HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

School of Information and Communications Technologyy

----- □ & □ -----

Statistical Applications To Economics, Modelling Of Economics And Financial Data

Subject:	Applied Statistics and Experimental Design
Supervisor:	Assoc. Prof. PhD. Nguyen Linh Giang
Authors:	Chu Trung Anh – 20225564
	Vu Duc Thang – 20225553
	Dao Minh Quang – 20225552
	Nguyen Sy Quan – 20225585

HANOI, 6/2024

Table of contents

ACKNOWLEDGMENT

The successful completion of this report would not have been possible without the invaluable contributions of several individuals and groups. I extend my heartfelt appreciation to my academic advisor, Assoc. Prof. PhD. Nguyen Linh Giang , for his expert guidance and constructive feedback throughout the project and writing process. Additionally, I am grateful for the support of my family and friends, whose encouragement was instrumental in maintaining my motivation.

CHAPTER 1. INTRODUCTION

1.1 Background

The stock market is a dynamic and complex system, heavily influenced by various economic, political, and social factors. Understanding stock price movements and predicting future trends is crucial for investors, financial analysts, and policymakers. The S&P 500 index, which includes 500 of the largest publicly traded companies in the U.S., serves as a key indicator of the overall health of the U.S. stock market and economy.

1.2 Problem Formulation

The objective of this report is to analyze the stock price movements of Apple Inc. (AAPL) using historical data. By leveraging statistical models and machine learning techniques, we aim to identify patterns, understand the factors driving stock prices, and develop predictive models to forecast future stock prices. This analysis can provide valuable insights for investors and help in making informed investment decisions.

1.3 Aims

1. To explore and preprocess the historical stock price data of Apple Inc.
2. To check the stationarity of the timeserieses data and transform it if necessary.
3. To decompose the timeserieses to understand its underlying components.
4. To build and evaluate predictive models: AR, MA, ARIMA and Prophet for forecasting future stock prices based on time series analysis.
5. To interpret the results and provide actionable insights for investors.

CHAPTER 2. DATASET

2.1 Dataset Description

Description

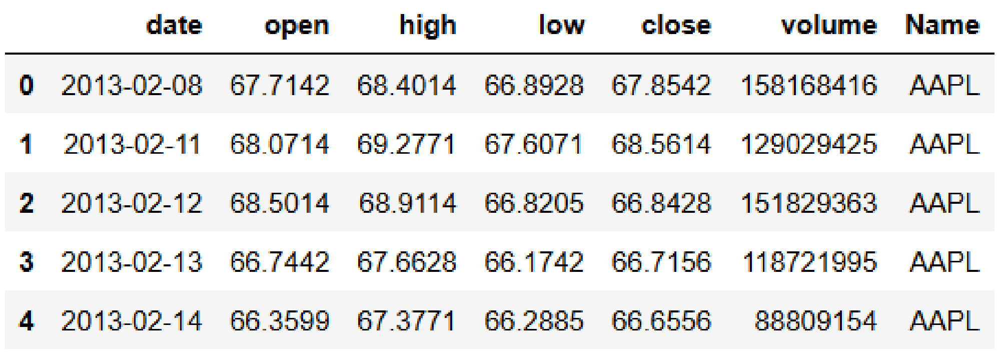
We are utilizing the S&P 500 stock data dataset from Kaggle for our analysis. This dataset, last updated in 2018, is a comprehensive collection of historical stock prices for all companies currently listed on the S&P 500 index. The data spans a period of 5 years, providing a rich source of information for our study.

The dataset is well-structured and can be divided to analyze each individual company separately. This is facilitated by the **individual_stocks_5yr** folder, which contains data files for individual stocks. Each file is labelled by the respective company's stock ticker name, making it easy to locate and analyze data for a specific company.

Each file in the dataset contains the following columns:

- **Date:** This column records the date of the trading day in the format: yy-mm-dd. It allows us to track the stock's performance over time.
- **Open:** This column records the price of the stock at market open. This data is from the NYSE, so all prices are in USD. It provides a starting point for the day's trading.
- **High:** This column records the highest price the stock reached during the trading day. It gives us an idea of the stock's potential for the day.
- **Low:** This column records the lowest price the stock reached during the trading day. It provides insight into the stock's risk for the day.
- **Close :** Closing price of the stock at the end of the trading day. Tt is a commonly used reference point for investors to assess the performance of a particular stock over time.
- **Volume:** This column records the number of shares traded during the trading day. It gives us an idea of the stock's liquidity and popularity.
- **Name:** This column records the ticker name of the stock. It allows us to identify the stock.

To provide a more focused context for this project, our team has chosen to analyze a specific company - Apple, one of the world's largest technology companies. Its stock is listed on the index as AAPL. The dataset we are using for this analysis is named **AAPL_data.csv**. Following is an overview visualization of the dataset:



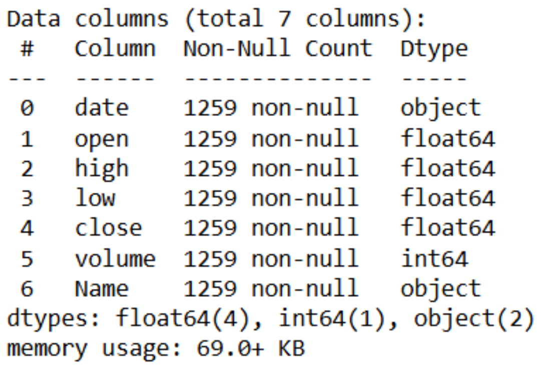
	date	open	high	low	close	volume	Name
0	2013-02-08	67.7142	68.4014	66.8928	67.8542	158168416	AAPL
1	2013-02-11	68.0714	69.2771	67.6071	68.5614	129029425	AAPL
2	2013-02-12	68.5014	68.9114	66.8205	66.8428	151829363	AAPL
3	2013-02-13	66.7442	67.6628	66.1742	66.7156	118721995	AAPL
4	2013-02-14	66.3599	67.3771	66.2885	66.6556	88809154	AAPL

In this project context, we are working with time series analysis, so we just need to use two columns: `date` and `close`.

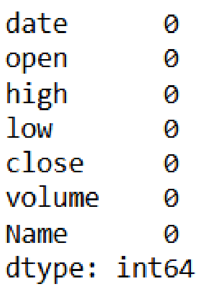
A noteworthy characteristic of this dataset is the data column. Notably, the date **08-02-2013** falls on a Friday, and the subsequent date in the dataset, **11-02-2013**, is a Monday. This suggests that the data is recorded only on weekdays.

Key Features

Some statistic about this dataset:

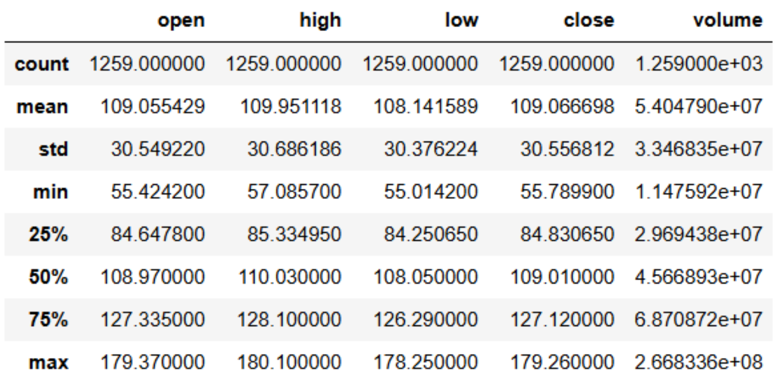


```
Data columns (total 7 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   date    1259 non-null   object
 1   open    1259 non-null   float64
 2   high    1259 non-null   float64
 3   low     1259 non-null   float64
 4   close   1259 non-null   float64
 5   volume  1259 non-null   int64
 6   Name    1259 non-null   object
dtypes: float64(4), int64(1), object(2)
memory usage: 69.0+ KB
```



```
date      0
open      0
high      0
low       0
close     0
volume    0
Name      0
dtype: int64
```

There is no null value in this dataset



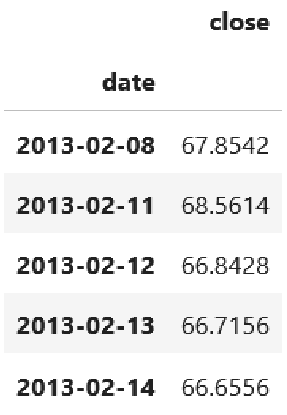
	open	high	low	close	volume
count	1259.000000	1259.000000	1259.000000	1259.000000	1.259000e+03
mean	109.055429	109.951118	108.141589	109.066698	5.404790e+07
std	30.549220	30.686186	30.376224	30.556812	3.346835e+07
min	55.424200	57.085700	55.014200	55.789900	1.147592e+07
25%	84.647800	85.334950	84.250650	84.830650	2.969438e+07
50%	108.970000	110.030000	108.050000	109.010000	4.566893e+07
75%	127.335000	128.100000	126.290000	127.120000	6.870872e+07
max	179.370000	180.100000	178.250000	179.260000	2.668336e+08

2.2 Preprocessing

2.2.1 Set the date column as index and set a fixed frequency

To perform timeserieses analysis, we first need to set the date column as the index so that Python can recognize and perform operations on it.

It is also essential to ensure that all time periods are equal and clearly defined, resulting in a constant frequency. As highlighted in the previous section, the data is recorded exclusively on weekdays. Therefore, we will set the frequency to 'business day', denoted as B, which includes Monday through Friday.



	close
date	
2013-02-08	67.8542
2013-02-11	68.5614
2013-02-12	66.8428
2013-02-13	66.7156
2013-02-14	66.6556

Initially, the dataset has no missing values. However, when we set the frequency, the records follow the frequency index, which may introduce missing values when resampling the dataset. As in our case, we observe 45 missing values, which is unexpected. As previously mentioned, the original dataset records dates only on weekdays. Therefore, when we set the frequency to 'B' for business days, we shouldn't encounter any missing values.

To investigate this, we display these missing values and observe that none of them fall on weekends. This suggests they might correspond to holidays or other non-trading days not included in the original dataset. Initially, these dates were not present in the dataset.

```
Date: 2013-02-18 00:00:00, Day of Week: Monday
Date: 2013-03-29 00:00:00, Day of Week: Friday
Date: 2013-05-27 00:00:00, Day of Week: Monday
Date: 2013-07-04 00:00:00, Day of Week: Thursday
Date: 2013-09-02 00:00:00, Day of Week: Monday
Date: 2013-11-28 00:00:00, Day of Week: Thursday
Date: 2013-12-25 00:00:00, Day of Week: Wednesday
Date: 2014-01-01 00:00:00, Day of Week: Wednesday
Date: 2014-01-20 00:00:00, Day of Week: Monday
```

To handle these missing values, we choose Front Filling, which assign the value of the previous period, e.g., the day 8/1 is missing then it will be assigned the value of the day 7/1.

2.2.2 Normalizing

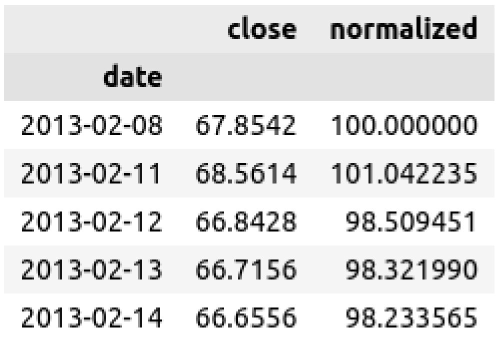
In time series analysis, normalization is a crucial preprocessing step that adjusts values to a common scale. Typically, each value in the series is transformed into a ratio relative to the first value. For example, normalizing the series [100, 105, 110, 120] by the first value results in [1, 1.05, 1.1, 1.2], where each value represents the proportion relative to the first value.

