Exercise1Solution

September 2, 2022

1 Solutions to week 1 exercises in Introduction to Financial Engineering

This Jupyter Notebook shows one way on how to do the exercises for week 1 in Python. We start by importing the needed libraries.

Note that you probably don't have yfinance installed. If this is the case, you can install yfinance to your Anaconda environment by copy pasting the following line into Anaconda Prompt (Windows) or the terminal (MacOS): pip install -i https://pypi.anaconda.org/ranaroussi/simple yfinance

And if you are not using an Anaconda environment, you can simply install the yfinance package by copy pasting the following line in Command Prompt (Windows) or Terminal (Mac): pip install yfinance

[ ]:
```
import numpy as np
import pandas as pd
import yfinance as yf
import datetime
import matplotlib.pyplot as plt
from scipy.stats.mstats import gmean
import pylab
import statsmodels.api as sm
import scipy.stats as stats
import utils
import os
import pandas as pd

# import seaborn as sns
# sns.set()

# plt.savefig("filnavn.eps") # filnavn.filtype: jpg, png, eps, pdf
```

1.1 Question 1

1.1.1 a) Get Weekly historical prices for Exxon Mobil (XOM)

Before we fetch historical weekly prices of the XOM stock, we first have to create *datetime* objects with our start and end dates which are 1/1/2005 and today, respectively.

We then use the yfinance method *download* with 'XOM', start date, end date, and the interval (weekly data) as our inputs, to collect data of the XOM stock.

The yfinance.download method gives us 6 columns (Open, High, Low, Close, Adj Close, Volume).

- Open:: The price of the stock at the very first trade of the week.
- High:: The highest price of the stock that was reached in a trade during the week.
- Low:: The lowest price of the stock that was reached in a trade during the week.
- Close:: The price of the stock at the very last trade of the week.
- Adj Close:: The adjusted price of the stock at the very last trade of the week. Read more about adjusted prices at https://www.investopedia.com/terms/a/adjusted_closing_price.asp
- Volume:: The number of shares of the stock traded during the week.

[ ]:
```
start = datetime.datetime(2005,1,1)
end = datetime.datetime(2022,8,30)

xom = yf.download('XOM', start=start, end=end, interval='1wk')  # Downloading␣
↪XOM data.
xom.tail(10)  # Displaying last 10 rows of the dataframe.
```

```
[*********************100%***********************]  1 of 1 completed
```

[ ]:
```
                 Open        High        Low       Close   Adj Close  \
Date
2022-07-04  86.339996   87.300003  81.010002   86.080002   85.251678
2022-07-11  85.169998   86.309998  80.690002   84.540001   83.726494
2022-07-18  86.580002   89.650002  85.209999   87.080002   86.242050
2022-07-25  88.160004   97.519997  87.419998   96.930000   95.997269
2022-08-01  94.790001   95.349998  86.279999   88.449997   87.598862
2022-08-08  88.610001   94.300003  88.220001   94.000000   93.095459
2022-08-11        NaN         NaN        NaN         NaN         NaN
2022-08-15  90.529999   95.309998  89.660004   94.080002   94.080002
2022-08-22  93.419998   99.910004  91.860001   97.870003   97.870003
2022-08-29  98.180000  101.559998  98.139999  100.120003  100.120003

                 Volume
Date
2022-07-04  109458000.0
2022-07-11  100132100.0
2022-07-18   91084300.0
2022-07-25  100883400.0
2022-08-01  110318700.0
2022-08-08   85694400.0
2022-08-11          NaN
2022-08-15   82368000.0
2022-08-22   87672000.0
2022-08-29   23059200.0
```

As is seen a row with NaN values are appended whenever a corporate event has occured. We can remove these with the dropna() method.

[ ]:
```
xom = xom.dropna(axis = 0) # axis = 0 refers to rows. You can do the same thing␣
↪with columns if needed.
```

[ ]:
```
xom.tail(10)  # Displaying first 10 rows of the dataframe.
```

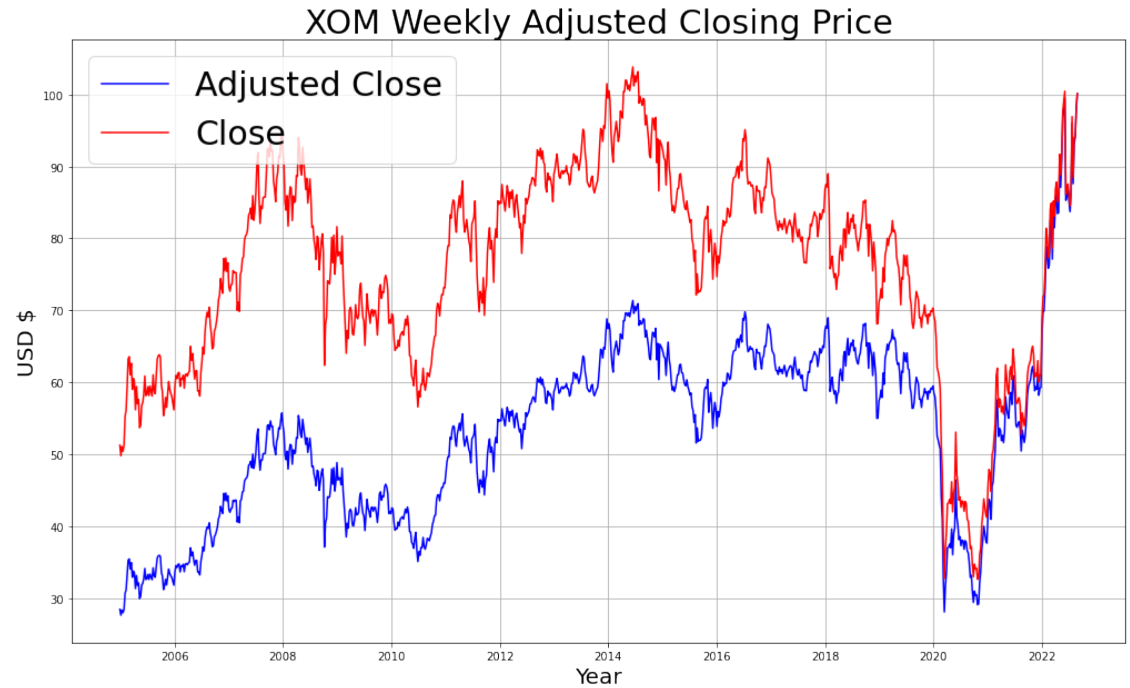
[ ]:
```
                 Open        High        Low       Close   Adj Close  \
Date
2022-06-27  87.809998   93.239998  84.669998   87.550003   86.707535
2022-07-04  86.339996   87.300003  81.010002   86.080002   85.251678
2022-07-11  85.169998   86.309998  80.690002   84.540001   83.726494
2022-07-18  86.580002   89.650002  85.209999   87.080002   86.242050
2022-07-25  88.160004   97.519997  87.419998   96.930000   95.997269
2022-08-01  94.790001   95.349998  86.279999   88.449997   87.598862
2022-08-08  88.610001   94.300003  88.220001   94.000000   93.095459
2022-08-15  90.529999   95.309998  89.660004   94.080002   94.080002
2022-08-22  93.419998   99.910004  91.860001   97.870003   97.870003
2022-08-29  98.180000  101.559998  98.139999  100.120003  100.120003

                 Volume
Date
2022-06-27  150252500.0
2022-07-04  109458000.0
2022-07-11  100132100.0
2022-07-18   91084300.0
2022-07-25  100883400.0
2022-08-01  110318700.0
2022-08-08   85694400.0
2022-08-15   82368000.0
2022-08-22   87672000.0
2022-08-29   23059200.0
```

It has now been removed.

1.1.2 b) Plot the adjusted closing price and the closing price in the same graph

Let's plot the weekly Adjusted Close prices and the Close prices for the XOM stock.

