Code + Writing | : P

---

1. Collection (of Data)

Collection of Data

This section collects historical data for specified ETFs using the `yfinance` library. The data includes adjusted close prices for each ETF within the specified date range.

Define the Tickers

A list of Exchange-Traded Fund (ETF) symbols is defined to download data for these tickers; each ETF has a (brief) description.

Download Historical Data

Using the `yfinance` library, historical data for each ETF is downloaded within the specified date range.

Keep Adjusted Close Prices

The adjusted close prices are extracted from the downloaded data. Adjusted close prices account for dividends and stock splits.

Save Data to CSV

The adjusted close prices are saved to a CSV file named `etf_prices.csv`.

In [ ]:
```
import numpy as np

# Set the seed for reproducibility [TATLTUAE :)]
np.random.seed(42)
```

In [ ]:
```
import yfinance as yf
import pandas as pd

# List of ETF tickers and their descriptions
tickers = [
    'FXE',  # Invesco CurrencyShares Euro Trust
    'EWJ',  # iShares MSCI Japan ETF
    'GLD',  # SPDR Gold Shares
    'QQQ',  # Invesco QQQ Trust (tracks the Nasdaq-100 Index)
    'SPY',  # SPDR S&P 500 ETF Trust
    'SHV',  # iShares Short Treasury Bond ETF
    'DBA',  # Invesco DB Agriculture Fund
    'USO',  # United States Oil Fund
```

```
    'XBI',  # SPDR S&P Biotech ETF
    'ILF',  # iShares Latin America 40 ETF
    'EPP',  # iShares MSCI Pacific ex Japan ETF
    'FEZ'   # SPDR EURO STOXX 50 ETF
]

# Define the date range
start_date = '2007-03-01'
end_date = '2024-03-31'

# Download historical data for each ETF
data = yf.download(tickers, start=start_date, end=end_date)

# Only keep the adjusted close prices
adj_close = data['Adj Close']

# Save data to CSV
adj_close.to_csv('etf_prices.csv')
```

```
[*********************100%%**********************]  12 of 12 completed
```

2. Construction (of) $\mathcal{THE}$ Factor Model

Construction of the Factor Model

This section constructs the factor model using Fama-French factors. The Fama-French three-factor model is commonly used in finance to describe stock returns. It includes market risk, size risk, and value risk factors.

Load the Fama-French Factors

The Fama-French factors are loaded from a CSV file. These factors will be used to construct the factor model.

In [ ]:
```
import pandas as pd

# Load the Fama-French factors CSV
file_path = './F-F_Research_Data_Factors_daily.CSV'

# Read the CSV with specified delimiter and skip initial rows if necessary
try:
    # Check if there are any header rows to skip
    with open(file_path, 'r') as file:
        lines = file.readlines()
        for i, line in enumerate(lines[:10]):  # Inspect the first 10 lines
            print(f"Line {i + 1}: {line}")

    # Adjust the skiprows parameter based on the output
    ff_data = pd.read_csv(file_path, skiprows=4, index_col=0)
    ff_data.index = pd.to_datetime(ff_data.index, format='%Y%m%d')
    ff_data = ff_data.loc['2007-03-01':'2024-03-31']

    print(ff_data.head())  # Display the first few rows to verify
```

```
except Exception as e:
    print(f"Error reading the CSV file: {e}")
```

```
Line 1: This file was created by CMPT_ME_BEME_RETS_DAILY using the 202403 CRSP datab
ase.

Line 2: The Tbill return is the simple daily rate that, over the number of trading d
ays

Line 3: in the month, compounds to 1-month TBill rate from Ibbotson and Associates I
nc.

Line 4:

Line 5: ,Mkt-RF,SMB,HML,RF

Line 6: 19260701,    0.10,   -0.25,   -0.27,   0.009

Line 7: 19260702,    0.45,   -0.33,   -0.06,   0.009

Line 8: 19260706,    0.17,    0.30,   -0.39,   0.009

Line 9: 19260707,    0.09,   -0.58,    0.02,   0.009

Line 10: 19260708,    0.21,   -0.38,    0.19,   0.009

Error reading the CSV file: time data "Copyright 2024 Kenneth R. French" doesn't mat
ch format "%Y%m%d", at position 25710. You might want to try:
    - passing `format` if your strings have a consistent format;
    - passing `format='ISO8601'` if your strings are all ISO8601 but not necessarily
in exactly the same format;
    - passing `format='mixed'`, and the format will be inferred for each element ind
ividually. You might want to use `dayfirst` alongside this.
```

Explanation

1. **Import Libraries**:
    - We import the `pandas` library for data manipulation.
2. **Load the Fama-French Factors**:
    - **file_path**: Specifies the path to the Fama-French factors CSV file.
    - **try block**: Handles reading the CSV file and manages potential errors.
    - **Inspecting the first 10 lines**: Checks the initial rows to determine the header and data format.
    - **skiprows parameter**: Adjusted based on the CSV inspection to skip non-data rows.
    - **pd.read_csv**: Reads the CSV file into a DataFrame, skipping the specified rows.
    - **pd.to_datetime**: Converts the index to datetime format.
    - **ff_data.loc**: Filters the data to the specified date range ('2007-03-01' to '2024-03-31').
    - **print(ff_data.head())**: Displays the first few rows of the DataFrame for verification.

