Scientific course work 3rd year

Why Are Logarithmic Returns Important in Finance?

Logarithmic returns are important in finance because they provide a more accurate measure of the percentage change in the value of an asset over a period of time. This is particularly important when analyzing financial data because the compounding effect of returns over time can have a significant impact on the value of an asset.

Logarithmic returns are also useful because they are additive. That is, the logarithmic return of a portfolio composed of multiple assets is simply the sum of the logarithmic returns of each individual asset. This makes it easy to calculate the overall performance of a portfolio over a period of time.

Another advantage of using logarithmic returns is that they are normally distributed, which makes them easier to work with mathematically. This is because the natural logarithm of a number is a continuous and smooth function, which means that the resulting distribution of logarithmic returns is also continuous and smooth.

By using logarithmic returns, we can get a more accurate measure of the percentage change in the value of an asset over a period of time, which is crucial when working with financial data.

```
!pip3 install datetime
!pip3 install yfinance
!pip3 install bs4
!pip3 install os
!pip3 install datetime
!pip3 install pandas
!pip3 install pandas_datareader
!pip3 install statsmodels
```

```
import bs4 as bs
import requests
import yfinance as yf
import datetime
```

Below we downloaded all historical data about S&P500 index from 01-05-1981 to our present days/nowadays

```
resp = requests.get('http://en.wikipedia.org/wiki/List_of_S%26P_500_companies')
soup = bs.BeautifulSoup(resp.text, 'lxml')
table = soup.find('table', {'class': 'wikitable sortable'})
tickers = []
for row in table.findAll('tr')[1:]:
    ticker = row.findAll('td')[0].text
    tickers.append(ticker)

tickers = [s.replace('\n', '') for s in tickers]
start = datetime.datetime(1981,5,1)
end = datetime.datetime.now()
data = yf.download(tickers, start=start, end=end)
print(data)
```

```
[*********************100%%**********************]  503 of 503 completed

2 Failed downloads:
['BRK.B']: Exception('%ticker%: No timezone found, symbol may be delisted')
['BF.B']: Exception('%ticker%: No price data found, symbol may be delisted (1d 1981-05-01 00:0
Price         Adj Close
Ticker                A    AAL        AAPL        ABBV        ABNB         ABT
Date
1981-05-01          NaN    NaN    0.097898         NaN         NaN    0.320195  \
1981-05-04          NaN    NaN    0.097467         NaN         NaN    0.320195
1981-05-05          NaN    NaN    0.097036         NaN         NaN    0.315301
1981-05-06          NaN    NaN    0.094448         NaN         NaN    0.313204
1981-05-07          NaN    NaN    0.095742         NaN         NaN    0.317399
...                 ...    ...         ...         ...         ...         ...
2024-04-24   137.490005  13.92  169.020004  167.800003  162.839996  106.889999
2024-04-25   136.369995  14.13  169.889999  167.289993  163.009995  106.860001
2024-04-26   137.740005  13.88  169.300003  159.619995  164.229996  107.529999
2024-04-29   139.589996  13.98  173.500000  161.520004  162.250000  107.269997
2024-04-30   137.039993  13.51  170.330002  162.639999  158.570007  105.970001

Price                                                             ...     Volume
Ticker            ACGL         ACN        ADBE         ADI  ...        WTW
Date                                                          ...
1981-05-01         NaN         NaN         NaN    0.808018  ...        NaN  \
1981-05-04         NaN         NaN         NaN    0.800135  ...        NaN
1981-05-05         NaN         NaN         NaN    0.796193  ...        NaN
1981-05-06         NaN         NaN         NaN    0.804076  ...        NaN
1981-05-07         NaN         NaN         NaN    0.811960  ...        NaN
...                ...         ...         ...         ...  ...        ...
2024-04-24   93.190002  313.540009  477.119995  196.500000  ...   480500.0
2024-04-25   93.050003  309.000000  473.440002  197.940002  ...  1476100.0
2024-04-26   90.900002  308.010010  477.559998  201.970001  ...   700700.0
2024-04-29   91.250000  303.160004  473.070007  203.899994  ...   474300.0
2024-04-30   93.540001  300.910004  462.829987  200.610001  ...   486923.0

Price
Ticker            WY       WYNN      XEL       XOM        XYL        YUM
Date
1981-05-01    102450        NaN    49600   5251200        NaN        NaN  \
1981-05-04    136800        NaN   472400   4275200        NaN        NaN
1981-05-05     87600        NaN   264000   6878400        NaN        NaN
1981-05-06    128700        NaN    67600   7416000        NaN        NaN
1981-05-07     64950        NaN    96400   9086400        NaN        NaN
...              ...        ...      ...       ...        ...        ...
2024-04-24   3019100  1256700.0  4614400  12101200  1053000.0  1909500.0
```

```
2024-04-25  2473900   948900.0  6717500  16041000   963600.0  1693100.0
2024-04-26  3616700  1379700.0  6334100  26718400  1112300.0  1979100.0
2024-04-29  2837700  1588300.0  3884600  17309900  1277800.0  2221000.0
2024-04-30  4608360  2625609.0  6804275  20770037  1282551.0  3504826.0

Price
Ticker            ZBH      ZBRA        ZTS
Date
1981-05-01        NaN       NaN        NaN
1981-05-04        NaN       NaN        NaN
1981-05-05        NaN       NaN        NaN
```

```
import pandas as pd
import numpy as np
import yfinance as yf
import matplotlib.pyplot as plt
from pandas_datareader import data
import plotly.express as px
import datetime as dt
import seaborn as sns
```

Now, let's extract the historical prices Apple company from yahoo finance using a python library yfinance from 01-05-1981 to the present days/nowadays. (AAPL is one of the 500 S&P 500 stocks)

```
sp500_aapl = yf.download('AAPL', start='1981-05-01', end =dt.datetime.now())
sp500_aapl.head()
```

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

Date	Open	High	Low	Close	Adj Close	Volume
1981-05-01	0.126674	0.127790	0.126674	0.126674	0.097898	16553600
1981-05-04	0.126674	0.126674	0.126116	0.126116	0.097467	14448000
1981-05-05	0.126116	0.126116	0.125558	0.125558	0.097036	17539200
1981-05-06	0.122768	0.122768	0.122210	0.122210	0.094448	18950400
1981-05-07	0.123884	0.124442	0.123884	0.123884	0.095742	9363200

```
sp500_aapl.tail()
```

Date	Open	High	Low	Close	Adj Close	Volume
2024-04-23	165.350006	167.050003	164.919998	166.899994	166.899994	49537800
2024-04-24	166.539993	169.300003	166.210007	169.020004	169.020004	48251800
2024-04-25	169.529999	170.610001	168.149994	169.889999	169.889999	50558300
2024-04-26	169.880005	171.339996	169.179993	169.300003	169.300003	44525100
2024-04-29	173.369995	176.029999	173.100006	173.500000	173.500000	66162100