[ ]:
```
plt.figure(figsize=(17,10))
plt.plot(xom['Adj Close'], color='b', label='Adjusted Close')
plt.plot(xom['Close'], color='r', label='Close')
plt.title('XOM Weekly Adjusted Closing Price', fontsize=30)
plt.xlabel('Year', fontsize=20)
plt.ylabel('USD $', fontsize=20)
plt.grid()
plt.legend(fontsize=30)
plt.show()
```



1.1.3 c) Any differences between Adjusted close prices and close prices?

Obviously - there is difference between the adjusted close prices and close prices. And to understand why, we need to understand what adjusted close prices and close prices are.

Weekly close price is simply the price which the price of the stock is at - at the end of the week. Meaning if the stock exchange closes at 4pm on Friday then the closing price is the price of the last trade of the stock before the exchange closed.

Weekly adjusted close prices is almost the same as weekly close prices with a few modiciations. The prices are **adjusted** after some corporative actions has been done. This means if the administration of the company - which the stock represents an ownership of - undertakes some actions which affects the performance of the stock. Such actions could be ;

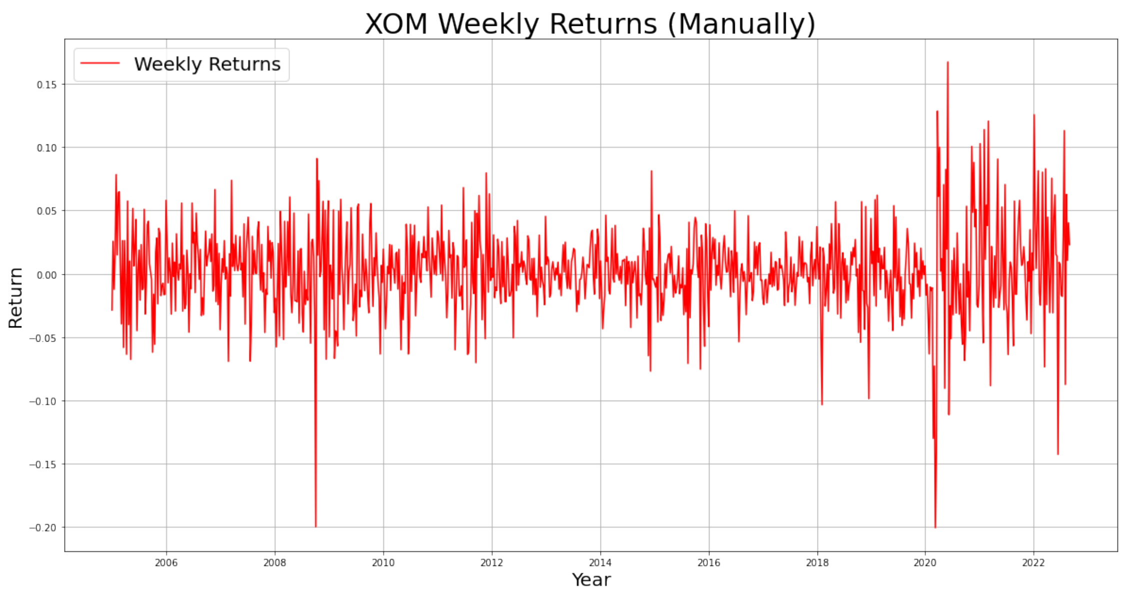
- **stock splits** - If there on 1/1/2020 exists 10.000.000 shares of a company and it then chooses to do a 1:3 split of the stocks the following day - Then the company would have 30.000.000 shares on 2/1/2020. This means that the price of the stock on 1/2/2020 would be 1/3 worth of what it was worth on 1/1/2020. Adjusted closing prices takes this into account whereas ordinary closing prices does not.
- **Dividend payouts** - Imagine that a company is worth 100.000.000$ and the company chooses to payout 1.000.000$ to all its investors. Then the company has lost 1 million of its worth because they do not have these money anymore and thus would only be worth 99.000.000$. Adjusted closing prices therefor takes dividend payouts into account and lowers the price whenever the company pays out dividends.

There exists multiple other reasons to adjust the prices which you can read of by performing a quick google search. But in short, adjusted closing prices takes multiple factors into considerations which is valuable for an investor to assess the performance of the stock. And thus it is more meaningful to calculate the returns of adjusted prices and not the simple closing prices.

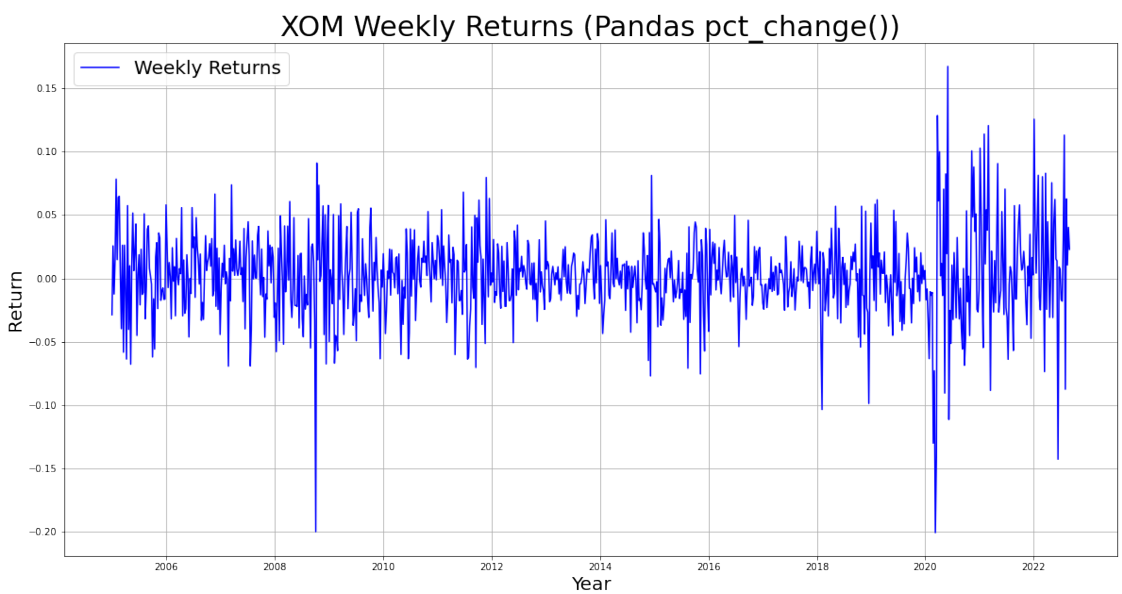
1.1.4 d) Calculate Weekly Returns

You can do this in two ways. Either, you can define your own function where you calculate the returns manually or using the inbuilt Pandas method *pct_change()* which does exactly the same. Note that the very first return (week 1 of the dataset is a NaN since the calculation of the return requires the stock price from the week prior).

