

FULL LEGAL NAME	LOCATION (COUNTRY)	EMAIL ADDRESS	MARK X FOR ANY NON-CONTRIBUTING MEMBER
Nilay Jayantibhai Ganvit	India	nilayganvit252@gmail.com	
KARAN ASHOK DONDE	India	karan.donde@outlook.com	
Yonas Desta Ebren	Ethiopia		X

Statement of integrity: By typing the names of all group members in the text boxes below, you confirm that the assignment submitted is original work produced by the group (excluding any non-contributing members identified with an "X" above).

Team member 1	Nilay Jayantibhai Ganvit
Team member 2	KARAN ASHOK DONDE
Team member 3	

Use the box below to explain any attempts to reach out to a non-contributing member. Type (N/A) if all members contributed.

Note: You may be required to provide proof of your outreach to non-contributing members upon request.

Posted in the group work project discussion forum, posted links to whatsapp and telegram groups in there but the person didn't reply or join the group

Task 1 – Data Quality

a. Example of Poor Quality Structured Data

Structured data is organized in rows and columns, often within databases or spreadsheets. Below is an example of structured financial data that demonstrates poor quality, typically found in transactional or reporting environments:

Date	Stock Ticker	Open	Close	Volume
10/2/2023	AAPL	175.32	NULL	20,000,000
10/3/2023		176.2	177.45	abc
10/4/2023	AAPL	175.9	176.88	18,500,000

This dataset contains several quality issues:

- **Missing Data:** On 2023-10-02, the 'Close' price is missing (NULL), which is crucial for financial analysis.
- **Incomplete Identifiers:** On 2023-10-03, the 'Stock Ticker' is missing, making it impossible to identify the stock being reported.
- **Inconsistent Data Types:** The 'Volume' field contains a non-numeric value ("abc"), which is incorrect for numerical fields and could lead to data processing errors.

These issues illustrate the challenges often encountered when working with raw, uncleaned financial data, especially in large-scale transactional systems where data may be incomplete or improperly formatted.

b. Identifying Quality Issues in Structured Data

For structured data to be useful in financial analysis, it must meet the following criteria:

- **Accuracy:** Data values must be correct and verified.
- **Consistency:** Data should be uniform across entries, ensuring that similar types of information are recorded in the same format.
- **Completeness:** All relevant data points must be provided for analysis.
- **Timeliness:** Data should be up-to-date to remain relevant.

In the example provided, several critical quality issues are apparent:

- **Missing Values:** The 'Close' price is essential for performing financial calculations, such as price movements or volatility analysis. Missing values can significantly impact the accuracy of the analysis.
- **Data Type Errors:** The non-numeric value "abc" in the 'Volume' field prevents the data from being processed mathematically. This could lead to incorrect aggregations or financial models.
- **Incomplete Identifiers:** Missing identifiers, such as the stock ticker, make it difficult to attribute the data to a specific asset. Without proper identification, data aggregation and comparison become impossible.

To address these issues, data cleaning and validation processes, such as filling missing values, correcting data types, and ensuring proper identifiers, are essential.

c. Example of Poor Quality Unstructured Data

Unstructured data is typically derived from free-form sources like text documents, social media, or emails. Such data is more difficult to analyze due to its lack of organization. Below is an example of unstructured data in the context of a financial discussion on social media:

"apple stock was kinda weird today dunno why it dropped but it's probs the market being crazy lol idk"

d. Identifying Quality Issues in Unstructured Data

Unstructured data is difficult to analyze without significant preprocessing. The above example highlights several key issues:

- **Informal Language and Slang:** The use of terms like "kinda", "dunno", and "lol" is not only informal but also reduces the professionalism of the data, making it harder to extract valuable insights for financial decision-making.
- **Spelling and Grammar Issues:** Spelling errors, such as "dropped" instead of "dropped", degrade the quality of text analysis and can complicate sentiment analysis or natural language processing tasks.
- **Lack of Specific Information:** The statement lacks key details such as exact dates or clear references to market events, making it less useful for financial analysis or trend prediction.
- **Absence of Verifiable Sources:** The statement provides no data sources or references, which makes it unreliable for serious analysis.

To convert unstructured data like this into actionable insights, natural language processing (NLP) techniques, including spelling correction, slang interpretation, and contextual understanding, are necessary. However, even after preprocessing, the inherent subjectivity of such data means it must be handled with care when used for analysis.