Normalization makes it easier to compare different time series with each other, especially when the original series have different scales or units. By removing the effect of the scale, we can focus on underlying patterns and trends.

In practical terms, normalization is particularly useful in financial analysis. By normalizing time series of different assets, we can compare their performance relative to each other. For instance, if Apple (AAPL) increases by $1000 (10%) and Samsung increases by $100 (80%), the relative increase is much larger for Samsung. Despite the smaller absolute increase, Samsung's return is higher in percentage terms.

This is why we normalize time series data: it allows us to compare relative changes across different time series, leading to more informed investment decisions based on relative performance rather than absolute price levels.

Here is the data set after normalizing:



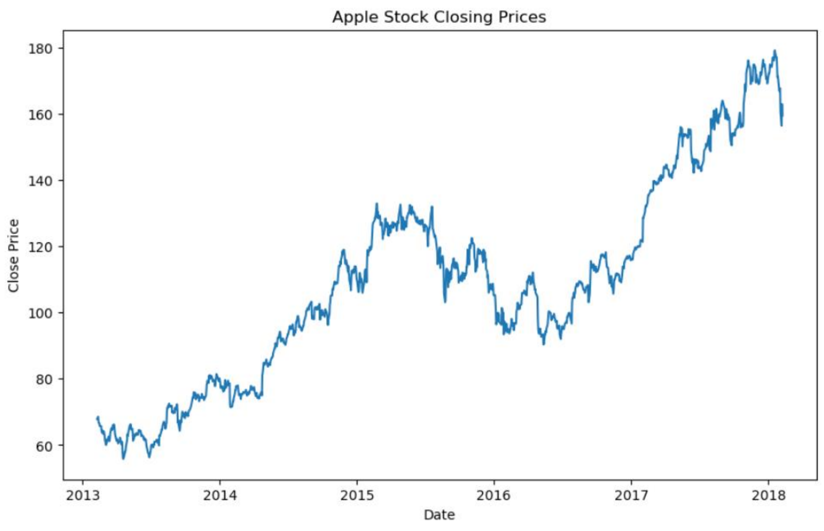
	close	normalized
date		
2013-02-08	67.8542	100.000000
2013-02-11	68.5614	101.042235
2013-02-12	66.8428	98.509451
2013-02-13	66.7156	98.321990
2013-02-14	66.6556	98.233565

Note that normalization does not affect the stationary or the model performance.

2.2.3 Stationary check

In order to use AR, MA, ARMA models, we first had to make sure that out timeseries is stationary (*theory about stationary and how to check for that will be discussed more detail in Part 3 Theory*).

First let have a look at our timeseries:



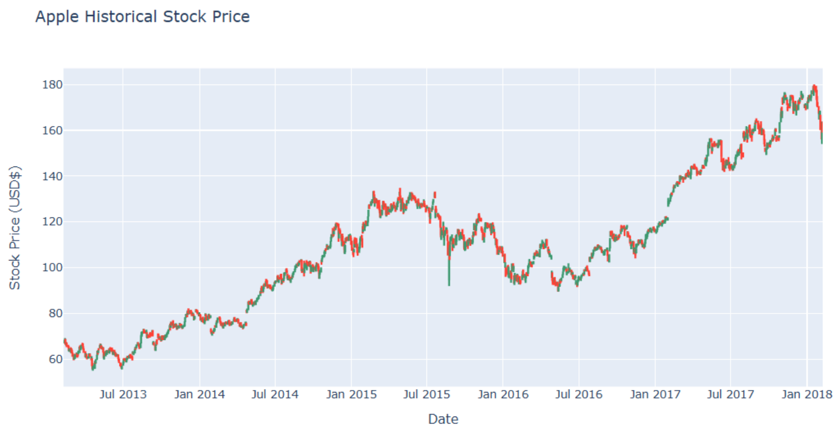
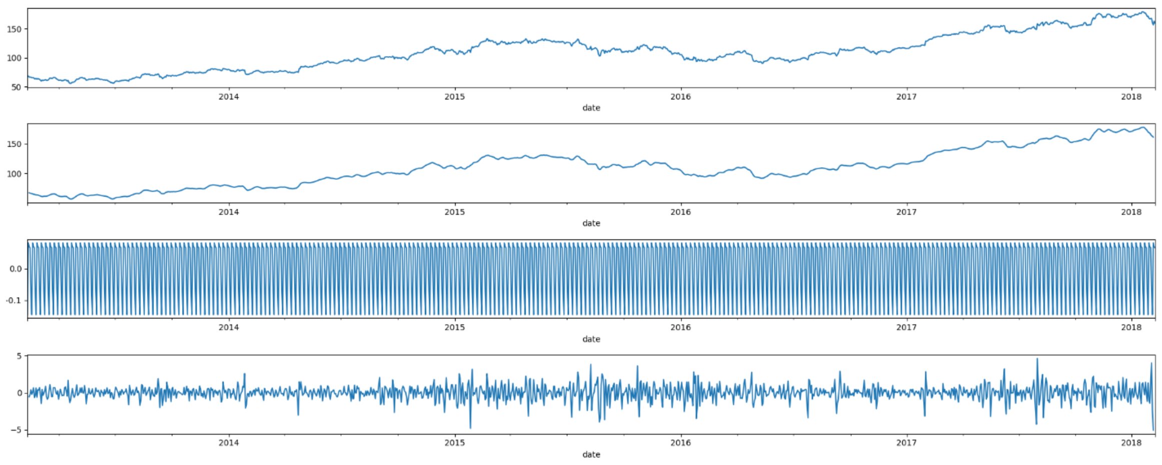


Chart 1 Candlestick chart of stocks

As seen from the graph, the timeserieses exhibits trends, which indicate non-stationarity.

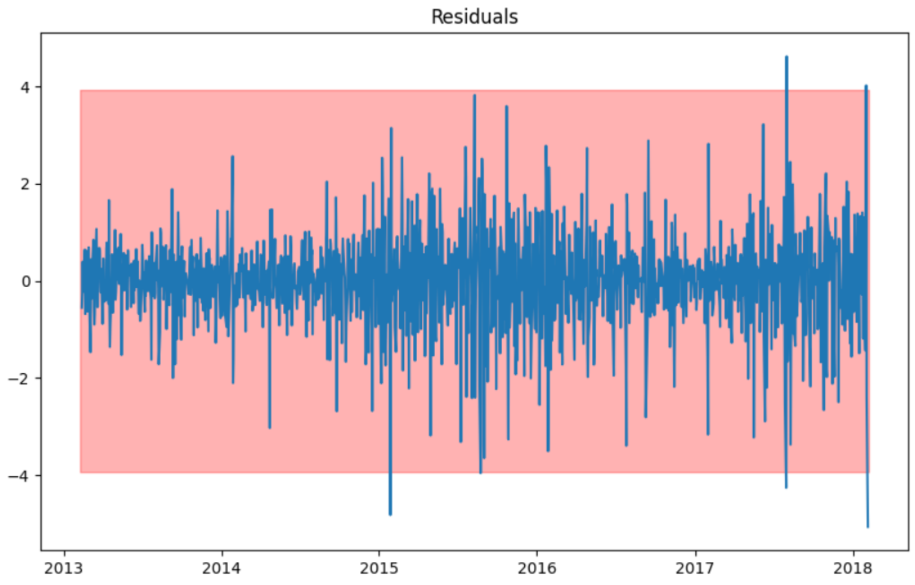
To confirm this, we can use decomposition or perform statistical tests such as the Augmented Dickey-Fuller (ADF) test.

a. Decomposition



The graphs show the original time series, trend, seasonal and residual, respectively.

- The Trend graph indicates there is an upward trend in our timeseries
- The Seasonality graph indicates there is no seasonality in our timeseries
- Using Residual graph to detect any anomalies/outlies.
  Here is the residual plot in range between: mean $\pm$ 4*standard derivation. There are some outlines but not too significant.



→ These information indicates that this is not a stationary timeseries.

b. ADF test

The ADF test shows the similar conclusion that this is not a stationary timeseries as the p values is much larger than 5%:

```
ADF Statistic: -0.660437
p-value: 0.856733
Critical Values:
        1%: -3.435
        5%: -2.864
        10%: -2.568
```

2.2.4 Transform to Stationary

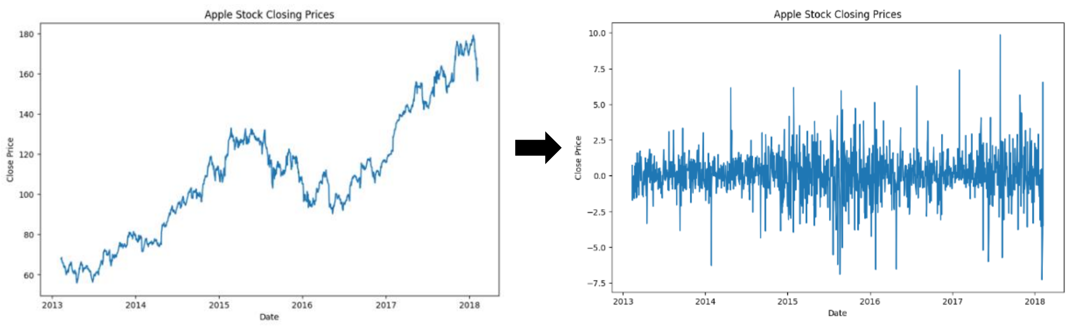
To transform a non-stationary timeseries into a stationary one, several methods exist:

- **Differencing**: Subtracting each data point from the previous one: $T_t = S_t - S_{t-1}$. This stabilizes the mean and often removes trends, making analysis and modeling easier. However, excessive differencing can remove under structure of the data, introduce unwanted white noise.

- **Returns** (Percentage Change): Calculating the percentage change between consecutive data points $T_t = \frac{S_t - S_{t-1}}{S_{t-1}}$. This can reveal hidden patterns (e.g., growth rates) but may not fully stabilize the mean if a strong trend exists.

Differencing is generally simpler to implement and is effective for removing strong trends. In this project, given the presence of a strong trend in our original dataset , we will prioritize differencing as our primary transformation method.

