2. Yield Curve Modeling

```
!pip install fredapi
!pip install nelson_siegel_svensson
```

```
Collecting fredapi
  Downloading fredapi-0.5.2-py3-none-any.whl.metadata (5.0 kB)
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (from fredapi) (2.2.2)
Requirement already satisfied: numpy>=1.23.2 in /usr/local/lib/python3.11/dist-packages (from pandas->fredapi) (2
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas->fr
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas->fredapi) (20
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas->fredapi) (
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->
Downloading fredapi-0.5.2-py3-none-any.whl (11 kB)
Installing collected packages: fredapi
Successfully installed fredapi-0.5.2
Collecting nelson_siegel_svensson
  Downloading nelson_siegel_svensson-0.5.0-py2.py3-none-any.whl.metadata (6.7 kB)
Requirement already satisfied: Click>=8.0 in /usr/local/lib/python3.11/dist-packages (from nelson_siegel_svensson
Requirement already satisfied: numpy>=1.22 in /usr/local/lib/python3.11/dist-packages (from nelson_siegel_svensso
Requirement already satisfied: scipy>=1.7 in /usr/local/lib/python3.11/dist-packages (from nelson_siegel_svensson
Requirement already satisfied: matplotlib>=3.5 in /usr/local/lib/python3.11/dist-packages (from nelson_siegel_sve
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.5-
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.5->nel
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.5
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.5
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.5->
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.5->nelson
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.5-
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.7->ma
Downloading nelson_siegel_svensson-0.5.0-py2.py3-none-any.whl (9.9 kB)
Installing collected packages: nelson_siegel_svensson
Successfully installed nelson_siegel_svensson-0.5.0
```

```
import pandas as pd
from fredapi import Fred
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from scipy.interpolate import CubicSpline
from nelson_siegel_svensson.calibrate import calibrate_ns_ols
sns.set()
```

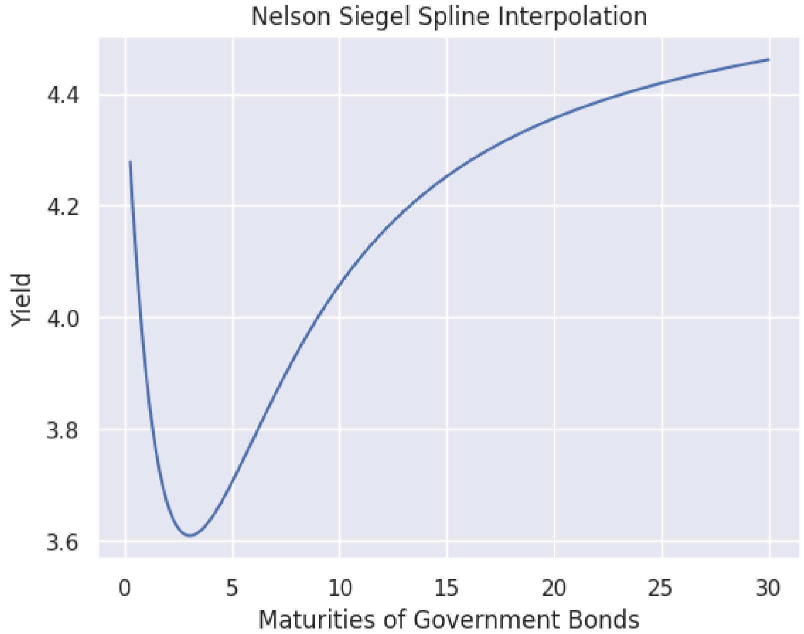
```
fred = Fred(api_key='642bcab619456030e8f5970e482486df')
# Maturities for Bonds
series_ids = ['DGS3MO', 'DGS6MO', 'DGS1', 'DGS2', 'DGS5', 'DGS10', 'DGS20', 'DGS30']
labels = ['3 Month', '6 Month', '1 Year', '2 Year', '5 Year', '10 Year', '20 Year', '30 Year']
maturities = np.array([0.25, 0.5, 1, 2, 5, 10, 20, 30])

# Collect the data from the FRED Api
def get_yield_data(series_id):
    return fred.get_series(series_id, observation_start='2000-01-01', observation_end="2025-04-04")

yields_dict = {series_id: get_yield_data(series_id) for series_id in series_ids}
yields = pd.DataFrame(yields_dict)
yields.columns = labels
yields.index = pd.to_datetime(yields.index)

# Select April 4th for the Date of the analysis
y = np.array(yields.loc["2025-04-04"])
```

```
# Fit Nelson-Siegel
curve, _ = calibrate_ns_ols(maturities, y, tau0=1.0)
y_hat = curve
print(y_hat)

# Fit Cubic Spline
cs = CubicSpline(maturities, y)
t_grid = np.linspace(0.25, 30, 300)  # finer grid for smooth curves
```

```
NelsonSiegelCurve(beta0=np.float64(4.671181963543552), beta1=np.float64(-0.20504031598437256), beta2=np.float64(-
```