Factor Model Construction and Analysis

In this section, we will load and process the Fama-French factors, merge them with ETF returns, and estimate the factor loadings for each ETF.

Load and Process Fama-French Factors

This function loads the Fama-French factors from a CSV file, processes the data, and returns it for the specified date range.

In [ ]:
```
import pandas as pd
import statsmodels.api as sm

# Function to load and process Fama-French factors
def fama_french_factors(start_date, end_date, file_path):
    try:
        # Load the Fama-French factors data, skipping the first 4 metadata lines
        ff_data = pd.read_csv(file_path, skiprows=3, index_col=0)
        ff_data.index = pd.to_datetime(ff_data.index, format='%Y%m%d', errors='coer

        # Drop rows with invalid dates
        ff_data = ff_data.dropna()

        # Sort the index to ensure it is in chronological order
        ff_data = ff_data.sort_index()

        # Slice the data for the specified date range
        ff_data = ff_data.loc[start_date:end_date]
        return ff_data
    except Exception as e:
        print(f"Error processing the Fama-French data: {e}")
        return None

# Usage example
start_date = '2007-03-01'
end_date = '2024-03-31'
file_path = './F-F_Research_Data_Factors_daily.CSV'

ff_factors = fama_french_factors(start_date, end_date, file_path)

# Proceed with the analysis
if ff_factors is not None:
    etf_data = pd.read_csv('etf_prices.csv', index_col=0, parse_dates=True)
    etf_returns = etf_data.pct_change().dropna()

    # Merge ETF returns with Fama-French factors
    merged_data = etf_returns.join(ff_factors, how='inner')

    # Estimate factor loadings for each ETF
    factor_loadings = {}
    for ticker in etf_data.columns:
        model = sm.OLS(merged_data[ticker], sm.add_constant(merged_data[['Mkt-RF',
        results = model.fit()
```

```
        factor_loadings[ticker] = results.params

    # Convert factor loadings to a DataFrame
    factor_loadings_df = pd.DataFrame(factor_loadings).T
    factor_loadings_df.columns = ['Alpha', 'Mkt-RF', 'SMB', 'HML']
    factor_loadings_df.to_csv('factor_loadings.csv')

    print(factor_loadings_df.head())
```

```
        Alpha    Mkt-RF       SMB       HML
DBA -0.000032  0.002360 -0.000186  0.000630
EPP -0.000142  0.010475 -0.000677  0.001387
EWJ -0.000139  0.007755 -0.000921  0.000244
FEZ -0.000184  0.011116 -0.000658  0.001652
FXE -0.000074  0.001090  0.000042  0.000097
```

Explanation

1. **Import Libraries**:
   - `pandas` is used for data manipulation.
   - `statsmodels.api` is used for statistical models, including OLS regression.
2. **Function to Load and Process Fama-French Factors**:
   - Loads the Fama-French factors data, skipping metadata lines.
   - Converts the index to datetime format and drops invalid dates.
   - Sorts the index and slices the data for the specified date range.
3. **Usage Example**:
   - Defines the date range and file path.
   - Calls the `fama_french_factors` function to load and process the Fama-French factors.
   - Loads ETF prices and calculates daily returns.
4. **Merge ETF Returns with Fama-French Factors**:
   - Merges ETF returns with Fama-French factors using an inner join.
5. **Estimate Factor Loadings**:
   - For each ETF, estimates factor loadings (coefficients) using OLS regression.
   - Converts the results to a DataFrame and saves to a CSV file.

2.5 Analysis of Estimator Coefficients

In this section, we analyze the estimated factor loadings (coefficients) for a subset of ETFs. The coefficients are derived from the Fama-French three-factor model, which includes Alpha, Market Risk Premium (Mkt-RF), Size (SMB), and Value (HML) factors.

Synthesis (Executive Summary)

The analysis of the estimator coefficients in percentage terms provides insights into the risk factors affecting each ETF. The negative alphas suggest that these ETFs have underperformed relative to the model's expectations. The varying coefficients for market risk premium, size, and value factors highlight the different sensitivities and exposures of these ETFs to market conditions, size segments, and value versus growth stocks. Notably, FXE has a positive SMB coefficient, indicating a preference for small-cap stocks.

