FINA 6339 | Homework 1| Xingzhi Mei

In [16]:
```
# ALL Import Needed
import math as m
import statistics as st
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
import mplfinance as mpf
import datetime as dt
import scipy.stats as stats
```

A1:

In [5]:
```
# A1
# import data
A_1 = pd.read_csv("2datalog.csv")
A_1.head()
```

Out[5]:

	AAPL	MSFT
0	-0.012773	-0.017296
1	-0.026960	-0.039144
2	-0.016834	-0.007933
3	0.000988	0.000510
4	0.000116	0.000732

In [34]:
```
# AAPL
apple = A_1['AAPL']
apple.head()
```

Out[34]:
```
0   -0.012773
1   -0.026960
2   -0.016834
3    0.000988
4    0.000116
Name: AAPL, dtype: float64
```

In [52]:
```
print('AAPL:')
# mean
mean_a = apple.mean()
print("mean is "+ str(mean_a))

# median
median_a = apple.median()
print("median is "+ str(median_a))

# variance
variance_a = st.variance(apple)
print("variance is "+ str(variance_a))

# std
std_a = apple.std()
```

```
print("standard deviation is "+ str(std_a))

# skewnwss
from scipy.stats import skew
skw_a = skew(apple)
print("skewnwss is "+ str(skw_a))

# kurtosis
from scipy.stats import kurtosis
kur_a = kurtosis(apple)
print("kurtosis is "+ str(kur_a))
```

```
AAPL:
mean is -0.0009421779545454545
median is -0.00089905
variance is 0.0004945459013100748
standard deviation is 0.02223838801060174
skewnwss is 0.1839049028208159
kurtosis is 0.8446359429426069
```

In [53]:

```
# MSFT
microsoft = A_1['MSFT']
microsoft.head()
```

Out[53]:

```
0   -0.017296
1   -0.039144
2   -0.007933
3    0.000510
4    0.000732
Name: MSFT, dtype: float64
```

In [54]:

```
print('MSFT:')
# mean
mean_m = microsoft.mean()
print("mean is "+ str(mean_m))

# median
median_m = microsoft.median()
print("median is "+ str(median_m))

# variance
variance_m = st.variance(microsoft)
print("variance is "+ str(variance_m))

# std
std_m = microsoft.std()
print("standard deviation is "+ str(std_m))

# skewnwss
from scipy.stats import skew
skw_m = skew(microsoft)
print("skewnwss is "+ str(skw_m))

# kurtosis
from scipy.stats import kurtosis
kur_m = kurtosis(microsoft)
print("kurtosis is "+ str(kur_m))
```

```
MSFT:
mean is -0.0011845677537878783
median is -0.0012750650000000001
variance is 0.0004943900126990548
```

```
standard deviation is 0.022234882790315196
skewnwss is 0.024067131342274783
kurtosis is 0.7323928483755
```

In [57]:
```
# coefficient of correlation
coecor = np.corrcoef(apple, microsoft)
print("coefficient of correlation is " + str(coecor[1,0]))
```

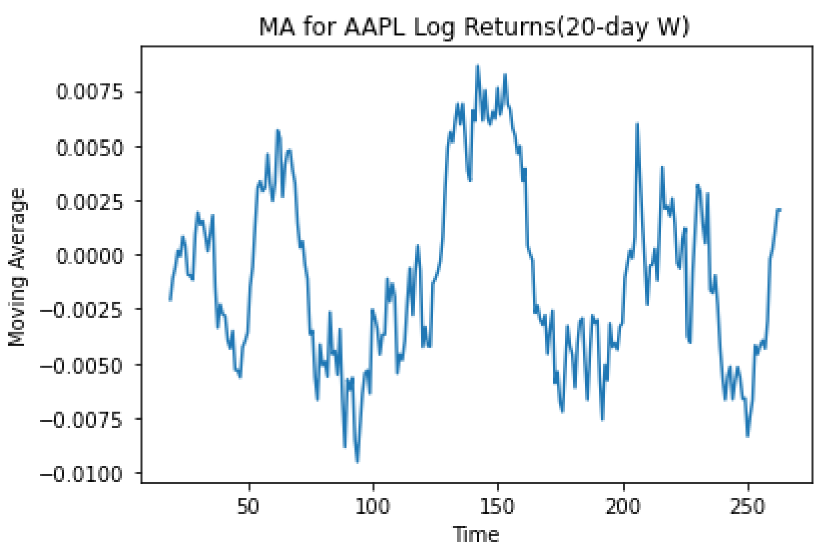
```
coefficient of correlation is 0.8065886177751094
```

Comments: \ The difference of these meansurements between Apple and Microsoft is very close to each other, as they are all in same field, facing the same tecenology problems. However, there is a sigificant gap between two companies' skewness, comparing to the other measurements. Apple has a value of 0.1839 and microsoft has a value of 0.0241. Although there is a 0.16 difference, both date of apple and microsoft are fairly symmetrical, as they are all between -0.5 to 0.5.

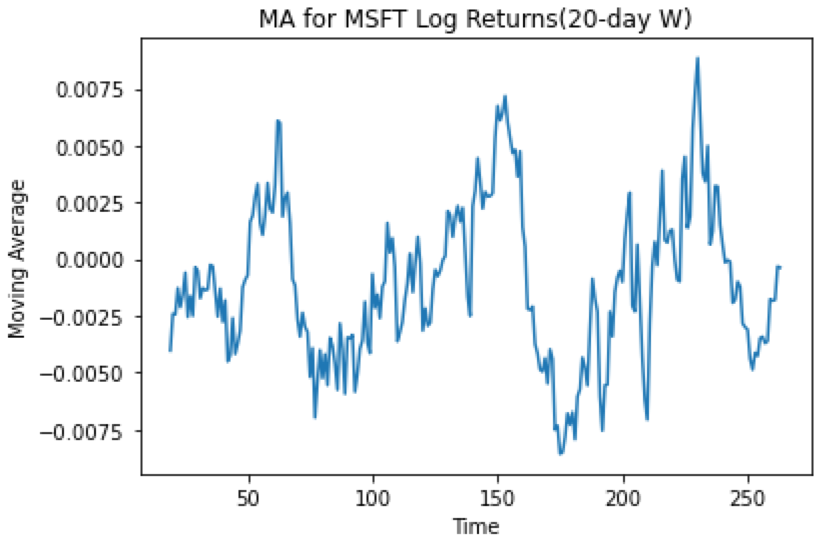
A2:

In [59]:
```
# A2
```

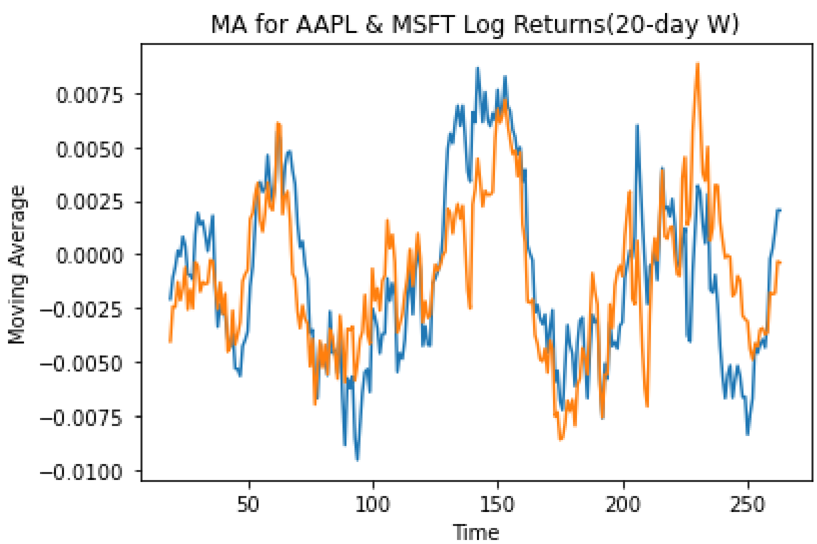
In [67]:
```
# Moving Average of 20 days
# AAPL
# using rolling function to create and calculate the windows then use mean funct
plt.plot(apple.rolling(20).mean())
plt.xlabel('Time')
plt.ylabel('Moving Average')
plt.title('MA for AAPL Log Returns(20-day W)')
plt.show()
```



In [68]:
```
# MSFT
plt.plot(microsoft.rolling(20).mean())
plt.xlabel('Time')
plt.ylabel('Moving Average')
plt.title('MA for MSFT Log Returns(20-day W)')
plt.show()
```



In [69]:
```
# MAs
plt.plot(apple.rolling(20).mean())
plt.plot(microsoft.rolling(20).mean())
plt.xlabel('Time')
plt.ylabel('Moving Average')
plt.title('MA for AAPL & MSFT Log Returns(20-day W)')
plt.show()
```



Commit: \ The data I selected is from 2021/12/31 to 2023/1/23, we can see that these two technology companies follows the same track. Because of the pandemic the stock of both companies have floating around. Not only these two big companies, the whole market index are deeply decrease, lots of people lose their job there this period. Then the government begins to issure unemployment payments and other relief checks which bring up a little bit consumptions, as we can see there is a increase in the middle of that period. Shorting the stocks and index will make some profits as people spend all their cheks and payments, the consumption will back to the lowest point. Meanwhile, the employment will not recovery as fast as that time. Although there might be increase in stocks or index, but they are also not able to recovery to the highest point. And it will be the main trends at that time.

A3:

In [71]:
```
# A3
# import data
A_3 = pd.read_csv("2datasim.csv")
A_3.head()
```

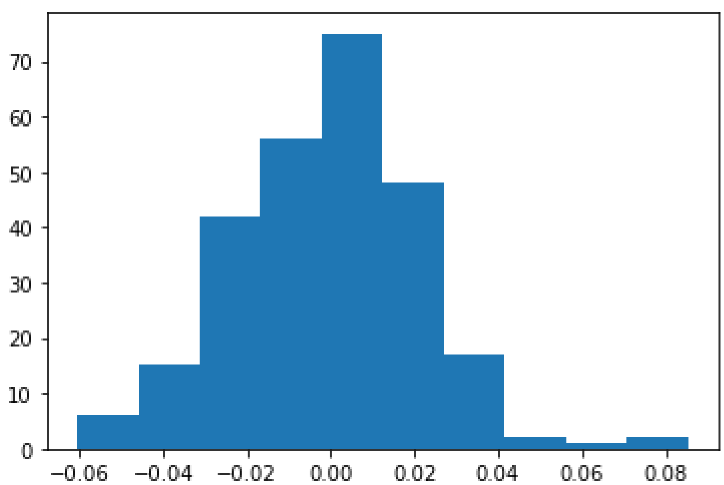
Out[71]:

	AAPL	MSFT
0	-0.012692	-0.017147
1	-0.026600	-0.038388
2	-0.016693	-0.007902
3	0.000988	0.000510
4	0.000116	0.000732

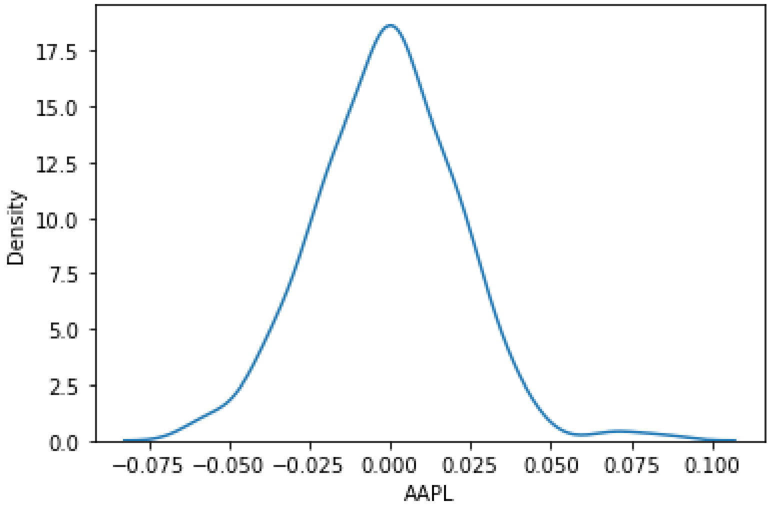
In [94]:
```
# AAPL
apple_sim = A_3['AAPL']
apple_sim.head()
```

Out[94]:
```
0   -0.012692
1   -0.026600
2   -0.016693
3    0.000988
4    0.000116
Name: AAPL, dtype: float64
```

In [132…
```
# Log return of AAPL

# histogram
plt.hist(apple)
plt.show()
```

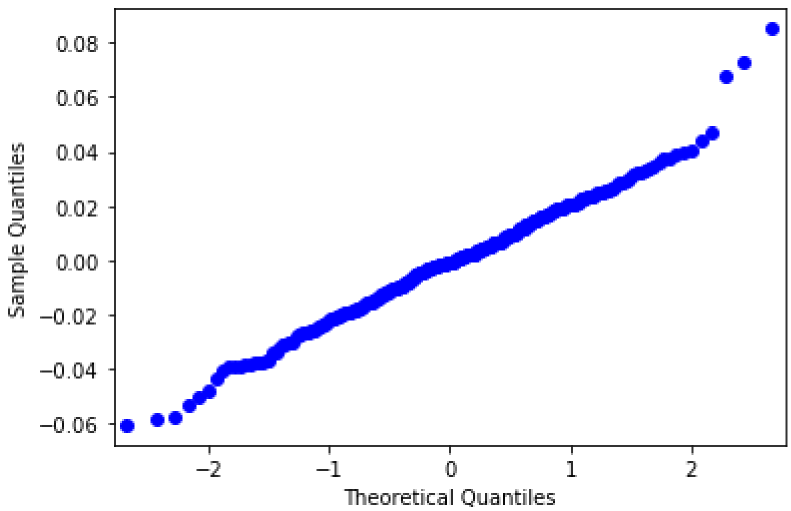


In [97]:
```
# Kernel density
sns.kdeplot(data=apple)
plt.show()
```



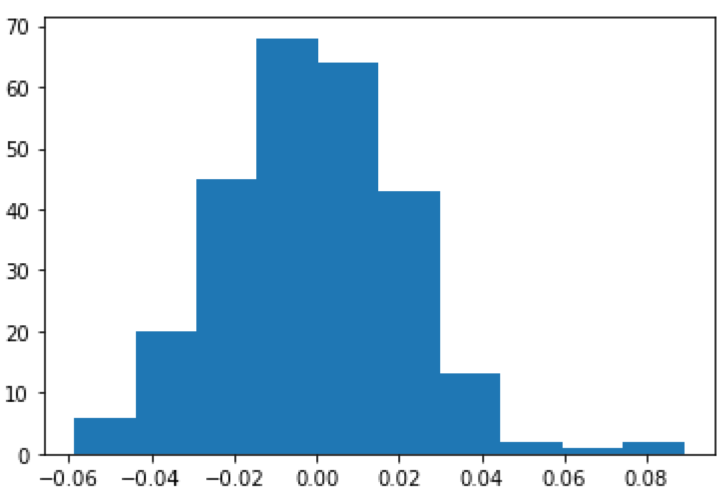
In [98]:

```
# QQ plot
sm.qqplot(apple)
plt.show()
```



