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

```
# This assignment implements the statistical analysis required for the 705 co-integration/causality assignment

# Imports important python packages and data from data prcocessing
import numpy as np # arithmetic operations
import pandas as pd # data analysis package
import csv as csv # read and write csvs
import random as rd # random functionality
import saspy as sas # Use saspy functionality in python
import matplotlib.pyplot as plt # Use MatLab functionality for plotting
import seaborn as sb # Imports seaborn library for use
import wrds as wrds# Wharton Research Data Services API
import pydatastream as pds # Thomas Reuters Datastream API
import yfinance as yf # Yahoo Finance API
import datetime as dt # Manipulate datetime values
import statsmodels.api as sm # Create Stats functionalities
# import johansen as jh # Ability to implement Johansen test to test for co-integration
import linearmodels as lp # Ability to use PooledOLS
from sklearn.linear_model import LinearRegression
from stargazer.stargazer import Stargazer
import finance_byu as fin # Python Package for Fama-MacBeth Regressions
from statsmodels.regression.rolling import RollingOLS # Use factor loadings
from stargazer.stargazer import Stargazer
import sympy as sy # convert latex code
import scipy as sc # Scipy packages
import tabulate as tb # Create tables in python
import itertools as it # Find combinations of lists

# This section contains useful links for mean reversion, pairs-trading
# https://letianzj.github.io/mean-reversion.html
# https://letianzj.github.io/cointegration-pairs-trading.html
# https://en.wikipedia.org/wiki/Cointegration#Engle%E2%80%93Granger_two-step_method

# Defines the pairs trading function for the first part of the assignment
# Requires cross-sectional, time-series data of stock/bond/forex returns
def pairs_trading(data, asset_1, asset_2):
    """[summary]

    Args:
        data ([type]): [description]
        asset_1 ([type]): [description]
        asset_2 ([type]): [description]

    Returns:
        [type]: [description]
```

```
    """
    # Start date to produce plots
    # Produces time-series plots overlaying one security with another, regression residuals
    # Add produces a one-dimensional array of residuals
    # Test both configurations
    # Initialise tstat

    # Determines suitable combinations of security pairs for pairs trading.
    # Firstly, mplements the Cointegrated Augmented Dicker-Fuller (CADF) test to determine optimal
    # Hedge ratio by linear regression against the two stocks and then tests for stationarity
    # of the residuals. CADR is also known as Engle-Granger Two-Step Method
    tstat_coint = np.inf
    # Set up defaults
    independant = [asset_1,asset_2]
    dependant = [asset_2, asset_1]
    for i in range(len(independant)):
        x = data[independant[i]].values.reshape(-1,1)
        y = data[dependant[i]].values
        lm_model = LinearRegression(copy_X=True, fit_intercept=True, normalize=False).fit(x, y) # fit() expects 2D array
        # print('pamameters: %.7f, %.7f' %(lm_model.intercept_, lm_model.coef_))
        yfit = lm_model.coef_ * data[independant[i]] + lm_model.intercept_
        res = data[dependant[i]] - yfit
        [tstat, pvalue, num_lags, num_obs, crit_values, icbest] = sm.tsa.stattools.adfuller(res, maxlag = 1)
        # print('tstat',tstat)
        if tstat < tstat_coint:
            # Update critical values
            tstat_coint = tstat
            pvalue_coint = pvalue
            num_lags_coint = num_lags
            num_obs_coint = num_obs
            crit_values_coint = crit_values
            icbest_coint = icbest
            hedge_ratio= lm_model.coef_
            inte_coint = lm_model.intercept_
            x_name = independant[i]
            y_name = dependant[i]
            data['res'] = res

    # Check if significant co-integration exists
    if tstat_coint < -2.86: # Critical Value
        status = 'Yes'
        # Plots the residuals and prices separately
        # Plots the prices
        data.plot(x='Date', y=[x_name, y_name], kind='line')
        plt.title('Time-series of Price: Independant:' + x_name + ', Dependant:' + y_name)
        plt.ylabel('Price')
        plt.xlabel('Date')
```

```
92          plt.savefig('results/regressions/'+ x_name + '-'+ y_name +'-
    regression.png')
93          # Plots the residuals
94          data.plot(x='Date', y='res', kind='line')
95          plt.title('Time-series of Residuals: Independant:' + x_name + '
    ,Dependant:' + y_name)
96          plt.ylabel('Residual')
97          plt.xlabel('Date')
98          plt.savefig('results/residual/'+ x_name + '-'+ y_name +'-
    residuals.png')
99          # Plots the residuals
100
101         # Implements the Johansen test to mitigate accumulating errors
    in the two step process
102         # Find the Hedge Ratio and Tests for Co-integration at the same
     time (Can be extended to
103         # more than two stocks (Implement if there is time)!