Estimated Coefficients in Absolute Basis

Here are the estimated coefficients for the selected ETFs:

ETF	Alpha	Mkt-RF	SMB	HML
DBA	-0.000032	0.002360	-0.000186	0.000630
EPP	-0.000142	0.010475	-0.000677	0.001387
EWJ	-0.000139	0.007755	-0.000921	0.000244
FEZ	-0.000184	0.011116	-0.000657	0.001652
FXE	-0.000074	0.001090	0.000042	0.000097

Estimated Coefficients in Percentage Basis

Here are the estimated coefficients for the selected ETFs, displayed in percentage terms:

ETF	Alpha (%)	Mkt-RF (%)	SMB (%)	HML (%)
DBA	-0.0032	0.2360	-0.0186	0.0630
EPP	-0.0142	1.0475	-0.0677	0.1387
EWJ	-0.0139	0.7755	-0.0921	0.0244
FEZ	-0.0184	1.1116	-0.0657	0.1652
FXE	-0.0074	0.1090	0.0042	0.0097

Interpretation in Absolute Basis

Alpha

- **Alpha** represents the ETF's performance relative to the expected return based on the three-factor model. A positive alpha indicates the ETF has outperformed the model's prediction, while a negative alpha suggests underperformance.

- **Observation**: All selected ETFs have negative alpha values, indicating underperformance relative to the model's prediction.

Market Risk Premium (Mkt-RF)

- **Mkt-RF** represents the sensitivity of the ETF's returns to the market risk premium (the excess return of the market over the risk-free rate). Higher values indicate higher sensitivity to market movements.
- **Observation**: FEZ and EPP have the highest market risk premiums, suggesting they are more sensitive to market movements. FXE has the lowest sensitivity among the selected ETFs.

Size (SMB)

- **SMB** (Small Minus Big) represents the ETF's sensitivity to returns of small-cap stocks relative to large-cap stocks. A positive value indicates higher sensitivity to small-cap stocks.
- **Observation**: Most selected ETFs have negative SMB coefficients, indicating a preference or higher exposure to large-cap stocks over small-cap stocks. However, FXE has a positive SMB coefficient, indicating a slight preference for small-cap stocks.

Value (HML)

- **HML** (High Minus Low) represents the ETF's sensitivity to value stocks (high book-to-market ratio) relative to growth stocks (low book-to-market ratio). A positive value indicates higher sensitivity to value stocks.
- **Observation**: FEZ and EPP show higher sensitivity to value stocks, while EWJ shows lower sensitivity.

Interpretation in Percentage Basis

Alpha

- **Alpha (%)** represents the ETF's performance relative to the expected return based on the three-factor model, expressed in percentage terms.
- **Observation**: All selected ETFs have negative alpha values, indicating underperformance relative to the model's prediction.

Market Risk Premium (Mkt-RF)

- **Mkt-RF (%)** represents the sensitivity of the ETF's returns to the market risk premium, expressed in percentage terms.

- **Observation**: FEZ and EPP have the highest market risk premiums, suggesting they are more sensitive to market movements. FXE has the lowest sensitivity among the selected ETFs.

Size (SMB)

- **SMB (%)** represents the ETF's sensitivity to returns of small-cap stocks relative to large-cap stocks, expressed in percentage terms.
- **Observation**: Most selected ETFs have negative SMB coefficients, indicating a preference or higher exposure to large-cap stocks over small-cap stocks. However, FXE has a positive SMB coefficient, indicating a slight preference for small-cap stocks.

Value (HML)

- **HML (%)** represents the ETF's sensitivity to value stocks (high book-to-market ratio) relative to growth stocks (low book-to-market ratio), expressed in percentage terms.
- **Observation**: FEZ and EPP show higher sensitivity to value stocks, while EWJ shows lower sensitivity.

In [ ]:
```
import pandas as pd

# Create a DataFrame for the coefficients in percentage terms
coefficients_pct = {
    'ETF': ['DBA', 'EPP', 'EWJ', 'FEZ', 'FXE'],
    'Alpha (%)': [-0.0032, -0.0142, -0.0139, -0.0184, -0.0074],
    'Mkt-RF (%)': [0.2360, 1.0475, 0.7755, 1.1116, 0.1090],
    'SMB (%)': [-0.0186, -0.0677, -0.0921, -0.0657, 0.0042],
    'HML (%)': [0.0630, 0.1387, 0.0244, 0.1652, 0.0097]
}

df_pct = pd.DataFrame(coefficients_pct)
df_pct.set_index('ETF', inplace=True)
print(df_pct)
```

```
     Alpha (%)  Mkt-RF (%)  SMB (%)  HML (%)
ETF
DBA    -0.0032      0.2360  -0.0186   0.0630
EPP    -0.0142      1.0475  -0.0677   0.1387
EWJ    -0.0139      0.7755  -0.0921   0.0244
FEZ    -0.0184      1.1116  -0.0657   0.1652
FXE    -0.0074      0.1090   0.0042   0.0097
```

3. Optim(ization)

3.05 misc

Validate Covariance Matrices

We need to ensure that our covariance matrices are symmetric/Hermitian. This is crucial for accurate risk and portfolio analysis.