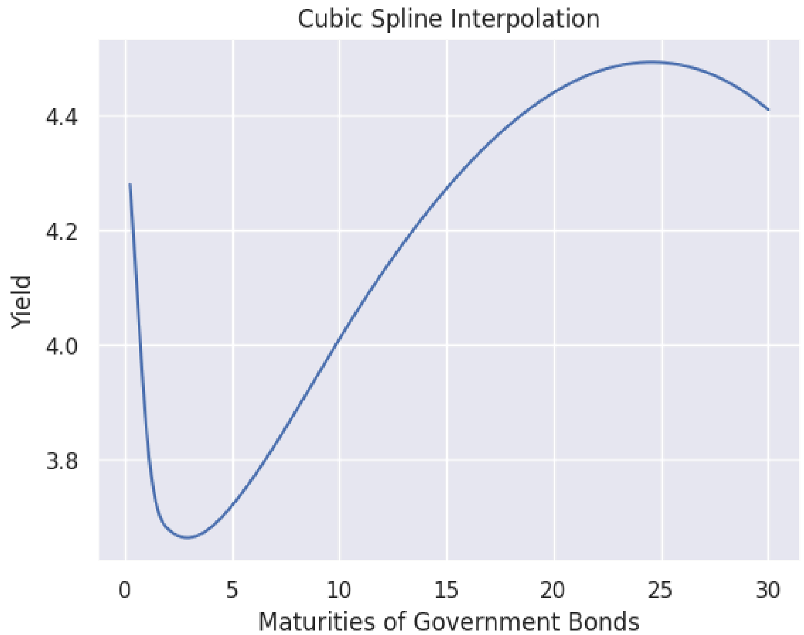
```
#Plot NS Individually
plt.plot(t_grid, y_hat(t_grid))
plt.xlabel('Maturities of Government Bonds')
plt.ylabel('Yield')
plt.title('Nelson Siegel Spline Interpolation')
```

```
Text(0.5, 1.0, 'Nelson Siegel Spline Interpolation')
```



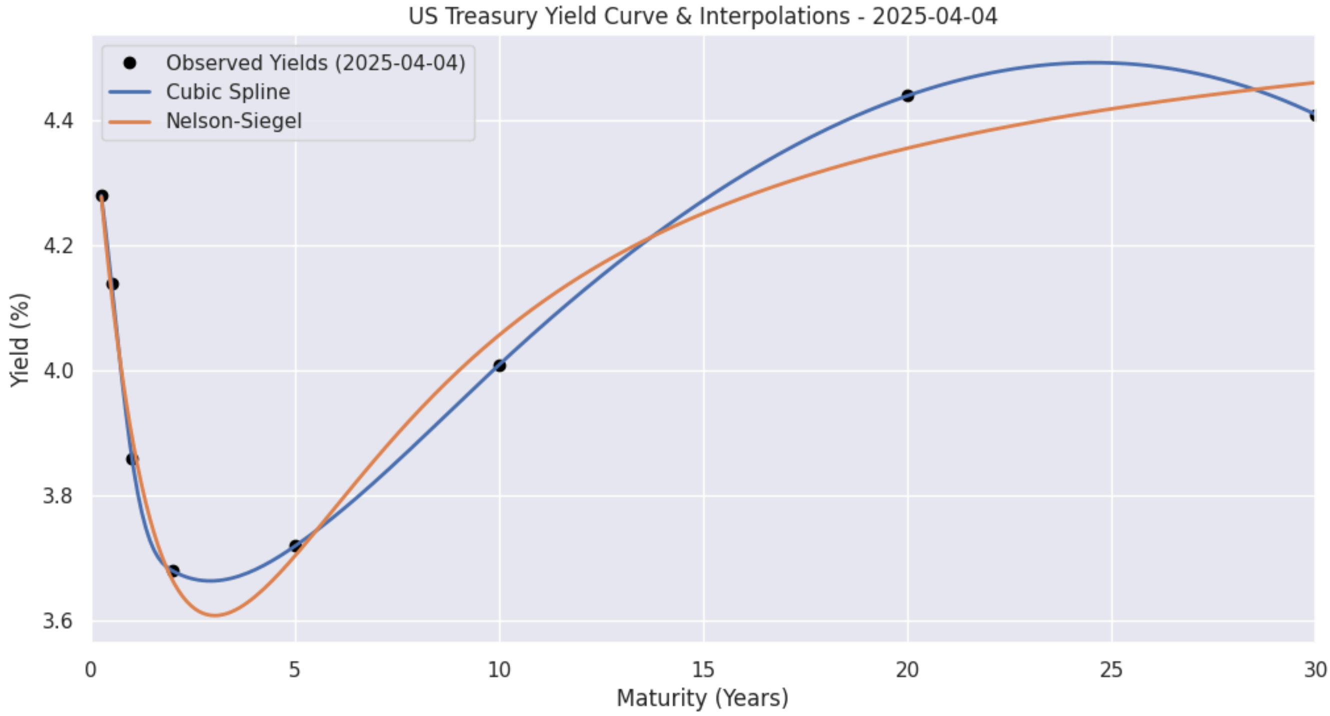

```
#Plot Cubic Spline Individually
plt.plot(t_grid, cs(t_grid))
plt.xlabel('Maturities of Government Bonds')
plt.ylabel('Yield')
plt.title('Cubic Spline Interpolation')
```

```
Text(0.5, 1.0, 'Cubic Spline Interpolation')
```



