1. TIME SERIES ANALYSIS

1.1 Acquisition of The Stock Prices and Data Preparation

The stock prices data is acquired from Yahoo Finance and in this example, the daily stock prices dataset belongs to Advanced Micro Devices Inc. which is in S&P 500.

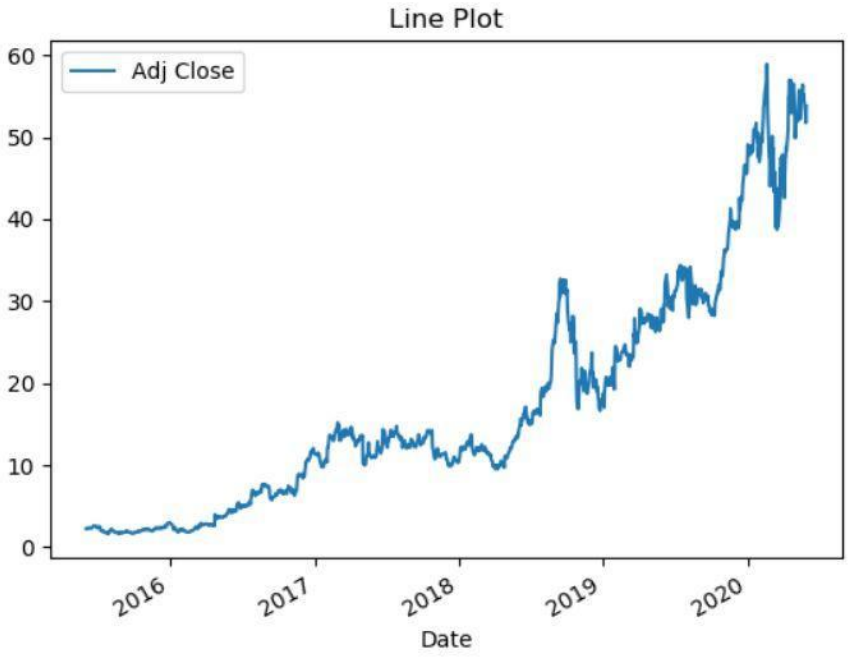
1.2 Analysis of The Dataset

Calculating descriptive statistics on the dataset to have an insight about it.

```
DESCRIPTION OF THE TIME SERIES DATA:
              High          Low         Open        Close       Volume    Adj Close
count  1259.000000  1259.000000  1259.000000  1259.000000  1.259000e+03  1259.000000
mean     17.668372    16.897824    17.283789    17.293193  5.671107e+07    17.293193
std      14.398599    13.790750    14.095815    14.107672  4.093656e+07    14.107672
min       1.690000     1.610000     1.620000     1.620000  2.606600e+06     1.620000
25%       6.740000     6.460000     6.585000     6.595000  2.923250e+07     6.595000
50%      13.130000    12.600000    12.850000    12.860000  4.887130e+07    12.860000
75%      27.490000    26.160000    26.820001    26.780000  7.551750e+07    26.780000
max      59.270000    57.509998    58.439999    58.900002  3.250584e+08    58.900002
```

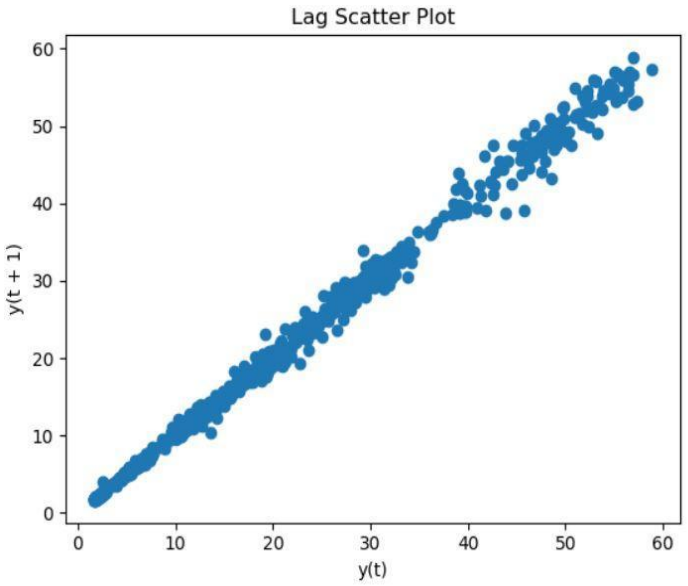
```
# a description of dataset
print('DESCRIPTION OF THE TIME SERIES DATA:')
print(series.describe())
```