104
105         # Calculate log prices and returns for trading strategies (
    Check when add more stocks to the mix)
106         # Prints proposed spreads
107         data[x_name+'-log-price'] = np.log(data[x_name])
108         data[y_name+'-log-price'] = np.log(data[y_name])
109         # Standard
110         data['trading-spread'] = data[y_name] - data[x_name]
111         data['trading-spread-mean'] = data['trading-spread'].rolling(
    window=20).mean()
112         data['trading-spread-std'] = data['trading-spread'].rolling(
    window=20).std()
113         # Log (spread = log(x) - nlog(b))
114         data['log-spread-price'] = data[y_name+'-log-price'] -
    hedge_ratio*data[x_name+'-log-price']
115         data['log-spread-price-mean'] = data['log-spread-price'].
    rolling(window=20).mean()
116         data['log-spread-price-std'] = data['log-spread-price'].rolling
    (window=20).std()
117         # Calculates the Bollinger Bands
118         # Sets scaler multipier
119         scaler = 2
120         # Calculates bands
121         # Log Bands
122         data['log-spread-price-upper'] = data['log-spread-price-mean']
    + (scaler*data['log-spread-price-std'])
123         data['log-spread-price-lower'] = data['log-spread-price-mean']
    - (scaler*data['log-spread-price-std'])
124         data['spread-price-upper'] = data['trading-spread-mean'] + (
    scaler*data['trading-spread-std'])
125         data['spread-price-lower'] = data['trading-spread-mean'] - (
    scaler*data['trading-spread-std'])
126
127         # Plot Log Price with Bollinger Bands
128         data.plot(x='Date', y=['log-spread-price','log-spread-price-
    mean','log-spread-price-upper','log-spread-price-lower'], kind='line
    ')
129         plt.title('20 Moving Average Log Spread with Bollinger Bands:'+
    x_name+'-'+y_name)
130         plt.ylabel('Spreads')
131         plt.xlabel('Date')
```

```
        plt.legend(loc=2)
        plt.savefig('results/logspreads/' +x_name+'-'+y_name+'.png')

        # Plot Price Spread with Bollinger Bands
         # Plot Log Price with Bollinger Bands
        # data.plot(x='Date', y=['trading-spread','trading-spread-mean','spread-price-upper','spread-price-lower'], kind='line')
        # plt.title('20 Moving Average Spread with Bollinger Bands:'+x_name+'-'+y_name)
        # plt.ylabel('Spreads')
        # plt.xlabel('Date')
        # plt.legend(loc=2)
        # plt.savefig('charts/spread-' +x_name+'-'+y_name+'.png')

        # Implements Bollinger Trading Strategy (Assumes only trading on the spread, not accounting for transactions costs)
        # Initialise size of trade (Assumes order size of 1000, no current positions
        initial_capital = 1000
        money_at_risk_percentage = 0.01
        cents_at_risk = 0.10
        # Equation to determine order size
        order_size = initial_capital * money_at_risk_percentage / cents_at_risk
        capital = initial_capital # Time zero
        hr = hedge_ratio
        x_name_current_size = 0
        y_name_current_size = 0

        # Set lagged variables, positions and capital
        data['lagged-'+x_name] = data[x_name].shift(1)
        data['lagged-'+y_name] = data[y_name].shift(1)
        data['lagged-log-spread-price'] = data['log-spread-price'].shift(1)
        data['lagged-log-spread-price-mean'] = data['log-spread-price-mean'].shift(1)
        data[x_name+'size'] = 0
        data[y_name+'size'] = 0
        data[x_name+'order_size'] = 0
        data[y_name+'order_size'] = 0
        data['capital'] = 0
         # margin = revenue - costs
        data['margin'] = 0

        # Implements statistical arbitrage trading strategies
        for index, row in data.iterrows():
            # Hit Upper Band, Short the Spread
            if (data.at[index,'log-spread-price'] > data.at[index,'log-spread-price-upper']) and (x_name_current_size >= 0):
