In [1]:

```
# Import necessary libraries
!pip install pandas_datareader
!pip install yfinance pandas beautifulsoup4 requests lxml matplotlib
import pandas_datareader.data as web
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.optimize import curve_fit
from scipy.interpolate import CubicSpline
from sklearn.decomposition import PCA
import datetime
import yfinance as yf
import requests
from bs4 import BeautifulSoup
```

```
Requirement already satisfied: pandas_datareader in /usr/local/lib/python3.11/
packages (0.10.0)
Requirement already satisfied: lxml in /usr/local/lib/python3.11/dist-package
(from pandas_datareader) (5.4.0)
Requirement already satisfied: pandas>=0.23 in /usr/local/lib/python3.11/dist
packages (from pandas_datareader) (2.2.2)
Requirement already satisfied: requests>=2.19.0 in /usr/local/lib/python3.11/
packages (from pandas_datareader) (2.32.3)
Requirement already satisfied: numpy>=1.23.2 in /usr/local/lib/python3.11/dis
packages (from pandas>=0.23->pandas_datareader) (2.0.2)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/pytho
dist-packages (from pandas>=0.23->pandas_datareader) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist
packages (from pandas>=0.23->pandas_datareader) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/di
packages (from pandas>=0.23->pandas_datareader) (2025.2)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/
python3.11/dist-packages (from requests>=2.19.0->pandas_datareader) (3.4.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist
packages (from requests>=2.19.0->pandas_datareader) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.1
dist-packages (from requests>=2.19.0->pandas_datareader) (2.4.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.1
dist-packages (from requests>=2.19.0->pandas_datareader) (2025.6.15)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-pac
(from python-dateutil>=2.8.2->pandas>=0.23->pandas_datareader) (1.17.0)
Requirement already satisfied: yfinance in /usr/local/lib/python3.11/dist-pac
(0.2.64)
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packa
(2.2.2)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.11/di
packages (4.13.4)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-pac
(2.32.3)
Requirement already satisfied: lxml in /usr/local/lib/python3.11/dist-package
(5.4.0)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-
packages (3.10.0)
Requirement already satisfied: numpy>=1.16.5 in /usr/local/lib/python3.11/dis
packages (from yfinance) (2.0.2)
Requirement already satisfied: multitasking>=0.0.7 in /usr/local/lib/python3.
dist-packages (from yfinance) (0.0.11)
Requirement already satisfied: platformdirs>=2.0.0 in /usr/local/lib/python3.
dist-packages (from yfinance) (4.3.8)
Requirement already satisfied: pytz>=2022.5 in /usr/local/lib/python3.11/dist
packages (from yfinance) (2025.2)
Requirement already satisfied: frozendict>=2.3.4 in /usr/local/lib/python3.11/
packages (from yfinance) (2.4.6)
Requirement already satisfied: peewee>=3.16.2 in /usr/local/lib/python3.11/di
packages (from yfinance) (3.18.1)
Requirement already satisfied: curl_cffi>=0.7 in /usr/local/lib/python3.11/di
packages (from yfinance) (0.11.4)
Requirement already satisfied: protobuf>=3.19.0 in /usr/local/lib/python3.11/
packages (from yfinance) (5.29.5)
Requirement already satisfied: websockets>=13.0 in /usr/local/lib/python3.11/
packages (from yfinance) (15.0.1)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/pytho
dist-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/di
packages (from pandas) (2025.2)
```

```
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.11/dis
packages (from beautifulsoup4) (2.7)
Requirement already satisfied: typing-extensions>=4.0.0 in /usr/local/lib/
python3.11/dist-packages (from beautifulsoup4) (4.14.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/
python3.11/dist-packages (from requests) (3.4.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist
packages (from requests) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.1
dist-packages (from requests) (2.4.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.1
dist-packages (from requests) (2025.6.15)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/
packages (from matplotlib) (1.3.2)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist
packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11
packages (from matplotlib) (4.58.4)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11
packages (from matplotlib) (1.4.8)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/d
packages (from matplotlib) (24.2)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-pa
(from matplotlib) (11.2.1)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/
packages (from matplotlib) (3.2.3)
Requirement already satisfied: cffi>=1.12.0 in /usr/local/lib/python3.11/dist
packages (from curl_cffi>=0.7->yfinance) (1.17.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-pac
(from python-dateutil>=2.8.2->pandas) (1.17.0)
Requirement already satisfied: pycparser in /usr/local/lib/python3.11/dist-pa
(from cffi>=1.12.0->curl_cffi>=0.7->yfinance) (2.22)
```

Data Quality

The current financial challenges and economic slowdown unvails the critical role of informed fina decision-making for individuals (W. Fan et al., 2001), businesses and organizations alike. Effecti decision-making begins with robust intelligence gathering, which in turn relies heavily on high-qu to drive optimal outcomes (C.W. Fisher et al., 2001).

Like everything else around us, the financial markets have drastically changed as a result of the information technology revolution which has enabled the automation and computerisation of wor processes and business functions, as well as the generation and fast processing of bogus volum data that has in turn fostered innovation in new financial products and strategies (N. Jenkinson e 2003).

A well structured data is highly essential for a well-functioning financial markets globally, caterin requirements of both public and private sectors.

As Nigel Jenkinson and Irina note: """*Inadequate quality and standardisation of financial Data le unacceptable high operational risk of trade processing; poor monitoring and management of fina risks at the individual firm and system-wide levels, by not allowing effective aggregation of positi the entity and product level across markets; as well as providing obstacles to the effective execu insolvency and resolution procedures.* Such inadequacies ultimately fuelled and exacerbated the financial crisis."""

Therefore, a proper and extensive data framework that delivers accurate and up-to-date insights financial positions, exposures and risks in an organization is highly essential so as to informed e risk management.

In [2]:
```
# Example of a poor quality structured data randomly initiated
data = {
    "Customer_ID": ['001', '002', None, '004', '005', '006'],
    "Name": ['John', '', 'Wale D.', 'Chris', 'TUNDE', 'johN'],
    "Order_ID": [1001, 1002, 1003, 1004, 1005, 1006],
    "Order_Date": ['01/05/24', '2024-06-02', '06/03/2024', '06-06-24', '0
    "Product": ['Beats speaker', 'Soundcore 14, Macbook', 'NULL', 'Apple
    "Quantity": ['1', 'two', '1, 2', '1', '-1', '1'],
    "Price": ['999', '', '999,1299', '799', '999', '899'],
    "Description": ['-', 'urgent delivery', 'Combined order', '-', 'retur
}

# Load into a DataFrame
df_uncleaned = pd.DataFrame(data)

# Display the raw dataset
print("Uncleaned Data:")
print(df_uncleaned)
```

```
Uncleaned Data:
  Customer_ID     Name  Order_ID  Order_Date                Product Quantity
0         001     John      1001    01/05/24          Beats speaker        1
1         002               1002  2024-06-02  Soundcore 14, Macbook      two
2        None  Wale D.      1003  06/03/2024                   NULL     1, 2
3         004    Chris      1004    06-06-24           Apple earpod        1
4         005    TUNDE      1005    05/06/24                    JBL       -1
5         006     johN      1006      June 6                   NULL        1

      Price                        Description
0       999                                  -
1                              urgent delivery
2  999,1299                     Combined order
3       799                                  -
4       999                   return processed
5       899  duplicate customer, lowercase name
```

This is a random example of a poor quality structured data. Although the data are structured in a format, in rows and columns, but has high level of errors, and inconsistencies that affect its relial usability.

The poor quality of this data can be idetified by the inconsistency in the data formatting (01/05/2 2024-06-02, June 6), likewise, the name column characters are not consistent, with some names an higher case while the rest with lower cases. Also, there are missing and invalid values in som which definitely would thwart the result of the analysis. Additionally, some columns contain multi values in a single cell, for instance, in the product column, a cell contains both 'soundcore 14' an 'Macbook' which are two seperate categories of product.

In [3]:
```
# Poor quality unstructured financial statements randomly initiated
unstructured_data_poor_formal = [
    "The stock moved qu0ote a bit today due to several factors, although
    "Management discussed some strategic priorities during the call, but%
    "There may be a $hift in macro conditions soon, but timing and impact
```

```
    "The company appears to be doing relatively well, th-ough exact figur
    "Some uncertainty remains aroundjnjthe interest[] rate environment, s
    "It seems th$at earnings were affected by a number of variables, incl
    "There& was some comme=tary around forward guidance, but the implicat
    "Valuations are arguably high, depending on how you interpret current
    "Profit margins could improve going forward, assuming current trends 
    "While t*here are headwinds, the overall outlook could still be descr
]
```

Although poor quality are quite difficult to spot in unstructure data, yet, this unstructured data (fin statement) is a poor quality data because it contains Inaccuracies (e.g., "qu0ote" instead of "quit ough" instead of "though" etc). Also, It contains incompleteness with the text lacking specific det concrete data (e.g., exact figures, specific dates, quantifiable metrics). Additionally, the data has or imprecise terminologyies with terms like "relatively well", "high valuations", and "cautiously op are subjective and may be interpreted differently by various readers.

Yield Curve

Yield curve is the relationship between bond yields (interest rates) and their respective maturity It reveals how bond yields varies across different bond durations, which may range from short-te long-term. Below is an analysis of US Treasury yields within 30yrs maturity date in 2024.