Furthermore, visualizing the raw dataset to have a better understanding. Columns, namely, High, Low, Open, Close, and Adjusted Close have shown exactly the same results, hence, in the following graphs only Adjusted Close is used for demonstration of the time series data for the sake of visual clarity.



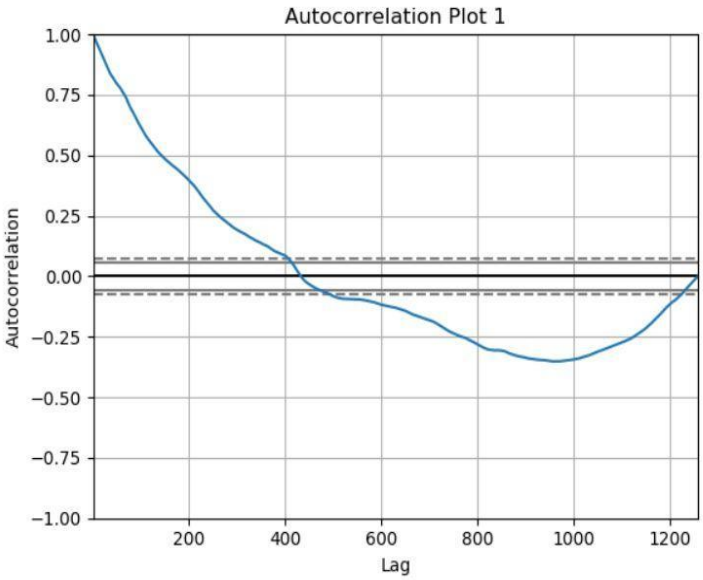

A line Plot is a great way the quickly have a grasp of the dataset. The prices through the years are easily observable.

```
# create a line plot
print('#LINE PLOT')
from matplotlib import pyplot
series.plot()
pyplot.title('Line Plot')
pyplot.show()
```



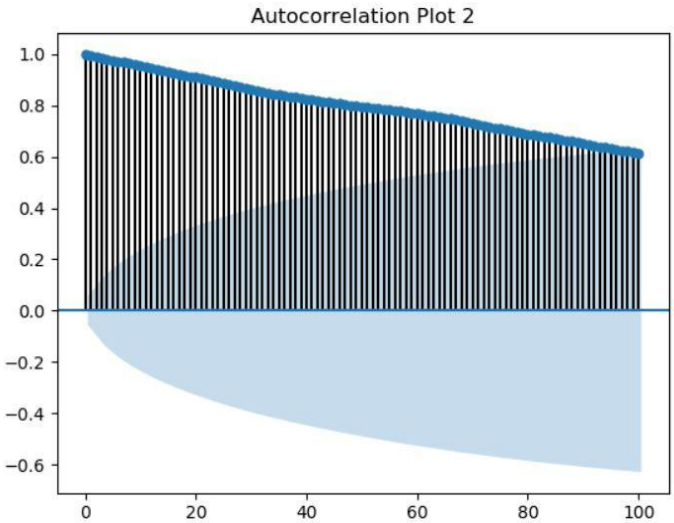

The interpretation of the Lag Scatter Plot is that there is a positive correlation relationship between an observation from the dataset and the previous observation. However, the spread of the point along the diagonal also shows us that the relationship is weak.

```
# create a scatter plot
from pandas.plotting import lag_plot
print('#SCATTER PLOT')
lag_plot(series)
pyplot.title('Lag Scatter Plot')
pyplot.show()
```



