```
C:\Users\Anis\MyArkives\Arki_Schools\Arki_UNIL_HEC\MScF_S2\Empirical_Methods_Finance\Exercises_S2C3\EMF_Project2\venv\Scripts\python.exe "C:/Program Files/JetBrains/PyCharm 2022.3.2/plugins/python/helpers/pydev/pydevconsole.py" --mode=client --host=127.0.0.1 --port=63906

import sys; print('Python %s on %s' % (sys.version, sys.platform))
sys.path.extend(['C:\\Users\\Anis\\MyArkives\\Arki_Schools\\Arki_UNIL_HEC\\MScF_S2\\Empirical_Methods_Finance\\Exercises_S2C3\\EMF_Project2'])

PyDev console: starting.

Python 3.10.9 | packaged by Anaconda, Inc. | (main, Mar  1 2023, 18:18:15) [MSC v.1916 64 bit (AMD64)] on win32
>>> # Import packages
... from pathlib import Path
... from scipy.stats import norm
... from scripts.parameters import paths
... from sklearn.linear_model import LinearRegression
... import matplotlib.dates as mdates
... import matplotlib.pyplot as plt
... import numpy as np
... import pandas as pd
... import scripts.functions as fn
... import seaborn as sns
... import statsmodels.api as sm
... import warnings
...
... # Packages for testing functions (DELETE LATER)
... from statsmodels.tsa.ar_model import AutoReg
... from statsmodels.tsa.stattools import adfuller
... from tqdm import tqdm
...
... # Suppress warnings
... warnings.filterwarnings('ignore')
...
...
... # %%
... # ************************************************
... # *** QUESTION 1: Stationarity                  ***
... # ************************************************
...
```

```
37 ... # Importing dataset (commodities)
38 ... df_data = fn.read_data(file_path=Path.joinpath(paths.
   get('data'), 'commodities.csv'))
39 ...
40 ... # Taking logarithm of the adjusted close price
41 ... l_adj_close_price = ['ZC Adj Close', 'ZW Adj Close', '
   ZS Adj Close', 'KC Adj Close', 'CC Adj Close']
42 ... df_data = df_data[l_adj_close_price]
43 ... df_data_ln = fn.log_transform_cols(df_data,
   l_adj_close_price)
44 ...
45 ...
46 ... # ************************************************
47 ... # *** QUESTION 1.1: Critical Values              ***
48 ... # ************************************************
49 ...
50 ... # Initialisation
51 ... T = len(df_data_ln.index)
52 ... N = 10000
53 ... column = 'ZC Adj Close'
54 ...
55 ... # First simulation
56 ... simulation_1 = fn.critical_value(df_data_ln, column, T
   , N)
57 ... ar_params_1 = simulation_1[1]
58 ...
59 ... # *** Question 1.4 ***
60 ... # Plot the histogram of the Test-Statistic
61 ... fig, ax = plt.subplots(figsize=(15, 10))
62 ... ax.hist(ar_params_1['DF_TS'], bins=50, edgecolor='
   black')
63 ... ax.set_title('Test Statistic Distribution for N: ' +
   str(N))
64 ... plt.xlabel('Test Statistic')
65 ... plt.ylabel('Frequency')
66 ... plt.show()
67 ... fig.savefig(Path.joinpath(paths.get('output'), 'Q1.4_T
   -Stat Distribution.png'))
68 ... plt.close()
69 ...
70 ... # *** Question 1.5 ***
71 ... # Compute the critical values of the DF test
72 ... critical_values_1 = simulation_1[0]
73 ... critical_values_1.to_latex(Path.joinpath(paths.get('
```

```
 73 output'), 'Q1.5_Critical_Values.tex'), float_format='%.2f
    ')
 74 ...
 75 ... # Plot the histogram of the Test-Statistic and
    Critical Values
 76 ... fig, ax = plt.subplots(figsize=(15, 10))
 77 ... ax.hist(ar_params_1['DF_TS'], bins=50, edgecolor='
    black')
 78 ... ax.set_title('Test Statistic Distribution for N: ' +
    str(N))
 79 ... plt.xlabel('Test Statistic')
 80 ... plt.ylabel('Frequency')
 81 ... plt.axvline(x=critical_values_1.loc[0.01], color='r
    ', label='CV 1%')
 82 ... plt.axvline(x=critical_values_1.loc[0.05], color='y
    ', label='CV 5%')
 83 ... plt.axvline(x=critical_values_1.loc[0.10], color='g
    ', label='CV 10%')
 84 ... plt.legend()
 85 ... plt.show()
 86 ... fig.savefig(Path.joinpath(paths.get('output'), 'Q1.
    5_T-Stat Distribution_CV.png'))
 87 ... plt.close()
 88 ...
 89 ... # *** Question 1.6 ***
 90 ... # Re-computing the simulation for T=500
 91 ... simulation_2 = fn.critical_value(df_data_ln, column,
    T=500, N=N)
 92 ... critical_values_2 = simulation_2[0]
 93 ...
 94 ...
 95 ... # ************************************************
 96 ... # *** QUESTION 1.2: Testing Non-Stationarity     ***
 97 ... # ************************************************
 98 ...
 99 ... # *** Question 1.7 ***
100 ... # Computing the DF Test
101 ... DF_Test = pd.DataFrame(index=['DF_TS', 'CV 1%', 'CV 5
    %', 'CV 10%', 'Reject H0 1%', 'Reject H0 5%', 'Reject H0
    10%',
102 ...                                 'P_Value'], columns=
    l_adj_close_price)
103 ...