From the original dataset, we use diff( ) in Python to transform the timeseries as visualized here:



Then we check the stationary again using ADF test

ADF Statistic: -17.326398

p-value: 0.000000

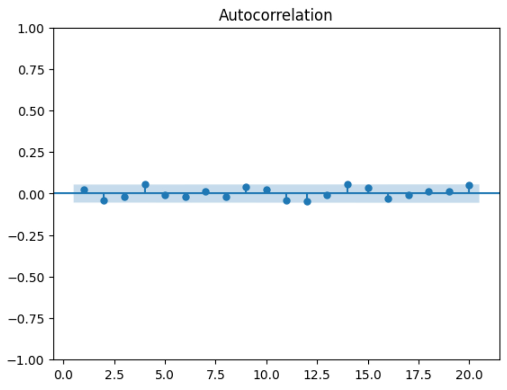
Critical Values:

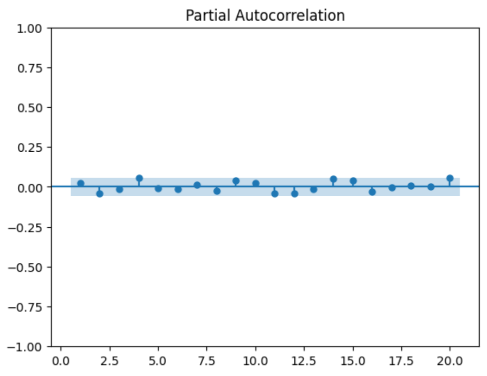
1%: -3.435

5%: -2.864

10%: -2.568

The p-value is close to zero so we can conclude that the timeseries is now stationary. However, when we plot the ACF and PACF from this timeseries:





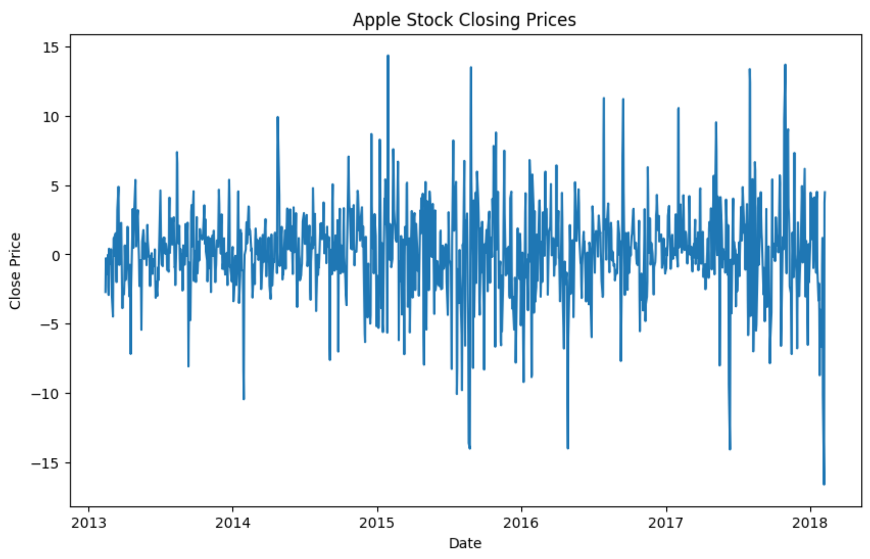
As shown from the graph, there is no correlation between lags, raising concerns that using diff() (by default diff() is diff(1) in Python library) might have over-differenced the series. It is possible that we have removed too much of the original structure. Alternatively, the original series might not have had strong autocorrelations to begin with and the timeseries now can be a white noise. To confirm this, we use Ljung Box to check:

```
from statsmodels.stats.diagnostic import acorr_ljungbox

# Ljung-Box test
result = acorr_ljungbox(df['differ1'].dropna(), lags=[10, 20, 30], return_df=True)
print(result)

# Check for white noise
is_white_noise = result['lb_pvalue'].iloc[-1] > 0.05

if is_white_noise:
    print("The time series is likely white noise.")
else:
    print("The time series is not likely white noise.")
```

Then we get the result:

```
      lb_stat  lb_pvalue
10  11.224603   0.340289
20  27.017072   0.134784
30  36.965348   0.178156
The time series is likely white noise.
```

The p-value significantly exceeds the threshold of 0.05, indicating that the data can be characterized as white noise. Consequently, attempting to construct models such as AR (Autoregressive) or MA (Moving Average) on this dataset would be futile, as white noise is inherently unpredictable. Considering the non-linear pattern observed, using diff(1) may not yield optimal results, so we try some other alternatively solution, such as using diff(2) instead. Below is the graph reflecting the changes after implementing diff(2):



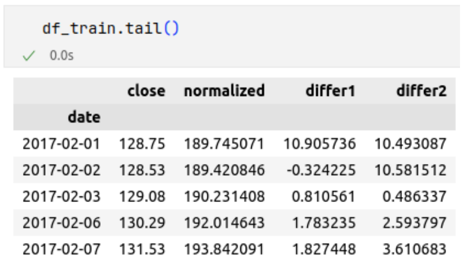
Run the check again:

```
ADF Statistic: -7.469759
p-value: 0.000000
Critical Values:
        1%: -3.435
        5%: -2.864
        10%: -2.568
       lb_stat     lb_pvalue
10  326.413563  3.996189e-64
20  350.074492  4.300175e-62
30  362.537781  9.699117e-59
The time series is not likely white noise.
```

The p-value of ADF test is close to 0 and p-value in Ljung Box is also much smaller than 0.05 indicates that the time series now is stationary but not a white noise, ensuring that we can build AR, MA models on top of this new time series.

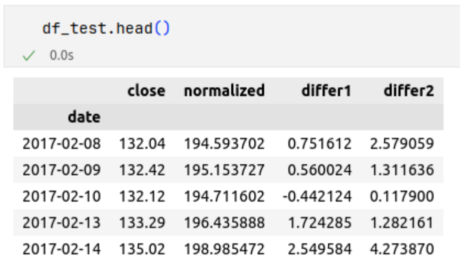
2.2.5 Split the Dataset

Like every machine problem, we need to divide the dataset to evaluate the model performance. Using hold out technique, we split out original dataset to 80% and 20% test. However, since we are working with time series, we rely on keeping the chronological order of the records. Therefore, data in time series should be uninterrupted sequences of values →The training set should include all values from the beginning to a specific point in time and the testing set should include from that time period to the ending point.



```
df_train.tail()
✓ 0.0s
```

	close	normalized	differ1	differ2
date				
2017-02-01	128.75	189.745071	10.905736	10.493087
2017-02-02	128.53	189.420846	-0.324225	10.581512
2017-02-03	129.08	190.231408	0.810561	0.486337
2017-02-06	130.29	192.014643	1.783235	2.593797
2017-02-07	131.53	193.842091	1.827448	3.610683



```
df_test.head()
✓ 0.0s
```

	close	normalized	differ1	differ2
date				
2017-02-08	132.04	194.593702	0.751612	2.579059
2017-02-09	132.42	195.153727	0.560024	1.311636
2017-02-10	132.12	194.711602	-0.442124	0.117900
2017-02-13	133.29	196.435888	1.724285	1.282161
2017-02-14	135.02	198.985472	2.549584	4.273870

The last day in the train set must be followed by the first day in the testing set.

CHAPTER 3. MODELS

3.1 Theory

3.1.1 Overview about time series analysis model

Timeserieses forecasting is a critical area in statistics, econometrics, and machine learning. These models are applied to data sequences that involve time. The primary assumption of time series models is that past patterns will repeat in the future. Thus, building a time series model involves modeling past relationships between the independent variable (input) and the dependent variable (target) to predict future values of the dependent variable.

Timeserieses forecasting has wide applications in various fields, including finance, banking, insurance, e-commerce, marketing, and policy management. Some common applications include:

- Forecasting market demand to plan production and business activities.
- Predicting financial asset returns, exchange rates, and commodity prices for effective trading in market risk.
- Forecasting stock prices and portfolio returns to manage investments.
- …

Or some application in Cyber Security field:

- Network Traffic Monitoring: Timeserieses analysis can identify unusual patterns in network traffic that may indicate cyber-attacks, such as DDoS attacks, port scanning, or data exfiltration attempts.
- Behavioral Analysis: Monitoring user and system behavior over time to detect anomalies that could signify compromised accounts or insider threats.
- Signature-Based Detection: Timeserieses analysis helps in identifying known attack patterns based on historical data, improving the detection of repeated attack vectors in Intrusion Detection Systems (IDS).
- …

Unlike typical forecasting models in machine learning, time series models in econometrics have unique characteristics. They require strict adherence to conditions like stationarity, white noise, and autocorrelation. Although, In reality, timeserieses data mostly exhibit characteristics such as cycles, seasonality, and trends, which require us to transform them into stationary form.

Various classes of time series models exist, each with specific application standards. Some common models include:

- **ARIMA Model:** Based on the assumption of stationarity and constant error variance. ARIMA uses past values of the series to forecast future values, incorporating autoregressive (AR) and moving average (MA) components. If the series is non-stationary, it is transformed using differencing. The model is specified by three parameters: ARIMA(p, d, q), where p is the order of the AR part, d is the degree of differencing, and q is the order of the MA part.
- **SARIMA Model:** An extension of ARIMA for seasonal timeserieses. SARIMA adjusts for seasonality by identifying and removing the seasonal pattern before applying the ARIMA model.
- **ARIMAX Model:** An extension of ARIMA that includes exogenous variables. ARIMAX models linear relationships between past values and variances with the current value, using a regression equation derived from past relationships. This model includes additional independent variables and is advantageous for forecasting by accounting for autocorrelation in the residuals, thus improving accuracy.
- **GARCH Model:** Addresses the issue of changing variance in financial and economic timeserieses, which often experience unexpected events and economic shocks. Unlike ARIMA, GARCH models can forecast variance to manage unexpected changes, resulting in better forecasting performance.
- **…**

In the scope of this project, we will only focus on some models: AR, MA, ARMA, ARIMA, Prophet

3.1.2 Stationary

Why do we need stationary time series?

In a stationary time series, the statistical properties of the data points are consistent and do not depend on the time at which they are observed. This means that the relationships and patterns observed in the data are dependable and can be used to make accurate forecasts.

In contrast, a non-stationary time series has statistical properties that change over time, which can make it difficult to draw reliable inferences or make accurate forecasts. As the statistical properties of the data keep changing, any model or analysis based on a non-stationary time series may not provide reliable results.

Therefore, analyzing stationary data is easier and more reliable than non-stationary data. Stationary data allow for the use of simpler models and statistical techniques, as well as more accurate predictions. Using non-stationary data can lead to inaccurate and misleading forecasts, as the underlying statistical properties of the data keep changing with time.

What is a stationary time series?

The concept of stationarity in time series analysis revolves around the idea of consistency. Specifically, it refers to the consistency of the distribution over time: the distribution depends only on the window size, not the specific location in time.

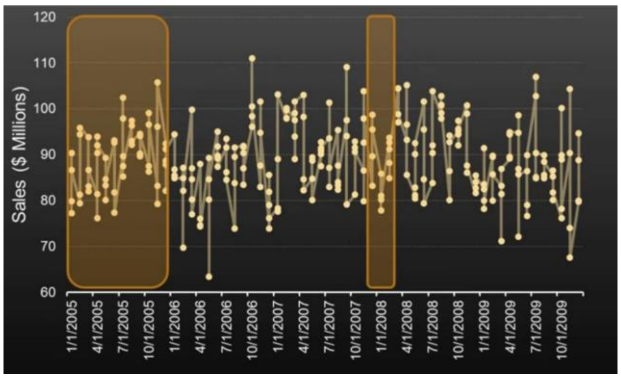


Figure1, with different window sizes, the distribution of time series values within these windows can be different.

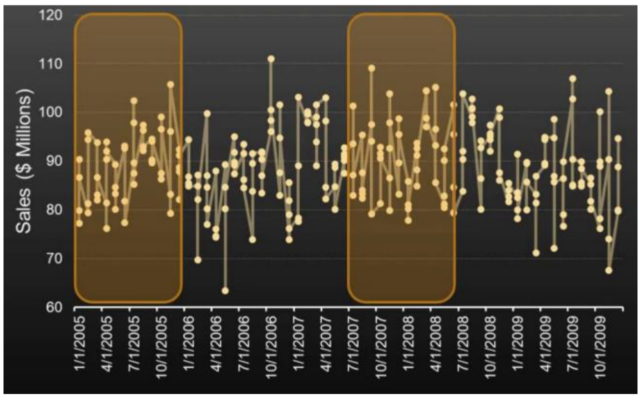


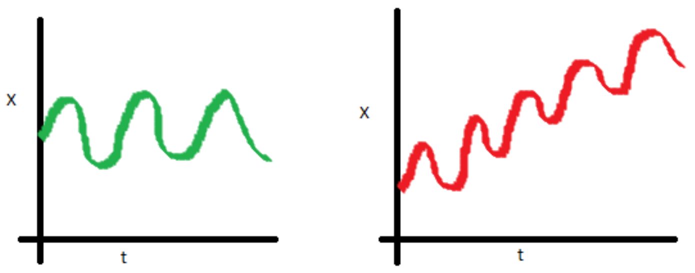
Figure2, with windows of the same size (or the same number of lags), we expect the distribution of values within these windows to be the same

This consistency across windows of the same size is what defines **strong stationarity**.

