Execises_1_and_4

July 14, 2025

First we download our file from the web straight into a data frame (notice that Tsay's files tend to be space separated as opposed to comma separated):

[14]:
```
import numpy as np
import pandas as pd

url = "https://faculty.chicagobooth.edu/-/media/faculty/ruey-s-tsay/teaching/
↪fts3/d-3stocks9908.txt"

data = pd.read_table(url, sep="\s+")
data.head()
```

```
<>:6: SyntaxWarning: invalid escape sequence '\s'
<>:6: SyntaxWarning: invalid escape sequence '\s'
/var/folders/f5/jdc2n89x0r11q0l7zdg17s9w0000gn/T/ipykernel_8852/4099323659.py:6:
SyntaxWarning: invalid escape sequence '\s'
  data = pd.read_table(url, sep="\s+")
```

[14]:

	date	axp	cat	sbux
0	19990104	-0.009756	0.029891	-0.040089
1	19990105	-0.019089	-0.002639	-0.034803
2	19990106	0.043063	0.026455	-0.008413
3	19990107	0.012063	0.009021	0.003636
4	19990108	0.030393	0.042146	0.021739

Notice that our date column has the wrong type and formatting:

[15]:
```
data['date'].dtype
data['date'].isnull().any()
```

[15]:
```
np.False_
```

Since we can see there aren't any empty date entires, we can transform them into the correct format and type as follows:

[16]:
```
data['date'] = pd.to_datetime(data['date'], format='%Y%m%d')
data['date'].head()
```

[16]:
```
0   1999-01-04
1   1999-01-05
```

```
2   1999-01-06
3   1999-01-07
4   1999-01-08
Name: date, dtype: datetime64[ns]
```

Now we can start the exercise. First we add a percentage return column for each company. First we check for null values:

[17]:
```
data.isnull().any()
```

[17]:
```
date     False
axp      False
cat      False
sbux     False
dtype: bool
```

Since we can see there aren't any null values in our data, we can go ahead with the calculation:

[18]:
```
import utils as ut
ut.add_percent(data, data.columns[1:])
data.head()
```

[18]:
```
        date       axp       cat      sbux  axp_percent  cat_percent  \
0 1999-01-04 -0.009756  0.029891 -0.040089       -0.976        2.989
1 1999-01-05 -0.019089 -0.002639 -0.034803       -1.909       -0.264
2 1999-01-06  0.043063  0.026455 -0.008413        4.306        2.645
3 1999-01-07  0.012063  0.009021  0.003636        1.206        0.902
4 1999-01-08  0.030393  0.042146  0.021739        3.039        4.215

   sbux_percent
0        -4.009
1        -3.480
2        -0.841
3         0.364
4         2.174
```

Next we compute the sample statistics:

[19]:
```
percent_data = data.filter(regex='_percent$')
percent_data.describe()
```

[19]:
```
       axp_percent  cat_percent  sbux_percent
count  2515.000000  2515.000000   2515.000000
mean      0.014564     0.059509      0.048058
std       2.446229     2.169657      2.682621
min     -17.595000   -14.518000    -28.286000
25%      -1.111000    -1.144000     -1.247000
50%      -0.018000     0.049000     -0.051000
75%       1.093000     1.206000      1.249000
```

```
max       17.927000     14.723000      14.635000
```

We are left with calculating the sample skewness and excess kurtosis:

[20]:
```python
print(percent_data.skew())
percent_data.kurtosis()
```

```
axp_percent    -0.034611
cat_percent     0.011685
sbux_percent   -0.082529
dtype: float64
```

[20]:
```
axp_percent     6.069710
cat_percent     4.470628
sbux_percent    8.774512
dtype: float64
```

Now we can move on to part two. We will add the log returns as a new column for our data frame:

[21]:
```python
ut.add_log_returns(data, data.drop(columns=['date', *percent_data], axis=1).
 ↪columns)
data.head()
```