In [ ]:
```
import numpy as np

# Load ETF returns
etf_returns = pd.read_csv('etf_prices.csv', index_col=0, parse_dates=True).pct_chan

# Calculate the covariance matrix
cov_matrix = etf_returns.cov()

# Check if the covariance matrix is symmetric
is_symmetric = np.allclose(cov_matrix, cov_matrix.T)
print(f"Covariance matrix is symmetric: {is_symmetric}")

# If using complex numbers, also check if it is Hermitian
if not is_symmetric:
    is_hermitian = np.allclose(cov_matrix, cov_matrix.T.conj())
    print(f"Covariance matrix is Hermitian: {is_hermitian}")
```

```
Covariance matrix is symmetric: True
```

Explanation

1. **Load ETF Returns**:
   - Read the ETF price data from `etf_prices.csv`.
   - Calculate the daily returns using the percentage change method.
2. **Calculate Covariance Matrix**:
   - Use `etf_returns.cov()` to compute the covariance matrix of the ETF returns.
3. **Check Symmetry**:
   - Use `np.allclose(cov_matrix, cov_matrix.T)` to verify if the covariance matrix is symmetric.
4. **Check Hermitian (if necessary)**:
   - If the covariance matrix is not symmetric and involves complex numbers, use `np.allclose(cov_matrix, cov_matrix.T.conj())` to check if it is Hermitian.

Results

- The output confirms whether the covariance matrix is symmetric. A symmetric covariance matrix indicates that the variance between any two assets is consistent in both directions.

Result

- **Covariance matrix is symmetric: True**

The covariance matrix is symmetric, indicating that the variance between any two assets is consistent in both directions. This validation ensures accurate risk and portfolio analysis.

Investment Strategies Explanation

In this section, we will discuss the investment strategies that will be implemented and backtested using the factor model. The strategies aim to leverage the information provided by the factor loadings to generate returns.

$3.1\ \text{Strat}\ \mathcal{I}$

Optimization Strategy I

This strategy uses convex optimization to determine the optimal weights for a portfolio based on expected returns, the covariance matrix, and factor loadings. The objective is to maximize the return minus the risk-adjusted return, subject to certain constraints.

Optimization Strategy I Analysis

Executive Summary

In this analysis, we derived the optimal weights for a portfolio using a convex optimization strategy. The objective was to maximize the return minus the risk-adjusted return, subject to certain constraints. The strategy provided a highly leveraged portfolio with significant long and short positions across different asset classes and regions. The analysis includes both absolute and relative (percentage) basis comparisons.

Optimal Weights for Strategy I

The optimal weights for the portfolio, determined using the convex optimization strategy, are as follows:

$$\text{Optimal (Absolute, Relative) Weights for Strategy I :}$$

- $$\text{FXE} : 1.57452868 \equiv 7.108791\%;$$
- $$\text{EWJ} : -1.999971 \equiv -9.029605\%;$$
- $$\text{GLD} : -1.999993 \equiv -9.029705\%;$$
- $$\text{QQQ} : -1.999993 \equiv -9.029706\%;$$

- $\text{SPY} : -1.999993 \equiv -9.029705\%;$
- $\text{SHV} : 1.999999 \equiv 9.029734\%;$
- $\text{DBA} : -0.574572 \equiv -2.594118\%;$
- $\text{USO} : 1.999999 \equiv 9.029733\%;$
- $\text{XBI} : 1.999996 \equiv 9.029719\%;$
- $\text{ILF} : 1.999998 \equiv 9.029728\%;$
- $\text{EPP} : -1.999998 \equiv -9.029727\%;$
- $\text{FEZ} : 1.999998 \equiv 9.029730\%.$

Weight Analysis in Relative (Percentage) Terms

To provide a clearer understanding, we convert the absolute weights into relative (percentage) terms.

In [ ]:
```
# Convert absolute weights to percentage weights
total_weight = sum(np.abs(optimal_weights_I))
relative_weights = (optimal_weights_I / total_weight) * 100

# Create a DataFrame for better visualization
etf_names = ['FXE', 'EWJ', 'GLD', 'QQQ', 'SPY', 'SHV', 'DBA', 'USO', 'XBI', 'ILF',
weights_df = pd.DataFrame({
    'ETF': etf_names,
    'Absolute Weight': optimal_weights_I,
    'Relative Weight (%)': relative_weights
})

weights_df.set_index('ETF', inplace=True)
print(weights_df)
```

```
     Absolute Weight  Relative Weight (%)
ETF
FXE         1.574531             7.108799
EWJ        -1.999971            -9.029603
GLD        -1.999993            -9.029704
QQQ        -1.999993            -9.029704
SPY        -1.999993            -9.029703
SHV         1.999999             9.029732
DBA        -0.574574            -2.594127
USO         1.999999             9.029731
XBI         1.999996             9.029717
ILF         1.999998             9.029727
EPP        -1.999998            -9.029725
FEZ         1.999998             9.029728
```

Interpretation

Positive Weights

- **FXE: 1.574529 (7.1088%)**: Long position in the Invesco CurrencyShares Euro Trust.
- **SHV: 1.999999 (9.0297%)**: Long position in the iShares Short Treasury Bond ETF.
- **USO: 1.999999 (9.0297%)**: Long position in the United States Oil Fund.
- **XBI: 1.999996 (9.0297%)**: Long position in the SPDR S&P Biotech ETF.
- **ILF: 1.999998 (9.0297%)**: Long position in the iShares Latin America 40 ETF.
- **FEZ: 1.999998 (9.0297%)**: Long position in the SPDR EURO STOXX 50 ETF.