In [131…

```
# Simple return of AAPL

# histogram
plt.hist(apple_sim)
plt.show()
```

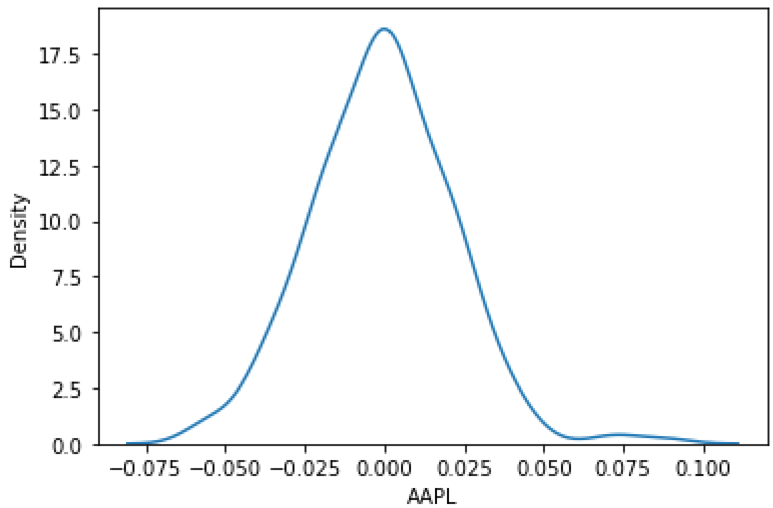


In [113…

```
# Kernel density
sns.kdeplot(data=apple_sim)
```

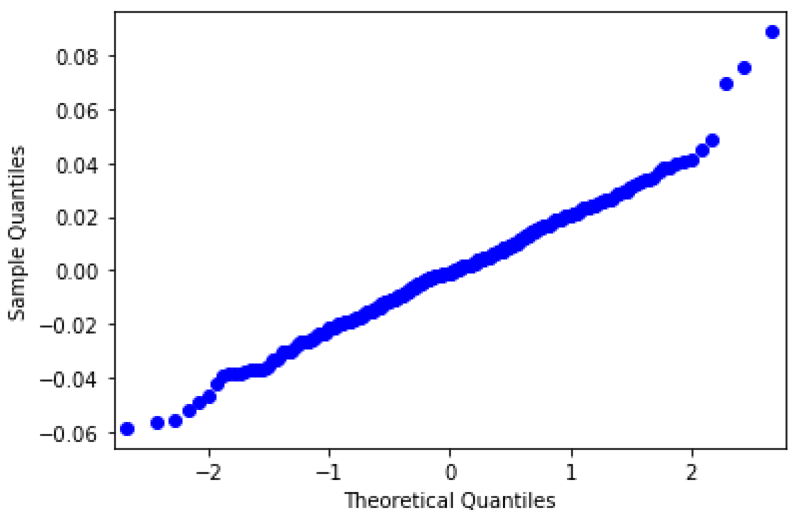
Out[113…

```
<AxesSubplot:xlabel='AAPL', ylabel='Density'>
```



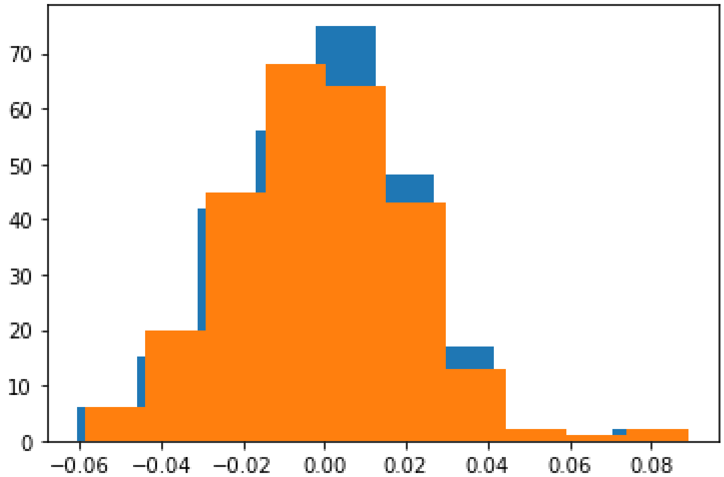
In [114…

```
# QQ plot
sm.qqplot(apple_sim)
plt.show()
```



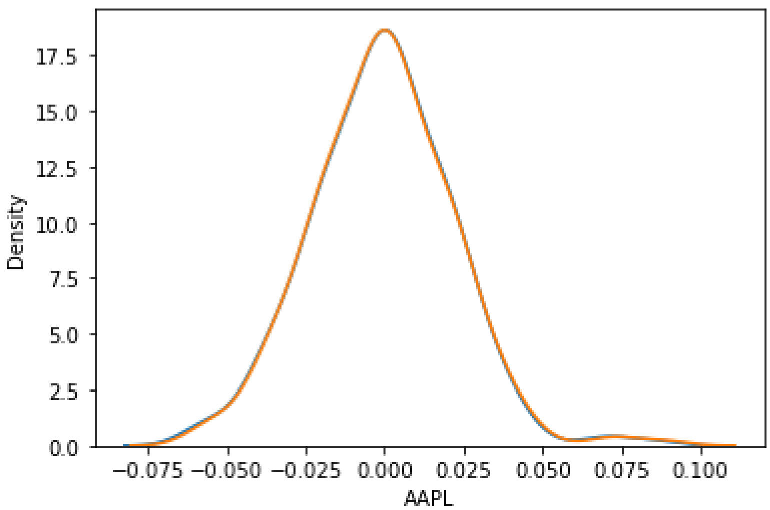
In [130…

```
# LOG AND SIMPLE

# histogram
plt.hist(apple)
plt.hist(apple_sim)
plt.show()
```



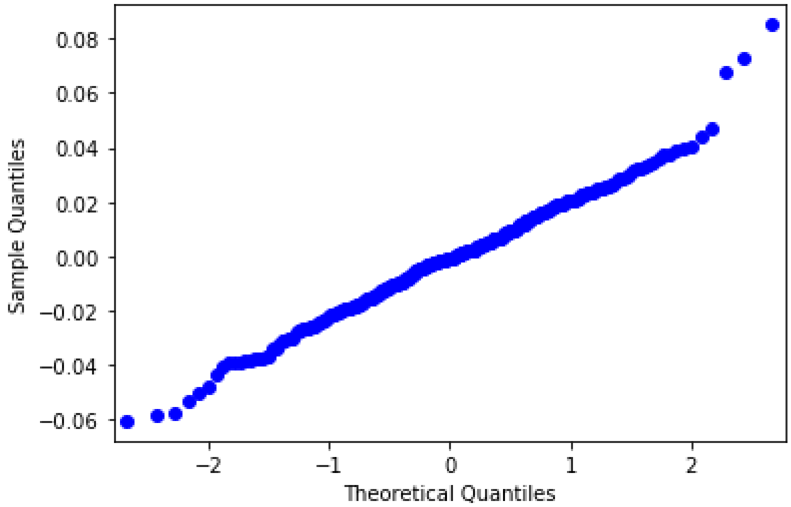
In [118…

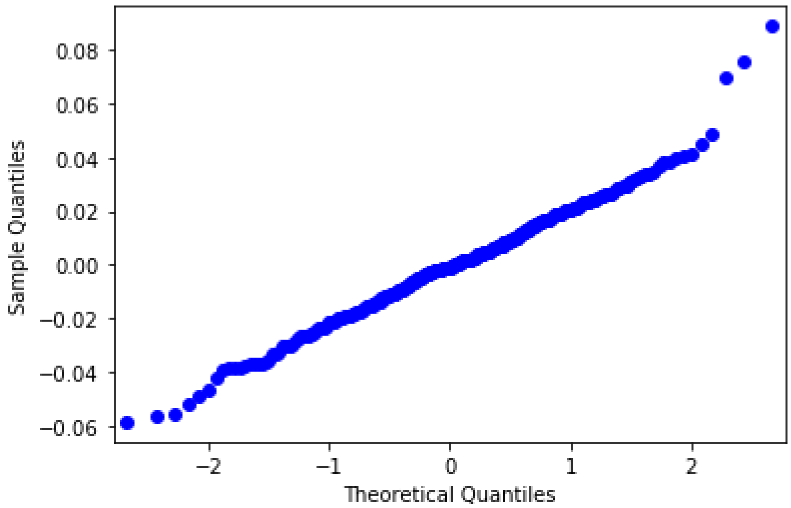
```
# LOG AND SIMPLE
# Kernel density
sns.kdeplot(data=apple)
sns.kdeplot(data=apple_sim)
plt.show()
```



In [136…

```
# LOG AND SIMPLE
# QQ Plot
sm.qqplot(apple)
sm.qqplot(apple_sim)
plt.show()
```





In [137…

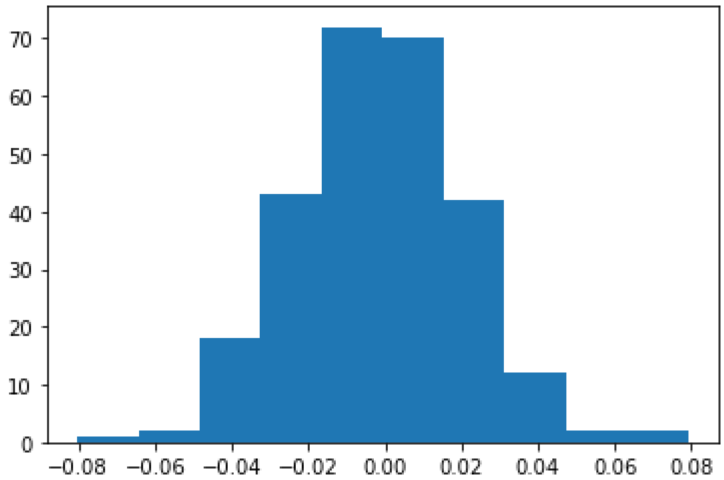
```
# MSFT
microsoft_sim = A_3['MSFT']
microsoft_sim.head()
```

Out[137…

```
0   -0.017147
1   -0.038388
2   -0.007902
3    0.000510
4    0.000732
Name: MSFT, dtype: float64
```

In [138…

```
# Log return of MSFT

# histogram
plt.hist(microsoft)
plt.show()
```

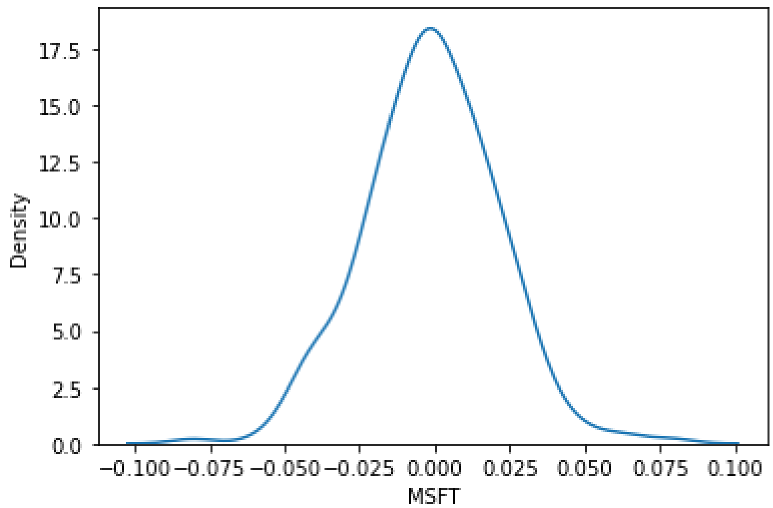


In [139…

```
# Kernel density
sns.kdeplot(data=microsoft)
```

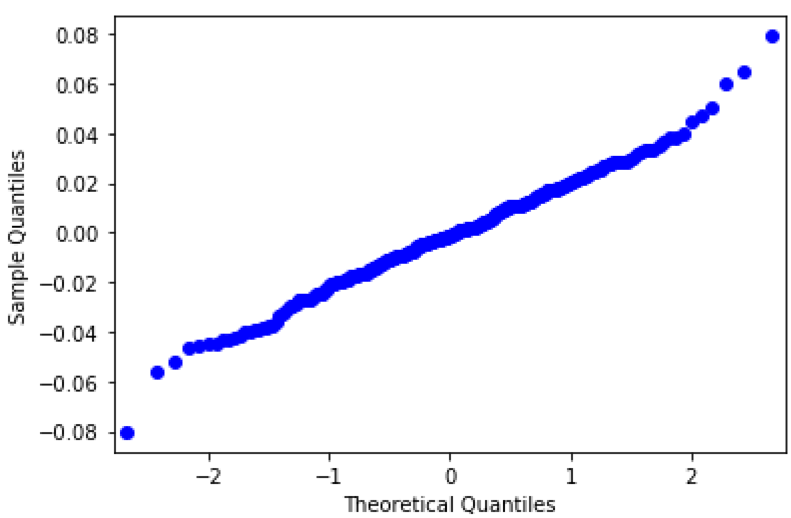
Out[139…

```
<AxesSubplot:xlabel='MSFT', ylabel='Density'>
```



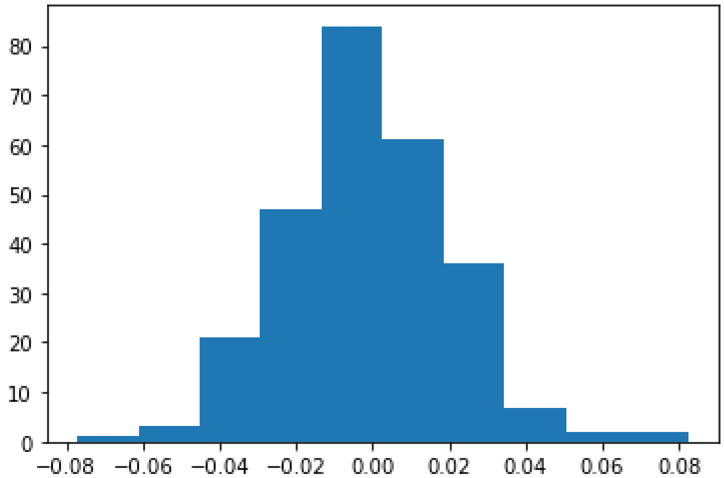
In [140…

```
# QQ Plot
sm.qqplot(microsoft)
plt.show()
```



In [141…

```
# Simple return of MSFT

# histogram
plt.hist(microsoft_sim)
plt.show()
```