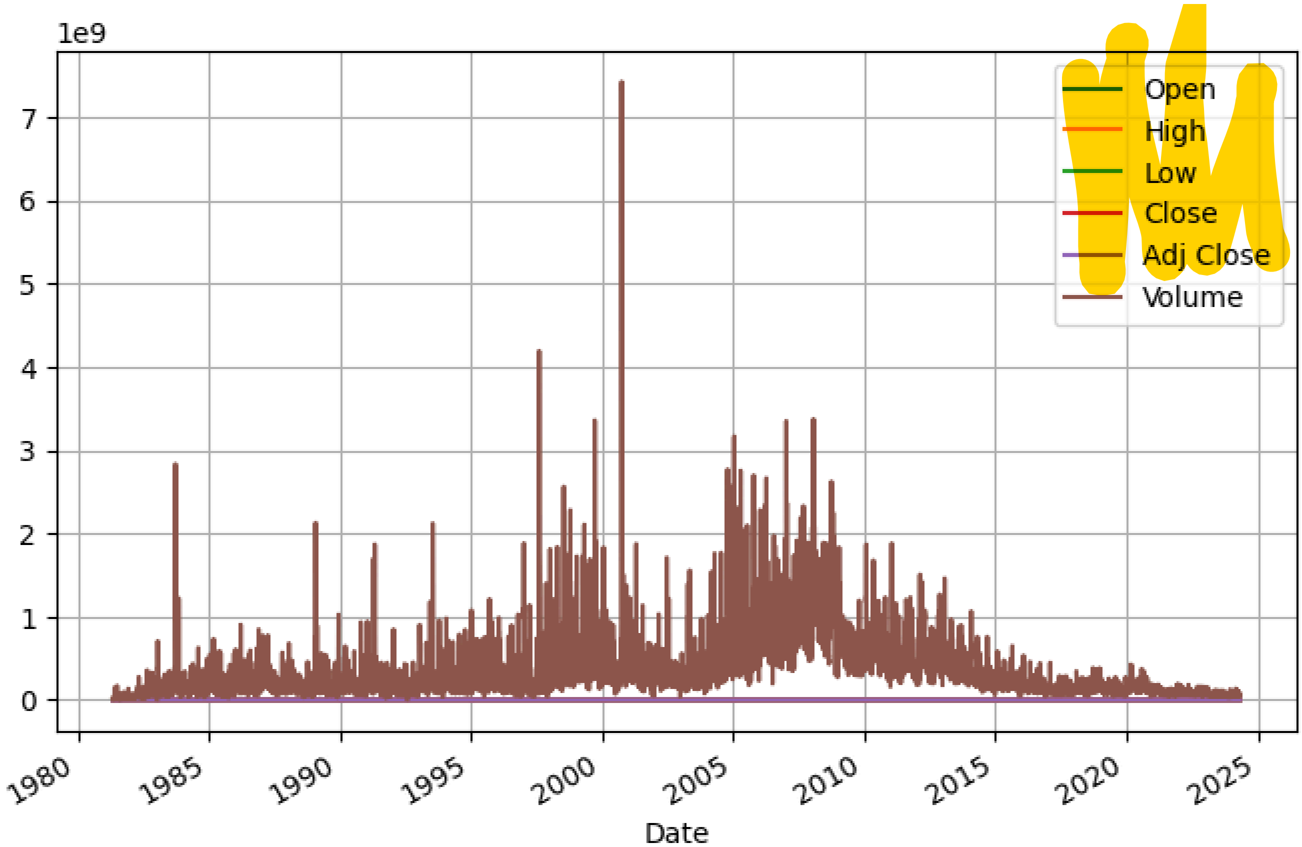
```
sp500_aapl
```

Date	Open	High	Low	Close	Adj Close	Volume
1981-05-01	0.126674	0.127790	0.126674	0.126674	0.097898	16553600
1981-05-04	0.126674	0.126674	0.126116	0.126116	0.097467	14448000
1981-05-05	0.126116	0.126116	0.125558	0.125558	0.097036	17539200
1981-05-06	0.122768	0.122768	0.122210	0.122210	0.094448	18950400
1981-05-07	0.123884	0.124442	0.123884	0.123884	0.095742	9363200
...	...	...	...	...	...	...
2024-04-23	165.350006	167.050003	164.919998	166.899994	166.899994	49537800
2024-04-24	166.539993	169.300003	166.210007	169.020004	169.020004	48251800
2024-04-25	169.529999	170.610001	168.149994	169.889999	169.889999	50558300
2024-04-26	169.880005	171.339996	169.179993	169.300003	169.300003	44525100
2024-04-29	173.369995	176.029999	173.100006	173.500000	173.500000	66162100

10839 rows × 6 columns

```
sp500_aapl.plot(grid=True,figsize=(8,5))
```

```
<Axes: xlabel='Date'>
```




```
sp500_aapl['Date'] = sp500_aapl.index

fig = px.area(
    sp500_aapl,
    x='Date',
    y='Volume',
    template='plotly_dark',
    color_discrete_sequence=['cyan'],
    title='AAPL index'
)

fig.show()
```

The logarithmic return is a way of calculating the rate of return on an investment. To calculate it you need the inital value of the investment V_i, the final value V_f and the number of time periods t. You then take the natural logarithm of V_f divided by V_i, and divide the result by t:

$$R = \frac{\ln(\frac{V_f}{V_i})}{t} \cdot 100\%$$

This value is normally expressed as a percentage, so you also multiply by 100.

The calculated rate will depend on the value of t that you use. If t is the number of years, then you get an annual rate. This then gives you the continuously compounded annual interest rate that you would need to receive in order to match the return on this investment.

Let p_t denote the price of an asset at time t. Then the return of an asset captures these relative movements and is defined as

$$r_t = \frac{p_t}{p_{t-1}} - 1$$

In words, a return is the change in price of an asset, relative to its previous value. Note that $p_t > 0$, and therefore $r_t > -1$.

In practice, "returns" often means "log returns". Log returns are defined as

$$z_t = \log(1 + r_t) = \log(\frac{p_t}{p_{t-1}}) = \log(p_t) - \log(p_{t-1})$$

Log returns are normally distributed if prices are log normally distributed.

$$z \sim N(0,1),\ x = e^{\mu + \sigma z}$$

We could write this as

$$\log(x) = \mu + \sigma z \sim \mu + N(0, \sigma^2) \sim N(\mu, \sigma^2)$$

So we see that the prices are normally distributed. Then

$$z_t = \log(p_t) - \log(p_{t-1})$$

is the sum of two normal random variables.