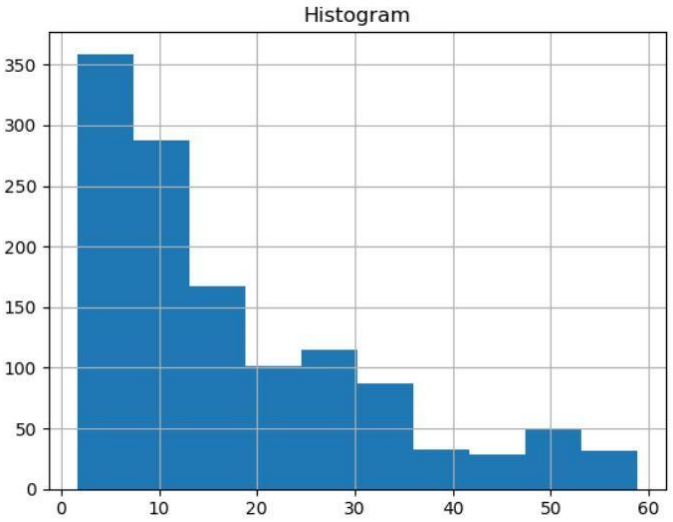

After the recognition of the correlation with the help of the Lag Scatter Plot, now we can also examine and discover a self-correlation with the help of the Autocorrelation Plots. The result of this tells us that there is no seasonality in this dataset which we will be investigating more.

```
# create an autocorrelation plot
print('#AUTOCORRELATION PLOT')
from pandas.plotting import autocorrelation_plot
autocorrelation_plot(series)
pyplot.title('Autocorrelation Plot 1')
pyplot.show()
```



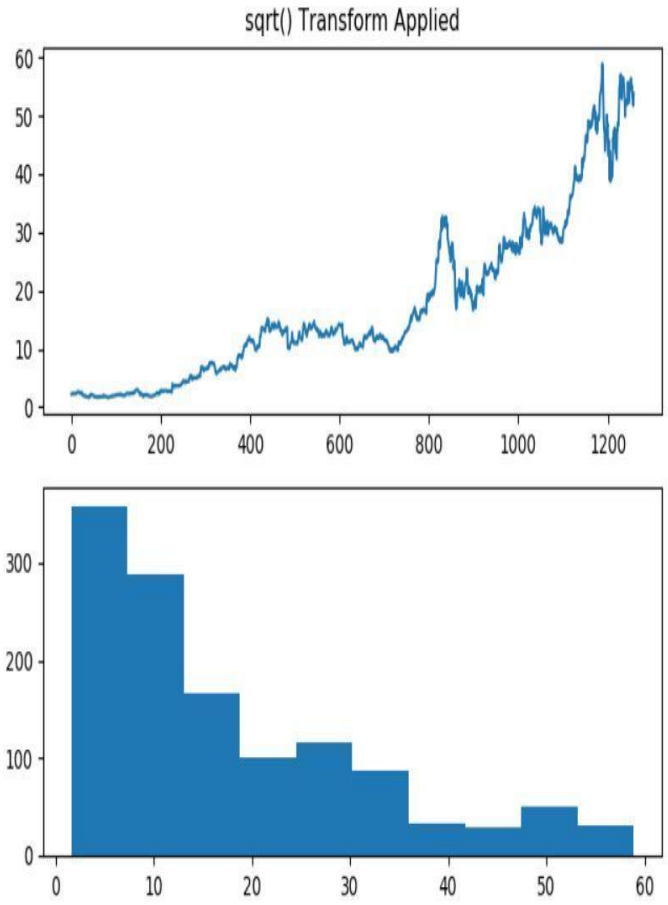

Another autocorrelation plot shows us the positive correlation (+1) and the lack of seasonality in the dataset since the plotted dots are out of the confidence interval of the seasonality (grey area).

```
# The ACF of trended time series tend to have positive
values that slowly decrease as the lags increase.
from statsmodels.graphics.tsaplots import plot_acf
plot_acf(series, lags=100)
pyplot.title('Autocorrelation Plot 2')
pyplot.show()
```



