Code (Vomit) | : P

1. Collection (of Data)

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

# List of ETF tickers
tickers = ['FXE', 'EWJ', 'GLD', 'QQQ', 'SPY', 'SHV', 'DBA', 'USO', 'XBI', 'ILF', 'E

# Download historical data for each ETF
start_date = '2007-03-01'
end_date = '2024-03-31'
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**

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.
```

In [ ]:

```python
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'
```

```python
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 -0.000657  0.001652
FXE -0.000074  0.001090  0.000042  0.000097
```

3. Optim(ization)

3.05 m i s c

In [ ]:
```python
import numpy as np

def is_symmetric(matrix, tol=1e-8):
    """Check if a matrix is symmetric/Hermitian within a tolerance."""
    return np.allclose(matrix, matrix.T, atol=tol)
```

3.1 Strat $\mathcal{I}$

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

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
    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

# 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: [ 1.57452502 -1.99997055 -1.99999286 -1.99999298 -1.
99999269  1.9999992
 -0.57456848  1.99999897  1.99999577  1.99999792 -1.99999753  1.99999822]
```

In [ ]:

```
import cvxpy as cp
import numpy as np

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

    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

# 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: [ 1.57452502 -1.99997055 -1.99999286 -1.99999298 -1.
99999269  1.9999992
 -0.57456848  1.99999897  1.99999577  1.99999792 -1.99999753  1.99999822]
```

3.2 Strat $\mathcal{II}$

In [ ]:
```
from scipy.optimize import minimize

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

# 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: [ 0.02058621  0.1072614   0.09340881  0.10845794
0.02415552  0.03810605
 0.08879167  0.19751428  0.05704948  0.17376242 -0.05986655  0.15077276]
```

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()
```