

QAM project

December 21, 2020

1 Import packages

```
[39]: import pandas as pd
import numpy as np
import datetime as dt
import matplotlib.pyplot as plt
from pandas.tseries.offsets import MonthEnd, YearEnd
import os
os.chdir("/Users/charlesrambo/Desktop/QIII/QAM project")
```

2 Load data

```
[40]: # FF25 portfolio
FF25 = pd.read_csv("FF_25_Portfolios.csv", nrows = 1127)

## Fama-French 3 factor
FF3 = pd.read_csv("FF3.csv")

## Fama-French mom
FFmom = pd.read_csv("FF_Mom.csv")
```

3 Clean data

```
[41]: # Record CRISP unknowns
unknowns = ["-66.0", "-77.0", "-88.0", "-99.0", "-99.99", "-999", "A", "B", "C", "D", "E", "S", "T", "P"]

# Create function to convert CRISP unknowns to np.nan
convert_unknowns = np.vectorize(lambda x: np.nan if x in unknowns else x)

# Convert to decimal
FF25.iloc[:, 1:] = FF25.iloc[:, 1:].apply(convert_unknowns, axis = 0).
    .astype(float).div(100)
FF3.iloc[:, 1:] = FF3.iloc[:, 1:].apply(convert_unknowns, axis = 0).astype(float).
    .div(100)
```

```

FFmom.iloc[:, 1] = FFmom.iloc[:, 1].apply(convert_unknows).div(100)

# Rename data columns
FF25.rename(columns = {"Unnamed: 0": "date"}, inplace = True)
FF3.rename(columns = {"Unnamed: 0": "date"}, inplace = True)
FFmom.rename(columns = {"Unnamed: 0": "date"}, inplace = True)

# Convert date column
FF25['date'] = pd.to_datetime(FF25['date'], format = "%Y%m")
FF3['date'] = pd.to_datetime(FF3['date'], format = "%Y%m")
FFmom['date'] = pd.to_datetime(FFmom['date'], format = "%Y%m")

# Remove the top 5 ME quintiles
ME = []
for i in range(len(FF25.columns)):
    if FF25.columns[i][:3] != 'ME5':
        ME.append(i)

FF25 = FF25.iloc[:, ME]
FF25.drop(['BIG LoBM', 'BIG HiBM'], axis = 1, inplace = True)

```

4 Merge data

```

[42]: col4 = FF3.drop('RF', axis = 1)
      col5 = FF25.merge(FF3[['date', 'Mkt-RF', 'RF']], on = 'date')
      col6 = FF25.merge(FF3, on = 'date').merge(FFmom, on = 'date')

```

5 Excess returns

```

[43]: col5.iloc[:, 1:21] = col5.iloc[:, 1:21].sub(col5['RF'], axis = 0)
      col6.iloc[:, 1:21] = col6.iloc[:, 1:21].sub(col6['RF'], axis = 0)

      col5.drop('RF', axis = 1, inplace = True)
      col6.drop('RF', axis = 1, inplace = True)

```

6 Subset date range

```

[44]: start = pd.to_datetime("1989", format = "%Y%m")
      end = pd.to_datetime("2012", format = "%Y%m")

      col4 = col4.loc[(col4.date >= start) & (col4.date <= end)]
      col5 = col5.loc[(col5.date >= start) & (col5.date <= end)]
      col6 = col6.loc[(col6.date >= start) & (col6.date <= end)]

```

7 Create function

```
[89]: def get_info(data, M):

    # define X
    X = data.loc[:, data.columns != "date"]

    # define N
    N = len(X.columns)

    # Define table
    table = pd.DataFrame(0, columns = ['mean', 'std', 'Sharpe', 'ceq',
    ↪ 'turn_over'], index = ['ew', 'vw', 'mve_in', 'mve_out', 'shrink_mve_out',
    ↪ 'rp'])

    # ==== Equal weighting ====
    table.loc['ew', 'mean'] = X.apply(np.mean, axis = 1).mean()
    table.loc['ew', 'std'] = X.apply(np.mean, axis = 1).std()
    table.loc['ew', 'Sharpe'] = table.loc['ew', 'mean']/table.loc['ew', 'std']
    table.loc['ew', 'ceq'] = table.loc['ew', 'mean'] - 0.5 * table.loc['ew',
    ↪ 'std']**2

    # Get turn_over
    turn_over = np.zeros(len(X))
    for i in range(len(X)):
        total = 1 + X.iloc[i - 1, :].mean()
        if i == 0:
            turn_over[i] = 1
        else:
            turn_over[i] = 1/N * np.absolute(1 - (1 + X.iloc[i - 1, :])/total).
    ↪ sum()

    table.loc['ew', 'turn_over'] = turn_over.mean()

    # === Value weighting ===
    table.loc['vw', 'mean'] = X['Mkt-RF'].mean()
    table.loc['vw', 'std'] = X['Mkt-RF'].std()
    table.loc['vw', 'Sharpe'] = table.loc['vw', 'mean']/table.loc['vw', 'std']
    table.loc['vw', 'turn_over'] = 0
    table.loc['vw', 'ceq'] = table.loc['vw', 'mean'] - 0.5 * table.loc['vw',
    ↪ 'std']**2

    # === MVE in-sample ===
    Sigma = X.cov()
    mu = X.mean()
    w = np.linalg.pinv(Sigma) @ mu/(np.ones(N).T @ np.linalg.pinv(Sigma) @ mu)
```

```

if w @ mu > 0:
    table.loc['mve_in', 'mean'] = w @ mu
    table.loc['mve_in', 'std'] = np.sqrt(w.T @ Sigma @ w)
    table.loc['mve_in', 'Sharpe'] = w @ mu/np.sqrt(w @ Sigma @ w.T)
else:
    table.loc['mve_in', 'mean'] = -w @ mu
    table.loc['mve_in', 'std'] = np.sqrt(w.T @ Sigma @ w)
    table.loc['mve_in', 'Sharpe'] = -w @ mu/np.sqrt(w @ Sigma @ w.T)

table.loc['mve_in', 'turn_over'] = np.nan
table.loc['mve_in', 'ceq'] = table.loc['mve_in', 'mean'] - 0.5 * table.
→loc['mve_in', 'std']**2

# === MVE out-sample ===
outsample = np.zeros(len(X) - M)
turn_over = np.zeros(len(X) - M)
w = np.zeros(N)

for i in range(M, len(X)):
    if i == M:
        total = 1
    else:
        total = 1 + outsample[i - M - 1]
    Sigma = X.iloc[(i - M):i, :].cov()
    mu = X.iloc[(i - M):i, :].mean()
    w_new = np.linalg.pinv(Sigma) @ mu/(np.ones(N).T @ np.linalg.
→pinv(Sigma) @ mu)
    turn_over[i - M] = np.sum(np.absolute(w_new - w @ (1 + X.iloc[i - 1, :
→]))/total))
    w = w_new
    if w @ mu > 0:
        outsample[i - M] = X.iloc[i, :] @ w
    else:
        outsample[i - M] = -X.iloc[i, :] @ w

table.loc['mve_out', 'mean'] = np.mean(outsample)
table.loc['mve_out', 'std'] = np.std(outsample)
table.loc['mve_out', 'Sharpe'] = np.mean(outsample)/np.std(outsample)
table.loc['mve_out', 'turn_over'] = turn_over.mean()
table.loc['mve_out', 'ceq'] = table.loc['mve_out', 'mean'] - 0.5 * table.
→loc['mve_out', 'std']**2

# === Shrink MVE out-of-sample ===
outsample = np.zeros(len(X) - M)
turn_over = np.zeros(len(X) - M)
w = np.zeros(N)

```

```

for i in range(M, len(X)):
    if i == M:
        total = 1
    else:
        total = 1 + outsample[i - M - 1]

    # Record sample covariance matrix
    S = X.iloc[(i - M):i, :].cov()

    # Define target matrix
    target = np.mean(np.diag(S)) * np.eye(N)

    # Define function to help compute omega2
    f = lambda row: ((X.iloc[row, :] @ X.iloc[row, :].T - S)**2).sum()

    # Compute non-idiosyncratic variance of variance
    omega2 = np.nanmean([f(x) for x in range(M + 1)])/(M - 1)

    # Calculate total variation of variance
    total_var = ((S - target)**2).sum().sum()

    # Calculate idiosyncratic variance of variance
    delta2 = total_var - omega2

    # Compute shrinkage parameter
    beta = np.max([delta2/total_var, 0])

    # Get Sigma_hat
    Sigma = (1 - beta) * target + beta * S

    # Record sample mean
    m = X.iloc[(i - M):i, :].mean()

    # Record target
    target = np.mean(m)

    # Calculate variance of mean estimate
    omega2 = (X.iloc[(i - M):i, :].std()/np.sqrt(M)).mean()

    # Calculate total variance
    total_var = (X.iloc[(i - M):i, :].sub(target, axis = 0)**2).sum().sum()/
→ (M * len(X.columns))

    # Calculate idiosyncratic variance
    delta2 = total_var - omega2

```

```

# Compute shrinkage parameter
beta = np.max([delta2/total_var, 0])

# Compute mu estimate
mu = (1 - beta) * target + beta * m

w_new = np.linalg.pinv(Sigma) @ mu/(np.ones(N).T @ np.linalg.
→pinv(Sigma) @ mu)
turn_over[i - M] = np.sum(np.absolute(w_new - w @ (1 + X.iloc[i - 1, :
→]))/total))

w = w_new
if w @ mu > 0:
    outsample[i - M] = X.iloc[i, :] @ w
else:
    outsample[i - M] = -X.iloc[i, :] @ w

table.loc['shrink_mve_out', 'mean'] = np.mean(outsample)
table.loc['shrink_mve_out', 'std'] = np.std(outsample)
table.loc['shrink_mve_out', 'Sharpe'] = np.mean(outsample)/np.std(outsample)
table.loc['shrink_mve_out', 'turn_over'] = turn_over.mean()
table.loc['shrink_mve_out', 'ceq'] = table.loc['shrink_mve_out', 'mean'] -
→0.5 * table.loc['shrink_mve_out', 'std']**2

# === RP ===
returns = np.zeros(len(X) - M)
turn_over = np.zeros(len(X) - M)
w = np.zeros(N)

for i in range(M, len(X)):
    if i == M:
        total = 1
    else:
        total = 1 + returns[i - M - 1]
    sigma = np.sqrt(np.diag(X.iloc[(i - M):i, :].cov()))
    w_new = (1/sigma)/np.sum(1/sigma)
    returns[i - M] = w_new.T @ X.iloc[i, :]
    turn_over[i - M] = np.sum(np.absolute(w_new - w @ (1 + X.iloc[i - 1, :
→]))/total))
    w = w_new

table.loc['rp', 'mean'] = np.mean(returns)
table.loc['rp', 'std'] = np.std(returns)
table.loc['rp', 'Sharpe'] = np.mean(returns)/np.std(returns)
table.loc['rp', 'turn_over'] = turn_over.mean()
table.loc['rp', 'ceq'] = table.loc['rp', 'mean'] - 0.5 * table.loc['rp',
→'std']**2

```

```
# === convert non-equal weighted turn-over to relative turn-over ===
# table.iloc[1:, 4] = table.iloc[1:, 4]/table.iloc[0, 4]

return(table)
```

8 Solutions

```
[90]: result_4 = get_info(col4, 60)
      result_5 = get_info(col5, 60)
      result_6 = get_info(col6, 60)
```

```
[91]: result_4
```

```
[91]:
```

| | mean | std | Sharpe | ceq | turn_over |
|----------------|----------|----------|----------|-----------|-----------|
| ew | 0.002909 | 0.019409 | 0.149891 | 0.002721 | 0.023718 |
| vw | 0.006558 | 0.042175 | 0.155488 | 0.005668 | 0.000000 |
| mve_in | 0.004402 | 0.025830 | 0.170433 | 0.004069 | NaN |
| mve_out | 0.003321 | 0.210896 | 0.015748 | -0.018917 | 6.078604 |
| shrink_mve_out | 0.001763 | 0.018273 | 0.096504 | 0.001596 | 1.997636 |
| rp | 0.002368 | 0.018359 | 0.128997 | 0.002200 | 1.996711 |

```
[92]: result_5
```

```
[92]:
```

| | mean | std | Sharpe | ceq | turn_over |
|----------------|----------|----------|----------|----------|-----------|
| ew | 0.007769 | 0.050617 | 0.153490 | 0.006488 | 0.018814 |
| vw | 0.006558 | 0.042175 | 0.155488 | 0.005668 | 0.000000 |
| mve_in | 0.025654 | 0.049555 | 0.517681 | 0.024426 | NaN |
| mve_out | 0.023020 | 0.161758 | 0.142310 | 0.009937 | 32.830331 |
| shrink_mve_out | 0.007333 | 0.047239 | 0.155231 | 0.006217 | 19.986497 |
| rp | 0.008547 | 0.050516 | 0.169190 | 0.007271 | 19.937500 |

```
[93]: result_6
```

```
[93]:
```

| | mean | std | Sharpe | ceq | turn_over |
|----------------|----------|----------|----------|-----------|-----------|
| ew | 0.007105 | 0.044485 | 0.159727 | 0.006116 | 0.022231 |
| vw | 0.006558 | 0.042175 | 0.155488 | 0.005668 | 0.000000 |
| mve_in | 0.006833 | 0.012363 | 0.552708 | 0.006757 | NaN |
| mve_out | 0.105233 | 1.733706 | 0.060698 | -1.397635 | 46.131542 |
| shrink_mve_out | 0.005015 | 0.034197 | 0.146642 | 0.004430 | 22.922215 |
| rp | 0.007191 | 0.041462 | 0.173446 | 0.006332 | 22.927632 |