Monthly Betas & Volatilities

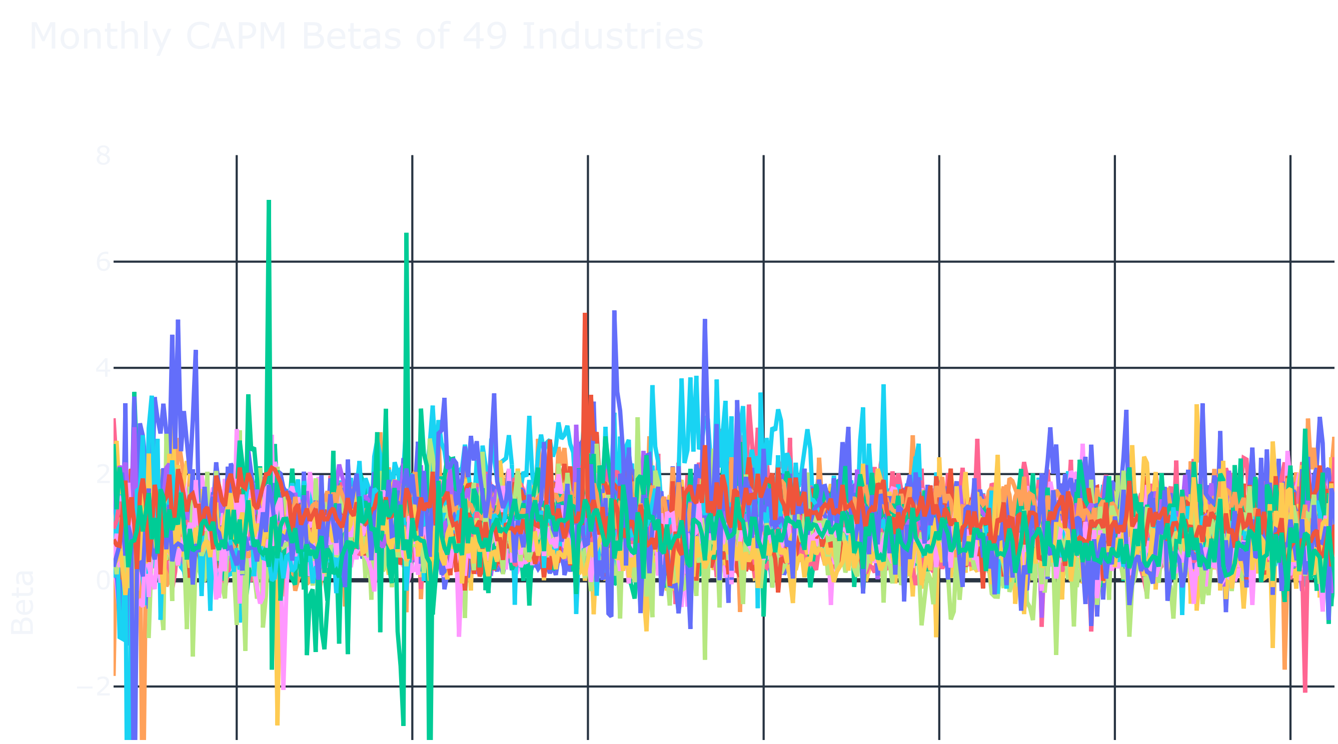
```
# Necessary Imports
import pandas as pd
import numpy as np
import plotly.graph_objects as go
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.covariance import LedoitWolf, OAS
import seaborn as sns
import matplotlib.pyplot as plt

# Load in data
def load_data(file_path, skiprows):
    df = pd.read_csv(file_path, skiprows=skiprows, dtype=str)
    df.columns = df.columns.str.strip()
    df = df.apply(lambda x: x.str.strip() if x.dtype == "object" else x)
    df = df[~df.iloc[:, 0].str.contains("Average", na=False)]
    df.iloc[:, 0] = pd.to_datetime(df.iloc[:, 0], format='%Y%m%d', errors='coerce')
    df = df.dropna(subset=[df.columns[0]])
    df.set_index(df.columns[0], inplace=True)
    df = df.apply(pd.to_numeric, errors='coerce')
    df[df == -99.99] = np.nan

    return df

# File paths
industry_file = "industry.csv"
market_file = "market.csv"

# Load the data with the correct number of rows to skip
industry_returns = load_data(industry_file, skiprows=9)
market_returns = load_data(market_file, skiprows=4)

# Small erroneous fix
#market_returns.index = pd.to_datetime(market_returns.index, errors='coerce')

# Replace -99.99 with NaN in the industry returns data
#industry_returns.replace(-99.99, np.nan, inplace=True)

print(market_returns.columns)