104 ... for col in l_adj_close_price:
```

```
105 ...     t_stat_data = fn.reg(df_data_ln, col, lag=1)
106 ...     DF_Test.loc['DF_TS'][col] = t_stat_data
107 ...     DF_Test.loc['CV 1%'][col] = critical_values_1.loc
    [0.01]
108 ...     DF_Test.loc['CV 5%'][col] = critical_values_1.loc
    [0.05]
109 ...     DF_Test.loc['CV 10%'][col] = critical_values_1.
    loc[0.10]
110 ...     DF_Test.loc['Reject H0 1%'][col] = np.abs(DF_Test
    .loc['DF_TS'][col]) > np.abs(DF_Test.loc['CV 1%'][col])
111 ...     DF_Test.loc['Reject H0 5%'][col] = np.abs(DF_Test
    .loc['DF_TS'][col]) > np.abs(DF_Test.loc['CV 5%'][col])
112 ...     DF_Test.loc['Reject H0 10%'][col] = np.abs(
    DF_Test.loc['DF_TS'][col]) > np.abs(DF_Test.loc['CV 10
    %'][col])
113 ...     DF_Test.loc['P_Value'][col] = fn.get_pvalue(
    ar_params_1['DF_TS'], DF_Test.loc['DF_TS'][col])
114 ...
115 ... DF_Test.columns = ['Corn', 'Wheat', 'Soybean', '
    Coffee', 'Cacao']
116 ... DF_Test = fn.format_float(DF_Test)
117 ... DF_Test.to_latex(Path.joinpath(paths.get('output'), '
    Q1.7_DF_Test.tex'))
118 ...
119 ...
120 ... # %%
121 ... # ************************************************
122 ... # *** QUESTION 2: Cointegration                  ***
123 ... # ************************************************
124 ...
125 ...
126 ... # ************************************************
127 ... # *** QUESTION 2.1: Critical Values              ***
128 ... # ************************************************
129 ...
130 ... # Simulate distribution
131 ... t_stat_coint = fn.simulate_coint_cv(T, N=N)
132 ... df_ts_coint = pd.DataFrame(data=t_stat_coint, columns
    =['DF_TS'])
133 ...
134 ... # Compute critical values
135 ... cv_coint = df_ts_coint['DF_TS'].quantile([0.01, 0.05
    , 0.1])
136 ... cv_coint = cv_coint.rename('Critical Value')
```

```
137 ... # Save as Latex
138 ... cv_coint.to_latex(Path.joinpath(paths.get('output
    '), 'Q2.1_Critical_Values_Coint.tex'), float_format='%.2f
    ')
139 ...
140 ... # Plotting Histogram of critical values.
141 ... fig, ax = plt.subplots(figsize=(15, 10))
142 ... ax.hist(df_ts_coint['DF_TS'], bins=50, edgecolor='
    black')
143 ... ax.set_title('Test Statistic Distribution for N: ' +
    str(N))
144 ... plt.xlabel('Test Statistic')
145 ... plt.ylabel('Frequency')
146 ... plt.axvline(x=cv_coint.loc[0.01], color='r', label='
    CV 1%')
147 ... plt.axvline(x=cv_coint.loc[0.05], color='y', label='
    CV 5%')
148 ... plt.axvline(x=cv_coint.loc[0.10], color='g', label='
    CV 10%')
149 ... plt.legend()
150 ... plt.show()
151 ... fig.savefig(Path.joinpath(paths.get('output'), 'Q2.
    1_T-Stat_Distribution_Coint.png'))
152 ... plt.close()
153 ...
154 ...
155 ... # ************************************************
156 ... # *** QUESTION 2.2: Testing for Cointegration    ***
157 ... # ************************************************
158 ...
159 ... # Cointegration results DataFrame, needed to
    construct another cointegration test statistics DataFrame
160 ... df_coint = fn.cointegration(df_data_ln, column_names
    =['Corn', 'Wheat', 'Soybean', 'Coffee', 'Cacao'], permut=
    True)
161 ...
162 ... # *** Question 2.2 ***
163 ... coint_test = pd.DataFrame(index=df_coint.index,
    columns=['DF_TS', 'CV_1%', 'CV_5%', 'CV_10%', 'P_Value
    ', 'Reject H0 1%', 'Reject H0 5%', 'Reject H0 10%'])
164 ...
165 ... for index in df_coint.index:
166 ...     coint_test.loc[index]['DF_TS'] = df_coint.loc[
    index]['DF_TS']
```

```
167 ...     coint_test.loc[index]['CV_1%'] = cv_coint.loc[0.
    01]
168 ...     coint_test.loc[index]['CV_5%'] = cv_coint.loc[0.
    05]
169 ...     coint_test.loc[index]['CV_10%'] = cv_coint.loc[0.
    1]
170 ...     coint_test.loc[index]['P_Value'] = fn.get_pvalue(
    t_stat_coint, df_coint.loc[index]['DF_TS'])
171 ...     coint_test.loc[index]['Reject H0 1%'] = np.abs(
    df_coint.loc[index]['DF_TS']) > np.abs(cv_coint.loc[0.01
    ])
172 ...     coint_test.loc[index]['Reject H0 5%'] = np.abs(
    df_coint.loc[index]['DF_TS']) > np.abs(cv_coint.loc[0.05
    ])
173 ...     coint_test.loc[index]['Reject H0 10%'] = np.abs(
    df_coint.loc[index]['DF_TS']) > np.abs(cv_coint.loc[0.1])
174 ...
175 ... coint_test_out = fn.format_float(coint_test)
176 ... coint_test_out.to_latex(Path.joinpath(paths.get('
    output'), 'Q2.2_Coint_Test_Results.tex'))
177 ... small_coint_test = coint_test_out[(coint_test_out['
    Reject H0 1%'] == True) | (coint_test_out['Reject H0 5
    %'] == True) | (coint_test_out['Reject H0 10%'] == True)]
178 ... small_coint_test.to_latex(Path.joinpath(paths.get('
    output'), 'Q2.2_Small_Coint_Test_Results.tex'))
179 ...