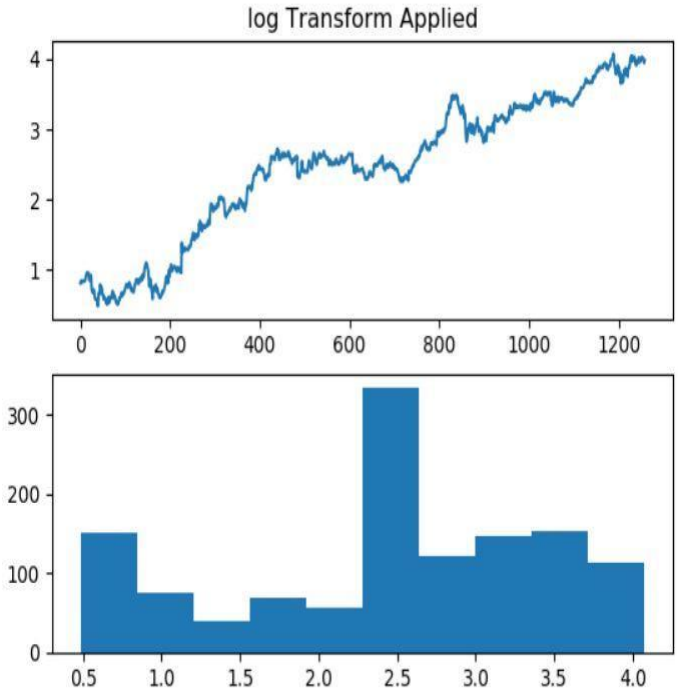

The histogram of the dataset indicates exponential growth. This fortifies our intuition about natural behavior the stock prices. It is reasonable to expect exponentiality at stock prices because of the snowball effect. This kind of behavior was also clearly observable in cryptocurrency prices.

```
# create a histogram plot
print('#HISTOGRAM')
series.hist()
pyplot.title('Histogram')
pyplot.show()
```




Applying square root transform to reduce the growing trend to be linear and change the distribution of the observations to be nearly Gaussian had less effect than we expected. This suggests us a strong exponential behavior as we assumed or long-tail distribution. Hence, we continued with Log Transform.