# Compute CAPM Betas
def compute_beta(industry_returns, market_returns):
    # Ensure that market returns have the expected columns
```

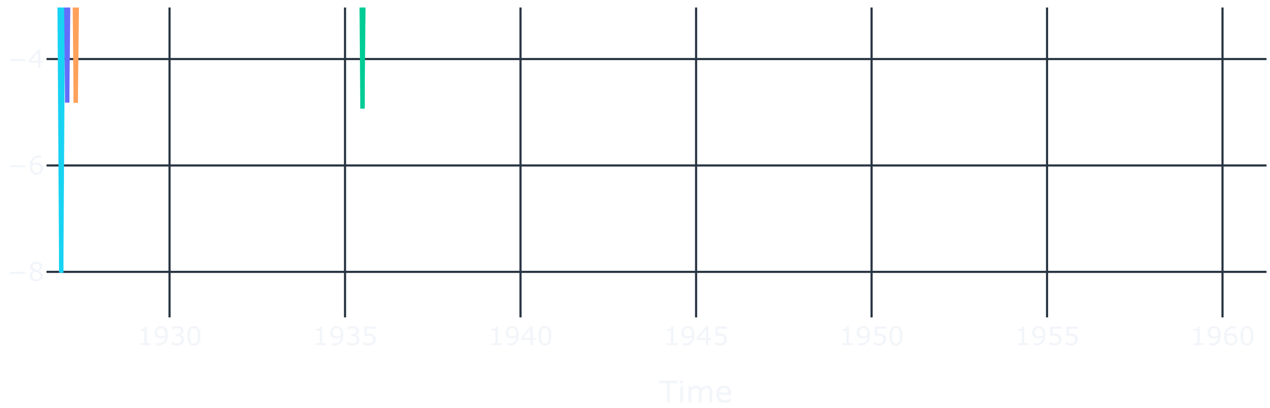
```
    # Ensure that market returns have the expected columns
    combined_returns = industry_returns.join(market_returns, how='inner', rsuffix='

    monthly_betas = {}

    # Calculate betas
    for industry in industry_returns.columns:
        monthly_betas[industry] = combined_returns.groupby(combined_returns.index.t
            lambda x: np.cov(x[industry], x['Mkt-RF'])[0, 1] / np.var(x['Mkt-RF'])
        )

    return pd.DataFrame(monthly_betas)

monthly_betas = compute_beta(industry_returns, market_returns)
threshold = 0.5
industries_to_drop = monthly_betas.columns[monthly_betas.isna().mean() > threshold]
print(f"Dropping industries: {industries_to_drop}")
monthly_betas.drop(columns=industries_to_drop, inplace=True)

# Convert the index to datetime to resolve the 'Period' issue
monthly_betas.index = monthly_betas.index.to_timestamp()

# Create the figure for Monthly Betas
fig = go.Figure()

for industry in monthly_betas.columns:
    fig.add_trace(go.Scatter(x=monthly_betas.index, y=monthly_betas[industry], mode

fig.update_layout(
    title="Monthly CAPM Betas of 49 Industries",
    xaxis_title="Time",
    yaxis_title="Beta",
    template="plotly_dark",
    height=600,
    width=800