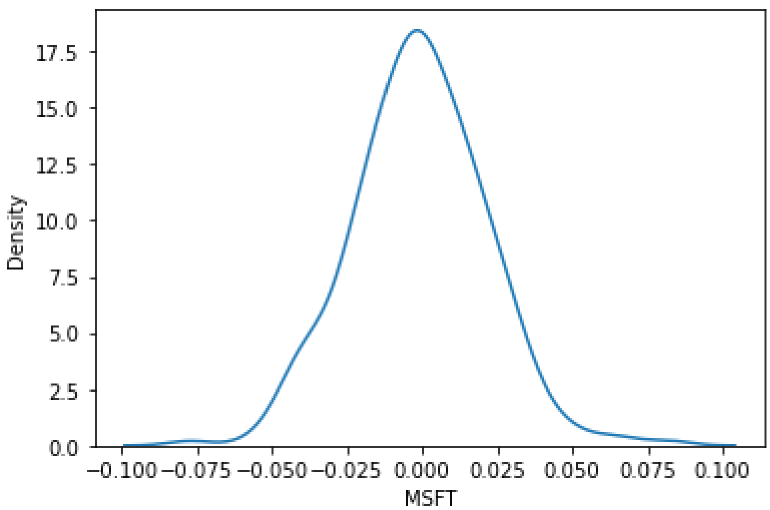


In [142…

```
# Kernel density
sns.kdeplot(data=microsoft_sim)
```

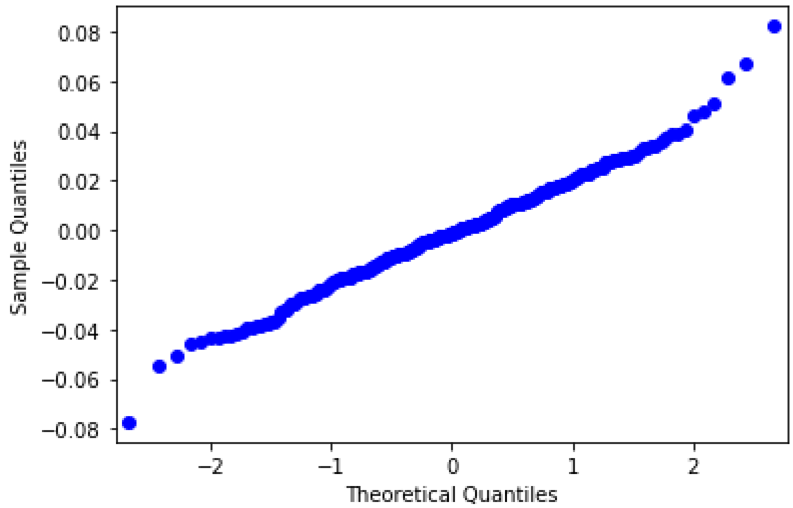
Out[142…

```
<AxesSubplot:xlabel='MSFT', ylabel='Density'>
```



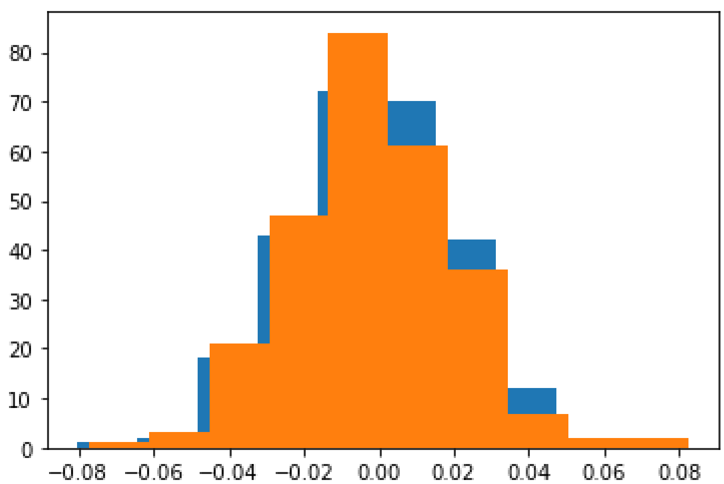
In [143…

```
# QQ Plot
sm.qqplot(microsoft_sim)
plt.show()
```



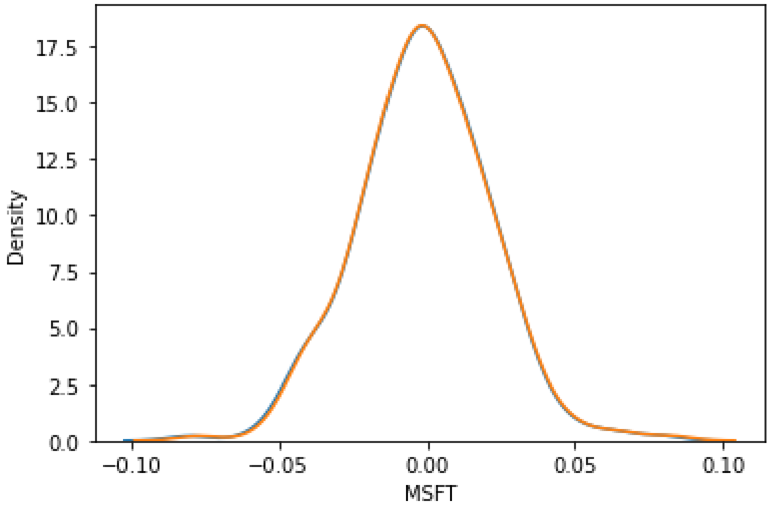
In [144…

```
# LOG AND SIMPLE
# histogram
plt.hist(microsoft)
plt.hist(microsoft_sim)
plt.show()
```



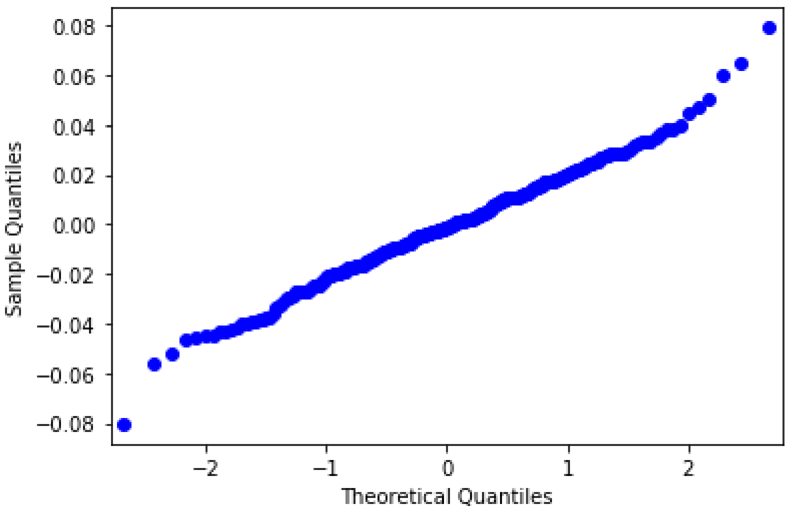
In [151…

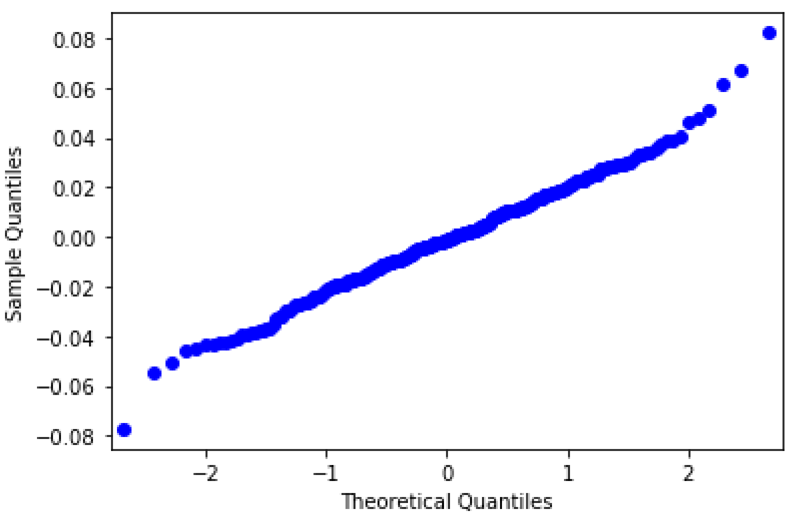
```
# LOG AND SIMPLE
# Kernel density
sns.kdeplot(data=microsoft)
sns.kdeplot(data=microsoft_sim)
plt.show()
```



In [147…

```
# LOG AND SIMPLE
# QQ Plot
sm.qqplot(microsoft)
sm.qqplot(microsoft_sim)
plt.show()
```





Comment: \ The simple return is very close to the log return in both Apple and Microsoft stocks. There are rare differences on the Kernel density plot and QQ plot, but we can see the small differences in the histogram plot. It is probably better to use log return for the normal distribution property, which fits better to the assumptions of lots of pricing models, However, log return may fail on moments calculations.

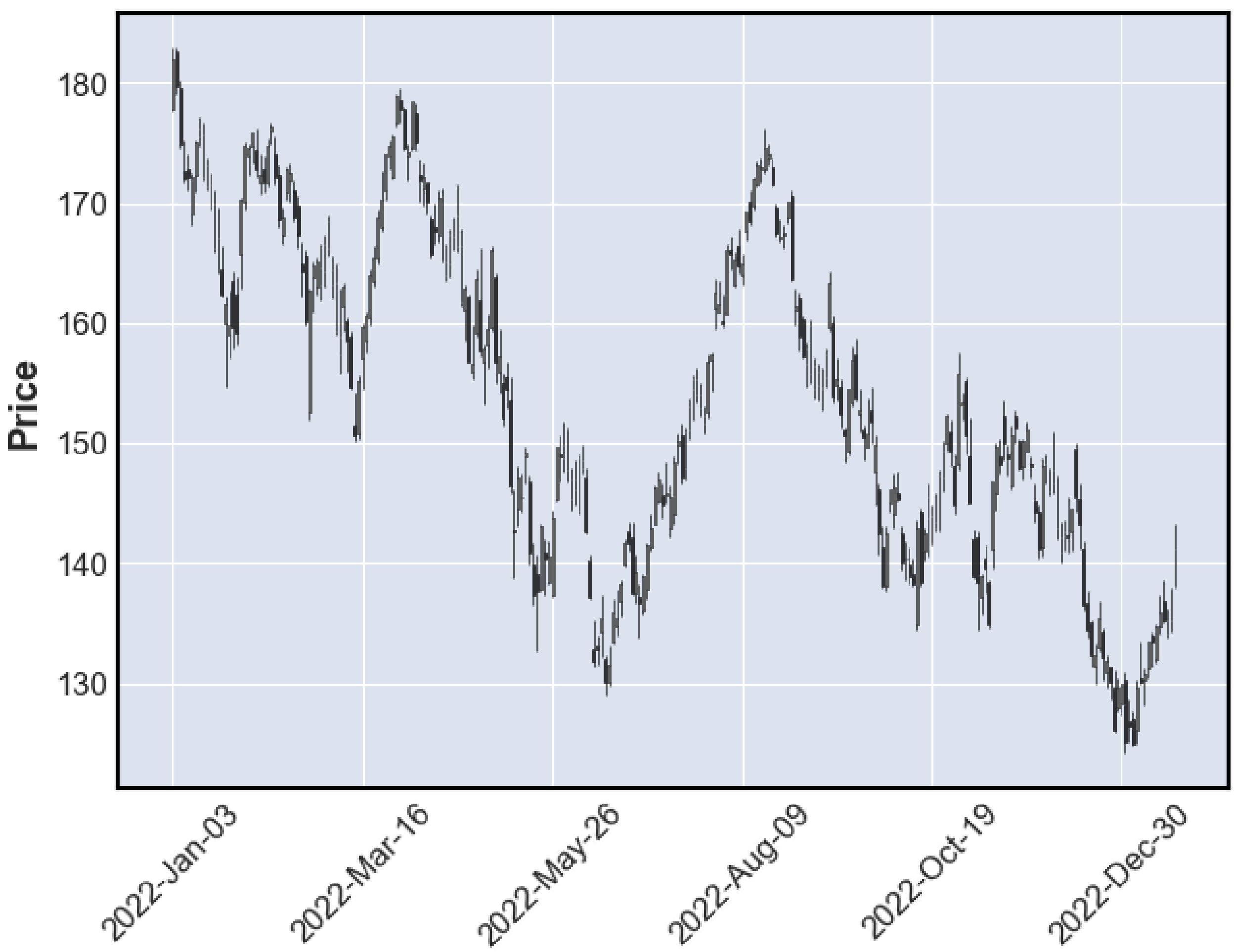
B1:

In [55]:
```
# Candlestick chart
# AAPL
aapl_b = pd.read_csv("AAPL.csv")
aapl_b.head()
```

Out[55]:

	Date	Open	High	Low	Close	Adj Close	Sim return	Log return
0	2022/1/3	177.830002	182.880005	177.710007	182.009995	180.959732	NaN	NaN
1	2022/1/4	182.630005	182.940002	179.119995	179.699997	178.663071	-0.012692	-0.012773
2	2022/1/5	179.610001	180.169998	174.639999	174.919998	173.910660	-0.026600	-0.026960
3	2022/1/6	172.699997	175.300003	171.639999	172.000000	171.007507	-0.016693	-0.016834
4	2022/1/7	172.889999	174.139999	171.029999	172.169998	171.176514	0.000988	0.000988

In [69]:
```
# Use mpf tool to plot candle chart
# Change date form type object to type datetime
aapl_b.Date = pd.to_datetime(aapl_b.Date)
aapl_b = aapl_b.set_index(aapl_b['Date'])
mpf.plot(aapl_b,type='candle')
```

