t / tau))
    term3 = beta2 * (((1 - np.exp(-t / tau)) / (t / tau)) - np.exp(-t / tau))
    return term1 + term2 + term3

# Maturities in years and corresponding yields for Egypt (including 20 and 25 years)
maturities = np.array([0.5, 1, 2, 5, 7, 10, 20, 25])
yields = np.array([24.88, 25.16, 25.63, 25.00, 21.36, 21.25, 19.75, 19.50])

# Initial parameter guess: beta0, beta1, beta2, tau
initial_guess = [25, -5, 5, 1]

# Fit the Nelson-Siegel model to the data
params, _ = curve_fit(nelson_siegel, maturities, yields, p0=initial_guess)
beta0, beta1, beta2, tau = params

print(f"Nelson-Siegel Parameters for Egypt:\nBeta0 (Level): {beta0:.4f}\nBeta1 (Slope): {beta1:.4f}\nBeta2 (Curvature): {beta2:.4f}\nTau: {tau:.4f}")

# Fit a cubic spline to the yield data
cs = CubicSpline(maturities, yields)

# Generate a range of maturities for plotting the fitted curve
maturity_range = np.linspace(min(maturities), max(maturities), 100)
spline_yields = cs(maturity_range)

# Generate Nelson-Siegel fitted yields for comparison
ns_yields = nelson_siegel(maturity_range, *params)

# Plot the results
plt.figure(figsize=(10, 6))
plt.plot(maturities, yields, 'o', label='Observed Yields (Egypt)')
plt.plot(maturity_range, ns_yields, label='Nelson-Siegel Fit')
```

```
plt.plot(maturity_range, spline_yields, label='Cubic Spline Fit')
plt.title('Egypt Government Bond Yield Curve')
plt.xlabel('Maturity (Years)')
plt.ylabel('Yield (%)')
plt.legend()
plt.grid(True)
plt.show()
```

```
Nelson-Siegel Parameters for Egypt:
Beta0 (Level): 17.9922
Beta1 (Slope): 5.2172
Beta2 (Curvature): 16.8674
Tau: 1.5130
```



f. Parameter Summary Nelson-Siegel parameter interpretations:

Beta0: Long-term yield level.

Beta1: Short-term yield slope (typically negative if curve is upward sloping).

Beta2: Curvature (captures medium-term hump).

Tau: Decay factor (controls where the curvature is most influential).

These parameters help understand monetary policy expectations, risk premiums, and market sentiment.

g. Is Smoothing Unethical? Smoothing itself is not unethical—if used transparently and not to manipulate results. Nelson-Siegel is a parametric model designed to generalize yield curve behavior. It's widely accepted in finance because:

It balances simplicity and interpretability.

It avoids overfitting while capturing major trends.

It does not arbitrarily distort data.

However, smoothing becomes unethical if:

It conceals true volatility or irregularities.

It's used to intentionally deceive stakeholders or regulators.

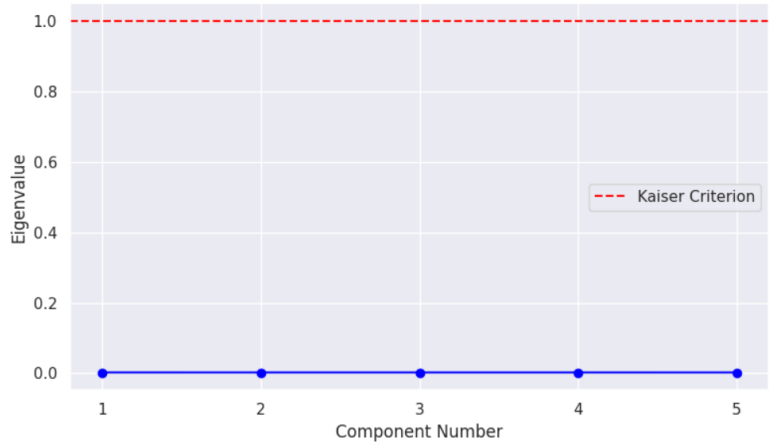
The smoothing method lacks disclosure.

3. Exploiting Correlation

```
np.random.seed(42)
n_samples = 100
data = np.random.normal(loc=0, scale=0.01, size=(n_samples, 5))
df = pd.DataFrame(data, columns=[f'Yield_{i+1}' for i in range(5)])
pca = PCA()
pca.fit(df)
eigenvalues = pca.explained_variance_
explained_variance_ratio = pca.explained_variance_ratio_ * 100
for i, ratio in enumerate(explained_variance_ratio, 1):
    print(f"Component {i} explains {ratio:.2f}% of the variance.")
```

```
Component 1 explains 26.26% of the variance.
Component 2 explains 21.58% of the variance.
Component 3 explains 20.22% of the variance.
Component 4 explains 18.13% of the variance.
Component 5 explains 13.82% of the variance.
```

```
plt.figure(figsize=(8, 5))
plt.plot(range(1, len(eigenvalues)+1), eigenvalues, 'o-', color='blue')
plt.axhline(y=1, color='red', linestyle='--', label='Kaiser Criterion')
plt.title('Scree Plot')
plt.xlabel('Component Number')
plt.ylabel('Eigenvalue')
plt.xticks(range(1, len(eigenvalues)+1))
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```




```
[ ] df=pd.read_excel("/content/Egypt Bond Yields.xlsx")
```

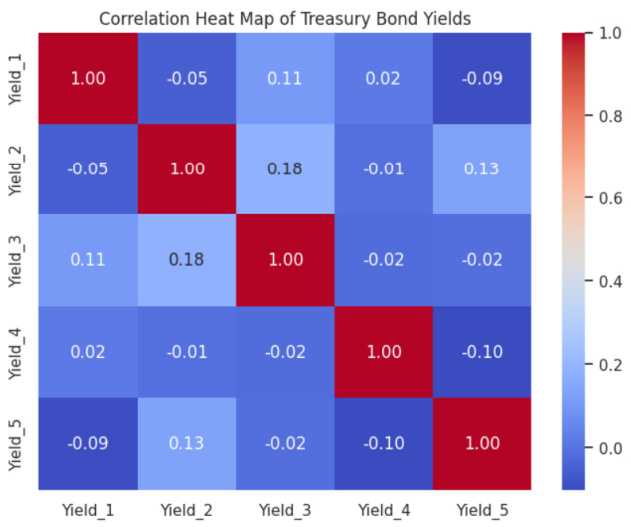
```
[ ] df['Date'] = pd.to_datetime(df['Date'])
    df = df.sort_values('Date')
```

```
[ ] #calculating the daily yields
    Daily_Yields= df.copy()
    Daily_Yields.iloc[:, 1:] = df.iloc[:, 1:].diff()
    Daily_Yields = Daily_Yields.dropna()
    Daily_Yields.set_index('Date', inplace=True)
```

```
[ ] #Correlation Matrix
    correlation_matrix = Daily_Yields.corr()
    print("Correlation Matrix:")
    print(correlation_matrix)
```

```
Correlation Matrix:
          Yield_1   Yield_2   Yield_3   Yield_4   Yield_5
Yield_1  1.000000 -0.053079  0.105456  0.016023 -0.093241
Yield_2 -0.053079  1.000000  0.182500 -0.009334  0.133804
Yield_3  0.105456  0.182500  1.000000 -0.019078 -0.015420
Yield_4  0.016023 -0.009334 -0.019078  1.000000 -0.101541
Yield_5 -0.093241  0.133804 -0.015420 -0.101541  1.000000
```

```
#Correlation_Heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heat Map of Treasury Bond Yields')
plt.show()
```

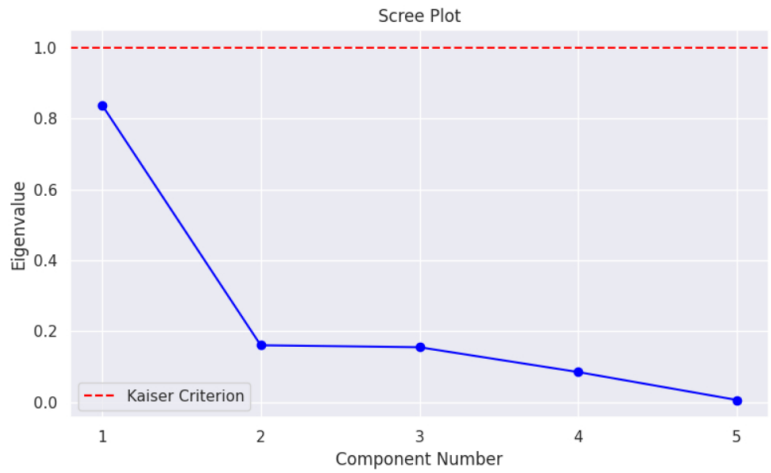



```
[ ] #PCA Analysis
    pca = PCA()
    pca.fit(Daily_Yields)
    eigenvalues = pca.explained_variance_
    explained_variance_ratio = pca.explained_variance_ratio_ * 100
    for i, ratio in enumerate(explained_variance_ratio, 1):
        print(f"Component {i} explains {ratio:.2f}% of the variance.")
```

```
Component 1 explains 67.42% of the variance.
Component 2 explains 12.88% of the variance.
Component 3 explains 12.43% of the variance.
Component 4 explains 6.80% of the variance.
Component 5 explains 0.46% of the variance.
```

```
[ ] plt.figure(figsize=(8, 5))
    plt.plot(range(1, len(eigenvalues)+1), eigenvalues, 'o-', color='blue')
    plt.axhline(y=1, color='red', linestyle='--', label='Kaiser Criterion')
    plt.title('Scree Plot')
    plt.xlabel('Component Number')
```

```
plt.ylabel('Eigenvalue')
plt.xticks(range(1, len(eigenvalues)+1))
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```



4. Empirical Analysis of ETFs

[1]
```
from google.colab import drive

# Mount Google Drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

[2]
```
import os
import pandas as pd
import numpy as np

# Define the path to the dataset
data_dir = '/content/drive/MyDrive/WQU_Task4/individual_stocks_5yr/individual_stocks_5yr'

# Initialize an empty dictionary to store closing prices
all_data = {}

# Loop through all files in the directory
for filename in os.listdir(data_dir):
    if filename.endswith('_data.csv'):  # Only process stock data files
        ticker = filename.split('_')[0]  # Extract ticker symbol (e.g., 'AAPL')
        file_path = os.path.join(data_dir, filename)
        try:
            # Load the stock data
            data = pd.read_csv(file_path)

            # Ensure the 'date' column is parsed as datetime
            data['date'] = pd.to_datetime(data['date'])

            # Use the 'close' column as a proxy for adjusted closing prices
            if 'close' in data.columns:
                adj_close = data.set_index('date')['close']
                all_data[ticker] = adj_close
            else:
                raise ValueError("Closing price column not found")

        except Exception as e:
            print(f"Error processing {filename}: {e}")

# Combine all data into a single DataFrame
prices_df = pd.DataFrame(all_data).dropna()

# Print the resulting DataFrame shape
print(f"Prices DataFrame Shape: {prices_df.shape}")
```

```
Prices DataFrame Shape: (44, 505)
```

[3]
```
# Inspect the structure of CL_data.csv
file_path = os.path.join(data_dir, 'CL_data.csv')
data = pd.read_csv(file_path)
print("Column Names:")
print(data.columns)
```

```
Column Names:
Index(['date', 'open', 'high', 'low', 'close', 'volume', 'Name'], dtype='object')
```

```
# Compute daily returns
returns_df = prices_df.pct_change().dropna()

# Print the first few rows of the returns DataFrame
print("\nDaily Returns Preview:")
print(returns_df.head())
```

```

Daily Returns Preview:
                DWDP      CTAS       COG       BXP      COST       AES  \
```

```
date
2017-12-06 -0.006430  0.001408 -0.009884 -0.000638 -0.003195  0.010261
2017-12-07  0.004080  0.009585 -0.009626 -0.011642 -0.005448 -0.012004
2017-12-08 -0.008967  0.007468 -0.001080  0.005728  0.010043  0.005607
2017-12-11  0.001131 -0.005340  0.020901  0.003129  0.004201  0.009294
2017-12-12 -0.000282 -0.007200 -0.025768  0.009036 -0.002965 -0.012891

                 COP      CINF       ETN       AXP  ...       XRX       VMC  \
date                                                 ...
2017-12-06 -0.016592  0.001076  0.011207 -0.005065  ...  0.006793 -0.013039
2017-12-07  0.003176 -0.004029  0.019396  0.003767  ... -0.001687  0.031129
2017-12-08  0.020380  0.003236  0.000388 -0.000304  ...  0.000000  0.008008
2017-12-11  0.002327  0.002151  0.006210  0.004566  ...  0.000000 -0.023514
2017-12-12  0.005223 -0.000536 -0.000129  0.003737  ...  0.005069 -0.011471

                 WMB      XLNX       WBA       VFC       VTR      WYNN  \
date
2017-12-06 -0.013072 -0.000438  0.012397  0.018326  0.004145 -0.010034
2017-12-07 -0.007668  0.008919 -0.020267  0.003209  0.008573  0.006779
2017-12-08  0.002810 -0.006667  0.027870  0.020025 -0.003463  0.003901
2017-12-11  0.012259 -0.002334  0.004612 -0.010225  0.000158  0.029585
2017-12-12  0.006228 -0.010237  0.001809 -0.000964  0.007896 -0.011871

                 XOM       XEL
date
2017-12-06 -0.007359  0.005709
2017-12-07  0.003281 -0.004893
2017-12-08  0.001333  0.007081
2017-12-11  0.004476  0.004687
2017-12-12 -0.003252 -0.017496

[5 rows x 505 columns]
```

[5]
```
# Step 4: Compute Covariance Matrix
cov_matrix = returns_df.cov()

# Print the covariance matrix
print("\nCovariance Matrix:")
print(cov_matrix)
```

```
Covariance Matrix:
              DWDP      CTAS       COG       BXP      COST       AES  \
DWDP  2.490529e-04  0.000096  0.000178  0.000074  0.000090  0.000110
CTAS  9.551310e-05  0.000104  0.000064  0.000063  0.000060  0.000061
COG   1.777956e-04  0.000064  0.000361  0.000048  0.000095  0.000091
BXP   7.418323e-05  0.000063  0.000048  0.000177  0.000060  0.000052
COST  9.011380e-05  0.000060  0.000095  0.000060  0.000147  0.000108
...            ...       ...       ...       ...       ...       ...
VFC   3.735924e-05  0.000044  0.000029  0.000020  0.000055  0.000068
VTR   5.750155e-05  0.000051  0.000023  0.000130  0.000058  0.000051
WYNN -3.883925e-05  0.000019  0.000009 -0.000019  0.000017 -0.000145
XOM   1.034506e-04  0.000083  0.000169  0.000043  0.000098  0.000104
XEL  -7.506973e-07  0.000038  0.000019  0.000053  0.000036  0.000029

             COP      CINF       ETN       AXP  ...       XRX       VMC  \
DWDP  0.000091  0.000049  0.000087  0.000128  ...  0.000076  0.000118
CTAS  0.000060  0.000051  0.000048  0.000060  ...  0.000092  0.000078
COG   0.000166  0.000073  0.000137  0.000090  ...  0.000067  0.000144
BXP   0.000024  0.000064  0.000009  0.000053  ...  0.000088  0.000022
COST  0.000056  0.000071  0.000093  0.000077  ...  0.000061  0.000116
...        ...       ...       ...       ...  ...       ...       ...
VFC   0.000053  0.000038  0.000024  0.000020  ...  0.000053  0.000052
VTR  -0.000022  0.000048  0.000022  0.000041  ...  0.000043  0.000064
WYNN  0.000021  0.000088  0.000002  0.000009  ... -0.000022 -0.000005
XOM   0.000148  0.000082  0.000155  0.000109  ...  0.000071  0.000119
XEL  -0.000018  0.000044  0.000007  0.000009  ...  0.000020  0.000033

             WMB      XLNX           WBA       VFC       VTR      WYNN  \
DWDP  0.000135  0.000192  8.814572e-05  0.000037  0.000058 -0.000039
CTAS  0.000059  0.000086  4.244215e-05  0.000044  0.000051  0.000019
COG   0.000170  0.000174  1.281764e-04  0.000029  0.000023  0.000009
BXP   0.000047  0.000080  8.001417e-05  0.000020  0.000130 -0.000019
COST  0.000073  0.000048  1.383163e-04  0.000055  0.000058  0.000017
...        ...       ...           ...       ...       ...       ...
VFC   0.000035  0.000051  6.087456e-05  0.000108  0.000017  0.000006
VTR   0.000020  0.000033  5.247087e-05  0.000017  0.000174 -0.000043
WYNN  0.000033 -0.000162 -8.763984e-05  0.000006 -0.000043  0.001070
XOM   0.000138  0.000126  1.355359e-04  0.000038  0.000030  0.000037
XEL  -0.000012 -0.000003 -6.491973e-07  0.000013  0.000067  0.000060

             XOM           XEL
DWDP  0.000103 -7.506973e-07
CTAS  0.000083  3.803561e-05
COG   0.000169  1.933293e-05
BXP   0.000043  5.285115e-05
COST  0.000098  3.606966e-05
...        ...           ...
VFC   0.000038  1.304749e-05
VTR   0.000030  6.687788e-05
WYNN  0.000037  6.034601e-05
XOM   0.000210  3.218190e-05
XEL   0.000032  8.527323e-05

[505 rows x 505 columns]
```

```
from sklearn.decomposition import PCA
# Step 5: Perform PCA
pca = PCA()
principal_components = pca.fit_transform(returns_df)
explained_variance = pca.explained_variance_ratio_

# Print explained variance ratio
print("\nExplained Variance Ratio (PCA):")
print(explained_variance)
```

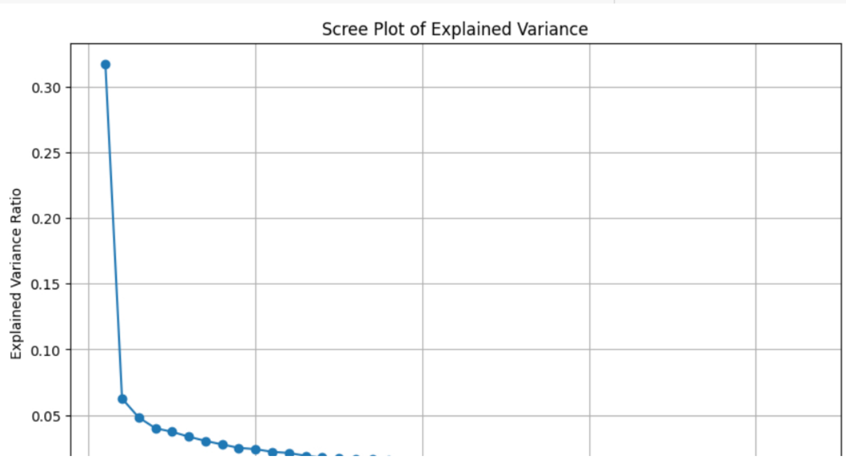
```
Explained Variance Ratio (PCA):
[3.17449626e-01 6.25541078e-02 4.81491888e-02 3.99602164e-02
 3.72209475e-02 3.35148205e-02 3.02986704e-02 2.76202654e-02
 2.49953077e-02 2.40067125e-02 2.20018266e-02 2.10962344e-02
 1.89976536e-02 1.75864572e-02 1.73234476e-02 1.66352073e-02
```

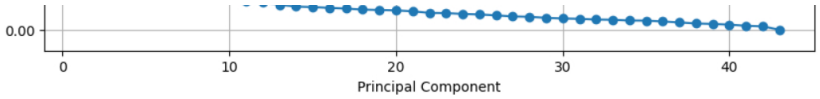
```
 1.89970550e-02 1.73804372e-02 1.72324470e-02 1.66352075e-02
 1.60704359e-02 1.53185136e-02 1.50225471e-02 1.46946848e-02
 1.40154830e-02 1.28167575e-02 1.26505508e-02 1.20192354e-02
 1.16142627e-02 1.09308114e-02 1.01139439e-02 9.84301438e-03
 9.02898135e-03 8.56451611e-03 8.29440878e-03 7.73843638e-03
 7.59545988e-03 6.99678737e-03 6.94571582e-03 6.27899887e-03
 5.55020627e-03 4.74799835e-03 4.41418094e-03 3.86191474e-03
 2.99658108e-03 2.55588370e-03 6.26058999e-32]
```

```python
# Step 6: Perform SVD
U, S, Vt = np.linalg.svd(cov_matrix)

# Print singular values
print("\nSingular Values (SVD):")
print(S)
```

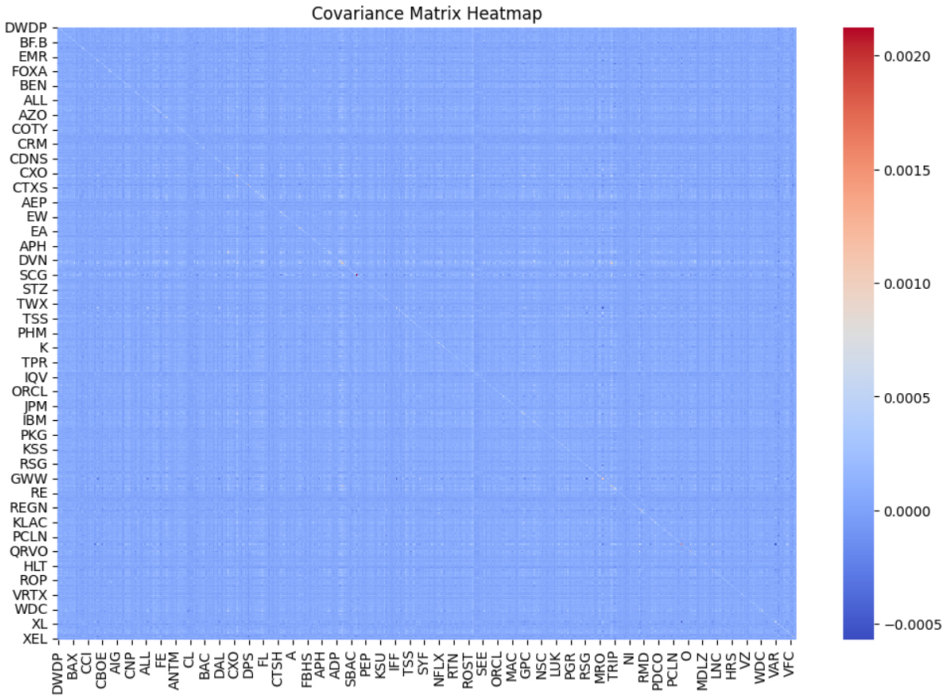
```
 4.08755153e-18 4.08755153e-18 4.08755153e-18 4.08755153e-18
 4.08755153e-18 4.08755153e-18 4.08755153e-18 4.08755153e-18
 4.08755153e-18 4.08755153e-18 4.08755153e-18 4.08755153e-18
 4.08755153e-18 4.08755153e-18 4.08755153e-18 4.08755153e-18
 4.08755153e-18 4.08755153e-18 4.08755153e-18 4.08755153e-18
 4.08755153e-18 4.08755153e-18 4.08755153e-18 4.08755153e-18
 4.08755153e-18 4.08755153e-18 4.08755153e-18 4.08755153e-18
 4.08755153e-18 4.08755153e-18 4.08755153e-18 4.08755153e-18
 4.08755153e-18 4.08755153e-18 4.08755153e-18 4.08755153e-18
 4.08755153e-18 4.08755153e-18 4.08755153e-18 4.08755153e-18
 4.08755153e-18 4.08755153e-18 4.08755153e-18 4.08755153e-18
 4.08755153e-18 4.08755153e-18 4.08755153e-18 4.08755153e-18
 4.08755153e-18 4.08755153e-18 4.08755153e-18 4.08755153e-18
 4.08755153e-18 4.08755153e-18 4.08755153e-18 4.08755153e-18
 4.08755153e-18 4.08755153e-18 4.08755153e-18 4.08755153e-18
 4.08755153e-18 4.08755153e-18 4.08755153e-18 4.08755153e-18
 4.08755153e-18 4.08755153e-18 4.08755153e-18 4.08755153e-18
 4.08755153e-18 4.08755153e-18 4.08755153e-18 4.08755153e-18
 4.08755153e-18 4.08755153e-18 4.08755153e-18 4.08755153e-18
 4.08755153e-18 4.08755153e-18 4.08755153e-18 4.08755153e-18
 4.08755153e-18 4.08755153e-18 4.08755153e-18 4.08755153e-18
 4.08755153e-18 4.08755153e-18 4.08755153e-18 4.08755153e-18
 4.08755153e-18 4.08755153e-18 4.08755153e-18 4.08755153e-18
 4.08755153e-18 4.08755153e-18 4.08755153e-18 4.08755153e-18
 4.08755153e-18 4.08755153e-18 4.08755153e-18 4.08755153e-18
 4.08755153e-18 4.08755153e-18 4.08755153e-18 4.08755153e-18
 4.08755153e-18 4.08755153e-18 4.08755153e-18 4.08755153e-18
 4.08755153e-18 4.08755153e-18 4.08755153e-18 4.08755153e-18
 4.08755153e-18 4.08755153e-18 4.08755153e-18 4.08755153e-18
 4.08755153e-18 4.08755153e-18 4.08755153e-18 4.08755153e-18
 4.08755153e-18 4.08755153e-18 4.08755153e-18 4.08755153e-18
 4.08755153e-18 4.08755153e-18 4.08755153e-18 4.08755153e-18
 4.08755153e-18 4.08755153e-18 4.08755153e-18 4.08755153e-18
 4.08755153e-18 4.08755153e-18 4.08755153e-18 4.08755153e-18
 4.08755153e-18 4.08755153e-18 4.08755153e-18 4.08755153e-18
 4.08755153e-18 4.08755153e-18 4.08755153e-18 4.08755153e-18
 4.08755153e-18 4.08755153e-18 4.08755153e-18 4.08755153e-18
 4.08755153e-18 4.08755153e-18 4.08755153e-18 4.08755153e-18
 4.08755153e-18 4.08755153e-18 4.08755153e-18 4.08755153e-18
 4.08755153e-18 4.08755153e-18 4.08755153e-18 4.08755153e-18
 4.08755153e-18 4.08755153e-18 4.08755153e-18 4.08755153e-18
 4.08755153e-18 4.08755153e-18 4.08755153e-18 4.08755153e-18
 4.08755153e-18 4.08755153e-18 4.08755153e-18 4.08755153e-18
 4.08755153e-18 4.08755153e-18 4.08755153e-18 4.08755153e-18
 4.08755153e-18 4.08755153e-18 4.08755153e-18 4.08755153e-18
 4.08755153e-18 4.08755153e-18 4.08755153e-18 4.08755153e-18
 4.08755153e-18 4.08755153e-18 4.08755153e-18 4.08755153e-18
 4.08755153e-18 4.08755153e-18 4.08755153e-18 4.08755153e-18
 4.08755153e-18 4.08755153e-18 4.08755153e-18 4.08755153e-18
 4.08755153e-18 4.08755153e-18 4.08755153e-18 4.08755153e-18
 4.08755153e-18 4.08755153e-18 4.08755153e-18 4.08755153e-18
 4.08755153e-18 4.08755153e-18 4.08755153e-18 4.08755153e-18
 4.08755153e-18 4.08755153e-18 4.08755153e-18 4.08755153e-18
 4.08755153e-18 4.08755153e-18 4.08755153e-18 4.08755153e-18
 4.08755153e-18 4.08755153e-18 4.08755153e-18 4.08755153e-18
 4.08755153e-18 4.08755153e-18 4.08755153e-18 4.08755153e-18
 4.08755153e-18 4.08755153e-18 4.08755153e-18 4.08755153e-18
 4.08755153e-18 4.08755153e-18 4.08755153e-18 2.78518903e-18
 9.75074605e-19]
```

```python
import matplotlib.pyplot as plt
# Step 7: Visualizations
# Scree Plot (Explained Variance Ratio)
plt.figure(figsize=(10, 6))
plt.plot(range(1, len(explained_variance) + 1), explained_variance, marker='o')
plt.title('Scree Plot of Explained Variance')
plt.xlabel('Principal Component')
plt.ylabel('Explained Variance Ratio')
plt.grid(True)
plt.show()
```





[9]
```
import seaborn as sns
# Heatmap of Covariance Matrix
plt.figure(figsize=(12, 8))
sns.heatmap(cov_matrix, cmap='coolwarm', annot=False)
plt.title('Covariance Matrix Heatmap')
plt.show()
```



[10]
```
# Step 8: Interpretation (Print Key Results)
print("\nTop 5 Eigenvectors (PCA):")
print(pca.components_[:5])

# Additional Interpretation
print("\nKey Insights:")
print("1. The first principal component explains the largest proportion of variance.")
print("2. Singular values indicate the importance of each dimension in the data.")
```

```
Top 5 Eigenvectors (PCA):
[[ 0.05811365  0.03214686  0.06091315 ... -0.00281216  0.05437171
   0.00346748]
 [ 0.01275586  0.04263893 -0.04366108 ...  0.0058134   0.00794854
   0.06810065]
 [ 0.05101396  0.01053209  0.05228765 ... -0.10484237 -0.02186964
   0.02040563]
 [-0.02854154  0.02051434  0.02599461 ...  0.15131361  0.01156248
   0.05452788]
 [-0.02828475 -0.00320748 -0.05707429 ...  0.10273525 -0.04540468
   0.00468335]]

Key Insights:
1. The first principal component explains the largest proportion of variance.
2. Singular values indicate the importance of each dimension in the data.
```

Start coding or generate with AI.