On the other hand, **weak stationarity** (also known as second-order stationarity or wide-sense stationarity) does not require the entire distribution to be consistent across windows of the same size. Instead, it requires only certain statistical properties, such as the mean, variance, and autocorrelation to remain constant over time. This is sufficient for many class time series analysis models to operate. Therefore, in this report, we will primarily rely on weak stationarity to ensure the validity of our time series analysis models.

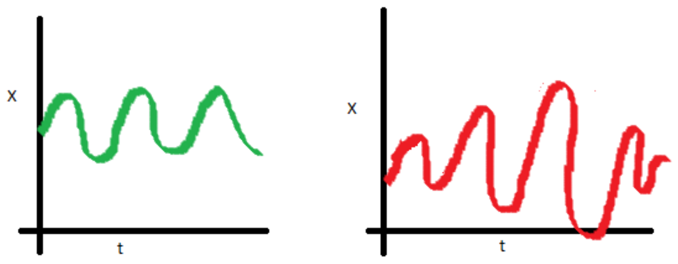
For a time, series to be weakly stationary, it must satisfy the following conditions:

1. **The mean (μ)** of the series should be constant over time, not dependent on the time at which it is measured. In the image below, the left-hand graph shows a series with a constant mean, whereas the graph in red shows a series with a time-dependent mean.



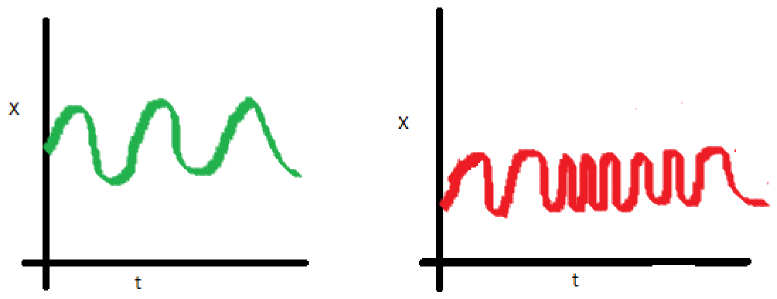
Stationary series Non-Stationary series

2. **The variance** of the series should be constant over time. The following graph depicts what is and what is not a stationary series. (Notice the varying spread of distribution in the right-hand graph)



Stationary series Non-Stationary series

3. **Constant Autocorrelation**: The autocorrelation between values of the series at different times should depend only on the time lag between them, not on their absolute position in time. In the following graph, you will notice the spread becomes closer as the time increases. Hence, the covariance is not constant with time for the 'red series'.

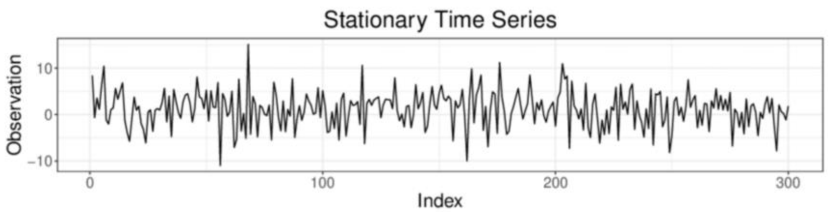


Stationary series Non-Stationary series

Some definitions of weak stationarity may replace this third condition with the requirement that there is no trend or seasonality in the time series.

By ensuring these properties, we can apply various time series analysis models effectively to our data. To test whether a time series is stationary, there are several methods that can be used. Here are some common techniques:

- **Visual Inspection**: Plot the time series data and observe whether there is a clear trend or seasonality. If the data appears to fluctuate around a consistent mean with no noticeable pattern, it is likely stationary. A stationarity time series may look pretty flat like this:



- **Splitting time / Decomposition** series into the different partitions and compare the statistical inference.

- **Summary Statistics**: Calculate the mean and standard deviation of the time series and check if they are consistent over time. If they are constant, the time series may be considered stationary.
- **Dickey-Fuller Test/Augmented Dickey-Fuller (ADF) Test** is a statistical test that belongs to the unit root test which tests the null hypothesis.

- **Kwiatkowski-Phillips-Schmidt-Shin (KPSS) Test**: This test checks for the presence of a trend or structural break in the time series. If the test indicates that there is no trend or structural break, then the time series may be considered stationary.

In this report, we will only focus on three common methods which are visualization, decomposition and ADF test.

What if a time series is not stationary ?

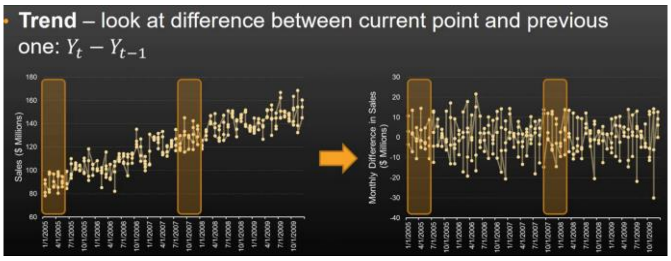
If the data is not stationary, there are some approaches we can try to make it become stationary. The two most common methods to transform series into stationarity are:

- **Transformation**: Using log or square root to stabilize the non-constant variance: $y_t' = \log(y_t)$

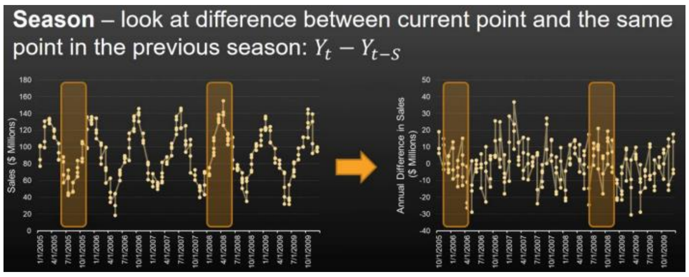
- **Differencing**: Subtracts the current value from the previous one, in order to remove trends and stabilize the mean of a timeserieses. Differencing can be done in different orders like first-order differencing(will have linear trend), second-order differencing (will have quadratic trends). Differencing can also deal with seasonality.

  The first difference of the series can be calculated as: $y_t' = y_t - y_{t-1}$, where $y_t$is the original value and y_t' is differenced value.

- **Returns** (Percentage Change): Calculating the percentage change between consecutive data points T$_t$ = $\frac{S_t - S_{t-1}}{S_{t-1}}$. This can reveal hidden patterns (e.g., growth rates) but may not fully stabilize the mean if a strong trend exists.


Trend – look at difference between current point and previous one: $Y_t - Y_{t-1}$

Transform Seri contains Trend into stationary Seri


Season – look at difference between current point and the same point in the previous season: $Y_t - Y_{t-s}$

Transform Seri contains seasonality into stationary Seri

There are still more techniques to transform a time series to stationarity, more detail can be found here.

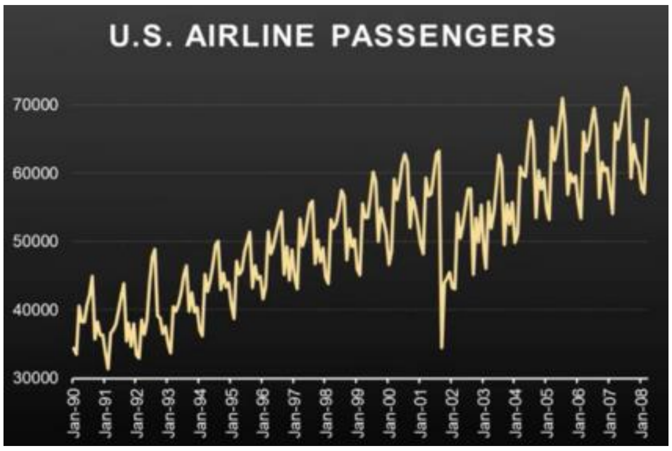
3.1.3 Decomposition of the Series

Decomposition involves breaking down the timeserieses into its fundamental components: trend, seasonality, and residuals(noise). This helps in understanding the underlying patterns and in building better predictive models.

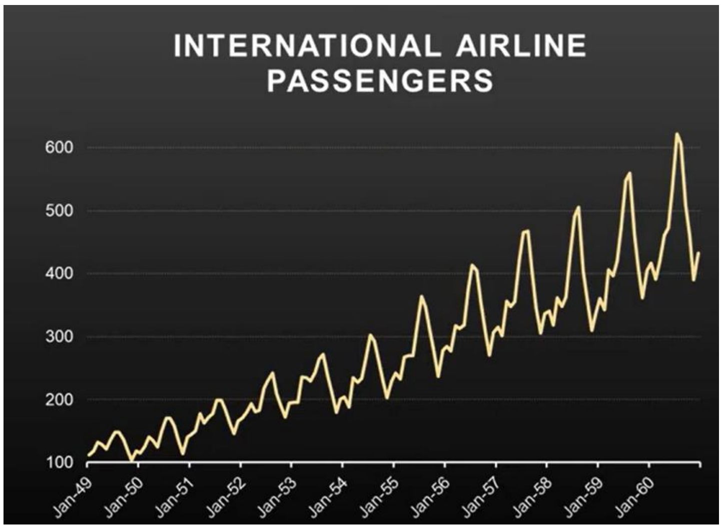
- **Trend:** This is the overall motion of the series. It may be consistently increasing overtime, decreasing overtime or a combination of both.
- **Seasonality:** Any regular seasonal pattern in the series. For example, ice cream sales are regularly higher in summer than winter.
- **Residual/Remainder:** This is the bit that is left over after we take into account the trend and seasonality. It can also be thought of as just statistical noise.

These three main components can form the time series by two ways:

- **Additive Decomposition:** the time series is expressed as the sum of its components: Y(t) = Trend(t) + Seasonal(t) + Residual(t). It is suitable when the magnitude of seasonality does not vary with the magnitude of the time series.
- **Multiplicative Decomposition:** the time series is expressed as the product of its components: Y(t) = Trend(t) x Seasonal(t) x Residual(t). It is suitable when the magnitude of seasonality scales with the magnitude of the time series.



Additive Timeserieses: Trend and season are added on behind each other, the amplitude of these seasons does not change.



Multiplicative Timeserieses: The amplitude is growing; it is sort of multiplying upon itself

3.1.4 Augmented Dickey-Fuller (ADF) test

The Augmented Dickey-Fuller Test is one of the most popular tests to check for stationarity. It tests the below hypothesis.

- Null Hypothesis, H0: The time series is not stationary.
- Alternative Hypothesis, H1: The time series is stationary.

To perform this test, we use the **adfuller** method from the **statsmodels** in Python and compare the value of the test statistics or the p-value.

- If the p-value is less than or equal to **0.05** or the absolute value of the test statistics is greater than the critical value, you reject H0 and conclude that the timeseries is stationary.
- If the p-value is greater than 0.05 or the absolute value of the test statistics is less than the critical value, you fail to reject H0 and conclude that the timeseries is not stationary.