`Nelson-Siegel model`

The Nelson-Siegel model is simply a mathematical formula that helps us draw and understand t curve. This model basically analyzes the yield curve by sectionizing it into three components:

1. Level: An overview average level of the interest rates of the bond
2. Slope: The difference between short-term and long-term interest rates
3. Curvature: The shape of the yield curve, displaying a bend from a simple linear relationship

Nelson-Siegel model I mathematicaly expressed as:

R(t)=β 0 +β 1 (λt 1−e −λt)+β 2 (λt 1−e −λt −e −λt)

Where:

- R(t): Yield at maturity
- β0: Long-term level
- β1: Short-term slope component
- β2: Curvature component
- λ: Decay parameter
- e: Exponential function (e^x)

In [4]:
```
# Define date range
start = datetime.datetime(2024, 1, 1)
end = datetime.datetime(2024, 12, 31)

# FRED codes for Treasury yields
tickers = {
    '1M': 'DGS1MO',
```

```python
    '3M': 'DGS3MO',
    '6M': 'DGS6MO',
    '1Y': 'DGS1',
    '2Y': 'DGS2',
    '5Y': 'DGS5',
    '7Y': 'DGS7',
    '10Y': 'DGS10',
    '20Y': 'DGS20',
    '30Y': 'DGS30'
}

# Load data
data = pd.DataFrame()
for label, code in tickers.items():
    data[label] = web.DataReader(code, 'fred', start, end)

# Drop missing values
data.dropna(inplace=True)
print(data.tail())
```

```
              1M    3M    6M    1Y    2Y    5Y    7Y   10Y   20Y   30Y
DATE
2024-12-24  4.44  4.40  4.30  4.24  4.29  4.43  4.52  4.59  4.84  4.76
2024-12-26  4.45  4.35  4.31  4.23  4.30  4.42  4.49  4.58  4.83  4.76
2024-12-27  4.44  4.31  4.29  4.20  4.31  4.45  4.53  4.62  4.89  4.82
2024-12-30  4.43  4.37  4.25  4.17  4.24  4.37  4.46  4.55  4.84  4.77
2024-12-31  4.40  4.37  4.24  4.16  4.25  4.38  4.48  4.58  4.86  4.78
```

In [5]:

```python
# Get most recent date's yield curve
latest_yields = data.iloc[-1]
print(f"\nDate: {data.index[-1].date()}")
print(latest_yields)
```

```
Date: 2024-12-31
1M     4.40
3M     4.37
6M     4.24
1Y     4.16
2Y     4.25
5Y     4.38
7Y     4.48
10Y    4.58
20Y    4.86
30Y    4.78
Name: 2024-12-31 00:00:00, dtype: float64
```

In [6]:

```python
# Maturities in years
maturity_map = {
    '1M': 1/12, '3M': 3/12, '6M': 6/12,
    '1Y': 1, '2Y': 2, '5Y': 5,
    '7Y': 7, '10Y': 10, '20Y': 20, '30Y': 30
}

maturities = np.array([maturity_map[key] for key in latest_yields.index])
yields = latest_yields.values

def nelson_siegel(tau, beta0, beta1, beta2, lambd):
    term1 = (1 - np.exp(-lambd * tau)) / (lambd * tau)
    term2 = term1 - np.exp(-lambd * tau)
    return beta0 + beta1 * term1 + beta2 * term2
```

```
# Initial guess
initial_guess = [4.0, -2.0, 1.0, 0.5]

# Fitting Nelson's model
params, _ = curve_fit(nelson_siegel, maturities, yields, p0=initial_guess
beta0, beta1, beta2, lambd = params
print(f"Fitted Parameters:\n β0 = {beta0:.4f}, β1 = {beta1:.4f}, β2 = {be
```

```
Fitted Parameters:
 β0 = 4.9331, β1 = -0.5158, β2 = -1.6016, λ = 0.6673
```

Based on the Nelson-Siegel model analysis Above, the result parameters include: Based on you

β_0 =4.9331 a value of 4.9331 indicate that very long-term interest rates are expected to be arou 4.9331%. This gives the overall height of the yield curve.

$\beta_1=-0.5158$ a value of −0.5158 indicates a negative slope for the short end of the yield curve. Tl means that short-term interest rates are currently higher than longer-term rates, inferring an "rev portion of the yield curve. An inverted yield curve usually suggest a potential economic downtime recession.

$\beta_2=-1.6016$ a value of −1.6016 s a indicate curvature, which suggest a "dip" in the yield curve fc medium-term maturities, or that the curve is somehow concave. This infer that the medium-term relatively lower than what a simple upward or downward slope would suggest.

$\lambda=0.6673$ a value of 0.6673 indicates that short-term dynamics have a significant but moderate ii the yield curve, primarily influencing short to medium-term maturities. The curvature component' peaks at a relatively short to medium maturity and diminishes quickly as maturities lengthen, trai to long-term trends at a noticeable pace.

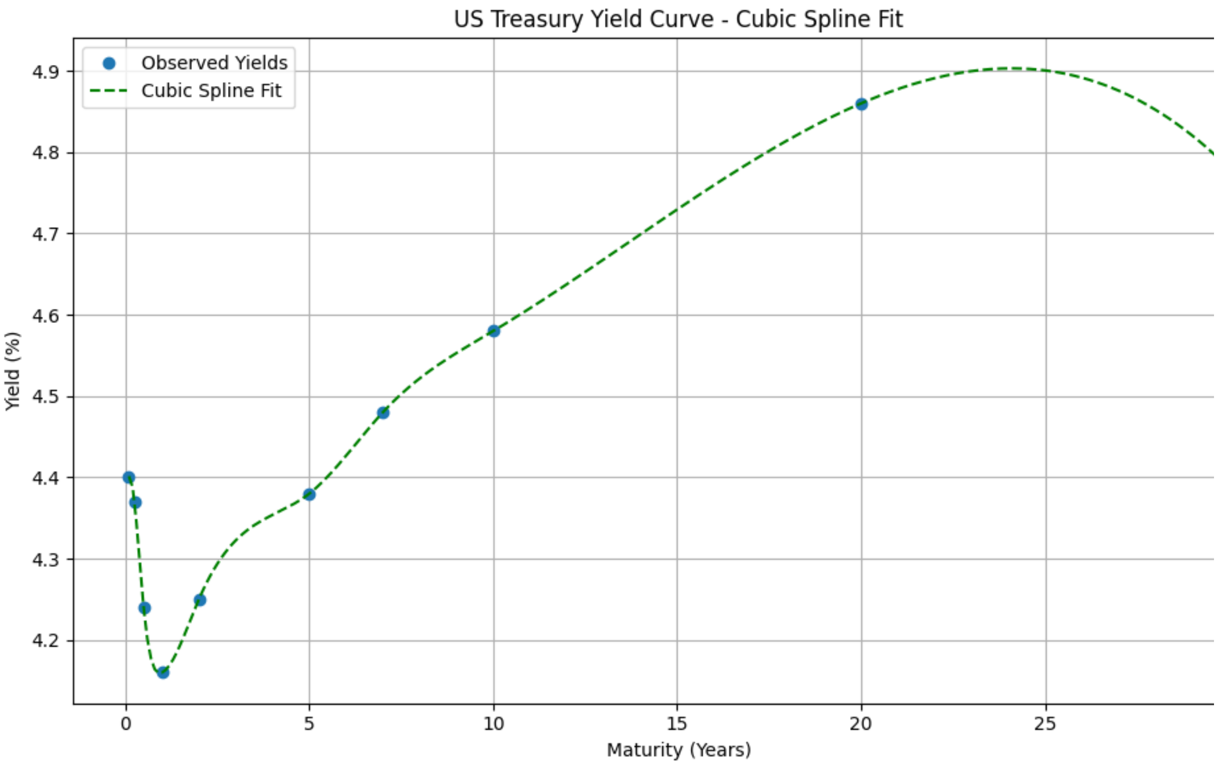
`Cubic Splines`

Cubic spline is a highly efficient way to illustrate a moderately smooth, flexible, and natural-lookii through a set of unarraged points by breaking the problem into manageable, smoothly connecte segments.

Real-world data often poses analytical challenges due to its complexity. Finding a single functior accurately model the data can be elusive and result in cumbersome equations. To address this, spline method was developed, which involves fitting a series of cubic polynomials between data create a smooth and continuous curve. This technique enables the calculation of rates of change cumulative changes over specific intervals (McKinley et al., 1998).

In [7]:
```
# Fit Cubic Spline
spline_model = CubicSpline(maturities, yields)

# Generate smooth curve
maturity_smooth = np.linspace(0.1, 30, 300)
yield_smooth = spline_model(maturity_smooth)

plt.figure(figsize=(10, 6))
plt.plot(maturities, yields, 'o', label='Observed Yields')
```

```
plt.plot(maturity_smooth, yield_smooth, 'g--', label='Cubic Spline Fit')
plt.xlabel('Maturity (Years)')
plt.ylabel('Yield (%)')
plt.title('US Treasury Yield Curve - Cubic Spline Fit')
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```



The Cubic spline graphical visualize further explain the Nelson Sielgel analysis done above. The shows:

Short End (0.1 to ~ 0.5 years): The curve shows an inverted section where yields initially drop sh This means very short-term interest rates are higher than slightly longer-term short-term rates.

Transition (around 1-2 years): The curve then quickly changes direction, sloping upward.

Mid-to-Long Term (2 to ~ 25 years): The curve generally slopes upward, indicating that longer m command higher yields. This is the more typical "normal" shape, compensating investors for taki more interest rate risk over longer periods.

Very Long End (~25 to 30 years): The curve shows a slight flattening and then a slight decline at longest maturities. This could suggest that for the longest terms, either inflation expectations are there's less demand for extremely long-term debt at very high rates.

Exploiting Correlation

Correlation refers to a statistical relationship between two or more variables. It indicate if two thir linked and how they behave, also how strongly they're linked, and if they move in the same or op directions.

`Gaussian variables`

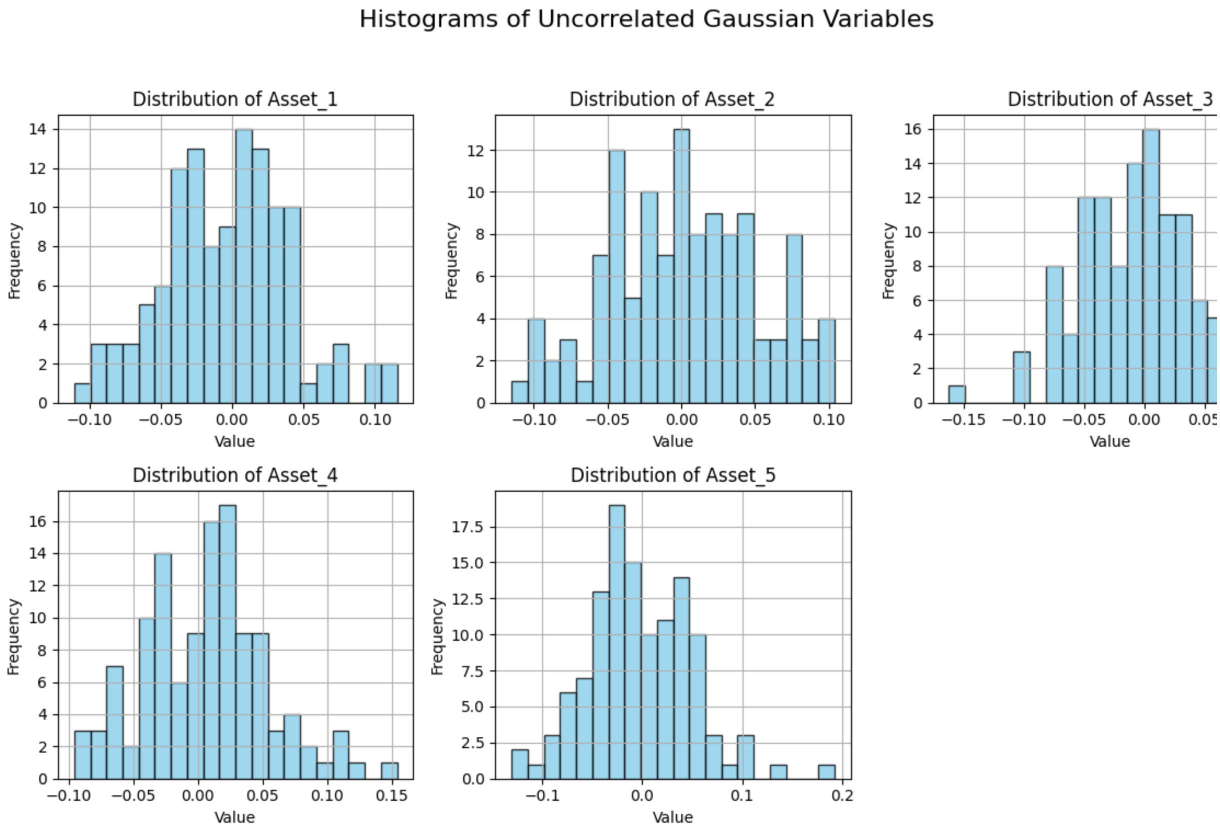
Gaussian variables is a type of data that follows a very specific and common pattern of distributi the Normal Distribution. It's a concept used to understanding how data is spread out, with most clustering around the average and fewer values at the extremes.

Below are block of codes used to Generate 5 uncorrelated Gaussian random variables that simu changes with a mean close to 0 and a standard deviation that is small.

In [8]:
```
# Set random seed for reproducibility
np.random.seed(42)