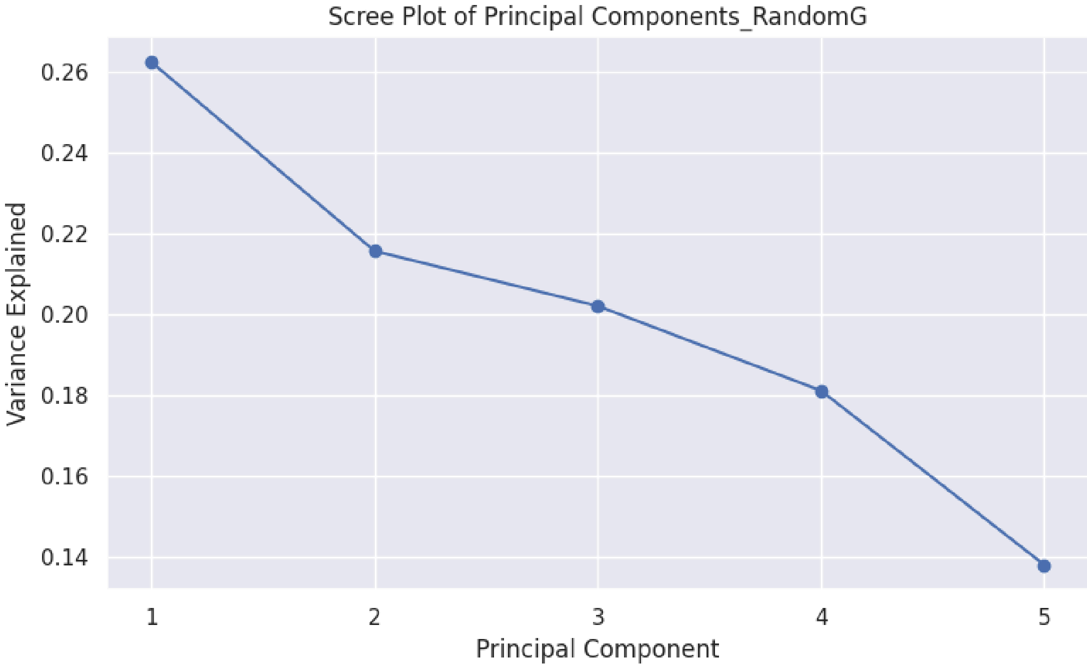

```
# Plot the NS, Cubic Spline interpolation, and actual yield values together
plt.figure(figsize=(12, 6))
plt.plot(maturities, y, 'o', label='Observed Yields (2025-04-04)', color='black')
plt.plot(t_grid, cs(t_grid), label='Cubic Spline', linewidth=2)
plt.plot(t_grid, y_hat(t_grid), label='Nelson-Siegel', linewidth=2)
plt.xlabel("Maturity (Years)")
plt.ylabel("Yield (%)")
plt.title("US Treasury Yield Curve & Interpolations - 2025-04-04")
plt.legend()
```

```
plt.grid(True)
plt.xlim(0, 30)
plt.show()
```



Q3. Exploiting Correlation

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


# Step a: Generate 5 uncorrelated Gaussian random variables
np.random.seed(42)
num_samples = 100
mean = 0
std_dev = 0.05
uncorrelated_data = np.random.normal(mean, std_dev, size=(num_samples, 5))
uncorrelated_df = pd.DataFrame(uncorrelated_data, columns=[f'Yield_{i+1}' for i in range(5)])


# Step b: Perform PCA using the covariance matrix
pca = PCA()
pca.fit(uncorrelated_df)

# Explained variance ratio
explained_variance_ratio = pca.explained_variance_ratio_

# Step d: Scree plot
plt.figure(figsize=(8, 5))
plt.plot(range(1, 6), explained_variance_ratio, marker='o')
plt.title('Scree Plot of Principal Components_RandomG')
plt.xlabel('Principal Component')
plt.ylabel('Variance Explained')
plt.grid(True)
plt.xticks(range(1, 6))
plt.tight_layout()
plt.savefig("my_plot.png", dpi=300)
plt.show()