3.1.5 Log Likelihood Ratio Test

The Log Likelihood Ratio (LLR) test in time series analysis is a statistical method used to compare the goodness of fit between two competing models. Specifically, it evaluates whether the inclusion of additional parameters in a model significantly improves the model's ability to describe the observed data1.

Here is how the LLR test works:

- It calculates the log likelihood values for both models.
- The test statistic is computed as
  −2×(log likelihood of the simpler model−log likelihood of the more complex model)
- This statistic follows a chi-square distribution, and if it exceeds a critical value, the more complex model is considered to have a significantly better fit2.

```
def LLR(simple_model, complex_model, df):
    #df: Degrees of freedom (difference in parameters between models).
    llr = 2 * (complex_model.llf - simple_model.llf)
    p_value = 1 - chi2.cdf(llr, df)
    return  p_value
```

- The LLR test is used to compare the same model but with different lags.

3.1.6 AIC and BIC

AIC stands for Akaike Information Criteria, and it is a statistical measure that we can use to compare different models for their relative quality. It measures the quality of the model in terms of its goodness-of-fit to the data, its simplicity, and how much it relies on the tuning parameters. The formula for AIC is:

$$AIC = 2k - 2l$$

With:

- l: measures how strong the model fit the data, the more complicated model, the better it can fit the data
- k: number of parameters, for example an AR(4) model will have 5 parameters (4 coefficients and 1 constant/bias)

From the formula above, we can conclude that AIC prefers a higher log-likelihood that indicates how strong the model is in fitting the data and a simpler model in terms of parameters. We will want to choose the model with AIC as small as possible, since this is a simple model but still fit the data well

- AIC help use choose a good one between many models.

In addition to AIC, the BIC (Bayesian Information Criteria) uses one more indicator n that defines the number of samples used for fitting. The formula for BIC is:

$$BIC = k \log n - 2l$$

From the formula, BIC will only be useful if we want to compare between models training on different dataset with different size.

3.1.7 Autoregressive (AR) Model

The AR model is a statical model that expresses the dependent of one variable on an earlier time period. It is a model where signal y_t depends only on its own past value. We can select the order p for AR(p) model based on significant spikes from the PACF model. The formula of AR model is as follow:

$$y_t = c + \sum_{i=1}^{p} \phi_i y_{t-i} + \epsilon_t$$

where y_t is the value at time t, c is a constant, ϕ_i are the coefficients, and ϵ_t is error term, often the white noise.

3.1.8 Moving Average (MA) Model

The MA model specifies that the output variable depends linearly on the current and various past error terms. In constract to AR model, we can select the order q for the model MA(q) from ACF if this ploit has a sharp cut-off after lag q

$$y_t = c + \epsilon_t + \sum_{i=1}^{q} \theta_i \epsilon_{t-i}$$

where θ_t are the coefficients.

3.1.9 ARMA Model

The ARMA model combines AR and MA models:

$$y_t = c + \sum_{i=1}^{p} \phi_i y_{t-i} + \epsilon_t + \sum_{j=1}^{q} \theta_j \epsilon_{t-j}$$

3.1.10 ARIMA Model

Manually finding the right parameters (p and q) for an ARMA model can be challenging and time-consuming, relying solely on visual analysis of ACF and PACF plots. Fortunately, simpler approaches exist, often referred to as "auto ARIMA."

Many statistical tools, including those in Python and R, have built-in auto ARIMA functionality that automatically determines the optimal p and q parameters for your dataset, leading to better forecasting. The concept behind this is similar to hyperparameter tuning in machine learning: trying different combinations of p and q and comparing their performance on a validation set. A common method for this is grid search.

The ARIMA model (AutoRegressive Integrated Moving Average) is a generalization of the ARMA model that includes differencing to make the timeserieses stationary:

$$y'_t = c + \sum_{i=1}^{p} \phi_i y'_{t-i} + \epsilon_t + \sum_{j=1}^{q} \theta_j \epsilon_{t-j}$$

where y'_t is the differenced series. The parameters are:

- **p**: number of lag observations in the model
- **d**: number of times that the raw observations are differenced
- **q**: size of the moving average window

3.1.11 Prophet

Prophet is a forecasting tool developed by Facebook that is particularly effective for timeserieses with daily observations that display patterns on different time scales. It decomposes the timeserieses into trend, seasonality, and holiday effects:

$$y(t) = g(t) + s(t) + h(t) + \epsilon_t$$

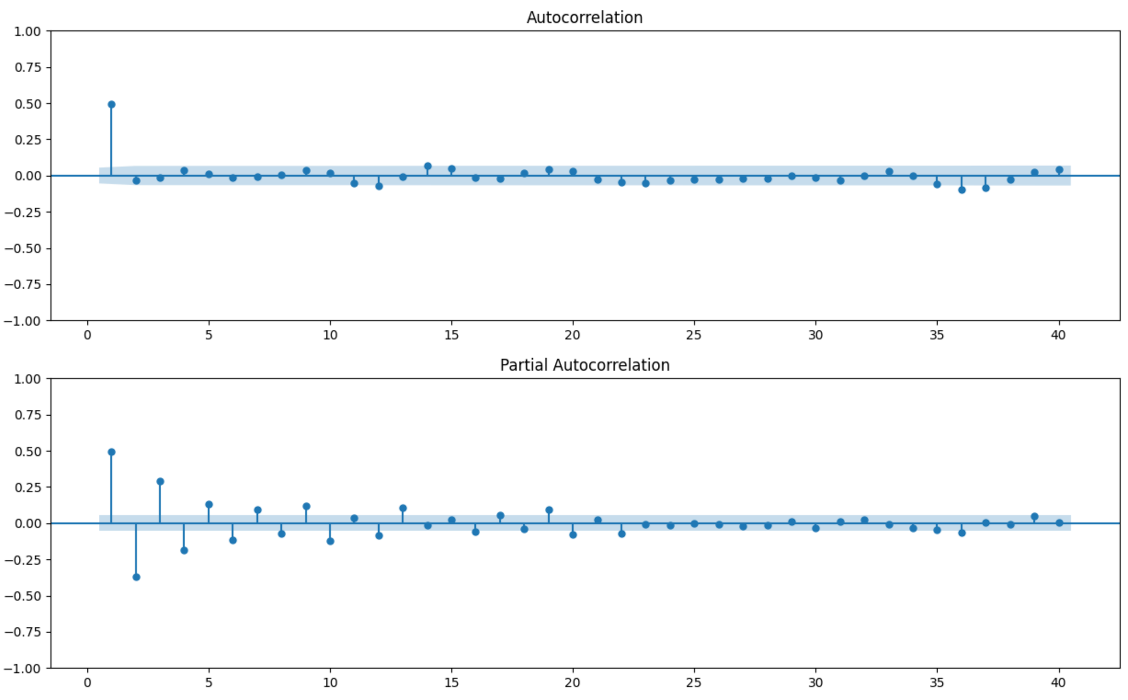
where:

- $g(t)$ models the trend,
- $s(t)$ models the seasonality,
- $h(t)$ models the holidays,
- ϵ_t is the error term.

3.2 Implementation

3.2.1 Hyperparameter for AR, MA

Using PACF and ACF plots, we determine the appropriate values for p and q parameters for the AR and MA components. Here is the ACF and PACF plot from the timeseries after preprocessing:



The Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF) are essential tools for identifying patterns in timeseries data and selecting appropriate model orders for AR, MA models.

The ACF measures the correlation between a time series and its lagged values, including both direct and indirect influences. In contrast, the PACF isolates the direct correlation between a timeseries and its lagged values, removing the effects of intervening lags.

When selecting model orders:

- ACF: Significant spikes in the ACF suggest the presence of Moving Average (MA) terms. Each lag with a significant correlation greater than zero indicates a potential MA term to include in the model.
- PACF: Significant spikes in the PACF suggest the presence of Autoregressive (AR) terms. Each lag with a significant correlation greater than zero indicates a potential AR term to include.

Any line the exceed the light blue rectangle is considered as significant term

In general, simpler models (those with fewer lags) are preferred unless a more complex model offers significantly better predictive performance. To assess this "significance," we can employ statistical tests like LLR.

A well-fitting model should capture all underlying trends, leaving residuals that resemble white noise. This indicates that no significant patterns have been missed, and the model is not overfit.

Of course, when looking into this graph, we should use MA due to simplicity, but for the demonstration purpose, we will try both of them.

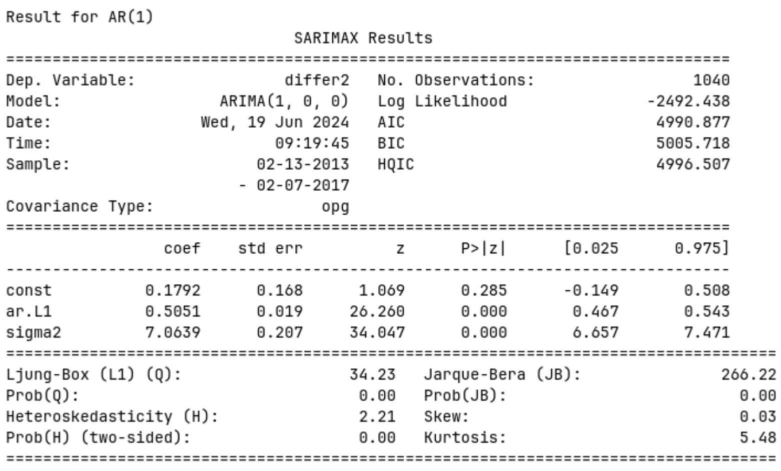
3.3 Train

3.3.1 AR

We defined a function for testing AR with many orders:

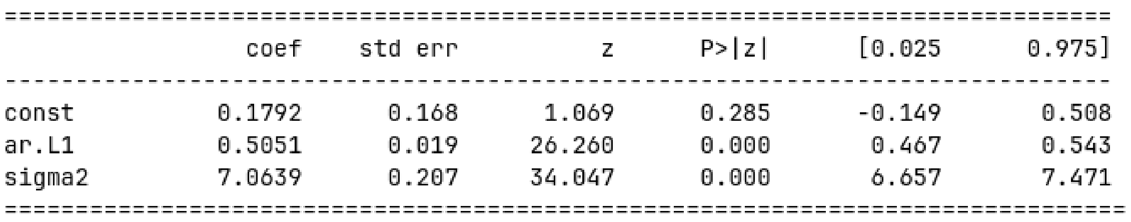
```python
from statsmodels.tsa.arima.model import ARIMA
def train_ar_model(df_train, order):
    # Create the ARIMA model
    model = ARIMA(df_train[column_used], order=order)
    # Fit the model
    result = model.fit()
    print("Result for AR(%d)" % order[0])
    print(result.summary())
    return result
```

Try with AR(1) with order specified as order=(1,0,0), we get the summary result table:



```
Result for AR(1)
                               SARIMAX Results
==============================================================================
Dep. Variable:                differ2   No. Observations:                 1040
Model:                 ARIMA(1, 0, 0)   Log Likelihood               -2492.438
Date:                Wed, 19 Jun 2024   AIC                           4990.877
Time:                        09:19:45   BIC                           5005.718
Sample:                    02-13-2013   HQIC                          4996.507
                         - 02-07-2017
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.1792      0.168      1.069      0.285      -0.149       0.508
ar.L1          0.5051      0.019     26.260      0.000       0.467       0.543
sigma2         7.0639      0.207     34.047      0.000       6.657       7.471
===================================================================================
Ljung-Box (L1) (Q):                  34.23   Jarque-Bera (JB):               266.22
Prob(Q):                              0.00   Prob(JB):                         0.00
Heteroskedasticity (H):               2.21   Skew:                             0.03
Prob(H) (two-sided):                  0.00   Kurtosis:                         5.48
===================================================================================
```

But we just focus on the main content here:



```
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.1792      0.168      1.069      0.285      -0.149       0.508
ar.L1          0.5051      0.019     26.260      0.000       0.467       0.543
sigma2         7.0639      0.207     34.047      0.000       6.657       7.471
===============================================================================
```

- The first column expresses the coefficients of the AR equation. For example, with AR(1): $y_t = c + \theta_1 y_{t-1} + \epsilon_t$ then we can read from the table: c = 0.1792, $\theta_1 = 0.5051$.
- The second column is the mean and standard deviation
- The fourth column, which is important, is used to determine if the coefficient is significant. The p-value tests the null hypothesis that the coefficient is equal to zero (no effect). A low p-value (< 0.05) indicates that we can reject the null hypothesis. In other words, the coefficient is significant and can be added to the model.

