

Appendix

Data Analysis

This script implements all data processing, co-integration testing, calculations of buy and hold cumulative returns, and implementations of Bollinger Bands Statistical Arbitrage

```
1 # This assignment implements the statistical analysis required for the
2 # 705 co-integration/causality assignment
3
4 import numpy as np # arithmetic operations
5 import pandas as pd # data analysis package
6 import csv as csv # read and write csvs
7 import random as rd # random functionality
8 import saspy as sas # Use saspy functionality in python
9 import matplotlib.pyplot as plt # Use MatLab functionality for plotting
10 import seaborn as sb # Imports seaborn library for use
11 import wrds as wrds# Wharton Research Data Services API
12 import pydatastream as pds # Thomas Reuters Datastream API
13 import yfinance as yf # Yahoo Finance API
14 import datetime as dt # Manipulate datetime values
15 import statsmodels.api as sm # Create Stats functionalities
16 # import johansen as jh # Ability to implement Johansen test to test
17 # for co-integration
18 import linearmodels as lp # Ability to use PooledOLS
19 from sklearn.linear_model import LinearRegression
20 from stargazer.stargazer import Stargazer
21 import finance_byu as fin # Python Package for Fama-MacBeth Regressions
22 from statsmodels.regression.rolling import RollingOLS # Use factor
23 # loadings
24 from stargazer.stargazer import Stargazer
25 import sympy as sy # convert latex code
26 import scipy as sc # Scipy packages
27 import tabulate as tb # Create tables in python
28 import itertools as it # Find combinations of lists
29
30 # This section contains useful links for mean reversion, pairs-trading
31 # https://letianzj.github.io/mean-reversion.html
32 # https://letianzj.github.io/cointegration-pairs-trading.html
33 # https://en.wikipedia.org/wiki/Cointegration#Engle%20%93Granger\_two-step\_method
34
35 # Defines the pairs trading function for the first part of the
36 # assignment
37 # Requires cross-sectional, time-series data of stock/bond/forex
38 # returns
39 def pairs_trading(data, asset_1, asset_2):
40     """[summary]
41
42     Args:
43         data ([type]): [description]
44         asset_1 ([type]): [description]
45         asset_2 ([type]): [description]
46
47     Returns:
48         [type]: [description]
```

```

45 """
46 # Start date to produce plots
47 # Produces time-series plots overlaying one security with another,
48 regression residuals
49 # Add produces a one-dimensional array of residuals
50 # Test both configurations
51 # Initialise tstat
52
53 # Determines suitable combinations of security pairs for pairs
54 # trading.
55 # Firstly, implements the Cointegrated Augmented Dickey-Fuller (CADF)
56 # test to determine optimal
57 # Hedge ratio by linear regression against the two stocks and then
58 tests for stationarity
59 # of the residuals. CADF is also known as Engle-Granger Two-Step
60 Method
61 tstat_coint = np.inf
62 # Set up defaults
63 independant = [asset_1,asset_2]
64 dependant = [asset_2, asset_1]
65 for i in range(len(independant)):
66     x = data[independant[i]].values.reshape(-1,1)
67     y = data[dependant[i]].values
68     lm_model = LinearRegression(copy_X=True, fit_intercept=True,
69 normalize=False).fit(x, y) # fit() expects 2D array
70     # print('parameters: %.7f, %.7f' %(lm_model.intercept_,
71     lm_model.coef_))
72     yfit = lm_model.coef_* data[independant[i]] + lm_model.
73     intercept_
74     res = data[dependant[i]] - yfit
75     [tstat, pvalue, num_lags, num_obs, crit_values, icbest] = sm.
76     tsa.stattools.adfuller(res, maxlag = 1)
77     # print('tstat',tstat)
78     if tstat < tstat_coint:
79         # Update critical values
80         tstat_coint = tstat
81         pvalue_coint = pvalue
82         num_lags_coint = num_lags
83         num_obs_coint = num_obs
84         crit_values_coint = crit_values
85         icbest_coint = icbest
86         hedge_ratio= lm_model.coef_
87         inte_coint = lm_model.intercept_
88         x_name = independant[i]
89         y_name = dependant[i]
90         data['res'] = res
91
92
93 # Check if significant co-integration exists
94 if tstat_coint < -2.86: # Critical Value
95     status = 'Yes'
96     # Plots the residuals and prices separately
97     # Plots the prices
98     data.plot(x='Date', y=[x_name, y_name], kind='line')
99     plt.title('Time-series of Price: Independant:' + x_name + ','
00     Dependant:' + y_name)
01     plt.ylabel('Price')
02     plt.xlabel('Date')

```

```

92     plt.savefig('results/regressions/' + x_name + '-' + y_name + '-'
93     regression.png')
94     # Plots the residuals
95     data.plot(x='Date', y='res', kind='line')
96     plt.title('Time-series of Residuals: Independant:' + x_name + ', '
97     ,Dependant:' + y_name)
98     plt.ylabel('Residual')
99     plt.xlabel('Date')
100    plt.savefig('results/residual/' + x_name + '-' + y_name + '-'
101    residuals.png')
102    # Plots the residuals
103
104
105    # Implements the Johansen test to mitigate accumulating errors
106    in the two step process
107    # Find the Hedge Ratio and Tests for Co-integration at the same
108    time (Can be extended to
109    # more than two stocks (Implement if there is time)!

110
111    # Calculate log prices and returns for trading strategies (
112    Check when add more stocks to the mix)
113    # Prints proposed spreads
114    data[x_name+'-log-price'] = np.log(data[x_name])
115    data[y_name+'-log-price'] = np.log(data[y_name])
116    # Standard
117    data['trading-spread'] = data[y_name] - data[x_name]
118    data['trading-spread-mean'] = data['trading-spread'].rolling(
119    window=20).mean()
120    data['trading-spread-std'] = data['trading-spread'].rolling(
121    window=20).std()
122    # Log (spread = log(x) - nlog(b))
123    data['log-spread-price'] = data[y_name+'-log-price'] -
124    hedge_ratio*data[x_name+'-log-price']
125    data['log-spread-price-mean'] = data['log-spread-price'].rolling(
126    window=20).mean()
127    data['log-spread-price-std'] = data['log-spread-price'].rolling(
128    window=20).std()
129    # Calculates the Bollinger Bands
130    # Sets scalar multipier
131    scaler = 2
132    # Calculates bands
133    # Log Bands
134    data['log-spread-price-upper'] = data['log-spread-price-mean'] +
135    (scaler*data['log-spread-price-std'])
136    data['log-spread-price-lower'] = data['log-spread-price-mean'] -
137    (scaler*data['log-spread-price-std'])
138    data['spread-price-upper'] = data['trading-spread-mean'] + (
139    scaler*data['trading-spread-std'])
140    data['spread-price-lower'] = data['trading-spread-mean'] - (
141    scaler*data['trading-spread-std'])

142
143    # Plot Log Price with Bollinger Bands
144    data.plot(x='Date', y=['log-spread-price','log-spread-price-
145    mean','log-spread-price-upper','log-spread-price-lower'], kind='line
146    ')
147    plt.title('20 Moving Average Log Spread with Bollinger Bands:' +
148    x_name + '-' + y_name)
149    plt.ylabel('Spreads')
150    plt.xlabel('Date')