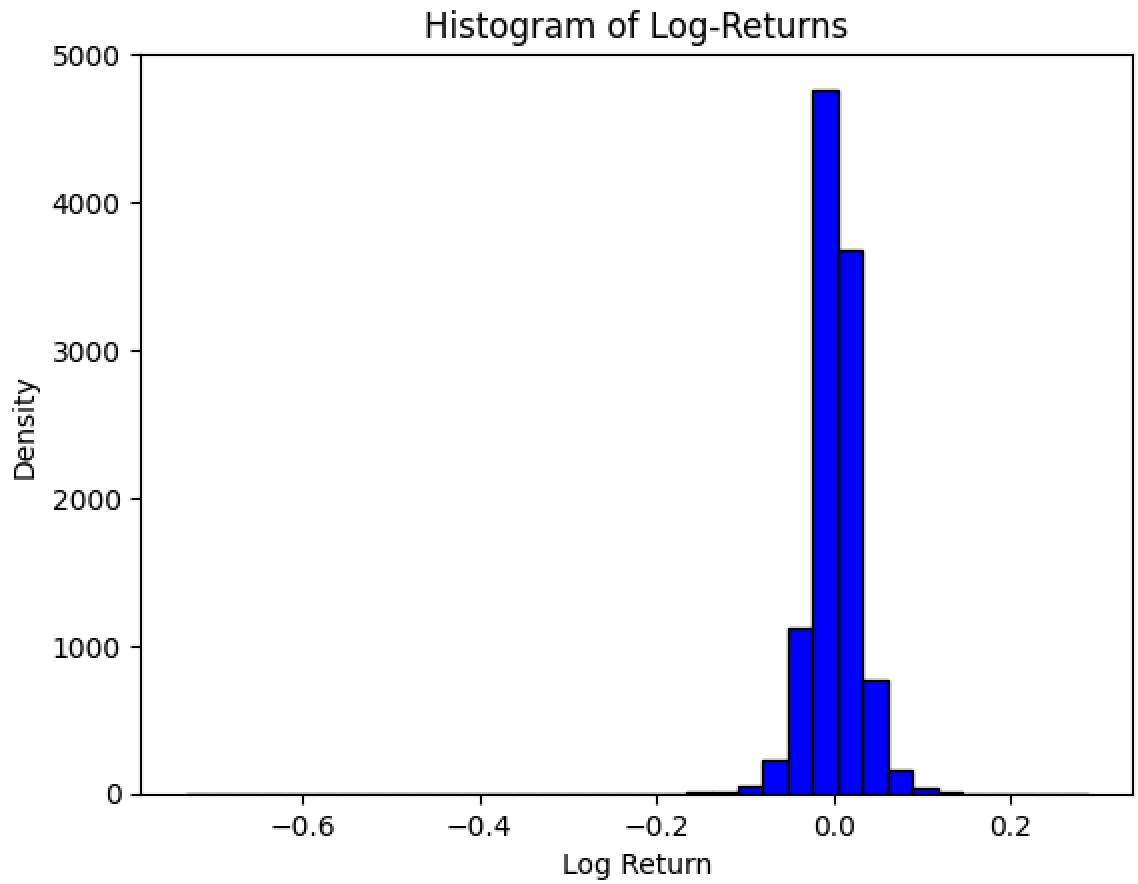
```
sp500_aapl['Log Return'] = np.log(sp500_aapl['Adj Close']/sp500_aapl['Adj Close'].shift(1))

print(sp500_aapl['Log Return'])
```

```
Date
1981-05-01         NaN
1981-05-04   -0.004415
1981-05-05   -0.004434
1981-05-06   -0.027027
1981-05-07    0.013604
                ...
2024-04-23    0.006371
2024-04-24    0.012622
2024-04-25    0.005134
2024-04-26   -0.003479
2024-04-29    0.024505
Name: Log Return, Length: 10839, dtype: float64
```

Histogram of log-returns

```
import seaborn as sns

plt.hist(sp500_aapl['Log Return'], color = 'blue', edgecolor = 'black',
         bins = int(180/5))

plt.title('Histogram of Log-Returns')
plt.xlabel('Log Return')
plt.ylabel('Density')
```

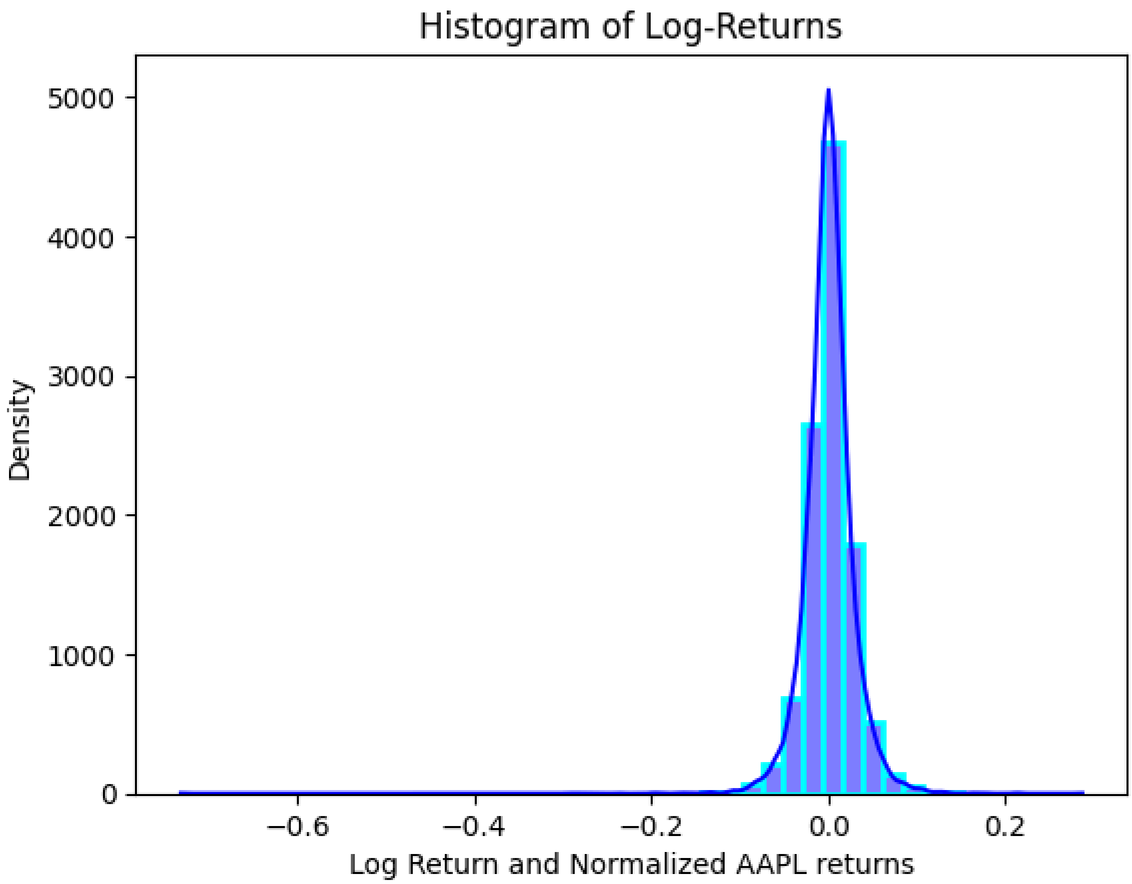
```
Text(0, 0.5, 'Density')
```




```
fig = px.histogram(
    sp500_aapl,
    x='Log Return',
    color_discrete_sequence=['cyan'],
    title='Histogram of Log-Returns',
    labels={'Log Return': 'Log Return', 'Density': 'Density'},
    template='plotly_dark'
)

fig.show()
```

We see that log-returns are normally distributed

```
sns.histplot(sp500_aapl['Log Return'], bins=int(180/4), color='blue',
             edgecolor='cyan', kde=True, linewidth=2)

# Set title and axis labels
plt.title('Histogram of Log-Returns')
plt.xlabel('Log Return and Normalized AAPL returns')
plt.ylabel('Density')

# Show the plot
plt.show()
```



Calculating Daily Average Returns

Daily Average Returns are given by computing the mean of the log rate of return series.

```
daily_log_returns = sp500_aapl['Log Return'].mean()
daily_log_returns
```

```
0.0006901645243884324
```