[ ]:
```
# Manually calculate the return
def calculate_returns(prices):
    return (prices / np.roll(prices, 1) - 1)[1:]

# Computing returns using the home-made function.
returns = calculate_returns(xom['Adj Close'].to_numpy())
plt.figure(figsize=(20,10))
plt.plot(xom.index[1:], returns, color='r', label="Weekly Returns")
plt.title('XOM Weekly Returns (Manually)', fontsize=30)
plt.xlabel('Year', fontsize=20)
plt.ylabel('Return', fontsize=20)
plt.grid()
plt.legend(fontsize=20)
plt.show()

# Using the inbuilt Pandas method 'pct_change()'
xom['Returns'] = xom['Adj Close'].pct_change()
plt.figure(figsize=(20,10))
plt.plot(xom['Returns'][1:], color='b', label="Weekly Returns")
plt.title('XOM Weekly Returns (Pandas pct_change())', fontsize=30)
plt.xlabel('Year', fontsize=20)
plt.ylabel('Return', fontsize=20)
plt.grid()
plt.legend(fontsize=20)
plt.show()
```




```
C:\Users\nilan\AppData\Local\Temp\ipykernel_17656\489933967.py:17:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  xom['Returns'] = xom['Adj Close'].pct_change()
```



1.1.5 e) Calculate the weekly average return (Note: Remember the difference between arithmetic and geometric averages!)**

Now, we will compute the average weekly return of the XOM stock. If X_0 is our investment in the XOM stock in week 0 and R_i is the return in week i, then our investment in week n is grown to: $X_n = X_0(1 + R_1)(1 + R_2)...(1 + R_n)$. Since our investment grows to our original investment multiplied with the product of 1 + each weekly return for all weeks, it doesn't make sense to do the arithmetic average (summing all returns up and divide by number of weeks). Thus, we need to do the geometric average which is computed as $\bar{R} = \left(\prod_{i=1}^{N}(1 + R_i)\right)^{\frac{1}{N}} - 1$. Since there may be numerically issues writing the geometric average as a function in Python, it is safer and easier to use a prewritten function for this. From the scipy library, there is a method called gmean which computes the geometric average. Likewise, the standard deviation can be computed using the *std* method from the numpy library.

```
[ ]: xom_geomean_weekly = gmean(1 + xom['Returns'].to_numpy()[1:]) - 1
     print('Weekly Geometric Mean of XOM Returns:: {}%'.format(np.
      ↪round(xom_geomean_weekly*100, 4)))
```

```
Weekly Geometric Mean of XOM Returns:: 0.1366%
```

1.1.6 f) Calculate the standard deviation of returns

From the numpy library, there exists a method called *std* that computes the standard deviation of an array.

```
[ ]: xom_sd_weekly = np.std(xom['Returns'].to_numpy()[1:])
     print('Weekly Standard Deviation of XOM Returns:: {}%'.format(np.
      ↪round(xom_sd_weekly*100, 4)))
```

```
Weekly Standard Deviation of XOM Returns:: 3.4171%
```

1.1.7 g) Calculate log returns and the weekly average log return and compare with the actual weekly returns

Log returns are calculated as $\log\left(\frac{P_t}{P_{t-1}}\right)$ which is equivalent to $\log(R_t + 1)$ where P_t is the adjusted closing price in week t. After computing the log returns, we can compute the average weekly log return by computing the arthimetric average of the log returns. Why not the geometric average? Remember that the average return can be expressed as

$$(1 + \bar{R})^N = (1 + R_1)(1 + R_2)...(1 + R_3)$$

By taking the log on both sides and hereafter divide by N, we get:

$$log(1 + \bar{R}) = \frac{log(1 + R_1) + log(1 + R_2) + ... + log(1 + R_N)}{N}$$

which is the arthimetric average of the log returns which also is a good approximation for the true value of $\bar{R}$ which we can call $\hat{R}$ s.t.

$$\hat{R} = log(1+\bar{R}) = \frac{log(1+R_1) + log(1+R_2) + ... + log(1+R_N)}{N}$$

We can also choose to get the exact value by computing $\bar{R} = e^{\hat{R}} - 1$

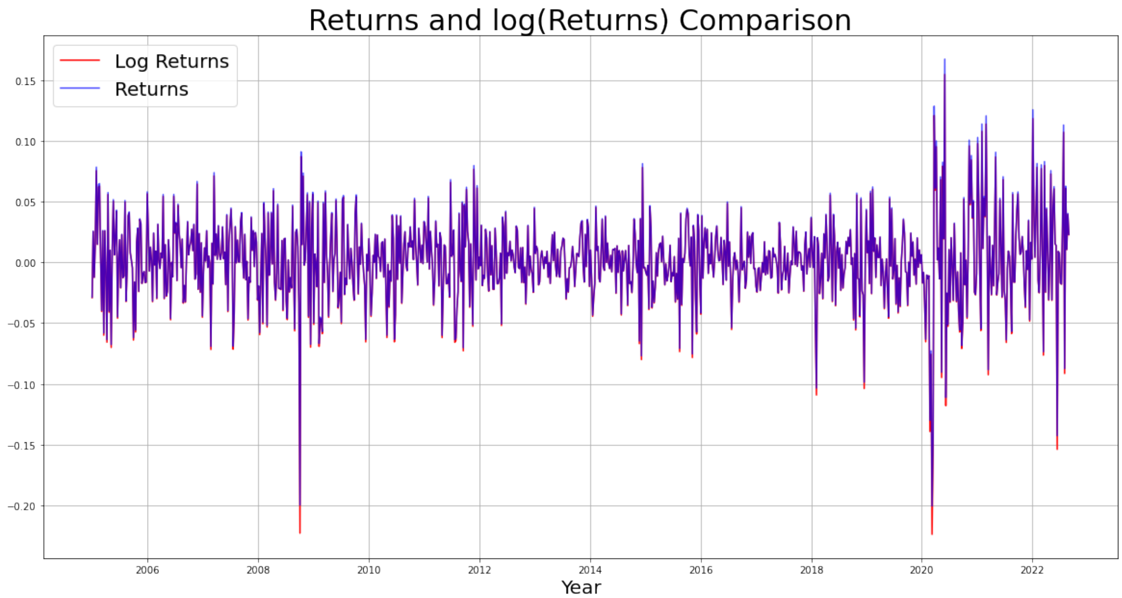
Why would anyone work with the log returns instead of the real returns then? First of all, it is computationally easier (more stable) and faster to work with log returns. Moreover, and you'll learn this later, the log returns tends to have a nicer distribution than the actual returns and thus more suitable for statistical models and future predictions.

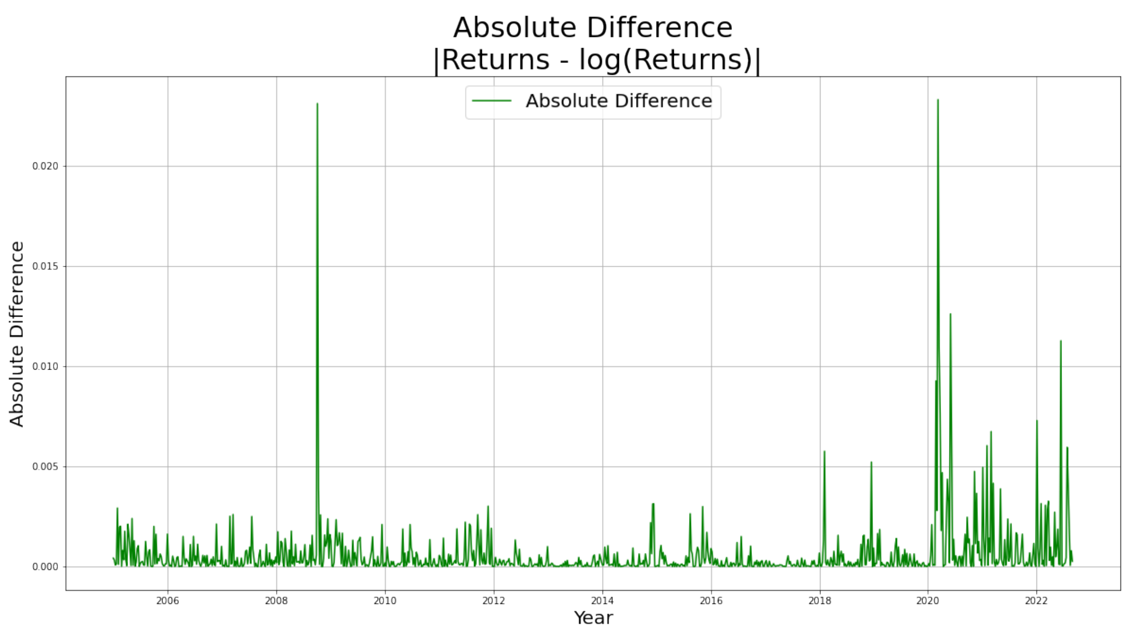
[ ]:
```
xom['Log Returns'] = np.log(1 + xom['Returns'])
xom_logreturn_mean = np.mean(xom['Log Returns'].to_numpy()[1:])
xom_logreturn_sd = np.std(xom['Log Returns'].to_numpy()[1:])

print('Weekly Mean of Log 1 + R^XOM:: {}%'.format(np.
 ↪round(xom_logreturn_mean*100, 4)))
print('Weekly Standard Deviation of Log 1 + R^XOM:: {}%'.format(np.
 ↪round(xom_logreturn_sd*100, 4)))

print('Weekly Geometric Mean of XOM Returns:: {}%'.format(np.
 ↪round(xom_geomean_weekly*100, 4)))
print('Weekly Standard Deviation of XOM Returns:: {}%'.format(np.
 ↪round(xom_sd_weekly*100, 4)))

plt.figure(figsize=(20,10))
plt.plot(xom['Log Returns'], color='r', label='Log Returns')
plt.plot(xom['Returns'], color='b', alpha=0.7, label='Returns')
plt.title('Returns and log(Returns) Comparison', fontsize=30)
plt.xlabel('Year', fontsize=20)
plt.grid()
plt.legend(fontsize=20)
plt.show()

diff = np.abs(xom['Returns'] - xom['Log Returns'])
plt.figure(figsize=(20,10))
plt.plot(diff, color='g', label='Absolute Difference')
plt.title('Absolute Difference\n |Returns - log(Returns)|', fontsize=30)
plt.xlabel('Year', fontsize=20)
plt.ylabel('Absolute Difference', fontsize=20)
plt.grid()
plt.legend(fontsize=20)
plt.show()
```

```
C:\Users\nilan\AppData\Local\Temp\ipykernel_17656\2667039793.py:1:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  xom['Log Returns'] = np.log(1 + xom['Returns'])
```

```
Weekly Mean of Log 1 + R^XOM:: 0.1365%
Weekly Standard Deviation of Log 1 + R^XOM:: 3.4435%
Weekly Geometric Mean of XOM Returns:: 0.1366%
Weekly Standard Deviation of XOM Returns:: 3.4171%
```