)

fig.show()

# Compute CAPM Volatilities
def compute_volatility(industry_returns):
    return industry_returns.groupby(industry_returns.index.to_period('M')).std() *

monthly_volatilities = compute_volatility(industry_returns)
monthly_volatilities.index = monthly_volatilities.index.to_timestamp()
threshold = 0.5
industries_to_drop = monthly_volatilities.columns[monthly_volatilities.isna().mean(
print(f"Dropping industries: {industries_to_drop}")
monthly_volatilities.drop(columns=industries_to_drop, inplace=True)
```

```python
# Perform PCA on Betas
pca = PCA()
pca.fit(monthly_betas.dropna())
explained_variance = pca.explained_variance_ratio_

# Perform PCA on Volatility
pca_volatility = PCA()
pca_volatility.fit(monthly_volatilities.dropna())
explained_variance_volatility = pca_volatility.explained_variance_ratio_

# Create the figure for PCA Beta Cumulative Explained Variance
fig_pca = go.Figure()

fig_pca.add_trace(go.Scatter(
    x=list(range(1, len(explained_variance) + 1)),
    y=np.cumsum(explained_variance),
    mode='lines+markers',
    name='Cumulative Explained Variance'
))

fig_pca.update_layout(
    title="PCA of Monthly CAPM Betas",
    xaxis_title="Number of Principal Components",
    yaxis_title="Cumulative Explained Variance",
    template="plotly_dark",
    height=500,
    width=800
)

fig_pca.show()

