Problem1

April 23, 2023

[ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

[ ]:
```
crsp = pd.read_csv("data/crsp_1926_2020.csv")
crsp = crsp[(crsp['SHRCD'] == 10) | (crsp['SHRCD']==11)]
crsp = crsp[(crsp['EXCHCD']==1) | (crsp['EXCHCD']==2) | (crsp['EXCHCD']==3)]
crsp.loc[crsp['PRC']<0, 'PRC'] = pd.NA
```

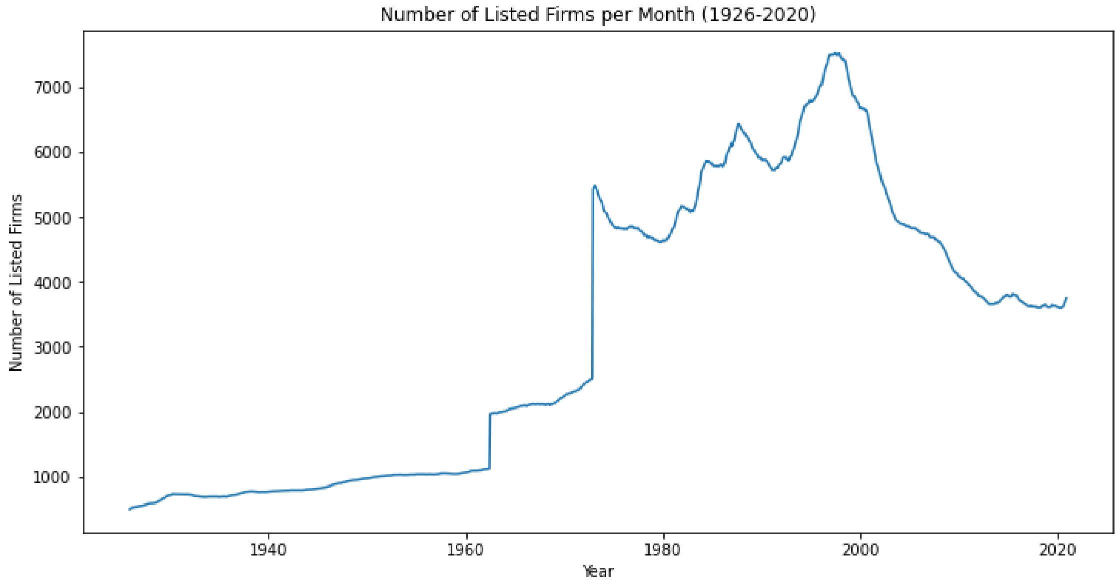
[ ]:
```
crsp
```

[ ]:
```
         PERMNO        date  SHRCD  EXCHCD        PRC       RET    SHROUT
1         10000  1986-01-31   10.0     3.0       <NA>         C    3680.0
2         10000  1986-02-28   10.0     3.0       <NA> -0.257143    3680.0
3         10000  1986-03-31   10.0     3.0       <NA>  0.365385    3680.0
4         10000  1986-04-30   10.0     3.0       <NA> -0.098592    3793.0
5         10000  1986-05-30   10.0     3.0       <NA> -0.222656    3793.0
…           …         …     …      …         …         …         …
4705164   93436  2020-08-31   11.0     3.0  498.32001  0.741452  931809.0
4705165   93436  2020-09-30   11.0     3.0  429.01001 -0.139087  948000.0
4705166   93436  2020-10-30   11.0     3.0  388.04001 -0.095499  947901.0
4705167   93436  2020-11-30   11.0     3.0  567.59998  0.462736  947901.0
4705168   93436  2020-12-31   11.0     3.0  705.66998  0.243252  959854.0

[3630644 rows x 7 columns]
```