To choose a good AR model, we will need to try adding more lags to the model until we observe:

- Non – significant p-value for the highest lag coefficients (to ensure that coefficient is still above the significant line)
- Non-significant p-value for the LLR test (LLR fails or adding more lags does not make the model much better).

In our project context, we tried from AR(1) to AR(14), at each step we all compare AR(n) and AR(n+1) by LLR test, here is summary result:

AR(2):

ar.L1	0.6863	0.021	33.296	0.000	0.646	0.727
ar.L2	-0.3587	0.022	-16.117	0.000	-0.402	-0.315

AR(3):

ar.L1	0.7790	0.023	33.706	0.000	0.734	0.824
ar.L2	-0.5355	0.029	-18.211	0.000	-0.593	-0.478
ar.L3	0.2574	0.024	10.516	0.000	0.209	0.305

…..

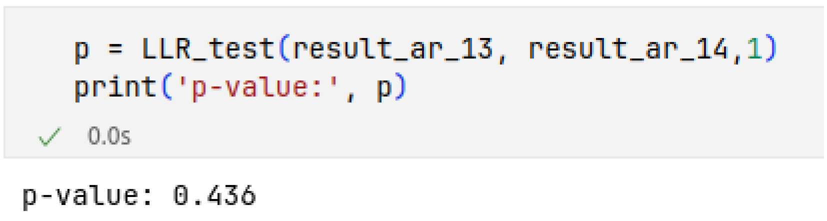
AR(13) (only the p-value column)

ar.L12	9.065850e-07
ar.L13	5.288544e-05

AR(14)

ar.L13	1.569782e-04
ar.L14	2.325376e-01

Finally, at AR(14), we see the last coefficient L14 is above 0.05, indicating that this coefficient is not significant, and we should stop adding more coefficient. Also, the LLR test fails when compared to AR(13):

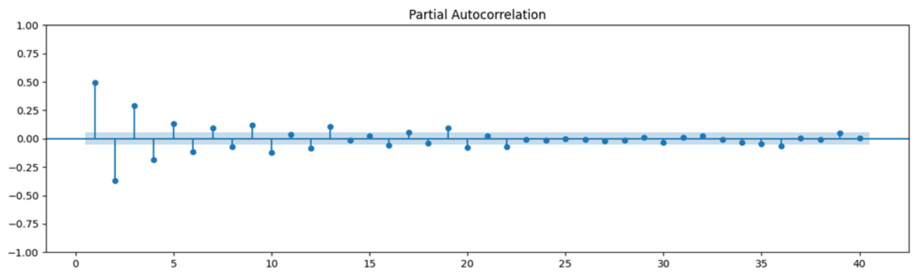


```
p = LLR_test(result_ar_13, result_ar_14,1)
print('p-value:', p)
```
✓ 0.0s

```
p-value: 0.436
```

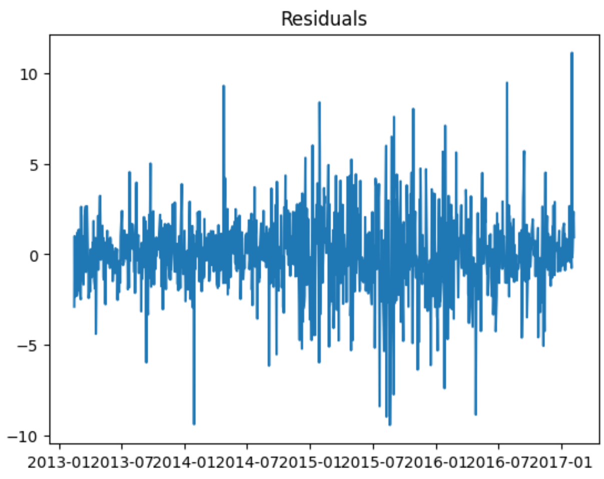
Therefore, with AR model, our final model should be AR(13).

Look back the PACF from beginning, we see the corresponding to the number of orders:



From some lags near 15 start to go down the significant level

Anylysis the residual of the AR(13), we see:



Test for this residual, it is truly a white noise, meaning that our model fit the data well and do not miss and underlying pattern:

```
       lb_stat  lb_pvalue
10    8.640308   0.566545
20   26.283395   0.156655
30   32.761847   0.332970
The time series is likely white noise.
```

3.3.2 MA

In time series analysis, Moving Average (MA) models are used to model the dependency of an observation on residual errors from a moving average model applied to lagged observations. The key characteristic of MA models is that they depend on past forecast errors rather than past values of the series itself. To determine the order for a MA mode, we use the ACF

For MA models, the ACF has a distinct pattern:

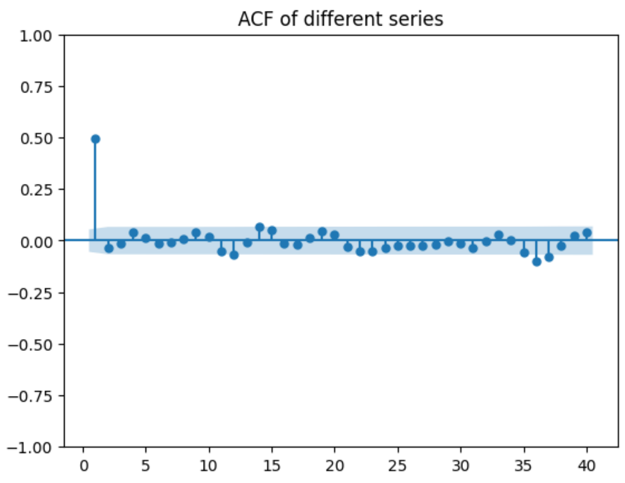
For an MA(1) model, the ACF will show a significant spike at lag 1 and then cut off abruptly.

For an MA(2) model, significant spikes will be observed at lags 1 and 2, followed by an abrupt cutoff.

Generally, for an MA(q) model, the ACF will show significant spikes for the first q lags and then cut off to near zero after that.

This cutoff behavior occurs because MA models incorporate past error terms. Once the model order q is reached, subsequent autocorrelations are not significant because they do not directly depend on the observed values of the series, but rather on the residuals of the past values.\

AS in our case, with the ACF we should use MA(1):



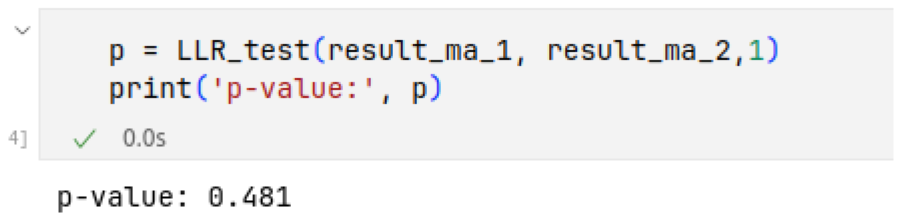
The result for MA(1) as follows:

```
              coef    std err         z     P>|z|     [0.025      0.975]
--------------------------------------------------------------------------------
const        0.1783      0.134      1.331      0.183      -0.084      0.441
ma.L1         0.9978      0.006    164.488      0.000       0.986       1.010
sigma2        4.6127      0.119     38.735      0.000       4.379       4.846
```

And when we try with MA(2):

```
              coef    std err         z     P>|z|     [0.025      0.975]
------------------------------------------------------------------------------
const        0.1782      0.139      1.284      0.199      -0.094      0.450
ma.L1         1.0292      0.024     42.220      0.000       0.981      1.077
ma.L2         0.0316      0.024      1.296      0.195      -0.016      0.079
sigma2        4.6085      0.119     38.766      0.000       4.376      4.842
```

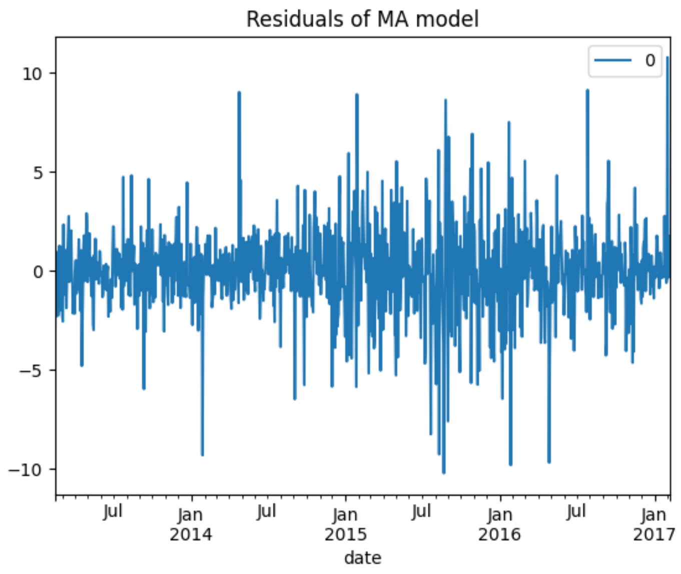
Runing the LLR shows that the MA(2) is not better than MA(1)



```
p = LLR_test(result_ma_1, result_ma_2,1)
print('p-value:', p)
```

```
p-value: 0.481
```

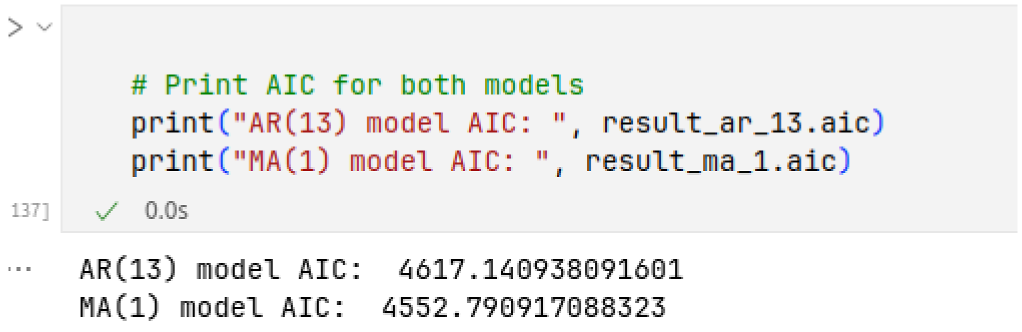
After training the MA(1) model, we get the residual as follows:



It also is white noise, indicated that our model fit well on the data.

```
      lb_stat  lb_pvalue
10   5.280209   0.871691
20  16.975366   0.654573
30  22.775576   0.824372
The time series is likely white noise.
```

And to compare between two model AR and MA, we use AIC:



```
# Print AIC for both models
print("AR(13) model AIC: ", result_ar_13.aic)
print("MA(1) model AIC: ", result_ma_1.aic)
```

```
137]  ✓  0.0s
```

```
...  AR(13) model AIC:  4617.140938091601
     MA(1) model AIC:  4552.790917088323
```

The lower AIC value for the MA model indicates that it strikes a better balance between fitting the data and model complexity compared to the AR model. AIC considers both the goodness of fit of the model and the number of parameters used, penalizing models that are overly complex.