# Create the figure for PCA Volatility Cumulative Explained Variance
fig_pca_volatility = go.Figure()
fig_pca_volatility.add_trace(go.Scatter(
    x=list(range(1, len(explained_variance_volatility) + 1)),
    y=np.cumsum(explained_variance_volatility),
    mode='lines+markers',
    name='Cumulative Explained Variance'
))
fig_pca_volatility.update_layout(
    title="PCA of Monthly Industry Volatilities",
    xaxis_title="Number of Principal Components",
    yaxis_title="Cumulative Explained Variance",
    template="plotly_dark",
    height=500,
    width=800
)
```

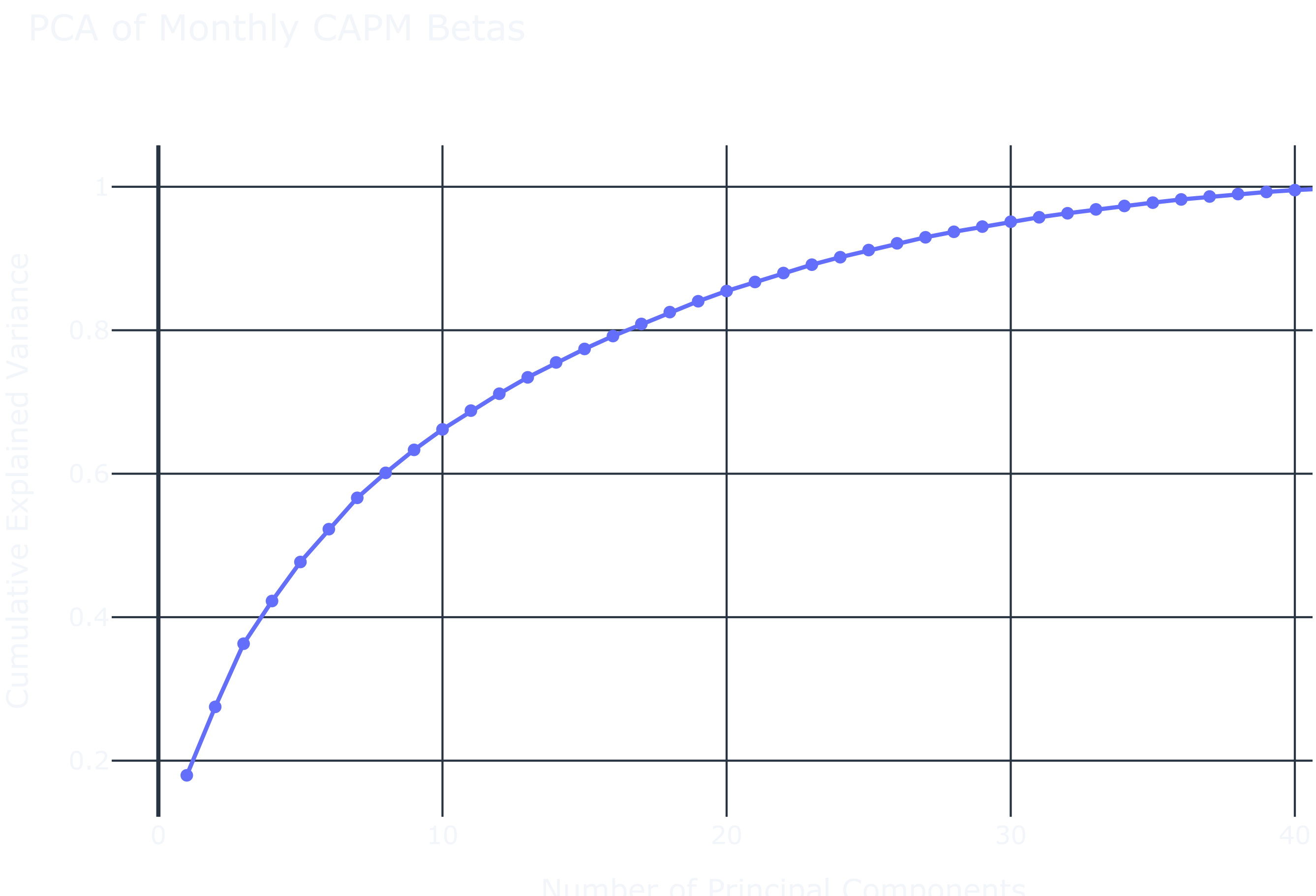
```
fig_pca_volatility.show()

# Perform PCA on Industry Betas and Industry Volatilities
pca_betas = PCA()
pca_vols = PCA()
pca_betas.fit(monthly_betas.dropna())
pca_vols.fit(monthly_volatilities.dropna())

# Extract first 3 Principal Components from both analyses
pca_betas_components = pd.DataFrame(pca_betas.transform(monthly_betas.dropna()))
pca_vols_components = pd.DataFrame(pca_vols.transform(monthly_volatilities.dropna()

# Select first 3 PCs from each
pca_betas_3 = pca_betas_components.iloc[:, :3]
pca_vols_3 = pca_vols_components.iloc[:, :3]

# Construct Correlation Matrix of 6 Principal Components
pca_combined = pd.concat([pca_betas_3, pca_vols_3], axis=1)
pca_combined.columns = ['Beta_PC1', 'Beta_PC2', 'Beta_PC3', 'Vol_PC1', 'Vol_PC2', '