1.2 Question 2

On to question 2 of the exercises. Here, we have to download daily data of three different ETFs. We can do it in the same way as before, but where we put a list of tickers in the yfinance.download method. We then get a dataframe which has columns in columns (nested columns). The way to access a specific column is shown beneath.

[ ]:
```python
start = datetime.datetime(year=2005, month=1, day=1)
end = datetime.datetime(year=2022, month=8, day=31)

ticks = ['SPY', 'XLF', 'EEM']
stocks = yf.download(ticks, start=start, end=end, interval='1d')
stocks = stocks[stocks.notna()]

# if you e.g. want to access the Close prices of SPY, you can do the following:
SPY_close_prices = stocks['Close']['SPY']

stocks.head()
```

```
[*********************100%***********************]  3 of 3 completed
```

[ ]:
```
             Adj Close                                Close                          \
                   EEM        SPY        XLF           EEM         SPY        XLF
Date
2004-12-31  16.087749  86.040253  17.286310  22.427778  120.870003  24.800976
2005-01-03  15.920377  85.634491  17.224022  22.194445  120.300003  24.711617
2005-01-04  15.430210  84.588089  17.059816  21.511110  118.830002  24.476036
2005-01-05  15.241319  84.004379  17.031517  21.247778  118.010002  24.435417
2005-01-06  15.230959  84.431511  17.116444  21.233334  118.610001  24.557270

                 High                                  Low                   \
                  EEM         SPY        XLF           EEM         SPY
Date
2004-12-31  22.433332  121.660004  24.947197  22.330000  120.800003
2005-01-03  22.494444  121.760002  25.012184  22.153334  119.900002
2005-01-04  22.150000  120.540001  24.857840  21.511110  118.440002
2005-01-05  21.530001  119.250000  24.638506  21.244444  118.000000
2005-01-06  21.346666  119.150002  24.630383  21.125557  118.260002

                             Open                              Volume             \
                  XLF         EEM         SPY        XLF        EEM        SPY
Date
2004-12-31  24.776604  22.433332  121.300003  24.857840  1507500  28648800
2005-01-03  24.679123  22.411112  121.559998  24.979691  4275000  55748000
2005-01-04  24.427296  22.138889  120.459999  24.817223  4205700  69167600
2005-01-05  24.427296  21.488890  118.739998  24.451666  3006900  65667300
2005-01-06  24.476036  21.316668  118.440002  24.500406  2268000  47814700
```

```
                  XLF
Date
2004-12-31   701424
2005-01-03  5011647
2005-01-04  7426746
2005-01-05  2943321
2005-01-06  3211925
```

[ ]:
```
(stocks['Adj Close'] / stocks['Adj Close'].iloc[0,:]) * 100
```

[ ]:
```
                   EEM         SPY         XLF
Date
2004-12-31  100.000000  100.000000  100.000000
2005-01-03   98.959626   99.528405   99.639667
2005-01-04   95.912795   98.312228   98.689750
2005-01-05   94.738663   97.633813   98.526041
2005-01-06   94.674267   98.130245   99.017335
…                  …           …           …
2022-08-24  247.082415  480.786609  198.017964
2022-08-25  252.428102  487.574126  201.083967
2022-08-26  248.698542  471.070208  195.067662
2022-08-29  247.144564  467.955396  193.679271
2022-08-30  244.223103  462.818250  192.637983

[4447 rows x 3 columns]
```

1.2.1 a) Plot the adjusted closing prices in the same graph. Does it make sense to plot them in the same graph or should any adjustment be made?

As you can see in the first plot, it is hard to compare the closing prices from the three ETFs since they are on different scales (SPY has a much higher price than EEM and XLF). Likewise, they are hard to compare when they are in 3 distinct plots. Thus, one way to compare them is to index the time series such that they all start at 100.

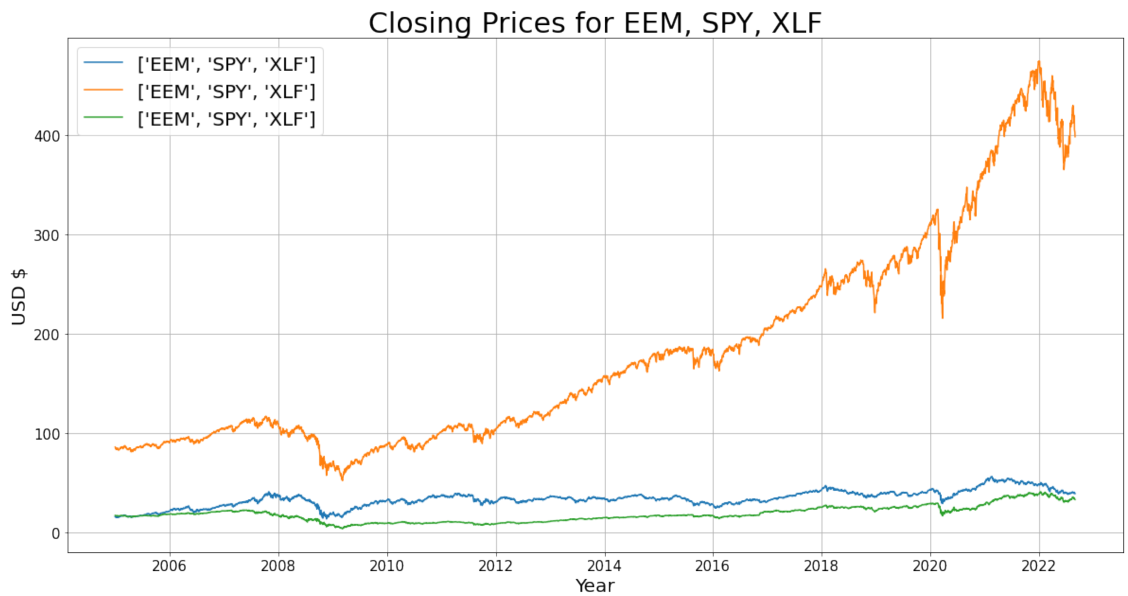
$$P_t^{(indexed)} = \frac{P_t}{P_0} \cdot 100$$

Any point on the graph now represent the percentage return since time 0 (e.g. 131 corresponds to 31% increase). Hence, if we use the adjusted close price we can quickly see which stock has performed the best in terms of returns.

We start by doing a non-indexed plot.

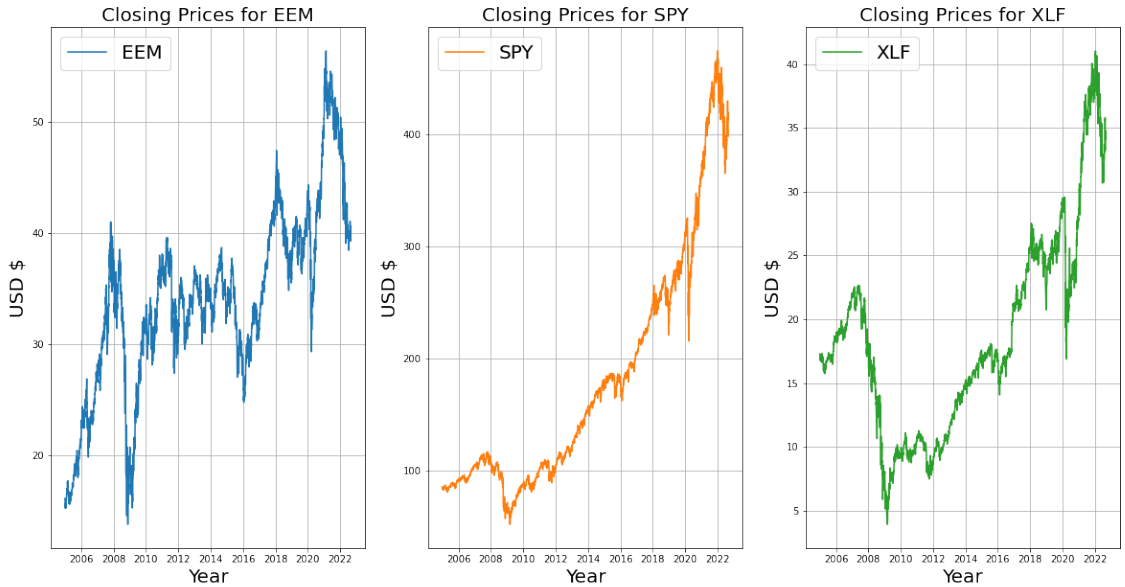
[ ]:
```
colors = ['C0', 'C1', 'C2']
plt.figure(figsize=(20,10))
plt.plot(stocks['Adj Close'], label=list(stocks['Adj Close']))
plt.title('Closing Prices for EEM, SPY, XLF', fontsize=30)
```

```
plt.xlabel('Year', fontsize=20)
plt.ylabel('USD $', fontsize=20)
plt.legend(fontsize=20)
plt.grid()
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.show()
```



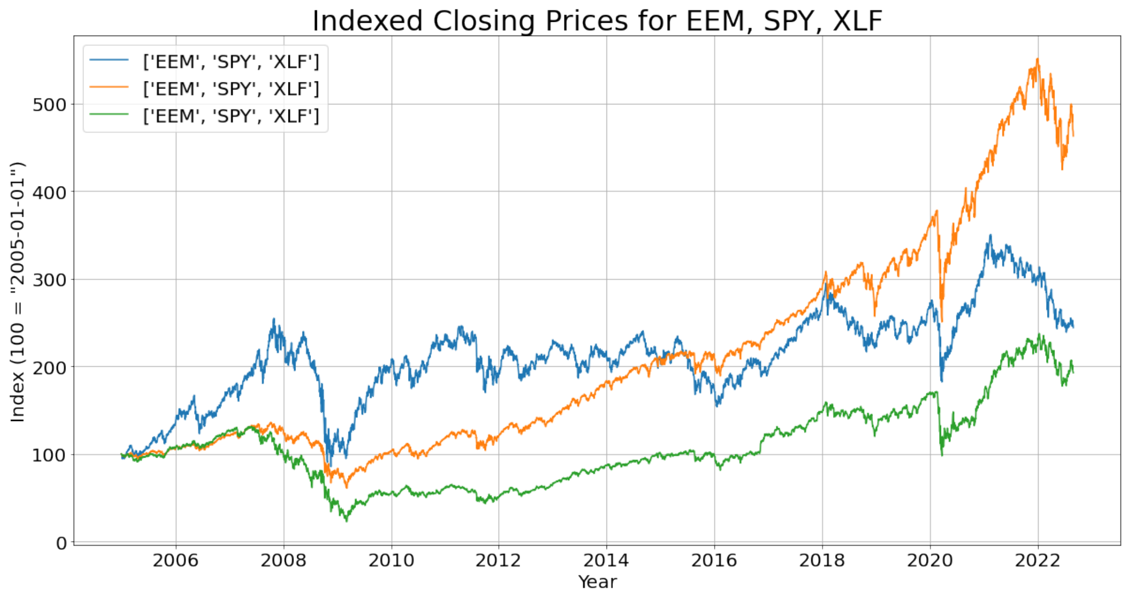
Then we can plot them separately.

```
[ ]: figs, axs = plt.subplots(1, 3, figsize=(20, 10))
     for i,etf in enumerate(list(stocks['Adj Close'])):
         axs[i].plot(stocks['Adj Close'][etf], label=etf, color=colors[i])
         axs[i].set_title('Closing Prices for {}'.format(etf), fontsize=20)
         axs[i].set_xlabel('Year', fontsize=20)
         axs[i].set_ylabel('USD $', fontsize=20)
         axs[i].grid()
         axs[i].legend(fontsize=20)
```



Then we use the index introduced above.

```
[ ]: normalized_adjusted_closing_prices = (stocks['Adj Close'] / stocks['Adj Close'].
      ↪iloc[0,:]) * 100; # this needs to be changed to all stocks
     plt.figure(figsize=(20,10))
     plt.plot(normalized_adjusted_closing_prices, label=list(stocks['Adj Close']))
     plt.title('Indexed Closing Prices for EEM, SPY, XLF', fontsize=30)
     plt.xlabel('Year', fontsize=20)
     plt.ylabel('Index (100 = "2005-01-01")', fontsize=20)
     plt.legend(fontsize=20)
     plt.grid()
     plt.xticks(fontsize=20)
     plt.yticks(fontsize=20)
     plt.show()
```

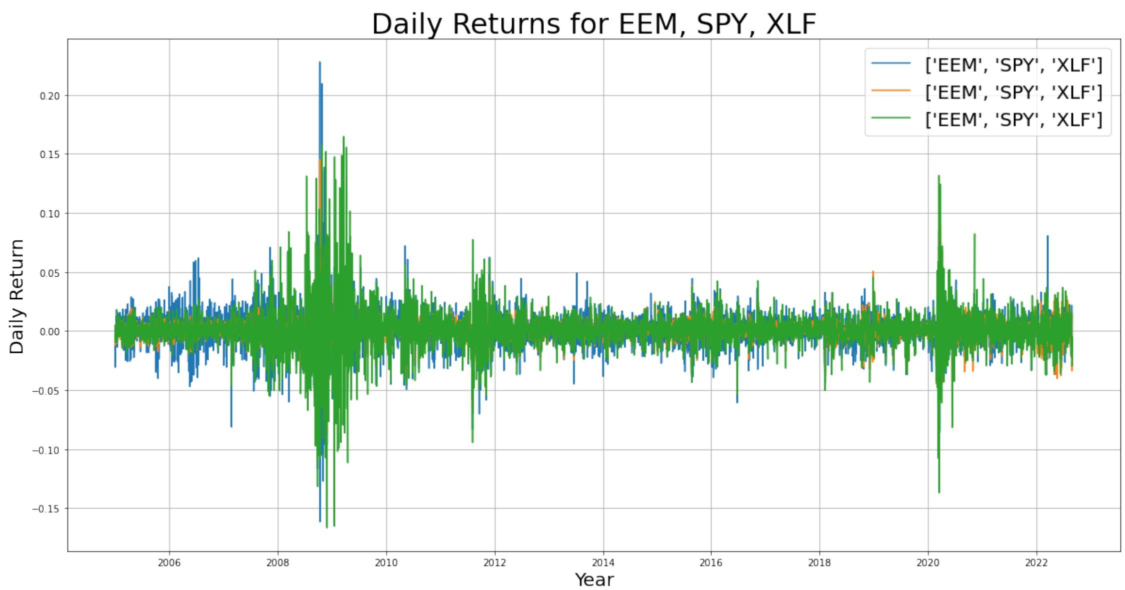


As is seen the SPY index has outperformed. If prices are very different, it can be good to normalise when plotting. You can also just think of the graph as the evolution after investing 100 USD in each asset.

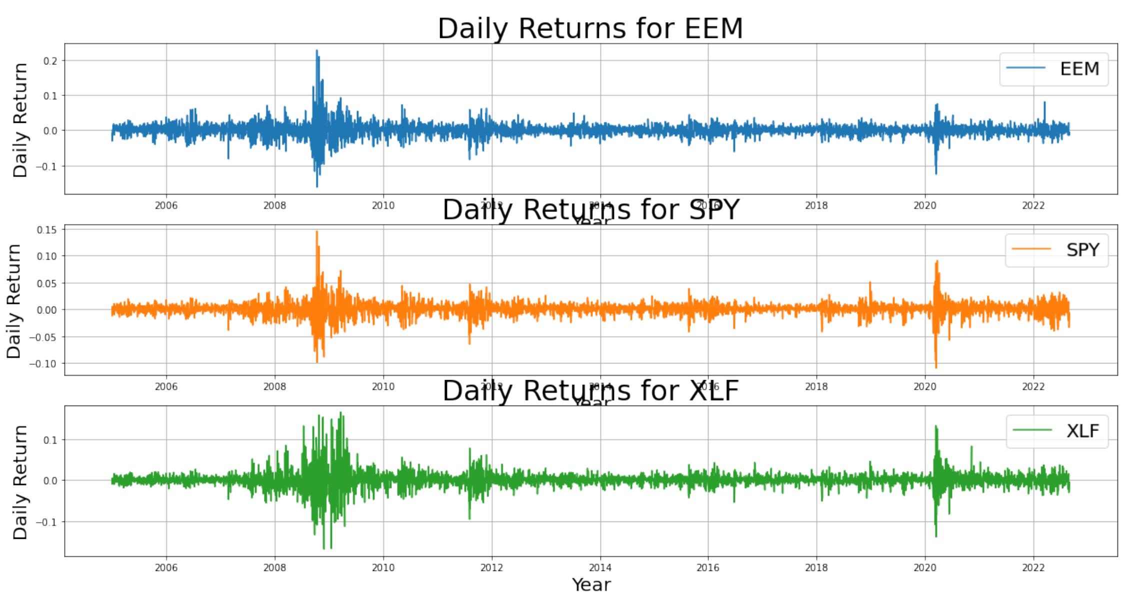
1.2.2 b) Calculate the daily returns, average daily returns, and standard deviation for each of the three securities

In this solution, the Pandas *pct_change()* is used to compute the returns for each ETF. Hereafter, we are using the *gmean* from the Scipy library to compute the (daily) geometric mean of the returns and the (daily) standard deviation using the *std* function from the Numpy library.

[ ]:
```
returns = stocks['Adj Close'].pct_change()

plt.figure(figsize=(20,10))
plt.plot(returns, label=list(stocks['Adj Close']))
plt.title('Daily Returns for EEM, SPY, XLF', fontsize=30)
plt.xlabel('Year', fontsize=20)
plt.ylabel('Daily Return', fontsize=20)
plt.legend(fontsize=20)
plt.grid()
plt.show()
```



[ ]:
```
figs, axs = plt.subplots(3, 1, figsize=(20,10))
for i, etf in enumerate(list(returns)):
    axs[i].plot(returns[etf], label=etf, color=colors[i])
    axs[i].set_title('Daily Returns for {}'.format(etf), fontsize=30)
    axs[i].set_xlabel('Year', fontsize=20)
    axs[i].set_ylabel('Daily Return', fontsize=20)
    axs[i].legend(fontsize=20)
    axs[i].grid()
```



```
[ ]: geomean_returns_daily = np.zeros(len(ticks))
sd_returns_daily = np.zeros(len(ticks))

for i, etf in enumerate(list(returns)):
    geomean_returns_daily[i] = gmean(1 + returns[etf].to_numpy()[1:]) - 1
    sd_returns_daily[i] = np.std(returns[etf].to_numpy()[1:])

    print('Geometric Average of Daily Returns for {}:: {}%'.format(etf, np.
 ↪round(geomean_returns_daily[i]*100, 4)))
    print('Standard Deviation of Daily Returns for {}:: {}%\n'.format(etf, np.
 ↪round(sd_returns_daily[i]*100, 4)))
```

```
Geometric Average of Daily Returns for EEM:: 0.0201%
Standard Deviation of Daily Returns for EEM:: 1.8372%

Geometric Average of Daily Returns for SPY:: 0.0345%
Standard Deviation of Daily Returns for SPY:: 1.2276%

Geometric Average of Daily Returns for XLF:: 0.0147%
Standard Deviation of Daily Returns for XLF:: 1.9459%
```

1.2.3 c) Calculate the variance-covariance matrix of the securities' daily returns

Here we are using the Pandas inbuilt *cov* function which computes the variance/covariance matrix by giving the function the daily returns matrix as input.

```
[ ]: covmat = returns.cov()

print('Covariance matrix of daily returns')
print(covmat)
```

```
Covariance matrix of daily returns
          EEM       SPY       XLF
EEM  0.000338  0.000189  0.000256
SPY  0.000189  0.000151  0.000203
XLF  0.000256  0.000203  0.000379
```

1.2.4 d) Calculate the correlation of the securities' daily returns

We compute the correlation matrix by using the Pandas *corr()* function.

```
[ ]: corrmat = returns.corr()

print('Correlation matrix of daily returns')
print(corrmat)
```

```
Correlation matrix of daily returns
          EEM       SPY       XLF
```

```
EEM  1.000000  0.837251  0.714923
SPY  0.837251  1.000000  0.848189
XLF  0.714923  0.848189  1.000000
```

1.3 Question 4

In this part, we will look into annualizing the statistics of the ETF performances. Since we are dealing with daily observations, we need to find a way to "annualize" the mean return and the standard deviation of the stocks. The reason we want to annualize anything is to have a common unit for our statistics - just like SI-units in physics.

If we for instance had some daily and weekly data, it is impossible to compare these with each other if we don't have a common unit for both. And thus, we annualize the statistics.

1.3.1 a) Find the average return of the three ETFs in annual terms using the returns

If we the daily geometric mean, R^D of a stock, then we know how much we expect to get in return per day. And if we assume there is 252 business days on a year, then over a year, we expect to have gained $R^A = (1 + R^D)^{252} - 1$ in return. Why?

If we expect to gain R^D on a single day, then we expect to gain $R^{2d} = (1 + R^D)(1 + R^D) - 1$ over two days. Likewise, we expect to gain $R^{5d} = (1 + R^D)(1 + R^D)(1 + R^D)(1 + R^D)(1 + R^D) - 1$ over 5 days. And since we have 252 business days, we get that on an annual basis, we expect to gain $(1 + R^D)^{252} - 1$.

Just to refresh, these are the expected daily returns for each ETF.

[ ]:
```
for etf, geomean_daily in zip(ticks, geomean_returns_daily):
    print("{} daily geomean return:: {}%".format(etf, round(geomean_daily*100,␣
↪3)))
```

```
SPY daily geomean return:: 0.02%
XLF daily geomean return:: 0.034%
EEM daily geomean return:: 0.015%
```

Here we compute the annualized returns.

[ ]:
```
geomean_returns_annualized = (1 + geomean_returns_daily)**252 - 1
```

The following shows the expected return for each ETF on an annual basis.

[ ]:
```
for etf, geomean_annual in zip(ticks, geomean_returns_annualized):
    print("{} annual geomean return:: {}%".format(etf,␣
↪round(geomean_annual*100, 2)))
```

```
SPY annual geomean return:: 5.19%
XLF annual geomean return:: 9.07%
EEM annual geomean return:: 3.79%
```

1.3.2 b) Find the standard deviation of the three ETFs in annual terms

In order to annualize the standard deviation, we need to multiply with $\sqrt{252}$ since we are going from daily to annual. I recommend reading this page How to Annualize Volatility to understand why.

To refresh, here are our daily standard deviations of each ETF.

[ ]:
```
for etf, std_daily in zip(ticks, sd_returns_daily):
    print("{} daily standard deviation of returns:: {}%".format(etf,␣
 ↪round(std_daily*100, 2)))
```

```
SPY daily standard deviation of returns:: 1.84%
XLF daily standard deviation of returns:: 1.23%
EEM daily standard deviation of returns:: 1.95%
```

Computing the annualized standard deviations.

[ ]:
```
sd_returns_annualized = np.sqrt(252) * sd_returns_daily
```

Following cell shows the standard deviation in annual basis.

[ ]:
```
for etf, std_annual in zip(ticks, sd_returns_annualized):
    print("{} annual standard deviation of returns:: {}%".format(etf,␣
 ↪round(std_annual*100, 2)))
```

```
SPY annual standard deviation of returns:: 29.16%
XLF annual standard deviation of returns:: 19.49%
EEM annual standard deviation of returns:: 30.89%
```

1.3.3 c) Find the variance-covariance matrix in annual terms

Again, we are refreshing the covariance matrix that we computed in part 2.c.

[ ]:
```
covmat
```

[ ]:

	EEM	SPY	XLF
EEM	0.000338	0.000189	0.000256
SPY	0.000189	0.000151	0.000203
XLF	0.000256	0.000203	0.000379

Here we are annualizing the covariance matrix.

[ ]:
```
covmat_annualized = 252 * covmat
```

The following cell shows the covariance matrix in annual basis.

[ ]:
```
covmat_annualized
```

[ ]:

	EEM	SPY	XLF
EEM	0.085073	0.047594	0.064421
SPY	0.047594	0.037985	0.051071

```
XLF  0.064421  0.051071  0.095444
```

1.3.4 d) Do we need to do anything with the correlation matrix to get it in annual terms? Why/why not?

In order to answer this question, we need to consider how correlation is calculated.

$$\rho = \frac{cov(x,y)}{\sigma_x \sigma_y}$$

Lets imagine that the covariance and standard deviations are daily, i.e:

$$\rho = \frac{cov(x,y)^D}{\sigma_x^D \sigma_y^D}$$

then let's annualize each term. I.e. annualize the covariance and the standard deviations:

$$\rho = \frac{cov(x,y)^D 252}{\sigma_x^D \sqrt{252} \sigma_y^D \sqrt{252}} = \frac{cov(x,y)^D 252}{\sigma_x^D \sigma_y^D 252} = \frac{cov(x,y)^A}{\sigma_x^A \sigma_y^A}$$

We see that we get the same correlation if we use daily standard deviations and covariance as if we used annualized terms. This is because correlation is unitless and thus not dependent on time. Therefore we do not need to annualize the correlation matrix.

1.4 Question 5

1.4.1 a) Plot the week weekly returns. Whay do you see? Can you identify any major events by inspecting the data?

Lets display the data matrix of the XOM stock.

[ ]:
```
xom
```

[ ]:
```
                 Open        High        Low       Close    Adj Close  \
Date
2004-12-27  51.009998   51.400002  51.000000   51.259998   28.450693
2005-01-03  51.020000   51.020000  49.250000   49.790001   27.634802
2005-01-10  49.860001   51.209999  49.630001   51.070000   28.345232
2005-01-17  50.950001   51.549999  50.349998   50.439999   27.995577
2005-01-24  50.939999   51.950001  50.910000   51.270000   28.456244
…                …          …          …          …           …
2022-08-01  94.790001   95.349998  86.279999   88.449997   87.598862
2022-08-08  88.610001   94.300003  88.220001   94.000000   93.095459
2022-08-15  90.529999   95.309998  89.660004   94.080002   94.080002
2022-08-22  93.419998   99.910004  91.860001   97.870003   97.870003
2022-08-29  98.180000  101.559998  98.139999  100.120003  100.120003

                 Volume   Returns  Log Returns
Date
2004-12-27    8497000.0       NaN          NaN
2005-01-03   69330300.0 -0.028677    -0.029097
```

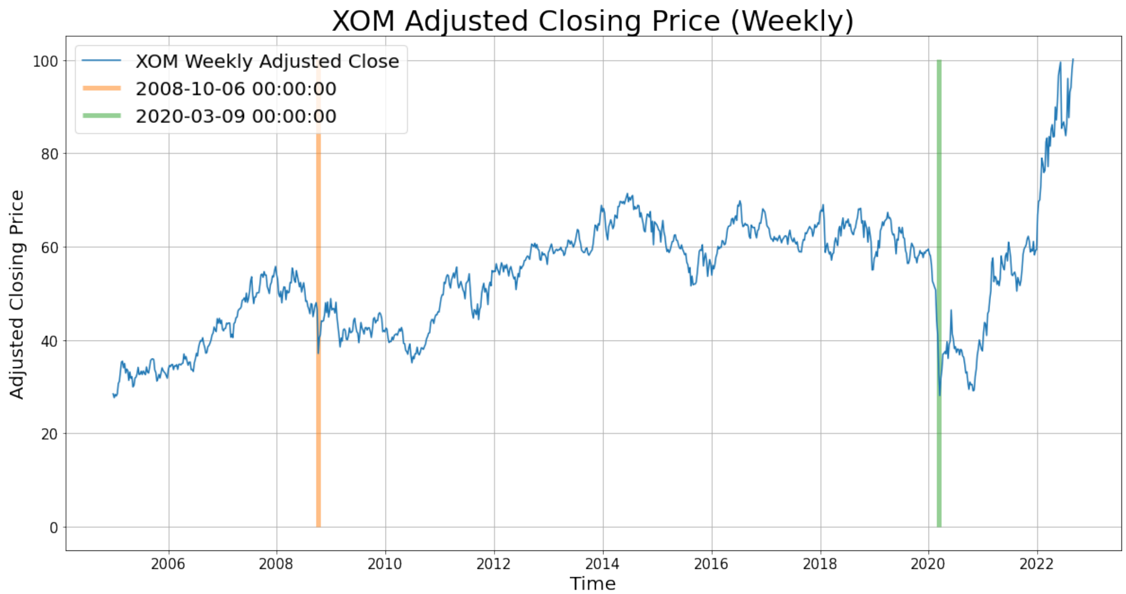
```
2005-01-10   56046800.0  0.025708     0.025383
2005-01-17   48206200.0 -0.012336    -0.012412
2005-01-24   58189200.0  0.016455     0.016321
…                 …         …        …
2022-08-01  110318700.0 -0.087486    -0.091552
2022-08-08   85694400.0  0.062747     0.060857
2022-08-15   82368000.0  0.010576     0.010520
2022-08-22   87672000.0  0.040285     0.039495
2022-08-29   23059200.0  0.022990     0.022729

[923 rows x 8 columns]
```

We can see some big losses, so we choose to consider the worst returns. In the following cell, we find all the dates where the return is below the 0.2% quantile. I.e. we find the most extreme losses for the XOM stock. As the cell shows, we get two observations, one in 2008 and the other in 2020. The first is during the financial crisis and the second in the beginning of Covid-19.

