STATISTICS PROJECT 3

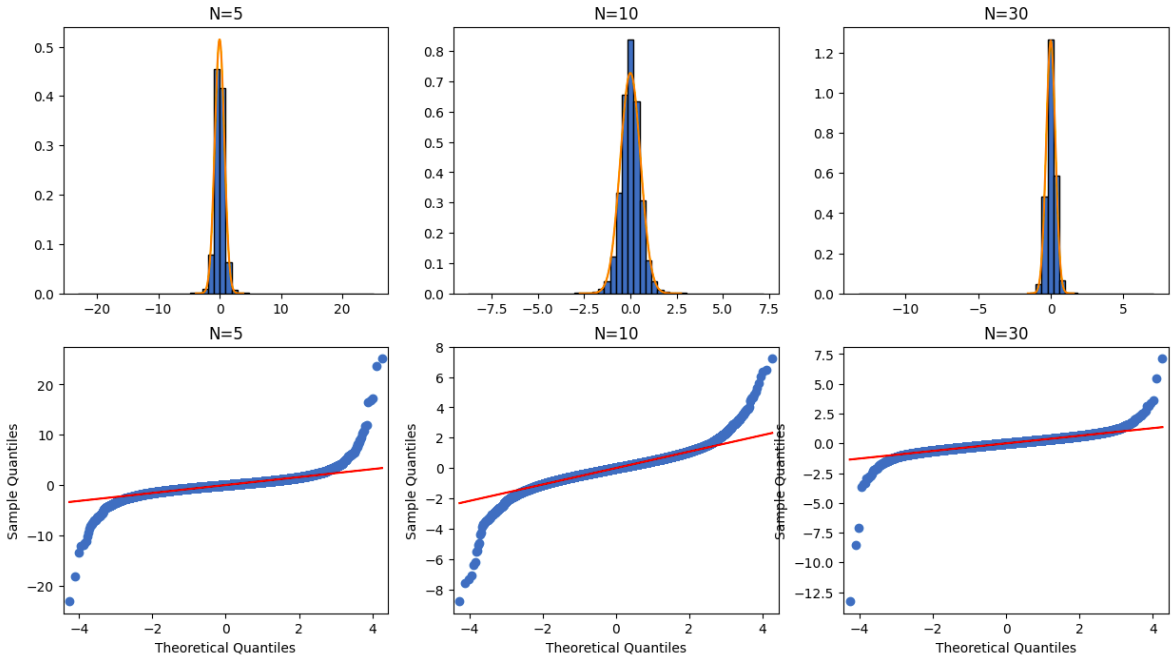
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy.stats as stats
import pandas_datareader.data as web
import statsmodels.api as sm
from statistics import mean
from scipy.stats import t
from scipy.stats import skew, kurtosis
from scipy.optimize import minimize
from scipy.stats import logistic
!pip install pandas_datareader
import warnings
warnings.filterwarnings("ignore")
```

1.Follow these steps: (a) Create a 2-by-3 array of subplots. Fix k = 3 and instantiate an array N = [5, 10, 30] and a variable size = 100,000. (b) Using a for loop, for each value n in N sample a size × n array of samples from the distribution t(k) (c) Compute the sample average along each row (i.e. you should get size sample averages), and plot their histogram in a subplot. (d) Over the histogram (i.e. in the same subplot), plot the approximate density of the distribution of sample averages which is given by the CLT as described above. (e) In a separate subplot, display the QQ plot of the sample means versus their approximate distribution For which value N is the approximate density of sample means given by the CLT a good approximation of the actual distribution from which you've drawn samples? Briefly justify your answer.

```
N = [5,10,30]
K = 3
size = 100000
sample = [0]*3
sample_mean = [0]*3
sample_mean_pdf = [0]*3
stud_mean = 0

def stud_variance(number):
    return K/((K-2)*number)

for ind, n in enumerate(N):
    array = [0]*size
    for j in range(size):
        array[j] = t.rvs(K,size=n)
    sample[ind] = array
for ind, n in enumerate(N):
    array_1 = [0]*size
```