correlation_matrix = pca_combined.corr()

# Plot Correlation Matrix
plt.figure(figsize=(8,6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title("Correlation Matrix of First 3 PCs from Betas and Volatilities")
plt.show()
```

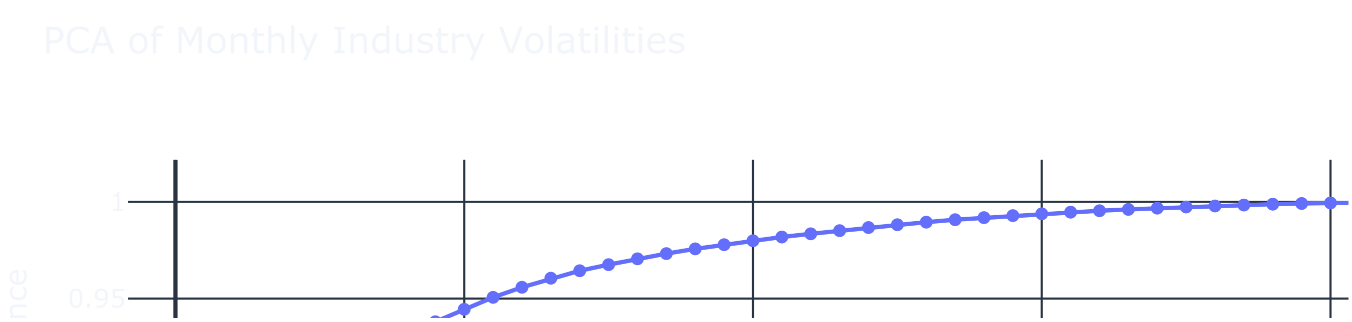
```
Index(['Mkt-RF', 'SMB', 'HML', 'RF'], dtype='object')
Dropping industries: Index(['Soda', 'Hlth', 'FabPr', 'Guns', 'Gold', 'Softw'],
```

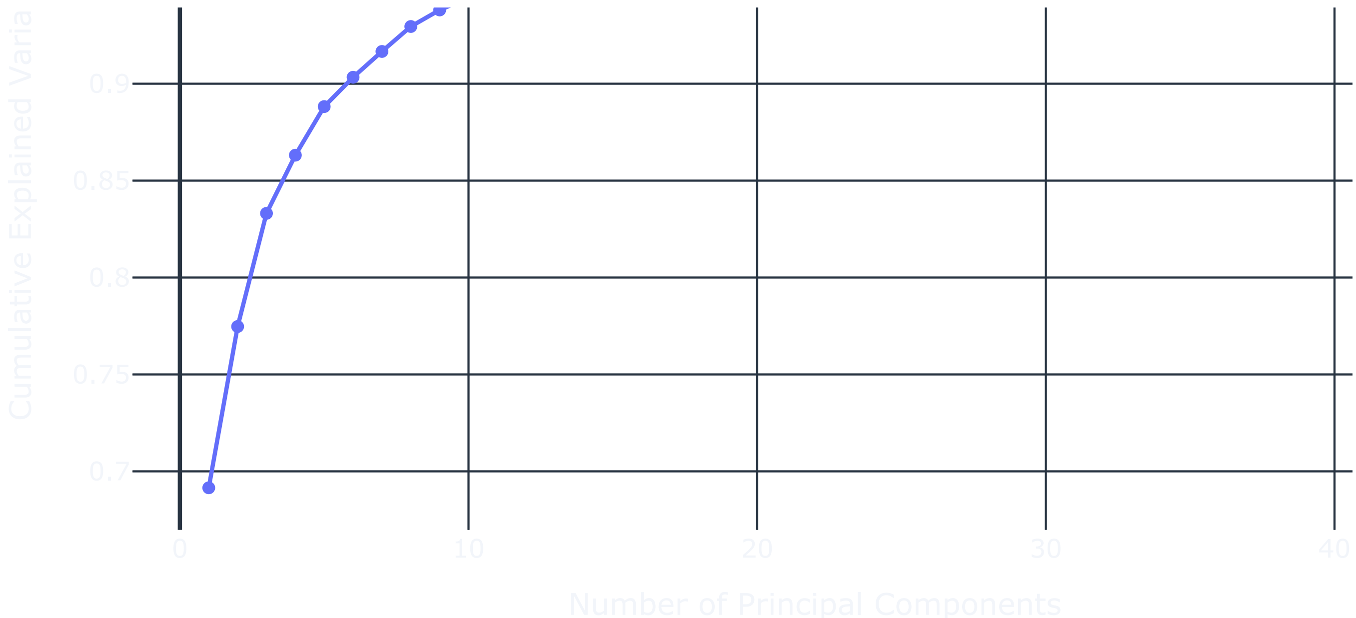


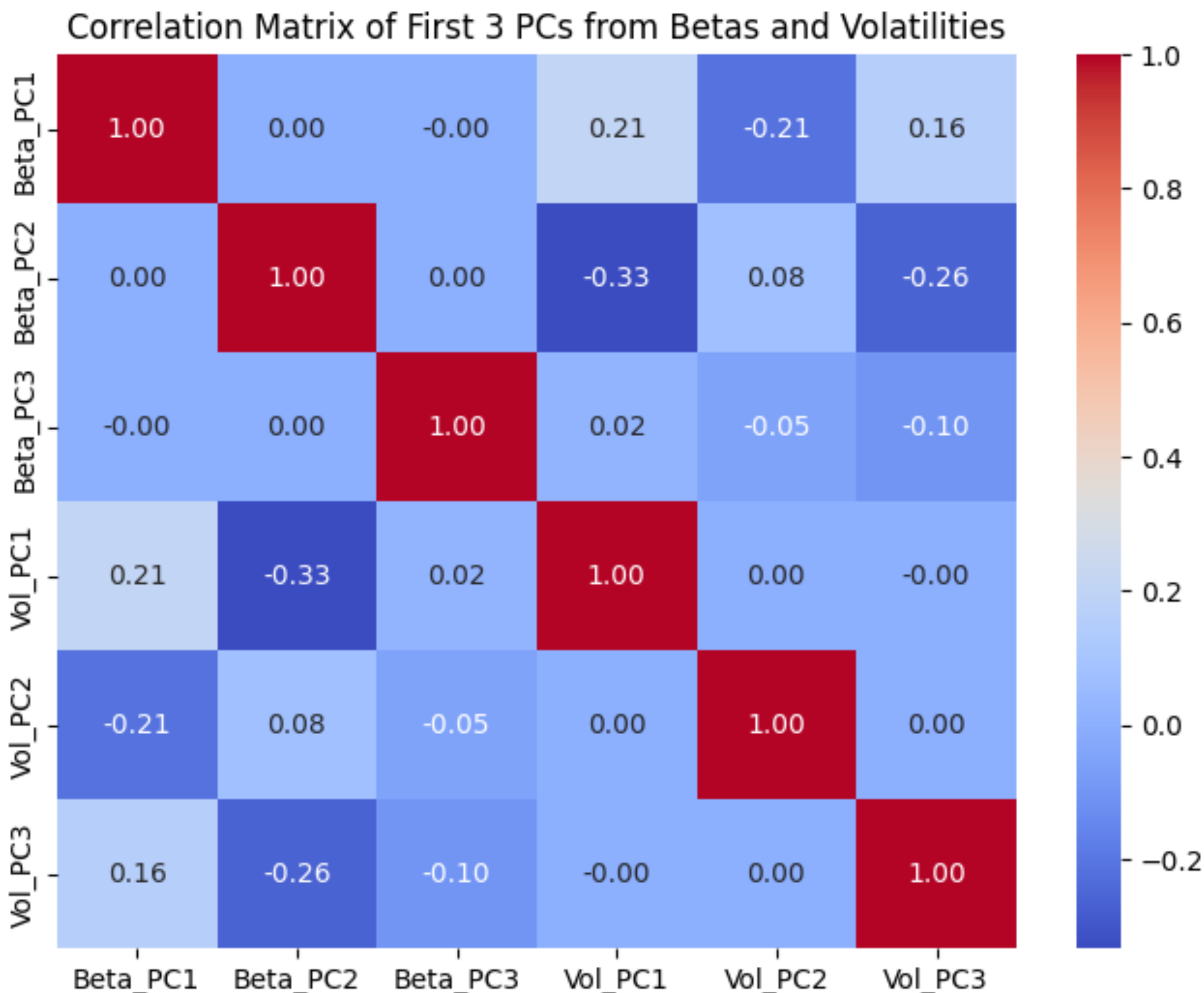



```
Dropping industries: Index(['Soda', 'Hlth', 'FabPr', 'Guns', 'Gold', 'Softw'],
```



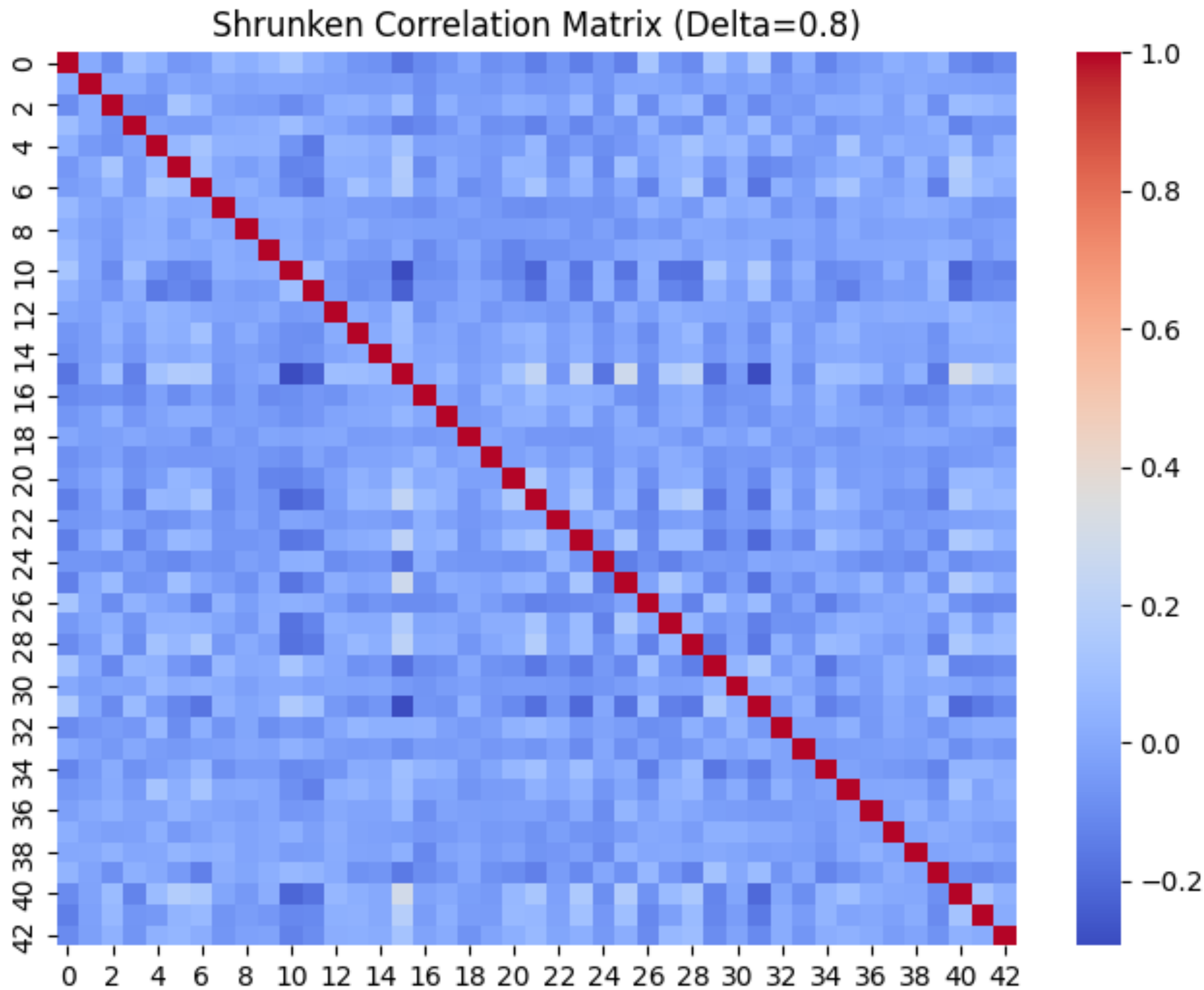



Cumulative Explained Varia
0.9
0.85
0.8
0.75
0.7
0
10
20
30
40
Number of Principal Components


Correlation Matrix of First 3 PCs from Betas and Volatilities
Beta_PC1
Beta_PC2
Beta_PC3
Vol_PC1
Vol_PC2
Vol_PC3
1.00 0.00 -0.00 0.21 -0.21 0.16
0.00 1.00 0.00 -0.33 0.08 -0.26
-0.00 0.00 1.00 0.02 -0.05 -0.10
0.21 -0.33 0.02 1.00 0.00 -0.00
-0.21 0.08 -0.05 0.00 1.00 0.00
0.16 -0.26 -0.10 -0.00 0.00 1.00
1.0
0.8
0.6
0.4
0.2
0.0
−0.2

Shrinkage

```
# Compute sample covariance matrix for monthly betas or volatilities
sample_cov_matrix = monthly_betas.dropna().cov()

# Shrinkage function
def shrink_cov(sigma, delta):
    n = sigma.shape[0]
    target = np.dot(np.identity(n), np.trace(sigma)) / n  # Target matrix: averag