# Optional: Print the explained variances
for i, var in enumerate(explained_variance_ratio, 1):
    print(f"Component {i}: {var:.4f}")
```




```
Component 1: 0.2626
Component 2: 0.2158
Component 3: 0.2022
Component 4: 0.1813
Component 5: 0.1382
```

```
from google.colab import drive
drive.mount('/content/drive')
```

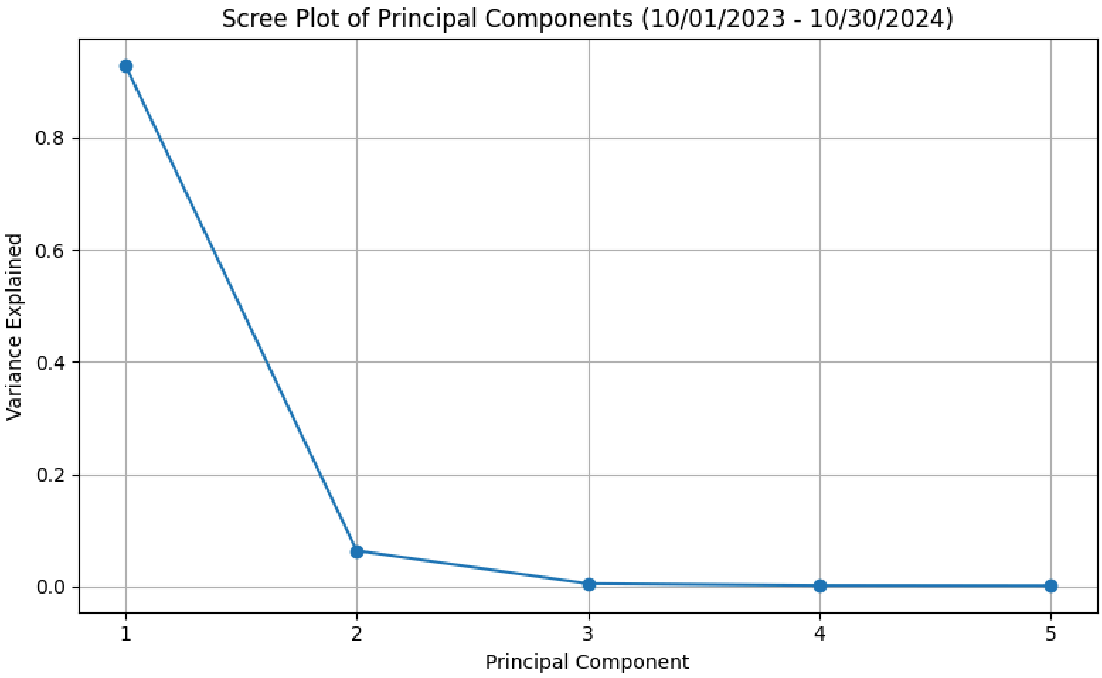
```
---------------------------------------------------------------------------
MessageError                              Traceback (most recent call last)
<ipython-input-15-d5df0069828e> in <cell line: 0>()
      1 from google.colab import drive
----> 2 drive.mount('/content/drive')

3 frames
/usr/local/lib/python3.11/dist-packages/google/colab/_message.py in read_reply_from_input(message_id,
timeout_sec)
    101     ):
    102       if 'error' in reply:
--> 103         raise MessageError(reply['error'])
    104       return reply.get('data', None)
    105

MessageError: Error: credential propagation was unsuccessful
```

Étapes suivantes : Expliquer l'erreur

```
# Step 1: Load Excel File
file_path = '/content/drive/My Drive/WQU/Task 1/par-real-yield-curve-rates-2003-2024.xlsx'
df = pd.read_excel(file_path)
df.head()

# Step 2: Convert 'Date' column to datetime
df['Date'] = pd.to_datetime(df['Date'])

# Step 3: Filter by date range
start_date = "2023-10-01"
end_date = "2024-10-30"
df_filtered = df[(df['Date'] >= start_date) & (df['Date'] <= end_date)].sort_values('Date')

# Step 4: Select relevant maturity columns
selected_columns = ['5 YR', '7 YR', '10 YR', '20 YR', '30 YR']
df_yields = df_filtered[selected_columns]

# Step 5: Compute daily yield changes
df_yield_changes = df_yields.diff().dropna()

# Step 6: Perform PCA using the covariance matrix
pca = PCA()
pca.fit(df_yield_changes)
explained_variance_ratio = pca.explained_variance_ratio_
```

```
explained_variance_ratio = pca.explained_variance_ratio_

# Step 7: Print explained variance for each component
print("Explained Variance Ratio by Component:")
for i, ratio in enumerate(explained_variance_ratio, 1):
    print(f"Component {i}: {ratio:.2%}")

# Step 8: Plot Scree Plot
plt.figure(figsize=(8, 5))
plt.plot(range(1, len(explained_variance_ratio) + 1), explained_variance_ratio, marker='o')
plt.title('Scree Plot of Principal Components (10/01/2023 - 10/30/2024)')
plt.xlabel('Principal Component')
plt.ylabel('Variance Explained')
plt.grid(True)
plt.xticks(range(1, len(explained_variance_ratio) + 1))
plt.tight_layout()
plt.savefig("my_plot2.png", dpi=300)
plt.show()
```

```
Explained Variance Ratio by Component:
Component 1: 92.88%
Component 2: 6.36%
Component 3: 0.49%
Component 4: 0.16%
Component 5: 0.11%
```



4.Empirical Analysis of ETFs

```
# 1. Import Libraries and Define Holdings
```

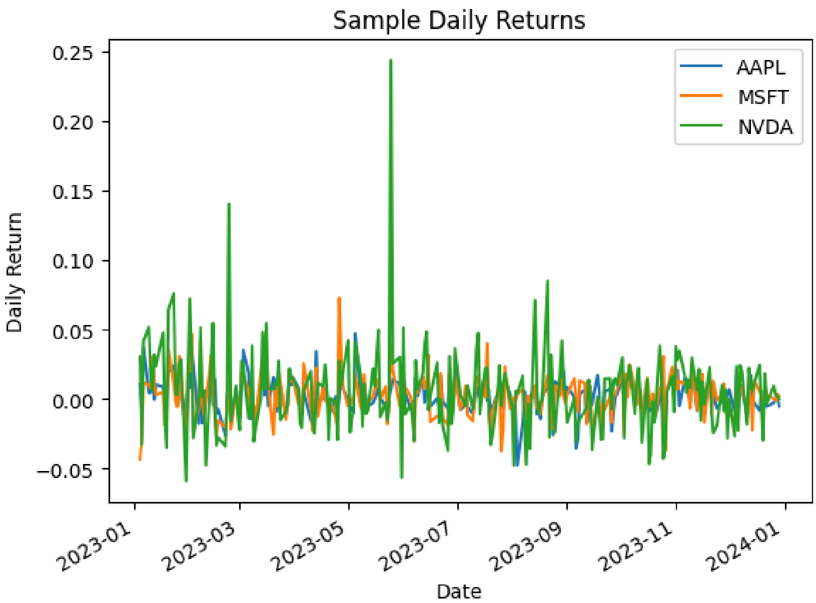
```
import yfinance as yf
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler

# List of the top 30 holdings
holdings = [
    'AAPL', 'MSFT', 'NVDA', 'AVGO', 'CRM', 'CSCO', 'ORCL', 'IBM', 'PLTR', 'ACN',
    'NOW', 'INTU', 'ADBE', 'QCOM', 'AMD', 'TXN', 'AMAT', 'PANW', 'ADI', 'KLAC',
    'CRWD', 'INTC', 'LRCX', 'MU', 'APH', 'ANET', 'CDNS', 'MSI', 'SNPS', 'FTNT'
]
```

```
# 2. Download Historical Data
```

```
# Download historical data
start_date = '2023-01-01'
```

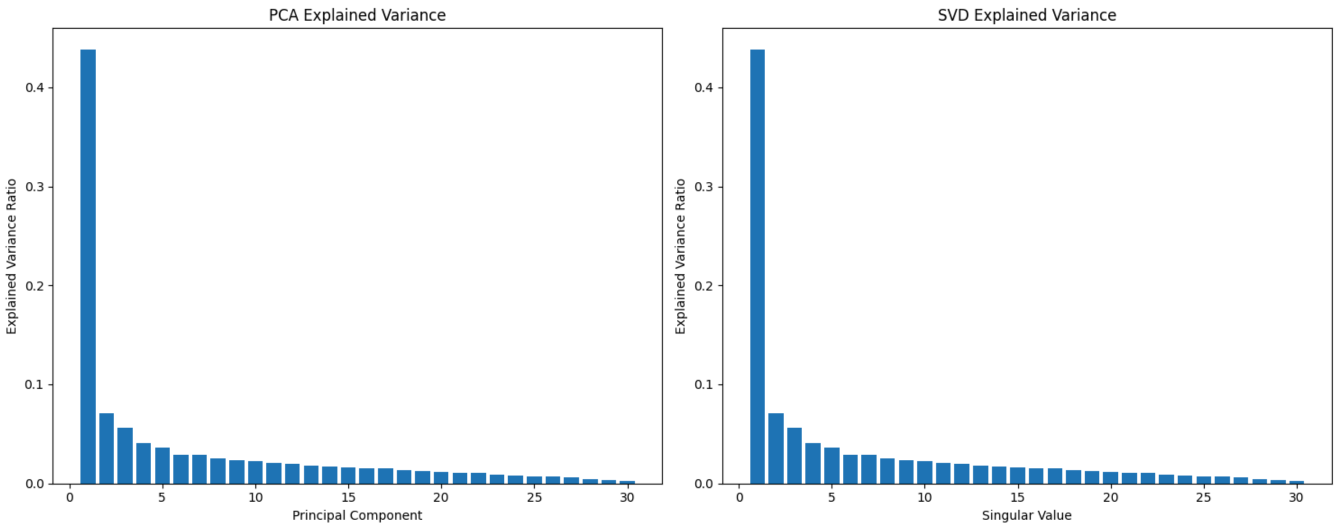
```
end_date = '2024-01-01'

# Fetch price data with error handling
def fetch_stock_data(tickers, start, end):
    data = {}
    for ticker in tickers:
        try:
            stock = yf.Ticker(ticker)
            hist = stock.history(start=start, end=end)
            if not hist.empty:
                data[ticker] = hist['Close']
            else:
                print(f"No data found for {ticker}")
        except Exception as e:
            print(f"Error fetching data for {ticker}: {e}")
    return pd.DataFrame(data)

# Fetch price data
price_data = fetch_stock_data(holdings, start_date, end_date)


# 3. Compute Daily Returns


# Compute daily returns
returns = price_data.pct_change().dropna()

# Plot daily returns for a sample of holdings
plt.figure(figsize=(12, 6))
returns[['AAPL', 'MSFT', 'NVDA']].plot(title='Sample Daily Returns')
plt.xlabel('Date')
plt.ylabel('Daily Return')
plt.show()
```

```
<Figure size 1200x600 with 0 Axes>
```




```
# 4. Perform PCA


def perform_pca(returns):
    # Standardize the returns
    scaler = StandardScaler()
    scaled_returns = scaler.fit_transform(returns)

    # Compute covariance matrix
    cov_matrix = np.cov(scaled_returns.T)

    # Compute eigenvalues and eigenvectors
    eigenvalues, eigenvectors = np.linalg.eig(cov_matrix)

    # Sort eigenvalues and eigenvectors
    idx = eigenvalues.argsort()[::-1]
    eigenvalues = eigenvalues[idx]
    eigenvectors = eigenvectors[:, idx]
```

```
    # Compute explained variance ratio
    explained_variance_ratio = eigenvalues / np.sum(eigenvalues)

    return eigenvalues, eigenvectors, explained_variance_ratio

# Perform PCA
pca_eigenvalues, pca_eigenvectors, pca_explained_variance = perform_pca(returns)


# 5. Perform SVD


def perform_svd(returns):
    # Standardize the returns
    scaler = StandardScaler()
    scaled_returns = scaler.fit_transform(returns)

    # Perform SVD
    U, S, Vt = np.linalg.svd(scaled_returns, full_matrices=False)

    # Compute explained variance ratio
    explained_variance_ratio = (S**2) / (S**2).sum()

    return U, S, Vt, explained_variance_ratio

# Perform SVD
svd_U, svd_S, svd_Vt, svd_explained_variance = perform_svd(returns)


# 6. Plot Explained Variance


# Plotting
plt.figure(figsize=(15, 6))

# PCA Explained Variance
plt.subplot(1, 2, 1)
plt.bar(range(1, len(pca_explained_variance) + 1), pca_explained_variance)
plt.title('PCA Explained Variance')
plt.xlabel('Principal Component')
plt.ylabel('Explained Variance Ratio')

# SVD Explained Variance
plt.subplot(1, 2, 2)
plt.bar(range(1, len(svd_explained_variance) + 1), svd_explained_variance)
plt.title('SVD Explained Variance')
plt.xlabel('Singular Value')
plt.ylabel('Explained Variance Ratio')