```
    for j in range(size):
        array_1[j] = mean(sample[ind][j])
    sample_mean[ind] = array_1

samples = [0]*3
for ind, n in enumerate(N):
    samples[ind] = np.linspace(stud_mean - 5 *
np.sqrt(stud_variance(n)), stud_mean + 5 * np.sqrt(stud_variance(n)),
100)
    sample_mean_pdf[ind]=stats.norm.pdf(samples[ind], stud_mean,
np.sqrt(stud_variance(n)))
```

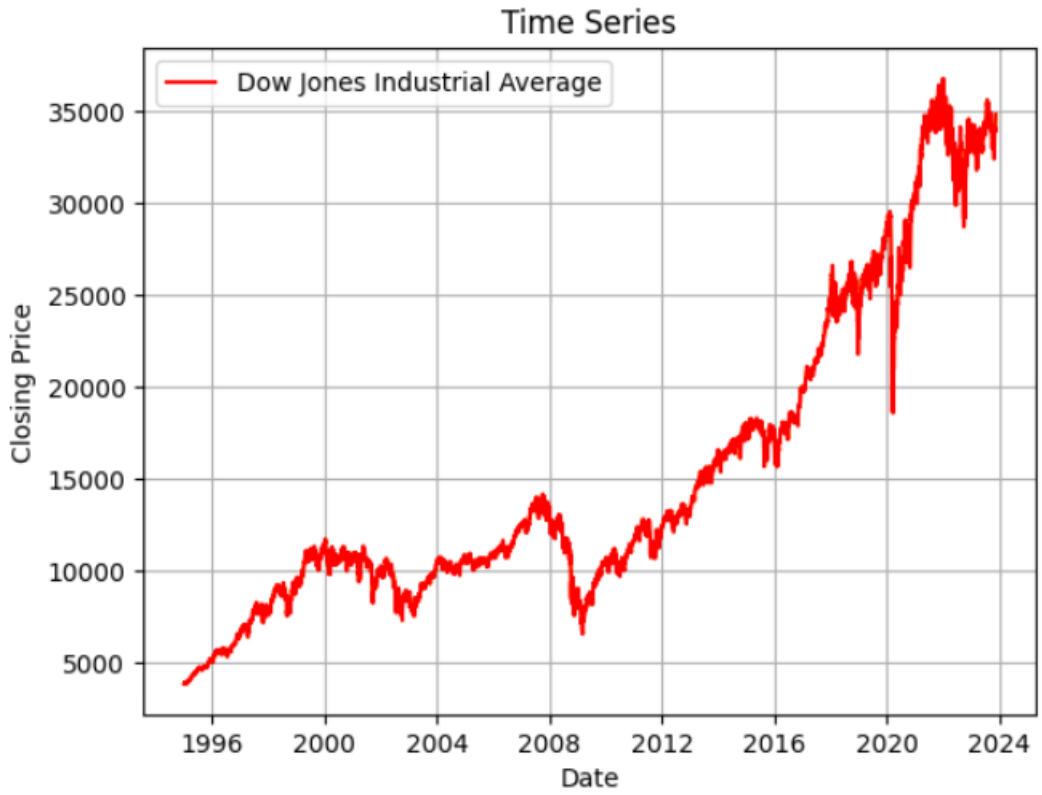
```
fig, axs = plt.subplots(2, 3, figsize=(15, 8))

#plotting Histogram
axs[0, 0].hist(sample_mean[0],bins=50, density=True, edgecolor=
'black')
axs[0,0].plot(samples[0],sample_mean_pdf[0])
axs[0, 0].set_title('N=5')

axs[0, 1].hist(sample_mean[1],bins=50, density=True, edgecolor=
'black')
axs[0,1].plot(samples[1],sample_mean_pdf[1])
axs[0, 1].set_title('N=10')

axs[0, 2].hist(sample_mean[2],bins=50, density=True, edgecolor=
'black')
axs[0,2].plot(samples[2],sample_mean_pdf[2])
axs[0, 2].set_title('N=30')


#qq plotting
sm.qqplot(np.array(sample_mean[0]),line='s',ax=axs[1, 0]);
axs[1, 0].set_title('N=5')

sm.qqplot(np.array(sample_mean[1]),line='s',ax=axs[1, 1]);
axs[1, 1].set_title('N=10')

sm.qqplot(np.array(sample_mean[2]),line='s',ax=axs[1, 2]);
axs[1, 2].set_title('N=30')

plt.show()
```



Justification : The Central Limit Theorem (CLT) states that as the sample size increases, the distribution of the sample means becomes approximately normal, regardless of the shape of the original distribution.By observing the three visualisation, the third, that is N=30 is giving a good approximation of the Central Limit Theorem. In the both histogram and the pdf plotting, we can observe that, the graph is gradually coverging in to normal distribution with the increase of sample size. Again in the qq plot we can confirm that the quantile of sample means follows the quantiles of normal distribution. Hence N=30 is a good approximation.

Plot this times series.