                capital = capital - int(hr*order_size)*data.at[index,x_name] + int(order_size)*data.at[index,y_name]
                # x_name
                data.at[index,x_name+'order_size'] = - int(hr*order_size) - x_name_current_size
                x_name_current_size = - int(hr*order_size)
                # y_name
                data.at[index,y_name+'order_size'] = order_size - y_name_current_size
```

```
                y_name_current_size = order_size
                # margin = revenue - costs
                data.at[index,'margin'] = -1*data.at[index,x_name+'
    order_size']*data.at[index,x_name] - 1*data.at[index,y_name+'
    order_size']*data.at[index,y_name]

            # Hit Lower Band, Long the Spread
            elif (data.at[index,'log-spread-price'] < data.at[index,'
    log-spread-price-lower']) and (x_name_current_size <= 0):
                capital = capital + int(hr*order_size)*data.at[index,
    x_name] - int(order_size)*data.at[index,y_name]
                 # x_name
                data.at[index,x_name+'order_size'] = int(hr*order_size)
     - x_name_current_size
                x_name_current_size = int(hr*order_size)
                # y_name
                data.at[index,y_name+'order_size'] = - order_size -
    y_name_current_size
                y_name_current_size = - order_size
                # margin = revenue - costs
                data.at[index,'margin'] = -1*data.at[index,x_name+'
    order_size']*data.at[index,x_name] - 1*data.at[index,y_name+'
    order_size']*data.at[index,y_name]

            # Spread crosses from below average, flat long position
            elif (data.at[index,'log-spread-price'] > data.at[index,'
    log-spread-price-mean']) and (data.at[index,'lagged-log-spread-price
    '] < data.at[index,'lagged-log-spread-price-mean']) and (
    x_name_current_size > 0):
                capital = capital - int(x_name_current_size)*data.at[
    index,x_name] + int(y_name_current_size)*data.at[index,y_name]
                 # x_name
                data.at[index,x_name+'order_size'] = -
    x_name_current_size
                x_name_current_size = 0
                # y_name
                data.at[index,y_name+'order_size'] = -
    y_name_current_size
                y_name_current_size = 0
                # margin = revenue - costs
                data.at[index,'margin'] = -1*data.at[index,x_name+'
    order_size']*data.at[index,x_name] - 1*data.at[index,y_name+'
    order_size']*data.at[index,y_name]

            # Spread crosses from above average, flat/cover short
    position
            elif (data.at[index,'log-spread-price'] < data.at[index,'
    log-spread-price-mean']) and (data.at[index,'lagged-log-spread-price
    '] > data.at[index,'lagged-log-spread-price-mean']) and (
    x_name_current_size < 0):
                capital = capital + int(x_name_current_size)*data.at[
    index,x_name] - int(y_name_current_size)*data.at[index,y_name]
                # x_name
                data.at[index,x_name+'order_size'] = -
    x_name_current_size
                x_name_current_size = 0
                # y_name
```

```
                data.at[index,y_name+'order_size'] = -
y_name_current_size
                y_name_current_size = 0
                # margin = revenue - costs
                data.at[index,'margin'] = -1*data.at[index,x_name+'
order_size']*data.at[index,x_name] - 1*data.at[index,y_name+'
order_size']*data.at[index,y_name]


            # # Sets the capital level depending on the position
            # data.set_value(index, 'capital',capital)
            data.at[index, x_name+'size'] = x_name_current_size
            data.at[index, y_name+'size'] = y_name_current_size

        # Determine

        # Calculates the margin generated from holding all the
realative positions
        # This is the price of the position multiplied by the position
size held
        data['gain'] = data[x_name+'size']*data[x_name] + data[y_name+'
size']*data[y_name]