    sigma_shrink = delta * target + (1 - delta) * sigma  # Shrinkage formula
    return sigma_shrink

# Compute the shrunk covariance matrix
shrinked_cov_matrix = shrink_cov(sample_cov_matrix.values, delta=0.8)

# Convert it back to a DataFrame and compute correlation
shrinked_corr_matrix = pd.DataFrame(shrinked_cov_matrix).corr()

# Plot the shrinked correlation matrix
plt.figure(figsize=(8, 6))
sns.heatmap(shrinked_corr_matrix, annot=False, cmap='coolwarm', fmt='.2f')
plt.title("Shrunken Correlation Matrix (Delta=0.8)")
plt.show()
```



Shrinkage Estimator

```
# Compute the rolling Betas on a 500 day window

def rolling_betas(industry_returns, market_returns, window=500, market_column='Mk
    rolling_betas = {}

    # Make sure the indices are aligned (common dates between industry and market
    combined_data = industry_returns.join(market_returns[market_column], how='inn

    for industry in industry_returns.columns:
        betas = []
        for i in range(window, len(combined_data)):
            # Slice the window of returns for industry and the market
            x = combined_data[industry].iloc[i-window:i]
            y = combined_data[market_column].iloc[i-window:i]
```

```
            # Calculate beta using the covariance between industry and market ret
            cov_matrix = np.cov(x, y)
            beta = cov_matrix[0, 1] / cov_matrix[1, 1]