```
data = web.DataReader('^DJI', 'stooq', start='1995-01-01', end='2023-11-14')
data = data.reset_index()
dates = data["Date"]
dow = data["Close"].to_numpy()

#plotting time series
plt.plot(dates, dow,label='Dow Jones Industrial Average',color='red')
plt.xlabel('Date')
plt.ylabel('Closing Price')
plt.title('Time Series')
plt.legend()
plt.grid(True)
plt.show()
```



Compute the time series of (percentage) daily returns using the formula Returnt = 100 ×(Closet/Closet−1)− 1

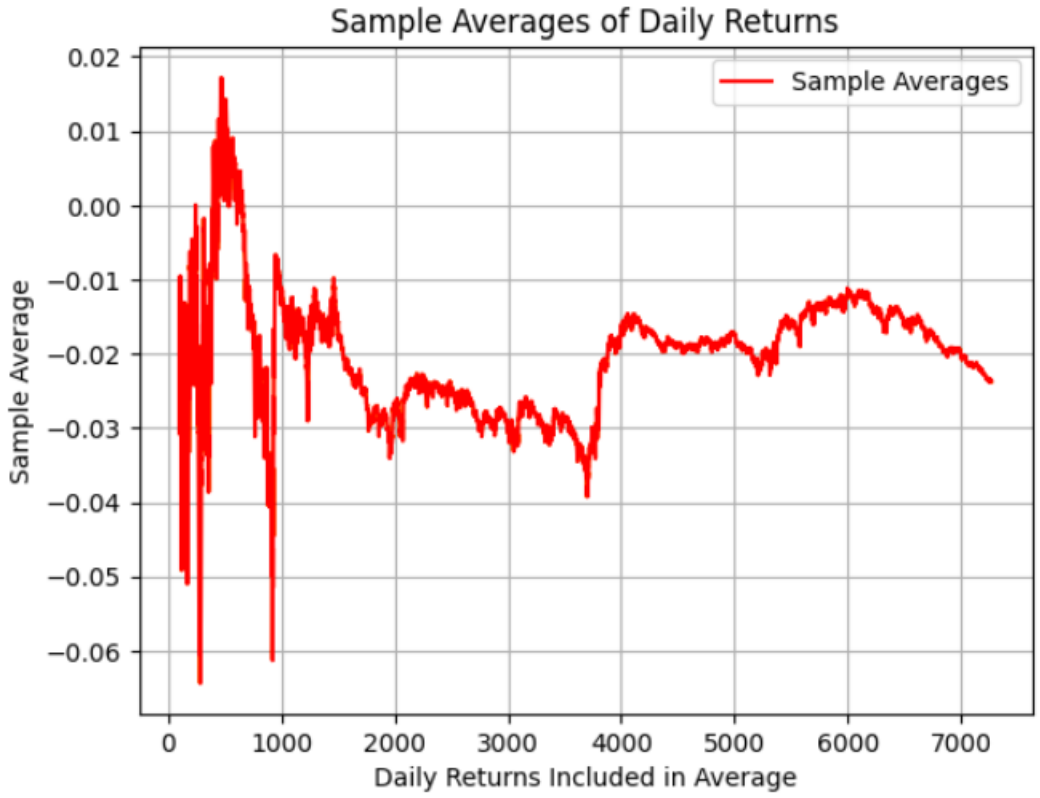
```
data['Return'] = 100*(data["Close"]/data["Close"].shift(1)-1).dropna()
data['Return']
```

```
0           NaN
1     -1.406438
2     -0.159503
3     -1.140970
4      0.650096
         ...
7264  -0.137068
7265   0.155384
7266  -0.426643
7267   0.176582
7268  -0.497706
Name: Return, Length: 7269, dtype: float64
```

Compute the length-n sample averages of daily returns, starting at the first datapoint, for every n ≥ 100. Thus the first datapoint in this time series will be the average of the first 100 daily returns, the second will be the average of the first 101 daily returns, etc., and the last will be the average of all daily returns. Plot this timeseries. Does it look like it obeys the weak Law of Large Numbers? If yes explain why, if not explain why this might be the case.