[ ]:
```
crash_dates = xom.index[xom.Returns.between(xom.Returns.quantile(0), xom.
↪Returns.quantile(0.002))]
crash_dates
```

[ ]:
```
DatetimeIndex(['2008-10-06', '2020-03-09'], dtype='datetime64[ns]', name='Date',
freq=None)
```

[ ]:
```
plt.figure(figsize=(20,10))
plt.plot(xom['Adj Close'], label='XOM Weekly Adjusted Close')
plt.title('XOM Adjusted Closing Price (Weekly)', fontsize=30)
plt.ylabel('Adjusted Closing Price', fontsize=20)
plt.xlabel('Time', fontsize=20)
plt.grid()
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)

for i, crash in enumerate(crash_dates):
    plt.vlines(x=crash, linewidth=5, alpha=0.5, ymin=0, ymax=xom['Adj Close'].
↪max(), label=crash, color="C{}".format(i+1))
plt.legend(fontsize=20)
plt.show()
```



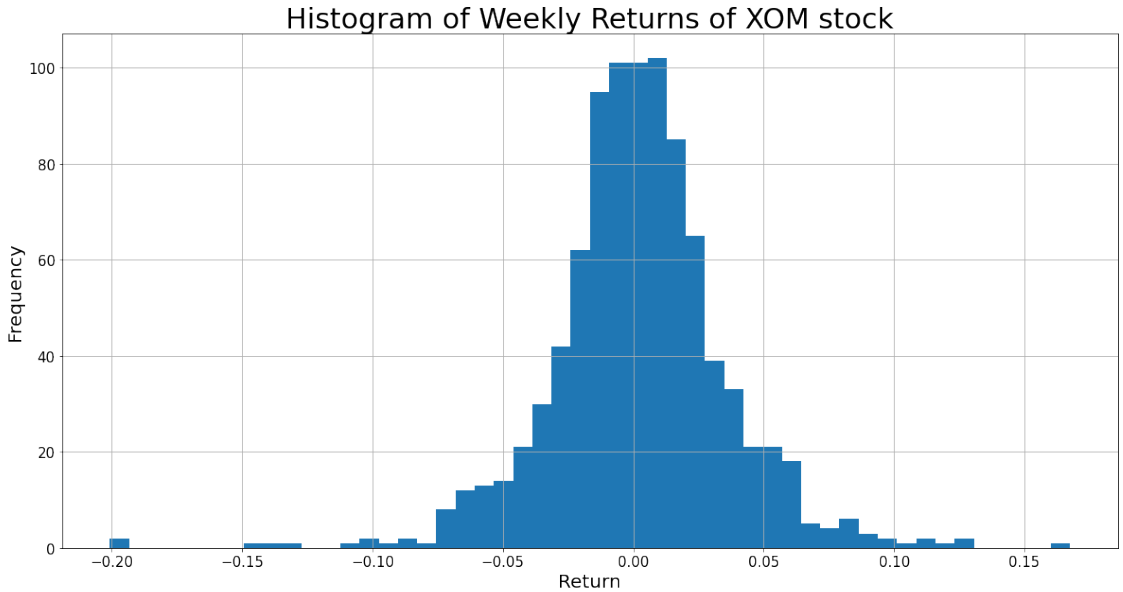
The crash on 06/10/2008 is a reaction to the 2008 financial crisis after Lehman Brothers collapsed and other banks such as Bear Stearns, citygroup and the CDS provider AIG was on the brink of collapse.

The crash on 03/09/2020 was an reaction to the countries beginning to go in lockdowns across the globe and the high uncertainty at that time.

1.4.2 b) Make a histogram of weekly returns. What do you see?

The following code computes the histogram of the weekly returns.

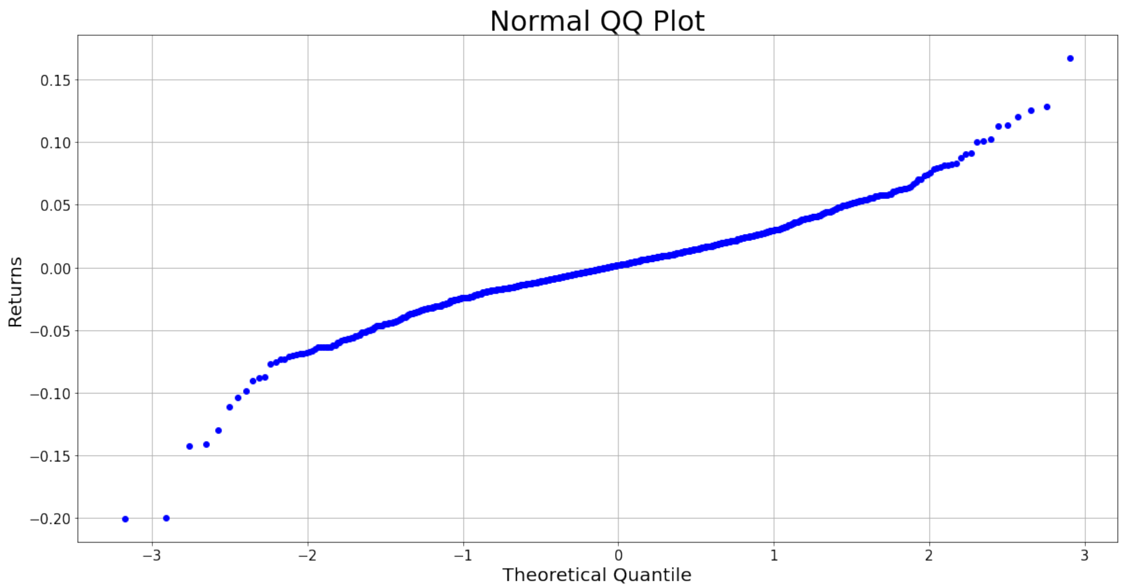
[ ]:
```
plt.figure(figsize=(20, 10))
plt.hist(xom['Returns'], bins=50)
plt.title('Histogram of Weekly Returns of XOM stock', fontsize=30)
plt.xlabel('Return', fontsize=20)
plt.ylabel('Frequency', fontsize=20)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.grid()
plt.show()
```



1.4.3 c) Make a QQ-plot of weekly returns. Do weekly returns seem for follow a normal distribution?

Now we will make a QQ plot of the weekly returns.

[ ]:
```
plt.figure(figsize=(20,10))
stats.probplot(x=xom.Returns, dist='norm', plot=plt)
#stats.probplot(x=np.random.normal(size = 10000), dist='norm', plot=plt)
plt.title('Normal QQ Plot', fontsize=30)
plt.xlabel('Theoretical Quantile', fontsize=20)
plt.ylabel('Returns', fontsize=20)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.grid()
plt.show()
```



It seems as if the weekly returns has some heavier tails than a normal distribution.

1.4.4 d) Calculate the skewness and excess kurtosis. Do these value indicate that weekly stock returns follow a normal distribution?

[ ]:
```
skewness = xom.Returns.skew()
kurtosis = xom.Returns.kurtosis()

print("Skewness of Weekly Returns:: {}".format(round(skewness, 4)))
print("Kurtosis of Weekly Returns:: {}".format(round(kurtosis, 4)))
```

```
Skewness of Weekly Returns:: -0.3315
Kurtosis of Weekly Returns:: 4.3874
```

As you may or may not know, a normal distribution has a skewness of 0 and a excess kurtosis of 0. This means that the distribution of the returns seems to be close to a normal distribution but is slightly skewed to the left and has heavier tails than a normal distribution.