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




```
# 7. Print Top 5 PCA Components
```

```
# Print top 5 components details
print("Top 5 PCA Components:")
for i in range(min(5, len(pca_explained_variance))):
    print(f"PC{i+1} Explained Variance: {pca_explained_variance[i]:.4f}")
    print("Top 5 holdings weights:")

    # Get the absolute values of eigenvector components
    component_weights = np.abs(pca_eigenvectors[:, i])

    # Get indices of top 5 holdings by weight
    top_holdings_indices = component_weights.argsort()[-5:][::-1]

    for idx in top_holdings_indices:
        print(f"{returns.columns[idx]}: {pca_eigenvectors[idx, i]:.4f}")
    print("\n")
```

```
Top 5 PCA Components:
PC1 Explained Variance: 0.4378
Top 5 holdings weights:
AMAT: -0.2272
KLAC: -0.2267
SNPS: -0.2194
LRCX: -0.2189
CDNS: -0.2183


PC2 Explained Variance: 0.0705
Top 5 holdings weights:
PANW: -0.3551
NOW: -0.2627
CRWD: -0.2572
ADI: 0.2546
TXN: 0.2534


PC3 Explained Variance: 0.0563
Top 5 holdings weights:
IBM: 0.4179
MSI: 0.3442
CSCO: 0.3279
NVDA: -0.3030
AMD: -0.2780


PC4 Explained Variance: 0.0405
Top 5 holdings weights:
ORCL: -0.3431
INTC: 0.3076
MSI: -0.2995
CRWD: 0.2976
FTNT: 0.2908


PC5 Explained Variance: 0.0366
Top 5 holdings weights:
FTNT: 0.3756
CRM: -0.3258
PANW: 0.3248
CSCO: 0.3030
ANET: 0.2937
```

```
# 8. Additional Analysis
```

```
# Additional analysis
print("Correlation Matrix:")
print(returns.corr())

# Cumulative explained variance
cumulative_variance = np.cumsum(pca_explained_variance)
plt.figure(figsize=(10, 6))
plt.plot(range(1, len(cumulative_variance) + 1), cumulative_variance, marker='o')
plt.title('Cumulative Explained Variance')
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Explained Variance')
plt.show()
```

```
Correlation Matrix:
          AAPL      MSFT      NVDA      AVGO       CRM      CSCO      ORCL  \
AAPL  1.000000  0.547988  0.444878  0.434549  0.377975  0.328291  0.370751
MSFT  0.547988  1.000000  0.537430  0.376222  0.377104  0.223530  0.391094
NVDA  0.444878  0.537430  1.000000  0.531366  0.343846  0.221271  0.371308
AVGO  0.434549  0.376222  0.531366  1.000000  0.353904  0.423490  0.343016
CRM   0.377975  0.377104  0.343846  0.353904  1.000000  0.254338  0.259192
CSCO  0.328291  0.223530  0.221271  0.423490  0.254338  1.000000  0.258416
ORCL  0.370751  0.391094  0.371308  0.343016  0.259192  0.258416  1.000000
IBM   0.182767  0.070176  0.104757  0.245552  0.153265  0.321830  0.253418
PLTR  0.375956  0.346665  0.381440  0.361398  0.372947  0.265009  0.281685
ACN   0.451339  0.473672  0.375202  0.450890  0.387718  0.337262  0.375924
NOW   0.458602  0.569288  0.502150  0.421653  0.535508  0.333753  0.444088
INTU  0.455333  0.498651  0.408834  0.442168  0.465353  0.309624  0.334302
ADBE  0.527474  0.579054  0.541431  0.565212  0.478978  0.360373  0.456711
QCOM  0.464025  0.323573  0.431194  0.489638  0.401756  0.298713  0.265858
AMD   0.402136  0.530361  0.668893  0.519088  0.395401  0.227546  0.334531
TXN   0.488830  0.312355  0.429724  0.586272  0.368211  0.414509  0.353159
AMAT  0.451134  0.421851  0.591245  0.641544  0.377457  0.328567  0.330942
PANW  0.378021  0.335065  0.330223  0.341099  0.342323  0.316487  0.262216
ADI   0.447299  0.295796  0.388842  0.538457  0.354983  0.383433  0.267515
KLAC  0.480677  0.444998  0.563281  0.651273  0.362196  0.311058  0.352004
CRWD  0.426177  0.436996  0.391229  0.401886  0.486871  0.279109  0.276268
INTC  0.352117  0.344144  0.213205  0.390712  0.382972  0.218625  0.225804
LRCX  0.425432  0.387017  0.552172  0.610009  0.350876  0.272292  0.317949
MU    0.337207  0.301900  0.478683  0.481585  0.268574  0.256862  0.202411
APH   0.450822  0.276597  0.377724  0.500839  0.338881  0.391490  0.268847
ANET  0.325698  0.418948  0.448562  0.496426  0.265638  0.392156  0.299080
CDNS  0.429293  0.512292  0.669369  0.530861  0.379072  0.284035  0.458817