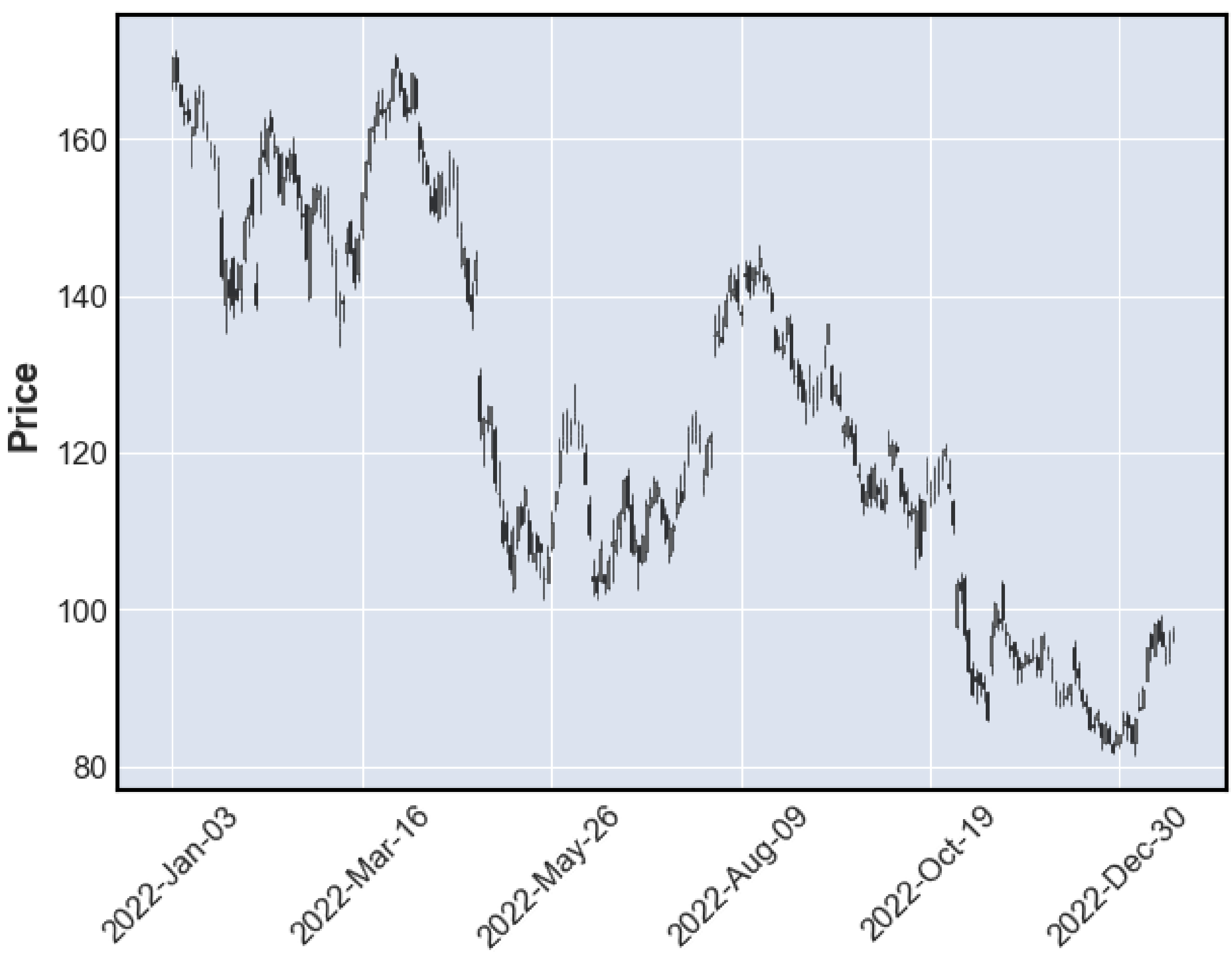


In [65]:
```
# AMZN
amzn_b = pd.read_csv("AMZN.csv")
amzn_b.head()
```

Out[65]:

	Date	Open	High	Low	Close	Adj Close	Sim return	Log return
0	2022/1/3	167.550003	170.703506	166.160507	170.404495	170.404495	NaN	NaN
1	2022/1/4	170.438004	171.399994	166.349503	167.522003	167.522003	-0.016916	-0.017060
2	2022/1/5	166.882996	167.126495	164.356995	164.356995	164.356995	-0.018893	-0.019074
3	2022/1/6	163.450500	164.800003	161.936996	163.253998	163.253998	-0.006711	-0.006734
4	2022/1/7	163.839005	165.243500	162.031006	162.554001	162.554001	-0.004288	-0.004297

In [70]:
```
# Use mpf tool to plot candle chart
# Change date form type object to type datetime
amzn_b.Date = pd.to_datetime(amzn_b.Date)
amzn_b = amzn_b.set_index(amzn_b['Date'])
mpf.plot(amzn_b,type='candle')
```



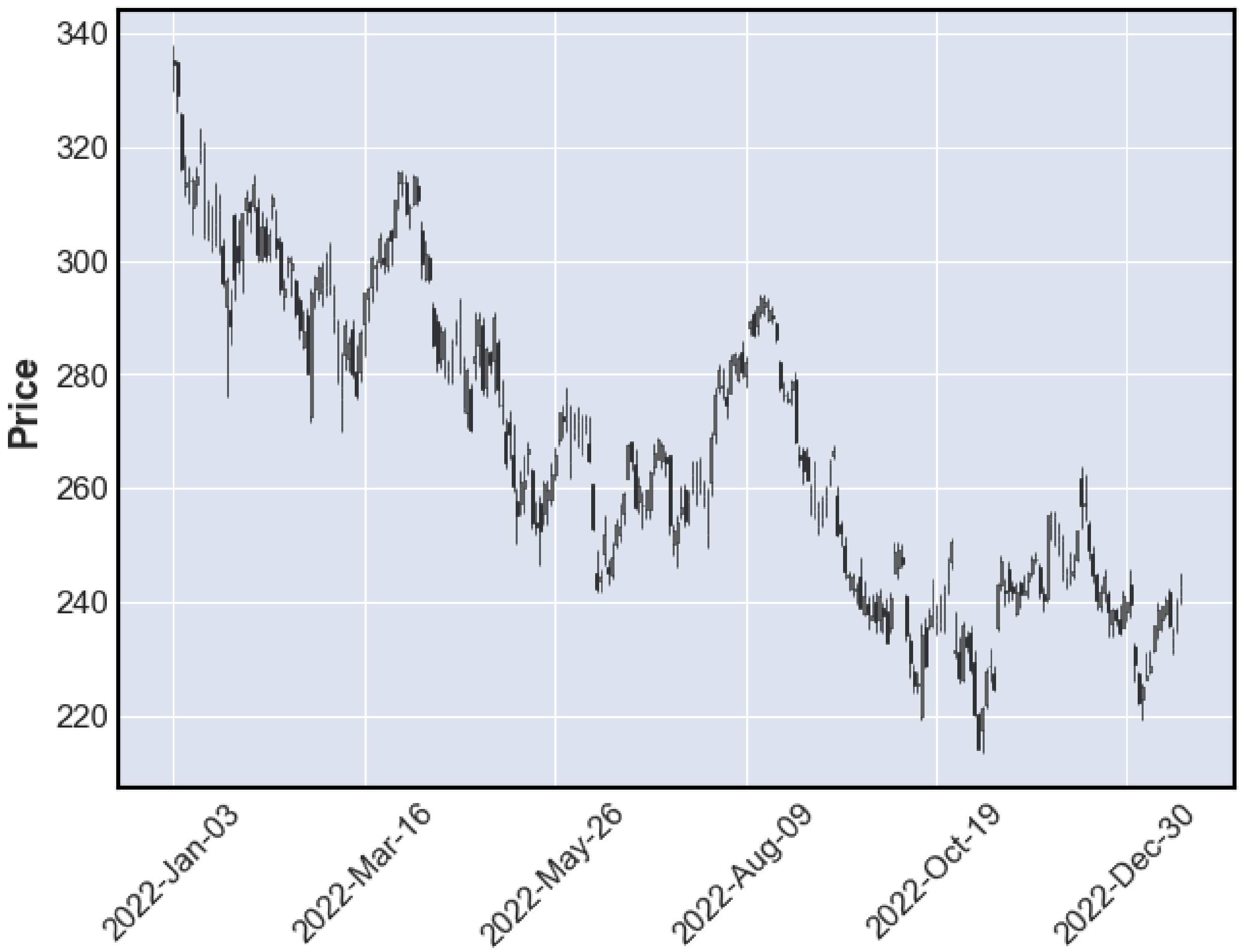
In [67]:
```
# AMZN
msft_b = pd.read_csv("MSFT.csv")
msft_b.head()
```

Out[67]:

	Date	Open	High	Low	Close	Adj Close	Sim return	Log return
0	2022/1/3	335.350006	338.000000	329.779999	334.750000	331.642456	NaN	NaN
1	2022/1/4	334.829987	335.200012	326.119995	329.010010	325.955750	-0.017147	-0.017296
2	2022/1/5	325.859985	326.070007	315.980011	316.380005	313.443024	-0.038388	-0.039144
3	2022/1/6	313.149994	318.700012	311.489990	313.880005	310.966217	-0.007902	-0.007933
4	2022/1/7	314.149994	316.500000	310.089996	314.040009	311.124725	0.000510	0.000510

In [71]:

```
# Use mpf tool to plot candle chart
# Change date form type object to type datetime
msft_b.Date = pd.to_datetime(msft_b.Date)
msft_b = msft_b.set_index(msft_b['Date'])
mpf.plot(msft_b,type='candle')
```



B2:

In [179…

```
# S&P500
sp500_b = pd.read_csv("SP500.csv")
sp500_b.head()
adj_close = pd.read_csv("Adjclose.csv")
adj_close.head()
```

Out[179...

	Date	AAPL Adj Close	AMZN Adj Close	MSFT Adj Close	SP500 Adj Close
0	2022/1/3	180.959732	170.404495	331.642456	4,019.81
1	2022/1/4	178.663071	167.522003	325.955750	3,972.61
2	2022/1/5	173.910660	164.356995	313.443024	3,898.85
3	2022/1/6	171.007507	163.253998	310.966217	3,928.86
4	2022/1/7	171.176514	162.554001	311.124725	3,990.97

In [211...

```
# AAPL
# Covariance
aapl_cov = np.cov(aapl_b['Log return'][1:])
aapl_cov

# Variance
aapl_var = np.var(aapl_b['Log return'])
aapl_var

# Beta
aapl_beta = aapl_cov/aapl_var
print('AAPL Deta: '+ str(aapl_beta))
```

```
AAPL Deta: 1.0038022813688208
```

In [ ]:

In [210...

```
# AMZN
# Covariance
amzn_cov = np.cov(amzn_b['Log return'][1:])
amzn_cov

# Variance
amzn_var = np.var(amzn_b['Log return'])
amzn_var

# Beta
amzn_beta = amzn_cov/amzn_var
print('AMZN Deta: '+ str(amzn_beta))
```

```
AMZN Deta: 1.0038022813688214
```

In [ ]:

In [212...

```
# MSFT
# Covariance
msft_cov = np.cov(msft_b['Log return'][1:])
msft_cov

# Variance
msft_var = np.var(msft_b['Log return'])
msft_var

# Beta
msft_beta = msft_cov/amzn_var
print('MSFT Deta: '+ str(msft_beta))
```

```
MSFT Deta: 0.5035990169052097
```

In [ ]:

Commend: \ The Beta value of these three stocks is quiet close to the real beta that these company have. The difference casue by the data amount, as the Beta show on the Yahoo finance is 5 year Beta, the amount I apply for my calculation is one year data. I believe if we have enough data, the answer will be much closer to the official data publish online.

B3:

In [187…

```
# AAPL
# Sharpe ratio
# 0.005 is the average risk free rate from 2022/01/03 to the end of 2022 calcula
sharpe_ratio_aapl = (aapl_b['Log return'][1:].mean()-0.005)/aapl_b['Log return']
print('AAPL Sharpe ratio is '+ str(sharpe_ratio_aapl))

# Treynor’s measure
treynor_aapl = (aapl_b['Log return'][1:].mean()-0.005)/aapl_beta
print('AAPL Treynor’s measure is '+ str(treynor_aapl))

# Jensen’s alpha
alpha_aapl = (0.005 + aapl_beta * (sp500_b['Log return'][1:].mean()-0.005))
print('AAPL Jensen’s alpha is '+ str(alpha_aapl))
```

```
AAPL Sharpe ratio is -0.2672036042682617
AAPL Treynor’s measure is -0.00594217725757576
AAPL Jensen’s alpha is 0.0006691831893939409
```

In [186…

```
# AMZN
# Sharpe ratio
sharpe_ratio_amzn = (amzn_b['Log return'][1:].mean()-0.005)/amzn_b['Log return']
print('AMZN Sharpe ratio is '+ str(sharpe_ratio_amzn))

# Treynor’s measure
treynor_amzn = (amzn_b['Log return'][1:].mean()-0.005)/amzn_beta
print('AMZN Treynor’s measure is '+ str(treynor_amzn))

# Jensen’s alpha
alpha_amzn = (0.005 + amzn_beta * (sp500_b['Log return'][1:].mean()-0.005))
print('AMZN Jensen’s alpha is '+ str(alpha_amzn))
```

```
AMZN Sharpe ratio is -0.22662214428524952
AMZN Treynor’s measure is -0.007114081583333331
AMZN Jensen’s alpha is 0.0006691831893939383
```

In [189…

```
# MSFT
# Sharpe ratio
sharpe_ratio_msft = (msft_b['Log return'][1:].mean()-0.005)/msft_b['Log return']
print('MSFT Sharpe ratio is '+ str(sharpe_ratio_msft))

# Treynor’s measure
treynor_msft = (msft_b['Log return'][1:].mean()-0.005)/msft_beta
print('MSFT Treynor’s measure is '+ str(treynor_msft))

# Jensen’s alpha
alpha_msft = (0.005 + msft_beta * (sp500_b['Log return'][1:].mean()-0.005))
print('MSFT Jensen’s alpha is '+ str(alpha_msft))
```

```
MSFT Sharpe ratio is -0.27814708141756783
MSFT Treynor’s measure is -0.012327433160380534
```

```
MSFT Jensen's alpha is 0.00282726625681336
```

Commend: \ The Sharpe ratio and Treynor's measure that I got are negative which indicates that the investment has performed worse than a risk free instrument, also a positive alpha indicates the security is outperforming the market. The. investors are lossing money in these three stocks. Shorting will be the main trends during last year in stocks and market indexes.

B4:

In [209…

```
# AAPL AMZN MSFT RF
# Sharpe ratio
# Equal weight 33.33%
profolio_return = 0.3333 * aapl_b['Log return'][1:].mean() + 0.3333 * amzn_b['Lo
profolio_std = np.sqrt(0.3333**2 * aapl_b['Log return'][1:].var() + 0.3333**2 *
sharpe_ratio_profolio = (profolio_return-0.005)/profolio_std
print('Equal weight profolio Sharpe ratio is ' + str(sharpe_ratio_profolio))
```

```
Equal weight profolio Sharpe ratio is -0.012861205043746875
```

Commends \ We can see that the equal weight profolio Sharpe ratio is negaitive. This means that portfolio's return is expected to be negative. The reason I think is that it relate to the time period I choose, during this time period, market and companies are all at the lowest point. So it is reasonable that the return of investing this equal weight profolio is negative.