```
sample_average = []
n_values = range(100, len(data['Return']) + 1)
for i in n_values:
    sample_average.append(np.mean(data['Return'][:i]))
```

```
plt.plot(n_values, sample_average, label='Sample Averages',
color='red')
plt.grid(True)
plt.xlabel('Daily Returns Included in Average')
plt.ylabel('Sample Average')
plt.title('Sample Averages of Daily Returns')
plt.legend()
plt.show()
```



Justification : Yes it obeys the weak law of large large numbers, because by looking into the graph we can observe that, as the sample size of sample averages increases the graph starts converging towards the true mean.

Compute the length-100 rolling averages of daily returns. Plot a histogram of these sample averages. Repeat with length-400 rolling averages. Does it look like these obey the Central Limit Theorem? If yes explain why, if not explain why this might be the case.

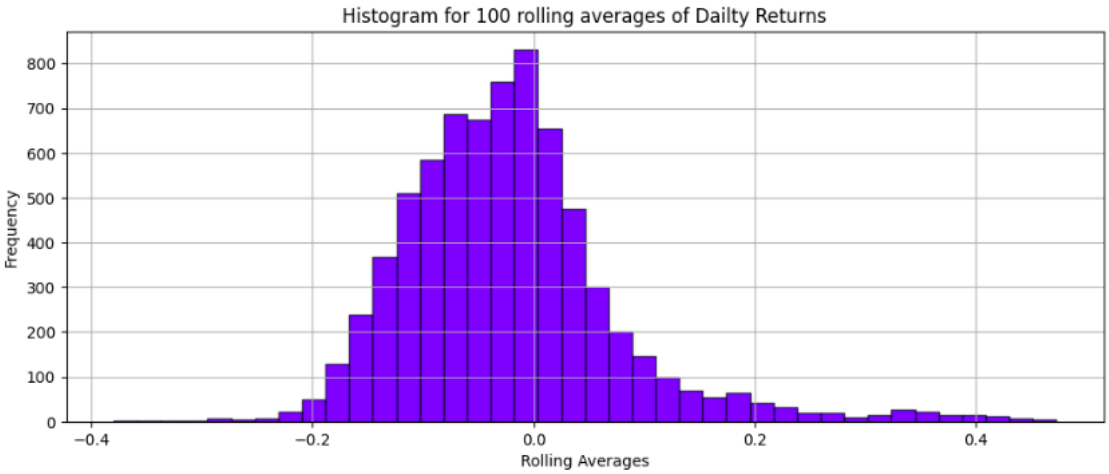
```
rolling_average_100 =
data['Return'].rolling(window=100).mean().dropna()
print(rolling_average_100)
```

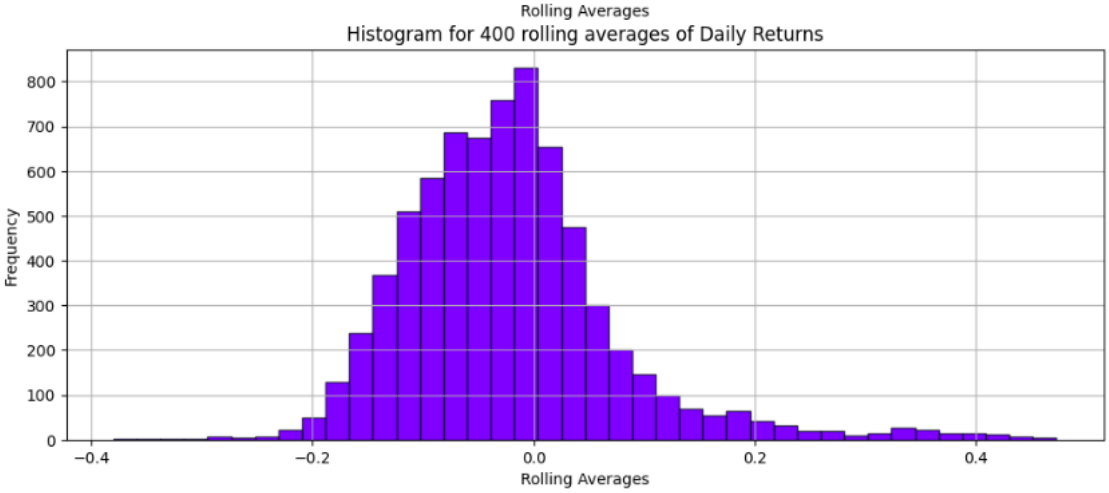
```
100    -0.030040
101    -0.009474
102    -0.007737
103     0.006687
104     0.007388
          ...
7264   -0.145311
7265   -0.142036
7266   -0.126952
7267   -0.122971
7268   -0.137836
Name: Return, Length: 7169, dtype: float64
```