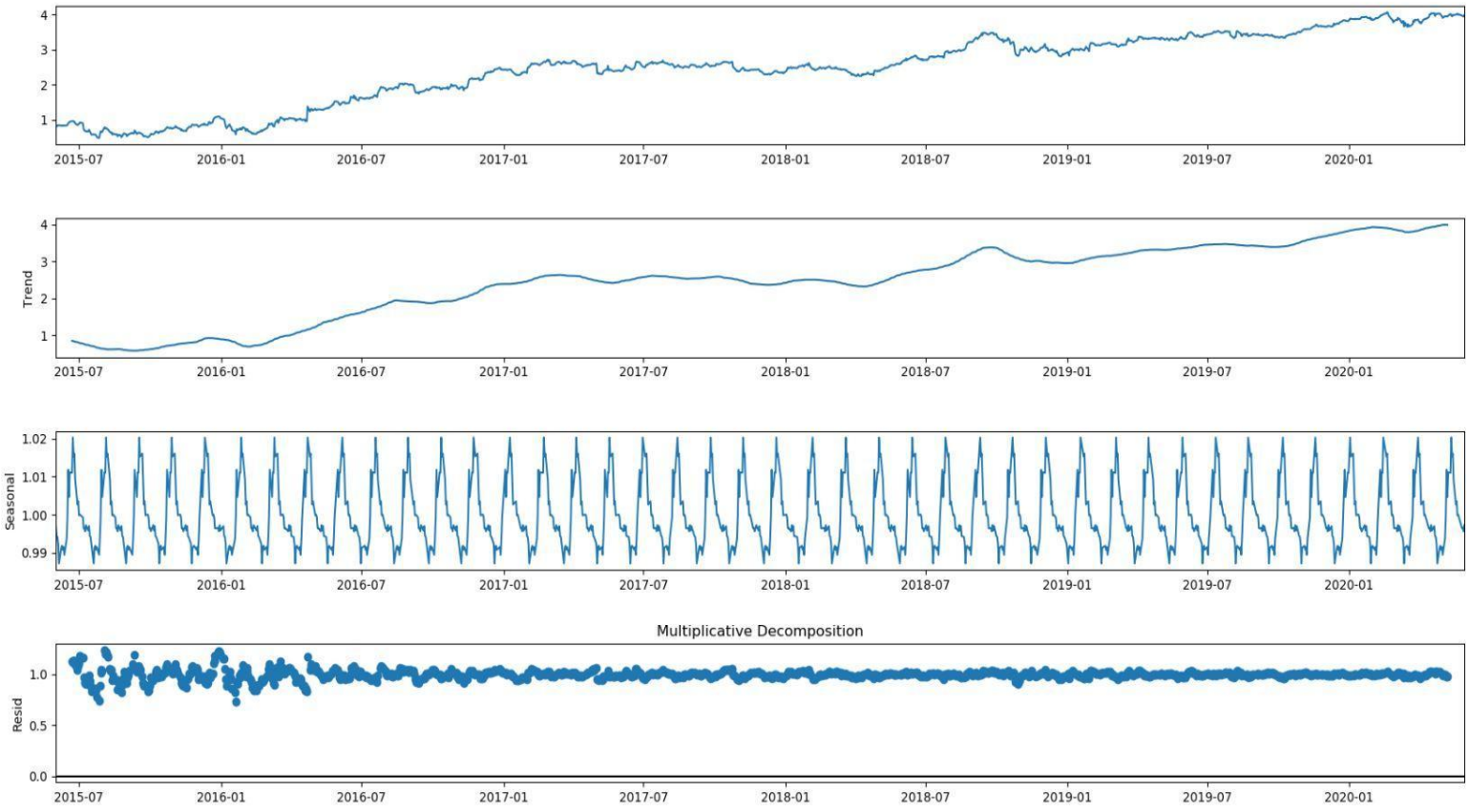
```
# square root transform a time series
from pandas import DataFrame
from numpy import sqrt
from matplotlib import pyplot
dataframe = DataFrame(series.values)
dataframe.columns = ['Adj Close']
dataframe['Adj Close'] = sqrt(dataframe['Adj Close'])
pyplot.figure(1)
# line plot
pyplot.subplot(211)
pyplot.title('sqrt() Transform Applied')
pyplot.plot(dataframe['Adj Close'])
# histogram
pyplot.subplot(212)
pyplot.hist(dataframe['Adj Close'])
pyplot.show()
```




Applying the log transform yielded a better result in getting a more linear trend. Also, the histogram shows more uniformity or Gaussian-like shape which supports the idea.

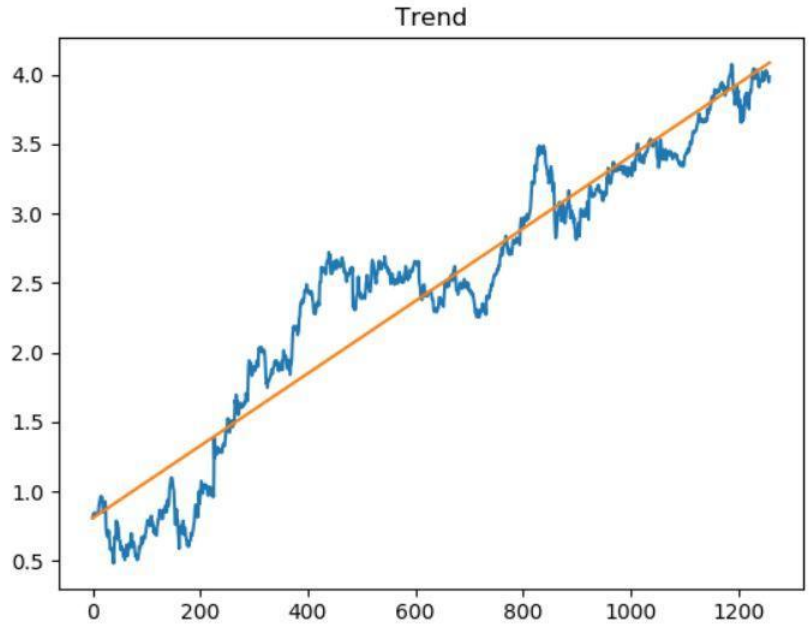
```
# log transform a time series
from numpy import log
dataframe = DataFrame(series.values)
dataframe.columns = ['Adj Close']
dataframe['Adj Close'] = log(dataframe['Adj Close'])
pyplot.figure(1)
# line plot
pyplot.subplot(211)
pyplot.title('log Transform Applied')
pyplot.plot(dataframe['Adj Close'])
# histogram
pyplot.subplot(212)
pyplot.hist(dataframe['Adj Close'])
pyplot.show()
```

Exploratory Analysis

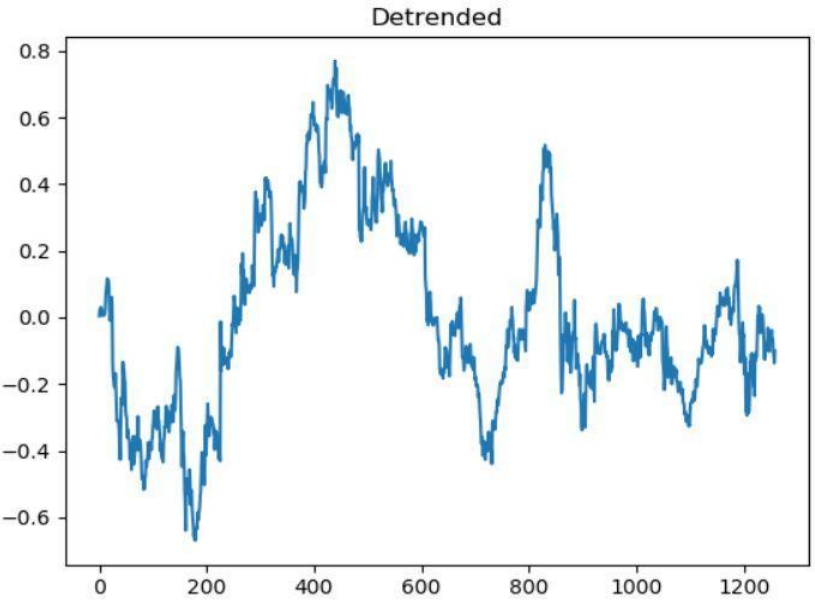



After analyzing the dataset, we have now decomposed the time series data into trend, seasonality, and noise components. As we have introduced the exponential behavior in our dataset, our model is multiplicative. Hence, we applied multiplicative decomposition. Here, we used in-built functions to perform the decomposition and so far, the information we get from this decomposition and our previous investigations coincide.

```
# EXPLORATORY ANALYSIS
print('#MULTIPLICATIVE DECOMPOSITION')
from statsmodels.tsa.seasonal import seasonal_decompose
result = seasonal_decompose(series, model='multiplicative', period = 30)
print(result.trend)
print(result.seasonal)
print(result.resid) # the time series after the trend, and seasonal components are removed
print(result.observed)
result.plot()
pyplot.title('Multiplicative Decomposition')
pyplot.show()
```




So far, all our observations on the dataset have left us to realize the trend is the only obstacle between stationary data and non-stationary data for this dataset. Hence, showing the trend here clearly, we move on to removing it.




The end product of our time series dataset is now ready to split into train and test parts. For detrending, we implemented the model fitting instead of detrending by differencing because even fitting a linear model to a trend that is clearly super-linear or exponential is favorable.

```
# Trends
print('#TRENDS')
from pandas import datetime
from sklearn.linear_model import LinearRegression
import numpy
# fit linear model
X = [i for i in range(0, len(series))]
X = numpy.reshape(X, (len(X), 1))
y = series.values
model = LinearRegression()
model.fit(X, y)
# calculate trend
trend = model.predict(X)
# plot trend
pyplot.plot(y)
pyplot.plot(trend)
pyplot.title('Trend')
pyplot.show()
# detrend
detrended = [y[i]-trend[i] for i in range(0, len(series))]
# plot detrended
pyplot.plot(detrended)
pyplot.title('Detrended')
pyplot.show()
```