Therefore, based on this lower AIC value, the MA model appears to be a more suitable choice for this particular dataset.

3.3.3 ARIMA

The ARIMA has already have the I components inside it, so we no need to use differencing technique to make the timeseries stationary in Section 2.

In Auto ARIMA, the model itself will generate the optimal p, d, and q values which would be suitable for the data set to provide better forecasting.

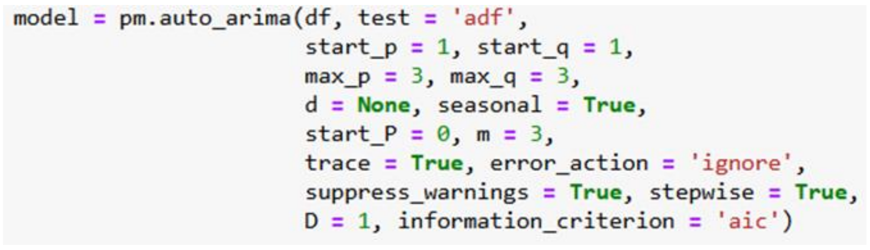
Auto-Regressive (p) -> Number of autoregressive terms.

Integrated (d) -> Number of nonseasonal differences needed for stationarity.

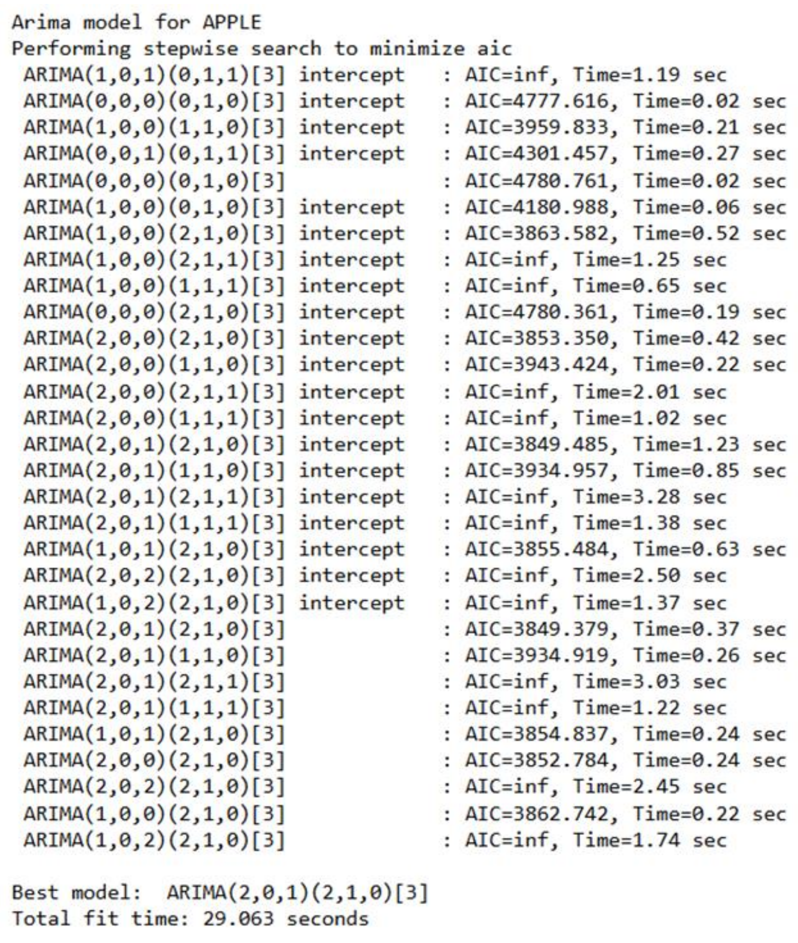
Moving Average (q) -> Number of lagged forecast errors in the prediction equation.

In the Auto ARIMA model, note that small p, d, q values represent non-seasonal components, and capital P, D, Q represent seasonal components. It works similarly like hyper tuning techniques to find the optimal value of p, d, and q with different combinations and the final values would be determined with the lower AIC, BIC parameters taking into consideration

Here, we are trying with the p, d, q values ranging from 0 to 3 to get better optimal values from the model.



```
model = pm.auto_arima(df, test = 'adf',
                      start_p = 1, start_q = 1,
                      max_p = 3, max_q = 3,
                      d = None, seasonal = True,
                      start_P = 0, m = 3,
                      trace = True, error_action = 'ignore',
                      suppress_warnings = True, stepwise = True,
                      D = 1, information_criterion = 'aic')
```

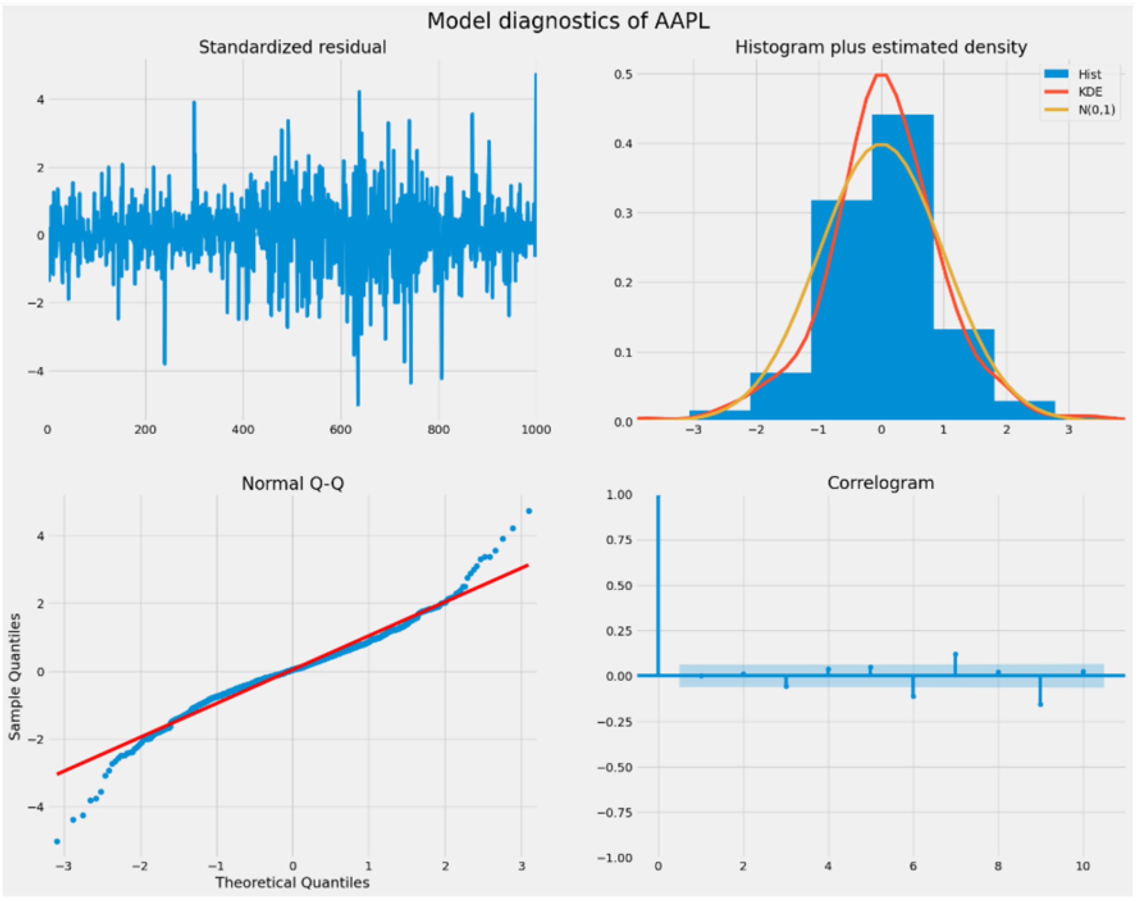


```
Arima model for APPLE
Performing stepwise search to minimize aic
 ARIMA(1,0,1)(0,1,1)[3] intercept   : AIC=inf, Time=1.19 sec
 ARIMA(0,0,0)(0,1,0)[3] intercept   : AIC=4777.616, Time=0.02 sec
 ARIMA(1,0,0)(1,1,0)[3] intercept   : AIC=3959.833, Time=0.21 sec
 ARIMA(0,0,1)(0,1,1)[3] intercept   : AIC=4301.457, Time=0.27 sec
 ARIMA(0,0,0)(0,1,0)[3]             : AIC=4780.761, Time=0.02 sec
 ARIMA(1,0,0)(0,1,0)[3] intercept   : AIC=4180.988, Time=0.06 sec
 ARIMA(1,0,0)(2,1,0)[3] intercept   : AIC=3863.582, Time=0.52 sec
 ARIMA(1,0,0)(2,1,1)[3] intercept   : AIC=inf, Time=1.25 sec
 ARIMA(1,0,0)(1,1,1)[3] intercept   : AIC=inf, Time=0.65 sec
 ARIMA(0,0,0)(2,1,0)[3] intercept   : AIC=4780.361, Time=0.19 sec
 ARIMA(2,0,0)(2,1,0)[3] intercept   : AIC=3853.350, Time=0.42 sec
 ARIMA(2,0,0)(1,1,0)[3] intercept   : AIC=3943.424, Time=0.22 sec
 ARIMA(2,0,0)(2,1,1)[3] intercept   : AIC=inf, Time=2.01 sec
 ARIMA(2,0,0)(1,1,1)[3] intercept   : AIC=inf, Time=1.02 sec
 ARIMA(2,0,1)(2,1,0)[3] intercept   : AIC=3849.485, Time=1.23 sec
 ARIMA(2,0,1)(1,1,0)[3] intercept   : AIC=3934.957, Time=0.85 sec
 ARIMA(2,0,1)(2,1,1)[3] intercept   : AIC=inf, Time=3.28 sec
 ARIMA(2,0,1)(1,1,1)[3] intercept   : AIC=inf, Time=1.38 sec
 ARIMA(1,0,1)(2,1,0)[3] intercept   : AIC=3855.484, Time=0.63 sec
 ARIMA(2,0,2)(2,1,0)[3] intercept   : AIC=inf, Time=2.50 sec
 ARIMA(1,0,2)(2,1,0)[3] intercept   : AIC=inf, Time=1.37 sec
 ARIMA(2,0,1)(2,1,0)[3]             : AIC=3849.379, Time=0.37 sec
 ARIMA(2,0,1)(1,1,0)[3]             : AIC=3934.919, Time=0.26 sec
 ARIMA(2,0,1)(2,1,1)[3]             : AIC=inf, Time=3.03 sec
 ARIMA(2,0,1)(1,1,1)[3]             : AIC=inf, Time=1.22 sec
 ARIMA(1,0,1)(2,1,0)[3]             : AIC=3854.837, Time=0.24 sec
 ARIMA(2,0,0)(2,1,0)[3]             : AIC=3852.784, Time=0.24 sec
 ARIMA(2,0,2)(2,1,0)[3]             : AIC=inf, Time=2.45 sec
 ARIMA(1,0,0)(2,1,0)[3]             : AIC=3862.742, Time=0.22 sec
 ARIMA(1,0,2)(2,1,0)[3]             : AIC=inf, Time=1.74 sec

Best model:  ARIMA(2,0,1)(2,1,0)[3]
Total fit time: 29.063 seconds
```

Below is the summary of the model.

```
                                   SARIMAX Results
==========================================================================================
Dep. Variable:                                  y   No. Observations:                 1007
Model:             SARIMAX(2, 0, 1)x(2, 1, [], 3)   Log Likelihood               -1918.689
Date:                            Tue, 18 Jun 2024   AIC                           3849.379
Time:                                    16:54:13   BIC                           3878.849
Sample:                                         0   HQIC                          3860.577
                                           - 1007
Covariance Type:                              opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          1.5449      0.133     11.630      0.000       1.285       1.805
ar.L2         -0.6180      0.111     -5.560      0.000      -0.836      -0.400
ma.L1         -0.6089      0.139     -4.394      0.000      -0.880      -0.337
ar.S.L3       -0.6548      0.029    -22.594      0.000      -0.712      -0.598
ar.S.L6       -0.3174      0.026    -12.160      0.000      -0.369      -0.266
sigma2         2.6703      0.079     33.806      0.000       2.515       2.825
===================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):               360.62
Prob(Q):                              1.00   Prob(JB):                         0.00
Heteroskedasticity (H):               2.04   Skew:                            -0.15
Prob(H) (two-sided):                  0.00   Kurtosis:                         5.92
===================================================================================
```

Model diagnostics interpretation:



1) Standardized residual: It is an error term of price forecasting and actual price of stocks

2) Histogram plus estimated density: Histogram represents normal distribution of errors; KDE plots and N(0,1) is notation of indicate mean is ZERO and variance of the distribution is ONE.

3) Normal Q-Q: Normal Q-Q plot implies normality of distribution as sample quantities mostly in line with theoretical quantities. any deviation in such alignment would indicate distribution is skewed, or in layman terms error is either positive or negative side.

4) Correlogram: It simply indicates partial autocorrelation of time-series and shows which laged time-series is significant in forecasting actual time-series

3.3.4 Prophet

Prophet is a procedure for forecasting time series data based on an additive model where non-linear trends are fit with yearly, weekly, and daily seasonality, plus holiday effects. It works best with time series that have strong seasonal effects and several seasons of historical data. Prophet is robust to missing data and shifts in the trend, and typically handles outliers well.

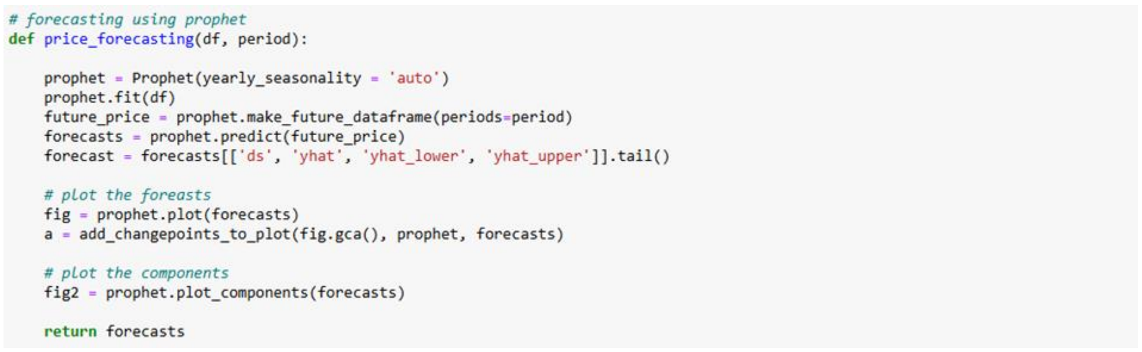
Out-of-Sample Forecast

That is, we can make a forecast on data used as input to train the model. Ideally, the model has seen the data before and would make a perfect prediction.

Nevertheless, this is not the case as the model tries to generalize across all cases in the data.

This is called an out-of-sample forecast. We can achieve this in the same way as an in-sample forecast and simply specify a different forecast period.

A forecast is made by calling the predict() function and passing a DataFrame that contains one column named ‘ds‘ and rows with date-times for all the intervals to be predicted.

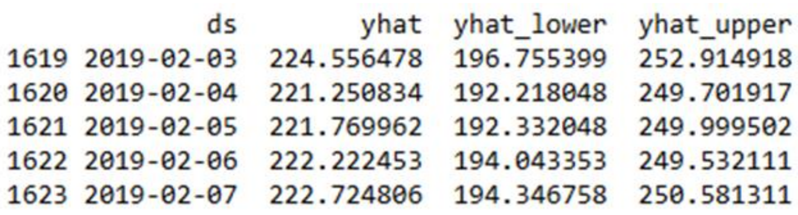


```python
# forecasting using prophet
def price_forecasting(df, period):

    prophet = Prophet(yearly_seasonality = 'auto')
    prophet.fit(df)
    future_price = prophet.make_future_dataframe(periods=period)
    forecasts = prophet.predict(future_price)
    forecast = forecasts[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].tail()

    # plot the foreasts
    fig = prophet.plot(forecasts)
    a = add_changepoints_to_plot(fig.gca(), prophet, forecasts)

    # plot the components
    fig2 = prophet.plot_components(forecasts)

    return forecasts
```

This DataFrame can then be provided to the predict() function to calculate a forecast.

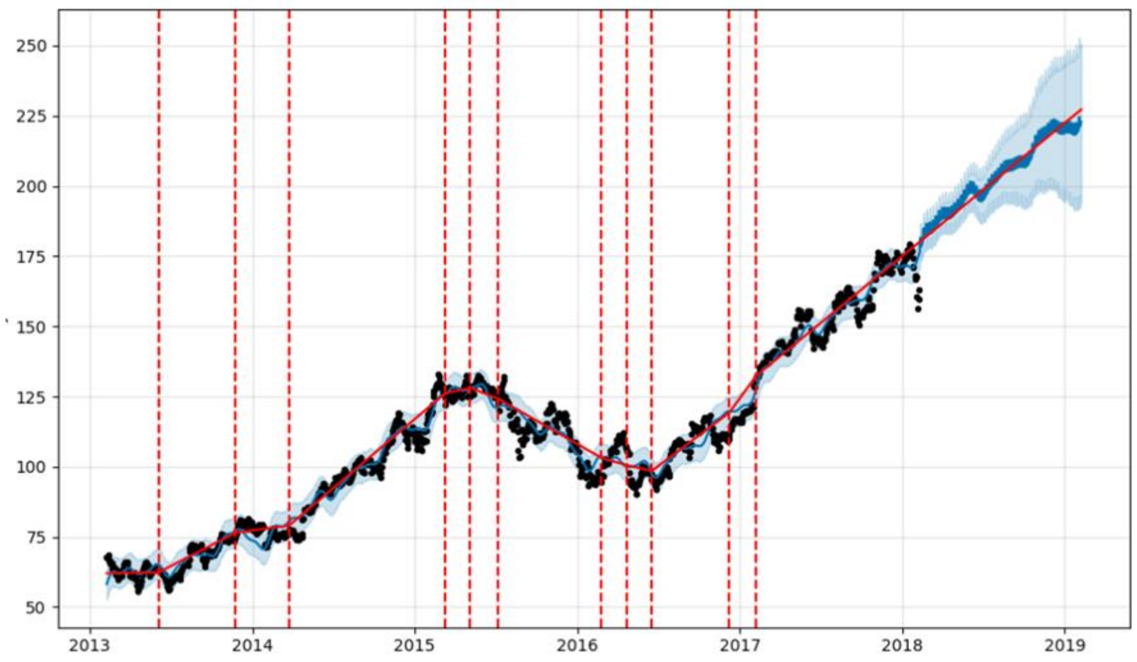
The result of the predict() function is a DataFrame that contains many columns. Perhaps the most important columns are the forecast date time (‘ds‘), the forecasted value (‘yhat‘), and the lower and upper bounds on the predicted value (‘yhat_lower‘ and ‘yhat_upper‘) that provide uncertainty of the forecast.

For example, we can print the first few predictions



```
             ds        yhat  yhat_lower  yhat_upper
1619 2019-02-03  224.556478  196.755399  252.914918
1620 2019-02-04  221.250834  192.218048  249.701917
1621 2019-02-05  221.769962  192.332048  249.999502
1622 2019-02-06  222.222453  194.043353  249.532111
1623 2019-02-07  222.724806  194.346758  250.581311
```

Forecasted results



We can see the training data are represented as black dots and the forecast is a blue line with upper and lower bounds in a blue shaded area.

3.4 Evaluation

3.4.1 ARIMA Model Results

After fitting the ARIMA model and generating forecasts, we compare the forecasted values with the actual test data and calculate the evaluation metrics.

Root Mean Square Error: 27.330504568707298

3.4.2 `Prophet Model Results

Similarly, after fitting the Prophet model and generating forecasts, we compare the forecasted values with the actual test data and calculate the evaluation metrics.

Mean Absolute Error: 95.11716778408054

Root Mean Square Error: 95.42642555183939

CHAPTER 4. Future Work and Improvements

While this project has provided valuable insights into time series analysis and stock price prediction, several areas could be further explored and improved upon in future work:

1. **Expanded Theoretical Research**

- ***In-depth Exploration of Time Series Models***: Due to time constraints, this project has not delved deeply into the theoretical aspects and practical implementations of various time series models. Future work should include comprehensive research on additional models such as Seasonal ARIMA (SARIMA), GARCH (Generalized Autoregressive Conditional Heteroskedasticity), and advanced machine learning approaches like Long Short-Term Memory (LSTM) networks.
- ***Model Comparison and Evaluation***: A systematic approach to comparing different models using various metrics (e.g., AIC, BIC, MAPE, MAE) would provide a clearer understanding of each model's performance. Future studies should also explore ensemble methods that combine multiple models to improve forecast accuracy.

2. Enhanced Practical Implementation

- ***Automated Data Pipeline***: Building a real-time application that can automatically crawl stock data from the web, preprocess it, and update the analysis regularly would greatly enhance the practical utility of this project. This could involve using web scraping tools or APIs provided by financial data services. This will be a tracking system within the application to monitor stock prices continuously and generate interactive visualizations and forecasts for users.
- ***Interactive Dashboards***: Developing interactive dashboards using tools could make the results more accessible and user-friendly. These dashboards could provide real-time updates, visualizations, and user interaction capabilities.

3. Handling of Anomalies and Noise

- ***Implement more sophisticated*** methods for detecting and handling outliers and noise in the data. Techniques such as robust statistical methods or anomaly detection algorithms can be explored.
- ***Assess the impact of different data preprocessing*** methods on model performance and establish best practices for preparing time series data.

4. Scalability and Performance

- ***Optimize the application for scalability*** to handle large volumes of data and multiple users simultaneously.
- Ensure that the system's performance remains robust even as the dataset grows, and the complexity of the models increases.

By addressing these areas, future iterations of this project can achieve a more comprehensive and practical implementation of time series analysis for stock price prediction. These improvements will not only enhance the accuracy and reliability of the models but also provide users with valuable tools for making informed investment decisions.

CHAPTER 5. Reference:

https://medium.com/@ritusantra/stationarity-in-time-series-887eb42f62a9

https://medium.com/codex/what-is-stationarity-in-time-series-how-it-can-be-detected-7e5dfa7b5f6b

https://www.analyticsvidhya.com/blog/2015/12/complete-tutorial-time-series-modeling/

https://prof-frenzel.medium.com/kb-time-series-data-part-3-6e32032f7b49

https://www.geeksforgeeks.org/time-series-decomposition-techniques/

https://www.baeldung.com/cs/acf-pacf-plots-arma-modeling

https://phamdinhkhanh.github.io/2019/12/12/ARIMAmodel.html