```

```

132     plt.legend(loc=2)
133     plt.savefig('results/logspreads/' + x_name + '-' + y_name + '.png')
134
135     # Plot Price Spread with Bollinger Bands
136     # Plot Log Price with Bollinger Bands
137     # data.plot(x='Date', y=['trading-spread','trading-spread-mean',
138     ', 'spread-price-upper','spread-price-lower'], kind='line')
139     # plt.title('20 Moving Average Spread with Bollinger Bands:' +
140     x_name + '-' + y_name)
141     # plt.ylabel('Spreads')
142     # plt.xlabel('Date')
143     # plt.legend(loc=2)
144     # plt.savefig('charts/spread-' + x_name + '-' + y_name + '.png')
145
146     # Implements Bollinger Trading Strategy (Assumes only trading
147     on the spread, not accounting for transactions costs)
148     # Initialise size of trade (Assumes order size of 1000, no
149     current positions
150     initial_capital = 1000
151     money_at_risk_percentage = 0.01
152     cents_at_risk = 0.10
153     # Equation to determine order size
154     order_size = initial_capital * money_at_risk_percentage /
155     cents_at_risk
156     capital = initial_capital # Time zero
157     hr = hedge_ratio
158     x_name_current_size = 0
159     y_name_current_size = 0
160
161     # Set lagged variables, positions and capital
162     data['lagged-' + x_name] = data[x_name].shift(1)
163     data['lagged-' + y_name] = data[y_name].shift(1)
164     data['lagged-log-spread-price'] = data['log-spread-price'].shift(1)
165     data['lagged-log-spread-price-mean'] = data['log-spread-price-
166     mean'].shift(1)
167     data[x_name + 'size'] = 0
168     data[y_name + 'size'] = 0
169     data[x_name + 'order_size'] = 0
170     data[y_name + 'order_size'] = 0
171     data['capital'] = 0
172     # margin = revenue - costs
173     data['margin'] = 0
174
175     # Implements statistical arbitrage trading strategies
176     for index, row in data.iterrows():
177         # Hit Upper Band, Short the Spread
178         if (data.at[index, 'log-spread-price'] > data.at[index, 'log-
179         spread-price-upper']) and (x_name_current_size >= 0):
180             capital = capital - int(hr*order_size)*data.at[index,
181             x_name] + int(order_size)*data.at[index, y_name]
182             # x_name
183             data.at[index, x_name + 'order_size'] = - int(hr*
184             order_size) - x_name_current_size
185             x_name_current_size = - int(hr*order_size)
186             # y_name
187             data.at[index, y_name + 'order_size'] = order_size -
188             y_name_current_size

```

```

179         y_name_current_size = order_size
180         # margin = revenue - costs
181         data.at[index, 'margin'] = -1*data.at[index, x_name+',
182             order_size]*data.at[index, x_name] - 1*data.at[index, y_name+',
183             order_size]*data.at[index, y_name]
184
184         # Hit Lower Band, Long the Spread
185         elif (data.at[index, 'log-spread-price'] < data.at[index,
186             log-spread-price-lower]) and (x_name_current_size <= 0):
187             capital = capital + int(hr*order_size)*data.at[index,
188                 x_name] - int(order_size)*data.at[index, y_name]
189                 # x_name
190                 data.at[index, x_name+order_size] = int(hr*order_size)
191                 - x_name_current_size
192                 x_name_current_size = int(hr*order_size)
193                 # y_name
194                 data.at[index, y_name+order_size] = - order_size -
195                 y_name_current_size
196                 y_name_current_size = - order_size
197                 # margin = revenue - costs
198                 data.at[index, 'margin'] = -1*data.at[index, x_name+',
199             order_size]*data.at[index, x_name] - 1*data.at[index, y_name+',
200             order_size]*data.at[index, y_name]
201
201         # Spread crosses from below average, flat long position
202         elif (data.at[index, 'log-spread-price'] > data.at[index,
203             log-spread-price-mean]) and (data.at[index, 'lagged-log-spread-price
204             '] < data.at[index, 'lagged-log-spread-price-mean']) and (
205             x_name_current_size > 0):
206             capital = capital - int(x_name_current_size)*data.at[
207                 index, x_name] + int(y_name_current_size)*data.at[index, y_name]
208                 # x_name
209                 data.at[index, x_name+order_size] = -
210                 x_name_current_size
211                 x_name_current_size = 0
212                 # y_name
213                 data.at[index, y_name+order_size] = -
214                 y_name_current_size
215                 y_name_current_size = 0
216                 # margin = revenue - costs
217                 data.at[index, 'margin'] = -1*data.at[index, x_name+',
218             order_size]*data.at[index, x_name] - 1*data.at[index, y_name+',
219             order_size]*data.at[index, y_name]
220
220         # Spread crosses from above average, flat/cover short
221         position
222         elif (data.at[index, 'log-spread-price'] < data.at[index,
223             log-spread-price-mean]) and (data.at[index, 'lagged-log-spread-price
224             '] > data.at[index, 'lagged-log-spread-price-mean']) and (
225             x_name_current_size < 0):
226             capital = capital + int(x_name_current_size)*data.at[
227                 index, x_name] - int(y_name_current_size)*data.at[index, y_name]
228                 # x_name
229                 data.at[index, x_name+order_size] = -
230                 x_name_current_size
231                 x_name_current_size = 0
232                 # y_name

```

```

214         data.at[index,y_name+'order_size'] = -
215         y_name_current_size
216         y_name_current_size = 0
217         # margin = revenue - costs
218         data.at[index,'margin'] = -1*data.at[index,x_name+',
219         order_size']*data.at[index,x_name] - 1*data.at[index,y_name+',
220         order_size']*data.at[index,y_name]
221
222
223
224
225     # Determine
226
227
228     # Calculates the margin generated from holding all the
229     # realative positions
230     # This is the price of the position multiplied by the position
231     # size held
232     data['gain'] = data[x_name+'size']*data[x_name] + data[y_name+',
233     size]*data[y_name]
234
235
236
237     # Calculate cumulative-gains returns dataframe
238     for index, row in data.iterrows():
239         if index == 0:
240             data.at[index,'accum-gain'] = data.at[index, 'gain']
241         if index > 0:
242             data.at[index,'accum-gain'] = data.at[index, 'gain'] +
243             data.at[index-1,'accum-gain']
244
245
246
247     # Calculate statistical arbitrage cumulative return by dividing
248     # accumulative gain by initial cpatial
249
250
251     # Plots the gains
252     # data.plot(x='Date', y=['gain'], kind='line', figsize=(28,18))
253     # plt.title('Gains on Trades-:' +x_name+'-' +y_name)
254     # plt.ylabel('Gain', fontsize = 30)
255     # plt.xlabel('Date', fontsize = 30)
256     # plt.legend(loc=2, prop={'size': 30})
257     # plt.xticks(size = 18)
258     # plt.yticks(size = 18)
259     # plt.savefig('charts/gain-' +x_name+'-' +y_name+'.png')
260
261
262     # Plots the accumukated margin
263     data.plot(x='Date', y=['accum-gain'], kind='line')
264     plt.title('Accumulated Gain ($):' +x_name+'-' +y_name)
265     plt.ylabel('Gain')
266     plt.xlabel('Date')
267     plt.savefig('results/gains/' +x_name+'-' +y_name+'.png')
268
269
270     # Plots the order sizes
271     # data.plot(x='Date', y=[x_name+'order_size',y_name+'order_size'],
272     # ], kind='line', figsize=(28,18))
273     # plt.title('Order Sizes-:' +x_name+'-' +y_name, fontsize = 30)
274     # plt.ylabel('Order Size', fontsize = 30)
275     # plt.xlabel('Date', fontsize = 30)
276     # plt.legend(loc=2, prop={'size': 30})