[ ]:
```
# Group by month and count unique permno
monthly_counts = crsp.groupby(pd.to_datetime(crsp['date']).dt.
↪to_period('M'))['PERMNO'].nunique()

# Plot monthly counts over time
fig, ax = plt.subplots(figsize=(12, 6))
ax.plot(monthly_counts.index.to_timestamp(), monthly_counts.values)
ax.set_xlabel('Year')
ax.set_ylabel('Number of Listed Firms')
ax.set_title('Number of Listed Firms per Month (1926-2020)')
plt.show()
```



[ ]:
```
crsp.to_csv("data/cleaned_crsp.csv")
```

[ ]:

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')
```

```python
vw_returns = portfolio_returns.pivot_table(values='vw_ret', index='date',
    columns='decile')
```

2 B

[ ]:
```python
# 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()
```

```python
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.

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

[ ]:
```
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}")
```

```python
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

[ ]:
```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

# 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
```

[ ]:
```python
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.

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)

[ ]:
```
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
```

```python
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)

[ ]:
```python
# 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']])
```

```
# 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

Problem5

April 23, 2023

[ ]:
```python
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()
```

```
/Users/esmirmesic/opt/anaconda3/envs/bem114/lib/python3.11/site-
packages/tqdm/auto.py:21: TqdmWarning: IProgress not found. Please update
jupyter and ipywidgets. See
https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

1 A

[ ]:
```python
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')
```

[ ]:
```python
import pandas_datareader as pdr

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


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
```

```
ff12 = pdr.get_data_famafrench('12_industry_Portfolios', start=start_date,␣
    ↪end=end_date)[0]
ff12 = ff12 / 100
ff12.index = ff12.index.to_timestamp('M')
```

```
/var/folders/sg/4dp480wd1cjd288xvby34rpr0000gn/T/ipykernel_12833/3641822038.py:7
: FutureWarning: The argument 'date_parser' is deprecated and will be removed in
a future version. Please use 'date_format' instead, or read your data in as
'object' dtype and then call 'to_datetime'.
  ff5_factors = pdr.get_data_famafrench('F-F_Research_Data_5_Factors_2x3',
start=start_date, end=end_date)[0]
/var/folders/sg/4dp480wd1cjd288xvby34rpr0000gn/T/ipykernel_12833/3641822038.py:7
: FutureWarning: The argument 'date_parser' is deprecated and will be removed in
a future version. Please use 'date_format' instead, or read your data in as
'object' dtype and then call 'to_datetime'.
  ff5_factors = pdr.get_data_famafrench('F-F_Research_Data_5_Factors_2x3',
start=start_date, end=end_date)[0]
/var/folders/sg/4dp480wd1cjd288xvby34rpr0000gn/T/ipykernel_12833/3641822038.py:1
1: FutureWarning: The argument 'date_parser' is deprecated and will be removed
in a future version. Please use 'date_format' instead, or read your data in as
'object' dtype and then call 'to_datetime'.
  ff12 = pdr.get_data_famafrench('12_industry_Portfolios', start=start_date,
end=end_date)[0]
/var/folders/sg/4dp480wd1cjd288xvby34rpr0000gn/T/ipykernel_12833/3641822038.py:1
1: FutureWarning: The argument 'date_parser' is deprecated and will be removed
in a future version. Please use 'date_format' instead, or read your data in as
'object' dtype and then call 'to_datetime'.
  ff12 = pdr.get_data_famafrench('12_industry_Portfolios', start=start_date,
end=end_date)[0]
/var/folders/sg/4dp480wd1cjd288xvby34rpr0000gn/T/ipykernel_12833/3641822038.py:1
1: FutureWarning: The argument 'date_parser' is deprecated and will be removed
in a future version. Please use 'date_format' instead, or read your data in as
'object' dtype and then call 'to_datetime'.
  ff12 = pdr.get_data_famafrench('12_industry_Portfolios', start=start_date,
end=end_date)[0]
/var/folders/sg/4dp480wd1cjd288xvby34rpr0000gn/T/ipykernel_12833/3641822038.py:1
1: FutureWarning: The argument 'date_parser' is deprecated and will be removed
in a future version. Please use 'date_format' instead, or read your data in as
'object' dtype and then call 'to_datetime'.
  ff12 = pdr.get_data_famafrench('12_industry_Portfolios', start=start_date,
end=end_date)[0]
/var/folders/sg/4dp480wd1cjd288xvby34rpr0000gn/T/ipykernel_12833/3641822038.py:1
1: FutureWarning: The argument 'date_parser' is deprecated and will be removed
in a future version. Please use 'date_format' instead, or read your data in as
'object' dtype and then call 'to_datetime'.
  ff12 = pdr.get_data_famafrench('12_industry_Portfolios', start=start_date,
end=end_date)[0]
```

```
/var/folders/sg/4dp480wd1cjd288xvby34rpr0000gn/T/ipykernel_12833/3641822038.py:1
1: FutureWarning: The argument 'date_parser' is deprecated and will be removed
in a future version. Please use 'date_format' instead, or read your data in as
'object' dtype and then call 'to_datetime'.
  ff12 = pdr.get_data_famafrench('12_industry_Portfolios', start=start_date,
end=end_date)[0]
/var/folders/sg/4dp480wd1cjd288xvby34rpr0000gn/T/ipykernel_12833/3641822038.py:1
1: FutureWarning: The argument 'date_parser' is deprecated and will be removed
in a future version. Please use 'date_format' instead, or read your data in as
'object' dtype and then call 'to_datetime'.
  ff12 = pdr.get_data_famafrench('12_industry_Portfolios', start=start_date,
end=end_date)[0]
/var/folders/sg/4dp480wd1cjd288xvby34rpr0000gn/T/ipykernel_12833/3641822038.py:1
1: FutureWarning: The argument 'date_parser' is deprecated and will be removed
in a future version. Please use 'date_format' instead, or read your data in as
'object' dtype and then call 'to_datetime'.
  ff12 = pdr.get_data_famafrench('12_industry_Portfolios', start=start_date,
end=end_date)[0]
```

[ ]:
```
import yfinance as yf

top10_holdings = pd.read_csv("data/top10_holdings_brk_arkk.csv")

# Get BRK-A and ARKK data from Yahoo Finance
brk = yf.download("BRK-A", start="1980-01-31", end="2020-12-31", interval="1mo")
brk.index = pd.to_datetime(brk.index)
arkk = yf.download("ARKK", start="2014-10-31", end="2020-12-31", interval="1mo")
arkk.index = pd.to_datetime(arkk.index)

# Make sure the index is datetime
brk.index = pd.to_datetime(brk.index)
arkk.index = pd.to_datetime(arkk.index)

brk.index = brk.index.to_period("M").to_timestamp("M")
arkk.index = arkk.index.to_period("M").to_timestamp("M")

# Calculate monthly stock returns
brk["Return"] = brk["Adj Close"].pct_change()
arkk["Return"] = arkk["Adj Close"].pct_change()

brk = brk.dropna()
arkk = arkk.dropna()

# Estimate the FF5 model for each strategy over their full histories and the␣
 ↪same sample period
# Merge data
brk_ff5 = pd.merge(brk, ff5_factors, left_index=True, right_index=True)
```

```
arkk_ff5 = pd.merge(arkk, ff5_factors, left_index=True, right_index=True)

# Find the common time period for both stocks
start_date = max(brk_ff5.index.min(), arkk_ff5.index.min())
end_date = min(brk_ff5.index.max(), arkk_ff5.index.max())

# Create the same sample period data
brk_ff5_same_period = brk_ff5.loc[start_date:end_date]
arkk_ff5_same_period = arkk_ff5.loc[start_date:end_date]

# Perform regressions for the same sample period
X_brk_same_period = sm.add_constant(brk_ff5_same_period[["Mkt-RF", "SMB",␣
↪"HML", "RMW", "CMA"]])
X_arkk_same_period = sm.add_constant(arkk_ff5_same_period[["Mkt-RF", "SMB",␣
↪"HML", "RMW", "CMA"]])

model_brk_same_period = sm.OLS(brk_ff5_same_period["Return"],␣
↪X_brk_same_period).fit()
model_arkk_same_period = sm.OLS(arkk_ff5_same_period["Return"],␣
↪X_arkk_same_period).fit()

# Perform regressions
X_brk = sm.add_constant(brk_ff5[["Mkt-RF", "SMB", "HML", "RMW", "CMA"]])
X_arkk = sm.add_constant(arkk_ff5[["Mkt-RF", "SMB", "HML", "RMW", "CMA"]])

model_brk = sm.OLS(brk_ff5["Return"], X_brk).fit()
model_arkk = sm.OLS(arkk_ff5["Return"], X_arkk).fit()

# Regress returns for each strategy on the Fama French 12 Industry Portfolios
X_brk_ff12 = sm.add_constant(ff12.loc[brk_ff5.index])
X_arkk_ff12 = sm.add_constant(ff12.loc[arkk_ff5.index])

model_brk_ff12 = sm.OLS(brk_ff5["Return"], X_brk_ff12).fit()
model_arkk_ff12 = sm.OLS(arkk_ff5["Return"], X_arkk_ff12).fit()
```

```
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
```

[ ]:
```
model_arkk.summary(), model_brk.summary()
```

[ ]:
```
(<class 'statsmodels.iolib.summary.Summary'>
 """
                             OLS Regression Results
 ==============================================================================
 Dep. Variable:                 Return   R-squared:                       0.814
 Model:                            OLS   Adj. R-squared:                  0.800
 Method:                 Least Squares   F-statistic:                     58.61
```

```
Date:                Sun, 23 Apr 2023   Prob (F-statistic):           3.92e-23
Time:                        20:34:29   Log-Likelihood:                 136.38
No. Observations:                  73   AIC:                            -260.8
Df Residuals:                      67   BIC:                            -247.0
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0062      0.005      1.266      0.210      -0.004       0.016
Mkt-RF         1.5209      0.120     12.672      0.000       1.281       1.760
SMB            0.5290      0.212      2.493      0.015       0.105       0.953
HML           -0.7020      0.194     -3.619      0.001      -1.089      -0.315
RMW           -0.1581      0.341     -0.464      0.644      -0.839       0.523
CMA           -0.7889      0.348     -2.265      0.027      -1.484      -0.094
==============================================================================
Omnibus:                       10.376   Durbin-Watson:                   2.246
Prob(Omnibus):                  0.006   Jarque-Bera (JB):               10.269
Skew:                           0.839   Prob(JB):                      0.00589
Kurtosis:                       3.750   Cond. No.                         80.9
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
""",
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                 Return   R-squared:                       0.341
Model:                            OLS   Adj. R-squared:                  0.333
Method:                 Least Squares   F-statistic:                     43.96
Date:                Sun, 23 Apr 2023   Prob (F-statistic):           1.60e-36
Time:                        20:34:29   Log-Likelihood:                 670.86
No. Observations:                 431   AIC:                            -1330.
Df Residuals:                     425   BIC:                            -1305.
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0074      0.003      2.824      0.005       0.002       0.013
Mkt-RF         0.8224      0.063     13.044      0.000       0.698       0.946
SMB           -0.3503      0.094     -3.745      0.000      -0.534      -0.166
HML            0.4275      0.114      3.745      0.000       0.203       0.652
RMW            0.3473      0.123      2.832      0.005       0.106       0.588
```

```
CMA          -0.0129      0.176     -0.073      0.942      -0.358       0.333
==============================================================================
Omnibus:                       94.435   Durbin-Watson:                   2.078
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              212.375
Skew:                           1.125   Prob(JB):                     7.65e-47
Kurtosis:                       5.601   Cond. No.                         79.6
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
""")
```

[ ]:
```
model_arkk_ff12.summary(), model_brk_ff12.summary()
```

[ ]:
```
(<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                 Return   R-squared:                       0.833
Model:                            OLS   Adj. R-squared:                  0.800
Method:                 Least Squares   F-statistic:                     25.02
Date:                Sun, 23 Apr 2023   Prob (F-statistic):           5.99e-19
Time:                        20:34:29   Log-Likelihood:                 140.43
No. Observations:                  73   AIC:                            -254.9
Df Residuals:                      60   BIC:                            -225.1
Df Model:                          12
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0052      0.005      0.977      0.332      -0.005       0.016
NoDur          0.0154      0.262      0.059      0.953      -0.510       0.540
Durbl          0.3145      0.094      3.360      0.001       0.127       0.502
Manuf          0.4304      0.295      1.461      0.149      -0.159       1.020
Enrgy          0.0141      0.094      0.149      0.882      -0.174       0.203
Chems         -0.3705      0.272     -1.364      0.178      -0.914       0.173
BusEq          0.9235      0.194      4.750      0.000       0.535       1.312
Telcm         -0.2029      0.217     -0.936      0.353      -0.637       0.231
Utils         -0.0491      0.169     -0.290      0.773      -0.387       0.289
Shops          0.0344      0.231      0.149      0.882      -0.428       0.497
Hlth           0.5693      0.179      3.179      0.002       0.211       0.927
Money         -0.4316      0.208     -2.071      0.043      -0.849      -0.015
Other          0.0753      0.419      0.180      0.858      -0.763       0.914
==============================================================================
Omnibus:                        1.505   Durbin-Watson:                   2.602
Prob(Omnibus):                  0.471   Jarque-Bera (JB):                1.006
```

```
Skew:                           0.274   Prob(JB):                        0.605
Kurtosis:                       3.173   Cond. No.                         107.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
""",
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                 Return   R-squared:                       0.402
Model:                            OLS   Adj. R-squared:                  0.385
Method:                 Least Squares   F-statistic:                     23.45
Date:                Sun, 23 Apr 2023   Prob (F-statistic):           7.44e-40
Time:                        20:34:29   Log-Likelihood:                 691.94
No. Observations:                 431   AIC:                            -1358.
Df Residuals:                     418   BIC:                            -1305.
Df Model:                          12
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0066      0.003      2.602      0.010       0.002       0.011
NoDur          0.3477      0.123      2.833      0.005       0.106       0.589
Durbl         -0.0154      0.059     -0.260      0.795      -0.132       0.101
Manuf         -0.1576      0.143     -1.104      0.270      -0.438       0.123
Enrgy         -0.0013      0.055     -0.024      0.981      -0.109       0.107
Chems          0.0557      0.119      0.468      0.640      -0.178       0.290
BusEq         -0.2874      0.062     -4.634      0.000      -0.409      -0.165
Telcm          0.0550      0.076      0.722      0.470      -0.095       0.205
Utils          0.0322      0.080      0.404      0.686      -0.124       0.189
Shops          0.1805      0.103      1.755      0.080      -0.022       0.383
Hlth          -0.0022      0.083     -0.026      0.979      -0.165       0.161
Money          0.3294      0.087      3.803      0.000       0.159       0.500
Other          0.3441      0.149      2.303      0.022       0.050       0.638
==============================================================================
Omnibus:                       95.333   Durbin-Watson:                   2.075
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              245.090
Skew:                           1.080   Prob(JB):                     6.02e-54
Kurtosis:                       5.998   Cond. No.                         75.0
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
```

```
""")
```

[ ]:
```
model_arkk_same_period.summary(), model_brk_same_period.summary()
```

[ ]:
```
(<class 'statsmodels.iolib.summary.Summary'>
 """
                             OLS Regression Results
 ==============================================================================
 Dep. Variable:                 Return   R-squared:                       0.814
 Model:                            OLS   Adj. R-squared:                  0.800
 Method:                 Least Squares   F-statistic:                     58.61
 Date:                Sun, 23 Apr 2023   Prob (F-statistic):           3.92e-23
 Time:                        20:34:26   Log-Likelihood:                 136.38
 No. Observations:                  73   AIC:                            -260.8
 Df Residuals:                      67   BIC:                            -247.0
 Df Model:                           5
 Covariance Type:            nonrobust
 ==============================================================================
                  coef    std err          t      P>|t|      [0.025      0.975]
 ------------------------------------------------------------------------------
 const          0.0062      0.005      1.266      0.210      -0.004       0.016
 Mkt-RF         1.5209      0.120     12.672      0.000       1.281       1.760
 SMB            0.5290      0.212      2.493      0.015       0.105       0.953
 HML           -0.7020      0.194     -3.619      0.001      -1.089      -0.315
 RMW           -0.1581      0.341     -0.464      0.644      -0.839       0.523
 CMA           -0.7889      0.348     -2.265      0.027      -1.484      -0.094
 ==============================================================================
 Omnibus:                       10.376   Durbin-Watson:                   2.246
 Prob(Omnibus):                  0.006   Jarque-Bera (JB):               10.269
 Skew:                           0.839   Prob(JB):                      0.00589
 Kurtosis:                       3.750   Cond. No.                         80.9
 ==============================================================================

 Notes:
 [1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
 """,
 <class 'statsmodels.iolib.summary.Summary'>
 """
                             OLS Regression Results
 ==============================================================================
 Dep. Variable:                 Return   R-squared:                       0.703
 Model:                            OLS   Adj. R-squared:                  0.681
 Method:                 Least Squares   F-statistic:                     31.68
 Date:                Sun, 23 Apr 2023   Prob (F-statistic):           2.07e-16
 Time:                        20:34:26   Log-Likelihood:                 163.59
 No. Observations:                  73   AIC:                            -315.2
```

```
Df Residuals:                      67   BIC:                            -301.4
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0010      0.003      0.293      0.770      -0.006       0.008
Mkt-RF         0.9092      0.083     10.997      0.000       0.744       1.074
SMB           -0.4787      0.146     -3.275      0.002      -0.771      -0.187
HML            0.3490      0.134      2.612      0.011       0.082       0.616
RMW            0.0164      0.235      0.070      0.945      -0.453       0.485
CMA            0.3688      0.240      1.537      0.129      -0.110       0.848
==============================================================================
Omnibus:                        0.446   Durbin-Watson:                   2.159
Prob(Omnibus):                  0.800   Jarque-Bera (JB):                0.532
Skew:                          -0.174   Prob(JB):                        0.766
Kurtosis:                       2.769   Cond. No.                         80.9
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
""")
```

2 B.

Cathie Wood and Warren Buffet do not appear to have similar investment strategies (during the period in which we have data from both). Based on the OLS regression over the FF12 data, the coefficients across their two models vary drastically, indicating that their models are different.

Warren Buffett is more like a value investor due to his positive and statistically significant HML coefficient (both historically and recently, although he has been acting less like a value investor in recent periods, as indicated by his decline in HML). In contrast, Cathie Wood has a negative and statistically significant HML coefficient, indicating that she is acting more like a growth investor than a value investor.

Warren Buffett's portfolio behaves closest to Consumer Nondurables (NoDur), Shops, Banking Sector (Money), and (Other). Cathie Wood's portfolio behaves closest to Consumer Durables (Durble), Manufacturing (Manuf), Business Equipment (BusEq), Health (Hlth).

Both Buffett and Wood focus on consumer goods (although different types), and Buffett focuses on Banking. The top 10 holdings focus on banking and consumer goods, so the portfolio behavior analysis tracks.