# Simulate 5 uncorrelated yield changes over 120 days (~6 months)
n_obs = 120
n_vars = 5

# Generate from standard normal distribution (mean ≈ 0, std ≈ 0.05)
data = np.random.normal(loc=0.0, scale=0.05, size=(n_obs, n_vars))

# Put into a DataFrame for easier handling
labels = [f'Asset_{i+1}' for i in range(n_vars)]
uncorrelated_df = pd.DataFrame(data, columns=labels)

# Check the correlation matrix (should be close to identity)
print("Correlation matrix:\n", uncorrelated_df.corr())
```

```
Correlation matrix:
           Asset_1   Asset_2   Asset_3   Asset_4   Asset_5
Asset_1  1.000000 -0.106413  0.031354 -0.040742 -0.120425
Asset_2 -0.106413  1.000000  0.114715  0.048079  0.123577
Asset_3  0.031354  0.114715  1.000000  0.021320  0.074040
Asset_4 -0.040742  0.048079  0.021320  1.000000 -0.029666
Asset_5 -0.120425  0.123577  0.074040 -0.029666  1.000000
```

In [9]:
```
# Set up subplots
plt.figure(figsize=(12, 8))

for i, col in enumerate(uncorrelated_df.columns):
    plt.subplot(2, 3, i + 1)
    plt.hist(uncorrelated_df[col], bins=20, color='skyblue', edgecolor='b
    plt.title(f'Distribution of {col}')
    plt.xlabel('Value')
    plt.ylabel('Frequency')
    plt.grid(True)

plt.suptitle('Histograms of Uncorrelated Gaussian Variables', fontsize=16
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
```



The above graph shows that the bars are generally tallest around the center (close to 0) on the x
which indicates that most of the observed values for each asset are concentrated around the me

The overall shape of each histogram approximates a bell curve, which is characteristic of a Gaus
(normal) distribution. The slight irregularities are due to the random nature of the simulation and
number of observations (120 in this case).

Primarily, these histograms visually confirm that each simulated assets follows the expected Gau
distribution, with their values primarily centered around zero.

Principal Component Analysis (PCA)

PCA is a statistical technique used for dimensionality reduction and data visualization. It is mainl
transform a set of possibly correlated variables into a set of linearly uncorrelated variables callec
principal components (PCs).

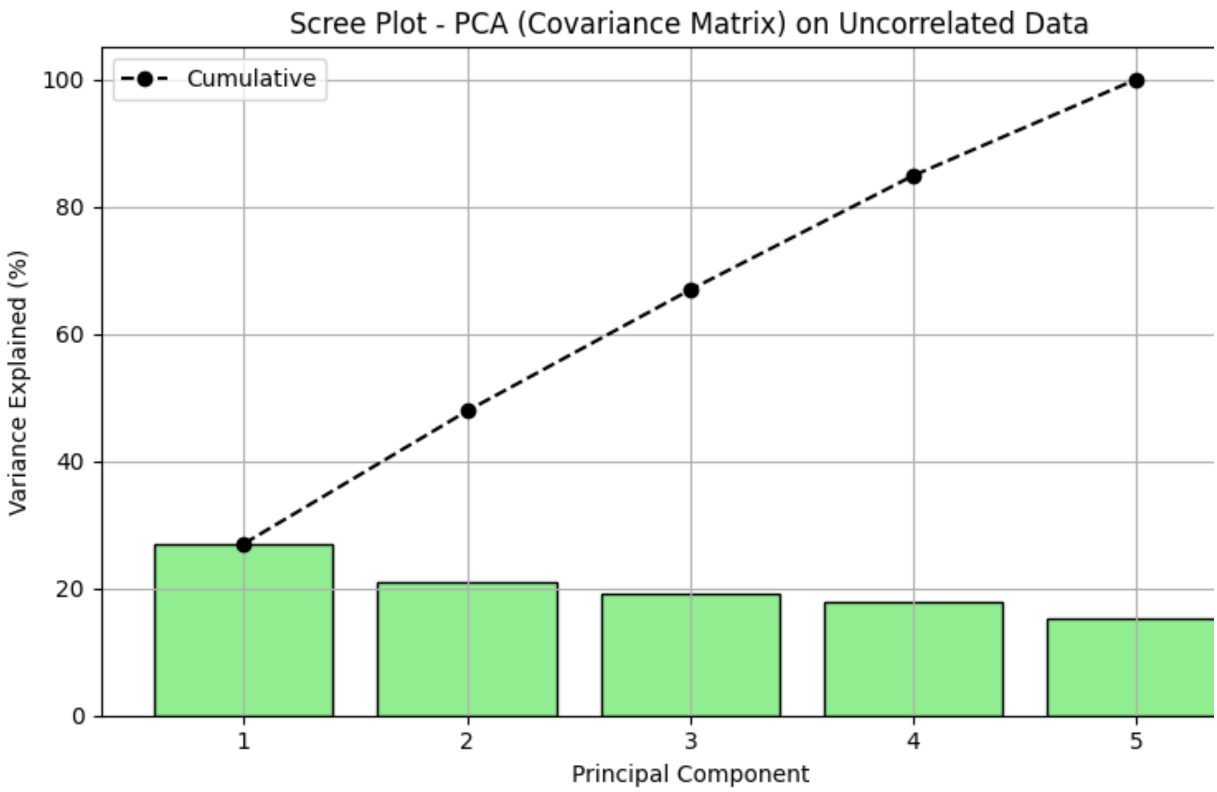
PCA using the covariance matrix help to find the directions (eigenvectors) along which the data v
most, and then using these directions to create a new, lower-dimensional representation of the d
retains the most important information.

In [10]:
```
# Fit PCA
pca = PCA()
pca.fit(uncorrelated_df)

# Explained variance ratios
explained_var_ratio = pca.explained_variance_ratio_

# Display variance explained
```

```
for i, ratio in enumerate(explained_var_ratio):
    print(f"Component {i+1}: {ratio:.4f} ({ratio * 100:.2f}%) of total va
```

```
Component 1: 0.2698 (26.98%) of total variance
Component 2: 0.2087 (20.87%) of total variance
Component 3: 0.1906 (19.06%) of total variance
Component 4: 0.1797 (17.97%) of total variance
Component 5: 0.1513 (15.13%) of total variance
```

In [11]:
```
# Visualize with a Scree Plot

components = np.arange(1, len(explained_var_ratio) + 1)

plt.figure(figsize=(8, 5))
plt.bar(components, explained_var_ratio * 100, color='lightgreen', edgeco
plt.plot(components, np.cumsum(explained_var_ratio * 100), 'o--', color='
plt.title('Scree Plot - PCA (Covariance Matrix) on Uncorrelated Data')
plt.xlabel('Principal Component')
plt.ylabel('Variance Explained (%)')
plt.xticks(components)
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```



In the provided Scree Plot, we can observe the individual contributions of the first three principal components to the total variance. Component 1 explains approximately 27% of the total variance it the single largest contributor to the data's spread. Following this, Component 2 accounts for ab of the variance, and Component 3 explains roughly 20%. These figures highlight that while Com captures the most variance, the subsequent components also contribute significantly and somev evenly, which is characteristic of PCA applied to uncorrelated data where variance is distributed multiple dimensions rather than concentrated in a few.

Using Real-time Data

In [12]:
```
# Step 1: Define date range (last 6 months)
end = datetime.datetime.today()
start = end - datetime.timedelta(days=6*30)  # approx 6 months

# Step 2: Choose 5 maturities
tickers = {
    '3M': 'DGS3MO',
    '1Y': 'DGS1',
    '2Y': 'DGS2',
    '5Y': 'DGS5',
    '10Y': 'DGS10'
}

# Step 3: Fetch data from FRED
yields_df = pd.DataFrame()
for label, fred_code in tickers.items():
    yields_df[label] = web.DataReader(fred_code, 'fred', start, end)

# Step 4: Drop missing values (weekends, holidays)
yields_df.dropna(inplace=True)

# Step 5: Compute daily yield changes
yield_changes = yields_df.diff().dropna()

# Step 6: Display results
print("Daily Closing Yields (last 5 rows):")
print(yields_df.tail())

print("\nDaily Yield Changes (last 5 rows):")
print(yield_changes.tail())
```

```
Daily Closing Yields (last 5 rows):
              3M    1Y    2Y    5Y   10Y
DATE
2025-06-26  4.39  3.96  3.70  3.79  4.26
2025-06-27  4.39  3.97  3.73  3.83  4.29
2025-06-30  4.41  3.96  3.72  3.79  4.24
2025-07-01  4.40  3.98  3.78  3.84  4.26
2025-07-02  4.41  3.99  3.78  3.87  4.30

Daily Yield Changes (last 5 rows):
              3M    1Y    2Y    5Y   10Y
DATE
2025-06-26  0.01 -0.03 -0.04 -0.04 -0.03
2025-06-27  0.00  0.01  0.03  0.04  0.03
2025-06-30  0.02 -0.01 -0.01 -0.04 -0.05
2025-07-01 -0.01  0.02  0.06  0.05  0.02
2025-07-02  0.01  0.01  0.00  0.03  0.04
```

In [13]:
```
pca = PCA()
pca.fit(yield_changes)  # sklearn automatically standardizes if needed

explained_var_ratio = pca.explained_variance_ratio_
loadings = pd.DataFrame(pca.components_, columns=yield_changes.columns)

print("Explained Variance Ratio:")
print(explained_var_ratio)
```

```python
# Cumulative explained variance
cumulative = np.cumsum(explained_var_ratio)
print("\nCumulative Explained Variance:")
print(cumulative)
```

```
Explained Variance Ratio:
[0.88227179 0.08250314 0.01570318 0.01247434 0.00704754]

Cumulative Explained Variance:
[0.88227179 0.96477494 0.98047812 0.99295246 1.        ]
```

A major outcome of this analysis is the obvious dominance of the first principal component (PC1 accounting for almost 90% of the yield movements. This infers that a single common factor (whic be the overall level of interest rates) drives majority of changes across all 5 US government secu When interest rates move, they tend to move all maturities in the same direction.

