Problem2

April 23, 2023

[ ]:
```
import pandas as pd
import numpy as np
import statsmodels.api as sm
from datetime import datetime
```

[ ]:
```
crsp_data = pd.read_csv("data/cleaned_crsp.csv")
crsp_data['date'] = pd.to_datetime(crsp_data['date'])
crsp_data['RET'] = crsp_data['RET'].replace('C', np.nan)
crsp_data['RET'] = pd.to_numeric(crsp_data['RET'], errors='coerce')
# crsp_data['ret'] = crsp_data['RET'].shift(-1)
```

1 A

[ ]:
```
# MV Calc
crsp_data['mkt_cap'] = np.abs(crsp_data['PRC']) * crsp_data['SHROUT']

# Deciles
def assign_deciles(data):
    data['decile'] = pd.qcut(data['mkt_cap'], 10, labels=False) + 1
    return data

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

# Get returns, maybe weighted
def calculate_portfolio_returns(data):
    ew_ret = data['RET'].mean()
    vw_ret = np.average(data['RET'], weights=data['mkt_cap'])
    return pd.Series({'ew_ret': ew_ret, 'vw_ret': vw_ret})

# Calc returns
portfolio_returns = crsp_data.groupby(['date', 'decile']).
 ↪apply(calculate_portfolio_returns).reset_index()

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')
```

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    -0.004884
2.0     0.011973
3.0     0.013357
4.0     0.015108
5.0     0.017202
6.0     0.017586
7.0     0.017644
8.0     0.017627
9.0     0.016295
10.0    0.014873
dtype: float64
Is monotonic: False

Mean value-weighted returns:
decile
1.0     0.005202
2.0     0.014845
3.0     0.017260
4.0     0.018287
5.0     0.019333
6.0     0.018718
7.0     0.016182
8.0     0.015942
9.0     0.014830
```

```
10.0    0.012607
dtype: float64
Is monotonic: False
```

3 C

[ ]:
```python
ew_smb = ew_returns[1] - ew_returns[10]
vw_smb = vw_returns[1] - vw_returns[10]

# Calculate mean returns
mean_ew_smb = ew_smb.mean()
mean_vw_smb = vw_smb.mean()

# Calculate volatility
vol_ew_smb = ew_smb.std()
vol_vw_smb = vw_smb.std()

# Calculate Sharpe ratio (assuming a risk-free rate of 0)
sharpe_ew_smb = mean_ew_smb / vol_ew_smb
sharpe_vw_smb = mean_vw_smb / vol_vw_smb

print("Equal-weighted SMB portfolio:")
print(f"Mean: {mean_ew_smb:.6f}")
print(f"Volatility: {vol_ew_smb:.6f}")
print(f"Sharpe Ratio: {sharpe_ew_smb:.6f}")

print("\nValue-weighted SMB portfolio:")
print(f"Mean: {mean_vw_smb:.6f}")
print(f"Volatility: {vol_vw_smb:.6f}")
print(f"Sharpe Ratio: {sharpe_vw_smb:.6f}")
```

```
Equal-weighted SMB portfolio:
Mean: -0.019758
Volatility: 0.088860
Sharpe Ratio: -0.222348

Value-weighted SMB portfolio:
Mean: -0.004710
Volatility: 0.097155
Sharpe Ratio: -0.048481
```

4 D

[ ]:
```python
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
```

[ ]:
```
def calculate_vw_returns(data):
    data['mkt_cap'] = data['PRC'] * data['SHROUT']
    data['wgt_ret'] = data['RET'] * data['mkt_cap']
    total_mkt_cap = data['mkt_cap'].sum()
    vw_ret = data['wgt_ret'].sum() / total_mkt_cap
    return vw_ret

vw_returns = crsp_data.groupby(['date', 'decile']).apply(calculate_vw_returns).
 ↪reset_index()
vw_returns = vw_returns.pivot_table(values=0, index='date', columns='decile')
```

[ ]:
```
def estimate_models(returns, factors):
    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()

    return capm_model.params, ff3_model.params

# Merge the factor data with the portfolio returns
# Add a constant column to the returns DataFrames
ew_returns['const'] = 1
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'))


# Calculate the CAPM and FF3 model parameters for each decile
ew_results = pd.DataFrame()
vw_results = pd.DataFrame()
```

```
for decile in range(1, 11):
    ew_capm_params, ew_ff3_params = estimate_models(ew_returns[decile],␣
 ↪ew_returns[['const', 'Mkt-RF', 'SMB', 'HML']])
    vw_capm_params, vw_ff3_params = estimate_models(vw_returns[decile],␣
 ↪vw_returns[['const', 'Mkt-RF', 'SMB', 'HML']])

    ew_results = pd.concat([ew_results, pd.concat([ew_capm_params,␣
 ↪ew_ff3_params], keys=['CAPM', 'FF3'])], axis=1)
    vw_results = pd.concat([vw_results, pd.concat([vw_capm_params,␣
 ↪vw_ff3_params], keys=['CAPM', 'FF3'])], axis=1)

ew_results.columns = range(1, 11)
vw_results.columns = range(1, 11)


print("Equal-weighted portfolio results:")
print(ew_results)

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

```
Equal-weighted portfolio results:
                   1         2         3         4         5         6   \
CAPM const  -0.015925  0.001009  0.003584  0.005511  0.007938  0.008830
     Mkt-RF  1.656576  1.599239  1.456843  1.432393  1.399146  1.340762
FF3  const  -0.019065 -0.001587  0.001684  0.003979  0.006661  0.007767
     Mkt-RF  1.134937  1.152042  1.108285  1.135423  1.140081  1.122068
     SMB     1.621889  1.440500  1.187582  1.058033  0.954193  0.813348
     HML     0.967517  0.750562  0.482891  0.338609  0.246224  0.195488

                   7         8         9         10
CAPM const   0.009166  0.009423  0.008992  0.008153
     Mkt-RF  1.276111  1.212673  1.138387  0.989791
FF3  const   0.008404  0.008890  0.008614  0.008164
     Mkt-RF  1.112478  1.091643  1.063974  0.993081
     SMB     0.626072  0.478973  0.267695 -0.014604
     HML     0.118697  0.062730  0.080782  0.000789

Value-weighted portfolio results:
                   1         2         3         4         5         6   \
CAPM const  -0.009810  0.001199  0.003483  0.005589  0.007891  0.008733
     Mkt-RF  1.628531  1.589552  1.447115  1.425369  1.392176  1.327447
FF3  const  -0.012750 -0.001376  0.001603  0.004058  0.006621  0.007703
     Mkt-RF  1.131423  1.146148  1.102706  1.129894  1.135195  1.114320
     SMB     1.573197  1.427876  1.172379  1.049061  0.944606  0.796081
     HML     0.878563  0.744832  0.478828  0.342646  0.247255  0.185094
```

```
                    7         8         9         10
CAPM const   0.009095  0.009279  0.008734  0.008034
     Mkt-RF  1.267836  1.203432  1.125723  0.934020
FF3  const   0.008348  0.008767  0.008370  0.008203
     Mkt-RF  1.107443  1.087439  1.055901  0.968997
     SMB     0.614235  0.459134  0.246864 -0.130755
     HML     0.115457  0.059971  0.082598 -0.030205
```

[ ]:
```
ew_results
```

[ ]:
```
                    1         2         3         4         5         6   \
CAPM const  -0.015925  0.001009  0.003584  0.005511  0.007938  0.008830
     Mkt-RF  1.656576  1.599239  1.456843  1.432393  1.399146  1.340762
FF3  const  -0.019065 -0.001587  0.001684  0.003979  0.006661  0.007767
     Mkt-RF  1.134937  1.152042  1.108285  1.135423  1.140081  1.122068
     SMB     1.621889  1.440500  1.187582  1.058033  0.954193  0.813348
     HML     0.967517  0.750562  0.482891  0.338609  0.246224  0.195488

                    7         8         9         10
CAPM const   0.009166  0.009423  0.008992  0.008153
     Mkt-RF  1.276111  1.212673  1.138387  0.989791
FF3  const   0.008404  0.008890  0.008614  0.008164
     Mkt-RF  1.112478  1.091643  1.063974  0.993081
     SMB     0.626072  0.478973  0.267695 -0.014604
     HML     0.118697  0.062730  0.080782  0.000789
```

[ ]:
```
vw_results
```

[ ]:
```
                    1         2         3         4         5         6   \
CAPM const  -0.009810  0.001199  0.003483  0.005589  0.007891  0.008733
     Mkt-RF  1.628531  1.589552  1.447115  1.425369  1.392176  1.327447
FF3  const  -0.012750 -0.001376  0.001603  0.004058  0.006621  0.007703
     Mkt-RF  1.131423  1.146148  1.102706  1.129894  1.135195  1.114320
     SMB     1.573197  1.427876  1.172379  1.049061  0.944606  0.796081
     HML     0.878563  0.744832  0.478828  0.342646  0.247255  0.185094

                    7         8         9         10
CAPM const   0.009095  0.009279  0.008734  0.008034
     Mkt-RF  1.267836  1.203432  1.125723  0.934020
FF3  const   0.008348  0.008767  0.008370  0.008203
     Mkt-RF  1.107443  1.087439  1.055901  0.968997
     SMB     0.614235  0.459134  0.246864 -0.130755
     HML     0.115457  0.059971  0.082598 -0.030205
```

5 E

[ ]:
```
# Set the date ranges
post_ff_paper_start = '1993-01-01'
post_ff_paper_end = '2001-12-31'
post_dotcom_start = '2002-01-01'

# Create the subsets
ew_returns_post_ff = ew_returns.loc[(ew_returns.index >= post_ff_paper_start) &␣
 ↪(ew_returns.index <= post_ff_paper_end)]
vw_returns_post_ff = vw_returns.loc[(vw_returns.index >= post_ff_paper_start) &␣
 ↪(vw_returns.index <= post_ff_paper_end)]

ew_returns_post_dotcom = ew_returns.loc[ew_returns.index >= post_dotcom_start]
vw_returns_post_dotcom = vw_returns.loc[vw_returns.index >= post_dotcom_start]
```

[ ]:
```
def calculate_statistics(returns):
    mean = returns.mean()
    volatility = returns.std()
    sharpe_ratio = mean / volatility
    return mean, volatility, sharpe_ratio

# Post Fama French 1992 paper
ew_mean_post_ff, ew_vol_post_ff, ew_sharpe_post_ff =␣
 ↪calculate_statistics(ew_returns_post_ff.iloc[:, -1] - ew_returns_post_ff.
 ↪iloc[:, 0])
vw_mean_post_ff, vw_vol_post_ff, vw_sharpe_post_ff =␣
 ↪calculate_statistics(vw_returns_post_ff.iloc[:, -1] - vw_returns_post_ff.
 ↪iloc[:, 0])

# Post Dot-Com Bubble
ew_mean_post_dotcom, ew_vol_post_dotcom, ew_sharpe_post_dotcom =␣
 ↪calculate_statistics(ew_returns_post_dotcom.iloc[:, -1] -␣
 ↪ew_returns_post_dotcom.iloc[:, 0])
vw_mean_post_dotcom, vw_vol_post_dotcom, vw_sharpe_post_dotcom =␣
 ↪calculate_statistics(vw_returns_post_dotcom.iloc[:, -1] -␣
 ↪vw_returns_post_dotcom.iloc[:, 0])
```

[ ]:
```
print("Post Fama French 1992 paper:")
print(f"Equal-weighted SMB portfolio - Mean: {ew_mean_post_ff}, Volatility:␣
 ↪{ew_vol_post_ff}, Sharpe Ratio: {ew_sharpe_post_ff}")
print(f"Value-weighted SMB portfolio - Mean: {vw_mean_post_ff}, Volatility:␣
 ↪{vw_vol_post_ff}, Sharpe Ratio: {vw_sharpe_post_ff}")

print("\nPost Dot-Com Bubble:")
print(f"Equal-weighted SMB portfolio - Mean: {ew_mean_post_dotcom}, Volatility:␣
 ↪{ew_vol_post_dotcom}, Sharpe Ratio: {ew_sharpe_post_dotcom}")
```

```
print(f"Value-weighted SMB portfolio - Mean: {vw_mean_post_dotcom}, Volatility:␣
↪{vw_vol_post_dotcom}, Sharpe Ratio: {vw_sharpe_post_dotcom}")
```

```
Post Fama French 1992 paper:
Equal-weighted SMB portfolio - Mean: 0.02872378902953428, Volatility:
0.09651448361997189, Sharpe Ratio: 0.2976111766046938
Value-weighted SMB portfolio - Mean: 0.01574140322997766, Volatility:
0.08941032442062882, Sharpe Ratio: 0.17605800372586267

Post Dot-Com Bubble:
Equal-weighted SMB portfolio - Mean: 0.0228619622648221, Volatility:
0.08119419005368132, Sharpe Ratio: 0.28157140615242265
Value-weighted SMB portfolio - Mean: 0.012345695771850814, Volatility:
0.07930922612602084, Sharpe Ratio: 0.15566531631810074
```

Some what still works after this.