Calculating Annual Average Returns

Annual Average Returns are given by computing the mean of the log rate of return series and then multiplying the value by 250 since 250 days exist in a business/trading day system. (because in one calendar year we have 250 workings days in a years; some people say that there are 252 tradings days, but we will consider 250 days, because it will convinient for us)

```
annual_log_returns = sp500_aapl['Log Return'].mean() * 250
annual_log_returns
```

```
0.17254113109710809
```

```
print(str(round(annual_log_returns*100, 2)) + ' %')
```

```
17.25 %
```

```
sp500_aapl['42d'] = np.round(sp500_aapl['Close'].rolling(window=42).mean(),2)
sp500_aapl['252d'] = np.round(sp500_aapl['Close'].rolling(window=252).mean(),2)


sp500_aapl.tail
```

```
<bound method NDFrame.tail of                      Open        High         Low       Close   Adj
Close
Date
1981-05-01    0.126674    0.127790    0.126674    0.126674    0.097898  \
1981-05-04    0.126674    0.126674    0.126116    0.126116    0.097467
1981-05-05    0.126116    0.126116    0.125558    0.125558    0.097036
1981-05-06    0.122768    0.122768    0.122210    0.122210    0.094448
1981-05-07    0.123884    0.124442    0.123884    0.123884    0.095742
...                ...         ...         ...         ...         ...
2024-04-18  168.029999  168.639999  166.550003  167.039993  167.039993
2024-04-19  166.210007  166.399994  164.080002  165.000000  165.000000
2024-04-22  165.520004  167.259995  164.770004  165.839996  165.839996
2024-04-23  165.350006  167.050003  164.919998  166.899994  166.899994
2024-04-24  166.399994  169.289993  166.210007  168.574997  168.574997

              Volume       Date  Log Return     42d    252d
Date
1981-05-01  16553600 1981-05-01         NaN     NaN     NaN
1981-05-04  14448000 1981-05-04   -0.004414     NaN     NaN
1981-05-05  17539200 1981-05-05   -0.004435     NaN     NaN
1981-05-06  18950400 1981-05-06   -0.027027     NaN     NaN
1981-05-07   9363200 1981-05-07    0.013605     NaN     NaN
...              ...        ...         ...     ...     ...
2024-04-18  43122900 2024-04-18   -0.005731  173.62  181.18
2024-04-19  67772100 2024-04-19   -0.012288  173.23  181.17
2024-04-22  48116400 2024-04-22    0.005078  172.84  181.16
2024-04-23  48917700 2024-04-23    0.006371  172.42  181.17
2024-04-24  30881995 2024-04-24    0.009986  172.09  181.18

[10836 rows x 10 columns]>
```

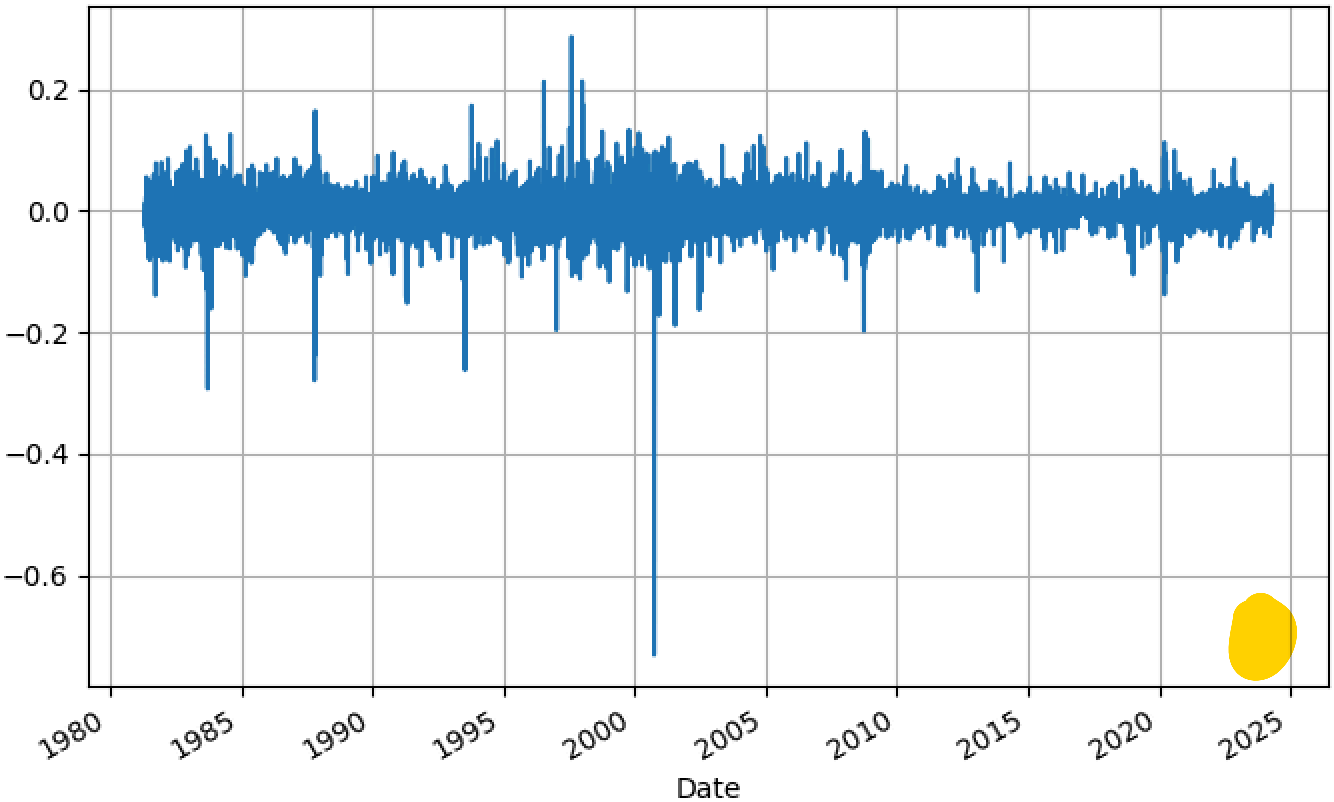
```
sp500_aapl[['Close','42d','252d']].plot(grid=True,figsize=(8,5))
```

```
<Axes: xlabel='Date'>
```


```python
sp500_aapl['Date'] = sp500_aapl.index

fig = px.line(
    sp500_aapl,
    x='Date',
    y=['Close', '42d', '252d'],
    template='plotly_dark',
    color_discrete_sequence=['cyan', 'orange', 'green'],
    title='Close Price with Moving Averages'
)

fig.show()
```

```python
sp500_aapl['Market Returns'] = np.log(sp500_aapl['Close'] / sp500_aapl['Close'].shift(1))
sp500_aapl['Market Returns'].plot(grid=True,figsize=(8,5))
```

```
<Axes: xlabel='Date'>
```




```
sp500_aapl['Market Returns'] = np.log(sp500_aapl['Close'] / sp500_aapl['Close'].shift(1))

fig = px.line(x=sp500_aapl.index, y=sp500_aapl['Market Returns'], template='plotly_dark', color_discr

fig.show()
```

```
sp500_aapl.keys()
```

```
Index(['Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume', 'Date',
       'Log Return', '42d', '252d', 'Market Returns'],
      dtype='object')
```