Also, the fact that the first two principal components explain nearly 96.5% of the total variance is significant. This implies that for practical purposes (e.g., risk management, hedging, or modeling could effectively reduce the dimensionality of our data from 5 variables to just 2 principal compo (level and slope) without losing much information. The remaining 3 components contribute very l the overall variability.

Basicaly, this PCA reveals that the complex movements of 5 different Treasury yields can be ver effectively summarized by just a couple of underlying, uncorrelated factors.

In [14]:
```python
print("\nFirst 3 Principal Components (Loadings):")
print(loadings.iloc[:3])
```

```
First 3 Principal Components (Loadings):
         3M        1Y        2Y        5Y       10Y
0  0.052497  0.323017  0.538801  0.592132  0.501974
1 -0.088528 -0.529913 -0.481035  0.164025  0.673094
2  0.979936 -0.164655  0.047466 -0.086177  0.054178
```

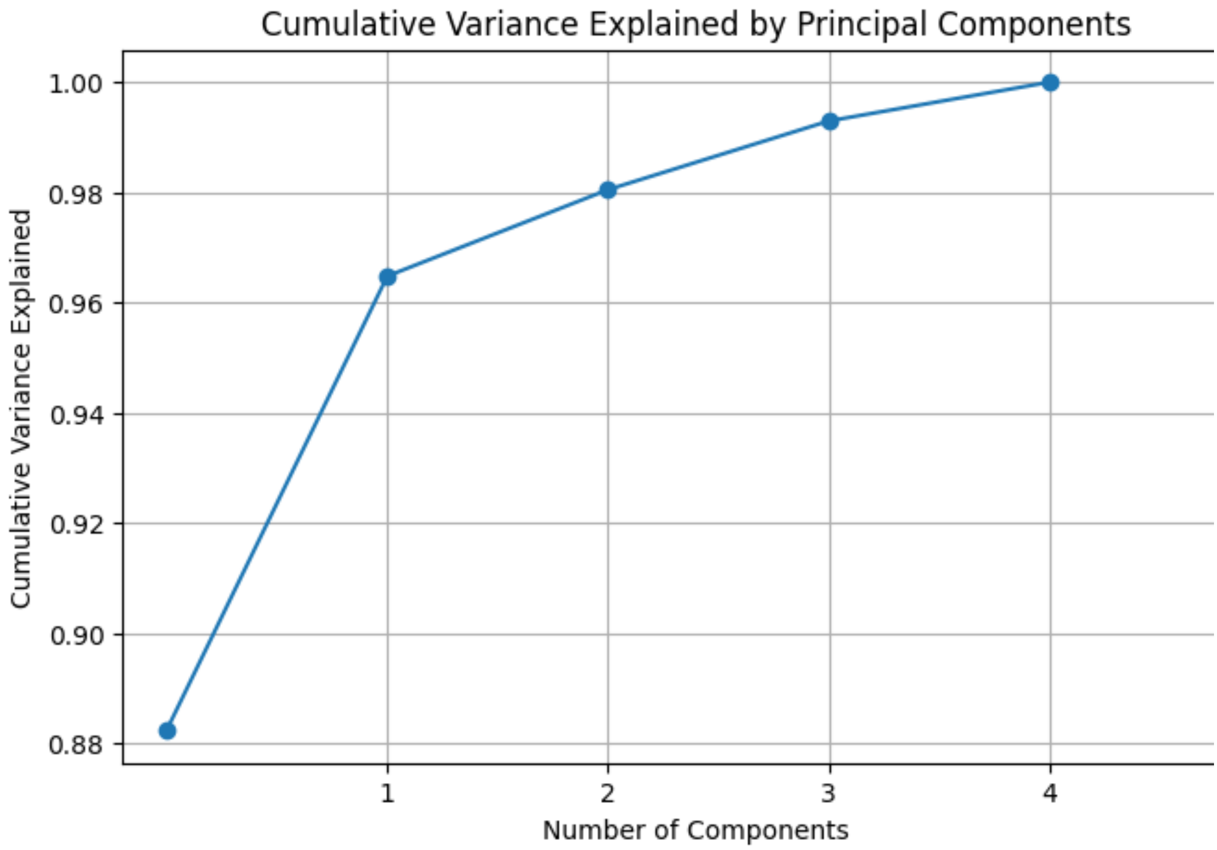
This output shows a matrix where the rows represent the principal components (0, 1, 2, correspo PC1, PC2, PC3) and columns represent the original yield maturities (3M, 1Y, 2Y, 5Y, 10Y).

Principal Component 1 (Row 0) The "Level" Factor: This component represents a parallel shift in curve. When PC1 increases, all maturities tend to increase together, and when it decreases, all maturities tend to decrease together. The higher positive loadings on the longer maturities (2Y, 5 suggest that these maturities are slightly more sensitive to this overall level change compared to short end (3M). This is the dominant factor, explaining almost 90% of the variance, as seen in yo previous output.

Principal Component 2 (Row 1) The "Slope" Factor: This component primarily captures changes slope or steepness of the yield curve. When PC2 increases, long-term yields (e.g., 10Y) tend to while short-to-medium-term yields (e.g., 1Y, 2Y) tend to decrease. This causes the yield curve to steepen. This infers that a decrease in PC2 would lead to a flattening of the curve. This factor ex about 8.25% of the variance.

Principal Component 3 (Row 2) The "Curvature" Factor: This component is typically interpreted a "curvature" or "butterfly" factor. It describes movements where the short end (3M) moves signific one direction, while the long end (10Y) moves slightly in the same direction, and the middle (1Y, moves in the opposite direction or very little. A strong positive loading on 3M and relatively small elsewhere suggests that this component primarily drives the short end of the curve relative to the This factor explains a much smaller portion of the variance (about 1.57%).

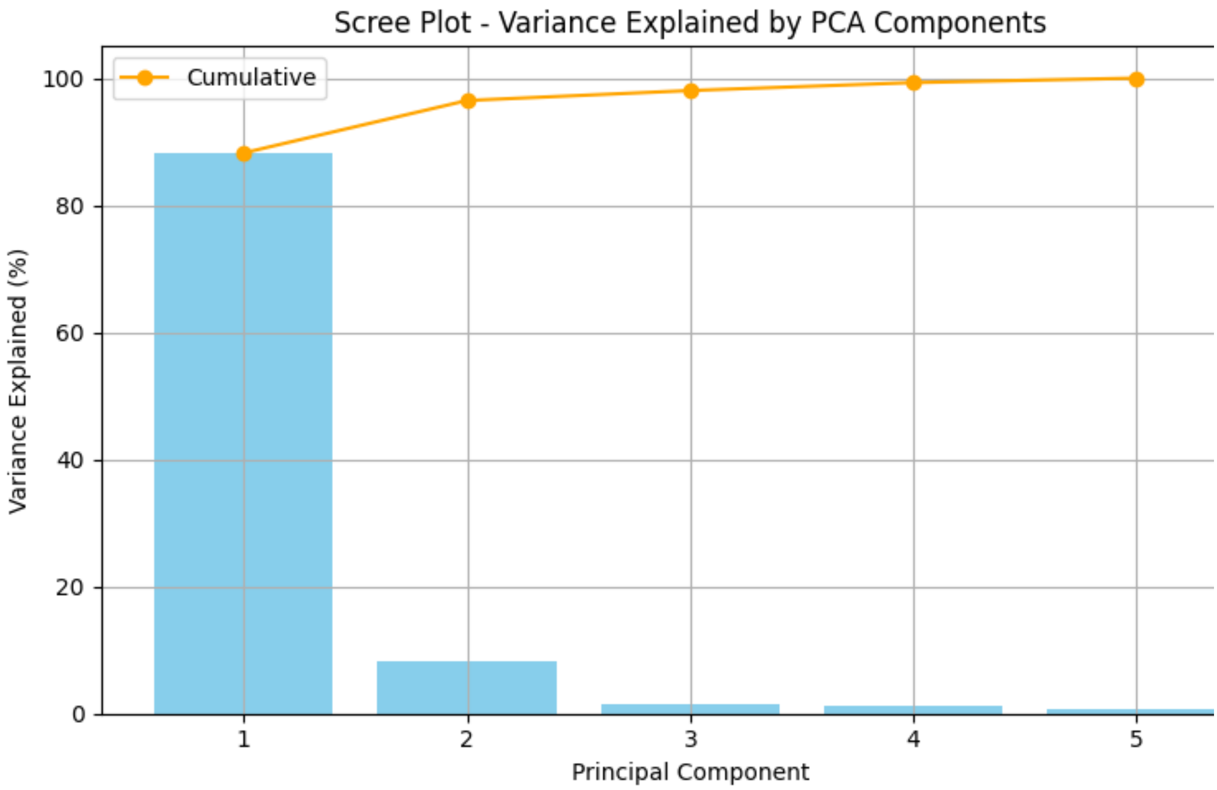
In [15]:
```
plt.figure(figsize=(8, 5))
plt.plot(cumulative, marker='o')
plt.title('Cumulative Variance Explained by Principal Components')
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Variance Explained')
plt.grid(True)
plt.xticks(range(1, len(explained_var_ratio)+1))
plt.show()
```