        # Calculate cumulative-gains returns dataframe
        for index, row in data.iterrows():
            if index == 0:
               data.at[index,'accum-gain'] = data.at[index, 'gain']
            if index > 0:
                data.at[index,'accum-gain'] = data.at[index,'gain'] +
data.at[index-1,'accum-gain']

        # Calculate statistical arbitrage cumulative return by dividing
 accumulative gain by initial cpatial

         # Plots the gains
        # data.plot(x='Date', y=['gain'], kind='line', figsize=(28,18))
        # plt.title('Gains on Trades-:'+x_name+'-'+y_name)
        # plt.ylabel('Gain', fontsize = 30)
        # plt.xlabel('Date', fontsize = 30)
        # plt.legend(loc=2, prop={'size': 30})
        # plt.xticks(size = 18)
        # plt.yticks(size = 18)
        # plt.savefig('charts/gain-' +x_name+'-'+y_name+'.png')

        # Plots the accumukated margin
        data.plot(x='Date', y=['accum-gain'], kind='line')
        plt.title('Accumulated Gain ($):'+x_name+'-'+y_name)
        plt.ylabel('Gain')
        plt.xlabel('Date')
        plt.savefig('results/gains/' +x_name+'-'+y_name+'.png')

        # Plots the order sizes
        # data.plot(x='Date', y=[x_name+'order_size',y_name+'order_size
'], kind='line', figsize=(28,18))
        # plt.title('Order Sizes-:'+x_name+'-'+y_name, fontsize = 30)
        # plt.ylabel('Order Size', fontsize = 30)
        # plt.xlabel('Date', fontsize = 30)
        # 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', figsize=(28,18))
269         # plt.title('Number of Positions-:'+x_name+'-'+y_name)
270         # plt.ylabel('Positions', fontsize = 30)
271         # plt.xlabel('Date', fontsize = 30)
272         # plt.legend(loc=2, prop={'size': 30})
273         # plt.xticks(size = 18)
274         # plt.yticks(size = 18)
275         # plt.savefig('charts/positions-' +x_name+'-'+y_name+'.png')
276 
277         # Calculate buy and hold return over forecast periods
278         data = data.sort_values(by='Date')
279         data['bhr'] = (data['trading-spread']/data['trading-spread'].
    shift(1)) - 1
280         # Calculate Statistical Arbitrage Returns
281         data['bbr'] = data['accum-gain']
282         # data['bbr'] = (data['accum-gain']/data['accum-gain'].shift(1)
    ) - 1
283         data = data.dropna()
284         data.reset_index(inplace = True, drop = True)
285 
286         # Calculates Buy and Hold Cuumulative Returns
287         for index, row in data.iterrows():
288             if index == 0:
289                 # Buy and Hold Strategy
290                 bhcr = data.at[index, 'bhr']
291                 data.at[index,'bhcr'] = bhcr
292                 # Statistical Arbitrage
293                 # bbcr = data.at[index, 'bbr']
294                 # data.at[index,'bbcr'] = bbcr
295             if index > 0:
296                 # Buys and hold strategy
297                 bhcr = ((1+data.at[index,'bhr'])*(1+data.at[index-1,'
    bhcr']))-1
298                 data.at[index,'bhcr'] = bhcr
299                 # Statistical Arbitrage
300                 # bbcr = ((1+data.at[index,'bbr'])*(1+data.at[index-1,'
    bbcr']))-1
301                 # data.at[index,'bbcr'] = bbcr
302 
303         # Return last values cumulative return
304         bhr = data.at[data.index[-1],'bhcr']
305         # Return
306         cg = data.at[data.index[-1],'accum-gain']
307 
308         # Calculate cumulative return from statistical arbitrage
    strategy
309         # Plots the accumukated margin
310         data.plot(x='Date', y=['bhcr'], kind='line')
311         plt.title('Cumulative Buy & Hold Returns ($):'+x_name+'-'+
    y_name)
312         plt.ylabel('Buy & Hold Returns')
313         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 ($):'+x_name+'-'+y_name)
318         plt.ylabel('Buy and Hold Returns')
319         plt.xlabel('Date')
320         plt.savefig('results/returns/bollinger/'+x_name+'-'+y_name+'.
    png')
321 
322         # Calculate cumulative return from statistical arbitrage
    strategy
323         # Plots the accumukated margin
324         # data.plot(x='Date', y=['bhr','bbr'], kind='line', figsize
    =(28,18))
325         # plt.title('Returns ($):'+x_name+'-'+y_name)
326         # plt.ylabel('Returns', fontsize = 30)
327         # plt.xlabel('Date', fontsize = 30)
328         # plt.legend(loc=2, prop={'size': 30})
329         # plt.xticks(size = 18)
330         # plt.yticks(size = 18)
331         # plt.savefig('charts/returns-' +x_name+'-'+y_name+'.png')
332 
333         # Calculates some average values for tranquil and crisis period
334         # Tranquil period
335         tran_start = '1/9/19'
336         tran_end = '28/02/20'
337         cris_start = '1/3/20'
338         cris_end = '31/8/20'
339         # Get tranquil dataframe
340         tranquil = data[data["Date"] > tran_start]
341         tranquil = tranquil[tranquil["Date"] <= tran_end]
342         # Get crisis dataframe
343         crisis = data[data["Date"] > cris_start]
344         crisis = crisis[crisis["Date"] <= cris_end]
345 
346         # Find the averages for those periods
347         tran_average_buy_hold = tranquil['bhr'].mean()
348         tran_average_gain = tranquil['gain'].mean()
349         cris_average_buy_hold = crisis['bhr'].mean()
350         cris_average_gain = crisis['gain'].mean()
351 
352 
353         # Returns variables from the function
354         return x_name, y_name, tstat_coint, hedge_ratio, bhr, cg,
    order_size, status, tran_average_buy_hold, tran_average_gain,
    cris_average_buy_hold, cris_average_gain
355 
356         # Implements the trading strategy with both bands
357     else:
358         print("Co-integration between pairs does not exist")
359         status = 'No'
360         hedge_ratio = np.nan
361         bhr = np.nan
362         cg = np.nan
363         order_size = np.nan
364         tran_average_buy_hold = np.nan
365         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,
    order_size, status, tran_average_buy_hold, tran_average_gain,
    cris_average_buy_hold, cris_average_gain
369
370
371
372 # Defines the benchmarking function for the second part of the
    assignment (Placeholder)
373 def benchmarking(self):
374     """[summary]
375
376     Returns:
377         [type]: [description]
378     """
379     return self
380 # Defines the financial co-integration and causality function (
    Placeholder)
381 def contagion_causality(self):
382     """[summary]
383
384     Returns:
385         [type]: [description]
386     """
387     return self
388
389 # Downloads financial data from yahoo finance (ExxonMobil and Chevron
    to Test Co-Integration Example)
390 # This is to be replaced with the data outputs
391 # https://towardsdatascience.com/a-comprehensive-guide-to-downloading-
    stock-prices-in-python-2cd93ff821d4
392 # Test case setup
393 start_date = '2000-01-01'
394 end_date = '2019-12-31'
395 prices_1 = 'EWA'
396 prices_2 = 'EWC'
397
398 asset_1 = yf.download(prices_1, start = start_date, end = end_date,
    progress = False)
399 asset_1.to_csv('data/'+prices_1+'.csv')
400
401 asset_2 = yf.download(prices_2, start = start_date, end = end_date,
    progress = False)
402 asset_2.to_csv('data/'+prices_2+'.csv')
403
404 # Import data
405 asset_1_df = pd.read_csv('data/'+prices_1+'.csv')
406 asset_2_df =  pd.read_csv('data/'+prices_2+'.csv')
407
408 # Calculate the returns
409 asset_1_df[prices_1 +'-ret-(%)'] = (asset_1_df['Adj Close']/asset_1_df[
    'Adj Close'].shift(1))-1
410 asset_1_df.rename(columns= {'Adj Close':prices_1}, inplace = True)
411 asset_1_df = asset_1_df.dropna(axis=0)
412
413 asset_2_df[prices_2 +'-ret-(%)'] = (asset_2_df['Adj Close']/asset_2_df[
    'Adj Close'].shift(1))-1
```

```
asset_2_df.rename(columns= {'Adj Close':prices_2}, inplace = True)
asset_2_df = asset_2_df.dropna(axis=0)

# Merge the data into one dataframe
data_df = pd.merge(asset_1_df[['Date',prices_1,prices_1 +'-ret-(%)']].
    copy(), asset_2_df[['Date',prices_2,prices_2 +'-ret-(%)']].copy(),
    how='left', left_on=['Date'], right_on = ['Date'])
data_df = data_df.dropna(axis=0)
data = data_df[['Date',prices_1,prices_2]].copy()

# Calls the pairs trading function
x_name, y_name, tstat_coint, hedge_ratio, bhr, cg, order_size, status,
    tran_average_buy_hold, tran_average_gain, cris_average_buy_hold,
    cris_average_gain = pairs_trading(data,prices_1,prices_2)
# Establishes
test_results_table = pd.DataFrame(columns=['Variable (x)', 'Variable (y
    )','tstat','Hedge Ratio', 'Buy & Hold Cumulative Return', '
    Cumulative Gain (Bollinger Bands)', 'Order Size', 'Co-integration'])
# Creates new row to add to empty dataframe
new_row = {'Variable (x)':x_name, 'Variable (y)': y_name,'tstat':
    tstat_coint,'Hedge Ratio': hedge_ratio, 'Buy & Hold Cumulative
    Return': bhr, 'Cumulative Gain (Bollinger Bands)':cg, 'Order Size':
    order_size, 'Co-integration': status}
test_results_table = test_results_table .append(new_row, ignore_index =
     True)
# Rank via tStat (Indicates stength of mean reversion
test_results_table.sort_values(by = 'tstat')
test_results_table.to_excel('results/test_results_table.xlsx')

# Conduct pairs trading analysis for list of resources (Steel Stocks)
# Loads in data for pairs trading analysis
resources_data = pd.read_excel('data/data.xlsx')

# Get the pricing information for the data (List of names)
assets = list(resources_data.columns.values)
assets = assets[1:-1]

# Creates combinations for pairs analysis
pair_order_list = list(it.combinations(assets,2))

# Cleans data of values and re_index
resources_data = resources_data.dropna(axis = 0)
resources_data.reset_index(inplace = True, drop = True)

# Initialises final resources table
final_results_table = pd.DataFrame(columns=['Variable (x)', 'Variable (
    y)','tstat','Hedge Ratio', 'Buy & Hold Cumulative Return', '
    Cumulative Gain (Bollinger Bands)', 'Order Size', 'Co-integration',
    'tran_average_buy_hold', 'tran_average_gain', 'cris_average_buy_hold
    ', 'cris_average_gain'])
for pair in pair_order_list:
    try:
        x_name, y_name, tstat_coint, hedge_ratio, bhr, cg, order_size,
    status, tran_average_buy_hold, tran_average_gain,
    cris_average_buy_hold, cris_average_gain = pairs_trading(
    resources_data,pair[0],pair[1])
        new_row = {'Variable (x)':x_name, 'Variable (y)': y_name,'tstat
    ': 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 stength of mean reversion
460 final_results_table.sort_values(by = 'tstat')
461 final_results_table.to_excel('results/rank/final_results_table.xlsx')
```