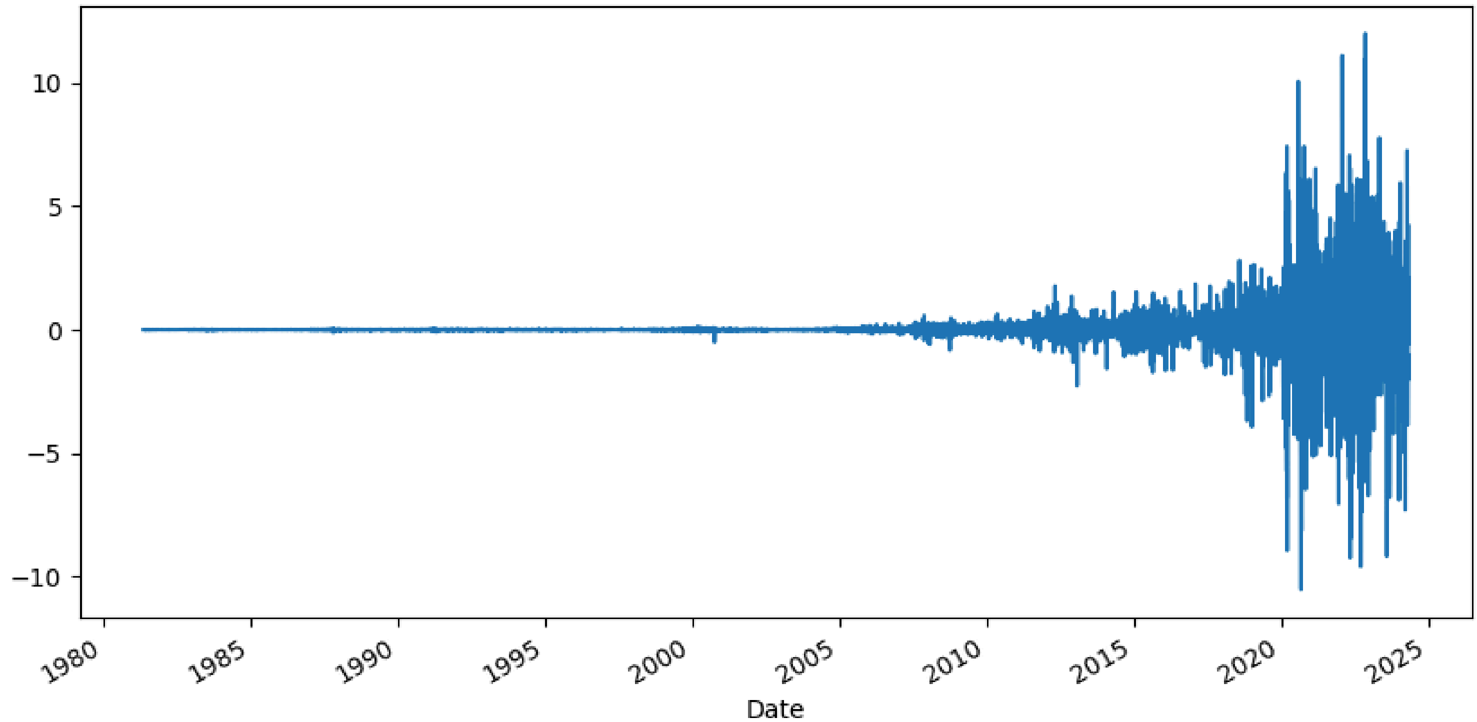
```
sp500_aapl['Date'] = sp500_aapl.index

fig = px.line(
    sp500_aapl,
    x='Date',
    y='Close',
    template='plotly_dark',
    color_discrete_sequence=['cyan', 'orange', 'green'],
    title='Close Price with Moving Averages'
)

fig.show()
```

The first obvious thing to note, aside from the two giant dips at the tail end corresponding to the market crashes in 2020 and 2023, is that the data is clearly non-stationary.

Below we will try to calculate the first difference of the series, i.e we will subtract the previous value t-1 from the current value t to get the difference.

```
sp500_aapl['First Difference'] = sp500_aapl['Close'] - sp500_aapl['Close'].shift()
sp500_aapl['First Difference'].plot(figsize=(10, 5))
```

```
<Axes: xlabel='Date'>
```




```
fig = px.area(
    sp500_aapl,
    x='Date',
    y='First Difference',
    template='plotly_dark',
    color_discrete_sequence=['cyan'],
    title='First Difference'
)

fig.show()
```

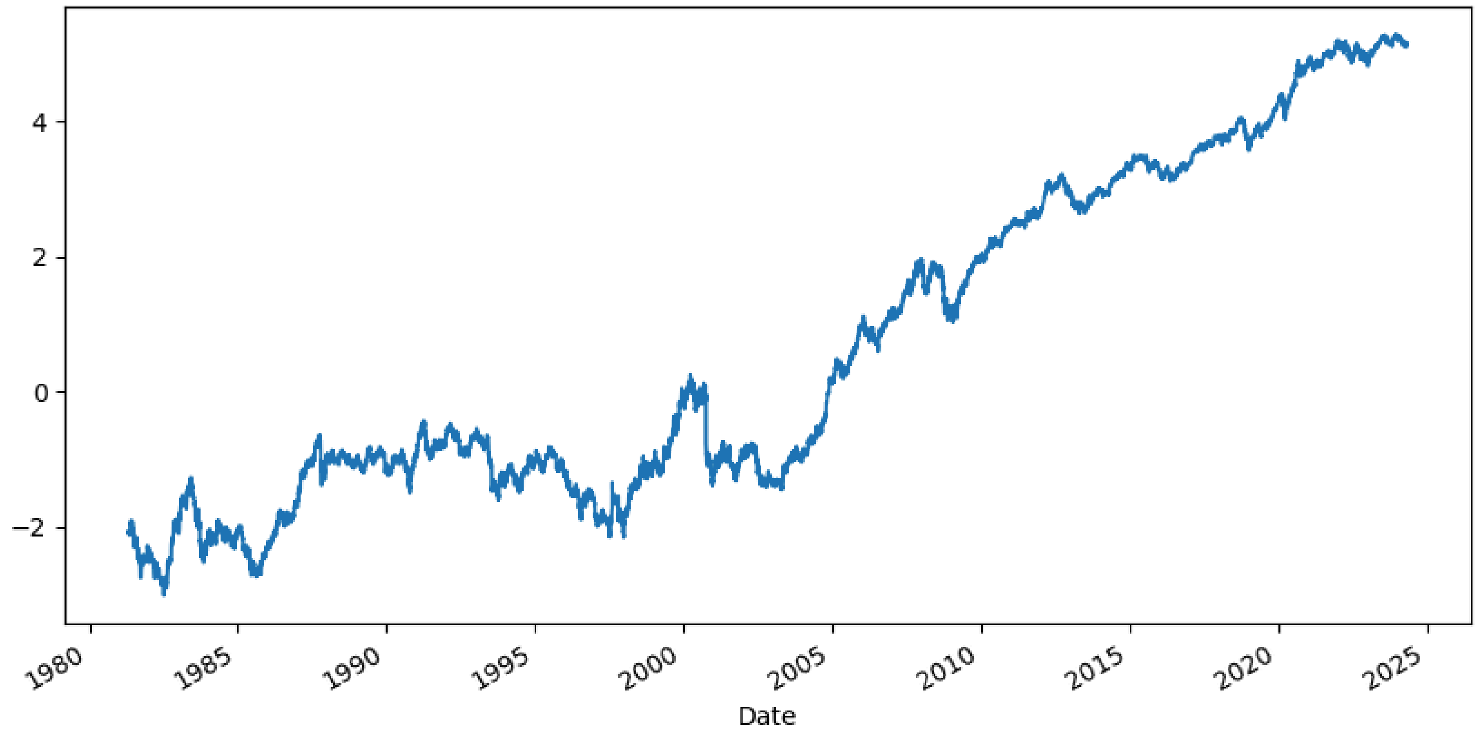
The data no longer appears to be trending up over time and is instead centered around 0, but we have another problem.

If we pay attention to the variance., we will see that it's very small early on and steadily increases over time.

This is a sign that the data is not only non-stationary but also exponentially increasing. Now we'll apply a log transform to the original series.

```
sp500_aapl['Natural Log'] = sp500_aapl['Close'].apply(lambda x: np.log(x))
sp500_aapl['Natural Log'].plot(figsize=(10, 5))
```

```
<Axes: xlabel='Date'>
```




```
fig = px.area(
    sp500_aapl,
    x='Date',
    y='Natural Log',
    template='plotly_dark',
    color_discrete_sequence=['cyan'],
    title='Natural Log'
)

fig.show()
```

```
sp500_aapl['Original Variance'] = pd.rolling_var(sp500_aapl['Close'], 30, min_periods=None, freq=None
sp500_aapl['Log Variance'] = pd.rolling_var(sp500_aapl['Natural Log'], 30, min_periods=None, freq=Non

fig, ax = plt.subplots(2, 1, figsize=(13, 12))
sp500_aapl['Original Variance'].plot(ax=ax[0], title='Original Variance')
sp500_aapl['Log Variance'].plot(ax=ax[1], title='Log Variance')
fig.tight_layout()
```

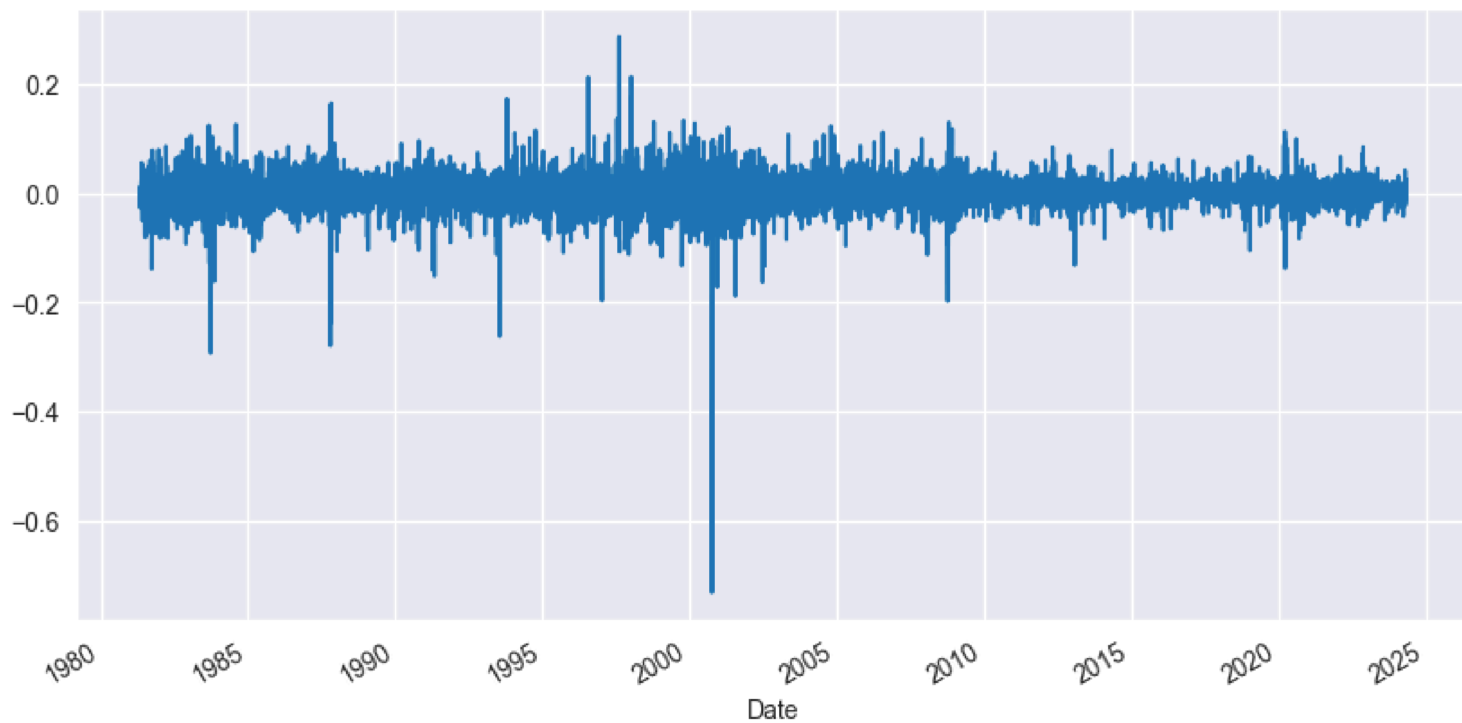
```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[139], line 1
----> 1 sp500_aapl['Original Variance'] = pd.rolling_var(sp500_aapl['Close'], 30,
min_periods=None, freq=None, center=True, how = None)
      2 sp500_aapl['Log Variance'] = pd.rolling_var(sp500_aapl['Natural Log'], 30,
min_periods=None, freq=None, center=True, how = None)
      4 fig, ax = plt.subplots(2, 1, figsize=(13, 12))

AttributeError: module 'pandas' has no attribute 'rolling_var'
```

Чтобы изменить содержимое ячейки, дважды нажмите на нее (или выберите "Ввод")

Now we'll calculate the first difference from the logged series, i.e we'll calculate log-returns

```
sp500_aapl['Logged First Difference'] = sp500_aapl['Natural Log'] - sp500_aapl['Natural Log'].shift()
sp500_aapl['Logged First Difference'].plot(figsize=(10, 5))
```

```
<Axes: xlabel='Date'>
```



A stationary sequence is a random sequence whose joint probability distribution is invariant over time. If a random sequence X j is stationary then the following holds:

$$F_{X_0,X_1,\dots,X_m}(x_0,x_1,\dots,x_m)=F_{X_k,X_{k+1},\dots,X_{k+m}}(x_0,x_1,\dots,x_m),$$

where F is the joint cumulative distribution function of the random variables in the subscript, i.e

$$(X_0,X_1,\dots,X_m)\overset{d}{\sim}(X_k,X_{k+1},\dots,X_{k+m})$$

If a sequence is stationary then it has a constant mean (which may not be finite): $\mathrm{E}(X[n])=\mu$ for all n

$\boxed{\text{\newline}}$ $\boxed{\text{\newline}}$

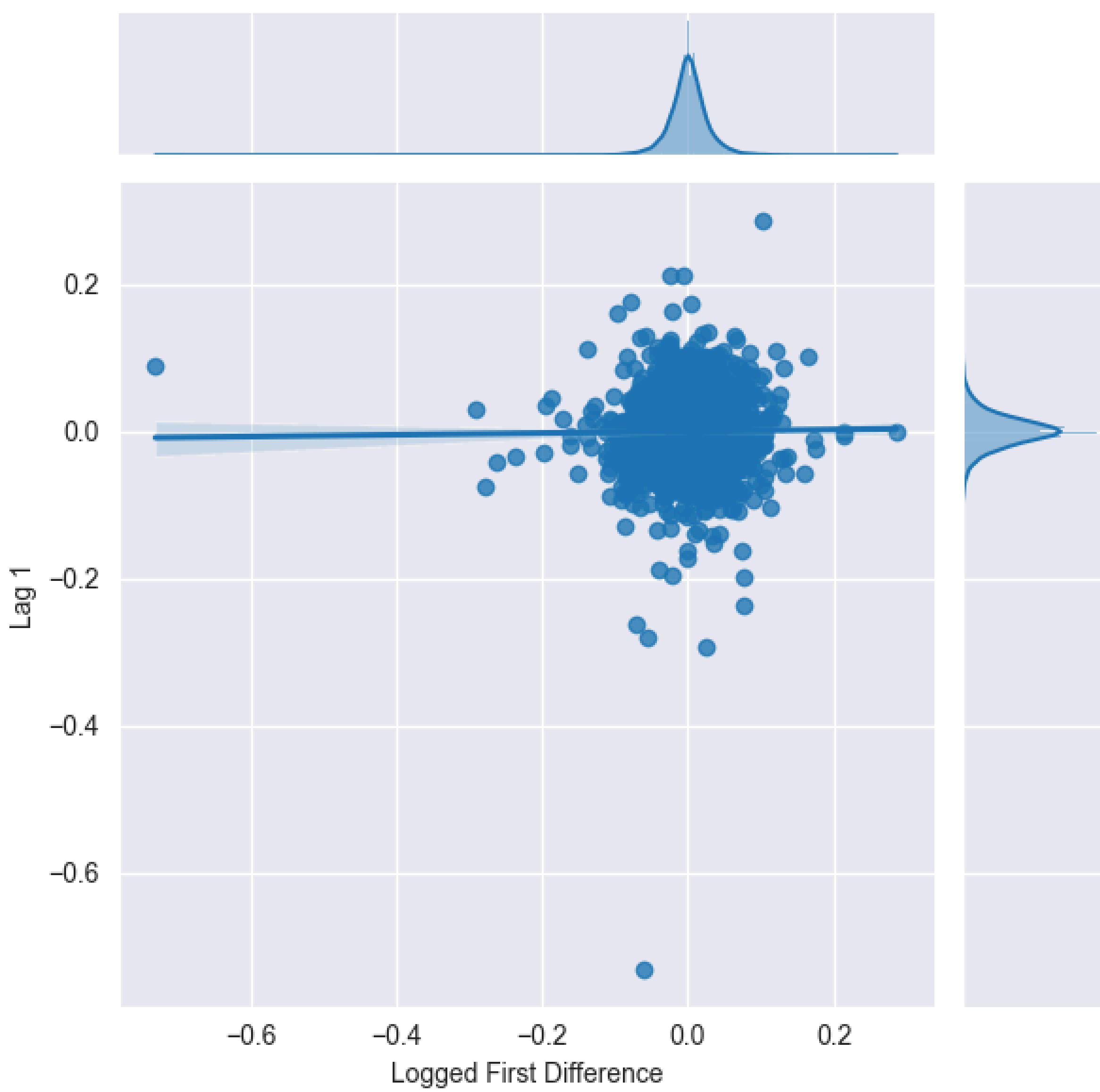
Now we have a stationary time series model of daily changes to the S&P 500 index Apple's company (this is just an assumed hypothesis, where next we'll try to show it).

Below we'll create some lag variables y(t-1), y(t-2) etc. and examine their relationship to y(t) and look at 1 and 2-day lags along with weekly and monthly lags to look for "seasonal" effects.

```
sp500_aapl['Lag 1'] = sp500_aapl['Logged First Difference'].shift()
sp500_aapl['Lag 2'] = sp500_aapl['Logged First Difference'].shift(2)
sp500_aapl['Lag 5'] = sp500_aapl['Logged First Difference'].shift(5)
sp500_aapl['Lag 30'] = sp500_aapl['Logged First Difference'].shift(30)
```

One interesting visual way to evaluate the relationship between lagged variables is to do a scatter plot of the original variable vs. the lagged variable and see where the distribution lies with a joint plot using the seaborn package.

```
sns.jointplot(data=sp500_aapl, x='Logged First Difference', y='Lag 1', palette='Set2', kind = 'reg')
plt.show()
```



Notice how tightly packed the mass is around 0, besides it also appears to be pretty evenly distributed - the marginal distributions on both axes are roughly normal.

This seems to indicate that knowing the index value one day doesn't tell us much about what it will do the next day.

```
from statsmodels.tsa.stattools import acf
from statsmodels.tsa.stattools import pacf

lag_correlations = acf(sp500_aapl['Logged First Difference'].iloc[1:])
lag_partial_correlations = pacf(sp500_aapl['Logged First Difference'].iloc[1:])
```

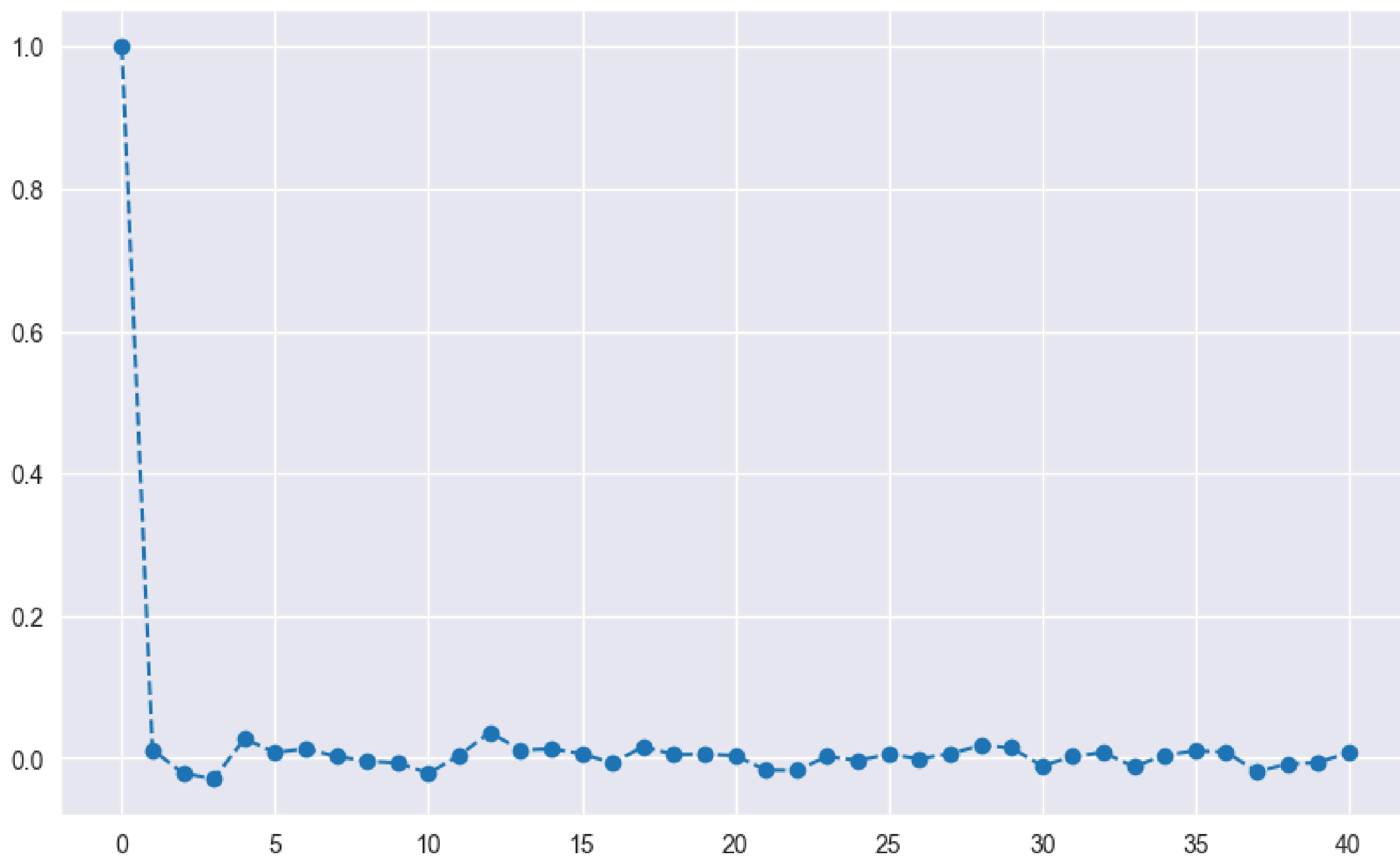
The auto-correlation function computes the correlation between a variable and itself at each lag step up to some limit (in this case 40).

The partial auto-correlation function computes the correlation at each lag step that is NOT already explained by previous, lower-order lag steps.

We can plot the results to see if there are any significant correlations.

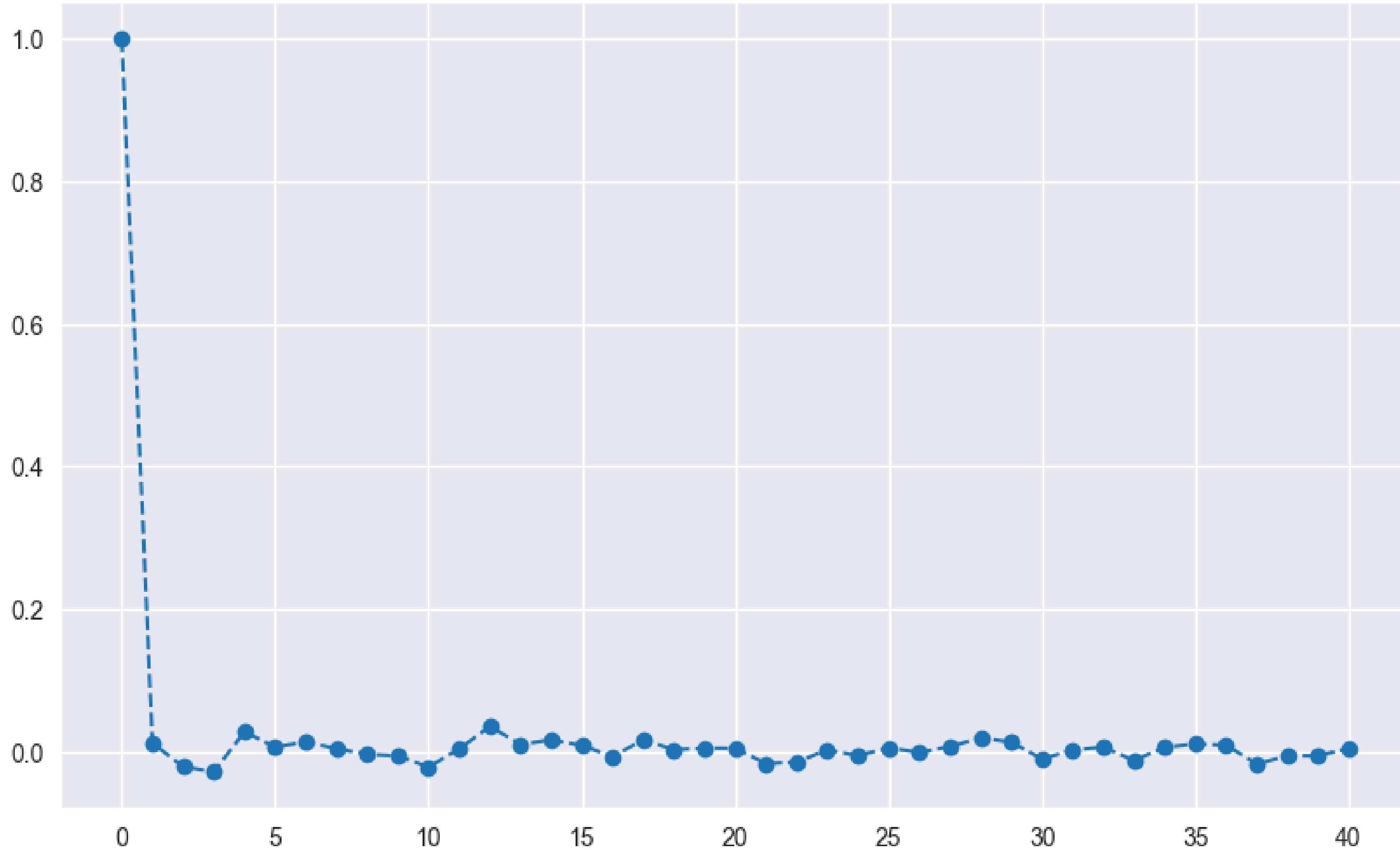
```
fig, ax = plt.subplots(figsize=(10, 6))
ax.plot(lag_correlations, marker='o', linestyle='--')
```

```
[<matplotlib.lines.Line2D at 0x1fe101d7dd0>]
```



```
fig, ax = plt.subplots(figsize=(10, 6))
ax.plot(lag_partial_correlations, marker='o', linestyle='--')
```

```
[<matplotlib.lines.Line2D at 0x1fe1012d450>]
```



The autocorrelation and partial-autocorrelation results are very close to each other (the first upper plot is the autocorrelation results and the second lower plot is the partial-autocorrelation results).

What this shows is that there is no significant correlation between the value at time t and at any time prior to t up to 40 steps behind. (the correlation coefficient of 0.2 before excluding outliers is considered as negligible correlation while 0.3 after excluding outliers may be interpreted as weak positive correlation).

In order words, the log-returns series is a random walk.

!!! Now I have little issues because I needed to show that log-returns are/aren't stationary time series, but I showed that log-returns series is a random walk !!!

Is there any statement or proof that we'll show that any random walk is non-stationary / random walk is stationary time series?

In the world website web I've read one fact that a random walk is a non-stationary, but I have no clue how to prove it mathematically correctly.

I saw the proof of this fact in this link: https://spureconomics.com/random-walk-model-and-stationarity/ , but I am not sure about it.

If it's true, i.e a log-returns which is a random walk cannot be a stationary time series.

```
%matplotlib inline
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
import seaborn as sb
sb.set_style('darkgrid')

path = os.getcwd() + '\data\stock_data.csv'
stock_data = pd.read_csv(path)
stock_data['Date'] = stock_data['Date'].convert_objects(convert_dates='coerce')
stock_data = stock_data.sort_index(by='Date')
stock_data = stock_data.set_index('Date')
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
Cell In[96], line 11
      8 sb.set_style('darkgrid')
     10 path = os.getcwd() + '\data\stock_data.csv'
---> 11 stock_data = pd.read_csv(path)
     12 stock_data['Date'] = stock_data['Date'].convert_objects(convert_dates='coerce')
     13 stock_data = stock_data.sort_index(by='Date')

File
~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-
packages\Python311\site-packages\pandas\io\parsers\readers.py:912, in
read_csv(filepath_or_buffer, sep, delimiter, header, names, index_col, usecols, dtype,
engine, converters, true_values, false_values, skipinitialspace, skiprows, skipfooter, nrows,
na_values, keep_default_na, na_filter, verbose, skip_blank_lines, parse_dates,
infer_datetime_format, keep_date_col, date_parser, date_format, dayfirst, cache_dates,
iterator, chunksize, compression, thousands, decimal, lineterminator, quotechar, quoting,
doublequote, escapechar, comment, encoding, encoding_errors, dialect, on_bad_lines,
delim_whitespace, low_memory, memory_map, float_precision, storage_options, dtype_backend)
    899 kwds_defaults = _refine_defaults_read(
    900     dialect,
    901     delimiter,
   (...)
    908     dtype_backend=dtype_backend,
    909 )
    910 kwds.update(kwds_defaults)
--> 912 return _read(filepath_or_buffer, kwds)
```