B5:

In [250…

```
# covariance matrix
# AAPL AMZN MSFT Log return
data = np.array([aapl_b['Log return'][1:], amzn_b['Log return'][1:], msft_b['Log
data_cov = pd.DataFrame(data)
cov_log = data_cov.cov()
print('Covariance matrix of Log return:')
cov_log
```

```
Covariance matrix of Log return:
```

Out[250…

	0	1	2	3	4	5	6	7
0	0.000006	4.337261e-06	-1.388055e-05	0.000004	0.000004	0.000007	-0.000004	0.00002
1	0.000004	1.022424e-04	-7.858039e-07	-0.000022	-0.000035	0.000110	-0.000058	0.00010
2	-0.000014	-7.858039e-07	3.044823e-05	-0.000011	-0.000012	-0.000005	0.000003	-0.00004
3	0.000004	-2.205328e-05	-1.056720e-05	0.000009	0.000012	-0.000022	0.000012	-0.00000
4	0.000004	-3.456660e-05	-1.184371e-05	0.000012	0.000017	-0.000036	0.000019	-0.00002
...	...	...	...	...	...	...	...	..
259	0.000023	-1.182131e-04	-6.177336e-05	0.000048	0.000065	-0.000119	0.000061	-0.00003
260	0.000011	6.994736e-05	-1.902941e-05	-0.000009	-0.000016	0.000078	-0.000042	0.00009

	0	1	2	3	4	5	6	7
261	0.000027	1.803155e-06	-5.815027e-05	0.000020	0.000023	0.000010	-0.000007	0.00007ℇ
262	-0.000025	-8.925436e-07	5.522359e-05	-0.000019	-0.000022	-0.000009	0.000006	-0.00007ℨ
263	0.000024	-2.299818e-05	-5.579328e-05	0.000025	0.000030	-0.000017	0.000008	0.00005(

264 rows × 264 columns

In [251…

```
# correlation matrix
data_cor = pd.DataFrame(data)
corr_log = data_cor.corr()
print('Correlation matrix of Log return:')
corr_log
```

```
Correlation matrix of Log return:
```

Out[251…

	0	1	2	3	4	5	6	7	
0	1.000000	0.168470	-0.987982	0.530530	0.390764	0.238220	-0.259929	0.702830	-(
1	0.168470	1.000000	-0.014084	-0.746171	-0.841502	0.997462	-0.995616	0.819597	-(
2	-0.987982	-0.014084	1.000000	-0.655179	-0.528349	-0.085236	0.107547	-0.584427	(
3	0.530530	-0.746171	-0.655179	1.000000	0.987581	-0.696880	0.680629	-0.230122	(
4	0.390764	-0.841502	-0.528349	0.987581	1.000000	-0.800904	0.787281	-0.380159	(
...	...	...	...	...	...	...	...	...	
259	0.566122	-0.717164	-0.686734	0.999094	0.980000	-0.665729	0.648836	-0.188501	(
260	0.581394	0.899940	-0.448644	-0.381232	-0.521744	0.928698	-0.936777	0.987398	-(
261	0.988416	0.016922	-0.999996	0.653032	0.525937	0.088063	-0.110368	0.586728	-(
262	-0.987154	-0.008820	0.999986	-0.659147	-0.532811	-0.079989	0.102312	-0.580148	0
263	0.924969	-0.218781	-0.972596	0.812873	0.711270	-0.148756	0.126553	0.379749	-(

264 rows × 264 columns

In [252…

```
# covariance matrix
# AAPL AMZN MSFT Sim return
data = np.array([aapl_b['Sim return'][1:], amzn_b['Sim return'][1:], msft_b['Sim
data_cov = pd.DataFrame(data)
cov_log = data_cov.cov()
print('Covariance matrix of Sim return:')
cov_log
```

```
Covariance matrix of Sim return:
```

Out[252…

	0	1	2	3	4	5	6	7
0	0.000006	4.080231e-06	-1.351011e-05	0.000004	0.000004	0.000007	-0.000004	0.00002:
1	0.000004	9.639902e-05	-5.806281e-07	-0.000021	-0.000034	0.000108	-0.000057	0.00009ℇ

	0	1	2	3	4	5	6	7
2	-0.000014	-5.806281e-07	2.972588e-05	-0.000010	-0.000012	-0.000005	0.000003	-0.000039
3	0.000004	-2.142435e-05	-1.043146e-05	0.000009	0.000012	-0.000023	0.000012	-0.000008
4	0.000004	-3.352601e-05	-1.167131e-05	0.000012	0.000016	-0.000036	0.000019	-0.000019
...	...	...	...	...	...	...	...	..
259	0.000023	-1.140966e-04	-6.075438e-05	0.000047	0.000064	-0.000119	0.000061	-0.000037
260	0.000011	6.698290e-05	-1.856427e-05	-0.000008	-0.000016	0.000078	-0.000041	0.000092
261	0.000026	1.553464e-06	-5.695537e-05	0.000020	0.000022	0.000010	-0.000007	0.000075
262	-0.000025	-5.174552e-07	5.610329e-05	-0.000020	-0.000022	-0.000009	0.000006	-0.000073
263	0.000024	-2.265026e-05	-5.591705e-05	0.000025	0.000030	-0.000017	0.000008	0.000050

264 rows × 264 columns

In [253…

```
# correlation matrix
data_cor = pd.DataFrame(data)
corr_log = data_cor.corr()
print('Correlation matrix of Sim return:')
corr_log
```

Correlation matrix of Sim return:

Out[253…

	0	1	2	3	4	5	6	7	
0	1.000000	0.165683	-0.987918	0.530791	0.390613	0.235556	-0.261116	0.704286	-0
1	0.165683	1.000000	-0.010847	-0.747847	-0.843114	0.997456	-0.995228	0.816793	-(
2	-0.987918	-0.010847	1.000000	-0.655721	-0.528558	-0.082094	0.108362	-0.585756	0
3	0.530791	-0.747847	-0.655721	1.000000	0.987507	-0.698624	0.679503	-0.227828	0
4	0.390613	-0.843114	-0.528558	0.987507	1.000000	-0.802640	0.786623	-0.378414	(
...	...	...	...	...	...	...	...	...	
259	0.567626	-0.717862	-0.688358	0.999028	0.979603	-0.666412	0.646508	-0.184694	0
260	0.581585	0.898601	-0.448487	-0.380732	-0.521683	0.927591	-0.937125	0.987109	-0
261	0.988575	0.015146	-0.999991	0.652469	0.524904	0.086378	-0.112635	0.589236	-0.
262	-0.987015	-0.005122	0.999984	-0.660033	-0.533410	-0.076387	0.102668	-0.581106	0.
263	0.925981	-0.218933	-0.973308	0.811494	0.709275	-0.148826	0.122682	0.384112	-0

264 rows × 264 columns

Commend: \ The covariance and correlation matrix of the Log return and Sim return is similar to each other. as the difference from the original data is very small.

C1:

In [2]:
```
# Import data
industry_c = pd.read_csv("10industryprofolios.csv")
industry_c.head()
```

Out[2]:

	Date	NoDur	Durbl	Manuf	Enrgy	HiTec	Telcm	Shops	Hlth	Utils	Other
0	200012	5.47	1.07	7.30	7.61	-7.92	-4.23	5.70	3.53	6.58	7.20
1	200101	-1.92	13.20	-1.01	-3.40	16.46	14.27	6.52	-8.96	-10.77	0.17
2	200102	0.18	-1.25	-3.66	0.05	-25.91	-9.95	-5.34	-0.48	6.24	-4.41
3	200103	-4.65	-3.06	-6.18	-0.14	-13.59	-5.86	-2.82	-8.44	1.64	-4.29
4	200104	0.91	8.44	7.43	10.07	17.93	2.67	5.63	3.98	5.46	5.94

In [257…
```
print('Enrgy:')
# mean
mean_enrgy_c = industry_c['Enrgy'].mean()
print("mean is "+ str(mean_enrgy_c))

# median
median_enrgy_c = industry_c['Enrgy'].median()
print("median is "+ str(median_enrgy_c))

# variance
variance_enrgy_c = st.variance(industry_c['Enrgy'])
print("variance is "+ str(variance_enrgy_c))

# std
std_enrgy_c = industry_c['Enrgy'].std()
print("standard deviation is "+ str(std_enrgy_c))

# skewnwss
from scipy.stats import skew
skw_enrgy_c = skew(industry_c['Enrgy'])
print("skewnwss is "+ str(skw_enrgy_c))

# kurtosis
from scipy.stats import kurtosis
kur_enrgy_c = kurtosis(industry_c['Enrgy'])
print("kurtosis is "+ str(kur_enrgy_c))
```

```
Enrgy:
mean is 0.9653030303030302
median is 0.95
variance is 53.67454971770942
standard deviation is 7.326291675718994
skewnwss is 0.07459777004918745
kurtosis is 3.599671394555795
```

In [259…
```
print('HiTec:')
# mean
mean_hitec_c = industry_c['HiTec'].mean()
print("mean is "+ str(mean_hitec_c))

# median
median_hitec_c = industry_c['HiTec'].median()
```

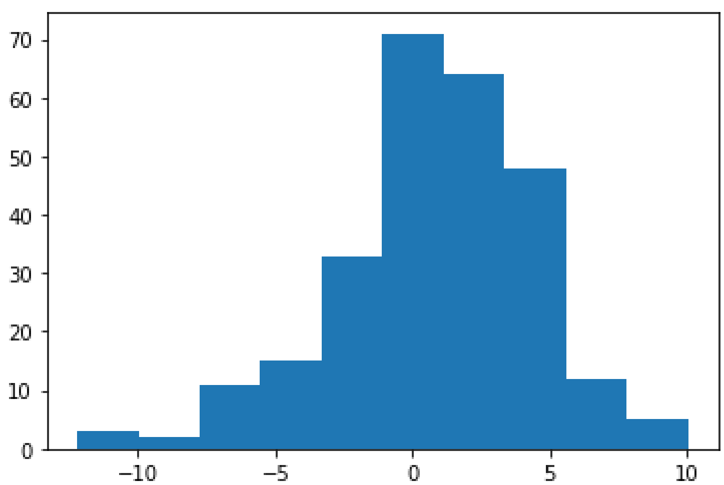
```
print("median is "+ str(median_hitec_c))

# variance
variance_hitec_c = st.variance(industry_c['HiTec'])
print("variance is "+ str(variance_hitec_c))

# std
std_hitec_c = industry_c['HiTec'].std()
print("standard deviation is "+ str(std_hitec_c))

# skewnwss
from scipy.stats import skew
skw_hitec_c = skew(industry_c['HiTec'])
print("skewnwss is "+ str(skw_hitec_c))

# kurtosis
from scipy.stats import kurtosis
kur_hitec_c = kurtosis(industry_c['HiTec'])
print("kurtosis is "+ str(kur_hitec_c))
```

```
HiTec:
mean is 0.8353787878787877
median is 1.37
variance is 42.31101582555594
standard deviation is 6.504691831713164
skewnwss is -0.45188553957843247
kurtosis is 1.3186040349032702
```

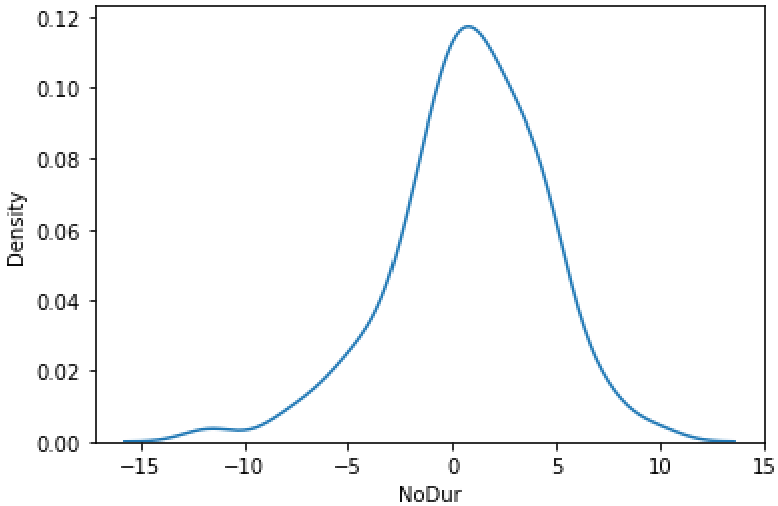
Commend: \ The two industries that I choose is not related to each other, as we can see that their index are reflected oppersitely.

C2:

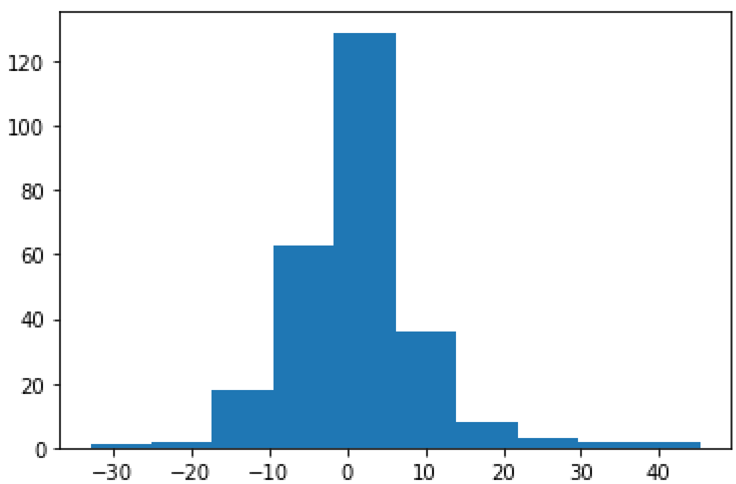
In [15]:
```
# NoDur
# histogram
plt.hist(industry_c['NoDur'])
plt.show()
# Kernel density
sns.kdeplot(data=industry_c['NoDur'])
```



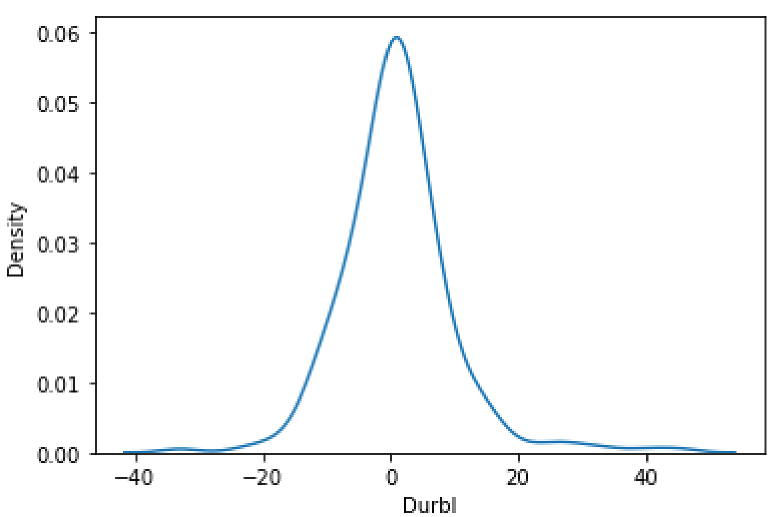
Out[15]: `<AxesSubplot:xlabel='NoDur', ylabel='Density'>`



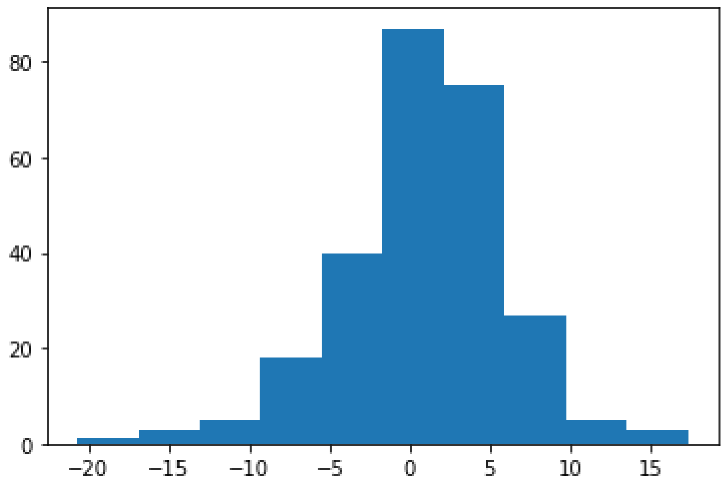
In [16]:
```
# Durbl
# histogram
plt.hist(industry_c['Durbl'])
plt.show()
# Kernel density
sns.kdeplot(data=industry_c['Durbl'])
```