Negative Weights

- **EWJ: -1.999971 (-9.0296%)**: Short position in the iShares MSCI Japan ETF.
- **GLD: -1.999993 (-9.0297%)**: Short position in the SPDR Gold Shares.
- **QQQ: -1.999993 (-9.0297%)**: Short position in the Invesco QQQ Trust.
- **SPY: -1.999993 (-9.0297%)**: Short position in the SPDR S&P 500 ETF Trust.
- **DBA: -0.574572 (-2.5941%)**: Short position in the Invesco DB Agriculture Fund.
- **EPP: -1.999998 (-9.0297%)**: Short position in the iShares MSCI Pacific ex Japan ETF.

Synthesis

The optimal weights derived from the optimization strategy show a highly leveraged portfolio with significant long and short positions. The strategy indicates a strong conviction in the expected returns derived from the factor model, leading to a diverse exposure across different asset classes and regions. This high leverage reflects a strategy that aims to maximize returns by taking advantage of market inefficiencies and leveraging factor exposures.

- **Positive Weights**: Indicate a long position in ETFs such as FXE, SHV, USO, XBI, ILF, and FEZ.
- **Negative Weights**: Indicate a short position in ETFs such as EWJ, GLD, QQQ, SPY, DBA, and EPP.
- **Relative Weight Analysis**: Provides insights into the proportion of the portfolio allocated to each ETF, showing a balanced yet leveraged approach.

The detailed weight analysis in both absolute and relative terms helps in understanding the portfolio composition and the strategic decisions based on the factor model.

In [ ]:
```
import numpy as np
import cvxpy as cp

def optimize_strategy_I(expected_returns, cov_matrix, factor_loadings, beta_constra
    n = len(expected_returns)
    w = cp.Variable(n)
    portfolio_return = expected_returns @ w

    # Ensure the covariance matrix is symmetric
    cov_matrix = (cov_matrix + cov_matrix.T) / 2
```

```
    # Check if the covariance matrix is symmetric/Hermitian
    def is_symmetric(matrix):
        return np.allclose(matrix, matrix.T)

    if not is_symmetric(cov_matrix):
        raise ValueError("Covariance matrix is not symmetric/Hermitian.")

    portfolio_risk = cp.quad_form(w, cov_matrix)

    # Calculate the portfolio beta using factor loadings
    portfolio_beta = factor_loadings['Mkt-RF'].values @ w

    constraints = [
        cp.sum(w) == 1,
        portfolio_beta >= beta_constraints[0],
        portfolio_beta <= beta_constraints[1],
        w >= -2,
        w <= 2
    ]

    # Objective function: maximizing return minus risk-adjusted return
    objective = cp.Maximize(portfolio_return - lambd * cp.norm(portfolio_risk, 2))
    prob = cp.Problem(objective, constraints)
    prob.solve()

    return w.value
```

Explanation

1. **Initialization**:
   - `n` : Number of assets.
   - `w` : A variable representing the weights of the assets in the portfolio.
   - `portfolio_return` : The expected return of the portfolio, calculated as the dot product of expected returns and weights.
2. **Symmetric Covariance Matrix**:
   - Ensure the covariance matrix is symmetric by averaging it with its transpose.
3. **Symmetry Check**:
   - `is_symmetric` : Function to check if the covariance matrix is symmetric.
   - Raise an error if the covariance matrix is not symmetric/Hermitian.
4. **Portfolio Risk**:
   - `portfolio_risk` : Calculated using the quadratic form of the weights and covariance matrix.
5. **Portfolio Beta**:

- `portfolio_beta`: Calculated as the dot product of the market risk premium factor loadings and weights.

6. **Constraints**:
   - Sum of weights equals 1.
   - Portfolio beta is within the specified constraints.
   - Individual weights are bounded between -2 and 2.
7. **Objective Function**:
   - Maximize the return minus the risk-adjusted return, using `lambda` as the risk aversion parameter.
8. **Solve the Optimization Problem**:
   - Define and solve the optimization problem using CVXPY.

Example Usage

We will now use the `optimize_strategy_I` function with example inputs to determine the optimal weights for the portfolio.

In [ ]:
```
# Example usage
beta_constraints = [-0.5, 0.5]
lambd = 0.1
expected_returns = factor_loadings_df['Alpha'].values
cov_matrix = etf_returns.cov().values

optimal_weights_I = optimize_strategy_I(expected_returns, cov_matrix, factor_loadin
print("Optimal weights for Strategy I:", optimal_weights_I)
```

```
Optimal weights for Strategy I: [ 1.57453086 -1.99997057 -1.99999286 -1.99999299 -1.
99999269  1.9999992
 -0.5745743   1.99999898  1.99999577  1.99999792 -1.99999754  1.99999822]
```

3.2 Strat $\mathcal{II}$

Optimization Strategy II Analysis

Executive Summary

In this analysis, we derived the optimal weights for a portfolio using a different optimization strategy, Strategy II. This strategy aims to minimize the tracking error volatility while maximizing the expected return, subject to certain constraints. The analysis includes both absolute and relative (percentage) basis comparisons.

Optimal Weights for Strategy II

The optimal weights for the portfolio, determined using Strategy II, are as follows:

$$\text{Optimal (Absolute, Relative) Weights for Strategy II:}$$

- $$\text{FXE} : 0.02058607 \equiv 1.769667\%;$$
- $$\text{EWJ} : 0.10726133 \equiv 9.219518\%;$$
- $$\text{GLD} : 0.09340871 \equiv 8.030507\%;$$
- $$\text{QQQ} : 0.10845783 \equiv 9.319907\%;$$
- $$\text{SPY} : 0.02415547 \equiv 2.076501\%;$$
- $$\text{SHV} : 0.0381061 \equiv 3.276467\%;$$
- $$\text{DBA} : 0.08879188 \equiv 7.626430\%;$$
- $$\text{USO} : 0.19751439 \equiv 16.957171\%;$$
- $$\text{XBI} : 0.05704953 \equiv 4.899822\%;$$
- $$\text{ILF} : 0.17376249 \equiv 14.916309\%;$$
- $$\text{EPP} : -0.05986654 \equiv -5.140592\%;$$
- $$\text{FEZ} : 0.15077273 \equiv 12.458295.$$

Weight Analysis

Absolute and Relative Weights

Below is a table showing the absolute weights and their corresponding relative (percentage) weights for each ETF:

ETF	Absolute Weight	Relative Weight (%)
FXE	0.020586	1.7697
EWJ	0.107261	9.2195
GLD	0.093409	8.0305
QQQ	0.108458	9.3199
SPY	0.024156	2.0765
SHV	0.038106	3.2765
DBA	0.088792	7.6264

ETF	Absolute Weight	Relative Weight (%)
USO	0.197514	16.9572
XBI	0.057050	4.8998
ILF	0.173762	14.9163
EPP	-0.059867	-5.1406
FEZ	0.150773	12.4583

Interpretation

Positive Weights

- **FXE**: 0.020586 (1.7697%)
- **EWJ**: 0.107261 (9.2195%)
- **GLD**: 0.093409 (8.0305%)
- **QQQ**: 0.108458 (9.3199%)
- **SPY**: 0.024155 (2.0765%)
- **SHV**: 0.038106 (3.2765%)
- **DBA**: 0.088792 (7.6264%)
- **USO**: 0.197514 (16.9572%)
- **XBI**: 0.057050 (4.8998%)
- **ILF**: 0.173762 (14.9163%)
- **FEZ**: 0.150773 (12.4583%)

Negative Weights

- **EPP**: -0.059867 (-5.1406%)

Summary

```
## Summary

The optimal weights derived from Strategy II show a portfolio with both
long and short positions. The strategy minimizes tracking error volatility
while maximizing returns, leading to a balanced yet diversified portfolio.
The detailed weight analysis in both absolute and relative terms helps in
understanding the portfolio composition and the strategic decisions based
on the factor model.

- **Positive Weights**: Indicate a long position in ETFs such as FXE, EWJ,
GLD, QQQ, SPY, SHV, DBA, USO, XBI, ILF, and FEZ.
- **Negative Weights**: Indicate a short position in the ETF EPP.
- **Relative Weight Analysis**: Provides insights into the proportion of
the portfolio allocated to each ETF, showing a balanced yet leveraged
approach.
```

Strategy II: Optimization Function

We will implement and analyze the optimization strategy that minimizes tracking error volatility.

Import Libraries

```
from scipy.optimize import minimize
import numpy as np
import pandas as pd
```

Define the Tracking Error Volatility Function

This function calculates the tracking error volatility, which is the standard deviation of the difference between the portfolio returns and benchmark returns.

```
def tracking_error_volatility(weights, returns_data, benchmark_returns):

    # Calculate portfolio returns
    portfolio_returns = returns_data @ weights

    # Calculate tracking error volatility
    return np.sqrt(np.var(portfolio_returns - benchmark_returns))
```

Define the Optimization Function for Strategy II

This function, `optimize_strategy_II`, takes the expected returns, returns data, factor loadings, beta constraints, a risk aversion parameter (lambda), and benchmark returns to optimize the portfolio weights.

```
def optimize_strategy_II(expected_returns, returns_data, factor_loadings,
beta_constraints, lambd, benchmark_returns):
    n = len(expected_returns)

    def objective(weights):
        portfolio_return = expected_returns @ weights
        te_vol = tracking_error_volatility(weights, returns_data,
benchmark_returns)
        return -(portfolio_return - lambd * te_vol)

    constraints = [
        {'type': 'eq', 'fun': lambda w: np.sum(w) - 1},
        {'type': 'ineq', 'fun': lambda w: beta_constraints[1] - np.sum(w *
factor_loadings['Mkt-RF'].values)},
        {'type': 'ineq', 'fun': lambda w: np.sum(w * factor_loadings['Mkt-
RF'].values) - beta_constraints[0]},
        {'type': 'ineq', 'fun': lambda w: 2 - w},
```

```
        {'type': 'ineq', 'fun': lambda w: w + 2}
    ]

    bounds = [(-2, 2) for _ in range(n)]
    result = minimize(objective, np.ones(n) / n, bounds=bounds,
constraints=constraints)
    return result.x
```

Example Usage

We will now use the `optimize_strategy_II` function with example inputs to determine the optimal weights for the portfolio.

```
# Example usage
beta_constraints = [-2, 2]
lambd = 0.1
benchmark_returns = etf_returns['SPY'].values
returns_data = etf_returns.values  # Adjusted to use returns data directly

optimal_weights_II = optimize_strategy_II(expected_returns, returns_data,
factor_loadings_df, beta_constraints, lambd, benchmark_returns)
print("Optimal weights for Strategy II:", optimal_weights_II)
```

```
Optimal weights for Strategy II: [ 0.02058607  0.10726133  0.09340871
0.10845783  0.02415547  0.0381061
  0.08879188  0.19751439  0.05704953  0.17376249 -0.05986654  0.15077273]
```

(Code To) Display (Relative) Weights in Tabular Format

```
# Output the optimal weights for better readability
etf_names = ['FXE', 'EWJ', 'GLD', 'QQQ', 'SPY', 'SHV', 'DBA', 'USO',
'XBI', 'ILF', 'EPP', 'FEZ']
weights_df_II = pd.DataFrame({
    'ETF': etf_names,
    'Absolute Weight': optimal_weights_II
})

# Convert absolute weights to percentage weights
total_weight_II = sum(np.abs(optimal_weights_II))
relative_weights_II = (optimal_weights_II / total_weight_II) * 100

weights_df_II['Relative Weight (%)'] = relative_weights_II
weights_df_II.set_index('ETF', inplace=True)
print(weights_df_II)
```

(Code) Analysis of Optimal Weights for Strategy II

The optimal weights for the portfolio, determined using Strategy II, are as follows:

In [ ]:
```
from scipy.optimize import minimize
import numpy as np
import pandas as pd

def tracking_error_volatility(weights, returns_data, benchmark_returns):
    # Calculate portfolio returns
    portfolio_returns = returns_data @ weights
    # Calculate tracking error volatility
    return np.sqrt(np.var(portfolio_returns - benchmark_returns))

def optimize_strategy_II(expected_returns, returns_data, factor_loadings, beta_cons
    n = len(expected_returns)

    def objective(weights):
        portfolio_return = expected_returns @ weights
        te_vol = tracking_error_volatility(weights, returns_data, benchmark_returns
        return -(portfolio_return - lambd * te_vol)

    constraints = [
        {'type': 'eq', 'fun': lambda w: np.sum(w) - 1},
        {'type': 'ineq', 'fun': lambda w: beta_constraints[1] - np.sum(w * factor_l
        {'type': 'ineq', 'fun': lambda w: np.sum(w * factor_loadings['Mkt-RF'].valu
        {'type': 'ineq', 'fun': lambda w: 2 - w},
        {'type': 'ineq', 'fun': lambda w: w + 2}
    ]

    bounds = [(-2, 2) for _ in range(n)]
    result = minimize(objective, np.ones(n) / n, bounds=bounds, constraints=constra
    return result.x
```

In [ ]:
```
# Example usage
beta_constraints = [-2, 2]
lambd = 0.1
benchmark_returns = etf_returns['SPY'].values
returns_data = etf_returns.values  # Adjusted to use returns data directly

optimal_weights_II = optimize_strategy_II(expected_returns, returns_data, factor_lo
print("Optimal weights for Strategy II:", optimal_weights_II)
```

```
Optimal weights for Strategy II: [ 0.02058613  0.10726124  0.09340882  0.10845798
0.02415555  0.038106
  0.08879178  0.19751432  0.05704953  0.1737624  -0.05986645  0.15077271]
```

In [ ]:
```
import pandas as pd

# Output the optimal weights for better readability
etf_names = ['FXE', 'EWJ', 'GLD', 'QQQ', 'SPY', 'SHV', 'DBA', 'USO', 'XBI', 'ILF',
weights_df_II = pd.DataFrame({
    'ETF': etf_names,
    'Absolute Weight': optimal_weights_II
})

# Convert absolute weights to percentage weights
```

```
total_weight_II = sum(np.abs(optimal_weights_II))
relative_weights_II = (optimal_weights_II / total_weight_II) * 100

weights_df_II['Relative Weight (%)'] = relative_weights_II
weights_df_II.set_index('ETF', inplace=True)
print(weights_df_II)
```

```
     Absolute Weight  Relative Weight (%)
ETF
FXE         0.020586             1.838485
EWJ         0.107261             9.579181
GLD         0.093409             8.342063
QQQ         0.108458             9.686058
SPY         0.024156             2.157260
SHV         0.038106             3.403133
DBA         0.088792             7.929729
USO         0.197514            17.639414
XBI         0.057050             5.094923
ILF         0.173762            15.518201
EPP        -0.059866            -5.346494
FEZ         0.150773            13.465060
```

---

4. The Test (of b A c K)

In [ ]:

```
import pandas as pd
import matplotlib.pyplot as plt

# Define the combined backtesting function
def backtest_combined(strategies, start_date, end_date, rebalance_period='7D', **kw
    results = {}
    dates = pd.date_range(start=start_date, end=end_date, freq=rebalance_period)

    for name, strategy_func in strategies.items():
        backtest_data = etf_returns[(etf_returns.index >= start_date) & (etf_return
        portfolio_values = [100]  # Starting portfolio value

        for i in range(len(dates) - 1):
            current_data = backtest_data.loc[:dates[i]]
            cov_matrix = current_data.cov().values
            # Ensure the covariance matrix is symmetric
            cov_matrix = (cov_matrix + cov_matrix.T) / 2
            if not is_symmetric(cov_matrix):
                raise ValueError("Covariance matrix is not symmetric/Hermitian.")
            expected_returns = current_data.mean().values
            optimal_weights = strategy_func(expected_returns, cov_matrix, **kwargs)

            # Calculate portfolio returns for the next period
            period_returns = (backtest_data.loc[dates[i]:dates[i+1]] @ optimal_weig
            portfolio_values.append(portfolio_values[-1] * (1 + period_returns).pro

        results[name] = (portfolio_values, dates)

    return results
```

```
# Strategy functions
strategy_func_I = lambda exp_returns, cov_mat, **kwargs: optimize_strategy_I(
    exp_returns, cov_mat, factor_loadings_df, beta_constraints=[-0.5, 0.5], lambd=0
)

strategy_func_II = lambda exp_returns, cov_mat, **kwargs: optimize_strategy_II(
    exp_returns, etf_returns.loc[:dates[0]].values, factor_loadings_df, beta_constr
    lambd=0.1, benchmark_returns=etf_returns['SPY'].loc[:dates[0]].values
)

strategies = {
    'Strategy I': strategy_func_I,
    'Strategy II': strategy_func_II
}

# Perform backtesting
results = backtest_combined(strategies, start_date='2007-03-01', end_date='2024-03-

# Plot results
plt.figure(figsize=(12, 6))
for name, (portfolio_values, dates) in results.items():
    plt.plot(dates, portfolio_values, label=name)
plt.xlabel('Date')
plt.ylabel('Portfolio Value')
plt.title('Backtesting Results for Strategy I and Strategy II')
plt.legend()
plt.show()
```

```
c:\Python312\Lib\site-packages\numpy\lib\function_base.py:520: RuntimeWarning: Mean
of empty slice.
  avg = a.mean(axis, **keepdims_kw)
c:\Python312\Lib\site-packages\numpy\core\_methods.py:121: RuntimeWarning: invalid v
alue encountered in divide
  ret = um.true_divide(
c:\Python312\Lib\site-packages\pandas\core\frame.py:11211: RuntimeWarning: Degrees o
f freedom <= 0 for slice
  base_cov = np.cov(mat.T, ddof=ddof)
c:\Python312\Lib\site-packages\numpy\lib\function_base.py:2748: RuntimeWarning: divi
de by zero encountered in divide
  c *= np.true_divide(1, fact)
c:\Python312\Lib\site-packages\numpy\lib\function_base.py:2748: RuntimeWarning: inva
lid value encountered in multiply
  c *= np.true_divide(1, fact)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[63], line 47
     41 strategies = {
     42     'Strategy I': strategy_func_I,
     43     'Strategy II': strategy_func_II
     44 }
     46 # Perform backtesting
---> 47 results = backtest_combined(strategies, start_date='2007-03-01', end_date='2
024-03-31')
     49 # Plot results
     50 plt.figure(figsize=(12, 6))

Cell In[63], line 18, in backtest_combined(strategies, start_date, end_date, rebalan
ce_period, **kwargs)
     16 # Ensure the covariance matrix is symmetric
     17 cov_matrix = (cov_matrix + cov_matrix.T) / 2
---> 18 if not is_symmetric(cov_matrix):
     19     raise ValueError("Covariance matrix is not symmetric/Hermitian.")
     20 expected_returns = current_data.mean().values

TypeError: 'bool' object is not callable
```