Problem4

April 23, 2023

1 Problem 4

[ ]:
```
import pandas as pd
import numpy as np
import pandas_datareader.data as pdr
import datetime
import statsmodels.api as sm
from tqdm import tqdm
from tqdm.contrib.concurrent import process_map
from tqdm.contrib import tmap

tqdm.pandas()
```

[ ]:
```
crsp_data = pd.read_csv("data/cleaned_crsp.csv")
crsp_data['date'] = pd.to_datetime(crsp_data['date'])
crsp_data['RET'] = crsp_data['RET'].str.replace('C', '')
crsp_data['RET'] = pd.to_numeric(crsp_data['RET'], errors='coerce')
```

[ ]:
```
# Calculate market value of equity (ME) for each stock
crsp_data['mkt_cap'] = np.abs(crsp_data['PRC']) * crsp_data['SHROUT']
```

[ ]:
```
def assign_deciles(data):
    # Check if there are any non-NaN values in the 'cum_ret' column
    if pd.notna(data['rolling_beta']).any():
        data['decile'] = pd.qcut(data['rolling_beta'], 10, labels=False) + 1
    else:
        # Set decile to NaN if there are no valid values in 'cum_ret'
        data['decile'] = np.nan
    return data

# get equal- and value-weighted portfolios
def calculate_portfolio_returns(data):
    ew_ret = data['RET'].mean()
    vw_ret = np.average(data['RET'], weights=data['rolling_beta'] + 1e-6)
    return pd.Series({'ew_ret': ew_ret, 'vw_ret': vw_ret})
```

1.1 (A)

[ ]:
```
import pandas_datareader as pdr

def estimate_beta(stock_returns, market_returns):
    if len(stock_returns) == 0 or len(market_returns) == 0:
        return np.nan
    else:
        return np.cov(stock_returns, market_returns)[0, 1] / np.
 ↪var(market_returns)

def calculate_rolling_beta(group):
    rolling_beta = group['excess_ret'].rolling(window=36).apply(
        lambda x: estimate_beta(x, ff5_factors.loc[ff5_factors['date'].
 ↪isin(group.loc[x.index, 'date']), 'Mkt-RF']),
        raw=False
    )
    return rolling_beta


start_date = '1926-01-01'
end_date = '2020-12-31'

# Download Fama-French 3-factor data
ff3_factors = pdr.get_data_famafrench('F-F_Research_Data_Factors',␣
 ↪start=start_date, end=end_date)[0]
ff3_factors = ff3_factors / 100  # Convert to decimal
ff3_factors.index = ff3_factors.index.to_timestamp('M')  # Convert index to␣
 ↪monthly-end dates

# FF5 - FIX DATA SOURCE
ff5_factors = pdr.get_data_famafrench('F-F_Research_Data_5_Factors_2x3',␣
 ↪start=start_date, end=end_date)[0]
ff5_factors = ff5_factors / 100  # Convert to decimal
ff5_factors.index = ff5_factors.index.to_timestamp('M')  # Convert index to␣
 ↪monthly-end dates

ff5_factors['date'] = ff5_factors.index

# Calculate market value of equity (ME) for each stock
crsp_data['mkt_cap'] = np.abs(crsp_data['PRC']) * crsp_data['SHROUT']

# Calculate excess returns
crsp_data = crsp_data.merge(ff5_factors[['date', 'RF']], on='date')
crsp_data['excess_ret'] = crsp_data['RET'] - crsp_data['RF']

# Calculate rolling betas for each stock
```

```
crsp_data['rolling_beta'] = crsp_data.groupby('PERMNO').
↪progress_apply(calculate_rolling_beta).reset_index(level=0, drop=True)

# Assign deciles based on rolling betas
crsp_data = crsp_data.groupby('date').progress_apply(assign_deciles).
↪reset_index(drop=True)

# Calculate equal- and value-weighted portfolio returns for each decile
portfolio_returns = crsp_data.groupby(['date', 'decile']).
↪apply(calculate_portfolio_returns).reset_index()

# Pivot the data to get a wide format with deciles as columns
ew_returns = portfolio_returns.pivot_table(values='ew_ret', index='date',␣
↪columns='decile')
vw_returns = portfolio_returns.pivot_table(values='vw_ret', index='date',␣
↪columns='decile')
```

```
100%|          | 24861/24861 [06:53<00:00, 60.16it/s]
100%|          | 482/482 [00:01<00:00, 473.82it/s]
```

1.2 (B)

[ ]:
```
# Calculate equal- and value-weighted portfolio returns
portfolio_returns = crsp_data.groupby(['date', 'decile']).
↪apply(calculate_portfolio_returns).reset_index()

# Pivot the data to get a wide format with deciles as columns
ew_returns = portfolio_returns.pivot_table(values='ew_ret', index='date',␣
↪columns='decile')
vw_returns = portfolio_returns.pivot_table(values='vw_ret', index='date',␣
↪columns='decile')

# Calculate mean returns for each decile
mean_ew_returns = ew_returns.mean()
mean_vw_returns = vw_returns.mean()

# Check if the returns are monotonic
is_monotonic_ew = mean_ew_returns.is_monotonic_decreasing
is_monotonic_vw = mean_vw_returns.is_monotonic_decreasing

print("Mean equal-weighted returns:")
print(mean_ew_returns)
print("Is monotonic:", is_monotonic_ew)
print("\nMean value-weighted returns:")
print(mean_vw_returns)
print("Is monotonic:", is_monotonic_vw)
```

```
Mean equal-weighted returns:
decile
1.0     0.013770
2.0     0.010966
3.0     0.011456
4.0     0.011264
5.0     0.011357
6.0     0.011658
7.0     0.012165
8.0     0.013267
9.0     0.014245
10.0    0.024120
dtype: float64
Is monotonic: False

Mean value-weighted returns:
decile
1.0     0.060841
2.0     0.011074
3.0     0.011406
4.0     0.011181
5.0     0.012375
6.0     0.011747
7.0     0.012140
8.0     0.013080
9.0     0.014326
10.0    0.028537
dtype: float64
Is monotonic: False
```

1.3 (C)

[ ]:
```python
ew_bab = ew_returns[1] - ew_returns[10]
vw_bab = vw_returns[1] - vw_returns[10]

# Calculate mean returns
mean_ew_bab = ew_bab.mean()
mean_vw_bab = vw_bab.mean()

# Calculate volatility
vol_ew_bab = ew_bab.std()
vol_vw_bab = vw_bab.std()

# Calculate Sharpe ratio (assuming a risk-free rate of 0)
sharpe_ew_bab = mean_ew_bab / vol_ew_bab
sharpe_vw_bab = mean_vw_bab / vol_vw_bab
```

```
print("Equal-weighted BAB portfolio:")
print(f"Mean: {mean_ew_bab:.6f}")
print(f"Volatility: {vol_ew_bab:.6f}")
print(f"Sharpe Ratio: {sharpe_ew_bab:.6f}")

print("Value-weighted BAB portfolio:")
print(f"Mean: {mean_vw_bab:.6f}")
print(f"Volatility: {vol_vw_bab:.6f}")
print(f"Sharpe Ratio: {sharpe_vw_bab:.6f}")
```

```
Equal-weighted BAB portfolio:
Mean: -0.010350
Volatility: 0.136471
Sharpe Ratio: -0.075839
Value-weighted BAB portfolio:
Mean: 0.032304
Volatility: 4.807499
Sharpe Ratio: 0.006719
```

1.4 (D)

[ ]:
```
# Function to calculate factor models
def calculate_factor_model(data, factors):
    aligned_data, aligned_factors = data.align(factors, join='inner')
    X = sm.add_constant(aligned_factors)
    model = sm.OLS(aligned_data, X).fit()
    return model.params

# Calculate the momentum factor
momentum_deciles = crsp_data.groupby(['date', 'decile']).
 ↪apply(calculate_portfolio_returns).reset_index()
momentum_returns = momentum_deciles.pivot_table(values='ew_ret', index='date',␣
 ↪columns='decile')
momentum_factor = momentum_returns[10] - momentum_returns[1]

# Merge the momentum factor with the FF5 factors
ff5_factors_plus_mom = ff5_factors.merge(pd.DataFrame(momentum_factor,␣
 ↪columns=['Mom']), left_index=True, right_index=True)

# Estimate the CAPM model for both equal- and value-weighted portfolios
capm_ew = calculate_factor_model(ew_bab, ff5_factors[['Mkt-RF']])
capm_vw = calculate_factor_model(vw_bab, ff5_factors[['Mkt-RF']])

# Estimate the FF3 model for both equal- and value-weighted portfolios
ff3_ew = calculate_factor_model(ew_bab, ff5_factors[['Mkt-RF', 'SMB', 'HML']])
ff3_vw = calculate_factor_model(vw_bab, ff5_factors[['Mkt-RF', 'SMB', 'HML']])
```

```python
# Estimate the FF5 model for both equal- and value-weighted portfolios
ff5_ew = calculate_factor_model(ew_bab, ff5_factors[['Mkt-RF', 'SMB', 'HML',␣
↪'RMW', 'CMA']])
ff5_vw = calculate_factor_model(vw_bab, ff5_factors[['Mkt-RF', 'SMB', 'HML',␣
↪'RMW', 'CMA']])

# Estimate the FF5+Momentum models for both equal- and value-weighted portfolios
ff5_mom_ew = calculate_factor_model(ew_bab, ff5_factors_plus_mom[['Mkt-RF',␣
↪'SMB', 'HML', 'RMW', 'CMA', 'Mom']])
ff5_mom_vw = calculate_factor_model(vw_bab, ff5_factors_plus_mom[['Mkt-RF',␣
↪'SMB', 'HML', 'RMW', 'CMA', 'Mom']])

print("Equal-weighted BAB portfolio:")
print("CAPM:", capm_ew)
print("FF3:", ff3_ew)
print("FF5:", ff5_ew)
print("FF5+Momentum:", ff5_mom_ew)

print("\nValue-weighted BAB portfolio:")
print("CAPM:", capm_vw)
print("FF3:", ff3_vw)
print("FF5:", ff5_vw)
print("FF5+Momentum:", ff5_mom_vw)
```

```
Equal-weighted BAB portfolio:
CAPM: const     0.004061
Mkt-RF   -2.477044
dtype: float64
FF3: const     0.003785
Mkt-RF   -2.281825
SMB      -0.625355
HML       0.362698
dtype: float64
FF5: const    -0.000166
Mkt-RF   -2.120048
SMB      -0.407309
HML      -0.176170
RMW       0.968409
CMA       1.005079
dtype: float64
FF5+Momentum: const    -2.864462e-16
Mkt-RF   -1.387779e-16
SMB       5.551115e-17
HML       1.110223e-16
RMW       2.775558e-17
CMA       1.387779e-15
Mom      -1.000000e+00
```

```
dtype: float64

Value-weighted BAB portfolio:
CAPM: const      0.120236
Mkt-RF   -15.114590
dtype: float64
FF3: const      0.164531
Mkt-RF   -16.590780
SMB       -1.006878
HML      -12.741982
dtype: float64
FF5: const      0.111092
Mkt-RF   -13.813737
SMB        1.405013
HML      -22.390868
RMW       10.782304
CMA       19.809657
dtype: float64
FF5+Momentum: const      0.111915
Mkt-RF    -3.288307
SMB        3.427185
HML      -21.516234
RMW        5.974432
CMA       14.819731
Mom       -4.964712
dtype: float64
```

1.5 (E)

To reduce the volatility of the BAB strategy, you can consider the following approaches:

Diversification: Increase the number of stocks in the long and short portfolios to diversify the idiosyncratic risk of individual stocks. This should result in a lower overall portfolio volatility. Time-varying risk: Consider incorporating a dynamic risk management strategy that adjusts portfolio exposure based on the prevailing market volatility. For example, you can reduce the portfolio's exposure during periods of high market volatility and increase