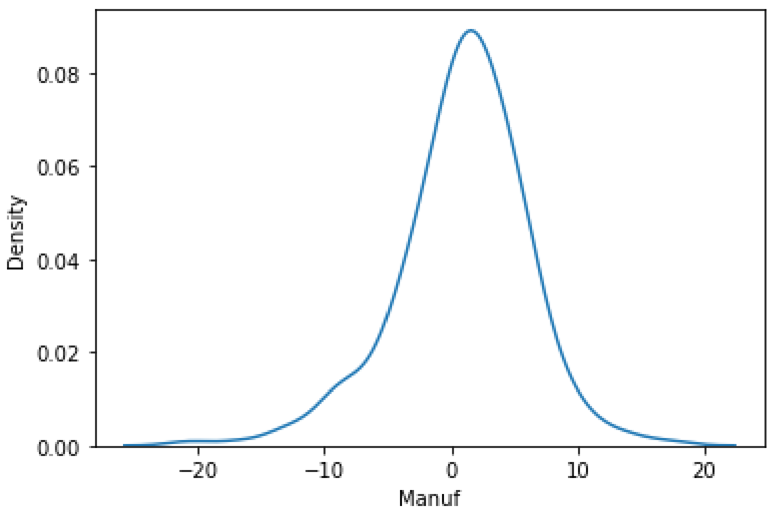
Out[16]: <AxesSubplot:xlabel='Durbl', ylabel='Density'>



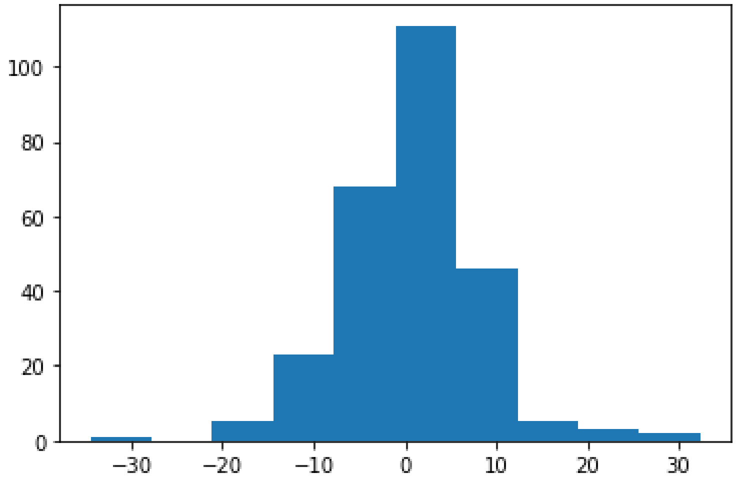
In [17]:
```
# Manuf
plt.hist(industry_c['Manuf'])
plt.show()
# Kernel density
sns.kdeplot(data=industry_c['Manuf'])
```



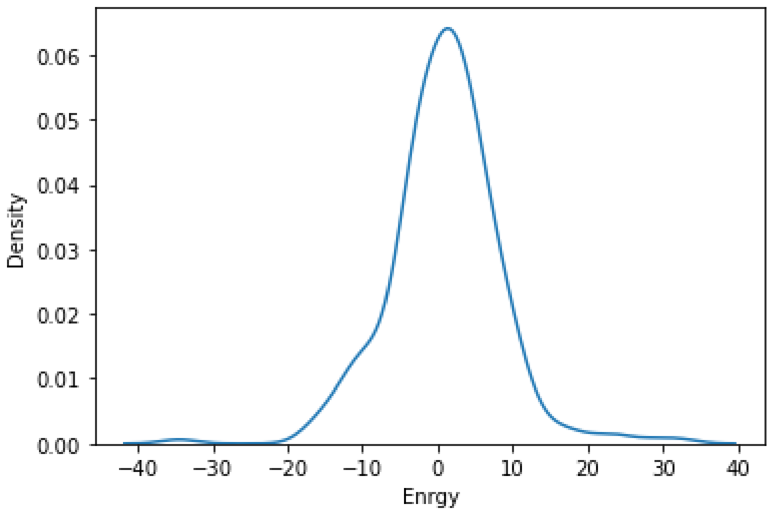
Out[17]: <AxesSubplot:xlabel='Manuf', ylabel='Density'>



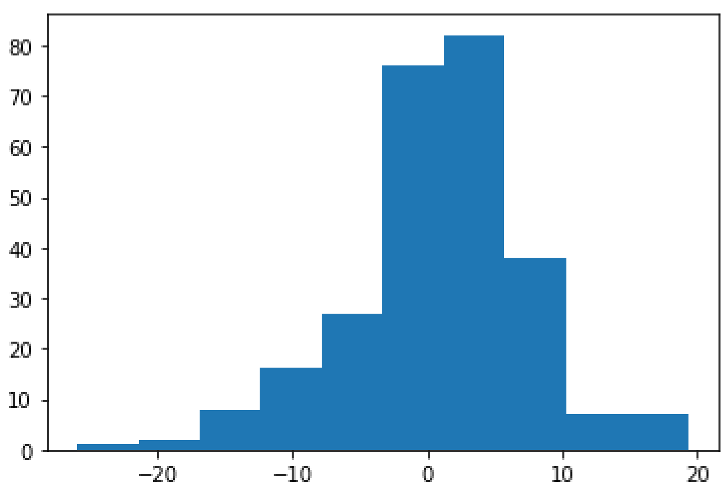
In [18]:
```
# Enrgy
plt.hist(industry_c['Enrgy'])
plt.show()
# Kernel density
sns.kdeplot(data=industry_c['Enrgy'])
```



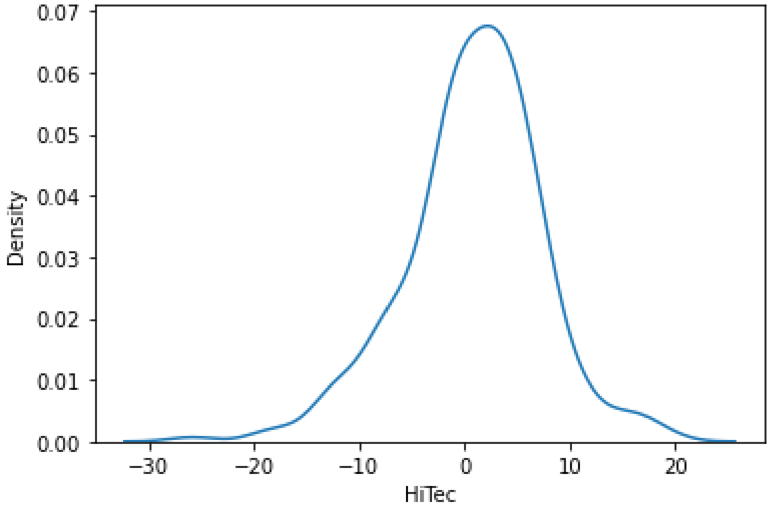
Out[18]: <AxesSubplot:xlabel='Enrgy', ylabel='Density'>



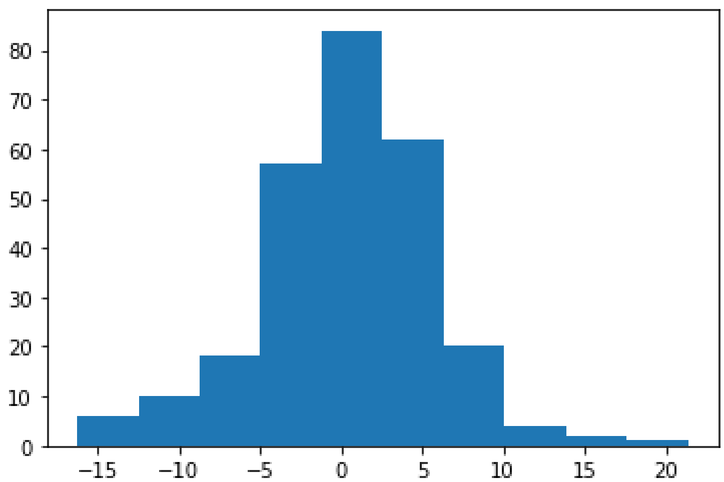
In [19]:
```
# HiTec
plt.hist(industry_c['HiTec'])
plt.show()
# Kernel density
sns.kdeplot(data=industry_c['HiTec'])
```



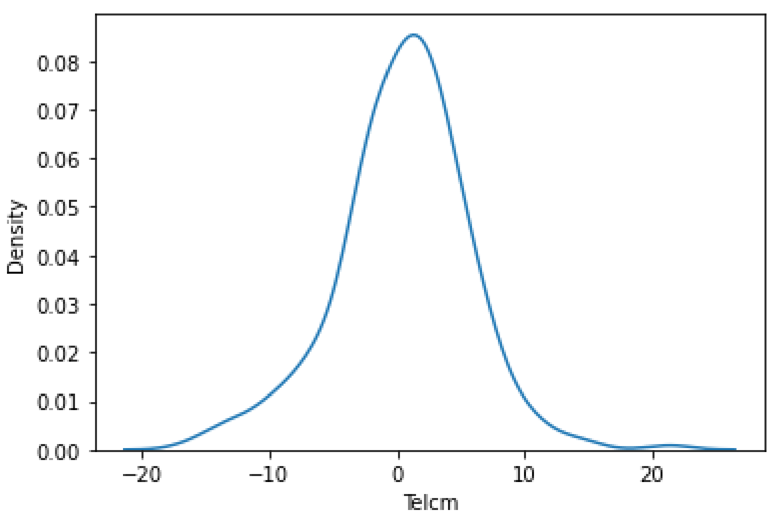
Out[19]: <AxesSubplot:xlabel='HiTec', ylabel='Density'>



In [20]:
```
# Telcm
plt.hist(industry_c['Telcm'])
plt.show()
# Kernel density
sns.kdeplot(data=industry_c['Telcm'])
```

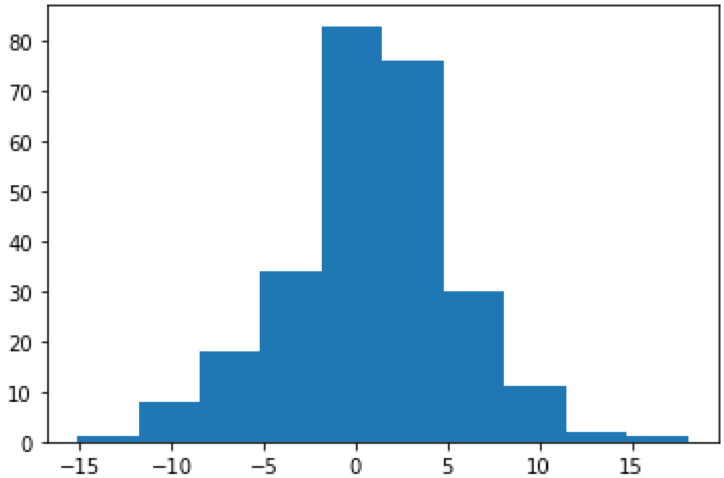


Out[20]: <AxesSubplot:xlabel='Telcm', ylabel='Density'>

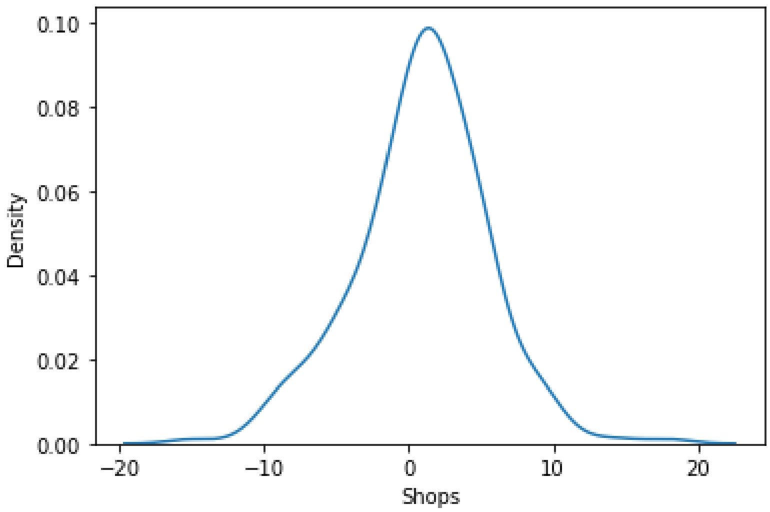


In [21]:
```
# Shops
plt.hist(industry_c['Shops'])
```

```
plt.show()
# Kernel density
sns.kdeplot(data=industry_c['Shops'])
```

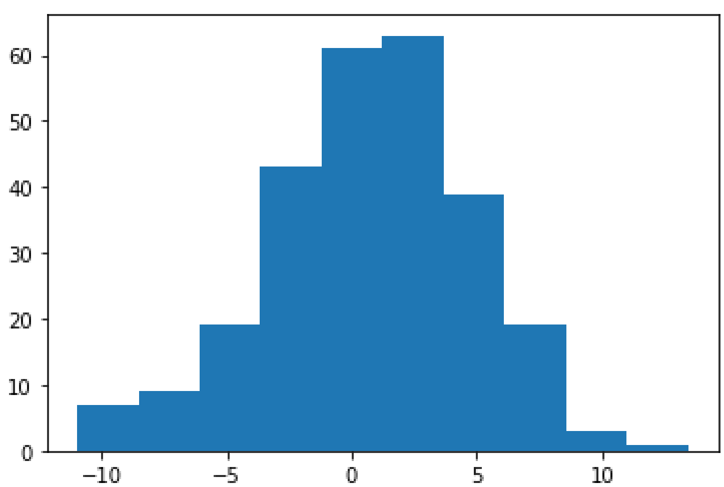


Out[21]: <AxesSubplot:xlabel='Shops', ylabel='Density'>

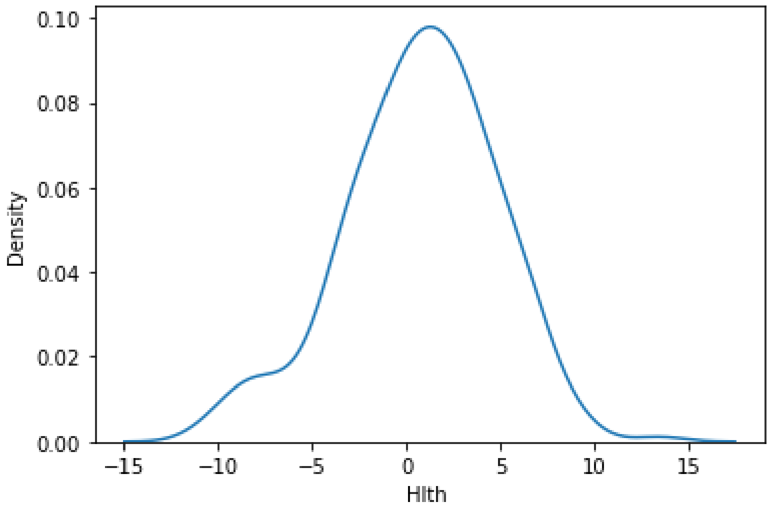


In [43]:

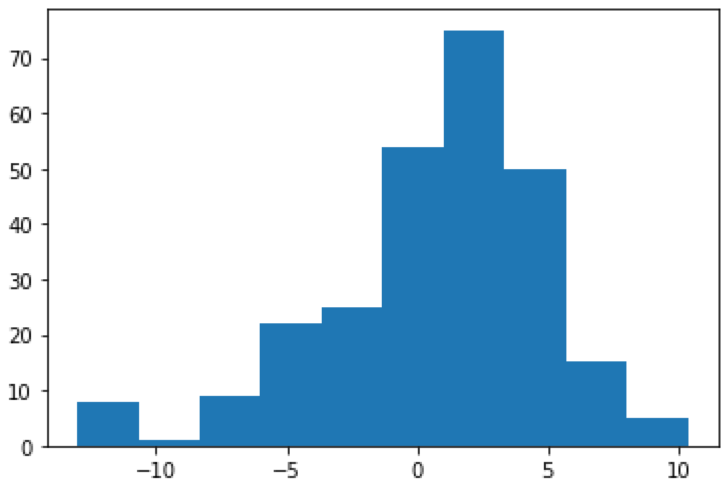
```
# Hlth
plt.hist(industry_c['Hlth '])
plt.show()
# Kernel density
sns.kdeplot(data=industry_c['Hlth '])
```



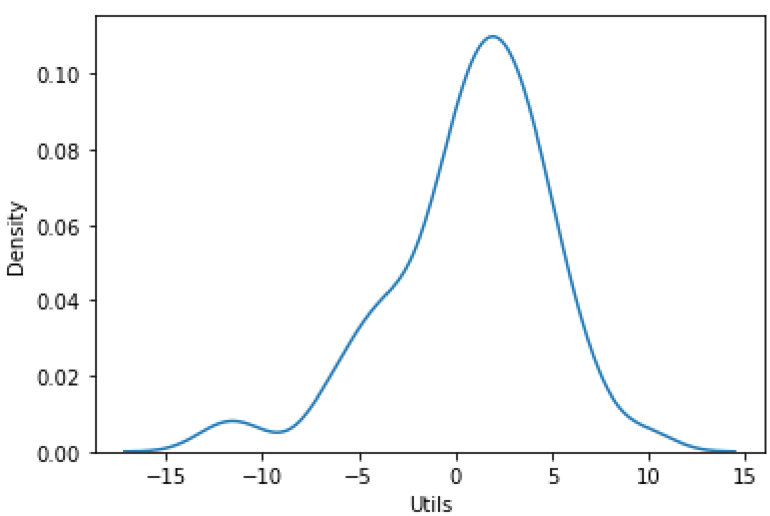
Out[43]: <AxesSubplot:xlabel='Hlth ', ylabel='Density'>



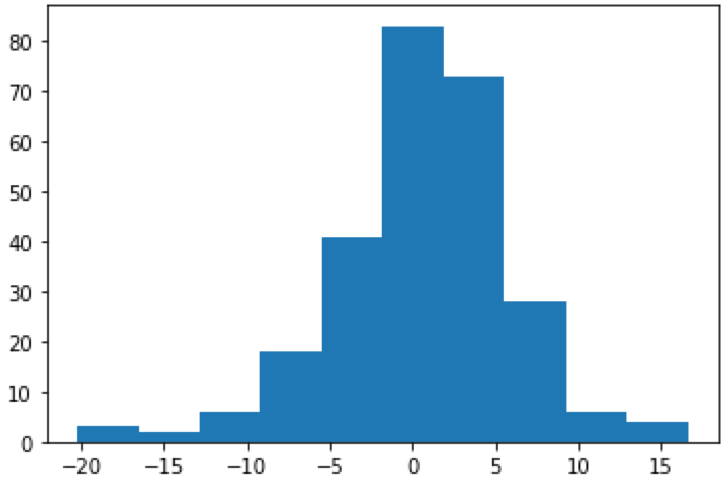
In [23]:
```python
# Utils
plt.hist(industry_c['Utils'])
plt.show()
# Kernel density
sns.kdeplot(data=industry_c['Utils'])
```



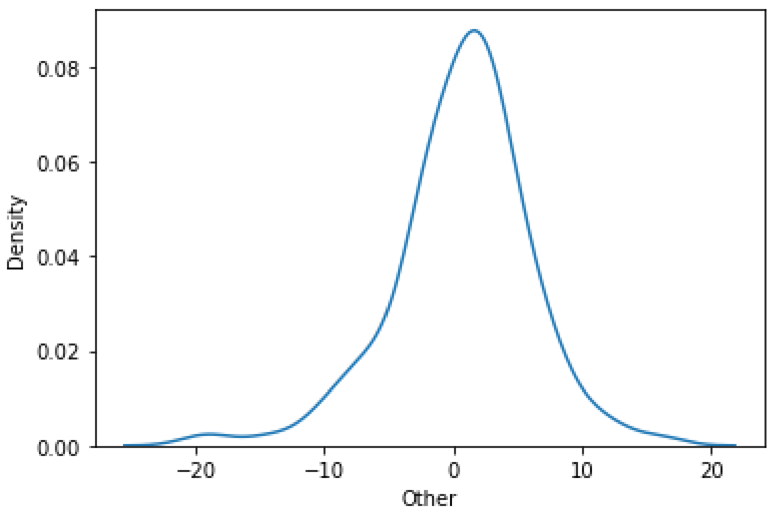
Out[23]: <AxesSubplot:xlabel='Utils', ylabel='Density'>



In [24]:
```
# Other
plt.hist(industry_c['Other'])
plt.show()
# Kernel density
sns.kdeplot(data=industry_c['Other'])
```



Out[24]: <AxesSubplot:xlabel='Other', ylabel='Density'>



Commend: \ From the histogram and kernel density plots of these ten industries, we can see that they are all follow the normal distribution, mostly are basis to the righ, meaning that the factors are increaseing in all fields.

C3:

In [3]:
```
# Import data
factors_c = pd.read_csv("factors.csv")
factors_c.head()
```

Out[3]:

	Date	Mkt-RF	SMB	HML	RMW	CMA	RF
0	200012	1.19	3.26	7.61	1.72	4.78	0.50
1	200101	3.13	5.50	-5.09	-4.70	-5.00	0.54
2	200102	-10.05	2.82	12.48	9.11	9.05	0.38
3	200103	-7.26	2.33	6.42	3.36	3.91	0.42

	Date	Mkt-RF	SMB	HML	RMW	CMA	RF
4	200104	7.94	-0.86	-4.68	-3.07	-3.19	0.39

In [73]:
```
# One-factor French-Fama model : Mkt-RF
# Nodur
# goodness-of-fit & significance & estimated coefficients
chi_square_test = stats.chisquare(industry_c['NoDur'], factors_c['Mkt-RF'])
print('Nodur goodness-of-fit value is : ' + str(chi_square_test[0]))
gradient, intercept, r_value, p_value, std_err = stats.linregress(industry_c['No
print('Nodur significance is ' + str(p_value))
print('Nodur R square is ' + str(r_value**2))
print('Nodur std.Error is ' + str(std_err))
```

```
Nodur goodness-of-fit value is : -843.9539878372768
Nodur significance is 2.0419842500698238e-53
Nodur R square is 0.5955006968025899
Nodur std.Error is 0.049546749914277724
```

In [61]:
```
# Durbl
chi_square_test = stats.chisquare(industry_c['Durbl'], factors_c['Mkt-RF'])
print('Durbl goodness-of-fit value is : ' + str(chi_square_test[0]))
gradient, intercept, r_value, p_value, std_err = stats.linregress(industry_c['Du
print('Durbl significance is ' + str(p_value))
print('Durbl R square is ' + str(r_value**2))
print('Durbl std.Error is ' + str(std_err))
```

```
Durbl goodness-of-fit value is : -4625.150361631209
Durbl significance is 6.073725184872438e-59
Durbl R square is 0.6328617386862075
Durbl std.Error is 0.019416654232979638
```

In [64]:
```
# Manuf
chi_square_test = stats.chisquare(industry_c['Manuf'], factors_c['Mkt-RF'])
print('Manuf goodness-of-fit value is : ' + str(chi_square_test[0]))
gradient, intercept, r_value, p_value, std_err = stats.linregress(industry_c['Ma
print('Manuf significance is ' + str(p_value))
print('Manuf R square is ' + str(r_value**2))
print('Manuf std.Error is ' + str(std_err))
```

```
Manuf goodness-of-fit value is : -253.65543739276546
Manuf significance is 1.545228445903044e-113
Manuf R square is 0.8592074538464006
Manuf std.Error is 0.020614170548720837
```

In [65]:
```
# Enrgy
chi_square_test = stats.chisquare(industry_c['Enrgy'], factors_c['Mkt-RF'])
print('Enrgy goodness-of-fit value is : ' + str(chi_square_test[0]))
gradient, intercept, r_value, p_value, std_err = stats.linregress(industry_c['En
print('Enrgy significance is ' + str(p_value))
print('Enrgy R square is ' + str(r_value**2))
print('Enrgy std.Error is ' + str(std_err))
```

```
Enrgy goodness-of-fit value is : -1872.1395889914859
Enrgy significance is 9.456696983246515e-31
Enrgy R square is 0.39843523893446486
Enrgy std.Error is 0.029905015609070513
```

In [66]:
```
# HiTec
chi_square_test = stats.chisquare(industry_c['HiTec'], factors_c['Mkt-RF'])
print('HiTec goodness-of-fit value is : ' + str(chi_square_test[0]))
gradient, intercept, r_value, p_value, std_err = stats.linregress(industry_c['Hi
```

```
print('HiTec significance is ' + str(p_value))
print('HiTec R square is ' + str(r_value**2))
print('HiTec std.Error is ' + str(std_err))
```

```
HiTec goodness-of-fit value is : 493.0756980915119
HiTec significance is 9.009177514200495e-90
HiTec R square is 0.7862768924695399
HiTec std.Error is 0.020076410097199883
```

In [67]:

```
# Telcm
chi_square_test = stats.chisquare(industry_c['Telcm'], factors_c['Mkt-RF'])
print('Telcm goodness-of-fit value is : ' + str(chi_square_test[0]))
gradient, intercept, r_value, p_value, std_err = stats.linregress(industry_c['Te
print('Telcm significance is ' + str(p_value))
print('Telcm R square is ' + str(r_value**2))
print('Telcm std.Error is ' + str(std_err))
```

```
Telcm goodness-of-fit value is : -491.53414514008267
Telcm significance is 1.3948008706834036e-63
Telcm R square is 0.661553121741757
Telcm std.Error is 0.03166126244291789
```

In [68]:

```
# Shops
chi_square_test = stats.chisquare(industry_c['Shops'], factors_c['Mkt-RF'])
print('Shops goodness-of-fit value is : ' + str(chi_square_test[0]))
gradient, intercept, r_value, p_value, std_err = stats.linregress(industry_c['Sh
print('Shops significance is ' + str(p_value))
print('Shops R square is ' + str(r_value**2))
print('Shops std.Error is ' + str(std_err))
```

```
Shops goodness-of-fit value is : 147.23907255514212
Shops significance is 3.661604291453356e-87
Shops R square is 0.7762585981341016
Shops std.Error is 0.029219852102448565
```

In [70]:

```
# Hlth
chi_square_test = stats.chisquare(industry_c['Hlth '], factors_c['Mkt-RF'])
print('Hlth goodness-of-fit value is : ' + str(chi_square_test[0]))
gradient, intercept, r_value, p_value, std_err = stats.linregress(industry_c['Hl
print('Hlth significance is ' + str(p_value))
print('Hlth R square is ' + str(r_value**2))
print('Hlth std.Error is ' + str(std_err))
```

```
Hlth goodness-of-fit value is : -85.92837451252662
Hlth significance is 9.695082848357192e-52
Hlth R square is 0.5834357618015076
Hlth std.Error is 0.04476635844262193
```

In [71]:

```
# Utils
chi_square_test = stats.chisquare(industry_c['Utils'], factors_c['Mkt-RF'])
print('Utils goodness-of-fit value is : ' + str(chi_square_test[0]))
gradient, intercept, r_value, p_value, std_err = stats.linregress(industry_c['Ut
print('Utils significance is ' + str(p_value))
print('Utils R square is ' + str(r_value**2))
print('Utils std.Error is ' + str(std_err))
```

```
Utils goodness-of-fit value is : -1006.4690915988849
Utils significance is 5.190901671637835e-23
Utils R square is 0.3114070488574054
Utils std.Error is 0.056148884760160965
```

In [72]:

```
# Other
chi_square_test = stats.chisquare(industry_c['Other'], factors_c['Mkt-RF'])
```

```
print('Other goodness-of-fit value is : ' + str(chi_square_test[0]))
gradient, intercept, r_value, p_value, std_err = stats.linregress(industry_c['Ot
print('Other significance is ' + str(p_value))
print('Other R square is ' + str(r_value**2))
print('Other std.Error is ' + str(std_err))
```

```
Other goodness-of-fit value is : -229.2657040588055
Other significance is 5.538386153758585e-113
Other R square is 0.8578296557589952
Other std.Error is 0.019906471617507988
```

Commend: \ For the goodness to fit, I use chi-square index to show the distribution of the industries. If there is more deviation between the observed and expected frequencies, the value of Chi-Square will be more, the perfect fit will be zero. The calculation we have provides that the industries datas are not close to the market premium rate(Mkt-Rf). For the significance part, we use p-value to show the significance two indexs, when the significance level or the P-value > 5%, we say that the data is Not significant, and we can reject the hypothesis, otherwise when the significance level or the P-value < 5%, we say the index is significant. We use R^2 to calculate the estimated coefficients, if the R2 of a model is 0.50, then approximately half of the observed variation can be explained by the model's inputs. Under the calculation, Engry and Utils are under 0.5, others are around 0.8 to 0.9, these high R-squared, indicates the indexes performance moves relatively in line with the market premium.