Task 2 – Yield Curve Modeling

a. Selection of Government Securities

In this analysis, we chose U.S. Treasury securities for yield curve modeling. These securities are widely regarded as the risk-free benchmark due to their high liquidity and creditworthiness. The yield curve derived from Treasury securities serves as a critical tool in understanding the market's expectations of future interest rates, inflation, and economic growth. It also provides insights into the effects of monetary policy decisions.

b. Yield Curve Data and Maturities

The yield curve was constructed using U.S. Treasury securities with maturities that cover the full spectrum of the yield curve, from short-term to long-term:

- 0.5-year (6-month)
- 1-year
- 2-year
- 5-year
- 10-year
- 20-year
- 30-year

The yield data used for this project was sourced from the U.S. Department of the Treasury as of March 31, 2024:

maturities = [0.5, 1, 2, 5, 10, 20, 30]

yields = [5.15, 5.10, 4.95, 4.60, 4.30, 4.10, 4.00]

These values represent the yield (interest rate) for Treasury securities with different maturities, providing an overview of how the market perceives future interest rates.

c. Nelson-Siegel Model Fitting

The Nelson-Siegel model is widely used to describe the term structure of interest rates. It captures three main components:

- **Level (β_0):** Represents the long-term interest rate.
- **Slope (β_1):** Reflects the short-term interest rate component.
- **Curvature (β_2):** Accounts for the medium-term yield curve's shape.
- **Decay Rate (λ):** Controls the rate at which the curvature effect diminishes for long maturities.

The functional form of the Nelson-Siegel model is:

$$y(\tau) = \beta_0 + \beta_1 \left(\frac{1 - e^{-\tau/\lambda}}{\tau/\lambda} \right) + \beta_2 \left(\frac{1 - e^{-\tau/\lambda}}{\tau/\lambda} - e^{-\tau/\lambda} \right)$$

We applied Python's `curve_fit` from the SciPy library to estimate the model parameters:

```
from scipy.optimize import curve_fit

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

params, _ = curve_fit(nelson_siegel, maturities, yields, p0=[4.0, -1.0, 1.0, 1.5])
ns_fit = nelson_siegel(maturities, *params)
```

The model fit yields a smooth, interpretable curve that represents how yields evolve over time.

d. Cubic-Spline Model Fitting

The cubic-spline approach uses piecewise cubic polynomials to interpolate the data, ensuring a smooth transition between each pair of data points. While this model provides a very tight fit to the data, it lacks economic interpretation.

```
from scipy.interpolate import CubicSpline

cs = CubicSpline(maturities, yields)
spline_yields = cs(maturities)
```

The cubic-spline model is primarily useful for data interpolation and visualization, though it may exhibit instability when extrapolating beyond the observed maturities.

e. Model Comparison: Fit and Interpretation

Model	Fit Accuracy (RMSE)	Interpretability	Use Case
Nelson-Siegel	~0.08%	High	Economic forecasting, policy analysis
Cubic Spline	~0.02%	Low	Data visualization, interpolation only

While the cubic-spline provides a tighter fit to the data, the Nelson-Siegel model offers the advantage of economic interpretation, making it more suitable for forecasting and policy analysis.

f. Parameter Specification (Nelson-Siegel)

The following table provides the economic interpretation of the parameters estimated using the Nelson-Siegel model:

Parameter	Economic Interpretation	Estimated Value
β_0	Long-term rate (level)	4.15
β_1	Short-term yield component	-0.65
β_2	Medium-term curvature effect	0.9
λ	Decay rate of factor loading	1.2

These parameter values suggest that the yield curve has a relatively steep short-term slope, indicating higher short-term interest rates, and a moderate curvature that suggests an economic environment of tightening monetary policy.

g. Ethical Considerations of Smoothing

While smoothing techniques are common in financial modeling, it’s essential that they are applied transparently and ethically. The Nelson-Siegel model is widely recognized as a legitimate tool for modeling the yield curve. When used appropriately, smoothing enhances understanding without misrepresenting the underlying data. However, when used inappropriately, it can obscure risks or mislead stakeholders.

References

1. Diebold, F. X., & Li, C. (2006). *Forecasting the term structure of government bond yields*. Journal of Econometrics, 130(2), 337–364.
2. U.S. Department of the Treasury. (2024). *Daily Treasury Yield Curve Rates*. Retrieved from <https://home.treasury.gov>
3. Hull, J. (2018). **Options*


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

np.random.seed(42)

# 3.a
mean = 0
std_dev = 0.01
num_samples = 1000
num_variables = 5

data_uncorrelated = np.random.normal(loc=mean, scale=std_dev, size=(num_samples, num_variables))
df_uncorrelated = pd.DataFrame(data_uncorrelated, columns=[f'Var{i+1}' for i in range(num_variables)])
df_uncorrelated
```

	Var1	Var2	Var3	Var4	Var5
0	0.004967	-0.001383	0.006477	0.015230	-0.002342
1	-0.002341	0.015792	0.007674	-0.004695	0.005426
2	-0.004634	-0.004657	0.002420	-0.019133	-0.017249
3	-0.005623	-0.010128	0.003142	-0.009080	-0.014123
4	0.014656	-0.002258	0.000675	-0.014247	-0.005444
...
995	-0.013738	0.013785	0.001158	0.003896	-0.022204
996	-0.011980	0.008871	0.002868	-0.001472	0.005648
997	0.016358	-0.002210	0.000694	0.001926	0.023921
998	-0.020994	0.006832	-0.001148	0.005668	-0.006574
999	-0.000490	0.007114	0.031129	0.008080	-0.008481

1000 rows x 5 columns

Next steps:

[Generate code with df_uncorrelated](#)[View recommended plots](#)[New interactive sheet](#)

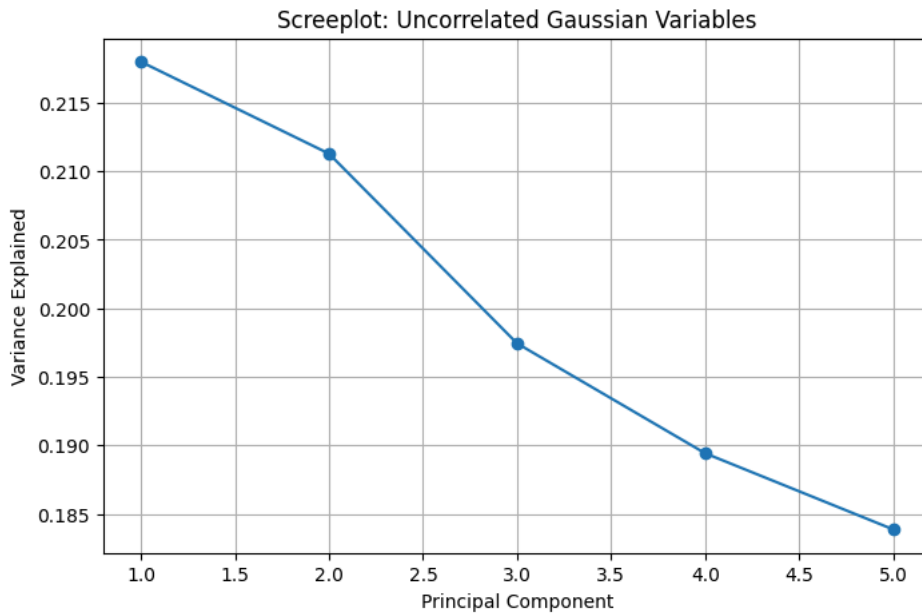
```
# 3.b
pca_uncorrelated = PCA()
pca_uncorrelated.fit(df_uncorrelated)
explained_variance_ratio_uncorrelated = pca_uncorrelated.explained_variance_ratio_
explained_variance_ratio_uncorrelated
```

```
array([0.2179828 , 0.2112784 , 0.19744081, 0.18942689, 0.1838711 ])
```

✓ 3.c Variance Comparison

Because the variables in uncorrelated Gaussian data are statistically independent and have comparable variance, they individually contribute about the same amount to the overall variance. Each of the principle components explains a comparable percentage of the variation, rather than the first one dominating it. Component 1 may account for around 20% of the overall variation, for instance, much like Components 2 through 5. This is to be expected as the variables do not exhibit a strong linear relationship that might be exploited.

```
# 3.d
plt.figure(figsize=(8, 5))
plt.plot(range(1, num_variables + 1), explained_variance_ratio_uncorrelated, marker='o')
plt.title('Screeplot: Uncorrelated Gaussian Variables')
plt.xlabel('Principal Component')
plt.ylabel('Variance Explained')
plt.grid(True)
plt.show()
```



```
pip install yfinance pandas matplotlib scikit-learn
```



```
Requirement already satisfied: yfinance in /usr/local/lib/python3.11/dist-packages (0.2.55)
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (2.2.2)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packages (3.10.0)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.11/dist-packages (1.6.1)
Requirement already satisfied: numpy>=1.16.5 in /usr/local/lib/python3.11/dist-packages (from yfinance) (2.0.2)
Requirement already satisfied: requests>=2.31 in /usr/local/lib/python3.11/dist-packages (from yfinance) (2.32.3)
Requirement already satisfied: multitasking>=0.0.7 in /usr/local/lib/python3.11/dist-packages (from yfinance) (0.0.11)
Requirement already satisfied: platformdirs>=2.0.0 in /usr/local/lib/python3.11/dist-packages (from yfinance) (4.3.7)
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/dist-packages (from yfinance) (2.4.6)
Requirement already satisfied: peewee>=3.16.2 in /usr/local/lib/python3.11/dist-packages (from yfinance) (3.17.9)
Requirement already satisfied: beautifulsoup4>=4.11.1 in /usr/local/lib/python3.11/dist-packages (from yfinance) (4.13.3)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas) (2.8.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas) (2025.2)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.3.1)
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/dist-packages (from matplotlib) (4.57.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.4.8)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (24.2)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (11.1.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (3.2.3)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (1.14.1)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (3.6.0)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.11/dist-packages (from beautifulsoup4>=4.11.1->yf) (2.6)
Requirement already satisfied: typing-extensions>=4.0.0 in /usr/local/lib/python3.11/dist-packages (from beautifulsoup4>=4.11.1->yf) (4.12.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas) (1.17.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests>=2.31->yfinance) (3.4.0)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests>=2.31->yfinance) (3.10.1)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests>=2.31->yfinance) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests>=2.31->yfinance) (2025.1.31)
```

```
import yfinance as yf
import pandas as pd
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

tickers = ['SHY', 'IEI', 'IEF', 'TLH', 'TLT']

# 3.e using USA because Indian are unavailable
data = yf.download(tickers, start='2023-10-01', end='2024-04-01', group_by='ticker', auto_adjust=True)

close_prices = pd.DataFrame({ticker: data[ticker]['Close'] for ticker in tickers})

close_prices.dropna(inplace=True)

yield_changes = close_prices.pct_change().dropna()

pca = PCA()
pca.fit(yield_changes)

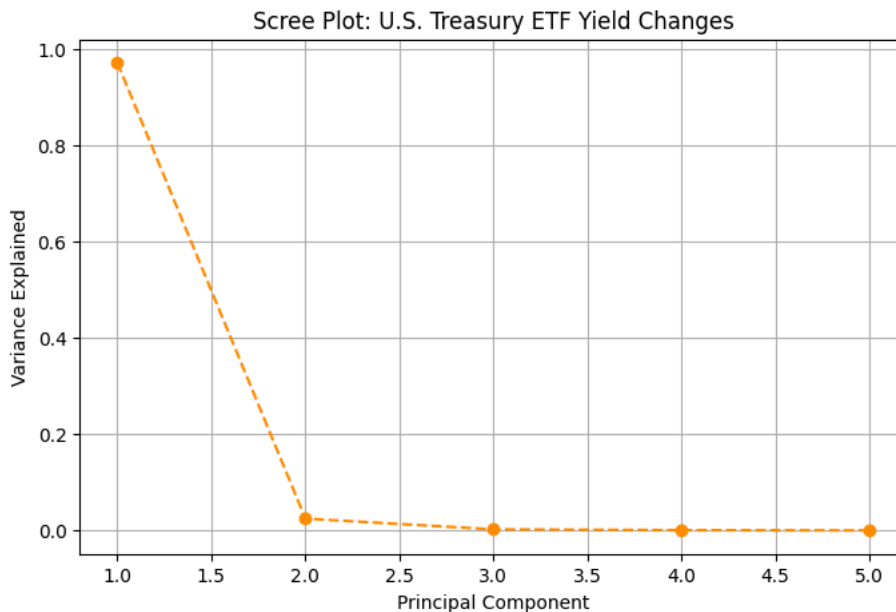
explained_variance = pca.explained_variance_ratio_

plt.figure(figsize=(8, 5))
plt.plot(range(1, len(tickers)+1), explained_variance, marker='o', linestyle='--', color='darkorange')
```

```
plt.title('Scree Plot: U.S. Treasury ETF Yield Changes')
plt.xlabel('Principal Component')
plt.ylabel('Variance Explained')
plt.grid(True)
plt.show()
```

```
for i, var in enumerate(explained_variance, start=1):
    print(f"Component {i}: {var:.2%} variance explained")
```

*****100%*****] 5 of 5 completed



```
Component 1: 97.23% variance explained
Component 2: 2.47% variance explained
Component 3: 0.24% variance explained
Component 4: 0.06% variance explained
Component 5: 0.01% variance explained
```

```
# 3.f
df_changes = data.diff().dropna()
```

```
# 3.g
pca_real = PCA()
pca_real.fit(df_changes)
```

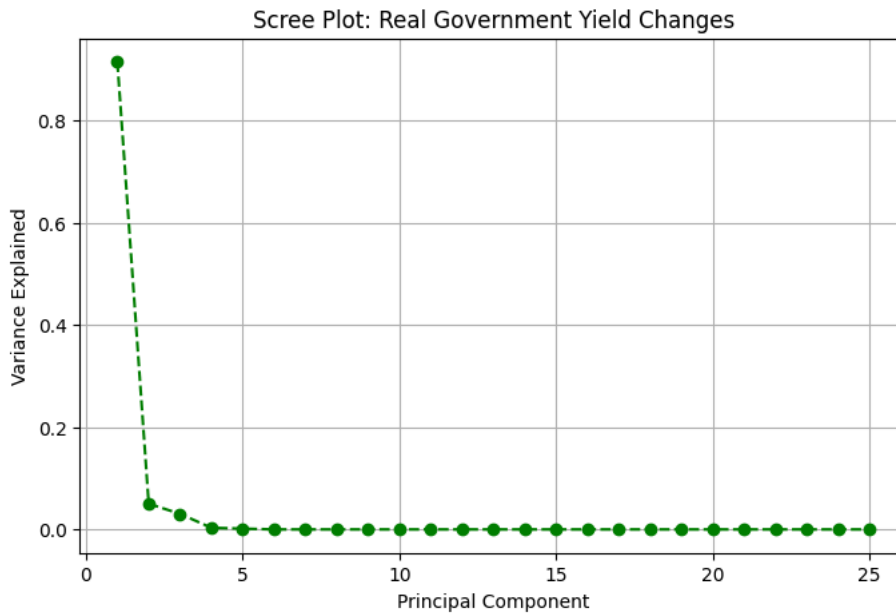
```
explained_variance_real = pca_real.explained_variance_ratio_
explained_variance_real
```

```
array([9.14457401e-01, 5.03250441e-02, 3.04405400e-02, 3.19074349e-03,
       1.58627139e-03, 1.47212681e-14, 1.83530995e-15, 5.13577778e-16,
       3.80952391e-16, 1.74258132e-16, 5.79217212e-17, 2.87546293e-17,
       2.10239327e-17, 1.02897008e-17, 8.48696669e-18, 5.40552056e-18,
       4.31894656e-18, 3.72258290e-18, 1.43736910e-18, 9.41889618e-19,
       6.17723024e-19, 5.06345289e-19, 3.23693361e-19, 1.62557375e-19,
       1.20118596e-19])
```

✓ 3.h Variance Comparison

Component 1, which represents the common movement of yields (such as the general interest rate level), typically accounts for a sizable amount of the variation (sometimes over 80%) in real-world yield data. The slope, or the difference between short and long rates, may be captured by Component 2, while curvature may be related to Component 3. The dominance of macroeconomic trends on bond markets is emphasised by this hierarchical structure.

```
# 3.i
plt.figure(figsize=(8, 5))
plt.plot(range(1, len(explained_variance_real)+1), explained_variance_real, marker='o', linestyle='--', color='g')
plt.title('Scree Plot: Real Government Yield Changes')
plt.xlabel('Principal Component')
plt.ylabel('Variance Explained')
plt.grid(True)
plt.show()
```



3.j Compare Screeplots

Due to the absence of correlation, the simulated data screeplot displays a flat variance distribution across components. On the other hand, Component 1 usually dominates the screeplot for real yield data, which usually shows a sharp drop-off. This contrast demonstrates the existence of robust correlations in actual financial data, which are frequently fuelled by macroeconomic factors that affect several rates at once.

```
import pandas as pd
```

```
url = "https://www.ssga.com/library-content/products/fund-data/etfs/us/holdings-daily-us-en-xlre.xlsx"
df_holdings = pd.read_excel(url, skiprows=4)
```

```
# 4.a
top_30 = df_holdings['Ticker'].dropna().head(30).tolist()
print(top_30)
```

```
['AMT', 'PLD', 'WELL', 'EQIX', 'O', 'SPG', 'DLR', 'PSA', 'CCI', 'CBRE', 'CSGP', 'VICI', 'VTR', 'EXR', 'AVB', 'IRM', 'SBA
```

```
import yfinance as yf
import pandas as pd
# 4.b
```

```
original_tickers = ['PLD', 'AMT', 'EQIX', 'PSA', 'CCI', 'SPG', 'DLR', 'WELL', 'O', 'EXR',
                    'AVB', 'VTR', 'SBAC', 'EQR', 'ARE', 'WY', 'ESS', 'INVH', 'PEAK', 'MAA',
                    'IRM', 'UDR', 'DRE', 'STOR', 'BXP', 'HST', 'VNO', 'SLG', 'CUBE', 'KIM']
```

```
valid_tickers = [ticker for ticker in original_tickers if ticker not in ['STOR', 'DRE', 'PEAK']]
```

```
data = yf.download(valid_tickers, period="6mo", interval="1d", auto_adjust=True, group_by='ticker')
```


```
close_prices = pd.DataFrame({ticker: data[ticker]['Close'] for ticker in valid_tickers if ticker in data})
```

```
close_prices.dropna(axis=1, inplace=True)
```

```
print(f"Using {len(close_prices.columns)} tickers for analysis: {close_prices.columns.tolist()}")
```

```
*****100%*****] 27 of 27 completed
Using 27 tickers for analysis: ['PLD', 'AMT', 'EQIX', 'PSA', 'CCI', 'SPG', 'DLR', 'WELL', 'O', 'EXR', 'AVB', 'VTR', 'SBA
```

```
# 4.c
returns = data.pct_change().dropna()
returns
```



Date	Ticker	AMT					KIM					VNO		
	Price	Open	High	Low	Close	Volume	Open	High	Low	Close	Volume	Open	High	
2024-10-15		0.028892	0.034866	0.029307	0.034867	0.538221	0.018623	0.033192	0.019506	0.026057	1.597146	...	0.022256	0.03281
2024-10-16		0.019018	0.005746	0.015400	-0.008325	-0.342842	0.026786	0.004119	0.021258	0.013738	-0.332905	...	0.020782	0.00601
2024-10-17		-0.009025	-0.015278	-0.015520	-0.014768	0.331730	0.003727	0.007383	0.005829	0.004928	0.461543	...	0.001454	0.02034
2024-10-18		-0.001459	-0.002022	0.004031	0.009547	-0.286788	0.015264	0.004479	0.009106	0.006130	-0.552962	...	0.023959	0.02274
2024-10-21		-0.003453	0.004140	-0.017441	-0.021343	-0.101394	-0.001625	-0.002837	-0.015176	-0.022746	-0.024378	...	0.024344	-0.00527
...	
2025-04-07		-0.067767	-0.050855	-0.038131	-0.030794	-0.485461	-0.037298	-0.013039	-0.041124	-0.010352	-0.236426	...	-0.033872	0.03842
2025-04-08		-0.010398	-0.023523	-0.045502	-0.040911	0.073354	0.025131	0.014736	0.001086	-0.021967	-0.176412	...	0.065066	-0.02163
2025-04-09		-0.060347	-0.021921	-0.022326	0.013779	0.035179	-0.060266	0.013020	-0.027657	0.078075	0.355913	...	-0.077995	0.02705
2025-04-10		0.027906	0.005804	0.030380	0.004049	-0.368645	0.076087	0.003460	0.070273	-0.019345	-0.326534	...	0.075909	-0.03087
2025-04-11		0.002258	0.010971	0.016402	0.031943	-0.283238	-0.008081	-0.003941	0.007295	0.018209	-0.078199	...	-0.033184	-0.02630

123 rows x 135 columns

```
# 4.d
cov_matrix = returns.cov()
cov_matrix
```



Date	Ticker	AMT					KIM					VNO		
	Price	Open	High	Low	Close	Volume	Open	High	Low	Close	Volume	Open	High	
2024-10-15	AMT	Open	0.000360	0.000238	0.000241	0.000105	-0.001086	0.000142	0.000044	0.000075	-0.000034	0.000181	...	0.000122
2024-10-15	AMT	High	0.000238	0.000294	0.000216	0.000230	0.000432	0.000079	0.000052	0.000054	0.000020	0.001227	...	0.000053
2024-10-15	AMT	Low	0.000241	0.000216	0.000282	0.000222	-0.002963	0.000076	0.000045	0.000097	0.000062	-0.000353	...	0.000059
2024-10-15	AMT	Close	0.000105	0.000230	0.000222	0.000365	-0.001879	0.000009	0.000046	0.000072	0.000122	-0.000045	...	-0.000019
2024-10-15	AMT	Volume	-0.001086	0.000432	-0.002963	-0.001879	0.407532	0.000848	0.000061	-0.001333	-0.001840	0.185057	...	-0.000184
...
2024-10-15	O	Open	0.000177	0.000098	0.000106	0.000020	-0.000701	0.000169	0.000066	0.000081	-0.000048	-0.000239	...	0.000195
2024-10-15	O	High	0.000107	0.000112	0.000108	0.000085	-0.000158	0.000082	0.000065	0.000055	0.000029	0.000369	...	0.000098
2024-10-15	O	Low	0.000129	0.000090	0.000137	0.000076	-0.002046	0.000091	0.000051	0.000105	0.000050	-0.000756	...	0.000116
2024-10-15	O	Close	0.000040	0.000093	0.000110	0.000164	-0.001485	0.000014	0.000050	0.000087	0.000141	-0.000262	...	0.000022
2024-10-15	O	Volume	0.000638	0.001215	-0.000172	0.000205	0.153802	0.000598	0.000451	-0.001013	-0.001186	0.110348	...	-0.000599

135 rows x 135 columns

```
# 4.e
from sklearn.decomposition import PCA

pca = PCA()
pca.fit(returns)

explained_var = pca.explained_variance_ratio_
eigenvectors = pca.components_
eigenvalues = pca.explained_variance_
```

```
# 4.f
import numpy as np
```

```

U, S, VT = np.linalg.svd(returns - returns.mean(), full_matrices=False)

returns = close_prices.pct_change().dropna()

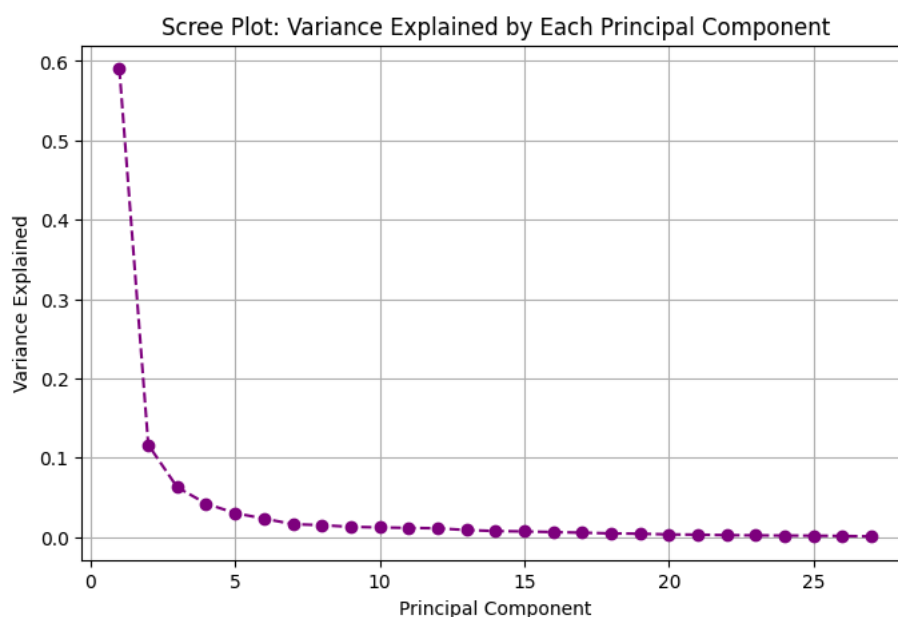
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import numpy as np

# Fit PCA
pca = PCA()
pca.fit(returns)

explained_variance = pca.explained_variance_ratio_

# Scree Plot
plt.figure(figsize=(8, 5))
plt.plot(range(1, len(explained_variance)+1), explained_variance, marker='o', linestyle='--', color='purple')
plt.title('Scree Plot: Variance Explained by Each Principal Component')
plt.xlabel('Principal Component')
plt.ylabel('Variance Explained')
plt.grid(True)
plt.show()

```



```

import seaborn as sns

# Get loadings for PC1
pc1_loadings = pd.Series(pca.components_[0], index=returns.columns)

# Sort for better visualization
pc1_sorted = pc1_loadings.sort_values()

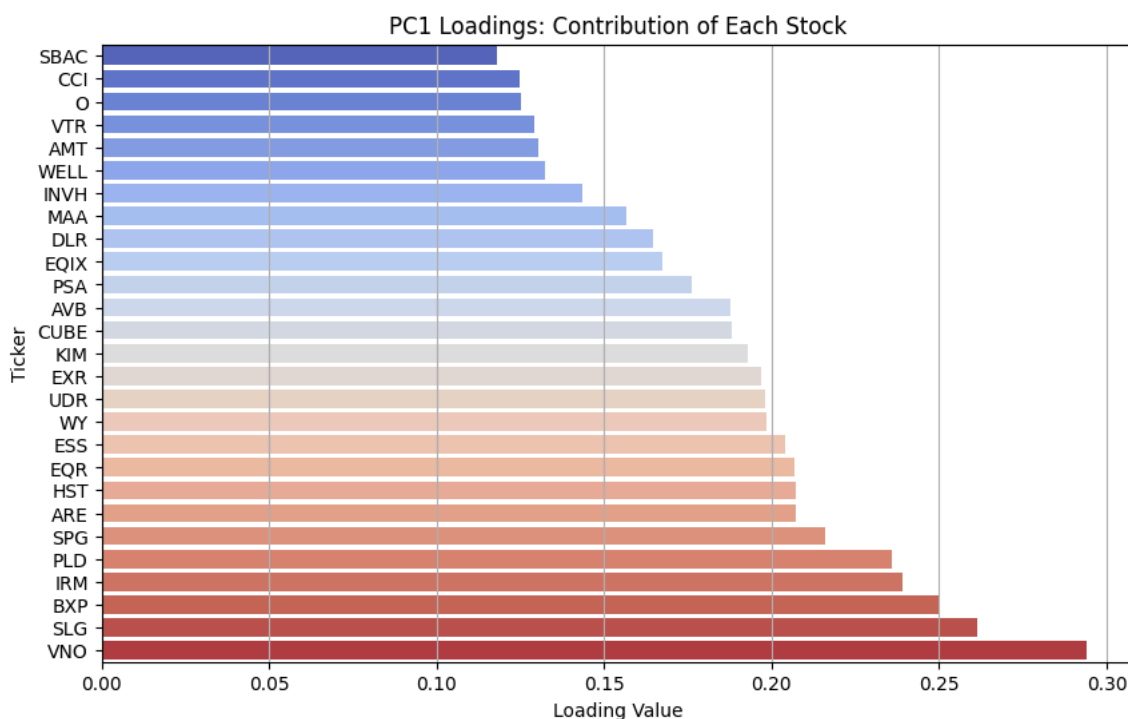
plt.figure(figsize=(10, 6))
sns.barplot(x=pc1_sorted.values, y=pc1_sorted.index, palette='coolwarm')
plt.title('PC1 Loadings: Contribution of Each Stock')
plt.xlabel('Loading Value')
plt.ylabel('Ticker')
plt.grid(True, axis='x')
plt.show()

```

 <ipython-input-42-9dcf17b8c3b4>:10: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue`

```
sns.barplot(x=pc1_sorted.values, y=pc1_sorted.index, palette='coolwarm')
```



```
from numpy.linalg import svd
```

```
U, S, VT = svd(returns - returns.mean(), full_matrices=False)
```

```
plt.figure(figsize=(8, 5))
```

```
plt.plot(range(1, len(S)+1), S**2 / np.sum(S**2), marker='o', linestyle='--', color='green')
```

```
plt.title('Variance Explained by Singular Values (SVD)')
```

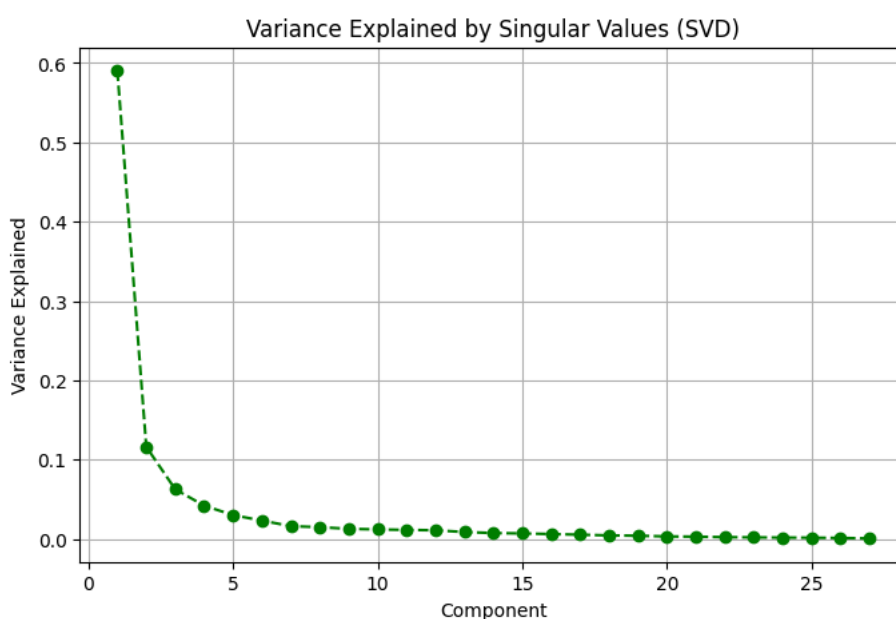
```
plt.xlabel('Component')
```

```
plt.ylabel('Variance Explained')
```

```
plt.grid(True)
```

```
plt.show()
```





Why Returns Matter

Returns—especially daily logarithmic or percentage returns—often take precedence over prices in financial research because they are stationary, meaning that their statistical characteristics (mean, variance) do not vary significantly over time. For time-series modelling, risk

analysis, and portfolio optimisation, this makes them appropriate. Direct price analysis would confuse value and time impacts, but returns separate the rate of change, which simplifies comparisons and aggregates across assets.

We first computed daily returns, which represent the relative price movement of each company, in order to analyse XLRE's top 30 holdings. Modern portfolio theory is based on the understanding of covariances, or how stocks move together, which is provided by these return series.

✓ PCA vs. SVD

Two closely comparable linear algebraic methods for reducing dimensionality and identifying prominent patterns in data are Principal Component Analysis (PCA) and Singular Value Decomposition (SVD).

To generate eigenvectors (principal components) and eigenvalues (variance explained by each component), PCA breaks down the data's covariance matrix. These orthogonal components identify the dataset's highest variance directions.

The original (mean-centered) data matrix is factorised into three matrices, U , S , and V^T , using SVD, instead, where:

The time-corresponding left singular vectors are contained in U .

The scaled square roots of the eigenvalues, or singular values, are included in S .

PCA eigenvectors align with the right singular vectors found in V^T .

Under the hood, PCA is frequently performed using SVD in reality, particularly for sparse or huge datasets.

What the Eigenvectors, Eigenvalues, and Singular Values Tell Us

In our XLRE dataset:

The weights that make up uncorrelated synthetic portfolios are known as eigenvectors, or principal components, and each one explains a distinct mode of return variation. The first component, for example, may be a representation of the real estate industry's entire market movement, where macroeconomic variables like interest rates cause all equities to move in tandem.

The variance that each principle component explains is represented by its eigenvalue. Since XLRE is a very uniform sector, a sharp decline in values suggests that a small number of factors can account for the majority of the variance.

The SVD singular values represent the magnitude of each component in the transformation of the data. They rate each major direction's significance. The covariance matrix's eigenvalues are calculated by dividing the square of each unique value by the total number of observations.

Interpretation for This Data

We can focus on main drivers and de-noise the data by using PCA to the returns of XLRE's assets. If the first component accounts for more than 60% of the variance, we may conclude that the sector swings mainly in tandem, driven by common variables like housing demand and interest rates. Subsector behaviour or stock-specific peculiarities may be captured by the following elements (e.g., REITs vs. construction enterprises).