[21]:
```
         date       axp       cat      sbux  axp_percent  cat_percent  \
0 1999-01-04 -0.009756  0.029891 -0.040089       -0.976        2.989
1 1999-01-05 -0.019089 -0.002639 -0.034803       -1.909       -0.264
2 1999-01-06  0.043063  0.026455 -0.008413        4.306        2.645
3 1999-01-07  0.012063  0.009021  0.003636        1.206        0.902
4 1999-01-08  0.030393  0.042146  0.021739        3.039        4.215

   sbux_percent   log_axp   log_cat  log_sbux
0        -4.009 -0.009804  0.029453 -0.040915
1        -3.480 -0.019274 -0.002642 -0.035423
2        -0.841  0.042162  0.026111 -0.008449
3         0.364  0.011991  0.008981  0.003629
4         2.174  0.029940  0.041282  0.021506
```

Moving on to part 3, we will add percentage columns for the log returns too:

[22]:
```python
ut.add_percent(data, data.drop(columns=['date', *percent_data], axis=1).columns)
data.head()
```

[22]:
```
         date       axp       cat      sbux  axp_percent  cat_percent  \
0 1999-01-04 -0.009756  0.029891 -0.040089       -0.976        2.989
1 1999-01-05 -0.019089 -0.002639 -0.034803       -1.909       -0.264
2 1999-01-06  0.043063  0.026455 -0.008413        4.306        2.645
3 1999-01-07  0.012063  0.009021  0.003636        1.206        0.902
4 1999-01-08  0.030393  0.042146  0.021739        3.039        4.215
```

```
   sbux_percent   log_axp   log_cat  log_sbux  log_axp_percent  \
0        -4.009 -0.009804  0.029453 -0.040915           -0.980
1        -3.480 -0.019274 -0.002642 -0.035423           -1.927
2        -0.841  0.042162  0.026111 -0.008449            4.216
3         0.364  0.011991  0.008981  0.003629            1.199
4         2.174  0.029940  0.041282  0.021506            2.994

   log_cat_percent  log_sbux_percent
0            2.945            -4.091
1           -0.264            -3.542
2            2.611            -0.845
3            0.898             0.363
4            4.128             2.151
```

Again, we calculate the statistics, this time all at once:

[23]:
```python
log_percent_data = data.filter(regex='^log_.*_percent$')
print(log_percent_data.skew())
print(log_percent_data.kurtosis())
log_percent_data.describe()
```

```
log_axp_percent    -0.336829
log_cat_percent    -0.201986
log_sbux_percent   -0.597794
dtype: float64
log_axp_percent      6.509215
log_cat_percent      4.712774
log_sbux_percent    12.936699
dtype: float64
```

[23]:

	log_axp_percent	log_cat_percent	log_sbux_percent
count	2515.000000	2515.000000	2515.000000
mean	-0.015436	0.035951	0.011879
std	2.452892	2.171488	2.695884
min	-19.352000	-15.686000	-33.249000
25%	-1.117500	-1.150500	-1.255000
50%	-0.018000	0.049000	-0.051000
75%	1.087000	1.199000	1.241000
max	16.489000	13.735000	13.659000

Lastly, we perform the three hypothesis tests for the mean. Assuming asymptotic normality here (quite a big assumption but seems to be what Tsay wants):

[24]:
```python
log_ret = data.filter(regex='^log_').drop(log_percent_data, axis=1)
ut.t_test_for_mean(log_ret, log_ret.columns)
```

```
For log_axp, the p_value is: 0.7523671583912229
For log_cat, the p_value is: 0.40648956069574815
For log_sbux, the p_value is: 0.8250359651358331
```

We can clearly see that in each case the p values are much greater then 5%, thus we can accept the null.

Now we continue with exercise 4. We perform the two requested tests using scipy.stats (notice that the built in scipy test implicitly tests for $H_0 := \mathbb{E}[(\frac{X-\mu}{\sigma})^3] = 0$ since it tests against a normal distribution which has 0 skewness):

[25]:
```
from scipy import stats
log_amex = data['log_axp']
skewness_p_val = stats.skewtest(log_amex)[1]
kurtosis_p_val = stats.kurtosistest(log_amex)[1]
print(f'The p value for the skewness test is: {skewness_p_val}')
print(f'The p value for the kurtosis test is: {kurtosis_p_val}')
```

```
The p value for the skewness test is: 1.675486855417496e-11
The p value for the kurtosis test is: 1.5749248344218033e-76
```

Since both are much smaller then 5%, we reject the null for both.