180 ... # *** Question 2.3 ***
181 ... df_coint_out = fn.format_float(df_coint[['Alpha', '
    Beta']])
182 ... df_coint_out.to_latex(Path.joinpath(paths.get('output
    '), 'Q2.3_A_B_Values.tex'))
183 ...
184 ... # *** Question 2.5 ***
185 ... pA = df_data_ln['ZW Adj Close']
186 ... alpha = df_coint.loc['Wheat-Corn']['Alpha']
187 ... beta = df_coint.loc['Wheat-Corn']['Beta']
188 ... pB = df_data_ln['ZC Adj Close']
189 ... comb = alpha + beta * pB
190 ...
191 ... # Plot PA and Linear combination
192 ... sns.set(context='paper', style='ticks', font_scale=1.
    0)
193 ... fig = plt.figure(figsize=(12, 7), dpi=600)
194 ... ax = fig.add_subplot()
```

```
195 ... ax.grid(False)
196 ... ax.set_title(label='Wheat-Corn Pair', size=28)
197 ... # Items
198 ... sns.lineplot(x=pd.to_datetime(pA.index), y=pA, label
    ='Wheat', color='sienna', lw=3)
199 ... sns.lineplot(x=pd.to_datetime(comb.index), y=comb,
    label='Alpha + Beta * PriceB (Corn)', color='gold', lw=3)
200 ... # X-axis settings
201 ... date_locator = mdates.YearLocator()
202 ... date_formatter = mdates.DateFormatter('%Y')
203 ... ax.tick_params(axis='x', labelrotation=0, labelsize=
    18)
204 ... ax.xaxis.set_major_locator(date_locator)
205 ... ax.xaxis.set_major_formatter(date_formatter)
206 ... ax.set_xlabel(xlabel='')
207 ... # Y-axis settings
208 ... ax.set_yticklabels(labels=['{:.1f}'.format(y) for y
    in ax.get_yticks()], size=18)
209 ... ax.set_ylabel(ylabel='Log Price', size=20)
210 ... # Legend settings
211 ... ax.legend(loc='upper left', fontsize=16)
212 ... # Show and save
213 ... plt.show()
214 ... fig.savefig(Path.joinpath(paths.get('output'), 'Q2.
    5_WC_Pair_Plot.png'))
215 ... plt.close()
216 ...
217 ...
218 ... # %%
219 ... # ************************************************
220 ... # *** QUESTION 3: Pair Trading                  ***
221 ... # ************************************************
222 ...
223 ... # Importing dataset (commodities)
224 ... df_data = fn.read_data(file_path=Path.joinpath(paths.
    get('data'), 'commodities.csv'))
225 ... l_adj_close_price = ['ZC Adj Close', 'ZW Adj Close
    ', 'ZS Adj Close', 'KC Adj Close', 'CC Adj Close']
226 ... df_data = df_data[l_adj_close_price]
227 ...
228 ... # Simulating distribution (cointegration)
229 ... t_stat_coint_500 = fn.simulate_coint_cv(T=500, N=
    10000)
230 ... df_ts_coint_500 = pd.DataFrame(data=t_stat_coint_500
```

```
230 , columns=['DF_TS'])
231 ...
232 ...
233 ... # ************************************************
234 ... # *** QUESTION 3.1: Trading Signal               ***
235 ... # ************************************************
236 ...
237 ... # Best pair: A=Wheat (ZW Adj Close), B=Corn (ZC Adj
    Close) ==> reg Corn (X) on Wheat (y)
238 ... # Concept: we will use spread to create trading
    signals
239 ...
240 ... # *** Question 3.2 ***
241 ... # Compute spread
242 ... X = df_data[['ZC Adj Close']]
243 ... y = df_data['ZW Adj Close']
244 ... lr_model = LinearRegression()
245 ... lr_model.fit(X, y)
246 ... s_spread = y - lr_model.predict(X)
247 ... s_spread = s_spread / s_spread.std(ddof=0)
248 ...
249 ... # Plot spread
250 ... sns.set(context='paper', style='ticks', font_scale=1.
    0)
251 ... fig = plt.figure(figsize=(12, 7), dpi=600)
252 ... ax = fig.add_subplot()
253 ... ax.grid(False)
254 ... ax.set_title(label='Wheat-Corn Spread', size=28)
255 ... # Items
256 ... ax.axhline(y=0, color='black', ls='--', lw=1)
257 ... ax.axhline(y=1.5, color='black', ls='--', lw=1)
258 ... ax.axhline(y=-1.5, color='black', ls='--', lw=1)
259 ... sns.lineplot(x=pd.to_datetime(s_spread.index), y=
    s_spread, label='Spread', color='red', lw=3)
260 ... # X-axis settings
261 ... date_locator = mdates.YearLocator()
262 ... date_formatter = mdates.DateFormatter('%Y')
263 ... ax.tick_params(axis='x', labelrotation=0, labelsize=
    18)
264 ... ax.xaxis.set_major_locator(date_locator)
265 ... ax.xaxis.set_major_formatter(date_formatter)
266 ... ax.set_xlabel(xlabel='')
267 ... # Y-axis settings
268 ... ax.set_yticklabels(labels=['{:.1f}'.format(y) for y
```

```
268 in ax.get_yticks()], size=18)
269 ... ax.set_ylabel(ylabel='')
270 ... # Legend settings
271 ... ax.legend(loc='upper left', fontsize=16)
272 ... # Show and save
273 ... plt.show()
274 ... fig.savefig(Path.joinpath(paths.get('output'), 'Q3.
    2_Spread.png'))
275 ... plt.close()
276 ...
277 ... # *** Question 3.3 ***
278 ... # Compute autocorrelogram of spread
279 ... df_autocorrelogram = fn.tab_autocorrelogram(s_data=
    s_spread, alpha=0.05, max_lags=10)
280 ...
281 ... # Plot autocorrelogram of spread
282 ... sns.set(context='paper', style='ticks', font_scale=1.
    0)
283 ... fig = plt.figure(figsize=(12, 7), dpi=600)
284 ... ax = fig.add_subplot()
285 ... ax.grid(False)
286 ... ax.set_title(label='Autocorrelogram Spread ({})'.
    format(df_autocorrelogram.columns[-1]), size=28)
287 ... # Items
288 ... s_autocorr = df_autocorrelogram['Autocorrelation']
289 ... s_ci_lower = pd.Series([df_autocorrelogram.iloc[:, -1
    ][i][0] for i in df_autocorrelogram.index], index=
    df_autocorrelogram.index)
290 ... s_ci_upper = pd.Series([df_autocorrelogram.iloc[:, -1
    ][i][1] for i in df_autocorrelogram.index], index=
    df_autocorrelogram.index)
291 ... ax.axhline(y=0, color='red', lw=3)
292 ... sns.lineplot(x=df_autocorrelogram.index, y=s_autocorr
    , color='black', lw=3)
293 ... sns.lineplot(x=df_autocorrelogram.index, y=s_ci_lower
    , color='blue', lw=3)
294 ... sns.lineplot(x=df_autocorrelogram.index, y=s_ci_upper
    , color='blue', lw=3)
295 ... # X-axis settings
296 ... ax.set_xticks(np.arange(df_autocorrelogram.index[0],
    df_autocorrelogram.index[-1]+1, 1))
297 ... ax.set_xlim(df_autocorrelogram.index[0],
    df_autocorrelogram.index[-1])
298 ... ax.set_xticklabels(labels=['{:.0f}'.format(x) for x
```

```
298 in ax.get_xticks()], size=18)
299 ... ax.set_xlabel(xlabel='Lags', size=20)
300 ... # Y-axis settings
301 ... ax.set_yticklabels(labels=['{:.1f}'.format(y) for y
    in ax.get_yticks()], size=18)
302 ... ax.set_ylabel(ylabel='k-Lags Autocorrelation', size=
    20)
303 ... # Show and save
304 ... plt.show()
305 ... fig.savefig(Path.joinpath(paths.get('output'), 'Q3.
    3_Autocorrelogram_Spread.png'))
306 ... plt.close()
307 ...
308 ... # Ljung-Box test with p=10 lags
309 ... fn.Ljung_Box_test(s_data=s_spread)
310 ...
311 ...
312 ... # ***************************************************
313 ... # *** QUESTION 3.2: In-Sample Pair Trading       ***
314 ... # ***************************************************
315 ...
316 ...
317 ... # ***************************************************
318 ... # *** QUESTION 3.2.1: Direct Strategy            ***
319 ... # ***************************************************
320 ...
321 ... # *** Question 3.4 ***
322 ... # Trading table
323 ... df_PT_insample1 = fn.tab_PT_insample(df_data=df_data
    , A='ZW Adj Close', B='ZC Adj Close', W=1000, L=2,
    in_level=1.5, stop_level=None)
324 ...
325 ... # Report
326 ... print('\nPair Trading Report (IS, W=1000, L=2, z_in=1
    .5, z_stop=None)')
327 ... fn.print_PT_report(df_PT=df_PT_insample1)
328 ...
329 ... # Plot wealth
330 ... sns.set(context='paper', style='ticks', font_scale=1.
    0)
331 ... fig = plt.figure(figsize=(12, 7), dpi=600)
332 ... ax = fig.add_subplot()
333 ... ax.grid(False)
334 ... ax.set_title(label='Evolution of Wealth (IS, L=2)',
```

```
size=28)
... # Items
... ax.axhline(y=df_PT_insample1.iloc[0, df_PT_insample1.columns.get_loc('Equity')], color='black', ls='--', lw=1)
... sns.lineplot(x=pd.to_datetime(df_PT_insample1.index), y=df_PT_insample1['Equity'], label='Wealth', color='blue', lw=3)
... # X-axis settings
... date_locator = mdates.YearLocator()
... date_formatter = mdates.DateFormatter('%Y')
... ax.tick_params(axis='x', labelrotation=0, labelsize=18)
... ax.xaxis.set_major_locator(date_locator)
... ax.xaxis.set_major_formatter(date_formatter)
... ax.set_xlabel(xlabel='')
... # Y-axis settings
... ax.set_yticklabels(labels=['{:.0f}'.format(y) for y in ax.get_yticks()], size=18)
... ax.set_ylabel(ylabel='')
... # Legend settings
... ax.legend(loc='upper left', fontsize=16)
... # Show and save
... plt.show()
... fig.savefig(Path.joinpath(paths.get('output'), 'Q3.4_Evolution_Wealth.png'))
... plt.close()
...
... # TODO: Plot positions (???)
...
... # Plot leverage
... sns.set(context='paper', style='ticks', font_scale=1.0)
... fig = plt.figure(figsize=(12, 7), dpi=600)
... ax = fig.add_subplot()
... ax.grid(False)
... ax.set_title(label='Evolution of Leverage (IS, L=2)', size=28)
... # Items
... ax.axhline(y=2, color='black', ls='--', lw=1)
... sns.lineplot(x=pd.to_datetime(df_PT_insample1.index), y=(df_PT_insample1['Short Securities'] / df_PT_insample1['Margin Account']).fillna(0), label='Leverage', color='purple', lw=3)
... # X-axis settings
```

```
367 ... date_locator = mdates.YearLocator()
368 ... date_formatter = mdates.DateFormatter('%Y')
369 ... ax.tick_params(axis='x', labelrotation=0, labelsize=
    18)
370 ... ax.xaxis.set_major_locator(date_locator)
371 ... ax.xaxis.set_major_formatter(date_formatter)
372 ... ax.set_xlabel(xlabel='')
373 ... # Y-axis settings
374 ... ax.set_yticklabels(labels=['{:.2f}'.format(y) for y
    in ax.get_yticks()], size=18)
375 ... ax.set_ylabel(ylabel='')
376 ... # Legend settings
377 ... ax.legend(loc='upper left', fontsize=16)
378 ... # Show and save
379 ... plt.show()
380 ... fig.savefig(Path.joinpath(paths.get('output'), 'Q3.
    4_Evolution_Leverage.png'))
381 ... plt.close()
382 ...
383 ... # *** Question 3.5 ***
384 ... # Trading table
385 ... df_PT_insample2 = fn.tab_PT_insample(df_data=df_data
    , A='ZW Adj Close', B='ZC Adj Close', W=1000, L=20,
    in_level=1.5, stop_level=None)
386 ...
387 ... # Report
388 ... print('\nPair Trading Report (IS, W=1000, L=20, z_in=
    1.5, z_stop=None)')
389 ... fn.print_PT_report(df_PT=df_PT_insample2)
390 ...
391 ... # Plot wealth
392 ... sns.set(context='paper', style='ticks', font_scale=1.
    0)
393 ... fig = plt.figure(figsize=(12, 7), dpi=600)
394 ... ax = fig.add_subplot()
395 ... ax.grid(False)
396 ... ax.set_title(label='Evolution of Wealth (IS, L=20)',
    size=28)
397 ... # Items
398 ... ax.axhline(y=df_PT_insample2.iloc[0, df_PT_insample2.
    columns.get_loc('Equity')], color='black', ls='--', lw=1)
399 ... sns.lineplot(x=pd.to_datetime(df_PT_insample2.index
    ), y=df_PT_insample2['Equity'], label='Wealth', color='
    blue', lw=3)
```

```
... # X-axis settings
... date_locator = mdates.YearLocator()
... date_formatter = mdates.DateFormatter('%Y')
... ax.tick_params(axis='x', labelrotation=0, labelsize=18)
... ax.xaxis.set_major_locator(date_locator)
... ax.xaxis.set_major_formatter(date_formatter)
... ax.set_xlabel(xlabel='')
... # Y-axis settings
... ax.set_yticklabels(labels=['{:.0f}'.format(y) for y in ax.get_yticks()], size=18)
... ax.set_ylabel(ylabel='')
... # Legend settings
... ax.legend(loc='upper left', fontsize=16)
... # Show and save
... plt.show()
... fig.savefig(Path.joinpath(paths.get('output'), 'Q3.5_Evolution_Wealth.png'))
... plt.close()
...
... # TODO: Plot positions (???)
...
... # Plot leverage
... sns.set(context='paper', style='ticks', font_scale=1.0)
... fig = plt.figure(figsize=(12, 7), dpi=600)
... ax = fig.add_subplot()
... ax.grid(False)
... ax.set_title(label='Evolution of Leverage (IS, L=20)', size=28)
... # Items
... ax.axhline(y=20, color='black', ls='--', lw=1)
... sns.lineplot(x=pd.to_datetime(df_PT_insample2.index), y=(df_PT_insample2['Short Securities'] / df_PT_insample2['Margin Account']).fillna(0), label='Leverage', color='purple', lw=3)
... # X-axis settings
... date_locator = mdates.YearLocator()
... date_formatter = mdates.DateFormatter('%Y')
... ax.tick_params(axis='x', labelrotation=0, labelsize=18)
... ax.xaxis.set_major_locator(date_locator)
... ax.xaxis.set_major_formatter(date_formatter)
... ax.set_xlabel(xlabel='')
```

```
435 ... # Y-axis settings
436 ... ax.set_yticklabels(labels=['{:.0f}'.format(y) for y
    in ax.get_yticks()], size=18)
437 ... ax.set_ylabel(ylabel='')
438 ... # Legend settings
439 ... ax.legend(loc='upper left', fontsize=16)
440 ... # Show and save
441 ... plt.show()
442 ... fig.savefig(Path.joinpath(paths.get('output'), 'Q3.
    5_Evolution_Leverage.png'))
443 ... plt.close()
444 ...
445 ...
446 ... # ***************************************************
447 ... # *** QUESTION 3.2.2: Stop Loss                   ***
448 ... # ***************************************************
449 ...
450 ... # *** Question 3.6 ***
451 ... # TODO: check conditional distribution AR(1)
452 ... # Fit AR(1) model with const (drift)
453 ... model = sm.tsa.AutoReg(df_PT_insample1['Spread'],
    lags=1, trend='c').fit()
454 ...
455 ... # Conditional distribution under stationarity
456 ... mu = model.params[0] / (1 - model.params[1])
457 ... var = model.sigma2 / ((1 - model.params[1]) ** 2)
458 ...
459 ... # Report
460 ... print('\nProb(z_t+1 > z_stop=1.75) = {:.2%}'.format(1
     - norm.cdf(x=1.75, loc=mu, scale=np.sqrt(var))))
461 ... print('Prob(z_t+1 > z_stop=2.75) = {:.2%}'.format(1
     - norm.cdf(x=2.75, loc=mu, scale=np.sqrt(var))))
462 ...
463 ... # *** Question 3.7 ***
464 ... df_PT_insample3 = fn.tab_PT_insample(df_data=df_data
    , A='ZW Adj Close', B='ZC Adj Close', W=1000, L=2,
    in_level=1.5, stop_level=2.75)
465 ...
466 ... # Report
467 ... print('\nPair Trading Report (IS, W=1000, L=2, z_in=1
    .5, z_stop=2.75)')
468 ... fn.print_PT_report(df_PT=df_PT_insample3)
469 ...
470 ...
```

```
471 ... # ************************************************
472 ... # *** QUESTION 3.3: Out-of-Sample Pair Trading   ***
473 ... # ************************************************
474 ...
475 ... # *** Question 3.8/9/10 ***
476 ... # Trading table
477 ... df_PT_outsample1 = fn.tab_PT_outsample(df_data=
    df_data, A='ZW Adj Close', B='ZC Adj Close', df_ts_coint=
    df_ts_coint_500, W=1000, L=2, in_level=1.5, stop_level=2.
    75, sample_size=500, pred_size=20, sig_coint=None)
478 ...
479 ... # Report
480 ... print('\nPair Trading Report (OS, W=1000, L=2, z_in=1
    .5, z_stop=2.75, sig_coint=None)')
481 ... fn.print_PT_report(df_PT=df_PT_outsample1)
482 ...
483 ... # Plot correlations
484 ... sns.set(context='paper', style='ticks', font_scale=1.
    0)
485 ... fig = plt.figure(figsize=(12, 7), dpi=600)
486 ... ax = fig.add_subplot()
487 ... ax.grid(False)
488 ... ax.set_title(label='Rolling Correlations (OS)', size=
    28)
489 ... # Items
490 ... sns.lineplot(x=pd.to_datetime(df_PT_outsample1.index
    ), y=df_PT_outsample1['Corr Prices'], label='Corr Prices
    ', color='green', lw=3)
491 ... sns.lineplot(x=pd.to_datetime(df_PT_outsample1.index
    ), y=df_PT_outsample1['Corr Returns'], label='Corr
    Returns', color='firebrick', lw=3)
492 ... # X-axis settings
493 ... date_locator = mdates.YearLocator()
494 ... date_formatter = mdates.DateFormatter('%Y')
495 ... ax.tick_params(axis='x', labelrotation=0, labelsize=
    18)
496 ... ax.xaxis.set_major_locator(date_locator)
497 ... ax.xaxis.set_major_formatter(date_formatter)
498 ... ax.set_xlabel(xlabel='')
499 ... # Y-axis settings
500 ... ax.set_yticklabels(labels=['{:.1f}'.format(y) for y
    in ax.get_yticks()], size=18)
501 ... ax.set_ylabel(ylabel='')
502 ... # Legend settings
```

```
503 ... ax.legend(loc='lower left', fontsize=16)
504 ... # Show and save
505 ... plt.show()
506 ... fig.savefig(Path.joinpath(paths.get('output'), 'Q3.
    9_Alphas.png'))
507 ... plt.close()
508 ...
509 ... # Plot alphas
510 ... sns.set(context='paper', style='ticks', font_scale=1.
    0)
511 ... fig = plt.figure(figsize=(12, 7), dpi=600)
512 ... ax = fig.add_subplot()
513 ... ax.grid(False)
514 ... ax.set_title(label='Alpha OS vs. IS', size=28)
515 ... # Items
516 ... sns.lineplot(x=pd.to_datetime(df_PT_outsample1.index
    ), y=df_PT_outsample1['Alpha'], label='Alpha OS', color='
    orange', lw=3)
517 ... sns.lineplot(x=pd.to_datetime(df_PT_insample3.index
    ), y=df_PT_insample3['Alpha'], label='Alpha IS', color='
    red', lw=3)
518 ... # X-axis settings
519 ... date_locator = mdates.YearLocator()
520 ... date_formatter = mdates.DateFormatter('%Y')
521 ... ax.tick_params(axis='x', labelrotation=0, labelsize=
    18)
522 ... ax.xaxis.set_major_locator(date_locator)
523 ... ax.xaxis.set_major_formatter(date_formatter)
524 ... ax.set_xlabel(xlabel='')
525 ... # Y-axis settings
526 ... ax.set_yticklabels(labels=['{:.0f}'.format(y) for y
    in ax.get_yticks()], size=18)
527 ... ax.set_ylabel(ylabel='')
528 ... # Legend settings
529 ... ax.legend(loc='upper left', fontsize=16)
530 ... # Show and save
531 ... plt.show()
532 ... fig.savefig(Path.joinpath(paths.get('output'), 'Q3.
    9_Alphas.png'))
533 ... plt.close()
534 ...
535 ... # Plot betas
536 ... sns.set(context='paper', style='ticks', font_scale=1.
    0)
```

```
... fig = plt.figure(figsize=(12, 7), dpi=600)
... ax = fig.add_subplot()
... ax.grid(False)
... ax.set_title(label='Beta OS vs. IS', size=28)
... # Items
... sns.lineplot(x=pd.to_datetime(df_PT_outsample1.index
    ), y=df_PT_outsample1['Beta'], label='Beta OS', color='
    orange', lw=3)
... sns.lineplot(x=pd.to_datetime(df_PT_insample3.index
    ), y=df_PT_insample3['Beta'], label='Beta IS', color='red
    ', lw=3)
... # X-axis settings
... date_locator = mdates.YearLocator()
... date_formatter = mdates.DateFormatter('%Y')
... ax.tick_params(axis='x', labelrotation=0, labelsize=
    18)
... ax.xaxis.set_major_locator(date_locator)
... ax.xaxis.set_major_formatter(date_formatter)
... ax.set_xlabel(xlabel='')
... # Y-axis settings
... ax.set_yticklabels(labels=['{:.1f}'.format(y) for y
    in ax.get_yticks()], size=18)
... ax.set_ylabel(ylabel='')
... # Legend settings
... ax.legend(loc='upper left', fontsize=16)
... # Show and save
... plt.show()
... fig.savefig(Path.joinpath(paths.get('output'), 'Q3.
    9_Betas.png'))
... plt.close()
...
... # Plot spreads
... sns.set(context='paper', style='ticks', font_scale=1.
    0)
... fig = plt.figure(figsize=(12, 7), dpi=600)
... ax = fig.add_subplot()
... ax.grid(False)
... ax.set_title(label='Spread OS vs. IS', size=28)
... # Items
... ax.axhline(y=0, color='black', ls='--', lw=1)
... ax.axhline(y=1.5, color='black', ls='--', lw=1)
... ax.axhline(y=-1.5, color='black', ls='--', lw=1)
... sns.lineplot(x=pd.to_datetime(df_PT_outsample1.index
    ), y=df_PT_outsample1['Spread'], label='Spread OS', color
```

```
571 ='orange', lw=3)
572 ... sns.lineplot(x=pd.to_datetime(df_PT_insample3.index
    ), y=df_PT_insample3['Spread'], label='Spread IS', color
    ='red', lw=3)
573 ... # X-axis settings
574 ... date_locator = mdates.YearLocator()
575 ... date_formatter = mdates.DateFormatter('%Y')
576 ... ax.tick_params(axis='x', labelrotation=0, labelsize=
    18)
577 ... ax.xaxis.set_major_locator(date_locator)
578 ... ax.xaxis.set_major_formatter(date_formatter)
579 ... ax.set_xlabel(xlabel='')
580 ... # Y-axis settings
581 ... ax.set_yticklabels(labels=['{:.1f}'.format(y) for y
    in ax.get_yticks()], size=18)
582 ... ax.set_ylabel(ylabel='')
583 ... # Legend settings
584 ... ax.legend(loc='upper left', fontsize=16)
585 ... # Show and save
586 ... plt.show()
587 ... fig.savefig(Path.joinpath(paths.get('output'), 'Q3.
    9_Spreads.png'))
588 ... plt.close()
589 ...
590 ... # *** Question 3.11/12 ***
591 ... # Trading table
592 ... df_PT_outsample2 = fn.tab_PT_outsample(df_data=
    df_data, A='ZW Adj Close', B='ZC Adj Close', df_ts_coint=
    df_ts_coint_500, W=1000, L=2, in_level=1.5, stop_level=2.
    75, sample_size=500, pred_size=20, sig_coint=0.1)
593 ...
594 ... # Report
595 ... print('\nPair Trading Report (OS, W=1000, L=2, z_in=1
    .5, z_stop=2.75, sig_coint=10%)')
596 ... fn.print_PT_report(df_PT=df_PT_outsample2)
597 ...
598 ... # Plot p-values (stem plot)
599 ... fig = plt.figure(figsize=(12, 7), dpi=600)
600 ... ax = fig.add_subplot()
601 ... ax.set_title(label='Cointegration P-values (OS)',
    size=28)
602 ... # Items
603 ... plt.stem(pd.to_datetime(df_PT_insample2.index),
    df_PT_outsample2['Coint PV'], bottom=0.1)
```

```
604 ... # X-axis settings
605 ... date_locator = mdates.YearLocator()
606 ... date_formatter = mdates.DateFormatter('%Y')
607 ... ax.tick_params(axis='x', labelrotation=0, labelsize=
    18)
608 ... ax.xaxis.set_major_locator(date_locator)
609 ... ax.xaxis.set_major_formatter(date_formatter)
610 ... ax.set_xlabel(xlabel='')
611 ... # Y-axis settings
612 ... ax.set_yticklabels(labels=['{:.1f}'.format(y) for y
    in ax.get_yticks()], size=18)
613 ... ax.set_ylabel(ylabel='')
614 ... # Show and save
615 ... plt.show()
616 ... fig.savefig(Path.joinpath(paths.get('output'), 'Q3.
    11_Cointegration_PV.png'))
617 ... plt.close()
618 ...
619 ...
620 ... # %%
621 ... # ***************************************************
622 ... # *** Branch: Florian  Test                        ***
623 ... # ***************************************************
624 ... """
625 ... T = len(df_data_ln.index)
626 ... N = 10000
627 ... column = 'ZC Adj Close'
628 ... df = df_data_ln
629 ...
630 ... # Generate white noise
631 ... np.random.seed(23031997)
632 ... white_noise = np.random.normal(0, 1, size=(T, N))
633 ...
634 ... # Output DataFrame
635 ... ar_parameters = pd.DataFrame(columns=['AR_Coeff', '
    AR_Coeff_SD', 'DF_TS'])
636 ...
637 ... # Select P(0)
638 ... p0 = df[column][0]
639 ... white_noise[0] = p0
640 ...
641 ... # Aggregate the shocks
642 ... white_noise_agg = white_noise.cumsum(axis=0)
643 ...
```

```
... for i in tqdm(range(0, N), desc="Simulating Test Statistics"):
...     # Step 3: Estimate AR(1) Model
...
...
...     ar_model = AutoReg(white_noise_agg[:, i], lags=1, trend='c').fit()
...     phi_hat = ar_model.params[1]
...     phi_std = ar_model.bse[1]
...
...     '''
...     print(phi_hat)
...     print(ar_model.params[0])
...     print(phi_std)
...
...
...     p_t = pd.Series(white_noise_agg[:, i])[1:]
...     p_t_1 = pd.Series(white_noise_agg[:, i]).shift(1)[1:]
...     T_1 = len(p_t)
...     phi_hat_1 = p_t.cov(p_t_1)/p_t_1.var()
...     print(phi_hat_1)
...
...     u = p_t.mean() - phi_hat*p_t_1.mean()
...     print(u)
...
...     #s2 = (1/(T_1-1))*sum((p_t - ar_model.params[0] - phi_hat*p_t_1)**2)
...     s2 = (1/(T_1-1))*sum((ar_model.resid)**2)
...     phi_hat_std_1 = s2/(sum((p_t_1 - p_t_1.mean())**2)**0.5)
...     print(phi_hat_std_1)
...
...
...
...     # Step 4: Compute the T-Statistic
...     df_stat = (phi_hat - 1) / phi_std
...     print(df_stat)
...
...     # Step 4: Compute the T-Statistic
...     df_stat_1 = (phi_hat - 1) / phi_hat_std_1
...     print(df_stat_1)
...
...     '''
```

```
683 ...
684 ...     # Step 4: Compute the T-Statistic
685 ...     df_stat = (phi_hat - 1) / phi_std
686 ...
687 ...     ar_parameters.loc[i] = [phi_hat, phi_std, df_stat
    ]
688 ...
689 ... # Computing the critical values
690 ... critical_val = ar_parameters['DF_TS'].quantile([0.01
    , 0.05, 0.1])
691 ... critical_val = critical_val.rename('Critical Value')
692 ...
693 ...
694 ... for col in l_adj_close_price:
695 ...     t_stat_data = fn.reg(df_data_ln, col, lag=1)
696 ...     print(t_stat_data)
697 ...
698 ...
699 ... import statsmodels.api as sm
700 ...
701 ...
702 ... df = df_data_ln
703 ... lag = 1
704 ...
705 ... for column in l_adj_close_price:
706 ...  # New DataFrame
707 ...     new_df = pd.DataFrame(df[column].copy())
708 ...
709 ...     # Creating lagged feature
710 ...     new_df['Lagged'] = new_df[column].shift(lag)
711 ...     new_df.dropna(inplace=True)
712 ...
713 ...     X = new_df['Lagged'].values
714 ...     y = new_df[column].values
715 ...
716 ...     # Run linear regression
717 ...     X = sm.add_constant(X)
718 ...     model = sm.OLS(y, X)
719 ...     reg_results = model.fit()
720 ...     #print(reg_results.params[1])
721 ...     #print(reg_results.bse[1])
722 ...     #print(reg_results.params[0])
723 ...     #print(reg_results.bse[1])
724 ...     # Computing the T-Stat from the regression
```

```
parameters
...     t_stat_data = (reg_results.params[1] - 1) /
    reg_results.bse[1]
...     s2 = (1 / (len(X[:,1]) - 1)) * sum((y -
    reg_results.params[0] - reg_results.params[1] * X[:,1
    ]) ** 2)
...     s2 = (1 / (len(X[:, 1]) - 1)) * sum((reg_results.
    resid)**2)
...     print(s2)
...
... for col in l_adj_close_price:
...     # White noise array.
...     p_t = pd.Series(df_data_ln[col])[1:]
...     # Lagged White noise array.
...     p_t_1 = pd.Series(df_data_ln[col]).shift(1)[1:]
...     T_1 = len(p_t)
...     # Phi hat calculation
...     phi_hat = p_t.cov(p_t_1) / p_t_1.var()
...
...     # Standard error calculation
...     u = p_t.mean() - phi_hat * p_t_1.mean()
...
...     s2 = (1/(T_1-1))*sum((p_t - u - phi_hat*p_t_1)**2
    )
...     phi_std = (s2 / (sum((p_t_1 - p_t_1.mean()) ** 2
    )))** 0.5
...
...     # Step 4: Compute the T-Statistic
...     df_stat = (phi_hat - 1) / phi_std
...     print(df_stat)
... """
...
Simulating Test Statistics: 100%|██████████| 10000/10000
     [00:21<00:00, 464.88it/s]
Simulating Test Statistics: 100%|██████████| 10000/10000
     [00:21<00:00, 471.90it/s]
Simulating Test Statistics: 100%|██████████| 10000/10000
     [00:26<00:00, 372.43it/s]
Simulating Test Statistics: 100%|██████████| 10000/10000
     [00:21<00:00, 469.88it/s]

Ljung-Box Test Report
Test stat (Q_p): 17834.77
P-value: 0.0
```

```
758
759 Pair Trading Report (IS, W=1000, L=2, z_in=1.5, z_stop=
    None)
760 Profit:         1078.72
761 ROE:            107.87%
762 Init wealth:1000.00
763 Final wealth:   2078.72
764 Min wealth:     976.82
765 Max wealth:     2078.72
766 Pos1 trades:4
767 Pos2 trades:2
768 Total trades:   6
769
770 Pair Trading Report (IS, W=1000, L=20, z_in=1.5, z_stop=
    None)
771 Profit:         1759.65
772 ROE:            175.97%
773 Init wealth:1000.00
774 Final wealth:   2759.65
775 Min wealth:     966.89
776 Max wealth:     2759.65
777 Pos1 trades:4
778 Pos2 trades:2
779 Total trades:   6
780
781 Prob(z_t+1 > z_stop=1.75) = 44.23%
782 Prob(z_t+1 > z_stop=2.75) = 41.03%
783
784 Pair Trading Report (IS, W=1000, L=2, z_in=1.5, z_stop=2.
    75)
785 Profit:         707.91
786 ROE:            70.79%
787 Init wealth:1000.00
788 Final wealth:   1707.91
789 Min wealth:     976.82
790 Max wealth:     1725.67
791 Pos1 trades:5
792 Pos2 trades:3
793 Total trades:   8
794
795 Pair Trading Report (OS, W=1000, L=2, z_in=1.5, z_stop=2.
    75, sig_coint=None)
796 Profit:         334.03
797 ROE:            33.40%
```

```
Init wealth:1000.00
Final wealth:   1334.03
Min wealth:     876.56
Max wealth:     1426.83
Pos1 trades:29
Pos2 trades:4
Total trades:   33

Pair Trading Report (OS, W=1000, L=2, z_in=1.5, z_stop=2.75, sig_coint=10%)
Profit:         -106.10
ROE:            -10.61%
Init wealth:1000.00
Final wealth:   893.90
Min wealth:     890.85
Max wealth:     1067.88
Pos1 trades:12
Pos2 trades:0
Total trades:   12

```