```

```

263     # plt.xticks(size = 18)
264     # plt.yticks(size = 18)
265     # plt.savefig('charts/order-size-' +x_name+'-'+y_name+'.png')
266
267     # Plots the Positions
268     # data.plot(x='Date', y=[x_name+'size',y_name+'size'], kind='line',
269     #             figsize=(28,18))
270     # plt.title('Number of Positions -:' +x_name+'-'+y_name)
271     # plt.ylabel('Positions', fontsize = 30)
272     # plt.xlabel('Date', fontsize = 30)
273     # plt.legend(loc=2, prop={'size': 30})
274     # plt.xticks(size = 18)
275     # plt.yticks(size = 18)
276     # plt.savefig('charts/positions-' +x_name+'-'+y_name+'.png')
277
278     # Calculate buy and hold return over forecast periods
279     data = data.sort_values(by='Date')
280     data['bhr'] = (data['trading-spread']/data['trading-spread'].
281 shift(1)) - 1
282     # Calculate Statistical Arbitrage Returns
283     data['bbr'] = data['accum-gain']
284     # data['bbr'] = (data['accum-gain']/data['accum-gain'].shift(1)
285 ) - 1
286     data = data.dropna()
287     data.reset_index(inplace = True, drop = True)
288
289     # Calculates Buy and Hold Cuumulative Returns
290     for index, row in data.iterrows():
291         if index == 0:
292             # Buy and Hold Strategy
293             bhcr = data.at[index, 'bhr']
294             data.at[index, 'bhcr'] = bhcr
295             # Statistical Arbitrage
296             # bbcr = data.at[index, 'bbr']
297             # data.at[index, 'bbcr'] = bbcr
298         if index > 0:
299             # Buys and hold strategy
300             bhcr = ((1+data.at[index, 'bhr'])*(1+data.at[index-1,
301 'bhcr']))-1
302             data.at[index, 'bhcr'] = bhcr
303             # Statistical Arbitrage
304             # bbcr = ((1+data.at[index, 'bbr'])*(1+data.at[index-1,
305 'bbcr']))-1
306             # data.at[index, 'bbcr'] = bbcr
307
308             # Return last values cumulative return
309             bhr = data.at[data.index[-1], 'bhcr']
310             # Return
311             cg = data.at[data.index[-1], 'accum-gain']
312
313             # Calculate cumulative return from statistical arbitrage
314             # strategy
315             # Plots the accumukated margin
316             data.plot(x='Date', y=['bhcr'], kind='line')
317             plt.title('Cumulative Buy & Hold Returns ($):' +x_name+'-'+
318 y_name)
319             plt.ylabel('Buy & Hold Returns')
320             plt.xlabel('Date')

```

```

314     plt.savefig('results/returns/buy-hold/' + x_name + '-' + y_name + '.png')
315
316     data.plot(x='Date', y=['accum-gain'], kind='line')
317     plt.title('Cumulative Returns ($):' + x_name + '-' + y_name)
318     plt.ylabel('Buy and Hold Returns')
319     plt.xlabel('Date')
320     plt.savefig('results/returns/bollinger/' + x_name + '-' + y_name + '.png')
321
322     # Calculate cumulative return from statistical arbitrage
323     # strategy
324     # Plots the accumulated margin
325     # data.plot(x='Date', y=['bhr', 'bbr'], kind='line', figsize
326     =(28,18))
327     # plt.title('Returns ($):' + x_name + '-' + y_name)
328     # plt.ylabel('Returns', fontsize = 30)
329     # plt.xlabel('Date', fontsize = 30)
330     # plt.legend(loc=2, prop={'size': 30})
331     # plt.xticks(size = 18)
332     # plt.yticks(size = 18)
333     # plt.savefig('charts/returns-' + x_name + '-' + y_name + '.png')
334
335     # Calculates some average values for tranquil and crisis period
336     # Tranquil period
337     tran_start = '1/9/19'
338     tran_end = '28/02/20'
339     cris_start = '1/3/20'
340     cris_end = '31/8/20'
341     # Get tranquil dataframe
342     tranquil = data[data["Date"] > tran_start]
343     tranquil = tranquil[tranquill["Date"] <= tran_end]
344     # Get crisis dataframe
345     crisis = data[data["Date"] > cris_start]
346     crisis = crisis[crisis["Date"] <= cris_end]
347
348     # Find the averages for those periods
349     tran_average_buy_hold = tranquil['bhr'].mean()
350     tran_average_gain = tranquil['gain'].mean()
351     cris_average_buy_hold = crisis['bhr'].mean()
352     cris_average_gain = crisis['gain'].mean()
353
354     # Returns variables from the function
355     return x_name, y_name, tstat_coint, hedge_ratio, bhr, cg,
356     order_size, status, tran_average_buy_hold, tran_average_gain,
357     cris_average_buy_hold, cris_average_gain
358
359     # Implements the trading strategy with both bands
360 else:
361     print("Co-integration between pairs does not exist")
362     status = 'No'
363     hedge_ratio = np.nan
364     bhr = np.nan
365     cg = np.nan
366     order_size = np.nan
367     tran_average_buy_hold = np.nan
368     tran_average_gain = np.nan

```

```

366     cris_average_buy_hold = np.nan
367     cris_average_gain = np.nan
368     return x_name, y_name, tstat_coint, hedge_ratio, bhr, cg,
369     order_size, status, tran_average_buy_hold, tran_average_gain,
370     cris_average_buy_hold, cris_average_gain
371
372 # Defines the benchmarking function for the second part of the
373 # assignment (Placeholder)
374 def benchmarking(self):
375     """[summary]
376
377     Returns:
378         [type]: [description]
379     """
380     return self
381
382 # Defines the financial co-integration and causality function (
383 # Placeholder)
384 def contagion_causality(self):
385     """[summary]
386
387     Returns:
388         [type]: [description]
389     """
390     return self
391
392 # Downloads financial data from yahoo finance (ExxonMobil and Chevron
393 # to Test Co-Integration Example)
394 # This is to be replaced with the data outputs
395 # https://towardsdatascience.com/a-comprehensive-guide-to-downloading-
396 # stock-prices-in-python-2cd93ff821d4
397 # Test case setup
398 start_date = '2000-01-01'
399 end_date = '2019-12-31'
400 prices_1 = 'EWA'
401 prices_2 = 'EWC'
402
403 asset_1 = yf.download(prices_1, start = start_date, end = end_date,
404 progress = False)
405 asset_1.to_csv('data/'+prices_1+'.csv')
406
407 asset_2 = yf.download(prices_2, start = start_date, end = end_date,
408 progress = False)
409 asset_2.to_csv('data/'+prices_2+'.csv')
410
411 # Import data
412 asset_1_df = pd.read_csv('data/'+prices_1+'.csv')
413 asset_2_df = pd.read_csv('data/'+prices_2+'.csv')
414
415 # Calculate the returns
416 asset_1_df[prices_1 + '-ret-(%)'] = (asset_1_df['Adj Close']/asset_1_df[
417     'Adj Close'].shift(1))-1
418 asset_1_df.rename(columns= {'Adj Close':prices_1}, inplace = True)
419 asset_1_df = asset_1_df.dropna(axis=0)
420
421 asset_2_df[prices_2 + '-ret-(%)'] = (asset_2_df['Adj Close']/asset_2_df[
422     'Adj Close'].shift(1))-1