            betas.append(beta)
        rolling_betas[industry] = betas

    return pd.DataFrame(rolling_betas, index=combined_data.index[window:])

# Call the function with the appropriate market column
rolling_betas_df = rolling_betas(industry_returns, market_returns, market_column=

threshold = 0.5
industries_to_drop = rolling_betas_df.columns[rolling_betas_df.isna().mean() > th
print(f"Dropping industries: {industries_to_drop}")
rolling_betas_df.drop(columns=industries_to_drop, inplace=True)

# Print the result
#print(rolling_betas_df.head())




# Computer the Principal Component of the Rolling Betas

# Standardize the rolling betas
scaler = StandardScaler()
scaled_betas = scaler.fit_transform(rolling_betas_df.dropna())

# Perform PCA
pca = PCA(n_components=1)  # We are interested in the first principal component
pca_rolling_betas = pca.fit(scaled_betas)

# Transform the data to the principal components
historical_trend = pca_rolling_betas.transform(scaled_betas)

# Create a DataFrame with the principal component
historical_trend_df = pd.DataFrame(historical_trend, index=rolling_betas_df.dropn

# Display the results
#historical_trend_df.head()




# Shrinkage Estimator

# Define function to apply Ledoit-Wolf and OAS shrinkage
```

```
def shrinkage_estimator(rolling_betas_df, shrinkage_method='LW'):
    if shrinkage_method == 'LW':
        estimator = LedoitWolf()