This is a visual expression for what have been previously explained

In [25]:
```
# Fit PCA again if needed
pca = PCA()
pca.fit(yield_changes)

# Get variance explained
explained_var_ratio = pca.explained_variance_ratio_
components = np.arange(1, len(explained_var_ratio)+1)

# Plot scree plot
plt.figure(figsize=(8, 5))
plt.bar(components, explained_var_ratio * 100, color='skyblue')
plt.plot(components, np.cumsum(explained_var_ratio * 100), marker='o', co
plt.xlabel('Principal Component')
plt.ylabel('Variance Explained (%)')
plt.title('Scree Plot - Variance Explained by PCA Components')
plt.xticks(components)
```

```
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```



Comparison between the Scree Plot from th uncorrelated Gaussian data and the Scree I from the US government securities (Treasu maturities)

The Scree Plot comparison highlights a stark contrast in data structure. For uncorrelated data, va is evenly distributed across all principal components, resulting in a linear cumulative curve and n "elbow." This indicates no dominant underlying factors. Conversely, the government securities da shows variance highly concentrated in the first principal component (explaining ~88-89%), with a dramatic drop-off thereafter. This creates a pronounced "elbow" after PC1, signifying strong corr and that most yield movements are driven by just one or two key factors (level and slope), allowi significant dimensionality reduction.

Empirical Analysis of ETFs

In [26]:
```
# 1. Get Top 30 Holdings from StockAnalysis.com
def get_xlre_holdings():
    url = 'https://stockanalysis.com/etf/xlre/holdings/'
    headers = {'User-Agent': 'Mozilla/5.0'}

    response = requests.get(url, headers=headers)
    soup = BeautifulSoup(response.text, 'lxml')

    table = soup.find('table')
    df = pd.read_html(str(table))[0]
```

```python
    # Keep only top 30
    df_top30 = df.head(30)
    return df_top30

# 2. Download 6-Month Daily Price Data (~120 trading days)
def get_xlre_price_history():
    xlre = yf.Ticker('XLRE')
    df_price = xlre.history(period='6mo')  # ~120 trading days
    return df_price

# 3. Plotting helper
def plot_price(df_price):
    df_price['Close'].plot(figsize=(10, 5), title="XLRE - Last 6 Months D
    plt.xlabel("Date")
    plt.ylabel("Price ($)")
    plt.grid(True)
    plt.show()

# Run All
holdings_df = get_xlre_holdings()
price_df = get_xlre_price_history()

# Display outputs
print("  Top 30 XLRE Holdings:")
print(holdings_df)

print("\n  XLRE Price Data (Last 6 Months):")
print(price_df.head())

# Plot price trend
plot_price(price_df)
```

```
/tmp/ipython-input-26-4063072089.py:10: FutureWarning: Passing literal html to
'read_html' is deprecated and will be removed in a future version. To read fro
literal string, wrap it in a 'StringIO' object.
  df = pd.read_html(str(table))[0]
```

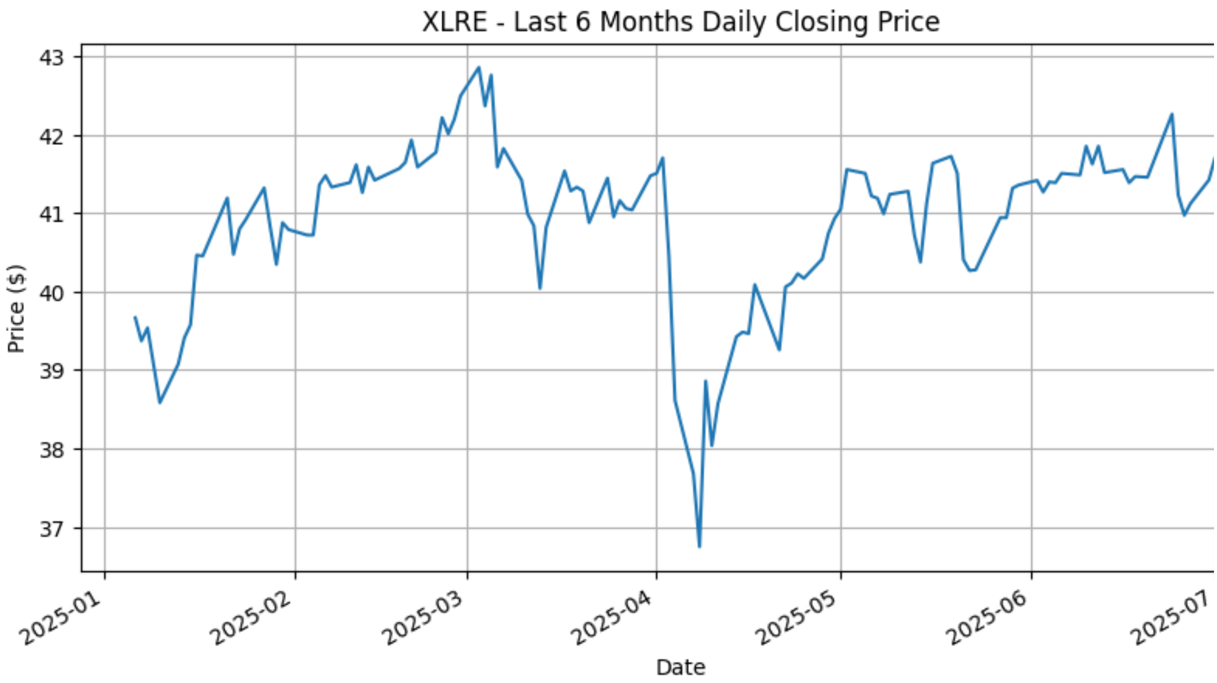
Top 30 XLRE Holdings:

	No.	Symbol	Name	% Weight	Shares
0	1	AMT	American Tower Corporation	9.55%	3231302
1	2	PLD	Prologis, Inc.	9.22%	6405025
2	3	WELL	Welltower Inc.	8.73%	4288246
3	4	EQIX	Equinix, Inc.	7.08%	675255
4	5	DLR	Digital Realty Trust, Inc.	5.02%	2185316
5	6	O	Realty Income Corporation	4.79%	6233805
6	7	SPG	Simon Property Group, Inc.	4.70%	2118026
7	8	PSA	Public Storage	4.30%	1089884
8	9	CCI	Crown Castle Inc.	4.12%	3005960
9	10	CBRE	CBRE Group, Inc.	3.87%	2027088
10	11	VICI	VICI Properties Inc.	3.25%	7294699
11	12	CSGP	CoStar Group, Inc.	3.19%	2912365
12	13	EXR	Extra Space Storage Inc.	2.98%	1464964
13	14	IRM	Iron Mountain Incorporated	2.73%	2036822
14	15	AVB	AvalonBay Communities, Inc.	2.65%	981631
15	16	VTR	Ventas, Inc.	2.61%	3115541
16	17	SBAC	SBA Communications Corporation	2.31%	741840
17	18	EQR	Equity Residential	2.10%	2360753
18	19	WY	Weyerhaeuser Company	1.76%	5007303
19	20	INVH	Invitation Homes Inc.	1.71%	3935623
20	21	ESS	Essex Property Trust, Inc.	1.68%	444477
21	22	MAA	Mid-America Apartment Communities, Inc.	1.62%	808177
22	23	KIM	Kimco Realty Corporation	1.35%	4670996
23	24	DOC	Healthpeak Properties, Inc.	1.16%	4788063
24	25	UDR	UDR, Inc.	1.13%	2077801
25	26	CPT	Camden Property Trust	1.12%	736362
26	27	ARE	Alexandria Real Estate Equities, Inc.	1.09%	1061073
27	28	REG	Regency Centers Corporation	1.06%	1126027
28	29	HST	Host Hotels & Resorts, Inc.	1.03%	4780997
29	30	BXP	BXP, Inc.	0.92%	1003867

XLRE Price Data (Last 6 Months):

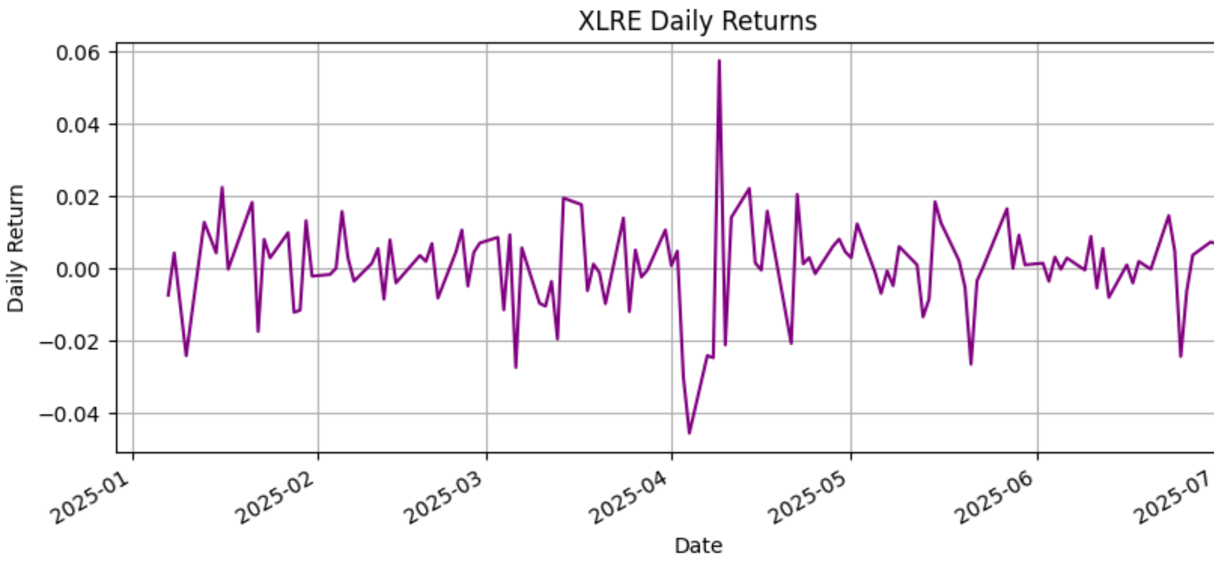
Date	Open	High	Low	Close \
2025-01-06 00:00:00-05:00	40.236159	40.364175	39.605931	39.665012
2025-01-07 00:00:00-05:00	39.802875	40.014593	39.276044	39.369595
2025-01-08 00:00:00-05:00	39.399133	39.556690	39.010167	39.536999
2025-01-10 00:00:00-05:00	38.916619	39.015091	38.547345	38.581810
2025-01-13 00:00:00-05:00	38.532575	39.083038	38.453794	39.074177

Date	Volume	Dividends	Stock Splits	Capital Gains
2025-01-06 00:00:00-05:00	5980200	0.0	0.0	0.0
2025-01-07 00:00:00-05:00	8867700	0.0	0.0	0.0
2025-01-08 00:00:00-05:00	6312800	0.0	0.0	0.0
2025-01-10 00:00:00-05:00	8343100	0.0	0.0	0.0
2025-01-13 00:00:00-05:00	5769300	0.0	0.0	0.0



In [19]:
```
# 4. Compute Daily Returns
def compute_daily_returns(price_df):
    daily_returns = price_df['Close'].pct_change().dropna()  # % change f
    return daily_returns
# Existing code
holdings_df = get_xlre_holdings()
price_df = get_xlre_price_history()