```
rolling_average_400 =
data['Return'].rolling(window=100).mean().dropna()
print(rolling_average_400)
```

```
100    -0.030040
101    -0.009474
102    -0.007737
103     0.006687
104     0.007388
          ...
7264   -0.145311
7265   -0.142036
7266   -0.126952
7267   -0.122971
7268   -0.137836
Name: Return, Length: 7169, dtype: float64
```

```
fig,axs=plt.subplots(2,1,figsize=(12,10))

axs[0].hist(rolling_average_100,bins=40,color='blue', alpha=0.7,
edgecolor='black')
axs[0].set_title('Histogram for 100 rolling averages of Dailty
Returns')
axs[0].grid(True)
axs[0].set_xlabel('Rolling Averages')
axs[0].set_ylabel('Frequency')

axs[1].hist(rolling_average_400,bins=40,color='blue', alpha=0.7,
edgecolor='black')
axs[1].set_title('Histogram for 400 rolling averages of Daily Returns')
axs[1].grid(True)
axs[1].set_xlabel('Rolling Averages')
axs[1].set_ylabel('Frequency')


plt.suptitle('Histograms for Rolling Averages')

plt.show()
```





Justification : No, it does not obeys central limit theorem. We can clearly see that, the distribution formed above is not normal distribution. The distribution formed above is a skewed distribution, because the plot above has heavy tails which means there are some extreme values present in the dataset that shifting the probability mass towards them thus results in the skewed distribution. central limit theorem requires data that are independent of other observations but here the case is, it is dependent on time series, it might be the reason for the data not following the central limit theorem.

Compute the sample mean, variance, skewness and kurtosis of the daily returns. Based on this information, suggest which family of distributions might model these daily returns. Briefly justify your choice.

```
descriptive_statistics = data['Return'].describe()
descriptive_statistics
```

```
count    7268.000000
mean       -0.023723
std         1.152497
min       -10.205212
25%        -0.576726
50%        -0.055137
75%         0.459137
max        14.845566
Name: Return, dtype: float64
```

```
print(f"Mean: {descriptive_statistics['mean'] :.5f}")
print(f"Kurtosis: {data['Return'].kurt() :.5f}")
print(f"Skewness: {data['Return'].skew() :.5f}")
variance = np.var(data['Return'])
print(f"variance: {variance :.5f}")
```

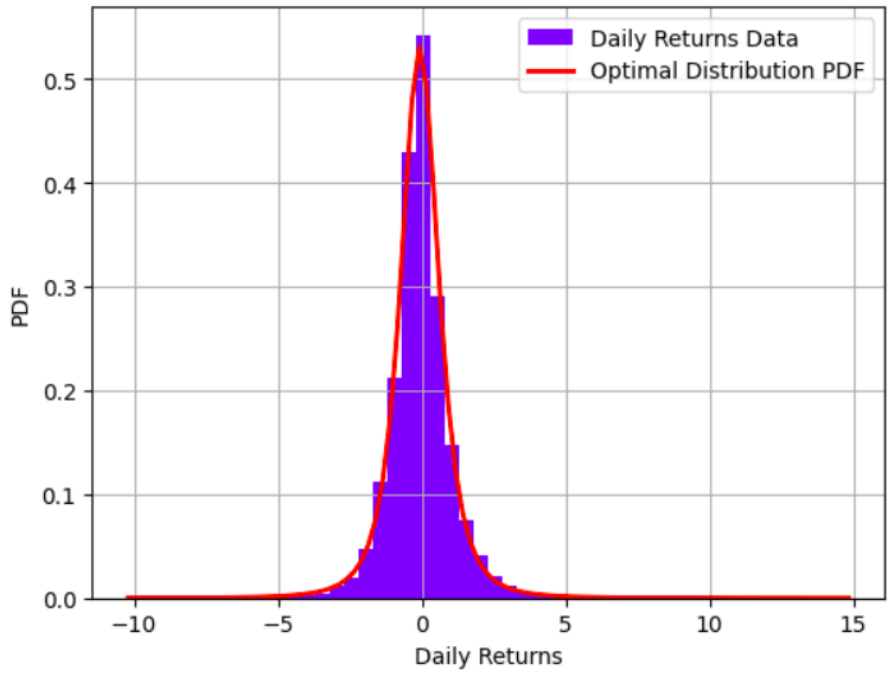
```
Mean: -0.02372
Kurtosis: 12.98757
Skewness: 0.64820
variance: 1.32807
```

Justification : The fact that the kurtosis's heavy tail indicates that the distribution is not normal can be inferred from this. Still, it also demonstrates how closely it adheres to the normal distribution, with the variance almost equal to 1 and the mean almost equal to 0, however slightly skewed. This confirms for me that the t distribution is the distribution that closely resembles the heavy-tailed normal distribution.

For this choice of family, you will now estimate the parameter(s) which best explain the data using the Maximum Likelihood Estimator approach. To achieve this: • Implement the function which needs to be maximized (this was explained in the lectures). The parameter(s) which you are trying to estimate must of course be inputs to this function. • Using the minimize function from scipy.optimize, find the optimal parameters. (Hint: maximizing f(x) is the same thing as minimizing −f(x)). You can use any of the actual minimization methods, as long as it gives you a sensible answer. • Once you have found the optimal parameters, plot the PDF of your optimal distribution against a histogram of the daily returns.

```
from scipy.optimize import minimize

data_1 = data['Return'].dropna()

# Define the negative log-likelihood function
def neg_log_likelihood(params):
    df, loc, scale = params
    dist = t(df, loc, scale)
    log_likelihood = np.sum(np.log(dist.pdf(data_1)))
    return -log_likelihood

# Initial guess for parameters
initial_params = [3, 0, 1]

# Minimize the negative log-likelihood function to find optimal
parameters
result = minimize(neg_log_likelihood, initial_params, method='BFGS')

# Extract optimal parameters
optimal_params = result.x
optimal_dist = t(*optimal_params)
```

```
x = np.linspace(data_1.min(), data_1.max(), 100)

plt.hist(data_1, bins=50, density=True, alpha=0.7, color='blue',
label='Daily Returns Data')
plt.plot(x, optimal_dist.pdf(x), 'r', linewidth=2, label='Optimal
Distribution PDF')
plt.xlabel('Daily Returns')
plt.ylabel('PDF')
plt.grid(True)
plt.legend()
plt.show()

print(f"Optimal Parameters (location, scale, degrees of freedom):
{optimal_params}")
```




```
Optimal Parameters (location, scale, degrees of freedom): [ 2.85348897 -0.06273683  0.68881381]
```

```
maximum_likelihood_params = t.fit(data_1)
print('Maximum Likelihood Location
Parameter',maximum_likelihood_params[0])
print('Maximum Likelihood Scale
Parameter',maximum_likelihood_params[1])

fit_dist = t(*maximum_likelihood_params)
```

```
Maximum Likelihood Location Parameter 2.853495932310387
Maximum Likelihood Scale Parameter -0.06274107728130038
```

Check the results you obtained in the previous step by comparing it with the parameters you obtain from scipy's fit function. Again, plot the PDF of the distribution with these parameters against a histogram of the daily returns.

```
plt.plot(x, fit_dist.pdf(x), '*b', color = 'black', linewidth=2,
label='Student\'s t Fit Distribution PDF ')
plt.plot(x, optimal_dist.pdf(x),'--r', linewidth=2, label='Student\'s t
Optimal Distribution PDF ')
plt.legend()
plt.show()
```