MSI   0.302548  0.258433  0.115651  0.281191  0.172782  0.308038  0.239925
SNPS  0.452209  0.516943  0.691689  0.588809  0.390972  0.259906  0.443926
FTNT  0.428492  0.269774  0.230336  0.272377  0.218829  0.246137  0.210779

           IBM      PLTR       ACN  ...      CRWD      INTC      LRCX  \
AAPL  0.182767  0.375956  0.451339  ...  0.426177  0.352117  0.425432
MSFT  0.070176  0.346665  0.473672  ...  0.436996  0.344144  0.387017
NVDA  0.104757  0.381440  0.375202  ...  0.391229  0.213205  0.552172
AVGO  0.245552  0.361398  0.450890  ...  0.401886  0.390712  0.610009
CRM   0.153265  0.372947  0.387718  ...  0.486871  0.382972  0.350876
CSCO  0.321830  0.265009  0.337262  ...  0.279109  0.218625  0.272292
ORCL  0.253418  0.281685  0.375924  ...  0.276268  0.225804  0.317949
IBM   1.000000  0.209665  0.431655  ...  0.101178  0.181371  0.245420
PLTR  0.209665  1.000000  0.369324  ...  0.447195  0.290468  0.347305
ACN   0.431655  0.369324  1.000000  ...  0.388987  0.367490  0.461082
NOW   0.234470  0.473494  0.547642  ...  0.621581  0.338860  0.406037
INTU  0.215067  0.466227  0.520936  ...  0.484849  0.424755  0.481302
ADBE  0.303732  0.429558  0.523507  ...  0.461797  0.345255  0.503562
QCOM  0.272332  0.388578  0.438981  ...  0.369392  0.472011  0.601911
AMD   0.130842  0.426791  0.371901  ...  0.475381  0.430021  0.657711
TXN   0.353997  0.426590  0.539714  ...  0.408451  0.503006  0.682388
AMAT  0.280461  0.400799  0.489045  ...  0.436304  0.467264  0.903877
PANW  0.071798  0.355873  0.243196  ...  0.610486  0.180521  0.263172
ADI   0.305449  0.444603  0.485688  ...  0.419013  0.521412  0.652935
KLAC  0.289699  0.354398  0.470092  ...  0.425002  0.494783  0.893561
CRWD  0.101178  0.447195  0.388987  ...  1.000000  0.355881  0.400285
INTC  0.181371  0.290468  0.367490  ...  0.355881  1.000000  0.479516
LRCX  0.245420  0.347305  0.461082  ...  0.400285  0.479516  1.000000
MU    0.167516  0.314376  0.449702  ...  0.298312  0.421427  0.633305
APH   0.440691  0.363589  0.570640  ...  0.355980  0.401100  0.551241
ANET  0.071676  0.272307  0.357330  ...  0.408833  0.231961  0.340780
CDNS  0.186205  0.504777  0.448008  ...  0.538248  0.313594  0.574732
MSI   0.312858  0.148703  0.346248  ...  0.225159  0.186192  0.236438
SNPS  0.173296  0.498357  0.436132  ...  0.544934  0.314324  0.596361
FTNT  0.146815  0.256367  0.294759  ...  0.418520  0.184902  0.259870

            MU       APH      ANET      CDNS       MSI      SNPS      FTNT
AAPL  0.337207  0.450822  0.325698  0.429293  0.302548  0.452209  0.428492
MSFT  0.301900  0.276597  0.418948  0.512292  0.258433  0.516943  0.269774
NVDA  0.478683  0.377724  0.448562  0.669369  0.115651  0.691689  0.230336
AVGO  0.481585  0.500839  0.496426  0.530861  0.281191  0.588809  0.272377
CRM   0.268574  0.338881  0.265638  0.379072  0.172782  0.390972  0.218829
CSCO  0.256862  0.391490  0.392156  0.284035  0.308038  0.259906  0.246137
ORCL  0.202411  0.268847  0.299080  0.458817  0.239925  0.443926  0.210779
IBM   0.167516  0.440691  0.071676  0.186205  0.312858  0.173296  0.146815
PLTR  0.314376  0.363589  0.272307  0.504777  0.148703  0.498357  0.256367
ACN   0.449702  0.570640  0.357330  0.448008  0.346248  0.436132  0.294759
NOW   0.303881  0.394682  0.411684  0.631406  0.252807  0.626721  0.391589
INTU  0.375062  0.445890  0.298013  0.555927  0.257418  0.519515  0.342884
ADBE  0.404619  0.418938  0.420078  0.640688  0.262756  0.615430  0.283131
QCOM  0.511734  0.492038  0.307353  0.485975  0.190464  0.504767  0.126221
AMD   0.508863  0.403130  0.497486  0.631923  0.186844  0.654275  0.242436
TXN   0.505652  0.594622  0.330218  0.493021  0.286676  0.508568  0.316898
AMAT  0.573669  0.559343  0.363826  0.613104  0.222931  0.645130  0.286357
PANW  0.177608  0.176130  0.330895  0.405077  0.177463  0.401187  0.506858
ADI   0.447351  0.614553  0.285745  0.482076  0.278304  0.465351  0.307534
```