```

```

414 asset_2_df.rename(columns= {'Adj Close':prices_2}, inplace = True)
415 asset_2_df = asset_2_df.dropna(axis=0)
416
417 # Merge the data into one dataframe
418 data_df = pd.merge(asset_1_df[['Date',prices_1,prices_1 +'-ret-(%)']].copy(),
419                     asset_2_df[['Date',prices_2,prices_2 +'-ret-(%)']].copy(),
420                     how='left', left_on=['Date'], right_on = ['Date'])
421 data_df = data_df.dropna(axis=0)
422 data = data_df[['Date',prices_1,prices_2]].copy()
423
424 # Calls the pairs trading function
425 x_name, y_name, tstat_coint, hedge_ratio, bhr, cg, order_size, status,
426     tran_average_buy_hold, tran_average_gain, cris_average_buy_hold,
427     cris_average_gain = pairs_trading(data,prices_1,prices_2)
428 # Establishes
429 test_results_table = pd.DataFrame(columns=[ 'Variable (x)', 'Variable (y)',
430                                         'tstat', 'Hedge Ratio', 'Buy & Hold Cumulative Return',
431                                         'Cumulative Gain (Bollinger Bands)', 'Order Size', 'Co-integration'])
432 # Creates new row to add to empty dataframe
433 new_row = {'Variable (x)':x_name, 'Variable (y)': y_name,'tstat':
434             tstat_coint,'Hedge Ratio': hedge_ratio, 'Buy & Hold Cumulative
435             Return': bhr, 'Cumulative Gain (Bollinger Bands)':cg, 'Order Size':
436             order_size, 'Co-integration': status}
437 test_results_table = test_results_table.append(new_row, ignore_index =
438                                               True)
439 # Rank via tStat (Indicates strength of mean reversion
440 test_results_table.sort_values(by = 'tstat')
441 test_results_table.to_excel('results/test_results_table.xlsx')
442
443 # Conduct pairs trading analysis for list of resources (Steel Stocks)
444 # Loads in data for pairs trading analysis
445 resources_data = pd.read_excel('data/data.xlsx')
446
447 # Get the pricing information for the data (List of names)
448 assets = list(resources_data.columns.values)
449 assets = assets[1:-1]
450
451 # Creates combinations for pairs analysis
452 pair_order_list = list(it.combinations(assets,2))
453
454 # Cleans data of values and re_index
455 resources_data = resources_data.dropna(axis = 0)
456 resources_data.reset_index(inplace = True, drop = True)
457
458 # Initialises final resources table
459 final_results_table = pd.DataFrame(columns=[ 'Variable (x)', 'Variable (y)',
460                                         'tstat', 'Hedge Ratio', 'Buy & Hold Cumulative Return',
461                                         'Cumulative Gain (Bollinger Bands)', 'Order Size', 'Co-integration',
462                                         'tran_average_buy_hold', 'tran_average_gain', 'cris_average_buy_hold',
463                                         'cris_average_gain'])
464 for pair in pair_order_list:
465     try:
466         x_name, y_name, tstat_coint, hedge_ratio, bhr, cg, order_size,
467         status, tran_average_buy_hold, tran_average_gain,
468         cris_average_buy_hold, cris_average_gain = pairs_trading(
469             resources_data,pair[0],pair[1])
470         new_row = {'Variable (x)':x_name, 'Variable (y)': y_name,'tstat':
471             tstat_coint,'Hedge Ratio': hedge_ratio, 'Buy & Hold Cumulative

```

```
        Return': bhr, 'Cumulative Gain (Bollinger Bands)':cg, 'Order Size':  
order_size, 'Co-integration': status,'tran_average_buy_hold':  
tran_average_buy_hold, 'tran_average_gain':tran_average_gain, '  
cris_average_buy_hold':cris_average_buy_hold, 'cris_average_gain':  
cris_average_gain}  
454     final_results_table = final_results_table.append(new_row,  
ignore_index = True)  
455     print('Finished: ', pair)  
456 except:  
457     print('Error occurred')  
458  
459 # Rank via tStat (Indicates strength of mean reversion  
460 final_results_table.sort_values(by = 'tstat')  
461 final_results_table.to_excel('results/rank/final_results_table.xlsx')
```

Appendix

Data Processing

This script imports and updates all raw portfolio returns, Fama-French, BMG, Momentum, daily, and monthly data from assignment and Kenneth R. French sources.

```
1 # This completes the data processing for the BMG Empirical Assignment
2
3 # Imports important python packages
4 import numpy as np # arithmetic operations
5 import pandas as pd # data analysis package
6 import csv as csv # read and write csvs
7 import random as rd # random functionality
8 import saspy as sas # Use saspy functionality in python
9 import matplotlib.pyplot as plt # Use MatLab functionality for plotting
10 import seaborn as sb # Imports seaborn library for use
11 import wrds as wrds# Wharton Research Data Services API
12 import pydatastream as pds # Thomas Reuters Datastream API
13 import yfinance as yf # Yahoo Finance API
14 import datetime as dt # Manipulate datetime values
15 import statsmodels.api as sm # Create Stats functionalities
16 import sklearn as sl # ML functionality
17 from stargazer.stargazer import Stargazer
18 import finance_byu as fin # Python Package for Fama-MacBeth Regressions
19
20
21 # Creates dataframes to convert data
22 # Daily
23 pd_df = pd.read_excel('data.xlsx', sheet_name = 'portfolio_daily')
24 ffd_df = pd.read_excel('data.xlsx', sheet_name = 'fama_french_daily')
25 bmgd_df = pd.read_excel('data.xlsx', sheet_name = 'bmg_daily')
26 da_df = pd.read_excel('data.xlsx', sheet_name = 'daily_all')
27
28 # Monthly
29 pm_df = pd.read_excel('data.xlsx', sheet_name = 'portfolio_monthly')
30 ffm_df = pd.read_excel('data.xlsx', sheet_name = 'fama_french_monthly')
31 bmgm_df = pd.read_excel('data.xlsx', sheet_name = 'bmg_monthly')
32 ma_df = pd.read_excel('data.xlsx', sheet_name = 'monthly_all')
33
34 # Converts fama-french factors from percentages to fractions
35 # Daily
36 ffd_df['mktrf'] = ffd_df['mktrf']/100
37 ffd_df['smb'] = ffd_df['smb']/100
38 ffd_df['hml'] = ffd_df['hml']/100
39 ffd_df['rf'] = ffd_df['rf']/100
40 ffd_df['umd'] = ffd_df['umd']/100
41
42 # Monthly
43 ffm_df['rf'] = ffm_df['rf']/100
44
45 # Convert data columns from timestamps to datetime to enable matching
46 pd_df['date'] = pd.to_datetime(pd_df['date'], unit='s')
47 ffd_df['date'] = pd.to_datetime(ffd_df['date'], unit='s')
48 bmgd_df['date'] = pd.to_datetime(bmgd_df['date'], unit='s')
49 da_df['date'] = pd.to_datetime(da_df['date'], unit='s')
50
51 # Creates to dataframes
```

```

52 uda_df = da_df.copy()
53 uma_df = ma_df.copy()
54
55 # Daily adjustments and additions
56 # Updates umd and bmg prior to adding new additions
57 for index, row in uda_df.iterrows():
58     date = uda_df.at[index, 'date']
59     try:
60         # Gets the factors
61         factors_df = ffd_df.loc[ffd_df['date']== date]
62         # print(factors_df.head())
63         bmgd_df = bmgd_df.loc[bmgd_df['date']== date]
64         # print(bmgd_df.head())
65         # Changes the value
66         uda_df.at[index, 'umd']= factors_df.iloc[0]['umd']
67         uda_df.at[index, 'bmg'] = bmgd_df.iloc[0]['bmg']
68     except:
69         print("Error updating the umd and bmg factor (2010 - 2016)")
70
71 # Add the portofolio returns data to the updated dataframes
72 # Sets sequence for portfolio returns
73 portfolios = list(range(1,31))
74 for index, row in pd_df.iterrows():
75     # Set the time period imformation
76     year = pd_df.at[index, 'year']
77     month = pd_df.at[index, 'month']
78     day = pd_df.at[index, 'day']
79     date = pd_df.at[index, 'date']
80     # Set the factor elements based on dates with index matching from
81     # dataframes
82     # Locates the factors at the required date
83     # Try statement to skip entries when portfolio, factor and bmg
84     # dates don't align.
85     try:
86         factors_df = ffd_df.loc[ffd_df['date']== date]
87         bmgd_df = bmwd_df.loc[bmgd_df['date']== date]
88         mktrf = factors_df.iloc[0]['mktrf']
89         smb = factors_df.iloc[0]['smb']
90         hml = factors_df.iloc[0]['hml']
91         rf = factors_df.iloc[0]['rf']
92         umd = factors_df.iloc[0]['umd']
93         bmg = bmgd_df.iloc[0]['bmg']
94         # Add the portfolio components
95         for portfolio in portfolios:
96             ret = pd_df.at[index,portfolio]
97             # Creates dataframe to append
98             d = {'ind': [portfolio], 'ret': [ret], 'year': [year],
99                  'month': [month], 'day': [day], 'date': [date], 'mktrf': [mktrf],
100                 'smb': [smb], 'hml': [hml], 'rf': [rf], 'umd': [umd], 'bmg': [bmg]}
101             row_df = pd.DataFrame(data=d)
102             # Append to dataframe (use assignment)
103             uda_df = uda_df.append(row_df, ignore_index= True)
104     except:
105         # Documents date omissions
106         print("Warning - Error")
107         line = date.strftime("%m/%d/%Y, %H:%M:%S")
108         with open('omissions-daily.txt', 'a+') as f:
109             f.seek(0)

```

```

106     data = f.read(100)
107     if len(data) > 0 :
108         f.write("\n")
109         # Append text at the end of file
110         f.write(line)
111 # Create new csv file
112 uda_df.to_csv('updated_daily_all.csv')
113
114 # Monthly adjustments and additions
115 # Add the portofolio returns data to the updated dataframes
116 # Sets sequence for portfolio returns
117 portfolios = list(range(1,31))
118 for index, row in pm_df.iterrows():
119     # Set the time period imformation
120     year = pm_df.at[index, 'year']
121     month = pm_df.at[index, 'month']
122     day = pm_df.at[index, 'day']
123     date = pm_df.at[index, 'date']
124     eom = pm_df.at[index, 'eom']
125     # Set the factor elements based on dates with index matching from
126     # dataframes
127     # Locates the factors at the required date
128     # print("index: ",index)
129     # print("date: ",date)
130     # Try statement to skip entries when portfolio, factor and bmg
131     # dates don't align.
132     try:
133         factors_df = ffm_df.loc[ffm_df['eom']== eom]
134         rf = factors_df.iloc[0]['rf']
135         # Add the portfolio components
136         for portfolio in portfolios:
137             ret = pm_df.at[index,portfolio]
138             # Creates dataframe to append
139             d = {'year': [year], 'month': [month], 'day': [day], 'date': [
140                 date], 'eom': [eom], 'ind': [portfolio] , 'ret': [ret], 'rf':[rf]}
141             row_df = pd.DataFrame(data=d)
142             # Append to dataframe (use assignment)
143             uma_df = uma_df.append(row_df,ignore_index= True)
144     except:
145         # Documents date omissions
146         print("Warning - Error")
147         line = date.strftime("%m/%d/%Y, %H:%M:%S")
148         with open('omissions-monthly.txt', 'a+') as f:
149             f.seek(0)
150             data = f.read(100)
151             if len(data) > 0 :
152                 f.write("\n")
153                 # Append text at the end of file
154                 f.write(line)
155 # Create new csv file
156 uma_df.to_csv('updated_monthly_all.csv')
157 uma_df.to_excel('updated_monthly_all.xlsx')

```

Data Analysis

This script implements all data analysis performed in the assignment.

```
1 # This completes the data analysis for the BMG Empirical Assignment
2 # Note: Data is processed using the finance-761-data-processing script
3
4 # Imports important python packages and data from data processing
5 import numpy as np # arithmetic operations
6 import pandas as pd # data analysis package
7 import csv as csv # read and write csvs
8 import random as rd # random functionality
9 import saspy as sas # Use saspy functionality in python
10 import matplotlib.pyplot as plt # Use MatLab functionality for plotting
11 import seaborn as sb # Imports seaborn library for use
12 import wrds as wrds# Wharton Research Data Services API
13 import pydatastream as pds # Thomas Reuters Datastream API
14 import yfinance as yf # Yahoo Finance API
15 import datetime as dt # Manipulate datetime values
16 import statsmodels.api as sm # Create Stats functionalities
17 import linearmodels as lp # Ability to use PooledOLS
18 import sklearn as sl # ML functionality
19 from stargazer.stargazer import Stargazer
20 import finance_byu as fin # Python Package for Fama-MacBeth Regressions
21 from statsmodels.regression.rolling import RollingOLS # Use factor
    loadings
22 from stargazer.stargazer import Stargazer
23 import sympy as sy # convert latex code
24 import scipy as sc # Scipy packages
25 import tabulate as tb # Create tables in python
26
27 # Establishes plotting setting for rolling regressions
28 sb.set_style('darkgrid')
29 pd.plotting.register_matplotlib_converters()
30
31 # Reads data csvs as dataframes
32 daily_all_df = pd.read_csv("updated_daily_all.csv")
33 monthly_all_df = pd.read_csv("updated_monthly_all.csv")
34
35
36 # Creates excess return variable for both daily and monthly
37 daily_all_df['eret'] = daily_all_df['ret']/100 - daily_all_df['rf'] # 
    Check if this step is necessary
38 monthly_all_df['eret'] = monthly_all_df['ret']/100 - monthly_all_df['rf']
    ] # Check if this step is necessary
39
40 # Creates month variables to both the daily and monthly sets
41 daily_all_df['m'] = daily_all_df['month'] + daily_all_df['year']*12
42 monthly_all_df['m'] = monthly_all_df['month'] + monthly_all_df['year']
    ]*12
43
44 daily_all_df.to_excel('excel/daily_all_df_excel_check.xlsx')
45 monthly_all_df.to_excel('excel/monthly_all_df_excel_check.xlsx')
46
47 # Correctly sorts the columns
48 daily_all_df.sort_values(by=['ind','year','month'],ascending=True,
    inplace=True)
49
50 # Creates a unique list of month values
```

```

51 m_list = sorted(np.unique(daily_all_df['m']))
52 ind_list = sorted(np.unique(daily_all_df['ind']))
53
54 # Shifted for Stage 2 of the Fama MacBeth Regression
55 # monthly_all_df['eret'] = (monthly_all_df.sort_values(by=['m'],
56 #                                         ascending=True)
57 #                                         .groupby(['ind'])['eret'].shift(-1))
58
59 # Drop NaN Datas
60 monthly_all_df = monthly_all_df.dropna(axis=0, how = 'any')
61
62 # Start Fama-Macbeth Regressions
63
64 # This is Stage 1 of the Fama-Macbeth Regression - Estimate Factor
65 # Loadings (Crossed Checked)
66 # https://en.wikipedia.org/wiki/Fama%20%93MacBeth_regression
67 # Create a new dataframe with every possible combination of month and
68 # index combinations available
69 factor_df = pd.DataFrame(columns=['ind', 'm', 'mktrf', 'smb', 'hml', 'umd',
70 'bmgi'])
71 for i in ind_list:
72     for j in m_list:
73         # Loops over factor dataframe to get the desired values
74         # Get slice of dataframe based on multiple columns
75         index_df = daily_all_df[daily_all_df['ind'] == i]
76         slice_df = index_df[index_df['m'] == j]
77         # Perform the OLS Regressions on the sliced dataframne
78         y = slice_df['eret']
79         x = slice_df[['mktrf', 'smb', 'hml', 'umd', 'bmgi']]
80         x = sm.add_constant(x)
81         model = sm.OLS(y,x).fit()
82         # Save model parameters to column dataframe
83         new_row = {'ind':i, 'm':j, 'mktrf':model.params[1], 'smb':model.
84         params[2], 'hml':model.params[3], 'umd':model.params[4], 'bmgi':model.
85         params[5]}
86         # Append factor loading to the factors to the dataframe
87         factor_df = factor_df.append(new_row, ignore_index = True)
88
89 # Produce average industry beta
90 average_ffb_df = pd.DataFrame(columns=['m', 'bmgi'])
91 for m in m_list:
92     cross_sectional_df = factor_df[factor_df['m'] == m]
93     new_row = {'m':m, 'bmgi':cross_sectional_df['bmgi'].mean()}
94     print(new_row)
95     average_ffb_df = average_ffb_df.append(new_row, ignore_index= True)
96
97 # PLOT average ffb
98 average_ffb_df.plot(x='m', y=['bmgi'], kind='line', figsize=(28,18))
99 plt.title('Average Industry BMG Time Series', fontsize = 30)
100 plt.ylabel('BMG', fontsize = 24)
101 plt.xlabel('Time (Date)', fontsize = 24)
102 plt.legend(loc=2, prop={'size': 18})
103 plt.xticks(size = 18)
104 plt.yticks(size = 18)
105 # Add caption to below plot python
106 plt.savefig('plots/bmg-time-series-premium.png')
107

```

```

103 raw_fama_macbeth_df = pd.merge(monthly_all_df, factor_df[['ind', 'm', 'mktrf', 'smb', 'hml', 'umd', 'bmg']].copy(), how='right', left_on=['ind', 'm'], right_on = ['ind', 'm'])
104 raw_fama_macbeth_df.to_excel('excel/raw_fama_macbeth_df.xlsx')
105
106
107 # # Shift factors forward by one value (month)
108 factor_df['mktrf'] = (factor_df.sort_values(by=['m'], ascending=True)
109                         .groupby(['ind'])['mktrf'].shift(1))
110 factor_df['smb'] = (factor_df.sort_values(by=['m'], ascending=True)
111                         .groupby(['ind'])['smb'].shift(1))
112 factor_df['hml'] = (factor_df.sort_values(by=['m'], ascending=True)
113                         .groupby(['ind'])['hml'].shift(1))
114 factor_df['umd'] = (factor_df.sort_values(by=['m'], ascending=True)
115                         .groupby(['ind'])['umd'].shift(1))
116 factor_df['bmg'] = (factor_df.sort_values(by=['m'], ascending=True)
117                         .groupby(['ind'])['bmg'].shift(1))
118
119 # Drop NaN Datas
120 factor_df = factor_df.dropna(axis=0, how = 'any')
121 # Merge the regression co-efficients with monthly dataset
122 fama_macbeth_df = pd.merge(monthly_all_df, factor_df[['ind', 'm', 'mktrf', 'smb', 'hml', 'umd', 'bmg']].copy(), how='right', left_on=['ind', 'm'], right_on = ['ind', 'm'])
123 # Drops rows with NaN
124 fama_macbeth_df = fama_macbeth_df.dropna(axis=0, how = 'any')
125 # Saves Fama-Macbeth Regression Results to Excel
126 fama_macbeth_df.to_excel('excel/processed_fama_macbeth_df.xlsx')
127
128
129 time_series_df = pd.DataFrame(columns=['ind', 'value', 'alpha', 'mktrf', 'smb', 'hml', 'umd', 'bmg'])
130 # Work out the Fama-French Time series
131 for i in ind_list:
132     # Gets section of dataframe for monthly date (i.e. days for ind x )
133     cross_sectional_df = daily_all_df[daily_all_df['ind'] == i]
134     # Performs regression using the cross section to get factor price
135     y = cross_sectional_df['eret']
136     x = cross_sectional_df[['mktrf', 'smb', 'hml', 'umd', 'bmg']]
137     # print(cross_sectional_df.head(n=30))
138     # input("Press Enter to continue...")
139     x = sm.add_constant(x)
140     model = RollingOLS(y,x).fit()
141     model = sm.OLS(y,x).fit()
142     # Save model parameters to column dataframe
143     new_row_coef = {'ind':i, 'value':'co-efficient', 'alpha': model.params[0], 'mktrf':model.params[1], 'smb':model.params[2], 'hml':model.params[3], 'umd':model.params[4], 'bmg':model.params[5]}
144     # Save model pvalues
145     new_row_pvalue = {'ind':i, 'value':'pvalue', 'alpha': model.pvalues[0], 'mktrf':model.pvalues[1], 'smb':model.pvalues[2], 'hml':model.pvalues[3], 'umd':model.pvalues[4], 'bmg':model.pvalues[5]}
146     # new_row_se = {'m':i, 'alpha': model.std_errors[0], 'mktrf':model.std_errors[1], 'smb':model.std_errors[2], 'hml':model.std_errors[3], 'umd':model.std_errors[4], 'bmg':model.std_errors[5]}
147     # Append the new row to the dataframe
148     time_series_df = time_series_df.append(new_row_coef, ignore_index = True)

```

```

149     time_series_df = time_series_df.append(new_row_pvalue, ignore_index
150         = True)
150 # factor_price_se_df = factor_price_se_df.append(new_row_se,
151 ignore_index = True)
151 # Append models to list
152 stargazer = Stargazer([model])
153 stargazer.custom_columns('FF Time-series-' + str(i))
154 expr = stargazer.render_latex()
155 sy.preview(expr, viewer='file', filename='time-series/' + str(i) + '-'
155 regression.png')
156
157 # Prints the time-series (all months) accross industries
158 time_series_df.to_excel('excel/time-series-regression-table.xlsx')
159
160
161 # This is Stage 2 of the Fama-Macbeth Regression - Estimate Factor
161     Prices from Monthly Data
162 # Create factor pricing dataframe
163 factor_price_df = pd.DataFrame(columns=['m', 'alpha', 'mktrf', 'smb',
163     'hml', 'umd', 'bmgi'])
164 factor_price_se_df = pd.DataFrame(columns=['m', 'alpha', 'mktrf', 'smb',
164     'hml', 'umd', 'bmgi'])
165
166 # Update the m list as excludes the first month
167 m_list = sorted(np.unique(fama_macbeth_df['m']))
168
169 # Run cross-sectional regression for each time period using monthly
169     data
170 for i in m_list:
171     # Gets section of dataframe for monthly date (i.e. ind 1-30 for
171     month x )
172     cross_sectional_df = fama_macbeth_df[fama_macbeth_df['m'] == i]
173     # Performs regression using the cross section to get factor price
174     y = cross_sectional_df['eret']
175     x = cross_sectional_df[['mktrf', 'smb', 'hml', 'umd', 'bmgi']]
176     # print(cross_sectional_df.head(n=30))
177     # input("Press Enter to continue...")
178     x = sm.add_constant(x)
179     # model = sm.OLS(endog = y,exog = x).fit()
180     model = sm.OLS(y,x).fit()
181     # Save model parameters to column dataframe
182     new_row_price = {'m':i, 'alpha': model.params[0], 'mktrf':model.
182     params[1], 'smb':model.params[2], 'hml':model.params[3], 'umd':model.
182     params[4], 'bmgi':model.params[5]}
183     # new_row_se = {'m':i, 'alpha': model.std_errors[0], 'mktrf':model.
183     std_errors[1], 'smb':model.std_errors[2], 'hml':model.std_errors[3],
183     'umd':model.std_errors[4], 'bmgi':model.std_errors[5]}
184     # Append the new row to the dataframe
185     factor_price_df = factor_price_df.append(new_row_price,
185     ignore_index = True)
186     # factor_price_se_df = factor_price_se_df.append(new_row_se,
186     ignore_index = True)
187     # Append models to list
188     stargazer = Stargazer([model])
189     expr = stargazer.render_latex()
190     sy.preview(expr, viewer='file', filename='statistical-tables/' +
190 str(i) + '-regression.png')
191

```

```

192 # Converts Factor Prices to Excel
193 factor_price_df.to_excel('excel/ffb_stage_2_df.xlsx')
194
195 # Plot the dataframe
196 factor_price_df.plot(x='m', y=['bmgi'], kind='line', figsize=(28,18))
197 plt.title('Average Industry BMG Risk Premium Time Series')
198 plt.ylabel('BMG Risk Premium', fontsize = 24)
199 plt.xlabel('Time (Date)', fontsize = 24)
200 plt.legend(loc=2, prop={'size': 18})
201 plt.xticks(size = 18)
202 plt.yticks(size = 18)
203 # Add caption to below plot python
204 plt.savefig('plots/bmg-risk-premium.png')
205
206 # This is Stage 3 of the Fama-Macbeth Regression - Estimate average
207 # factor pricing and error ()
208 # Calculate estimated factor prices across all time periods
209 factor_prices_average_dict = {'alpha': factor_price_df['alpha'].mean(),
210   'mktrf':factor_price_df['mktrf'].mean(),'smb':factor_price_df['smb']
211   ].mean(),'hml':factor_price_df['hml'].mean(),'umd':factor_price_df['
212   umd'].mean(),'bmgi':factor_price_df['bmgi'].mean()}
213 # factor_prices_se_average_dict = {'alpha': factor_price_se_df['alpha
214   '].mean(),'mktrf':factor_price_se_df['mktrf'].mean(),'smb':
215   factor_price_se_df['smb'].mean(),'hml':factor_price_se_df['hml']
216   ].mean(),'umd':factor_price_se_df['umd'].mean(),'bmgi':
217   factor_price_se_df['bmgi'].mean()}
218
219 # Calculates unbiased standard error of the mean over requested axis
220 # https://www.geeksforgeeks.org/python-pandas-dataframe-sem/
221 factor_prices_sem_average_dict = {'alpha': factor_price_df['alpha'].sem
222   (),'mktrf':factor_price_df['mktrf'].sem(),'smb':factor_price_df['smb
223   '].sem(),'hml':factor_price_df['hml'].sem(),'umd':factor_price_df['
224   umd'].sem(),'bmgi':factor_price_df['bmgi'].sem()}
225
226 # Print dictionaries to display factor prices and standard errors
227 alpha_mean = factor_price_df['alpha'].mean()
228 mktrf_mean = factor_price_df['mktrf'].mean()
229 smb_mean = factor_price_df['smb'].mean()
230 hml_mean = factor_price_df['hml'].mean()
231 umd_mean = factor_price_df['umd'].mean()
232 bmgi_mean = factor_price_df['bmgi'].mean()
233
234 # Performs one sample ttest on all variables
235 bmgi_tstat,bmgi_pvalue = sc.stats.ttest_1samp(a=factor_price_df['bmgi'],
236   popmean=factor_price_df['bmgi'].mean())
237 mktrf_tstat,mktrf_pvalue = sc.stats.ttest_1samp(a=factor_price_df['
238   mktrf'], popmean=factor_price_df['mktrf'].mean())
239 smb_tstat,smb_pvalue = sc.stats.ttest_1samp(a=factor_price_df['smb'],
240   popmean=factor_price_df['smb'].mean())
241 hml_tstat,hml_pvalue = sc.stats.ttest_1samp(a=factor_price_df['hml'],
242   popmean=factor_price_df['hml'].mean())
243 umd_tstat,umd_pvalue = sc.stats.ttest_1samp(a=factor_price_df['umd'],
244   popmean=factor_price_df['umd'].mean())
245 alpha_tstat,alpha_pvalue = sc.stats.ttest_1samp(a=factor_price_df['
246   alpha'], popmean=factor_price_df['alpha'].mean())
247
248 # Create dataframe
249 head = ['name', 'mean', 'tstat', 'pvalue']

```

```

233 names = ['alpha', 'mktrf', 'smb', 'hml', 'umd', 'bmgi']
234 means = [alpha_mean, mktrf_mean, smb_mean, hml_mean, umd_mean, bmgi_mean
235 ]
235 tstats = [alpha_tstat, mktrf_tstat, smb_tstat, hml_tstat, umd_tstat,
236     bmgi_tstat]
236 pvalues = [alpha_pvalue, mktrf_pvalue, smb_pvalue, hml_pvalue,
237     umd_pvalue, bmgi_pvalue]
237
238 ffb_statistics = pd.DataFrame(columns=[head[0], head[1], head[2], head
239 [3]])
239 # For loop to create
240 for i in range(len(names)):
241     new_row = {head[0]:names[i],head[1]:means[i], head[2]:tstats[i],
242     head[3]:pvalues[i]}
243     ffb_statistics = ffb_statistics.append(new_row, ignore_index = True
244 )
243 # Save to CSV
244 ffb_statistics.to_excel('excel/ffb_statistics.xlsx')
245
246 # Additional Analysis 1: Rolling Regression
247 # Implements Rolling Regressions for each industry (1-30) rolling
248 # through months in regressing
249 # https://www.statsmodels.org/dev/examples/notebooks/generated/
250 #     rolling_ls.html
250 exog_vars = ['mktrf', 'smb', 'hml', 'umd', 'bmgi']
251 for i in ind_list:
252     cross_sectional_df = daily_all_df[daily_all_df['ind'] == i]
253     # Create eret dataframe
254     eret_df = cross_sectional_df[['date', 'eret']].copy()
255     exog = sm.add_constant(cross_sectional_df[exog_vars])
256     rols = RollingOLS(eret_df['eret'], exog, window=len(m_list))
257     rres = rols.fit()
258     fig = rres.plot_recursive_coefficient(variables=exog_vars, figsize
259 =(14,18))
260     path = "rolling-regressions/" + str(i) + "-rolling-regression.png"
261     plt.savefig(path)
262
261 # Additional Analysis 2: Hedging Positions
262 # Implements Hedging Portfolio based on BMG rankings on the first date
263 # (Monthly)
263 # Imports S&P 500 Data
264 sp500_df = pd.read_excel('sp500.xlsx', sheet_name = 'sp500')
265
266 ranking_df = pd.DataFrame(columns=['ind', 'mean'])
267 # Sets bmgi ranking from fama_macbeth
268 for i in ind_list:
269     # This is an index
270     index_rank_df = fama_macbeth_df[fama_macbeth_df['ind'] == i]
271     new_row = {'ind':i, 'mean': index_rank_df['bmgi'].mean()}
272     ranking_df = ranking_df.append(new_row, ignore_index = True)
273     bmgi_good_df = ranking_df.sort_values(by = 'mean', ascending = True).head
274     (5)
274     bmgi_bad_df = ranking_df.sort_values(by = 'mean', ascending = True).tail
275     (5)
275 # Create green list (Top 5, Green Stocks)
276 bmgi_good = bmgi_good_df['ind'].to_list()
277 # Create brown list (Bottom 5, Brown Stocks)
278 bmgi_bad = bmgi_bad_df['ind'].to_list()

```

```

279 # Create new dateframes with Python
280 good_returns = pd.DataFrame(columns=[ 'm' , str(bmg_good[0]) , str(bmg_good
281 [1]) , str(bmg_good[2]) , str(bmg_good[3]) , str(bmg_good[4]) ])
282 bad_returns = pd.DataFrame(columns=[ 'm' , str(bmg_bad[0]) , str(bmg_bad[1])
283 , str(bmg_bad[2]) , str(bmg_bad[3]) , str(bmg_bad[4]) ])
284
285 # Starts cumulative returns calculations
286 # Top 5 Green Stocks
287 for j in m_list:
288     # Empty list to append to
289     emp = []
290     for i in bmg_good:
291         #Gets the slicesd row
292         index_df = fama_macbeth_df[fama_macbeth_df['ind'] == i]
293         slice_df = index_df[index_df['m'] == j]
294         idx = slice_df.loc[slice_df['ind'] == i].index
295         emp.append(slice_df.at[idx[0] , 'ret'])
296         # emp.append(slice_df.at[idx[0] , 'ret'] - slice_df.at[idx[0] , 'rf
297         '] *100)
298     # Append new row of returns data
299     new_row = { 'm':j , str(bmg_good[0]): emp[0] , str(bmg_good[1]): emp[1] ,
300     str(bmg_good[2]): emp[2] , str(bmg_good[3]): emp[3] , str(bmg_good[4]):
301     : emp[4] }
302     good_returns = good_returns.append(new_row , ignore_index = True)
303
304 # Create equally weighting returns from the columns
305 good_returns[ 'ret' ] = ((good_returns[ str(bmg_good[0]) ] + good_returns[ str(bmg_good[1]) ] + good_returns[ str(bmg_good[2]) ] + good_returns[ str(bmg_good[3]) ] + good_returns[ str(bmg_good[4]) ])/len(bmg_good)) /100
306
307 # Calculate cumulative returns dataframe
308 for index , row in good_returns.iterrows():
309     if index == 0:
310         good_returns.at[index , 'cr' ] = 0
311     if index > 0:
312         good_returns.at[index , 'cr' ] = ((1+good_returns.at[index , 'ret' ]) *
313         *(1+good_returns.at[index-1 , 'cr' ]))-1
314
315 good_returns.to_excel('excel/bmg_green.xlsx')
316 # Top 5 Brown Stocks
317 for j in m_list:
318     # Empty list to append to
319     emp = []
320     for i in bmg_bad:
321         #Gets the slicesd row
322         index_df = fama_macbeth_df[fama_macbeth_df['ind'] == i]
323         slice_df = index_df[index_df['m'] == j]
324         idx = slice_df.loc[slice_df['ind'] == i].index
325         emp.append(slice_df.at[idx[0] , 'ret'])
326         # emp.append(slice_df.at[idx[0] , 'ret'] - slice_df.at[idx[0] , 'rf
327         '] *100)
328     # Append new row of returns data
329     new_row = { 'm':j , str(bmg_bad[0]): emp[0] , str(bmg_bad[1]): emp[1] ,
330     str(bmg_bad[2]): emp[2] , str(bmg_bad[3]): emp[3] , str(bmg_bad[4]): emp[4] }
331     bad_returns = bad_returns.append(new_row , ignore_index = True)

```

```

325
326 # Create equally weighting returns from the columns
327 bad_returns['ret'] = ((bad_returns[str(bmg_bad[0])] + bad_returns[str(
328     bmg_bad[1])] + bad_returns[str(bmg_bad[2])] + bad_returns[str(
329     bmg_bad[3])] + bad_returns[str(bmg_bad[4]))]/len(bmg_bad))/100
330
331 # Calculate cumulative returns dataframe
332 for index, row in bad_returns.iterrows():
333     if index == 0:
334         bad_returns.at[index, 'cr'] = 0
335     if index > 0:
336         bad_returns.at[index, 'cr'] = ((1+bad_returns.at[index, 'ret']) *
337             *(1+bad_returns.at[index-1, 'cr']))-1
338
339 # Saves bad_returns to excel
340 bad_returns.to_excel('excel/bmg_brown.xlsx')
341 # Create the hedge cumulative returns
342 hedge_ret_df = good_returns[['m']].copy()
343 hedge_ret_df = pd.merge(hedge_ret_df, good_returns[['m', 'ret']], how='
344     left', left_on=['m'], right_on = ['m'])
345 hedge_ret_df = hedge_ret_df.rename(columns = {'ret':'bmg_green_ret'})
346 hedge_ret_df = pd.merge(hedge_ret_df, bad_returns[['m', 'ret']], how='
347     left', left_on=['m'], right_on = ['m'])
348 hedge_ret_df = hedge_ret_df.rename(columns = {'ret':'bmg_brown_ret'})
349 hedge_ret_df['hedge_ret'] = hedge_ret_df['bmg_green_ret'] -
350     hedge_ret_df['bmg_brown_ret']
351 for index, row in hedge_ret_df.iterrows():
352     if index == 0:
353         hedge_ret_df.at[index, 'cr'] = 0
354     if index > 0:
355         hedge_ret_df.at[index, 'cr'] = ((1+hedge_ret_df.at[index, 'hedge_ret']) *
356             *(1+hedge_ret_df.at[index-1, 'cr']))-1
357
358 # Plot the cumulartive returns
359 bad_returns.plot(x ='m', y='cr', kind = 'line')
360 plt.savefig('plots/bad_bmg_returns.png')
361
362 # Sets cumulative returns calculation
363 green_cr = good_returns[['m', 'cr']].copy()
364 green_cr = green_cr.rename(columns = {'cr':'bmg_green'})
365 brown_cr = bad_returns[['m', 'cr']].copy()
366 brown_cr = brown_cr.rename(columns = {'cr':'bmg_brown'})
367 sp500_cr = sp500_df[['m', 'cr']].copy()
368 sp500_cr = sp500_cr.rename(columns = {'cr':'sp500'})
369 hedge_cr = hedge_ret_df[['m', 'cr']].copy()
370 hedge_cr = hedge_cr.rename(columns = {'cr':'hedge'})
371
372 # hedge = pd.DataFrame(columns=['m', 'bmg_good', 'bmg_bad', 'sp500'])
373 hedge_df = good_returns[['m']].copy()
374
375 # Merge dataframes
376 hedge_df = pd.merge(hedge_df, green_cr, how='left', left_on=['m'],
377     right_on = ['m'])
378 hedge_df = pd.merge(hedge_df, brown_cr, how='left', left_on=['m'],
379     right_on = ['m'])
380 hedge_df = pd.merge(hedge_df, sp500_cr, how='left', left_on=['m'],
381     right_on = ['m'])

```

```

372 hedge_df = pd.merge(hedge_df, hedge_cr, how='left', left_on=['m'],
373                      right_on = ['m'])
374 hedge_df.to_excel('excel/hedge_df.xlsx')
375
376 # Plot the dataframe
377 hedge_df.plot(x='m', y=['bmgbrown', 'bmrgreen', 'hedge', 'sp500'],
378                 kind='line', figsize=(28,14))
379 plt.title('Hedging - BMG Green, BMG Brown, S&P 500', fontsize = 30)
380 plt.ylabel('Cumulative Return', fontsize = 24)
381 plt.xlabel('Time (m)', fontsize = 24)
382 plt.xticks(size = 18)
383 plt.yticks(size = 18)
384 plt.legend(loc=2, prop={'size': 18})
385 # Add caption to below plot python
386 plt.savefig('plots/hedging.png')
387
388 # Additional Analysis 3: Perform PooledOLS for event study
389 # Paris Agreement (24192) and Trump Election (24203)
390 # Sort the dataframe into the needed sections
391 pre_paris_df = daily_all_df[daily_all_df['m'] < 24192]
392 print(pre_paris_df.tail())
393 post_paris_df = daily_all_df[daily_all_df['m'] >= 24192]
394 print(post_paris_df.head())
395 pre_trump_df = daily_all_df[daily_all_df['m'] < 24203]
396 print(pre_trump_df.tail())
397 post_trump_df = daily_all_df[daily_all_df['m'] >= 24203]
398 print(post_trump_df.head())
399
400 # Reindex dataframes for PooledOLS
401 pre_paris_df = pre_paris_df.set_index(['m', 'ind'])
402 post_paris_df = post_paris_df.set_index(['m', 'ind'])
403 pre_trump_df = pre_trump_df.set_index(['m', 'ind'])
404 post_trump_df = post_trump_df.set_index(['m', 'ind'])
405
406 # Events
407 events = [pre_paris_df, post_paris_df, pre_trump_df, post_trump_df]
408 names = ['Pre-Paris-Agreement-(before-Dec-2015)', 'Post-Paris-Agreement
409           -(Dec-2015-onwards)', 'Pre-Trump-Election-(before-Nov-2016)', 'Post-
410           Trump-Election-(Nov-2016-onwards)']
411 name_count = 0
412
413 # Runs PooledOLS for
414 for ev in events:
415     # Performs PooledOLS
416     endo = ev['eret']
417     exog = ev[['mktrf', 'smb', 'hml', 'umd', 'bmgbrown']]
418     exog = sm.add_constant(exog)
419     model = lp.PooledOLS(endo, exog).fit(cov_type='clustered',
420                                 cluster_entity=True)
421     print(model, file=open("event-study/" + names[name_count] + "pooledOLS
422                           .txt", "w"))
423     name_count = name_count + 1

```