# Compute daily returns
returns_series = compute_daily_returns(price_df)

# Display
print("\n  Daily Returns (First 5 values):")
print(returns_series.head())

# Optional: Plot returns
returns_series.plot(figsize=(10, 4), title="XLRE Daily Returns", color='p
plt.xlabel("Date")
plt.ylabel("Daily Return")
plt.grid(True)
plt.show()
```

```
/tmp/ipython-input-18-3843035308.py:10: FutureWarning: Passing literal html to
'read_html' is deprecated and will be removed in a future version. To read fro
literal string, wrap it in a 'StringIO' object.
  df = pd.read_html(str(table))[0]
  Daily Returns (First 5 values):
Date
2025-01-07 00:00:00-05:00   -0.007448
2025-01-08 00:00:00-05:00    0.004252
2025-01-10 00:00:00-05:00   -0.024159
2025-01-13 00:00:00-05:00    0.012762
2025-01-14 00:00:00-05:00    0.008569
Name: Close, dtype: float64
```



In [20]:
```
# 1. Get top 30 XLRE holdings
def get_top_holdings(n=30):
    url = 'https://stockanalysis.com/etf/xlre/holdings/'
    headers = {'User-Agent': 'Mozilla/5.0'}
    response = requests.get(url, headers=headers)
    soup = BeautifulSoup(response.text, 'lxml')
    table = soup.find('table')
    df = pd.read_html(str(table))[0]
    return df['Symbol'].head(n).tolist()

# 2. Download price data and compute returns
def get_returns(tickers):
    data = yf.download(tickers, period="6mo")['Close']
    daily_returns = data.pct_change().dropna()
    return daily_returns

# 3. Compute Covariance Matrix
def compute_covariance_matrix(returns_df):
    return returns_df.cov()

# Run all steps
tickers = get_top_holdings(n=30)
returns_df = get_returns(tickers)
cov_matrix = compute_covariance_matrix(returns_df)

# Display result
print("  Covariance Matrix of Daily Returns (Top 30 XLRE Holdings):")
print(cov_matrix.round(6))
```

```
/tmp/ipython-input-20-1321677285.py:8: FutureWarning: Passing literal html to
'read_html' is deprecated and will be removed in a future version. To read fr
literal string, wrap it in a 'StringIO' object.
  df = pd.read_html(str(table))[0]
/tmp/ipython-input-20-1321677285.py:13: FutureWarning: YF.download() has chang
argument auto_adjust default to True
  data = yf.download(tickers, period="6mo")['Close']
[*********************100%***********************]  30 of 30 completed
```

Covariance Matrix of Daily Returns (Top 30 XLRE Holdings):

Ticker	AMT	ARE	AVB	BXP	CBRE	CCI	CPT
Ticker							
AMT	0.000317	0.000105	0.000092	0.000066	0.000041	0.000272	0.000101
ARE	0.000105	0.000459	0.000231	0.000315	0.000286	0.000148	0.000224
AVB	0.000092	0.000231	0.000245	0.000237	0.000231	0.000121	0.000215
BXP	0.000066	0.000315	0.000237	0.000477	0.000325	0.000101	0.000221
CBRE	0.000041	0.000286	0.000231	0.000325	0.000461	0.000072	0.000211
CCI	0.000272	0.000148	0.000121	0.000101	0.000072	0.000365	0.000133
CPT	0.000101	0.000224	0.000215	0.000221	0.000211	0.000133	0.000237
CSGP	0.000046	0.000224	0.000151	0.000278	0.000275	0.000082	0.000149
DLR	0.000019	0.000170	0.000130	0.000216	0.000254	0.000035	0.000114
DOC	0.000120	0.000248	0.000176	0.000208	0.000206	0.000148	0.000163
EQIX	0.000064	0.000167	0.000153	0.000218	0.000261	0.000060	0.000139
EQR	0.000105	0.000238	0.000248	0.000249	0.000239	0.000130	0.000226
ESS	0.000106	0.000258	0.000246	0.000245	0.000241	0.000146	0.000236
EXR	0.000144	0.000237	0.000199	0.000218	0.000203	0.000162	0.000184
HST	-0.000016	0.000283	0.000210	0.000338	0.000301	0.000028	0.000189
INVH	0.000124	0.000187	0.000167	0.000172	0.000136	0.000153	0.000168
IRM	0.000078	0.000247	0.000195	0.000275	0.000300	0.000080	0.000193
KIM	0.000053	0.000261	0.000199	0.000282	0.000255	0.000095	0.000186
MAA	0.000095	0.000199	0.000200	0.000180	0.000179	0.000126	0.000204
O	0.000126	0.000133	0.000123	0.000122	0.000097	0.000152	0.000114
PLD	0.000071	0.000295	0.000234	0.000326	0.000329	0.000092	0.000212
PSA	0.000147	0.000188	0.000170	0.000173	0.000164	0.000164	0.000155
REG	0.000089	0.000198	0.000183	0.000204	0.000207	0.000111	0.000170
SBAC	0.000287	0.000099	0.000107	0.000064	0.000053	0.000271	0.000113
SPG	0.000020	0.000281	0.000243	0.000306	0.000334	0.000049	0.000220
UDR	0.000115	0.000240	0.000237	0.000240	0.000231	0.000148	0.000232
VICI	0.000130	0.000186	0.000147	0.000168	0.000139	0.000148	0.000139
VTR	0.000114	0.000135	0.000144	0.000127	0.000152	0.000111	0.000128
WELL	0.000128	0.000130	0.000148	0.000131	0.000150	0.000133	0.000134
WY	0.000106	0.000272	0.000219	0.000260	0.000274	0.000128	0.000203

Ticker	CSGP	DLR	DOC	...	PLD	PSA	REG	\
Ticker				...				
AMT	0.000046	0.000019	0.000120	...	0.000071	0.000147	0.000089	
ARE	0.000224	0.000170	0.000248	...	0.000295	0.000188	0.000198	
AVB	0.000151	0.000130	0.000176	...	0.000234	0.000170	0.000183	
BXP	0.000278	0.000216	0.000208	...	0.000326	0.000173	0.000204	
CBRE	0.000275	0.000254	0.000206	...	0.000329	0.000164	0.000207	
CCI	0.000082	0.000035	0.000148	...	0.000092	0.000164	0.000111	
CPT	0.000149	0.000114	0.000163	...	0.000212	0.000155	0.000170	
CSGP	0.000482	0.000161	0.000155	...	0.000222	0.000107	0.000138	
DLR	0.000161	0.000393	0.000094	...	0.000230	0.000117	0.000126	
DOC	0.000155	0.000094	0.000234	...	0.000205	0.000157	0.000145	
EQIX	0.000136	0.000316	0.000099	...	0.000262	0.000147	0.000144	
EQR	0.000161	0.000138	0.000181	...	0.000250	0.000177	0.000192	
ESS	0.000169	0.000136	0.000184	...	0.000247	0.000178	0.000194	
EXR	0.000139	0.000124	0.000177	...	0.000251	0.000215	0.000173	
HST	0.000271	0.000214	0.000169	...	0.000333	0.000159	0.000195	
INVH	0.000087	0.000084	0.000133	...	0.000171	0.000151	0.000144	
IRM	0.000212	0.000347	0.000137	...	0.000279	0.000151	0.000170	
KIM	0.000205	0.000161	0.000181	...	0.000277	0.000161	0.000195	
MAA	0.000119	0.000085	0.000153	...	0.000184	0.000151	0.000152	
O	0.000070	0.000060	0.000125	...	0.000131	0.000130	0.000108	
PLD	0.000222	0.000230	0.000205	...	0.000444	0.000207	0.000220	
PSA	0.000107	0.000117	0.000157	...	0.000207	0.000212	0.000150	
REG	0.000138	0.000126	0.000145	...	0.000220	0.000150	0.000195	
SBAC	0.000058	0.000015	0.000124	...	0.000074	0.000153	0.000103	

```
SPG     0.000211  0.000231  0.000182  ...  0.000343  0.000173  0.000216
UDR     0.000158  0.000129  0.000184  ...  0.000242  0.000181  0.000191
VICI    0.000123  0.000082  0.000148  ...  0.000187  0.000143  0.000133
VTR     0.000078  0.000087  0.000152  ...  0.000137  0.000120  0.000116
WELL    0.000064  0.000106  0.000135  ...  0.000156  0.000133  0.000134
WY      0.000193  0.000128  0.000205  ...  0.000294  0.000190  0.000189

Ticker      SBAC       SPG       UDR      VICI       VTR      WELL        WY
Ticker
AMT     0.000287  0.000020  0.000115  0.000130  0.000114  0.000128  0.000106
ARE     0.000099  0.000281  0.000240  0.000186  0.000135  0.000130  0.000272
AVB     0.000107  0.000243  0.000237  0.000147  0.000144  0.000148  0.000219
BXP     0.000064  0.000306  0.000240  0.000168  0.000127  0.000131  0.000260
CBRE    0.000053  0.000334  0.000231  0.000139  0.000152  0.000150  0.000274
CCI     0.000271  0.000049  0.000148  0.000148  0.000111  0.000133  0.000128
CPT     0.000113  0.000220  0.000232  0.000139  0.000128  0.000134  0.000203
CSGP    0.000058  0.000211  0.000158  0.000123  0.000078  0.000064  0.000193
DLR     0.000015  0.000231  0.000129  0.000082  0.000087  0.000106  0.000128
DOC     0.000124  0.000182  0.000184  0.000148  0.000152  0.000135  0.000205
EQIX    0.000068  0.000230  0.000154  0.000102  0.000103  0.000125  0.000164
EQR     0.000117  0.000257  0.000254  0.000156  0.000141  0.000149  0.000235
ESS     0.000115  0.000262  0.000264  0.000150  0.000135  0.000153  0.000235
EXR     0.000150  0.000217  0.000210  0.000163  0.000130  0.000143  0.000217
HST     0.000008  0.000337  0.000216  0.000137  0.000075  0.000065  0.000275
INVH    0.000118  0.000159  0.000193  0.000123  0.000095  0.000119  0.000163
IRM     0.000068  0.000280  0.000208  0.000116  0.000077  0.000137  0.000199
KIM     0.000063  0.000277  0.000211  0.000144  0.000088  0.000116  0.000229
MAA     0.000108  0.000191  0.000214  0.000126  0.000126  0.000131  0.000182
O       0.000138  0.000108  0.000132  0.000128  0.000120  0.000112  0.000127
PLD     0.000074  0.000343  0.000242  0.000187  0.000137  0.000156  0.000294
PSA     0.000153  0.000173  0.000181  0.000143  0.000120  0.000133  0.000190
REG     0.000103  0.000216  0.000191  0.000133  0.000116  0.000134  0.000189
SBAC    0.000314  0.000035  0.000120  0.000135  0.000125  0.000134  0.000107
SPG     0.000035  0.000413  0.000246  0.000141  0.000138  0.000144  0.000274
UDR     0.000120  0.000246  0.000269  0.000154  0.000146  0.000159  0.000231
VICI    0.000135  0.000141  0.000154  0.000177  0.000133  0.000124  0.000160
VTR     0.000125  0.000138  0.000146  0.000133  0.000271  0.000199  0.000139
WELL    0.000134  0.000144  0.000159  0.000124  0.000199  0.000234  0.000139
WY      0.000107  0.000274  0.000231  0.000160  0.000139  0.000139  0.000353

[30 rows x 30 columns]
```

In [21]:

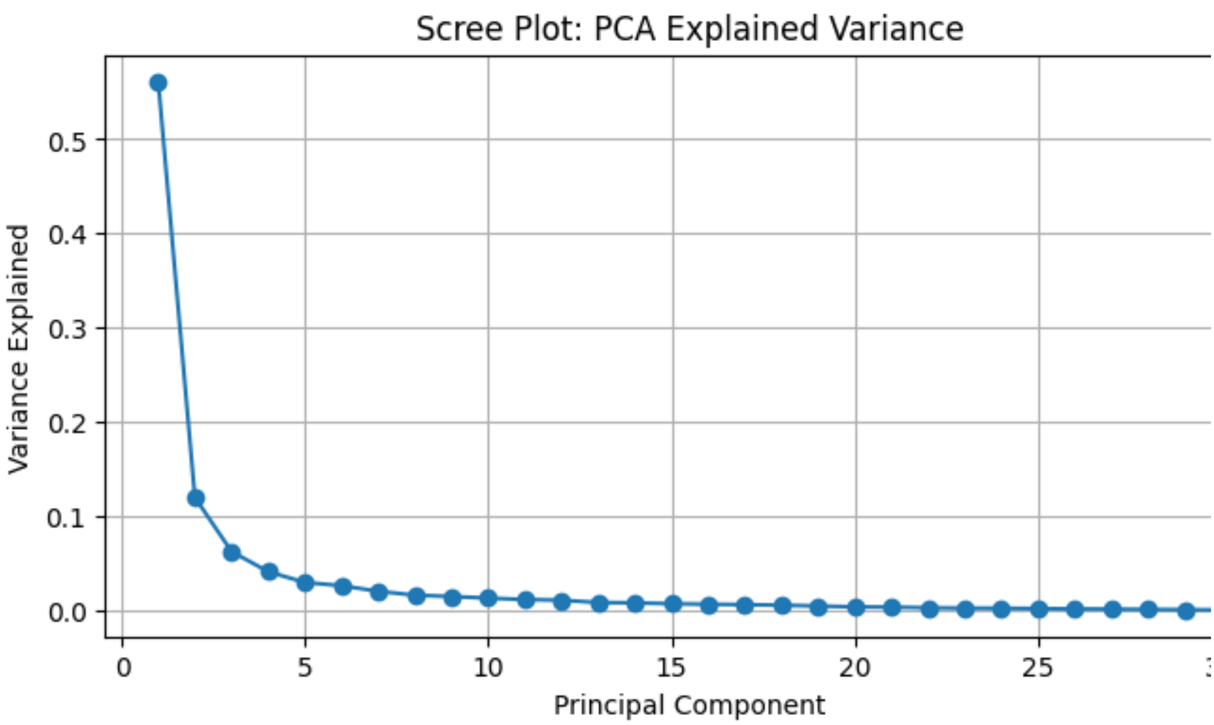
```python
# 4. Center the data (mean zero for each stock)
returns_centered = returns_df - returns_df.mean()

# 5. Apply PCA
pca = PCA()
pca.fit(returns_centered)

# Explained variance
explained_variance = pca.explained_variance_ratio_

# Print explained variance
print("  PCA Explained Variance Ratio (Top Components):")
for i, var in enumerate(explained_variance[:10]):
    print(f"Component {i+1}: {var:.4f}")
```

```
  PCA Explained Variance Ratio (Top Components):
Component 1: 0.5615
Component 2: 0.1200
Component 3: 0.0626
Component 4: 0.0412
Component 5: 0.0298
Component 6: 0.0264
Component 7: 0.0205
Component 8: 0.0163
Component 9: 0.0149
Component 10: 0.0132
```

In [22]:
```python
# 6. SVD (Singular Value Decomposition)
U, S, VT = np.linalg.svd(returns_centered, full_matrices=False)

print("\n  SVD Results:")
print(f"U shape: {U.shape}, S shape: {S.shape}, VT shape: {VT.shape}")

# Optional: Compare PCA eigenvalues to SVD singular values
print("\nFirst 5 Singular Values from SVD:")
print(S[:5])
```

```
  SVD Results:
U shape: (122, 30), S shape: (30,), VT shape: (30, 30)

First 5 Singular Values from SVD:
[0.81213674 0.37541122 0.27114445 0.2200053  0.18723302]
```

In [23]:
```python
import matplotlib.pyplot as plt

plt.figure(figsize=(8, 4))
plt.plot(np.arange(1, len(explained_variance)+1), explained_variance, mar
plt.title('Scree Plot: PCA Explained Variance')
plt.xlabel('Principal Component')
plt.ylabel('Variance Explained')
plt.grid(True)
plt.show()
```



References

C.W. Fisher et al. Criticality of data quality as exemplified in two disasters Information Managem (2001).

De Boor, C. (1978). A Practical Guide to Splines. *Springer-Verlag*.

Federal Reserve Bank of St. Louis. (n.d.). FRED – U.S. Treasury Constant Maturity Rates. Retri from https://fred.stlouisfed.org. *Accessed via: pandas_datareader Python library, using FRED da*

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). An introduction to statistical learning wi applications in R. Springer.

Hastie, T., Tibshirani, R., & Friedman, J. (2009). The elements of statistical learning: Data mining inference, and prediction (2nd ed.). Springer.

Tsay, R. S. (2005). Analysis of financial time series (2nd ed.). John Wiley & Sons.

Rendleman, R. J., Jr. (2006). Fixed income analytics. John Wiley & Sons.

Tsay, R. S. (2010). Quantitative finance and risk management: A practical guide. John Wiley & S

Nelson, C. R., & Siegel, A. F. (1987). Parsimonious modeling of yield curves. *Journal of Busines* 473–489.

N. Jenkinson and I. Leonova (2013). The importance of data quality for effective financial stabilit – Legal entity identifier: a first step towards necessary financial data reforms. *Banque de France Financial Stability Review* • No. 17

W. Fan et al. Discovering and reconciling value conflicts for numerical data integration Informatic Systems (2001)

S. McKinley and M. Levine (1998), Cubic Spline Interpolation. *College of the Redwoods*

In [1]:
```
!jupyter nbconvert --to html /content/FD Project1.ipynb
```

```
[NbConvertApp] WARNING | pattern '/content/Copy_of_PO_FD_Project1.ipynb' matc
files
This application is used to convert notebook files (*.ipynb)
        to various other formats.

        WARNING: THE COMMANDLINE INTERFACE MAY CHANGE IN FUTURE RELEASES.

Options
=======
The options below are convenience aliases to configurable class-options,
as listed in the "Equivalent to" description-line of the aliases.
To see all configurable class-options for some <cmd>, use:
    <cmd> --help-all

--debug
    set log level to logging.DEBUG (maximize logging output)
    Equivalent to: [--Application.log_level=10]
--show-config
    Show the application's configuration (human-readable format)
    Equivalent to: [--Application.show_config=True]
--show-config-json
    Show the application's configuration (json format)
    Equivalent to: [--Application.show_config_json=True]
--generate-config
    generate default config file
    Equivalent to: [--JupyterApp.generate_config=True]
-y
    Answer yes to any questions instead of prompting.
    Equivalent to: [--JupyterApp.answer_yes=True]
--execute
    Execute the notebook prior to export.
    Equivalent to: [--ExecutePreprocessor.enabled=True]
--allow-errors
    Continue notebook execution even if one of the cells throws an error and
include the error message in the cell output (the default behaviour is to abo
conversion). This flag is only relevant if '--execute' was specified, too.
    Equivalent to: [--ExecutePreprocessor.allow_errors=True]
--stdin
    read a single notebook file from stdin. Write the resulting notebook with
default basename 'notebook.*'
    Equivalent to: [--NbConvertApp.from_stdin=True]
--stdout
    Write notebook output to stdout instead of files.
    Equivalent to: [--NbConvertApp.writer_class=StdoutWriter]
--inplace
    Run nbconvert in place, overwriting the existing notebook (only
            relevant when converting to notebook format)
    Equivalent to: [--NbConvertApp.use_output_suffix=False --
NbConvertApp.export_format=notebook --FilesWriter.build_directory=]
--clear-output
    Clear output of current file and save in place,
            overwriting the existing notebook.
    Equivalent to: [--NbConvertApp.use_output_suffix=False --
NbConvertApp.export_format=notebook --FilesWriter.build_directory= --
ClearOutputPreprocessor.enabled=True]
--coalesce-streams
    Coalesce consecutive stdout and stderr outputs into one stream (within ea
cell).
    Equivalent to: [--NbConvertApp.use_output_suffix=False --
NbConvertApp.export_format=notebook --FilesWriter.build_directory= --
```

```
CoalesceStreamsPreprocessor.enabled=True]
--no-prompt
    Exclude input and output prompts from converted document.
    Equivalent to: [--TemplateExporter.exclude_input_prompt=True --
TemplateExporter.exclude_output_prompt=True]
--no-input
    Exclude input cells and output prompts from converted document.
            This mode is ideal for generating code-free reports.
    Equivalent to: [--TemplateExporter.exclude_output_prompt=True --
TemplateExporter.exclude_input=True --TemplateExporter.exclude_input_prompt=Tr
--allow-chromium-download
    Whether to allow downloading chromium if no suitable version is found on
system.
    Equivalent to: [--WebPDFExporter.allow_chromium_download=True]
--disable-chromium-sandbox
    Disable chromium security sandbox when converting to PDF..
    Equivalent to: [--WebPDFExporter.disable_sandbox=True]
--show-input
    Shows code input. This flag is only useful for dejavu users.
    Equivalent to: [--TemplateExporter.exclude_input=False]
--embed-images
    Embed the images as base64 dataurls in the output. This flag is only usef
the HTML/WebPDF/Slides exports.
    Equivalent to: [--HTMLExporter.embed_images=True]
--sanitize-html
    Whether the HTML in Markdown cells and cell outputs should be sanitized..
    Equivalent to: [--HTMLExporter.sanitize_html=True]
--log-level=<Enum>
    Set the log level by value or name.
    Choices: any of [0, 10, 20, 30, 40, 50, 'DEBUG', 'INFO', 'WARN', 'ERROR',
'CRITICAL']
    Default: 30
    Equivalent to: [--Application.log_level]
--config=<Unicode>
    Full path of a config file.
    Default: ''
    Equivalent to: [--JupyterApp.config_file]
--to=<Unicode>
    The export format to be used, either one of the built-in formats
            ['asciidoc', 'custom', 'html', 'latex', 'markdown', 'notebook', '
'python', 'qtpdf', 'qtpng', 'rst', 'script', 'slides', 'webpdf']
            or a dotted object name that represents the import path for an
            ``Exporter`` class
    Default: ''
    Equivalent to: [--NbConvertApp.export_format]
--template=<Unicode>
    Name of the template to use
    Default: ''
    Equivalent to: [--TemplateExporter.template_name]
--template-file=<Unicode>
    Name of the template file to use
    Default: None
    Equivalent to: [--TemplateExporter.template_file]
--theme=<Unicode>
    Template specific theme(e.g. the name of a JupyterLab CSS theme distribut
    as prebuilt extension for the lab template)
    Default: 'light'
    Equivalent to: [--HTMLExporter.theme]
--sanitize_html=<Bool>
    Whether the HTML in Markdown cells and cell outputs should be sanitized.T
```

```
        should be set to True by nbviewer or similar tools.
        Default: False
        Equivalent to: [--HTMLExporter.sanitize_html]
    --writer=<DottedObjectName>
        Writer class used to write the
                                            results of the conversion
        Default: 'FilesWriter'
        Equivalent to: [--NbConvertApp.writer_class]
    --post=<DottedOrNone>
        PostProcessor class used to write the
                                            results of the conversion
        Default: ''
        Equivalent to: [--NbConvertApp.postprocessor_class]
    --output=<Unicode>
        Overwrite base name use for output files.
                    Supports pattern replacements '{notebook_name}'.
        Default: '{notebook_name}'
        Equivalent to: [--NbConvertApp.output_base]
    --output-dir=<Unicode>
        Directory to write output(s) to. Defaults
                                      to output to the directory of each notebook
    recover
                                      previous default behaviour (outputting to tl
    current
                                      working directory) use . as the flag value.
        Default: ''
        Equivalent to: [--FilesWriter.build_directory]
    --reveal-prefix=<Unicode>
        The URL prefix for reveal.js (version 3.x).
                This defaults to the reveal CDN, but can be any url pointing to a
                of reveal.js.
                For speaker notes to work, this must be a relative path to a loca:
                copy of reveal.js: e.g., "reveal.js".
                If a relative path is given, it must be a subdirectory of the
                current directory (from which the server is run).
                See the usage documentation
                (https://nbconvert.readthedocs.io/en/latest/usage.html#reveal-js-l
    slideshow)
                for more details.
        Default: ''
        Equivalent to: [--SlidesExporter.reveal_url_prefix]
    --nbformat=<Enum>
        The nbformat version to write.
                Use this to downgrade notebooks.
        Choices: any of [1, 2, 3, 4]
        Default: 4
        Equivalent to: [--NotebookExporter.nbformat_version]

    Examples
    --------

        The simplest way to use nbconvert is

                > jupyter nbconvert mynotebook.ipynb --to html

                Options include ['asciidoc', 'custom', 'html', 'latex', 'markdown
    'notebook', 'pdf', 'python', 'qtpdf', 'qtpng', 'rst', 'script', 'slides',
    'webpdf'].

                > jupyter nbconvert --to latex mynotebook.ipynb
```

```
Both HTML and LaTeX support multiple output templates. LaTeX incl
'base', 'article' and 'report'.  HTML includes 'basic', 'lab' and
'classic'. You can specify the flavor of the format used.

> jupyter nbconvert --to html --template lab mynotebook.ipynb

You can also pipe the output to stdout, rather than a file

> jupyter nbconvert mynotebook.ipynb --stdout

PDF is generated via latex

> jupyter nbconvert mynotebook.ipynb --to pdf

You can get (and serve) a Reveal.js-powered slideshow

> jupyter nbconvert myslides.ipynb --to slides --post serve

Multiple notebooks can be given at the command line in a couple o
different ways:

> jupyter nbconvert notebook*.ipynb
> jupyter nbconvert notebook1.ipynb notebook2.ipynb

or you can specify the notebooks list in a config file, containin

    c.NbConvertApp.notebooks = ["my_notebook.ipynb"]

> jupyter nbconvert --config mycfg.py

To see all available configurables, use `--help-all`.
```

In [3]:

```
!apt-get install -y libpango-1.0-0 libpangocairo-1.0-0 libcairo2
!pip install weasyprint
from weasyprint import HTML

# Set paths
html_path = "/content/FD Project1.html"
pdf_path = "/content/Copy_of_PO_FD_Project1.pdf"

# Convert HTML to PDF
HTML(html_path).write_pdf(pdf_path)

print(f"PDF saved to: {pdf_path}")
```

```
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
libcairo2 is already the newest version (1.16.0-5ubuntu2).
libpango-1.0-0 is already the newest version (1.50.6+ds-2ubuntu1).
libpangocairo-1.0-0 is already the newest version (1.50.6+ds-2ubuntu1).
0 upgraded, 0 newly installed, 0 to remove and 35 not upgraded.
Requirement already satisfied: weasyprint in /usr/local/lib/python3.11/dist-
packages (65.1)
Requirement already satisfied: pydyf>=0.11.0 in /usr/local/lib/python3.11/dis
packages (from weasyprint) (0.11.0)
Requirement already satisfied: cffi>=0.6 in /usr/local/lib/python3.11/dist-pa
(from weasyprint) (1.17.1)
Requirement already satisfied: tinyhtml5>=2.0.0b1 in /usr/local/lib/python3.1
dist-packages (from weasyprint) (2.0.0)
Requirement already satisfied: tinycss2>=1.4.0 in /usr/local/lib/python3.11/d
packages (from weasyprint) (1.4.0)
Requirement already satisfied: cssselect2>=0.8.0 in /usr/local/lib/python3.11
packages (from weasyprint) (0.8.0)
Requirement already satisfied: Pyphen>=0.9.1 in /usr/local/lib/python3.11/dis
packages (from weasyprint) (0.17.2)
Requirement already satisfied: Pillow>=9.1.0 in /usr/local/lib/python3.11/dis
packages (from weasyprint) (11.2.1)
Requirement already satisfied: fonttools>=4.0.0 in /usr/local/lib/python3.11/
packages (from fonttools[woff]>=4.0.0->weasyprint) (4.58.4)
Requirement already satisfied: pycparser in /usr/local/lib/python3.11/dist-pa
(from cffi>=0.6->weasyprint) (2.22)
Requirement already satisfied: webencodings in /usr/local/lib/python3.11/dist
packages (from cssselect2>=0.8.0->weasyprint) (0.5.1)
Requirement already satisfied: brotli>=1.0.1 in /usr/local/lib/python3.11/dis
packages (from fonttools[woff]>=4.0.0->weasyprint) (1.1.0)
Requirement already satisfied: zopfli>=0.1.4 in /usr/local/lib/python3.11/dis
packages (from fonttools[woff]>=4.0.0->weasyprint) (0.2.3.post1)
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
/tmp/ipython-input-3-2680607252.py in <cell line: 0>()
      8 
      9 # Convert HTML to PDF
---> 10 HTML(html_path).write_pdf(pdf_path)
     11 
     12 print(f"PDF saved to: {pdf_path}")

/usr/local/lib/python3.11/dist-packages/weasyprint/__init__.py in __init__(se
guess, filename, url, file_obj, string, encoding, base_url, url_fetcher,
media_type)
    168         result = _select_source(
    169             guess, filename, url, file_obj, string, base_url, url_fetc
--> 170         with result as (source_type, source, base_url, protocol_encodi
    171             if isinstance(source, str):
    172                 result = tinyhtml5.parse(source, namespace_html_elemen
lse)

/usr/lib/python3.11/contextlib.py in __enter__(self)
    135         del self.args, self.kwds, self.func
    136         try:
--> 137             return next(self.gen)
    138         except StopIteration:
    139             raise RuntimeError("generator didn't yield") from None

/usr/local/lib/python3.11/dist-packages/weasyprint/__init__.py in _select_sou
ess, filename, url, file_obj, string, base_url, url_fetcher, check_css_mime_ty
    391             check_css_mime_type=check_css_mime_type,
    392             **{type_: guess})
--> 393         with result as result:
    394             yield result
    395     elif filename is not None:

/usr/lib/python3.11/contextlib.py in __enter__(self)
    135         del self.args, self.kwds, self.func
    136         try:
--> 137             return next(self.gen)
    138         except StopIteration:
    139             raise RuntimeError("generator didn't yield") from None

/usr/local/lib/python3.11/dist-packages/weasyprint/__init__.py in _select_sou
ess, filename, url, file_obj, string, base_url, url_fetcher, check_css_mime_ty
    396         if base_url is None:
    397             base_url = path2url(filename)
--> 398         with open(filename, 'rb') as file_obj:
    399             yield 'file_obj', file_obj, base_url, None
    400     elif url is not None:

FileNotFoundError: [Errno 2] No such file or directory: '/content/FD Project1
```

In [ ]: