Problem3

April 23, 2023

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

# Enable tqdm for Pandas
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')
crsp_data['date'] = pd.to_datetime(crsp_data['date'], format='%Y-%m-%d')
```

1 A

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

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


crsp_data['cum_ret'] = crsp_data.groupby('PERMNO')['RET'].rolling(window=11).
 ↪progress_apply(lambda x: np.prod(1 + x) - 1, raw=True).reset_index(0,␣
 ↪drop=True)




# # Define a function to assign deciles based on market cap
def assign_deciles(data):
    # Check if there are any non-NaN values in the 'cum_ret' column
    if pd.notna(data['cum_ret']).any():
        data['decile'] = pd.qcut(data['cum_ret'], 10, labels=False) + 1
```

```
    else:
        # Set decile to NaN if there are no valid values in 'cum_ret'
        data['decile'] = np.nan
    return data

crsp_data = crsp_data.groupby('date').progress_apply(assign_deciles).
 ↪reset_index(drop=True)

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

# Group the data by date and decile and calculate the returns for each group
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')
```

```
3279165it [00:15, 213405.01it/s]
100%|      | 1141/1141 [00:02<00:00, 557.39it/s]
```

2 B

[ ]:
```
# 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    -0.053691
```

```
2.0    -0.018407
3.0    -0.006398
4.0     0.001725
5.0     0.008325
6.0     0.015597
7.0     0.022531
8.0     0.031198
9.0     0.045244
10.0    0.085250
dtype: float64
Is monotonic: False

Mean value-weighted returns:
decile
1.0    -0.064252
2.0    -0.019621
3.0    -0.008597
4.0     0.002204
5.0     0.008493
6.0     0.017407
7.0     0.023234
8.0     0.030565
9.0     0.046216
10.0    0.105113
dtype: float64
Is monotonic: False
```

3 C

[ ]:
```python
def form_wml_portfolios(group):
    winners = group[group['decile'] == 10.0]
    losers = group[group['decile'] == 1.0]

    # Calculate equal-weighted average returns for winners and losers
    winners_ret_ew = winners['RET'].mean()
    losers_ret_ew = losers['RET'].mean()

    vw_winners_ret = np.average(winners['RET'], weights=winners['cum_ret']) if␣
 ↪winners['cum_ret'].sum() != 0 else np.nan
    vw_losers_ret = np.average(losers['RET'], weights=losers['cum_ret']) if␣
 ↪losers['cum_ret'].sum() != 0 else np.nan


    # Calculate winners-minus-losers return
    wml_ret_ew = winners_ret_ew - losers_ret_ew
    wml_ret_vw = vw_winners_ret - vw_losers_ret
```

```python
    return pd.Series({
        'ew_wml_ret': wml_ret_ew,
        'vw_wml_ret': wml_ret_vw
    })

wml_returns = crsp_data.groupby('date').apply(form_wml_portfolios)

# Extract equal-weighted and value-weighted WML returns
ew_wml_returns = wml_returns['ew_wml_ret']
vw_wml_returns = wml_returns['vw_wml_ret']

# Print the results
print("Equal-Weighted WML Portfolio Returns:")
print(ew_returns)
print("\nValue-Weighted WML Portfolio Returns:")
print(vw_returns)
```

```
Equal-Weighted WML Portfolio Returns:
decile          1.0       2.0       3.0       4.0       5.0       6.0   \
date
1926-11-30 -0.042136  0.005113 -0.009188  0.025732  0.017115  0.033036
1926-12-31 -0.002997  0.003170  0.016758  0.020720  0.027043  0.048024
1927-01-31 -0.054276 -0.039402  0.008722  0.030068  0.007751 -0.005085
1927-02-28  0.021823  0.022411  0.035212  0.055817  0.041982  0.051045
1927-03-31 -0.156663 -0.041989 -0.040131 -0.039031 -0.026726 -0.007436
…                …         …         …         …         …         …
2020-08-31 -0.028791  0.018453  0.027528  0.033558  0.046831  0.056140
2020-09-30 -0.136618 -0.064249 -0.055185 -0.047335 -0.027944 -0.019465
2020-10-30 -0.084762  0.002844  0.029962  0.024394  0.005931  0.017045
2020-11-30  0.275224  0.234795  0.165377  0.177001  0.150601  0.126346
2020-12-31  0.035540  0.051914  0.048137  0.058761  0.073260  0.078719

decile          7.0       8.0       9.0       10.0
date
1926-11-30  0.024237  0.059915  0.063451  0.089871
1926-12-31  0.028398  0.025735  0.033012  0.056693
1927-01-31  0.001483  0.025250  0.053090  0.104957
1927-02-28  0.051243  0.066560  0.088671  0.145928
1927-03-31  0.012752  0.015762  0.024614  0.082924
…                …         …         …         …
2020-08-31  0.066179  0.058601  0.080436  0.112239
2020-09-30 -0.015903 -0.007329  0.017154  0.092528
2020-10-30  0.008791  0.026992  0.044663  0.044760
2020-11-30  0.153868  0.174741  0.194155  0.407753
2020-12-31  0.076671  0.097152  0.144409  0.267880

[1130 rows x 10 columns]
```

```
Value-Weighted WML Portfolio Returns:
decile           1.0       2.0       3.0       4.0       5.0       6.0   \
date
1926-11-30 -0.047352  0.004998 -0.010349  0.028130  0.014234  0.035474
1926-12-31 -0.007831  0.003726  0.015093  0.020937  0.032208  0.022845
1927-01-31 -0.062235 -0.043152  0.010284  0.040709 -0.015947 -0.005598
1927-02-28  0.018180  0.011788  0.001475  0.056511  0.042634  0.051067
1927-03-31 -0.175633 -0.044984 -0.042529 -0.025489 -0.025075 -0.006684
…                …         …         …         …         …         …
2020-08-31 -0.042148  0.016506  0.027681  0.033121  0.046237  0.100749
2020-09-30 -0.143310 -0.063968 -0.056027 -0.049029 -0.031444 -0.031698
2020-10-30 -0.093789  0.000509  0.029508  0.024132  0.005862  0.013224
2020-11-30  0.279717  0.236362  0.166228  0.180236  0.153877  0.120735
2020-12-31  0.029954  0.054628  0.047325  0.056888  0.077520  0.078772

decile           7.0       8.0       9.0       10.0
date
1926-11-30  0.026027  0.063484  0.060720  0.092394
1926-12-31  0.026153  0.024980  0.034167  0.053468
1927-01-31  0.000528  0.026415  0.055906  0.138347
1927-02-28  0.051648  0.066411  0.088234  0.144009
1927-03-31  0.014451  0.015675  0.025575  0.102088
…                …         …         …         …
2020-08-31  0.069016  0.057873  0.084661  0.142364
2020-09-30 -0.015382 -0.008425  0.016401  0.136236
2020-10-30  0.012274  0.027685  0.046815  0.030145
2020-11-30  0.152788  0.175511  0.202967  0.539364
2020-12-31  0.076759  0.098797  0.152792  0.300522

[1130 rows x 10 columns]
```

[ ]:
```python
# Calculate mean returns
ew_wml_means = ew_wml_returns.mean()
vw_wml_means = vw_wml_returns.mean()

# Calculate volatility
ew_wml_vol = ew_wml_returns.std()
vw_wml_vol = vw_wml_returns.std()

# Calculate Sharpe ratio (assuming a risk-free rate of 0)
ew_wml_sharpe = ew_wml_means / ew_wml_vol
vw_wml_sharpe = vw_wml_means / vw_wml_vol

print("Equal-weighted SMB portfolio:")
print(f"Mean: {ew_wml_means:.6f}")
print(f"Volatility: {ew_wml_vol:.6f}")
```

```
print(f"Sharpe Ratio: {ew_wml_sharpe:.6f}")

print("\nValue-weighted SMB portfolio:")
print(f"Mean: {vw_wml_means:.6f}")
print(f"Volatility: {vw_wml_vol:.6f}")
print(f"Sharpe Ratio: {vw_wml_sharpe:.6f}")
```

```
Equal-weighted SMB portfolio:
Mean: 0.138941
Volatility: 0.091022
Sharpe Ratio: 1.526452

Value-weighted SMB portfolio:
Mean: 0.169365
Volatility: 0.182155
Sharpe Ratio: 0.929784
```

4 D

[ ]:

```
import pandas_datareader as pdr

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
```

[ ]:

```
def estimate_models(returns, factors, factors5):
    # Add a constant to the factors for regression
    factors = sm.add_constant(factors)

    # Estimate the CAPM model
    capm_model = sm.OLS(returns, factors[['const', 'Mkt-RF']]).fit()

    # Estimate the FF3 model
```

```
    ff3_model = sm.OLS(returns, factors).fit()

    # Estimate the FF5 model
    ff5_model = sm.OLS(returns, factors5).fit()

    return capm_model.params, ff3_model.params, ff5_model.params

# Assuming ew_wml_returns and vw_wml_returns are available as the␣
 ↪equal-weighted and value-weighted WML portfolio returns
# Assuming ff3_factors and ff5_factors are available as the Fama-French␣
 ↪3-factor and 5-factor data

# Add a constant column to the returns DataFrames
ew_returns = ew_wml_returns.to_frame(name='WML')
ew_returns['const'] = 1
vw_returns = vw_wml_returns.to_frame(name='WML')
vw_returns['const'] = 1

# Merge the factor data with the portfolio returns
ew_returns = ew_returns.merge(ff3_factors, left_index=True, right_index=True,␣
 ↪suffixes=('', '_y'))
vw_returns = vw_returns.merge(ff3_factors, left_index=True, right_index=True,␣
 ↪suffixes=('', '_y'))

# Merge the FF5 data with the portfolio returns
ew_returns = ew_returns.merge(ff5_factors, left_index=True, right_index=True,␣
 ↪suffixes=('', '_y'))
vw_returns = vw_returns.merge(ff5_factors, left_index=True, right_index=True,␣
 ↪suffixes=('', '_y'))

# Calculate the CAPM, FF3, and FF5 model parameters for both equal-weighted and␣
 ↪value-weighted WML portfolios
ew_capm_params, ew_ff3_params, ew_ff5_params =␣
 ↪estimate_models(ew_returns['WML'], ew_returns[['const', 'Mkt-RF', 'SMB',␣
 ↪'HML']], ew_returns[['const', 'Mkt-RF', 'SMB', 'HML', 'RMW', 'CMA']])
vw_capm_params, vw_ff3_params, vw_ff5_params =␣
 ↪estimate_models(vw_returns['WML'], vw_returns[['const', 'Mkt-RF', 'SMB',␣
 ↪'HML']], vw_returns[['const', 'Mkt-RF', 'SMB', 'HML', 'RMW', 'CMA']])
```

[ ]:
```
# Print the estimated alphas
print("Equal-weighted WML portfolio results:")
print("CAPM Alpha:", ew_capm_params['const'])
print("FF3 Alpha:", ew_ff3_params['const'])
print("FF5 Alpha:", ew_ff5_params['const'])

print("\nValue-weighted WML portfolio results:")
```

```python
print("CAPM Alpha:", vw_capm_params['const'])
print("FF3 Alpha:", vw_ff3_params['const'])
print("FF5 Alpha:", vw_ff5_params['const'])
```

```
Equal-weighted WML portfolio results:
CAPM Alpha: 0.1610638116563032
FF3 Alpha: 0.16309969309562983
FF5 Alpha: 0.1608084666302788

Value-weighted WML portfolio results:
CAPM Alpha: 0.19538097569006252
FF3 Alpha: 0.19766627399845
FF5 Alpha: 0.19593888213371594
```

5 E

The alphas are definitely positive, but this is likely due to the market doing well and the manager getting "paid" for taking on a bunch of risk. The alpha is coming largely from being exposed to risk, not necessarily from managerial skill.