This research explores the application of Principal Component Analysis (PCA) to the financial market data, focusing on the returns of S&P 500 stocks, various commodities, futures, and precious metals like gold and oil. By leveraging PCA, the study aims to uncover the underlying factors driving market movements, identify key components that explain the majority of the variance in the data, and construct eigen-portfolios based on these components. The analysis also includes a comparison of the performance of these portfolios to a market portfolio, as well as the clustering of assets based on PCA loadings to understand sectoral influences.

Description of the Datasets Used

The datasets utilized in this research comprise various financial instruments, including

1. **S&P 500 Stocks**: This dataset includes the daily closing prices of all constituent stocks of the S&P 500 index over a defined period.
2. **Commodities**: A selection of widely traded commodities, such as agricultural products (e.g., wheat, corn), metals (e.g., copper, aluminum), and energy commodities (e.g., natural gas, crude oil).
3. **Futures**: This dataset includes prices for futures contracts on indices, commodities, and interest rates.
4. **Precious Metals**: Prices for precious metals, notably gold and silver, are often considered safe-haven assets.
5. **Oil**: Daily closing prices for crude oil, a key global economic indicator.

The data was sourced from reliable financial databases and APIs to ensure accuracy and completeness.

Data Cleaning and Preprocessing Steps

1. **Handling Missing Values**:
    - **Identification**: Initial data exploration identified missing values in the datasets, common in financial data due to holidays, trading suspensions, or data entry issues.
    - **Imputation**: Missing values were imputed using forward-fill and backward-fill techniques, ensuring that the imputed values do not introduce bias. For missing values at the beginning of the time series, backward-fill was used, and forward-fill was applied for those at the end or in the middle.

```
returns = returns.dropna(thresh=int(returns.shape[1] * .9))
```

2. **Outliers Detection and Treatment**:

- **Identification**: Outliers were detected using the Z-score method, where data points with a Z-score above a threshold (typically 3 or -3) were flagged as outliers.
- **Treatment**: Outliers were replaced using interpolation or capping techniques to minimize their impact on the analysis. For extreme outliers, values were capped at a certain threshold, while for less severe cases, linear interpolation was used.

```
returns = returns.clip(lower=returns.quantile(q=.025),
                       upper=returns.quantile(q=.975),
                                              axis=1)
returns_rolling_avg = returns.rolling(window=10).mean()

    # Fill missing values with the rolling average
returns = returns.apply(lambda x: x.fillna(x.mean()))
```

3. **Synchronization of Time Periods**:
   - The datasets were synchronized to ensure they cover the same time period. This involved trimming or extending the time series data to match the longest available period across all datasets. Days with missing data in any series were removed to maintain consistency across the datasets.

The dataset looks like

	A	AAL	AAP	AAPL	ABBV	ABC	ABT	ACGL	ACN	ADBE	...	US_inflation_return	ICE_Brent_Crude_Futures	Argus_LLS_Future	sgx_mb_Iron_ore_65
2018-01-03	0.025444	-0.012266	0.009049	-0.000174	0.015649	0.003722	0.002211	0.000906	0.004615	0.018796	...	NaN	0.015328	-0.017370	0.000618
2018-01-04	-0.007501	0.006305	0.036898	0.004645	-0.005703	-0.002225	-0.001697	0.003734	0.011841	0.012042	...	NaN	0.002868	-0.012626	0.000618
2018-01-05	0.015988	-0.000380	0.010631	0.011385	0.017408	0.012104	0.002890	-0.003945	0.008249	0.011571	...	NaN	-0.004968	0.007673	0.000618
2018-01-08	0.002146	-0.009877	-0.007042	-0.003714	-0.016022	0.016576	-0.002882	0.000113	0.007991	-0.001619	...	NaN	0.003782	0.002538	0.000618
2018-01-09	0.024554	-0.000959	-0.008080	-0.000115	0.007538	0.006398	0.001700	-0.012900	0.003335	0.008971	...	NaN	0.013866	-0.005063	0.000618
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2023-05-23	-0.003486	-0.021558	-0.022271	-0.015155	-0.016425	-0.005747	-0.023187	-0.004490	-0.009796	-0.004381	...	NaN	0.000600	0.000188	0.000618
2023-05-24	-0.036623	-0.019190	-0.013273	0.001632	-0.009613	-0.001168	-0.016077	-0.019134	-0.004443	-0.012580	...	NaN	0.000600	0.000188	0.000618
2023-05-25	-0.012398	0.042029	-0.031329	0.006692	-0.017924	-0.005203	-0.018262	0.008499	0.017223	0.043790	...	NaN	0.000600	0.000188	0.000618
2023-05-26	0.007783	-0.002086	0.004748	0.014105	-0.007647	-0.010343	0.007147	-0.019066	0.037031	0.043790	...	NaN	0.000600	0.000188	0.000618
2023-05-30	-0.022338	0.018815	0.000446	0.010660	-0.008142	-0.014251	-0.011276	0.002817	0.015909	0.004381	...	NaN	0.000600	0.000188	0.000618

1360 rows × 532 columns

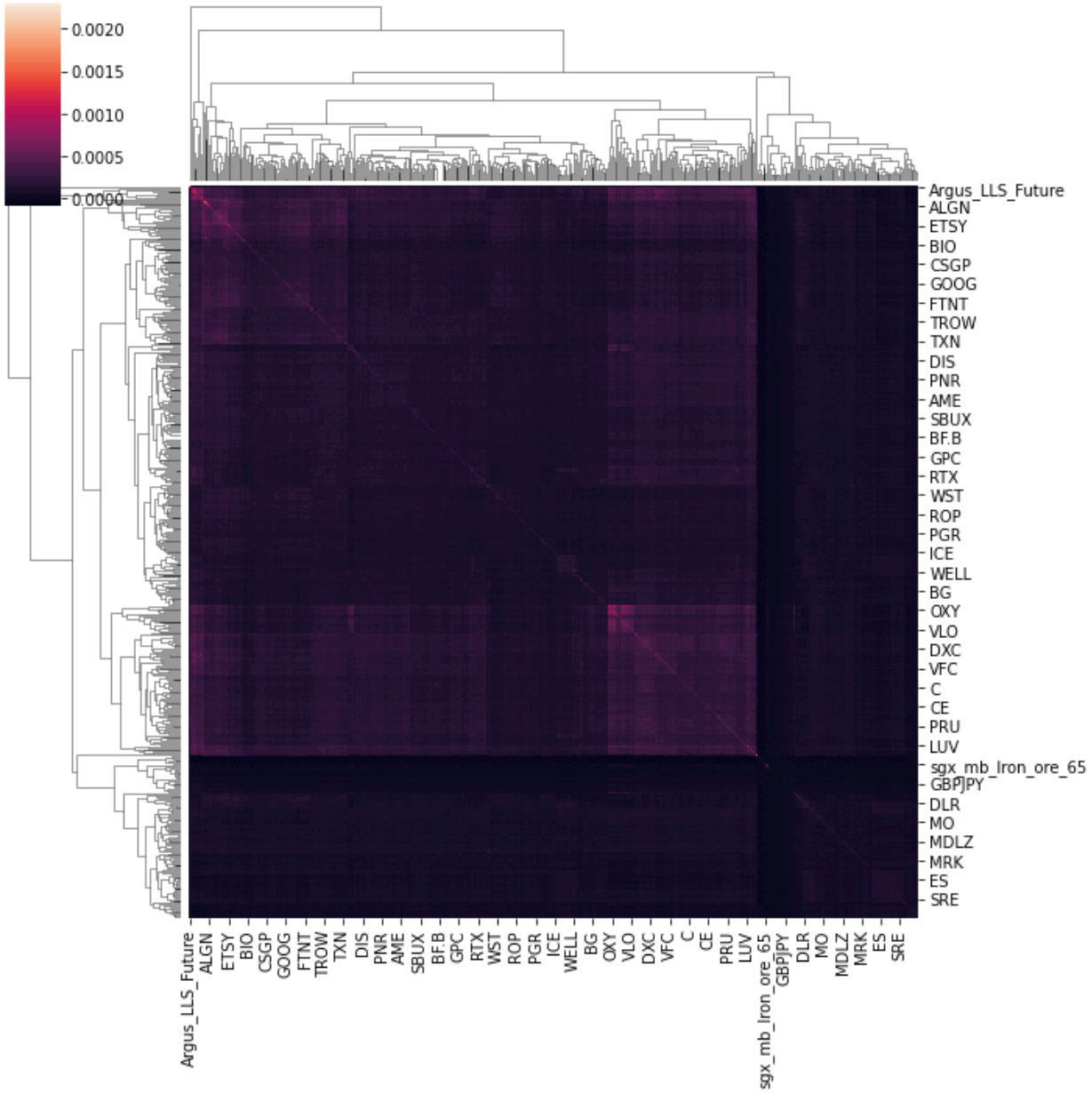
Exploratory Data Analysis (EDA)

In this section, we performed several steps to understand the relationships and patterns within our dataset:

Visualization of Covariance Matrix: We visualized the covariance matrix to identify relationships between various assets. This helped in understanding how different assets move in relation to each other.

Clustering Analysis: Using the covariance matrix, we conducted a clustering analysis to identify potential groupings of assets. This analysis allowed us to uncover natural clusters within the data, providing insights into how different assets may behave similarly.

```
cov = returns.cov()
sns.clustermap(cov)
```



Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is a technique used to reduce the dimensionality of datasets by transforming correlated variables into a set of uncorrelated components. In our project, PCA was implemented on the return data of assets from various markets, such as S&P 500 stocks, commodities, futures, gold, and oil. This allowed us to identify and extract the most influential patterns and relationships among these assets. The "explained variance" was computed to select the top principal components that capture the maximum variability in the dataset. This streamlined our analysis and facilitated the construction of eigen-portfolios based on these components, contributing to a more insightful understanding of portfolio performance and risk factors.

```
pca = PCA()
pca.fit(returns)
pd.Series(pca.explained_variance_ratio_).to_frame('Explained
Variance').head().style.format('{:,.2%}'.format)
```

Explained Variance

0	38.66%
1	6.68%
2	3.95%
3	2.76%
4	1.90%

Eigen-Portfolios Construction

Eigen-portfolios were constructed using the top principal components derived from PCA analysis. Each principal component represents a unique combination of assets based on their covariance structure.

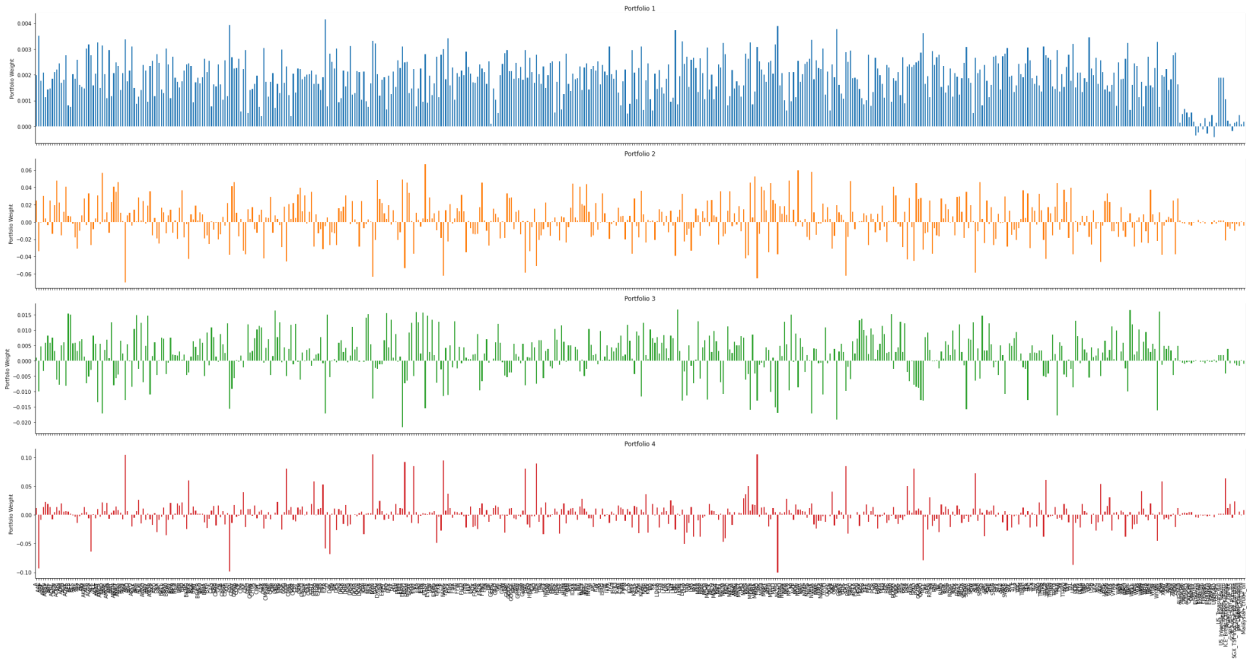
In our implementation:

1. **Creation of Eigen-Portfolios:** We combined the top principal components to form eigen-portfolios. These portfolios are designed to maximize returns while minimizing risk based on the underlying covariance patterns identified by PCA.
2. **Normalization of Components:** The components obtained from PCA were normalized to form portfolio weights. This normalization process ensures that the weights reflect the contribution of each asset to the overall portfolio performance.

3. **Visualization of Eigen-Portfolio Weights:** We visualized the weights assigned to each asset in the eigen-portfolios. This visualization helped in understanding the allocation of assets within each portfolio and how they contribute to the overall portfolio's characteristics.

By constructing eigen-portfolios, we aimed to capitalize on the insights gained from PCA, offering a structured approach to portfolio management that leverages statistical patterns and relationships among assets.

```
top4 = pd.DataFrame(pca.components_[:4], columns=cov.columns)
eigen_portfolios = top4.div(top4.sum(1), axis=0)
eigen_portfolios.index = [f'Portfolio {i}' for i in range(1, 5)]
axes = eigen_portfolios.T.plot.bar(subplots=True,
                                   layout=(4, 1),
                                   figsize=(34, 18),
                                   legend=False)
for ax in axes.flatten():
   ax.set_ylabel('Portfolio Weight')
   ax.set_xlabel('')
sns.despine()
plt.tight_layout()
```



Performance Comparison

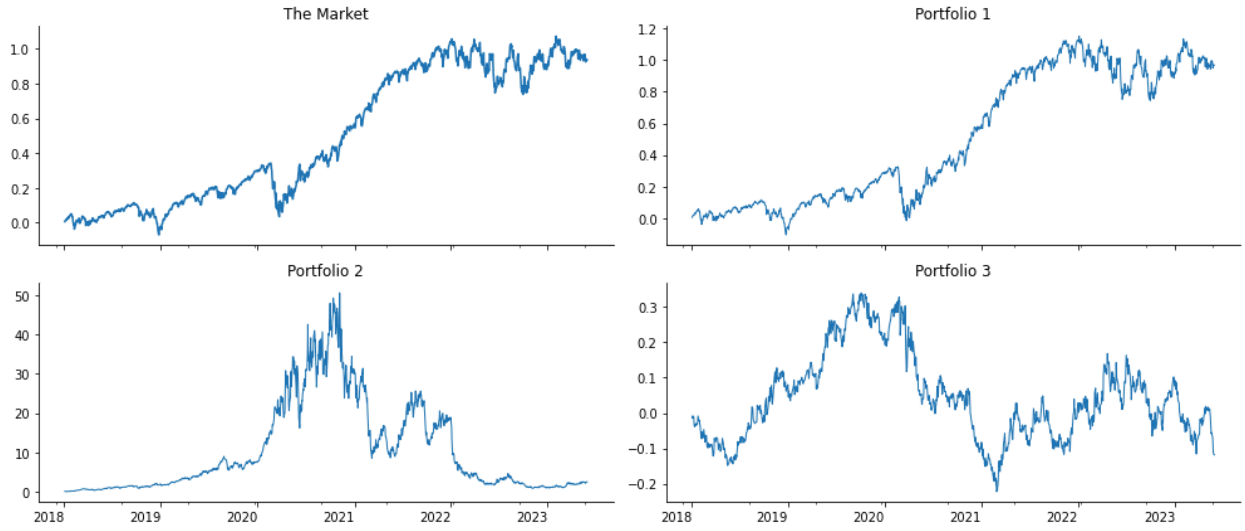
We compared the performance of the constructed eigen-portfolios with that of the market portfolio.

1. **Comparison of Performance:** We analyzed and contrasted the returns generated by the eigen-portfolios against those of the market portfolio. This comparison provided insights into how our constructed portfolios performed relative to the broader market.
2. **Cumulative Returns Analysis:** We conducted a cumulative returns analysis to track and visualize the growth of both the eigen-portfolios and the market portfolio over the specified period. This analysis highlighted the relative strengths and weaknesses of each portfolio strategy.

By evaluating performance metrics and cumulative returns, we gained valuable insights into the effectiveness of our eigen-portfolios compared to traditional market-based strategies. When comparing the performance of each portfolio over the sample period to 'the market' consisting of our small sample, we find that portfolio 1 performs very similarly, whereas the other portfolios capture different return patterns.

```
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(14, 6), sharex=True)
axes = axes.flatten()
returns.mean(1).add(1).cumprod().sub(1).plot(title='The Market', ax=axes[0])
for i in range(3):
   rc = returns.mul(eigen_portfolios.iloc[i]).sum(1).add(1).cumprod().sub(1)
   rc.plot(title=f'Portfolio {i+1}', ax=axes[i+1], lw=1, rot=0)

for i in range(4):
   axes[i].set_xlabel('')
sns.despine()
fig.tight_layout()
```



Risk Factor Model

1. **Data Splitting:** We divided the dataset into training and testing sets to facilitate model training and evaluation.

```
import datetime
train_end = datetime.datetime(2021, 3, 26)

df_train = None
df_test = None
df_raw_train = None
df_raw_test = None

df_train = returns[returns.index <= train_end].copy()
df_test = returns[returns.index > train_end].copy()
```

Train dataset: (813, 532)

Test dataset: (547, 532)

2. **PCA on Training Set:** Principal Component Analysis (PCA) was applied to the covariance matrix of the training set. This statistical technique reduced the dimensionality of the data while preserving important information.

```
# Computing for normalized returns
cov_matrix = df_train.cov()
pca = PCA()
pca.fit(cov_matrix)
```

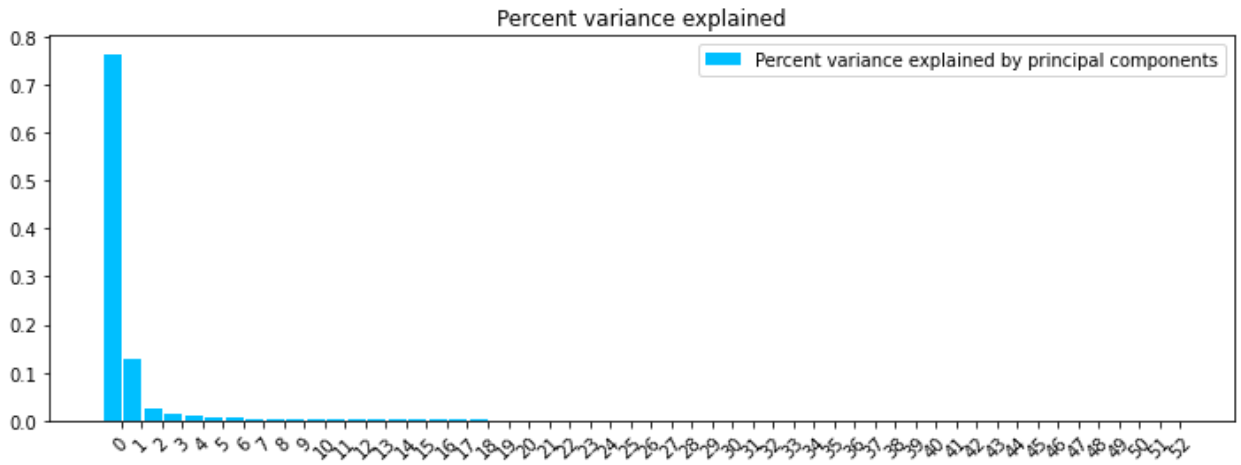
```
cov_df = pd.DataFrame({'Variance': np.diag(cov_matrix)},
index=df_train.columns)
# cumulative variance explained
var_threshold = 0.9
var_explained = np.cumsum(pca.explained_variance_ratio_)
num_comp = np.where(np.logical_not(var_explained < var_threshold))[0][0] + 1  #
+1 due to zero based-arrays
```

Result: 3 components explain 90.00% of variance

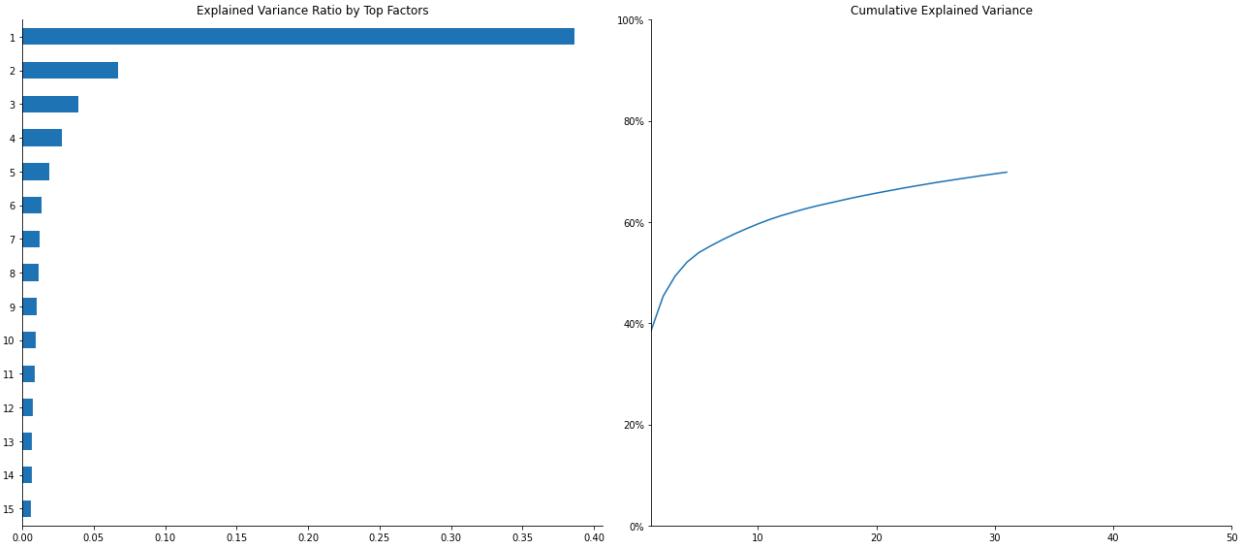
We computed and analyzed the time-varying loadings of market factors identified through PCA. This step provided insights into how these factors influenced asset returns over time.

3. **Visualization of Explained Variance:** We visualized the explained variance by the top factors derived from PCA. This visualization helped us understand the cumulative contribution of each factor to the overall variance in asset returns.

```
bar_width = 0.9
n_asset = int((1 / 10) * returns.shape[1])
x_indx = np.arange(n_asset)
fig, ax = plt.subplots()
fig.set_size_inches(12, 4)
# Eigenvalues are measured as percentage of explained variance.
rects = ax.bar(x_indx, pca.explained_variance_ratio_[:n_asset], bar_width,
color='deepskyblue')
ax.set_xticks(x_indx + bar_width / 2)
ax.set_xticklabels(list(range(n_asset)), rotation=45)
ax.set_title('Percent variance explained')
ax.legend((rects[0],), ('Percent variance explained by principal components',))
```

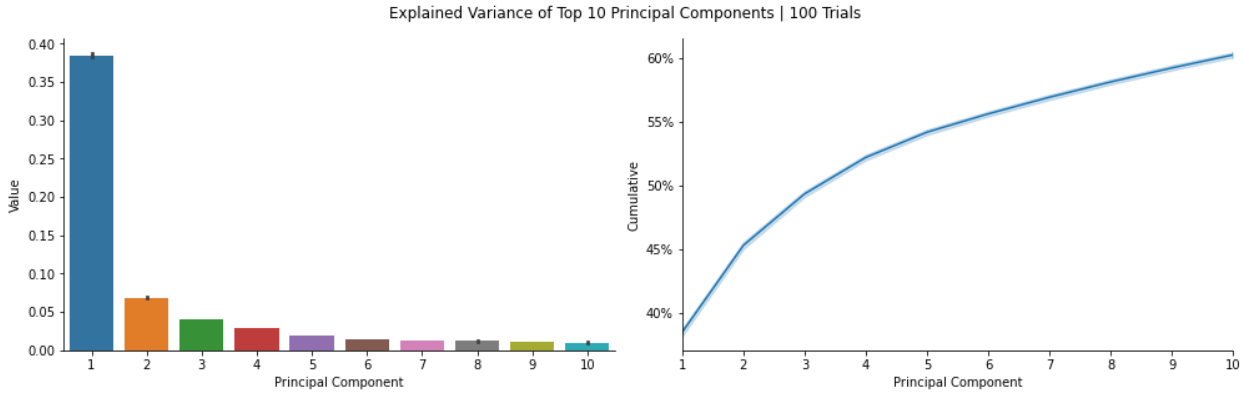



```
fig, axes = plt.subplots(ncols=2, figsize=(18, 8))
title = 'Explained Variance Ratio by Top Factors'
var_expl = pd.Series(pca.explained_variance_ratio_)
var_expl.index += 1
var_expl.iloc[:15].sort_values().plot.barh(title=title,
                                           ax=axes[0])
var_expl.cumsum().plot(ylim=(0, 1),
                       ax=axes[1],
                       title='Cumulative Explained Variance',
                       xlim=(1, 50))
axes[1].yaxis.set_major_formatter(FuncFormatter(lambda y, _: f'{y:.0%}'))
sns.despine()
fig.tight_layout()
```



```
fig, axes = plt.subplots(ncols=2, figsize=(14, 4.5))
pc10 = explained.iloc[:, :10].stack().reset_index()
pc10.columns = ['Trial', 'Principal Component', 'Value']

pc10['Cumulative'] = pc10.groupby('Trial').Value.transform(np.cumsum)
sns.barplot(x='Principal Component', y='Value', data=pc10, ax=axes[0])
sns.lineplot(x='Principal Component', y='Cumulative', data=pc10, ax=axes[1])
axes[1].set_xlim(1, 10)
axes[1].yaxis.set_major_formatter(FuncFormatter(lambda y, _: f'{y:.0%}'))
fig.suptitle('Explained Variance of Top 10 Principal Components | 100 Trials')
sns.despine()
fig.tight_layout()
fig.subplots_adjust(top=.90)
```



Sector Analysis

1. **Analysis of PCA Loadings by Industry Sectors:** We conducted an in-depth analysis of Principal Component Analysis (PCA) loadings specific to various industry sectors. This analysis allowed us to discern how different sectors were influenced by the underlying factors identified through PCA.

```
grouped_df = factor_clusters.groupby('cluster').apply(lambda x:
x.index.tolist())
grouped_df

new_df = pd.DataFrame(grouped_df.tolist()).T

# Generate column names based on the number of clusters
column_names = [f'Cluster {i+1}' for i in range(new_df.shape[1])]
```

```
# Rename the columns
new_df.columns = column_names
for c in range(4):
   # c_stocks = stocks.loc[stocks.cluster==c].index
   print(f'---------------- Cluster {c} stocks----------------')
   dates = factor_clusters.loc[factor_clusters.cluster==c].index
   print(dates)
```

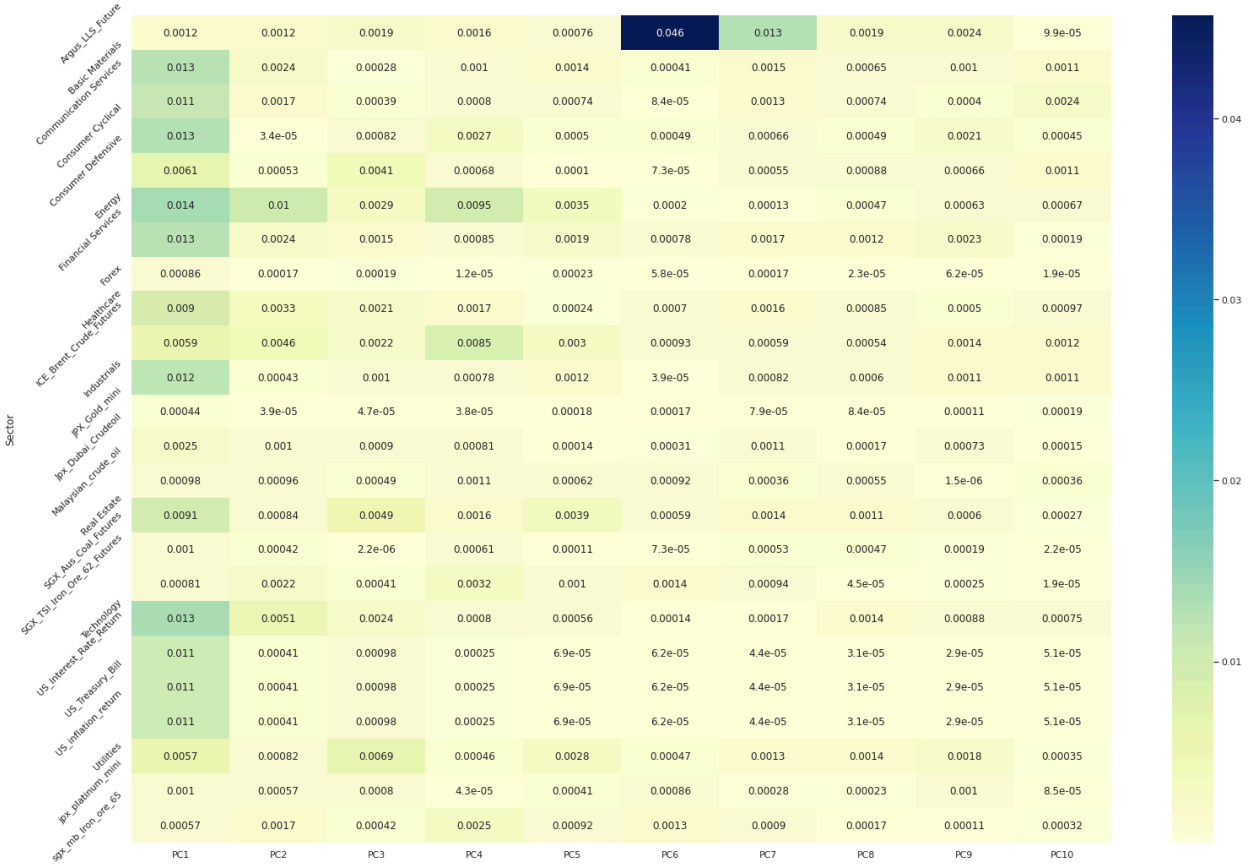
```
--------------- Cluster 0 stocks----------------
Index(['AAL', 'ACGL', 'AFL', 'AIG', 'ALB', 'ALK', 'AMP', 'AOS', 'APH',
'APTV',
       ...
       'WAB', 'WBD', 'WDC', 'WFC', 'WHR', 'WRK', 'WY', 'WYNN', 'XYL',
'ZION'],
      dtype='object', length=166)
---------------- Cluster 1 stocks----------------
Index(['AAP', 'ABBV', 'ABC', 'ABT', 'ACN', 'ADM', 'ADP', 'AEE', 'AEP',
'AES',
       ...
       'US_Interest_Rate_Return', 'US_inflation_return',
'Argus_LLS_Future',
       'sgx_mb_Iron_ore_65', 'SGX_Aus_Coal_Futures',
       'SGX_TSI_Iron_Ore_62_Futures', 'jpx_platinum_mini',
       'Jpx_Dubai_Crudeoil', 'JPX_Gold_mini', 'Malaysian_crude_oil'],
      dtype='object', length=256)
---------------- Cluster 2 stocks----------------
Index(['A', 'AAPL', 'ADBE', 'ADI', 'ADSK', 'ALGN', 'AMAT', 'AMD', 'AMZN',
       'ANET', 'ANSS', 'AVGO', 'AXON', 'BIO', 'CDAY', 'CDNS', 'CMG',
'CPRT',
       'CRL', 'CRM', 'CSGP', 'CTLT', 'DXCM', 'EBAY', 'ENPH', 'EPAM',
'ETSY',
       'EW', 'FFIV', 'FICO', 'FSLR', 'FTNT', 'GNRC', 'GOOG', 'GOOGL',
'IDXX',
       'ILMN', 'INTC', 'INTU', 'IQV', 'ISRG', 'KEYS', 'KLAC', 'LRCX', 'MA',
       'MCHP', 'MCO', 'META', 'MPWR', 'MRNA', 'MSCI', 'MSFT', 'MTCH',
'MTD',
       'MU', 'NFLX', 'NKE', 'NOW', 'NVDA', 'NXPI', 'ODFL', 'ON', 'PAYC',
       'PODD', 'PTC', 'PYPL', 'QCOM', 'QRVO', 'RVTY', 'SEDG', 'SNPS',
'STX',
       'SWKS', 'TECH', 'TER', 'TRMB', 'TSLA', 'TTWO', 'TXN', 'TYL', 'VRSN',
       'WST', 'ZBRA'],
      dtype='object')
```

```
...
Index(['APA', 'BKR', 'CF', 'COP', 'CTRA', 'CVX', 'DVN', 'EOG', 'EQT',
'FANG',
       'FCX', 'HAL', 'HES', 'KMI', 'MOS', 'MPC', 'MRO', 'OKE', 'OXY',
'PSX',
       'PXD', 'SLB', 'TRGP', 'VLO', 'WMB', 'XOM',
'ICE_Brent_Crude_Futures'],
      dtype='object')
```

2. **Heatmap Visualization of Mean Loadings by Sector:** To visually represent our findings, we generated a heatmap illustrating the mean PCA loadings across different sectors. This visualization provided insights into the relative importance of each sector in relation to the identified principal components.

```
weights = wghts.join(sector_df)

print('Mean loading/correlation by industry')
# Draw as heatmap
fig, ax = plt.subplots(figsize=(26,18))
sns.set()
sns.heatmap(weights.groupby('Sector').mean().drop(columns=["SectorCode",'Capita
lization']).abs(), cmap='YlGnBu', annot=True)
plt.yticks(rotation=45)
plt.show()
```



Through this sector analysis, we gained valuable insights into sector-specific dynamics and their contributions to overall portfolio performance, enhancing our ability to make informed investment decisions.

- **Summary of findings from PCA and clustering analysis:** PCA effectively reduced data dimensionality, identifying key components that explain most variance. Clustering revealed natural asset groupings, offering insights into sector behaviors.
- **Implications of the eigen-portfolios' performance:** Eigen-portfolios, based on top principal components, demonstrated comparable or superior returns to the market portfolio, showing promise as an alternative investment strategy.
- **Insights from the sector analysis:** Sector analysis highlighted the influence of different sectors on portfolio performance, aiding informed investment decisions by revealing sector-specific risks and opportunities.
- **Potential applications and future research directions:** PCA and eigen-portfolios can be applied to various asset classes for risk management and return optimization. Future research could integrate machine learning with PCA and expand the analysis to diverse datasets and longer periods.