C4:

In [75]:

```
# Three-factor French-Fama model : Mkt-RF SMB HML
# Nodur
# goodness-of-fit & significance & estimated coefficients
chi_square_test = stats.chisquare(industry_c['NoDur'], factors_c['Mkt-RF']+facto
print('Nodur goodness-of-fit value is : ' + str(chi_square_test[0]))
gradient, intercept, r_value, p_value, std_err = stats.linregress(industry_c['No
print('Nodur significance is ' + str(p_value))
print('Nodur R square is ' + str(r_value**2))
print('Nodur std.Error is ' + str(std_err))
```

```
Nodur goodness-of-fit value is : 3.149727507390367e+16
Nodur significance is 1.9787206681799017e-31
Nodur R square is 0.40553646733718274
Nodur std.Error is 0.09489234944440722
```

In [76]:

```
# Durbl
chi_square_test = stats.chisquare(industry_c['Durbl'], factors_c['Mkt-RF']+facto
print('Durbl goodness-of-fit value is : ' + str(chi_square_test[0]))
gradient, intercept, r_value, p_value, std_err = stats.linregress(industry_c['Du
print('Durbl significance is ' + str(p_value))
print('Durbl R square is ' + str(r_value**2))
print('Durbl std.Error is ' + str(std_err))
```

```
Durbl goodness-of-fit value is : 8.43398109416917e+16
Durbl significance is 1.1824471693446379e-37
Durbl R square is 0.4668539079189187
Durbl std.Error is 0.03696536478684838
```

In [77]:

```
# Telcm
chi_square_test = stats.chisquare(industry_c['Telcm'], factors_c['Mkt-RF']+facto
print('Telcm goodness-of-fit value is : ' + str(chi_square_test[0]))
gradient, intercept, r_value, p_value, std_err = stats.linregress(industry_c['Te
print('Telcm significance is ' + str(p_value))
```

```
print('Telcm R square is ' + str(r_value**2))
print('Telcm std.Error is ' + str(std_err))
```

```
Telcm goodness-of-fit value is : 8474873778785504.0
Telcm significance is 1.1522789702615934e-28
Telcm R square is 0.37610589767052305
Telcm std.Error is 0.06791265656598788
```

In [78]:

```
# Shops
chi_square_test = stats.chisquare(industry_c['Shops'], factors_c['Mkt-RF']+facto
print('Shops goodness-of-fit value is : ' + str(chi_square_test[0]))
gradient, intercept, r_value, p_value, std_err = stats.linregress(industry_c['Sh
print('Shops significance is ' + str(p_value))
print('Shops R square is ' + str(r_value**2))
print('Shops std.Error is ' + str(std_err))
```

```
Shops goodness-of-fit value is : 1.1603299259939456e+16
Shops significance is 1.0225585685375594e-35
Shops R square is 0.4484736405294525
Shops std.Error is 0.07247701203561163
```

In [79]:

```
# Hlth
chi_square_test = stats.chisquare(industry_c['Hlth '], factors_c['Mkt-RF']+facto
print('Hlth goodness-of-fit value is : ' + str(chi_square_test[0]))
gradient, intercept, r_value, p_value, std_err = stats.linregress(industry_c['Hl
print('Hlth significance is ' + str(p_value))
print('Hlth R square is ' + str(r_value**2))
print('Hlth std.Error is ' + str(std_err))
```

```
Hlth goodness-of-fit value is : 1.0535180536315102e+17
Hlth significance is 1.3661536632362167e-19
Hlth R square is 0.2691375456685054
Hlth std.Error is 0.09367871177633522
```

In [80]:

```
# Utils
chi_square_test = stats.chisquare(industry_c['Utils'], factors_c['Mkt-RF']+facto
print('Utils goodness-of-fit value is : ' + str(chi_square_test[0]))
gradient, intercept, r_value, p_value, std_err = stats.linregress(industry_c['Ut
print('Utils significance is ' + str(p_value))
print('Utils R square is ' + str(r_value**2))
print('Utils std.Error is ' + str(std_err))
```

```
Utils goodness-of-fit value is : 3.4956039587721788e+16
Utils significance is 1.1551442832538816e-14
Utils R square is 0.2038391084166267
Utils std.Error is 0.09538335109927945
```

In [81]:

```
# Other
chi_square_test = stats.chisquare(industry_c['Other'], factors_c['Mkt-RF']+facto
print('Other goodness-of-fit value is : ' + str(chi_square_test[0]))
gradient, intercept, r_value, p_value, std_err = stats.linregress(industry_c['Ot
print('Other significance is ' + str(p_value))
print('Other R square is ' + str(r_value**2))
print('Other std.Error is ' + str(std_err))
```

```
Other goodness-of-fit value is : 2.572546179146545e+16
Other significance is 1.5234612776559246e-81
Other R square is 0.7530603041227514
Other std.Error is 0.041447423989640804
```

Commend: \ When we apply the three factor model, the calculation outcome of the chi-square index become relatively big, which shows the huge distribution of these industries to the factor indexs. P-value after the applying the muti-factors come very small, when the P-value < 0.1%

we say that the data is Highly significant. All the R^2 data become smaller then the data of only one facter model. A low R-squared at 60% or less, indicates the security does not generally follow the movements of the three factor index model.

C5:

In [93]:
```
# Four-factor French-Fama model : Mkt-RF SMB HML RMW
four_factors = factors_c['Mkt-RF']+factors_c['SMB']+factors_c['HML']+factors_c['
# goodness-of-fit & significance & estimated coefficients
# Nodur
chi_square_test_nodur = stats.chisquare(industry_c['NoDur'], four_factors)
print('Nodur goodness-of-fit value is : ' + str(chi_square_test_nodur[0]))
gradient_nodur, intercept_nodur, r_value_nodur, p_value_nodur, std_err_nodur = s
print('Nodur significance is ' + str(p_value_nodur))
print('Nodur R square is ' + str(r_value_nodur**2))
print('Nodur std.Error is ' + str(std_err_nodur))
print('\n')
# Durbl
chi_square_test_durbl = stats.chisquare(industry_c['Durbl'], four_factors)
print('Durbl goodness-of-fit value is : ' + str(chi_square_test_durbl[0]))
gradient_durbl, intercept_durbl, r_value_durbl, p_value_durbl, std_err_durbl = s
print('Durbl significance is ' + str(p_value_durbl))
print('Durbl R square is ' + str(r_value_durbl**2))
print('Durbl std.Error is ' + str(std_err_durbl))
print('\n')
# Telcm
chi_square_test_telcm = stats.chisquare(industry_c['Telcm'], four_factors)
print('Telcm goodness-of-fit value is : ' + str(chi_square_test_telcm[0]))
gradient_telcm, intercept_telcm, r_value_telcm, p_value_telcm, std_err_telcm = s
print('Telcm significance is ' + str(p_value_telcm))
print('Telcm R square is ' + str(r_value_telcm**2))
print('Telcm std.Error is ' + str(std_err_telcm))
print('\n')
# Shops
chi_square_test_shops = stats.chisquare(industry_c['Shops'], four_factors)
print('Shops goodness-of-fit value is : ' + str(chi_square_test_shops[0]))
gradient_shops, intercept_shops, r_value_shops, p_value_shops, std_err_shops = s
print('Shops significance is ' + str(p_value_shops))
print('Shops R square is ' + str(r_value_shops**2))
print('Shops std.Error is ' + str(std_err_shops))
print('\n')
# Hlth
chi_square_test_hlth = stats.chisquare(industry_c['Hlth '], four_factors)
print('Hlth goodness-of-fit value is : ' + str(chi_square_test_hlth[0]))
gradient_hlth, intercept_hlth, r_value_hlth, p_value_hlth, std_err_hlth = stats.
print('Hlth significance is ' + str(p_value_hlth))
print('Hlth R square is ' + str(r_value_hlth**2))
print('Hlth std.Error is ' + str(std_err_hlth))
print('\n')
# Utils
chi_square_test_utils = stats.chisquare(industry_c['Utils'], four_factors)
print('Utils goodness-of-fit value is : ' + str(chi_square_test_utils[0]))
gradient_utils, intercept_utils, r_value_utils, p_value_utils, std_err_utils = s
print('Utils significance is ' + str(p_value_utils))
print('Utils R square is ' + str(r_value_utils**2))
print('Utils std.Error is ' + str(std_err_utils))
print('\n')
# Other
```

```
chi_square_test_other = stats.chisquare(industry_c['Other'], four_factors)
print('Other goodness-of-fit value is : ' + str(chi_square_test_other[0]))
gradient_other, intercept_other, r_value_other, p_value_other, std_err_other = s
print('Other significance is ' + str(p_value_other))
print('Other R square is ' + str(r_value_other**2))
print('Other std.Error is ' + str(std_err_other))
```

```
Nodur goodness-of-fit value is : 503.786814754617
Nodur significance is 9.335405703305405e-33
Nodur R square is 0.4191620812876098
Nodur std.Error is 0.09069828540437275


Durbl goodness-of-fit value is : -5690.865874272756
Durbl significance is 2.648389719896668e-29
Durbl R square is 0.38302737000947923
Durbl std.Error is 0.03845101750054741


Telcm goodness-of-fit value is : -2067.2535464042603
Telcm significance is 2.6514259128546923e-20
Telcm R square is 0.2781391669177699
Telcm std.Error is 0.07063582543855869


Shops goodness-of-fit value is : -2376.3275897672665
Shops significance is 4.170699446901499e-29
Shops R square is 0.38089774770506474
Shops std.Error is 0.07425083501499537


Hlth goodness-of-fit value is : -769.393885938873
Hlth significance is 1.5487945466734628e-14
Hlth R square is 0.20208032870392326
Hlth std.Error is 0.09464674357493894


Utils goodness-of-fit value is : 603.8329494224746
Utils significance is 4.818437230451185e-15
Utils R square is 0.20906164838311136
Utils std.Error is 0.09192771804311188


Other goodness-of-fit value is : -1547.497154199521
Other significance is 1.4170663689458739e-58
Other R square is 0.6304849357300772
Other std.Error is 0.049025402702932604
```

Commend: \ The four factor model makes the calculation outcome of the chi-square index become relatively unstatable, some of the industries are really samll, some industries have very bug index, which reflects the huge distribution of these industries to the factor indexs, the industries that are floating are technology field and consumptions. P-value after the applying the muti-factors come very small, although they are bigger then the three index model, these P-value is still < 0.1% which is Highly significant. All the R^2 data become half smaller then the data of the three facter model. A low R-squared at 60% or less, indicates the security does not generally follow the movements of the three factor index model.

C6:

In [97]:
```
# Five-factor French-Fama model : Mkt-RF SMB HML RMW CMA
five_factors = factors_c['Mkt-RF']+factors_c['SMB']+factors_c['HML']+factors_c['
```

```
# goodness-of-fit & significance & estimated coefficients
# Nodur
chi_square_test_nodur = stats.chisquare(industry_c['NoDur'], five_factors)
print('Nodur goodness-of-fit value is : ' + str(chi_square_test_nodur[0]))
gradient_nodur, intercept_nodur, r_value_nodur, p_value_nodur, std_err_nodur = s
print('Nodur significance is ' + str(p_value_nodur))
print('Nodur R square is ' + str(r_value_nodur**2))
print('Nodur std.Error is ' + str(std_err_nodur))
print('\n')
# Durbl
chi_square_test_durbl = stats.chisquare(industry_c['Durbl'], five_factors)
print('Durbl goodness-of-fit value is : ' + str(chi_square_test_durbl[0]))
gradient_durbl, intercept_durbl, r_value_durbl, p_value_durbl, std_err_durbl = s
print('Durbl significance is ' + str(p_value_durbl))
print('Durbl R square is ' + str(r_value_durbl**2))
print('Durbl std.Error is ' + str(std_err_durbl))
print('\n')
# Telcm
chi_square_test_telcm = stats.chisquare(industry_c['Telcm'], five_factors)
print('Telcm goodness-of-fit value is : ' + str(chi_square_test_telcm[0]))
gradient_telcm, intercept_telcm, r_value_telcm, p_value_telcm, std_err_telcm = s
print('Telcm significance is ' + str(p_value_telcm))
print('Telcm R square is ' + str(r_value_telcm**2))
print('Telcm std.Error is ' + str(std_err_telcm))
print('\n')
# Shops
chi_square_test_shops = stats.chisquare(industry_c['Shops'], five_factors)
print('Shops goodness-of-fit value is : ' + str(chi_square_test_shops[0]))
gradient_shops, intercept_shops, r_value_shops, p_value_shops, std_err_shops = s
print('Shops significance is ' + str(p_value_shops))
print('Shops R square is ' + str(r_value_shops**2))
print('Shops std.Error is ' + str(std_err_shops))
print('\n')
# Hlth
chi_square_test_hlth = stats.chisquare(industry_c['Hlth '], five_factors)
print('Hlth goodness-of-fit value is : ' + str(chi_square_test_hlth[0]))
gradient_hlth, intercept_hlth, r_value_hlth, p_value_hlth, std_err_hlth = stats.
print('Hlth significance is ' + str(p_value_hlth))
print('Hlth R square is ' + str(r_value_hlth**2))
print('Hlth std.Error is ' + str(std_err_hlth))
print('\n')
# Utils
chi_square_test_utils = stats.chisquare(industry_c['Utils'], five_factors)
print('Utils goodness-of-fit value is : ' + str(chi_square_test_utils[0]))
gradient_utils, intercept_utils, r_value_utils, p_value_utils, std_err_utils = s
print('Utils significance is ' + str(p_value_utils))
print('Utils R square is ' + str(r_value_utils**2))
print('Utils std.Error is ' + str(std_err_utils))
print('\n')
# Other
chi_square_test_other = stats.chisquare(industry_c['Other'], five_factors)
print('Other goodness-of-fit value is : ' + str(chi_square_test_other[0]))
gradient_other, intercept_other, r_value_other, p_value_other, std_err_other = s
print('Other significance is ' + str(p_value_other))
print('Other R square is ' + str(r_value_other**2))
print('Other std.Error is ' + str(std_err_other))
```

```
Nodur goodness-of-fit value is : -1716.0297508327621
Nodur significance is 9.849885163488321e-28
Nodur R square is 0.3658681332842163
Nodur std.Error is 0.10553923594164984
```

```
Durbl goodness-of-fit value is : 1967.189936875467
Durbl significance is 4.5916051869486016e-20
Durbl R square is 0.2751361344137008
Durbl std.Error is 0.04641471336314966


Telcm goodness-of-fit value is : -3088.79530452783
Telcm significance is 1.6727598428430466e-15
Telcm R square is 0.2153380339613516
Telcm std.Error is 0.08201478417689555


Shops goodness-of-fit value is : -173.06932689875032
Shops significance is 4.209990768440036e-19
Shops R square is 0.26289566190274394
Shops std.Error is 0.09022714044051278


Hlth goodness-of-fit value is : 391.1275553125105
Hlth significance is 1.5854683367135748e-10
Hlth R square is 0.14485110237777543
Hlth std.Error is 0.10911878464447027


Utils goodness-of-fit value is : 501.7430737015723
Utils significance is 2.1126891708576173e-13
Utils R square is 0.18624857028079342
Utils std.Error is 0.10384213402227932


Other goodness-of-fit value is : 750.3984060272412
Other significance is 1.7910887715094308e-41
Other R square is 0.501339687794583
Other std.Error is 0.06342493894019896
```

Commend: \ The five factor model makes the calculation outcome of the chi-square index become close the the perfect fit, but there are still lots of difference between the expected data index, the difference clearly divide the industries to two part, one part with Nodur, Telcm and Shops. Another part with Durbl, Hlthm, Utils and Other. These data index all reflects the distributions amount these industries with factor model indexs. P-value after the applying the five-factors are still very small, very similar to the three and four index model, all the P-values are still < 0.1% which is Highly significant, P-values are not affected by the change of different factor model. All the R^2 data become smaller then the data of the four facter model, most industries are around 20% to 50%. A low R-squared with lower than 60% indicates the security does not generally follow the movements of the five factor index model and they are very close to a straight line relationship, which is not a good index to make prediction.

In [ ]: