Exercises_2_and_3

July 14, 2025

First, we download the data. By looking at it in advance I've noticed it lacked column names, we'll fix it here:

[1]:
```python
import pandas as pd
import numpy as np

url = 'https://www.chicagobooth.edu/-/media/faculty/ruey-s-tsay/teaching/fts2/
  ↪m-ibm3dx7503.txt'
col_names = ['date', 'ibm', 'crsp_vw', 'crsp_ew', 's&p']

data = pd.read_table(url, header=None, names=col_names ,sep='\s+')
```

```
<>:7: SyntaxWarning: invalid escape sequence '\s'
<>:7: SyntaxWarning: invalid escape sequence '\s'
/var/folders/f5/jdc2n89x0r11q0l7zdg17s9w0000gn/T/ipykernel_9124/908117592.py:7:
SyntaxWarning: invalid escape sequence '\s'
  data = pd.read_table(url, header=None, names=col_names ,sep='\s+')
```

Now we inspect it:

[2]:
```python
data.head()
```

[2]:

	date	ibm	crsp_vw	crsp_ew	s&p
0	19750131	0.12054	0.14150	0.29921	0.12281
1	19750228	0.15272	0.05842	0.05392	0.05989
2	19750331	-0.04118	0.03019	0.08150	0.02169
3	19750430	0.01573	0.04649	0.03109	0.04726
4	19750530	0.03157	0.05514	0.07288	0.04410

We can immediately notice the same date issue as before, we'll fix it now:

[3]:
```python
data['date'] = pd.to_datetime(data['date'], format='%Y%m%d')
data.head()
```

[3]:

	date	ibm	crsp_vw	crsp_ew	s&p
0	1975-01-31	0.12054	0.14150	0.29921	0.12281
1	1975-02-28	0.15272	0.05842	0.05392	0.05989
2	1975-03-31	-0.04118	0.03019	0.08150	0.02169
3	1975-04-30	0.01573	0.04649	0.03109	0.04726
4	1975-05-30	0.03157	0.05514	0.07288	0.04410

We now add log returns and percentage columns for them and the simple *net* returns:

[4]:
```
import utils as ut

ut.add_log_returns(data, data.columns[1:])
ut.add_percent(data, data.columns[1:])

data.head()
```

[4]:
```
         date      ibm  crsp_vw  crsp_ew      s&p   log_ibm  log_crsp_vw  \
0  1975-01-31  0.12054  0.14150  0.29921  0.12281  0.113811     0.132343
1  1975-02-28  0.15272  0.05842  0.05392  0.05989  0.142124     0.056777
2  1975-03-31 -0.04118  0.03019  0.08150  0.02169 -0.042052     0.029743
3  1975-04-30  0.01573  0.04649  0.03109  0.04726  0.015608     0.045442
4  1975-05-30  0.03157  0.05514  0.07288  0.04410  0.031082     0.053673

   log_crsp_ew   log_s&p  ibm_percent  crsp_vw_percent  crsp_ew_percent  \
0     0.261756  0.115834       12.054           14.150           29.921
1     0.052517  0.058165       15.272            5.842            5.392
2     0.078349  0.021458       -4.118            3.019            8.150
3     0.030616  0.046177        1.573            4.649            3.109
4     0.070347  0.043155        3.157            5.514            7.288

   s&p_percent  log_ibm_percent  log_crsp_vw_percent  log_crsp_ew_percent  \
0       12.281           11.381               13.234               26.176
1        5.989           14.212                5.678                5.252
2        2.169           -4.205                2.974                7.835
3        4.726            1.561                4.544                3.062
4        4.410            3.108                5.367                7.035

   log_s&p_percent
0           11.583
1            5.817
2            2.146
3            4.618
4            4.316
```

Now to calculate the statistics:

[5]:
```
num_data = data[data.columns[1:]]
print(num_data.describe())
print(num_data.skew())
num_data.kurtosis()
```

```
              ibm     crsp_vw     crsp_ew         s&p     log_ibm  \
count  348.000000  348.000000  348.000000  348.000000  348.000000
mean     0.011753    0.011909    0.015908    0.009032    0.008722
std      0.078139    0.045618    0.056537    0.044435    0.077131
min     -0.261900   -0.225340   -0.272310   -0.217630   -0.303676
```

25%	-0.037757	-0.015807	-0.015977	-0.017552	-0.038489
50%	0.010110	0.014945	0.016995	0.010130	0.010059
75%	0.055410	0.043165	0.047565	0.038935	0.053929
max	0.353800	0.141500	0.299210	0.131770	0.302915

	log_crsp_vw	log_crsp_ew	log_s&p	ibm_percent	crsp_vw_percent \
count	348.000000	348.000000	348.000000	348.000000	348.000000
mean	0.010800	0.014210	0.008006	1.175336	1.190868
std	0.045949	0.056522	0.044695	7.813905	4.561798
min	-0.255331	-0.317880	-0.245428	-26.190000	-22.534000
25%	-0.015934	-0.016107	-0.017708	-3.775750	-1.580750
50%	0.014834	0.016852	0.010079	1.011000	1.494500
75%	0.042259	0.046468	0.038196	5.541000	4.316500
max	0.132343	0.261756	0.123783	35.380000	14.150000

	crsp_ew_percent	s&p_percent	log_ibm_percent	log_crsp_vw_percent \
count	348.000000	348.000000	348.000000	348.000000
mean	1.590805	0.903213	0.872198	1.079974
std	5.653741	4.443510	7.713083	4.594929
min	-27.231000	-21.763000	-30.368000	-25.533000
25%	-1.597750	-1.755250	-3.848750	-1.593750
50%	1.699500	1.013000	1.006000	1.483500
75%	4.756500	3.893500	5.393000	4.226250
max	29.921000	13.177000	30.292000	13.234000

	log_crsp_ew_percent	log_s&p_percent
count	348.000000	348.000000
mean	1.420997	0.800624
std	5.652252	4.469428
min	-31.788000	-24.543000
25%	-1.610750	-1.771250
50%	1.685500	1.008000
75%	4.647250	3.819750
max	26.176000	12.378000

ibm	0.332547
crsp_vw	-0.631580
crsp_ew	-0.182767
s&p	-0.476261
log_ibm	-0.071984
log_crsp_vw	-0.927199
log_crsp_ew	-0.735505
log_s&p	-0.749076
ibm_percent	0.332547
crsp_vw_percent	-0.631580
crsp_ew_percent	-0.182767
s&p_percent	-0.476261
log_ibm_percent	-0.071988
log_crsp_vw_percent	-0.927226

```
log_crsp_ew_percent   -0.735485
log_s&p_percent       -0.749123
dtype: float64
```

[5]:
```
ibm                    1.693570
crsp_vw                2.312882
crsp_ew                4.423737
s&p                    1.947149
log_ibm                1.559987
log_crsp_vw            3.545620
log_crsp_ew            5.403116
log_s&p                2.997021
ibm_percent            1.693570
crsp_vw_percent        2.312882
crsp_ew_percent        4.423737
s&p_percent            1.947149
log_ibm_percent        1.560051
log_crsp_vw_percent    3.545596
log_crsp_ew_percent    5.403043
log_s&p_percent        2.997222
dtype: float64
```

Finally we run the asymptotic z-tests under the asymptotic normality assumption:

[6]:
```
ut.t_test_for_mean(data, data.columns[1:5])
```

```
For ibm, the p_value is: 0.005299468414435815
For crsp_vw, the p_value is: 1.699901579383222e-06
For crsp_ew, the p_value is: 2.67082015055518e-07
For s&p, the p_value is: 0.0001763033264922363
```

Which means that we reject the null for all of them since they are all smaller then 0.05.

Now for exercise 3, we calculate the annual *avarage* log returns. Since the data is monthly, the mean we found for 'log_s&p' was $\frac{1}{m}\sum_{j=1}^{m} r_j$ where m is the number of months the data spans. Notice that, for any $t \in \mathbb{N}$ we have:

$$r_{t+12} = \ln(\frac{P_{t+12}}{P_t}) = \ln(\prod_{j=0}^{1} 1 \frac{P_{t+j+1}}{P_{t+j}}) = \sum_{j=0}^{1} 1 \ln(\frac{P_{t+j+1}}{P_{t+j}})$$

This means that, if $y = \frac{m}{12}$ is the amount of years the data spans (which is true in our case since our data spans from January of 75 to December of 03 so $m\%12 = 0$) and t is the earliest date in our data, we get:

$$\frac{1}{y}\sum_{j=0}^{y} r_{t+12j} = 12 * \frac{1}{m}\sum_{j=0}^{m} r_{t+j}$$

Thus, it is enough to multiply the monthly mean by 12 to get:

[7]:
```
12*data['log_s&p'].mean()
```

[7]: np.float64(0.09607478867326014)

Now, if we were to invest 1$ on the S&P composite index in January 75, we can calculate the investments value in December 2003 by summing the monthly log returns withing that span and exponentiating it. This is because the compounded simple returns turn from a product to a sum when the logarithm is applied. Thus, we get:

[8]:
```
np.exp(data['log_s&p'].sum())
```

[8]: np.float64(16.218764453481448)

Which means it would be worth about 16$ twenty nine years later.