    elif shrinkage_method == 'OAS':
        estimator = OAS()
    else:
        raise ValueError("Invalid shrinkage method. Choose 'LW' or 'OAS'.")

    # Fit the estimator and get the covariance matrix for betas
    covariance_matrix = estimator.fit(rolling_betas_df).covariance_

    # Calculate shrunk betas based on the estimated covariance
    shrunk_betas = rolling_betas_df @ covariance_matrix

    return pd.DataFrame(shrunk_betas, index=rolling_betas_df.index)

# Apply shrinkage to rolling betas
shrinked_betas = shrinkage_estimator(historical_trend_df, shrinkage_method='OAS')
print(shrinked_betas)
```

```
Dropping industries: Index(['Soda', 'Hlth', 'FabPr', 'Guns', 'Gold', 'Softw'],
                    0
Unnamed: 0
1932-03-03 -38.572263
1932-03-04 -38.479969
1932-03-05 -38.462854
1932-03-07 -38.716164
1932-03-08 -38.663034
...               ...
1963-07-12 -69.561965
1963-07-15 -69.591325
1963-07-16 -69.499292
1963-07-17 -69.508925
1963-07-18 -69.512076

[7947 rows x 1 columns]
```

Ledoit-Wolf Estimation

```
# Plot the shrunk betas
plt.figure(figsize=(10, 6))
for column in shrinked_betas.columns:
    plt.plot(shrinked_betas.index, shrinked_betas[column], label=column)

plt.title("Shrunk Industry Betas")
plt.xlabel("Date")
plt.ylabel("Beta")
plt.legend(loc='upper left', bbox_to_anchor=(1, 1))
plt.tight_layout()
plt.show()
```