Indian Institute of Technology, Bombay

Project Report

TOPIC: TIME SERIES ANALYSIS AND FORECASTING

DATASET: AMAZON STOCK PRICE

Supervised by: **Dr. Sanjeev Sabnis**

Submitted by: Subhojit Kayal (205280022)

Summary of this Project

In the following project the main task was to analyse the capabilities of ARIMA models to provide accurate forecasts of values of stock indexes and stock prices. It was discovered that ARIMA models are better suited for short-term forecasts of stock indexes while these models give on average less precise forecasting results for individual stocks.

Moreover, it was found that an appropriate model for stock price forecasting is Triple Exponential Smoothing with the help of Holt- Winter's Trend and Seasonality Method for Additive Model and Box-Cox Transformation. In addition, the conclusion was made that a one year time series is sufficient to provide forecasts for up to three days ahead while a five year time series can be considered for longer term predictions.

Table of contents

Chapter I. Introduction:

Forecasting stock prices has been always a fascinated topic in finance, drawing attention of leading economists and investors throughout the world. These subject gains popularity due to the fact that all investment decisions are based on anticipations of positive future outcomes, therefore correct predictions of investment results allow investors to select profitable stocks and apply right timing strategies. However, in the stock market there are many interrelated factors affecting stock prices, which make forecasting a very complicated task. Moreover, Fama French (1965) in his study "The behavior of stock market prices" suggested that stock prices move in a random and unpredictable manner in the efficient market resulting in impossibility of consistent forecasting. Nevertheless, there are researchers and investment professionals, who are skeptical towards the efficient market hypothesis and believe in the possibility to create forecasting models that allow stock prices to be predicted with high accuracy.

All contemporary stock price forecasting approaches can be broadly classified in three groups: **fundamental analysis, technical analysis** and **time series forecasting.** (Tsang et al., 2007)

The rationale behind the Fundamental analysis states that the stock price depends on its intrinsic value and expected return. These two components can be found by analyzing the company`s financials and the market where the company operates. Fundamental analysis is regarded as an appropriate forecasting method for long-term investments but not for short-term speculations. In addition, interpretations made from results of the fundamental analysis are considered to be subjective. (Tsang et al., 2007)

Technical analysis uses past price and other statistical information to make stock price predictions. Proponents of technical analysis believe that historical information contains patterns that can explain future price movements. Moreover, most of the technical analysis methods are regarded as highly subjective and statistically invalid. (Tsang et al., 2007)

Fundamental and technical analyses have one common feature –interpretations of their results are usually subjective points of view. Therefore, it is interesting to find a method that allows accurately predicting stock prices with unbiased conclusions over the final results. Time series method gained popularity over its statistical approach in forecasting that avoids subjectivity. A time series is a set that includes observations of one or more variables over time and is arranged in chronological order. The forecast is conducted by identifying and examining the dynamics of the data. (Asteriou and Hall, 2011) Applying time series technics allows modeling historical price information as a function that possesses a recurrence relation. This relation is used to forecast future values. Time series approach is appropriate for short-term forecasting, usually up to a year, but it requires a considerable amount of precise information. (Tsang et al., 2007)

Time series forecasting methods produce forecasts based solely on historical values and they are widely used in business situations where forecasts of a year or less are required. These methods used are particularly suited to Sales, Marketing, Finance, Production planning etc. and they have the advantage of relative simplicity. Time series forecasting is a technique for the prediction of events through a sequence of time.

The technique is used across many fields of study, from geology to economics. The techniques predict future events by analyzing the trends of the past, on the assumption that the future trends will hold similar to historical trends. Data is organized around relatively deterministic timestamps, and therefore, compared to random samples, may contain additional information that is tried to extract.

- Time series methods are better suited for short-term forecasts (i.e., less than a year).
- Time series forecasting relies on sufficient past data being available and that the data is of a high quality and truly representative.
- Time series methods are best suited to relatively stable situations. Where substantial fluctuations are common and underlying conditions are subject to extreme change, then time series methods may give relatively poor results.

Chapter II. Literature review:

There are many algorithms of forecasting stock prices using a time series. The most popular methods are Autoregressive model (AR), Moving Average model (MA), Autoregressive Moving Average (ARMA) and Autoregressive Moving Integrated Average (ARIMA). (Yi Zuo, 2011)

AR model bases its predictions on the historical stock prices, as against to MA model that riles on historical error terms. ARMA is a combination of the previous two models. It predicts future values according to the linear relationship of the past error terms and stock prices.(Yi Zuo, 2011) ARIMA model is essentially an improvement of the ARMA technic.

The focus of this study is to test the viability of ARIMA model that is considered by some researchers to be a superior model in short-term predictions.

ARIMA model is a development of ARMA in a way that the former solves the problem of non-stationarity, which leads to invalid results of the regression analysis. ARIMA model was introduced by George Box and Gwilym Jenkins in 1970. (Box, Jenkins and Reinsel, 2013) According to some researchers, ARIMA model is one of the most popular and widely-used methods in time series forecasting and was applied in various economic, ecological and engineering spheres. The model proved to be efficient in making short-term predictions. In addition, despite being relatively straightforward in applying, ARIMA model outperforms complex structural models in short-term periods. (Meyler, Kenny and Quinn, 1998)

The aim of this paper is to test the ability of ARIMA model to accurately predict stock prices and indices in the Amazon stock market.

The ARIMA model is classified as ARIMA (p, d, q), where p is related to the autoregressive (AR) part of the model and represents the number of lags of the dependent variable, d refers to the integrated part (I) and shows the number of differences that should be taken to meet the stationary requirement, and q denotes moving average part (MA) of the time series which indicates the number of lagged terms of the error term. Values of p, d, q should always be non-negative. According to Box and Jenkins, the values of p and q should not exceed 2.

In order to obtain the most accurate results from stock forecasting the appropriate model should be selected using the Box-Jenkins approach. Moreover, the model should include parameters with the smallest values. Since the direct forecasting ignores this procedure, it considered to be inferior to ARIMA.

Apart from the three major forecasting approaches described above, there were several other technics developed in the past years. Two of the most prominent methods are artificial neural networks model (ANNs) and hybrid method. ANNs relates to the artificial intelligence approach and finds unknown variables using patterns from the available information.Hybrids methods exploit strengths of other forecasting models to improve predictions. (Wang et al., 2012)

The past studies also classify forecasting models according to their prospective: statistical and artificial intelligence approaches. ARIMA model relates to the statistical prospective. ARIMA model is considered to be efficient and dominant in time series forecasting. Many researchers showed that ARIMA technic performs short-term predictions better than ANNs models.

Ayodele A. and Adebiyi (2014) in their study demonstrated the ability of ARIMA model to provide relatively accurate short-term predictions about stock prices.

The ARIMA time series method showed more accurate forecasting result for the amount of Taiwan export, compared to the fuzzy time series. However, such results require a bigger data sample than fuzzy time series does. (Wang, 2011)

The time series methods can be divided in two types. First, univariate methods are applied using only a time series of the examined variable in contrast to another one, multivariate method, which in addition requires time series of related variables. The main advantage of ARIMA model being a univariate method is that this model requires less data than a multivariable approach. This feature makes ARIMA model convenient in forecasting stock prices of many stocks. One time series also allows avoiding problem of inconsistent data that multivariate models may suffer, if the available time lengths of time series are not matched or have missing observations.

Chapter III. Methodology:

As it was discussed in the literature review, ARIMA model has gained popularity among researchers who consider it to be superior in providing short-term forecast. (Adebiyi and O. Adewumi, 2014) Therefore, ARIMA model was chosen to be the main forecasting instrument in this project.

Future values in the ARIMA model are projected using a linear combination of historical error terms and values. The formula is as follows:

$$Y_t = \phi_0 + \phi_1 Y_{t-1} + \phi_2 Y_{t-2} + \cdots + \phi_p Y_{t-p} + u_t - \Theta_1 u_{t-1} + \Theta_2 u_{t-2} + \cdots + \Theta_q u_{t-q}$$

, where

Y_t is an actual value, u_t- standard error, Θ and ϕ are coefficients, p and q are integers. (Ayodele A. Adebiyi, 2014)

Box and Jenkins created a three-stage method for the model selection. These stages are: identification, estimation, diagnostic checking. (Box, Jenkins and Reinsel, 2013)

Step 1: Model identification.

At this stage the time series is being checked for stationarity by examining the correlograms of autocorrelation function (ACF) and partial autocorrelation function (PACF). (Lee and Ko, 2011) Stationarity means that the mean, variance and covariance of a time series are all constant over time. However, most economic time series have trends, thus having different means over time. (Asteriou and Hall, 2011) The series is stationary if its values on the ACF graph either cut off quickly or die down quickly. If ACF graph dies down very slowly, then the time series is considered to be non-stationary. (Nochai R. and Nochai T, 2006)

If the time series data appears to be non-stationary, it should be transformed into a stationary one by using the appropriate number of differencing. Differencing is applied as many times as it is needed to meet stationary requirement. (Lee and Ko, 2011) Differencing serves the role of de-trending a time series data. (Asteriou and Hall, 2011) The formula of the first differences is as follows:

$$\Delta Y_t = Y_t - Y_{t-1}$$

Where, Y_t is a value of a single observation from the stationary time series at the time t.

If after first differencing a time series is non-stationary, second differencing should be applied by using the formula below:

$$\Delta\Delta Y_t = \Delta Y_t - \Delta Y_{t-1}$$; (Asteriou and Hall, 2011)

Correlograms are the values of the ACFs and PACFs plotted against lag lengths and are used to check a series for stationarity. The autocorrelation coefficient (ACF) estimates the correlation between a set of observations and a lagged set of observations in a time series. (Wasseja and Mwenda, 2015)

The formula for a sample autocorrelation coefficient is as follows:

$$r_k = \frac{\sum(Y_t - \hat{Y})(Y_{t+k} - \hat{Y})}{\sum(Y_t - \hat{Y})\,(Y_t - \hat{Y})}$$

Where, Y_t is a value of a single observation from the stationary time series. Y_{t+k} is the data from the period t+k . $\hat{Y}$ is the mean of the stationary time series. (Wasseja and Mwenda, 2015)

Partial autocorrelation function measures how Y_t and Y_{t+k} are related and used together with ACF as a guide in selecting an appropriate ARIMA model. (Wasseja and Mwenda, 2015)

Once series becomes stationary, the next task is to determine the p and q orders of the model. MA(q) process is used in order to find the value of q. According to it, the value of q is equal to the last lag in the ACF, where estimates are statistically significant. After that point estimates begin to die down immediately. The PACF for MA (q) process tends to die down quickly. In AR(p) process the value of p is defined as the last estimate that shows spikes in the PACF. In the pure AR(q) process an ACF dies down quickly. If neither the ACF nor the PACF functions indicate the number of orders, a combined process is applied. (Asteriou and Hall, 2011)

Finally, one or a few tentative models should be selected according to statistics, ACF and partial autocorrelation function (PACF). (Lee and Ko, 2011)

Step 2: Parameter estimation

After selecting tentative models, the models` parameters should be estimated using the least squares method. The parameters are calculated in a way to have zero gradient of forecasting errors to the historical data. At this stage the prime task is to minimize the error from forecasting and define model`s parameters and order. (Lee and Ko, 2011)

After estimating tentative models, their coefficients are compared. The statistical measures that are applied to select the best fitted model are the Akaike Information Criteria (AIC) and the Bayesian Information Criterion (BIC). The model with the lowest AIC and/or BIC should be chosen.

Step 3: Diagnostic checking.

After parameters` estimation, the tentative model`s accuracy is checked by studying the ACF and PACF residuals. The residuals should follow the white noise process. (Lee and Ko, 2011) Superfluous coefficients should not be added to the appropriate model, otherwise the model is overfitted. (Asteriou and Hall, 2011)

Afterwards, the Q-statistic is used to approve the tentative model. (O'Donovan, 1983) If the estimated value Q exceeds the critical value of χ^2 derived from the chi-square table, the tentative model is inadequate. (Lee and Ko, 2011) This statistic tests the model for autocorrelation of the residuals. (Asteriou and Hall, 2011)

If a tentative model is inadequate, the whole process should be repeated until an adequate one is identified. When the Box-Jenkings procedure is completed, the selected ARIMA model is used to forecast the future values usually over 24 hours ahead. (Lee and Ko, 2011)

Chapter IV. Data collection, Analysis & Forecasting:

In this study ARIMA model was tested on its capacity to accurately predict stock prices in the Amazon Stock Price data taken from Yahoo Finance. There are 6 attributes in the data

- High: Highest price of the stock that particular date.
- Low: Lowest price of the stock for that particular date.
- Open: Opening price of the stock for that particular date.
- Close: Closing price of the stock of that particular date.
- Volume: Total amount of Trading Activity.
- Adj. Close: Adjusted values factor in corporate actions such as dividends, stock splits and new share insurance.

The project data comprised daily closing values of FTSE All-Share Index and individual stocks. The index forecast was intended to show the ability of ARIMA model to predict values that did not contain unpredictable company specific risk. FTSE All-Share Index represents all companies listed in London Stock Exchange; therefore, this index is a benchmark of the British broad stock market including corporations with large capitalization as well as small cap companies.

Furthermore, after testing the model`s capacity to perform precise predictions in the market with virtually zero specific risk, the model was applied to forecast closing stock prices

Back testing approach applied in this research included two periods with different lengths: one and five years. Historical values of the stocks and the index were derived from the Yahoo database.

The one and the five-year sample periods had starting dates from 23.11.2015 to the end date 20.11.2020.

To observe the nature of the closing stock values of the data, we have plotted the Target variable closing stock value with respect to the index value.

Amazon Stock price

The Stock Price for 1825 days are given in this dataset, starting from 23rd November 2015 to 20th November 2020. This is a real time dataset which is taken from Yahoo Finance Official Website.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
import datetime
from datetime import datetime
import os
import math
import warnings
warnings.filterwarnings("ignore")


from statsmodels.tsa.statespace.sarimax import SARIMAX
from sklearn import metrics
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from pmdarima.arima.utils import ndiffs
from statsmodels.tsa.arima_model import ARIMA
from pmdarima.arima import auto_arima
from sklearn.metrics import mean_squared_error, mean_absolute_error
from pandas.plotting import lag_plot
from pmdarima.arima import ADFTest
from statsmodels.tsa.holtwinters import Holt,SimpleExpSmoothing,Exponent
```

In [2]:
```
# For graphing purpose, can change
plt.style.use('seaborn-bright')
plt.rcParams.update({'figure.figsize': (10, 6)})
matplotlib.rcParams['axes.labelsize'] = 14
matplotlib.rcParams['xtick.labelsize'] = 12
matplotlib.rcParams['ytick.labelsize'] = 12
matplotlib.rcParams['text.color'] = 'k'
```

In [3]:
```
df= pd.read_csv("D:/yahoo finance stock market/stock_market_yahoo_financ
df['Date'] = pd.to_datetime(df['Date'])
# Set the date as index
df = df.set_index('Date')
```

In [4]:
```
df.head(500)
```

Out[4]:

Date	High	Low	Open	Close	Volume	Adj Close
2015-11-23	2095.610107	2081.389893	2089.409912	2086.590088	3.587980e+09	2086.590088

2015-11-24	2094.120117	2070.290039	2084.419922	2089.139893	3.884930e+09	2089.139893
2015-11-25	2093.000000	2086.300049	2089.300049	2088.870117	2.852940e+09	2088.870117
2015-11-26	2093.000000	2086.300049	2089.300049	2088.870117	2.852940e+09	2088.870117
2015-11-27	2093.290039	2084.129883	2088.820068	2090.110107	1.466840e+09	2090.110107
...	...	...	...	...	...	...
2017-04-01	2370.350098	2362.600098	2364.820068	2362.719971	3.354110e+09	2362.719971
2017-04-02	2370.350098	2362.600098	2364.820068	2362.719971	3.354110e+09	2362.719971
2017-04-03	2365.870117	2344.729980	2362.340088	2358.840088	3.416400e+09	2358.840088
2017-04-04	2360.530029	2350.719971	2354.760010	2360.159912	3.206240e+09	2360.159912
2017-04-05	2378.360107	2350.520020	2366.590088	2352.949951	3.770520e+09	2352.949951

500 rows × 6 columns

In [5]:

```
df.isnull().sum()
```

Out[5]:

```
High         0
Low          0
Open         0
Close        0
Volume       0
Adj Close    0
dtype: int64
```

No Null values, complete Dataset

In [6]:

```
df.describe()
```

Out[6]:

	High	Low	Open	Close	Volume	Adj Close
count	1825.000000	1825.000000	1825.000000	1825.000000	1.825000e+03	1825.000000
mean	2660.718673	2632.817580	2647.704751	2647.856284	3.869627e+09	2647.856284
std	409.680853	404.310068	407.169994	407.301177	1.087593e+09	407.301177
min	1847.000000	1810.099976	1833.400024	1829.079956	1.296540e+09	1829.079956
25%	2348.350098	2322.250000	2341.979980	2328.949951	3.257950e+09	2328.949951
50%	2696.250000	2667.840088	2685.489990	2683.340088	3.609740e+09	2683.340088
75%	2930.790039	2900.709961	2913.860107	2917.520020	4.142850e+09	2917.520020
max	3645.989990	3600.159912	3612.090088	3626.909912	9.044690e+09	3626.909912

There are six columns given:

High -> Highest Price of the stock for that particular date.

Low -> Lowest Price of the stock for that particular date.

Open -> Opening Price of the stock.

Close -> Closing Price of the stock.

Volume -> Total amount of Trading Activity.

AdjClose -> Adjusted values factor in corporate actions such as dividends, stock splits, and new share issuance.

In [7]:
```
df.shape
```

Out[7]:
```
(1825, 6)
```

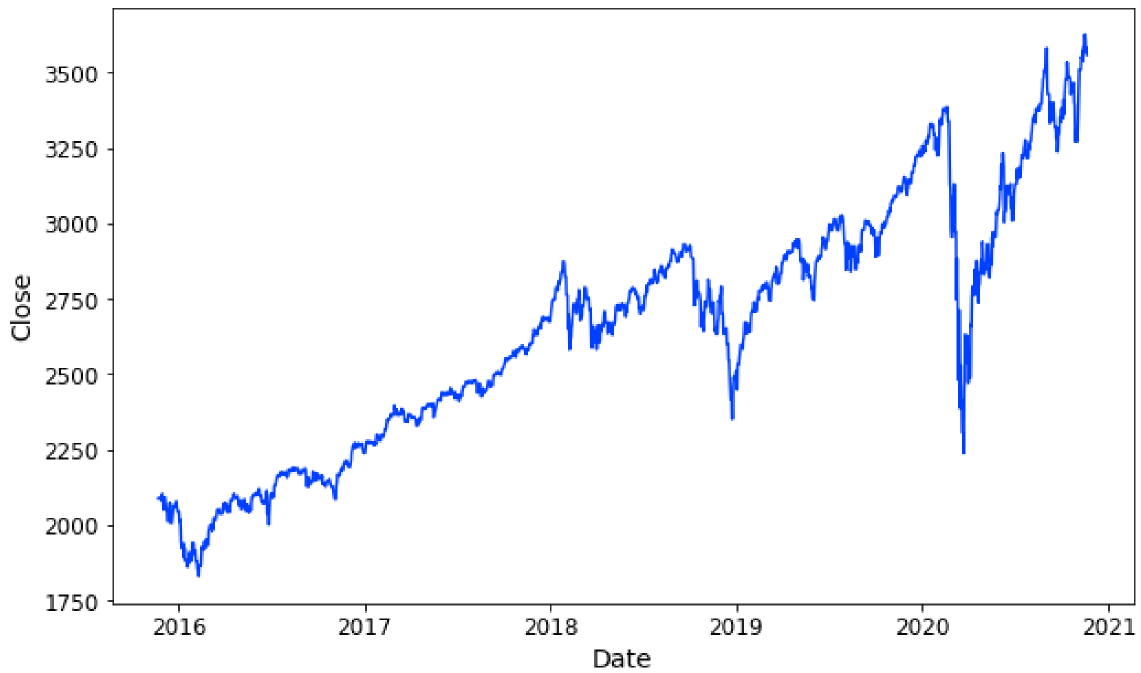
In [8]:
```
print(df.index.min())
print(df.index.max())
```

```
2015-11-23 00:00:00
2020-11-20 00:00:00
```

plotting the data

In [9]:
```
plt.plot(df["Close"])
plt.xlabel("Date")
plt.ylabel("Close")
```

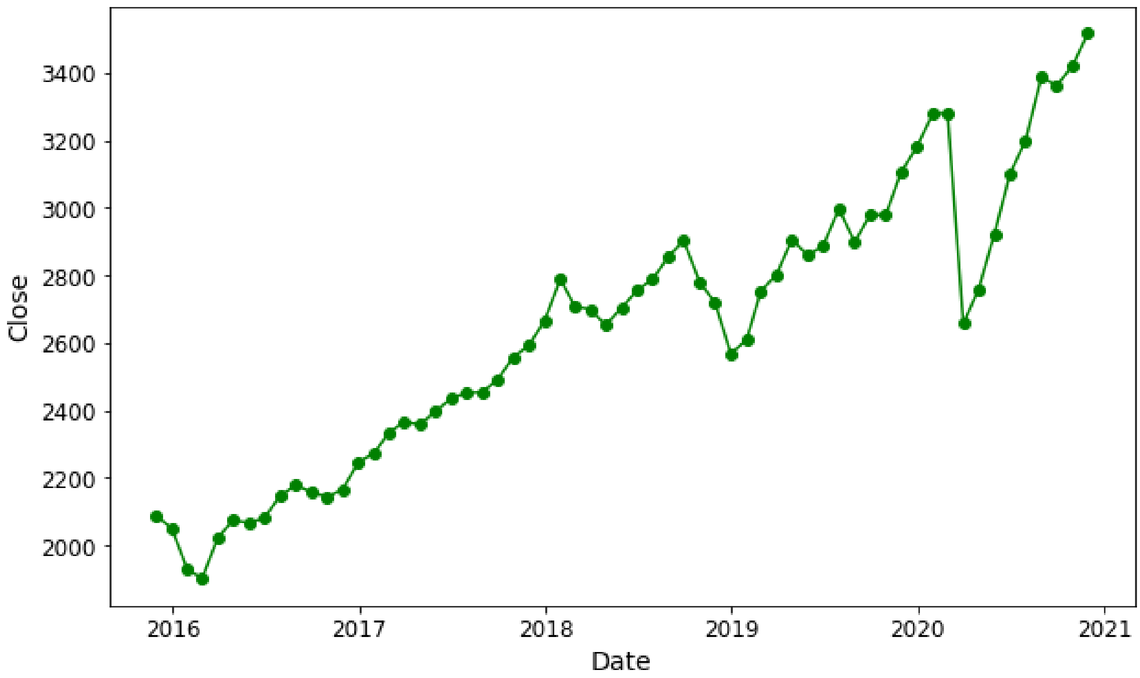
Out[9]:
```
Text(0, 0.5, 'Close')
```



Plotting the data after downsampling using mean

In [10]:
```
plt.plot(df["Close"].resample("M").mean(), color='g', marker='o')
plt.xlabel("Date")
plt.ylabel("Close")
```

Out[10]:
```
Text(0, 0.5, 'Close')
```



We can observe that there are no huge variations in the opening-closing price and the high-low prices & there is a upward trend with respect to Time.

```
There were huge dips in the stock prices 2 times, once
close to 2019 and once in March 2020(owing to Pandemic).

There was an overall increase in the stock price from 2017
to 2018.

The stock prices started to increase from the latter half
for the year 2020

The stock price went drastically down from starting of 2018
to 2019
```

Decomposition Implementation

A given time series is thought to consist of three systematic components including level, trend, seasonality, and one non-systematic component called noise.

These components are defined as follows: Level: The average value in the series. Trend: The increasing or decreasing value in the series. Seaso# nality: The repeating short-term cycle in the series. Noise: The random variation in the series.

All series have a level and noise. The trend and seasonality components are optional. It is helpful to think of the components as combining either additively or multiplicatively.

An additive model suggests that the components are added together as follows:

```
y(t) = Level + Trend + Seasonality + Noise
```

An additive model is linear where changes over time are consistently made by the same amount. A linear seasonality has the same frequency (width of cycles) and amplitude (height of cycles).

A multiplicative model suggests that the components are multiplied together as follows:

```
y(t) = Level * Trend * Seasonality * Noise
```

A multiplicative model is nonlinear, such as quadratic or exponential. Changes increase or decrease over time. A non-linear seasonality has an increasing or decreasing frequency and/or amplitude over time.

Decomposition provides a structured way of thinking about a time series forecasting problem, both generally in terms of modeling complexity and specifically in terms of how to best capture each of these components in a given model.

Each of these components are something you may need to think about and address during data preparation, model selection, and model tuning. You may address it explicitly in terms of modeling the trend and subtracting it from your data, or implicitly by providing enough history for an algorithm to model a trend if it may exist.

In order to implement the naive or classical decomposition method, we use the seasonal_decompose() method provided by the statsmodels library. It requires you to specify whether the model is Additive or Multiplicative.

In [11]:

```
df_new=df.drop(['High','Low','Open','Adj Close','Volume'],axis=1)
df_new.head()
```

Out[11]:

	Close
Date	
2015-11-23	2086.590088
2015-11-24	2089.139893
2015-11-25	2088.870117
2015-11-26	2088.870117
2015-11-27	2090.110107

In [12]:

```
df_new_close= df_new['Close']
```

In [13]:

```
df_close_month = df_new.resample('MS').mean()
df_close_month.head(10)
```

Out[13]:

	Close
Date	
2015-11-01	2088.026306
2015-12-01	2051.352913

2016-01-01	1927.887408
2016-02-01	1902.567938
2016-03-01	2023.688059
2016-04-01	2074.564001
2016-05-01	2066.167102
2016-06-01	2081.775667
2016-07-01	2147.336434
2016-08-01	2178.120983

In [14]:

```
df_close_month_1= df_close_month["Close"]
```

In [15]:

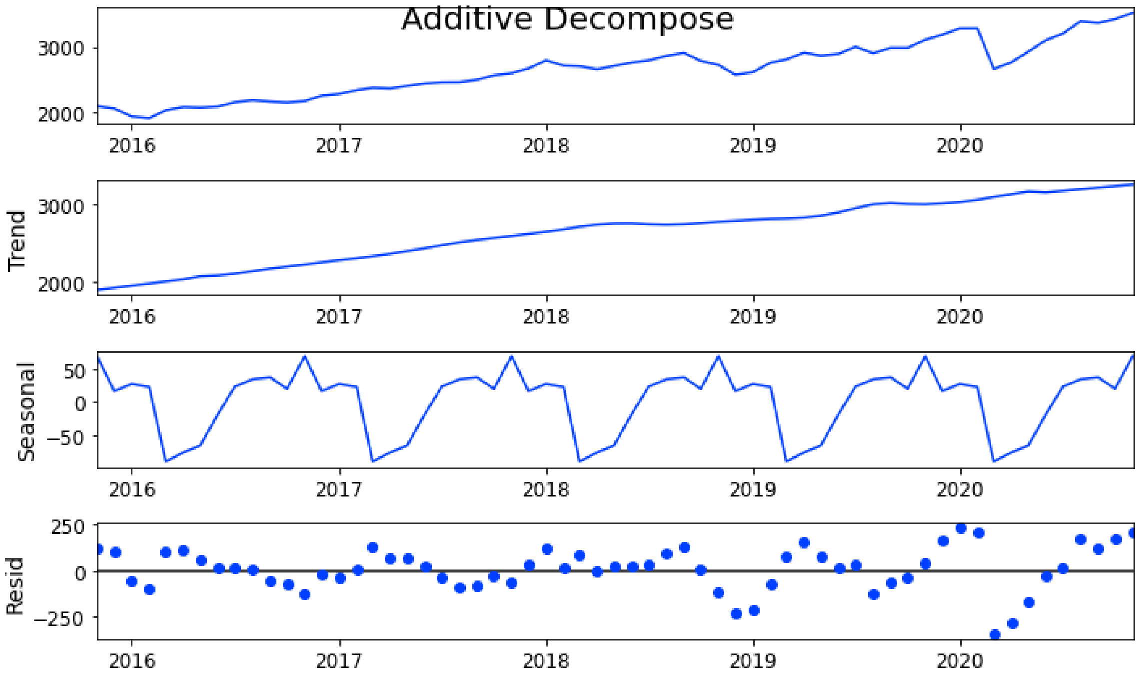
```
def decompose(df):
    """
    A function that returns the trend, seasonality and residual captured
    additive model."""
    result_additive = seasonal_decompose(df, model = 'add', extrapolate_

    plt.rcParams.update({'figure.figsize': (10, 6)})
    result_additive.plot().suptitle('Additive Decompose', fontsize=20)
    plt.show()

    return result_additive
```

The seasonal_decompose() function returns a result object. The result object contains arrays to access four pieces of data from the decomposition: Observed Series, Trend, Seasonality, and residual. We have plotted both Multiplicative as well as Additive model, so that we can decide which one of the two should be used.

Close

In [16]:

```
result_additive_close = decompose(df_close_month)
```



In [17]:

```
df_reconstructed_close= pd.concat([result_additive_close.seasonal, resul
df_reconstructed_close.columns= ['Seasonal','Trend','Residual','Actual_v
df_reconstructed_close
```

Out[17]:

Date	Seasonal	Trend	Residual	Actual_values
2015-11-01	67.952610	1901.021995	119.051701	2088.026306
2015-12-01	16.263919	1927.898884	107.190110	2051.352913
2016-01-01	26.831742	1954.775774	-53.720108	1927.887408
2016-02-01	22.268022	1981.652663	-101.352747	1902.567938
2016-03-01	-88.556289	2008.529553	103.714795	2023.688059
...	...	...	...	...
2020-07-01	22.890054	3163.201147	14.181378	3200.272579
2020-08-01	33.648586	3183.032445	171.278670	3387.959701
2020-09-01	36.658951	3202.863743	122.961632	3362.484326
2020-10-01	19.633700	3222.695041	178.022207	3420.350948
2020-11-01	67.952610	3242.526339	206.339019	3516.817969

61 rows × 4 columns

Stationarity

```
Subtract the previous value from the current value. Now if
we just difference once, we might not get the a stationary
series ; we might need to do that multiple times.
The minimum number of differencing operations needed to
make the stationary needs to be inputed into the ARIMA
model.
```

ADF Test

```
The Dickey-Fuller test is one of the most popular
statistical tests. It can be used to determine the presence
of unit root in the series, and hence help us understand if
the series is stationary or not. The null and alternate
hypothesis of this test is:
```

- Null Hypothesis: The series has a unit root (value of a =1)
- Alternate Hypothesis: The series has no unit root.

If we fail to reject the null hypothesis, we can say that the series is non-stationary. This means that the series can be linear or difference stationary.

We'll use the Augmented Dickey Fuller Test to check if the stock price series is stationary or not.

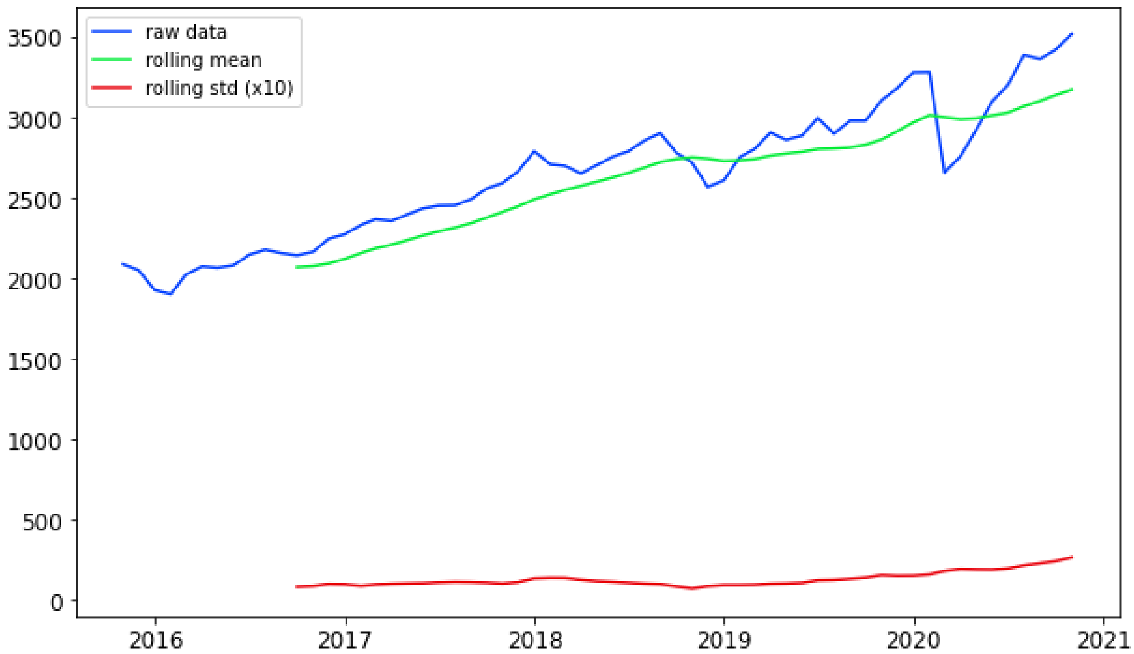
So, if the p-value of the test is less than the significance level(0.05) then we can reject the null hypothesis and infer that the time series model is indeed stationary. If the p-value is greater than 0.05 then we'll need to find the order of differencing.

In [18]:

```
### plot for Rolling Statistic for testing Stationarity
def test_stationarity(timeseries, title):

    #Determing rolling statistics
    rolmean = pd.Series(timeseries).rolling(window=12).mean()
    rolstd = pd.Series(timeseries).rolling(window=12).std()

    fig, ax = plt.subplots(figsize=(10, 6))
    ax.plot(timeseries, label= title)
    ax.plot(rolmean, label='rolling mean');
    ax.plot(rolstd, label='rolling std (x10)');
    ax.legend()
```

In [19]:

```
pd.options.display.float_format = '{:.8f}'.format
test_stationarity(df_close_month_1,'raw data')
```



Augmented Dickey-Fuller Test for checking the Stationarity

In [20]:

```
def ADF_test(timeseries, dataDesc):
    print(' > Is the {} stationary ?'.format(dataDesc))
    dftest = adfuller(timeseries.dropna(), autolag='AIC')
    print('Test statistic = {:.3f}'.format(dftest[0]))
    print('P-value = {:.3f}'.format(dftest[1]))
    print('Critical values :')
    for k, v in dftest[4].items():
        print('\t{}: {} - The data is {} stationary with {}% confidence'
```

In [21]:

```
ADF_test(df_close_month_1,'raw data')
```

```
 > Is the raw data stationary ?
Test statistic = -0.397
P-value = 0.911
Critical values :
        1%: -3.5443688564814813 - The data is not stationary with 99% con
fidence
        5%: -2.9110731481481484 - The data is not stationary with 95% con
fidence
        10%: -2.5931902777777776 - The data is not stationary with 90% co
nfidence
```

Through the above graph, we can see the increasing mean and standard deviation and hence our series is not stationary.

The p-value is obtained is greater than significance level of 0.05 and the ADF statistic is higher than any of the critical values.

Clearly, there is no reason to reject the null hypothesis. So, the time series is in fact non-stationary.

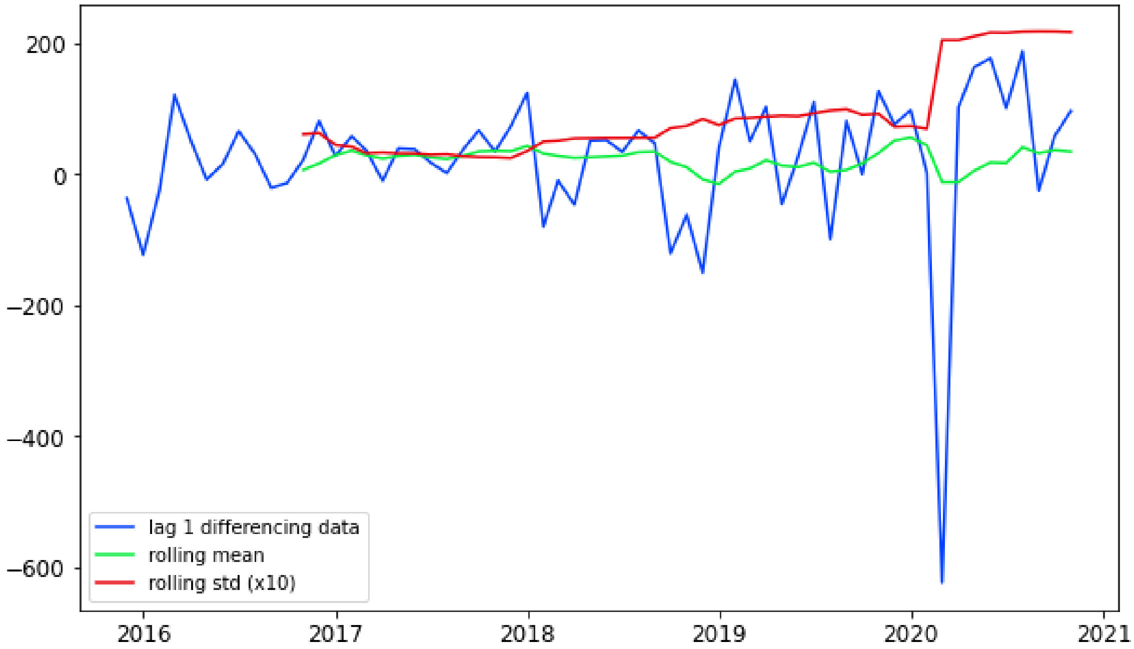
To make the Time series stationary, "Differencing once"

In [22]:

```
df_close_adj = df_close_month_1 - df_close_month_1.shift(1)
```

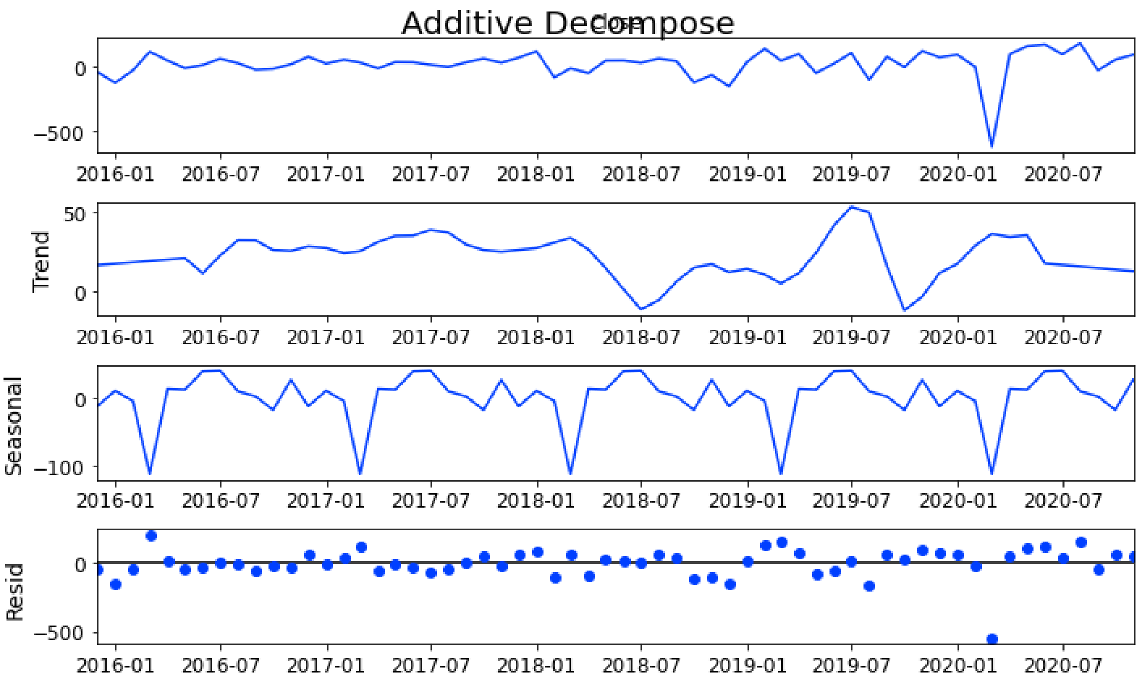
```
df_close_adj = df_close_adj.dropna()
test_stationarity(df_close_adj,'lag 1 differencing data')
ADF_test(df_close_adj,'lag 1 differencing data')
```

```
 > Is the lag 1 differencing data stationary ?
Test statistic = -7.144
P-value = 0.000
Critical values :
        1%: -3.5463945337644063 - The data is  stationary with 99% confid
ence
        5%: -2.911939409384601 - The data is  stationary with 95% confide
nce
        10%: -2.5936515282964665 - The data is  stationary with 90% confi
dence
```



In [23]:

```
decompose(df_close_adj)
```

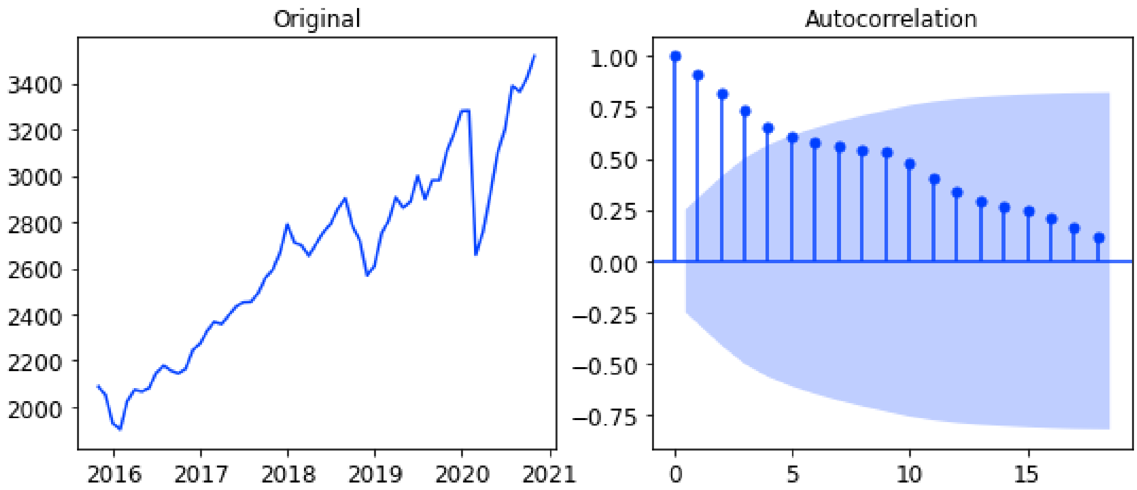


Out[23]: <statsmodels.tsa.seasonal.DecomposeResult at 0x221ff1b8190>

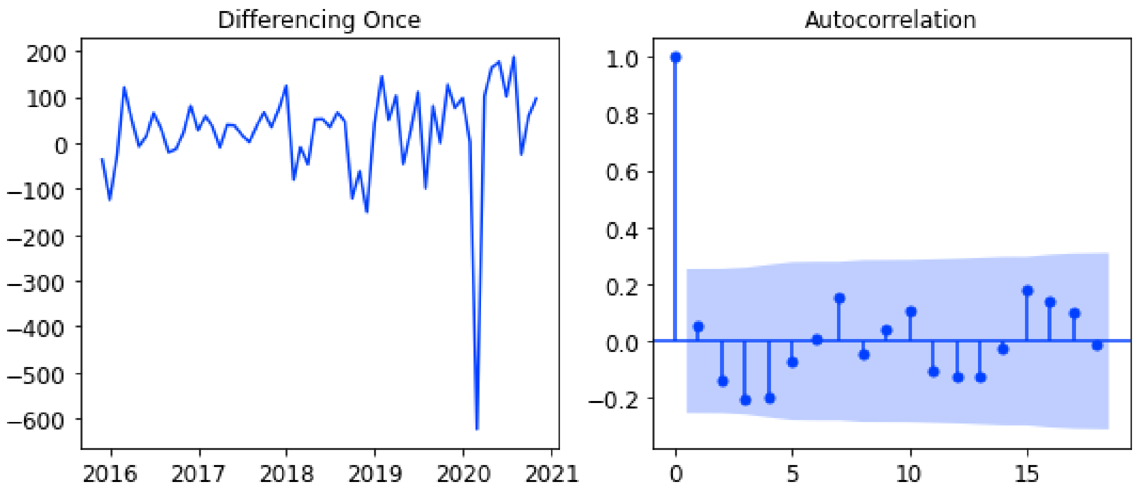
After differencing once, we get the p-value which is less than the significance level(0.05) and the ADF statistic is lower than any of the critical values.

Autocorrelation function

In [24]:
```
fig, (axis_1 , axis_2) = plt.subplots(1,2, figsize=(10,4))
axis_1.plot(df_close_month)
axis_1.set_title("Original")
plot_acf(df_close_month, ax=axis_2);
```



In [25]:
```
diff= df_close_month.diff().dropna()
fig, (axis_1 , axis_2) = plt.subplots(1,2, figsize=(10,4))
axis_1.plot(diff)
axis_1.set_title("Differencing Once")
plot_acf(diff, ax=axis_2);
```



Therefore, first order differencing is enough for our model. Hence, d is taken as "one"

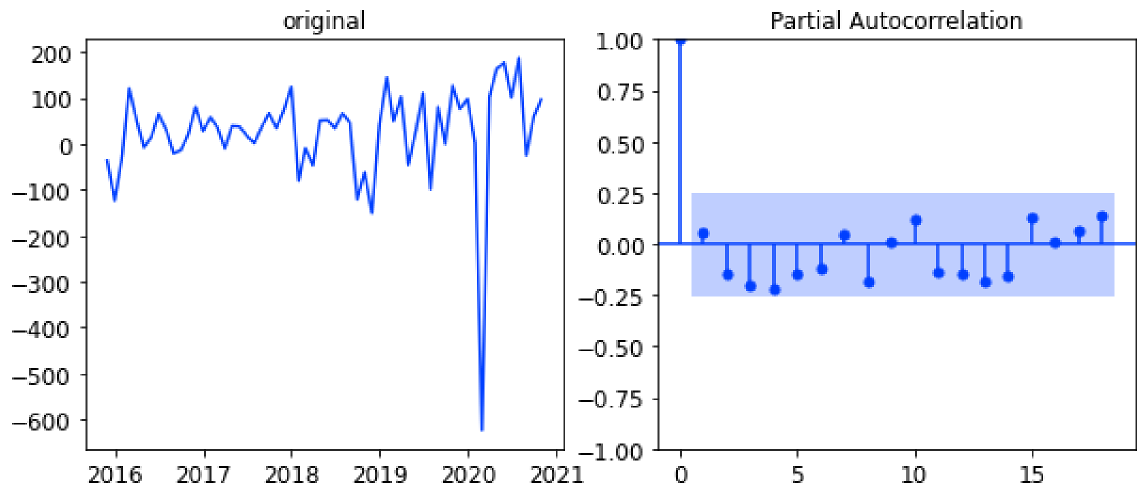
P

```
p is the order of the Auto Regressive(AR) term. It refers
to the number of lags to be used as Predictors.
We can find out required number of AR terms by inspecting
the Partial Autocorrelation(PACF) plot
```

```
The partial autocorrelation represents the correlation
between the series and its lags.
```

In [26]:

```
diff= df_close_month.diff().dropna()
fig, (axis_1 , axis_2) = plt.subplots(1,2, figsize=(10,4))
axis_1.plot(diff)
axis_1.set_title("original")
axis_2.set_ylim(-1,1)
plot_pacf(diff, ax=axis_2);
```



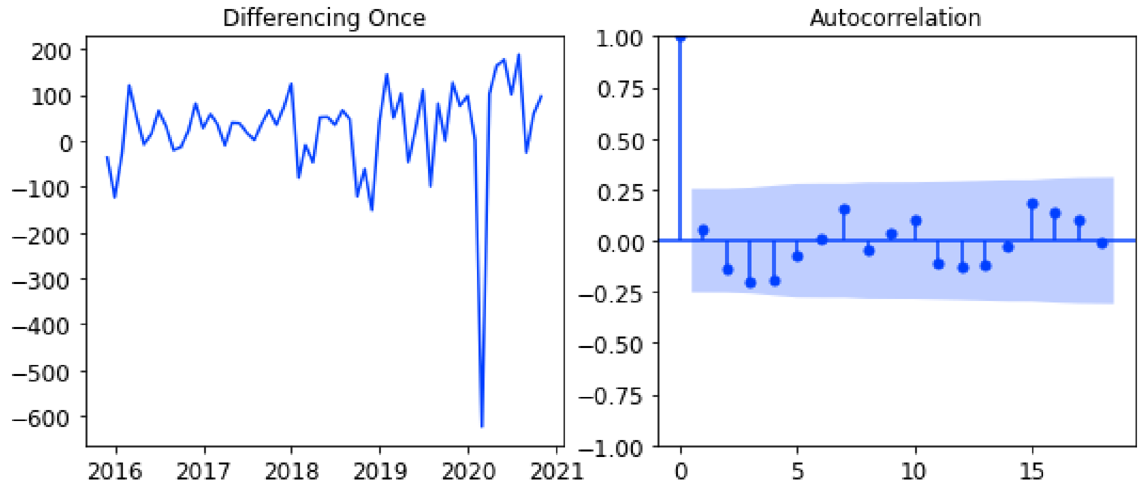
We can observe that there is no lag value present for which PACF crosses the upper confidence interval for the first time.

q

In moving average the current value of time series is a linear combination of past errors. We assume the errors to be independently distributed with the normal distribution. Order q of the MA process is obtained from the ACF plot, this is the lag after which ACF crosses the upper confidence interval for the first time

In [27]:

```
diff= df_close_month.diff().dropna()
fig, (axis_1 , axis_2) = plt.subplots(1,2, figsize=(10,4))
axis_1.plot(diff)
axis_1.set_title("Differencing Once")
axis_2.set_ylim(-1,1)
plot_acf(diff, ax=axis_2);
```



We can observe that there is no lag value present for which ACF crosses the upper confidence interval for the first time.

In [28]:
```
arima_model =  auto_arima(df_close_month_1)
arima_model
```

Out[28]:
```
ARIMA(order=(0, 1, 0), scoring_args={}, suppress_warnings=True)
```

Train test split

In [29]:
```
n= int(len(df_close_month_1)*0.75)
train_df= (df_close_month_1)[:n]
test_df= (df_close_month_1)[n:]
print(train_df.head())
print(len(train_df))
```

```
Date
2015-11-01    2088.02630615
2015-12-01    2051.35291315
2016-01-01    1927.88740786
2016-02-01    1902.56793844
2016-03-01    2023.68805916
Freq: MS, Name: Close, dtype: float64
45
```

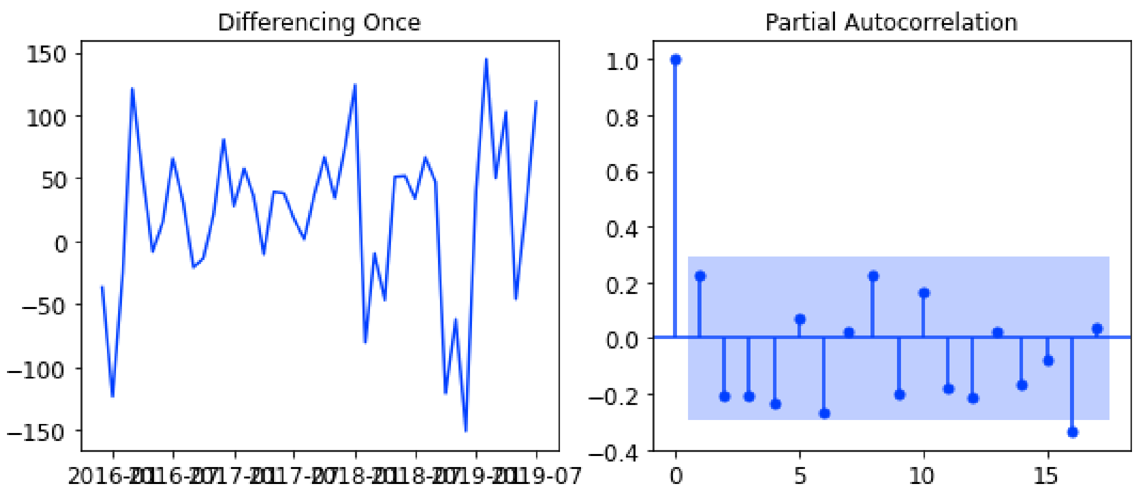
In [30]:
```
print(test_df.head())
print(len(test_df))
```

```
Date
2019-08-01    2898.17257592
2019-09-01    2979.19866536
2019-10-01    2978.98676128
2019-11-01    3105.80598958
2019-12-01    3181.54837135
Freq: MS, Name: Close, dtype: float64
16
```

PACF plot for Training set

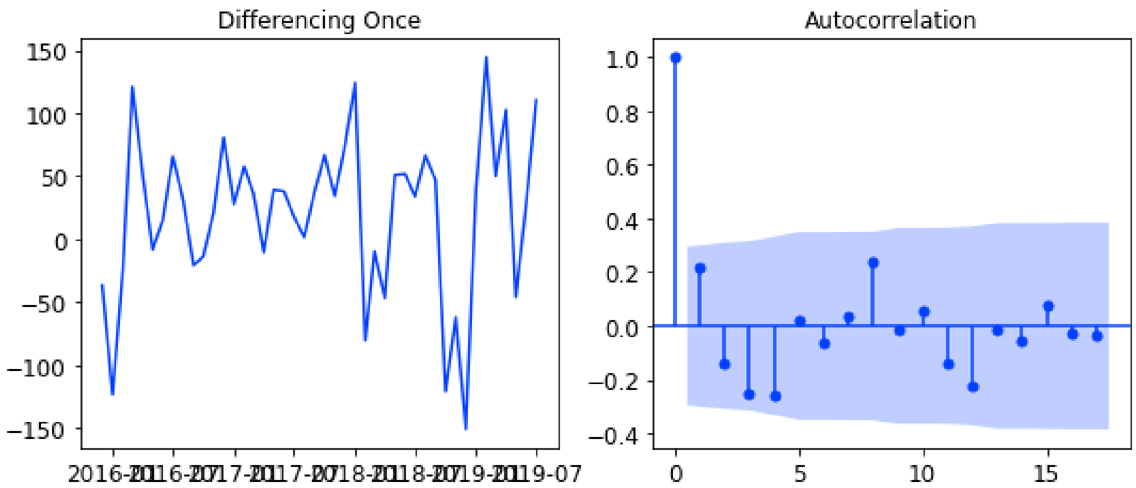
In [31]:
```
diff_train= train_df.diff().dropna()
fig, (axis_1 , axis_2) = plt.subplots(1,2, figsize=(10,4))
axis_1.plot(diff_train)
```

```
axis_1.set_title("Differencing Once")
plot_pacf(diff_train, ax=axis_2);
```



ACF plot for Training set

In [32]:
```
diff_train= train_df.diff().dropna()
fig, (axis_1 , axis_2) = plt.subplots(1,2, figsize=(10,4))
axis_1.plot(diff_train)
axis_1.set_title("Differencing Once")
plot_acf(diff_train, ax=axis_2);
```



In [33]:
```
auto_arima_train= auto_arima(train_df)
auto_arima_train
```

Out[33]:
```
ARIMA(order=(0, 1, 1), scoring_args={}, suppress_warnings=True)
```

In [34]:
```
train_model_autoARIMA=auto_arima(train_df, start_p=0, start_q=0,
                      test='adf',       # use adftest to find
                      max_p=4, max_q=4, # maximum p and q
                      m=12,
                      d=None,
                      seasonal=True,
                      start_P=0,
                      D=1,
                      trace=True,
                      error_action='ignore',
```

```
                              suppress_warnings=True,
                              stepwise=True)
print(train_model_autoARIMA.summary())
```

```
Performing stepwise search to minimize aic
 ARIMA(0,1,0)(0,1,1)[12]             : AIC=inf, Time=0.10 sec
 ARIMA(0,1,0)(0,1,0)[12]             : AIC=385.642, Time=0.01 sec
 ARIMA(1,1,0)(1,1,0)[12]             : AIC=385.420, Time=0.06 sec
 ARIMA(0,1,1)(0,1,1)[12]             : AIC=inf, Time=0.12 sec
 ARIMA(1,1,0)(0,1,0)[12]             : AIC=385.777, Time=0.02 sec
 ARIMA(1,1,0)(2,1,0)[12]             : AIC=384.854, Time=0.15 sec
 ARIMA(1,1,0)(2,1,1)[12]             : AIC=386.850, Time=0.50 sec
 ARIMA(1,1,0)(1,1,1)[12]             : AIC=inf, Time=0.22 sec
 ARIMA(0,1,0)(2,1,0)[12]             : AIC=384.082, Time=0.10 sec
 ARIMA(0,1,0)(1,1,0)[12]             : AIC=385.246, Time=0.04 sec
 ARIMA(0,1,0)(2,1,1)[12]             : AIC=386.077, Time=0.40 sec
 ARIMA(0,1,0)(1,1,1)[12]             : AIC=inf, Time=0.14 sec
 ARIMA(0,1,1)(2,1,0)[12]             : AIC=384.503, Time=0.17 sec
 ARIMA(1,1,1)(2,1,0)[12]             : AIC=386.026, Time=0.28 sec
 ARIMA(0,1,0)(2,1,0)[12] intercept   : AIC=385.814, Time=0.23 sec

Best model:  ARIMA(0,1,0)(2,1,0)[12]
Total fit time: 2.552 seconds
                                     SARIMAX Results
==========================================================================
=================
Dep. Variable:                                  y   No. Observations:
45
Model:             SARIMAX(0, 1, 0)x(2, 1, 0, 12)   Log Likelihood
-189.041
Date:                            Tue, 03 Aug 2021   AIC
384.082
Time:                                    12:18:49   BIC
388.479
Sample:                                         0   HQIC
385.539
                                             - 45
Covariance Type:                              opg
==========================================================================
=====
                 coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------------
-----
ar.S.L12      -0.6307      0.368     -1.716      0.086      -1.351
0.090
ar.S.L24      -0.4566      0.343     -1.333      0.183      -1.128
0.215
sigma2      6136.8046   2941.206      2.086      0.037     372.147    1.1
9e+04
==========================================================================
==========
Ljung-Box (L1) (Q):                   1.21   Jarque-Bera (JB):
0.16
Prob(Q):                              0.27   Prob(JB):
0.92
Heteroskedasticity (H):               2.43   Skew:
-0.15
Prob(H) (two-sided):                  0.16   Kurtosis:
2.82
==========================================================================
==========

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (co
mplex-step).
```

Ljung Box

```
The Ljung–Box test is a type of statistical test of whether
any of a group of autocorrelations of a time series are
different from zero. Instead of testing randomness at each
distinct lag, it tests the “overall” randomness based on a
number of lags and is, therefore, a portmanteau test.
```

- Ho: The model shows the goodness of fit(The autocorrelation is zero)
- Ha: The model shows a lack of fit(The autocorrelation is different from zero)

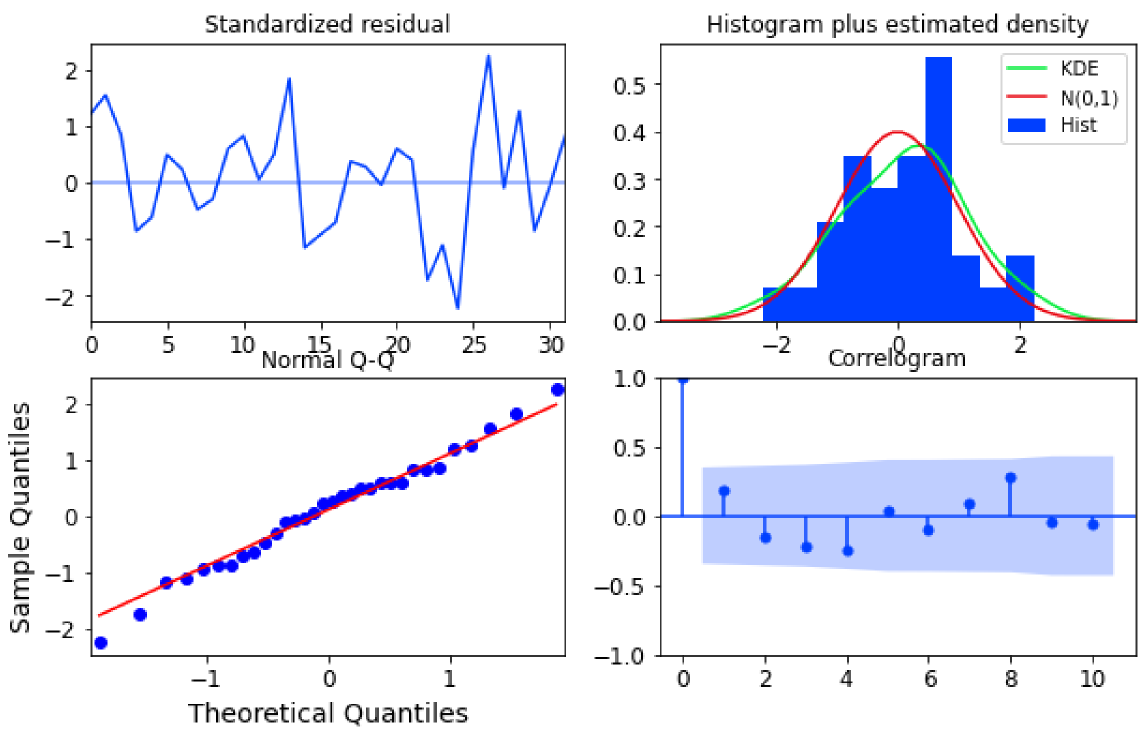
  My model here does not satisfy the goodness of fit condition because Probability(Q)=0.47.

Heteroscedasticity

```
Heteroscedasticity means unequal scatter. In regression
analysis, we talk about heteroscedasticity in the context
of the residuals or error term. Specifically,
heteroscedasticity is a systematic change in the spread of
the residuals over the range of measured values.

My residuals are heteroscedastic in nature since
Probability(Heteroskadisticy) is close to 0
```

In [35]:

```
train_model_autoARIMA.plot_diagnostics()
plt.show()
```



Interpretation

- Top left: The residual errors seem to fluctuate around a mean of zero and acted as white noise

- Top Right: The density plot suggest normal distribution with mean zero.
- Bottom left: All the dots should fall perfectly in line with the red line. Any significant deviations would imply the distribution is skewed.
- Bottom Right: The Correlogram , i.e , ACF plot shows the residual errors are not autocorrelated. Any autocorrelation would imply that there is some pattern in the residual errors which are not explained in the model. So you will need to look for more X’s (predictors) to the model.

Forecasting

In [36]:

```
test_df.head(10)
```

Out[36]:

```
Date
2019-08-01    2898.17257592
2019-09-01    2979.19866536
2019-10-01    2978.98676128
2019-11-01    3105.80598958
2019-12-01    3181.54837135
2020-01-01    3279.13679751
2020-02-01    3280.88376381
2020-03-01    2656.98193359
2020-04-01    2759.02799479
2020-05-01    2922.43740549
Freq: MS, Name: Close, dtype: float64
```

In [37]:

```
prediction = pd.DataFrame(train_model_autoARIMA.predict(n_periods = 16),
prediction.columns = ['predicted_stock_value']
prediction =prediction['predicted_stock_value']
prediction
```

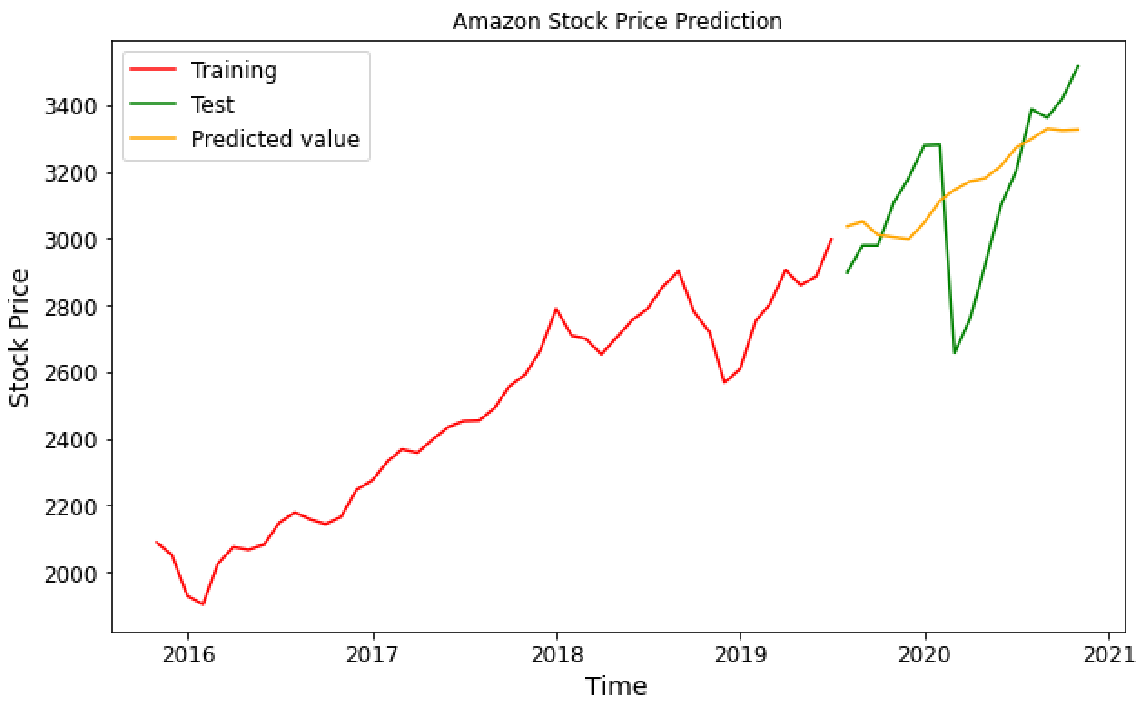
Out[37]:

```
Date
2019-08-01    3036.47640525
2019-09-01    3050.67451133
2019-10-01    3011.34797715
2019-11-01    3004.41371443
2019-12-01    2998.24852064
2020-01-01    3047.30570473
2020-02-01    3113.04643175
2020-03-01    3146.22430958
2020-04-01    3171.32415079
2020-05-01    3181.14213896
2020-06-01    3217.57453801
2020-07-01    3272.35292385
2020-08-01    3299.07640055
2020-09-01    3329.51779493
2020-10-01    3324.35430840
2020-11-01    3326.71963409
Freq: MS, Name: predicted_stock_value, dtype: float64
```

In [38]:

```
plt.figure(figsize=(10,6))
plt.plot(train_df,color='red',label="Training")
plt.plot(test_df,color='green',label="Test")

plt.plot(prediction,color='orange',label="Predicted value")
plt.title( 'Amazon Stock Price Prediction')
plt.xlabel('Time')
plt.ylabel('Stock Price')
```

```
plt.legend(loc='upper left', fontsize=12)
plt.show()
```



In [39]:

```
# prediction_1, se, conf = results.predict(13, alpha=0.05)  # 95% confid
# prediction_1_series = pd.Series(prediction_1, index=test_df.index)
# lower_series = pd.Series(conf[:, 0], index=test_df.index)
# upper_series = pd.Series(conf[:, 1], index=test_df.index)
# plt.figure(figsize=(12,5), dpi=100)
# plt.plot(train_df, label='training')
# plt.plot(test_df, color = 'blue', label='Actual Stock Price')
# plt.plot(prediction_1_series, color = 'orange',label='Predicted Stock
# plt.fill_between(lower_series.index, lower_series, upper_series,
#                  color='k', alpha=.10)
# plt.title('Amazon Stock Price Prediction')
# plt.xlabel('Month')
# plt.ylabel('Actual Stock Price')
# plt.legend(loc='upper left', fontsize=8)
# plt.show()
```

Report Performance

In [40]:

```
mse = mean_squared_error(test_df, prediction)
print('MSE: '+str(mse))
mae = mean_absolute_error(test_df, prediction)
print('MAE: '+str(mae))
rmse = math.sqrt(mean_squared_error(test_df, prediction))
print('RMSE: '+str(rmse))
mape = np.mean(np.abs(prediction - test_df)/np.abs(test_df))
print('MAPE: '+str(mape))
```

```
MSE: 43801.05946615053
MAE: 167.7994402154451
RMSE: 209.28702651179918
MAPE: 0.05597146674046523
```

RMSE = 209.3 & around 5.6% MAPE(Mean Absolute

Percentage Error) implies the model is about 94.4% accurate in predicting the test set observations.

Simple Exponential Smoothing

In [41]:
```
df_close_month=df['Close'].resample('MS').mean()
df_close_month.head(20)
n= int(len(df_close_month)*0.75)
train_df_1= df_close_month[:n]
test_df_1= df_close_month[n:]
print(len(train_df_1))
print(len(test_df_1))
```

```
45
16
```

Simple Exponential Smoothing

Prediction Using Simple Exponential Smoothing The simplest of the exponentially smoothing methods are naturally called simple exponential smoothing. This method is suitable for forecasting data with no clear trend or seasonal pattern.

Using the naïve method, all forecasts for the future are equal to the last observed value of the series. Hence, the naïve method assumes that the most recent observation is the only important one, and all previous observations provide no information for the future. This can be thought of as a weighted average where all of the weight is given to the last observation.

Using the average method, all future forecasts are equal to a simple average of the observed data. Hence, the average method assumes that all observations are of equal importance, and gives them equal weights when generating forecasts.

We often want something between these two extremes. For example, it may be sensible to attach larger weights to more recent observations than to observations from the distant past. This is exactly the concept behind simple exponential smoothing. Forecasts are calculated using weighted averages, where the weights decrease exponentially as observations come from further in the past — the smallest weights are associated with the oldest observations.

So large value of α (α denotes smoothing parameter)denotes that recent observations are given higher weight and a lower value of α denoted that more weightage is given to distant past values.

Modelling Using Simple Exponential Smoothing:

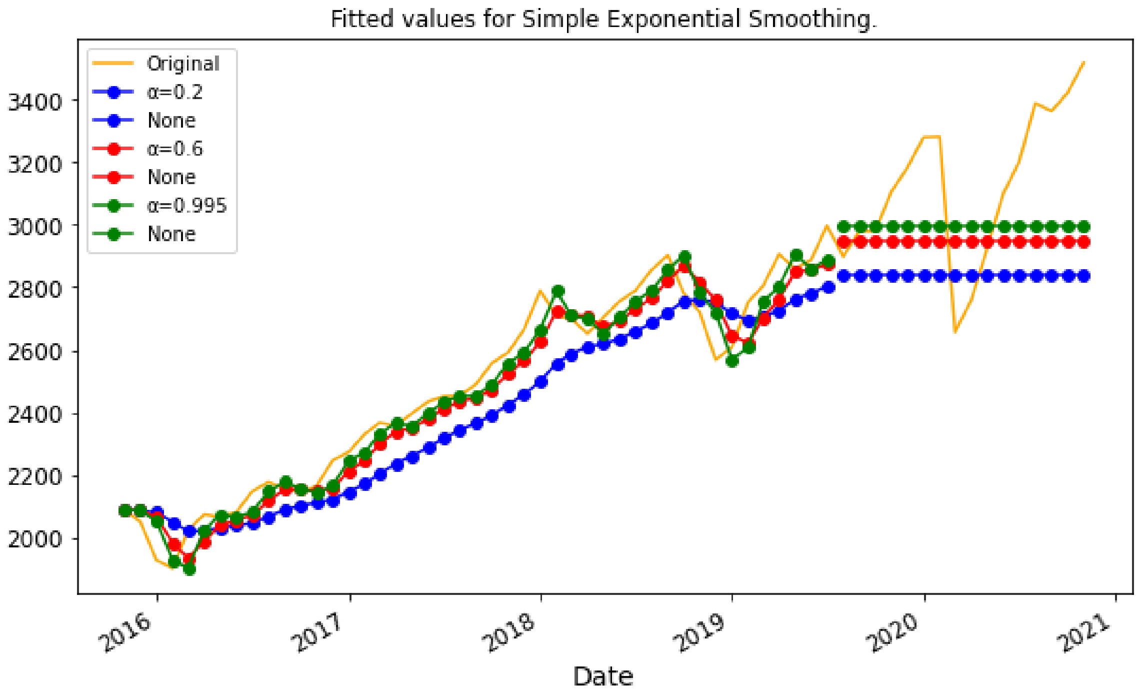
\begin{align} F_{t+1} = \sum_{i=0}^{t-1} α(1-α)^i y_{t-i} + (1-α)^t F_1 \end{align}
Where, F_{t+1} : Forecasted value of time series at time t+1 , F_t : Forecasted value of time series at time t

```
-In fit1, we explicitly provide the model with the
smoothing parameter α=0.2
-In fit2, we choose an α=0.6
```

```
-In fit3, we use the auto-optimization that allow
statsmodels to automatically find an optimized value for
us. This is  the recommended approach.
```

In [42]:

```
plt.plot(df_close_month,color='orange',label="Original")

# Simple Exponential Smoothing
fit_1 = SimpleExpSmoothing(train_df_1).fit(smoothing_level=0.2,optimized=
forecast_1 = fit_1.forecast(16).rename(r'α=0.2')
# plot 1
forecast_1.plot(marker='o', color='blue', legend=True)
fit_1.fittedvalues.plot(marker='o',  color='blue')

fit_2 = SimpleExpSmoothing(train_df_1).fit(smoothing_level=0.6,optimized=
forecast_2 = fit_2.forecast(16).rename(r'α=0.6')
# plot 2
forecast_2.plot(marker='o', color='red', legend=True)
fit_2.fittedvalues.plot(marker='o', color='red')


fit_3 = SimpleExpSmoothing(train_df_1).fit()
forecast_3 = fit_3.forecast(16).rename(r'α=%s'%fit_3.model.params['smoot
# plot 3
forecast_3.plot(marker='o', color='green', legend=True)
fit_3.fittedvalues.plot(marker='o', color='green')
plt.title("Fitted values for Simple Exponential Smoothing.")
plt.legend()
plt.show()
```



Fitted values for Simple Exponential Smoothing.

In [43]:

```
test_df.head()
```

Out[43]:

```
Date
2019-08-01    2898.17257592
2019-09-01    2979.19866536
2019-10-01    2978.98676128
2019-11-01    3105.80598958
```

```
2019-12-01    3181.54837135
Freq: MS, Name: Close, dtype: float64
```

In [44]:

```python
print( forecast_1.head())
```

```
2019-08-01    2841.41749337
2019-09-01    2841.41749337
2019-10-01    2841.41749337
2019-11-01    2841.41749337
2019-12-01    2841.41749337
Freq: MS, Name: α=0.2, dtype: float64
```

In [45]:

```python
print(forecast_2.head(5))
```

```
2019-08-01    2948.26080882
2019-09-01    2948.26080882
2019-10-01    2948.26080882
2019-11-01    2948.26080882
2019-12-01    2948.26080882
Freq: MS, Name: α=0.6, dtype: float64
```

In [46]:

```python
print(forecast_3.head(5))
```

```
2019-08-01    2996.98320438
2019-09-01    2996.98320438
2019-10-01    2996.98320438
2019-11-01    2996.98320438
2019-12-01    2996.98320438
Freq: MS, Name: α=0.995, dtype: float64
```

RMSE checking

In [47]:

```python
print(f"RMSE value for fit 1 : {math.sqrt(mean_squared_error(test_df_1,
print(f"RMSE value for fit 2 : {math.sqrt(mean_squared_error(test_df_1,
print(f"RMSE value for fit 3 : {math.sqrt(mean_squared_error(test_df_1,
```

```
RMSE value for fit 1 : 372.1372209521594
RMSE value for fit 2 : 298.1748464076985
RMSE value for fit 3 : 271.8076670971739
```

- Since the lowest RMSE score is for α=0.995 , The best output is given when α= 0.995, indicating recent observations are given the highest weight.

DOUBLE EXPONENTIAL SMOOTHING-HOLT'S TREND METHOD

- The basic equations for Holt's Method are:

\begin{align} μ_{t} = α y_{t} + (1-α) (μ_{t-i} + T_{t-1})\\ T_{t} = β (μ_{t} - μ_{t-1}) + (1-β) T_{t-1}\\ \end{align}\begin{align} F_{t+m} = μ_{t} + m T_{t} \end{align}
Where, $μ_{t}$: Exponentially smoothed value of the series at time t ,

y_{t} : Actual observation of time series at time t ,

T_{t} : Trend Estimate ,

$α$: Exponential Smoothing Constant for the data ,

β : Smoothing constant for trend ,

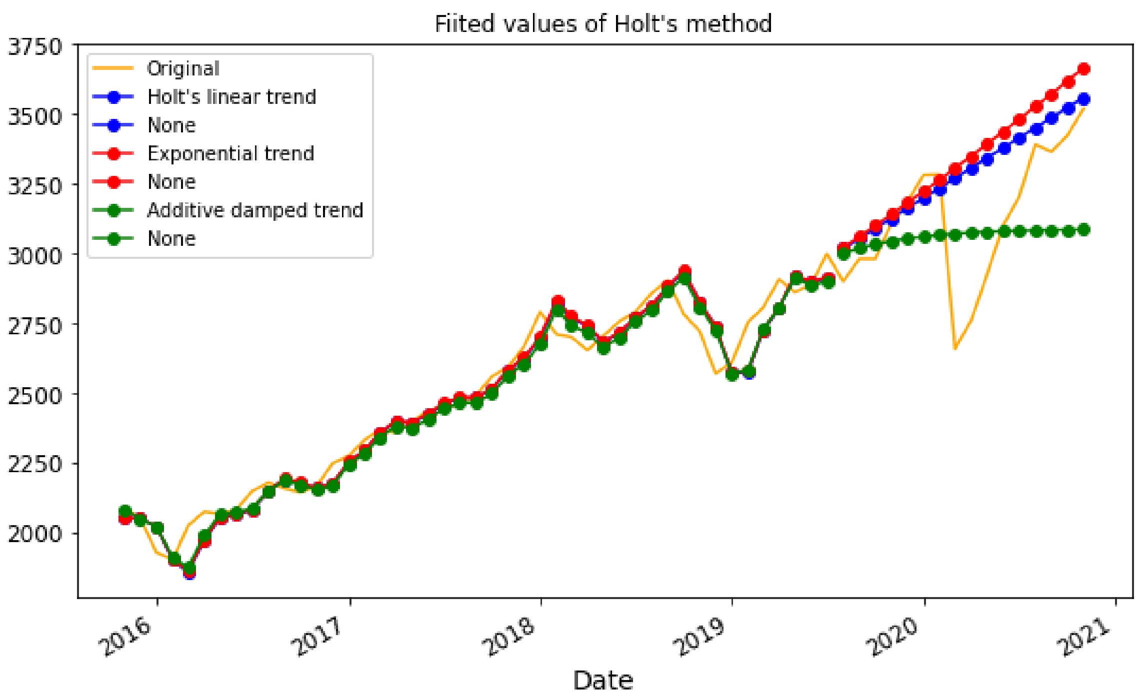
F_{t+m} : m period ahead forecasted value.

Modeling Using Holt's Model: Under this, we took three cases:

```
-In fit4, we explicitly provide the model with the
smoothing parameter α=0.8, β*=0.2.
-In fit5, we use an exponential model rather than a Holt's
additive model(which is the default).
-In fit6, we use a damped version of the Holt's additive
model but allow the dampening parameter ϕ to be optimized
while fixing the values for α=0.8, β*=0.2.
```

In [48]:

```
plt.plot(df_close_month,color='orange',label="Original")
# plot 4
fit_4= Holt(train_df_1).fit(smoothing_level=0.8, smoothing_slope=0.2, op
forecast_4= fit_4.forecast(16).rename("Holt's linear trend")
forecast_4.plot(marker='o', color='blue',legend=True)
fit_4.fittedvalues.plot(marker='o',  color='blue')

# plot 5
fit_5= Holt(train_df_1, exponential=True).fit(smoothing_level=0.8, smoot
forecast_5= fit_5.forecast(16).rename("Exponential trend")
forecast_5.plot(marker='o', color='red',legend=True)
fit_5.fittedvalues.plot(marker='o',  color='red')

# plot 6
fit_6= Holt(train_df_1, damped=True).fit(smoothing_level=0.8, smoothing_
forecast_6= fit_6.forecast(16).rename("Additive damped trend")
forecast_6.plot(marker='o', color='green',legend=True)
fit_6.fittedvalues.plot(marker='o',  color='green')
plt.title("Fiited values of Holt's method")
plt.legend()
plt.show()
```



Prediction Using Holt's model

In [49]:
```
test_df.head()
```

Out[49]:
```
Date
2019-08-01    2898.17257592
2019-09-01    2979.19866536
2019-10-01    2978.98676128
2019-11-01    3105.80598958
2019-12-01    3181.54837135
Freq: MS, Name: Close, dtype: float64
```

In [50]:
```
print(forecast_4.head(5))
```

```
2019-08-01    3016.18850459
2019-09-01    3052.19850828
2019-10-01    3088.20851198
2019-11-01    3124.21851568
2019-12-01    3160.22851938
Freq: MS, Name: Holt's linear trend, dtype: float64
```

In [51]:
```
print(forecast_5.head(5))
```

```
2019-08-01    3019.28281903
2019-09-01    3058.41457680
2019-10-01    3098.05350615
2019-11-01    3138.20618036
2019-12-01    3178.87925785
Freq: MS, Name: Exponential trend, dtype: float64
```

In [52]:
```
print(forecast_6.head(5))
```

```
2019-08-01    2999.56480227
2019-09-01    3017.12314450
2019-10-01    3031.16981829
2019-11-01    3042.40715732
2019-12-01    3051.39702854
Freq: MS, Name: Additive damped trend, dtype: float64
```

In [53]:
```
print(f"RMSE for Holt's linear trend : {math.sqrt(mean_squared_error(tes
print(f"RMSE for Exponential trend : {math.sqrt(mean_squared_error(test_
print(f"RMSE for Additive damped trend :{math.sqrt(mean_squared_error(te
```

```
RMSE for Holt's linear trend : 254.7468187625225
RMSE for Exponential trend : 288.7017424935487
RMSE for Additive damped trend :238.53592154909217
```

The lowest value of RMSE is when the model follows exponential trend with α=0.8 & β* = 0.2

TRIPLE EXPONENTIAL SMOOTHING HOLT’S WINTERS TREND AND SEASONALITY METHOD:

> Holt and Winters extended Holt’s method to capture seasonality. The Holt-Winters seasonal method comprises the forecast equation and three smoothing equations. It has three parameters alpha which is the level, Beta* which is the trend, and gamma which is the seasonality. The additive method is preferred when the seasonal variations are roughly constant through the series, while the

multiplicative method is preferred when the seasonal variations are changing proportionally to the level of the series.

Holt- Winter's Trend and Seasonality Method for Multiplicative Model:

It is generally considered to be best suited to forecasting time series that can be described by the equation:

$$y_t = (T_t * S_t * I_t)$$

This method is appropriate when a time series has a linear trend with a multiplicative seasonal pattern.

- Smoothing equation for the series
  $\mu_t = \alpha \frac{Yt}{St-p} + (1-\alpha)(\mu_{t-1} + b_{t-1})$ $\quad 0 \leq \alpha \leq 1$
- Trend estimating equation
  $b_t = \beta(\mu_t - \mu_{t-1}) + (1-\beta)\, b_{t-1}$
- Seasonality updating equation
  $S_t = \gamma \frac{Yt}{\mu t} + (1-\gamma)\, S_{t-p}$
- Forecast equation
  $F_{t+m} = (\mu_t + m\, b_t)\, S_{t+m-p}$

Where, $μ_{t}$: Exponentially smoothed value of the series at time t ,

y_{t} : Actual observation of time series at time t ,

T_{t} : Trend Estimate ,

$α$: Exponential Smoothing Constant for the data ,

$β$: Smoothing constant for trend ,

$γ$: Smoothing constant for seasonality ,

F_{t+m} : m period ahead forecasted value ,

p : the period of seasonality (p=4 for quarterly data & p=12 for monthly data.

Holt- Winter's Trend and Seasonality Method for Additive Model:

It is generally considered to be best suited to forecasting time series that can be described by the equation:

$$y_t=(T_t + S_t + I_t)$$

- Exponentially smoothed series equation

  $\mu_t=\alpha\ (y_t-S_{t-p}) + (1-\alpha)\ (\mu_{t-1}-+b_{t-1})$ $\qquad 0\leq \alpha \leq 1$
- Trend estimating equation

  $b_t\ = \beta(\mu_t- \mu_{t-1}) + (1-\beta)\ b_{t-1}$
- Seasonality updating equation

  $S_t = \gamma\ (y_t-\mu_t)+ (1-\gamma)\ S_{t-p}$
- Forecast equation

  $F_{t+m} = \mu_t + m\ b_t+S_{t+m-p}$

Where, $μ_{t}$: Exponentially smoothed value of the series at time t ,

y_{t} : Actual observation of time series at time t ,

T_{t} : Trend Estimate ,

$α$: Exponential Smoothing Constant for the data ,

$β$: Smoothing constant for trend ,

$γ$: Smoothing constant for seasonality ,

F_{t+m} : m period ahead forecasted value ,

p : the period of seasonality (p=4 for quarterly data & p=12 for monthly data).

Modeling Using Holt’s Winter Model

```
1.In fit 9, we use additive trend, additive seasonal of
period season_length=12, and a Box-Cox transformation.
2.In fit 10, we use additive trend, multiplicative seasonal
of period season_length=12, and a Box-Cox transformation.
3.In fit 11, we use additive damped trend, additive
seasonal of period season_length=12, and a Box-Cox
transformation.
4.In fit 12, we use multiplicative damped trend,
multiplicative seasonal of period season_length=4, and a
Box-Cox transformation.
```

Box-Cox Transformation

A Box-Cox transformation is a transformation of a non-normal dependent variable into a normal shape. Normality is an important assumption for many statistical techniques; if your data isn't normal, applying a Box-Cox means that you are able to run a broader number of tests.

In [54]:

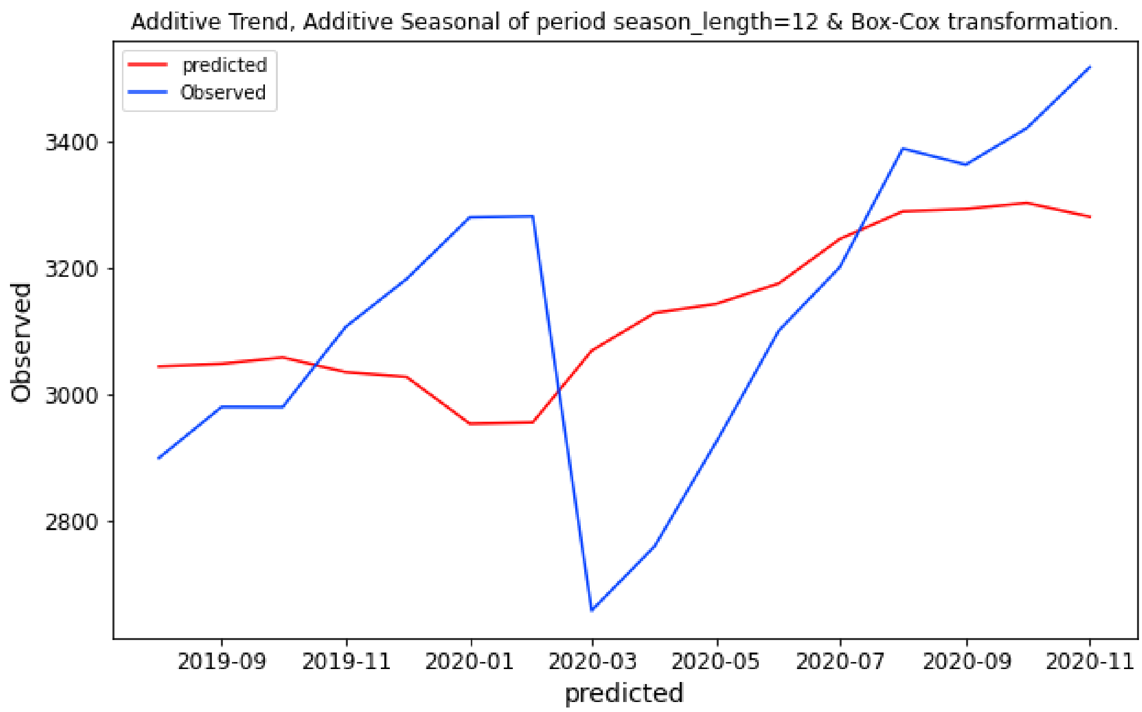
```
test_df_1=pd.DataFrame(test_df_1)
```

Fit 9

In [56]:

```
for i in range(1,17):
    fit_9= ExponentialSmoothing(train_df_1, seasonal_periods=12, trend='
    forecast_9= fit_9.forecast(i)
    first_forecast= pd.DataFrame(forecast_9, index= test_df_1.index, col

    first_forecast = first_forecast.join(test_df_1)
    first_forecast['RMSE']=np.sqrt(((first_forecast.Predicted_values-fir


print(first_forecast)

plt.plot(first_forecast.Predicted_values,color='red',label='predicted')
plt.plot(first_forecast.Close,label='Observed')
plt.xlabel('predicted')
plt.ylabel('Observed')
plt.title("Additive Trend, Additive Seasonal of period season_length=12
plt.legend()
plt.show()
import warnings
warnings.filterwarnings("ignore")
```

```
            Predicted_values         Close          RMSE
Date
2019-08-01      3043.28400687 2898.17257592 145.11143095
2019-09-01      3047.37427384 2979.19866536  68.17560848
2019-10-01      3057.35880881 2978.98676128  78.37204754
2019-11-01      3034.37464405 3105.80598958  71.43134553
2019-12-01      3026.64674166 3181.54837135 154.90162968
2020-01-01      2952.73243172 3279.13679751 326.40436579
2020-02-01      2954.87053910 3280.88376381 326.01322471
2020-03-01      3068.04343741 2656.98193359 411.06150382
2020-04-01      3127.82870980 2759.02799479 368.80071501
2020-05-01      3141.96221600 2922.43740549 219.52481051
2020-06-01      3174.35413040 3099.55335286  74.80077754
2020-07-01      3244.79719259 3200.27257907  44.52461352
2020-08-01      3288.47613540 3387.95970105  99.48356565
2020-09-01      3292.36878737 3362.48432617  70.11553880
2020-10-01      3301.87260407 3420.35094821 118.47834414
2020-11-01      3279.99859878 3516.81796875 236.81936997
```



In [57]:

```
prediction_first=first_forecast['Predicted_values']
mape_first = np.mean(np.abs(prediction_first - first_forecast['Close'])/
print(f"Mean Absolute Percentage Error for first forecast : {mape_first}
print(f"Average RMSE value of Fit 9 : {np.mean(first_forecast.RMSE)}")
```

```
Mean Absolute Percentage Error for first forecast : 0.057724603677793376
Average RMSE value of Fit 9 : 175.87618072662758
```

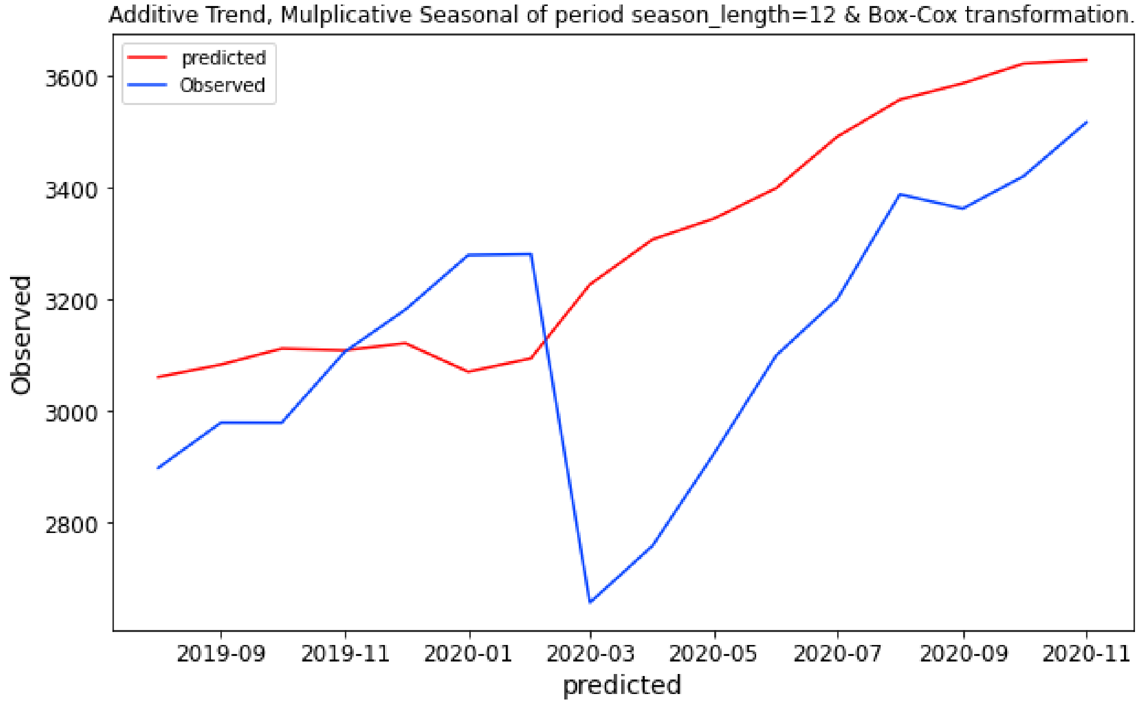
fit 10

In [58]:

```
for i in range(1,17):
    fit_10= ExponentialSmoothing(train_df_1, seasonal_periods=12, trend=
    forecast_10= fit_10.forecast(i)
    second_forecast= pd.DataFrame(forecast_10, index= test_df_1.index, c

    second_forecast = second_forecast.join(test_df_1)
    second_forecast['RMSE']=np.sqrt(((second_forecast.Predicted_values-s
print(second_forecast)

plt.plot(second_forecast.Predicted_values,color='red',label='predicted')
plt.plot(second_forecast.Close,label='Observed')
plt.xlabel('predicted')
plt.ylabel('Observed')
plt.title("Additive Trend, Mulplicative Seasonal of period season_length
plt.legend()
plt.show()
```

```
                Predicted_values        Close          RMSE
Date
2019-08-01        3060.90320313 2898.17257592 162.73062721
2019-09-01        3083.16403940 2979.19866536 103.96537403
2019-10-01        3111.89239572 2978.98676128 132.90563444
2019-11-01        3108.75370653 3105.80598958   2.94771695
2019-12-01        3121.44681343 3181.54837135  60.10155792
2020-01-01        3070.31489201 3279.13679751 208.82190550
2020-02-01        3094.40463960 3280.88376381 186.47912421
2020-03-01        3226.59714566 2656.98193359 569.61521206
2020-04-01        3307.17951477 2759.02799479 548.15151998
```

```
2020-05-01     3344.13086659 2922.43740549 421.69346109
2020-06-01     3399.14814743 3099.55335286 299.59479457
2020-07-01     3490.91821908 3200.27257907 290.64564001
2020-08-01     3557.44232471 3387.95970105 169.48262366
2020-09-01     3586.66775523 3362.48432617 224.18342905
2020-10-01     3621.87321214 3420.35094821 201.52226393
2020-11-01     3628.29829523 3516.81796875 111.48032648
```



In [59]:

```
prediction_second=second_forecast['Predicted_values']
mape_second = np.mean(np.abs(prediction_second - second_forecast['Close'
print(f"Mean Absolute Percentage Error for second forecast : {mape_secon
print(f"Average RMSE value of Fit 10 : {np.mean(second_forecast.RMSE)}")
```

```
Mean Absolute Percentage Error for second forecast : 0.0767604289262821
Average RMSE value of Fit 10 : 230.89507569369073
```

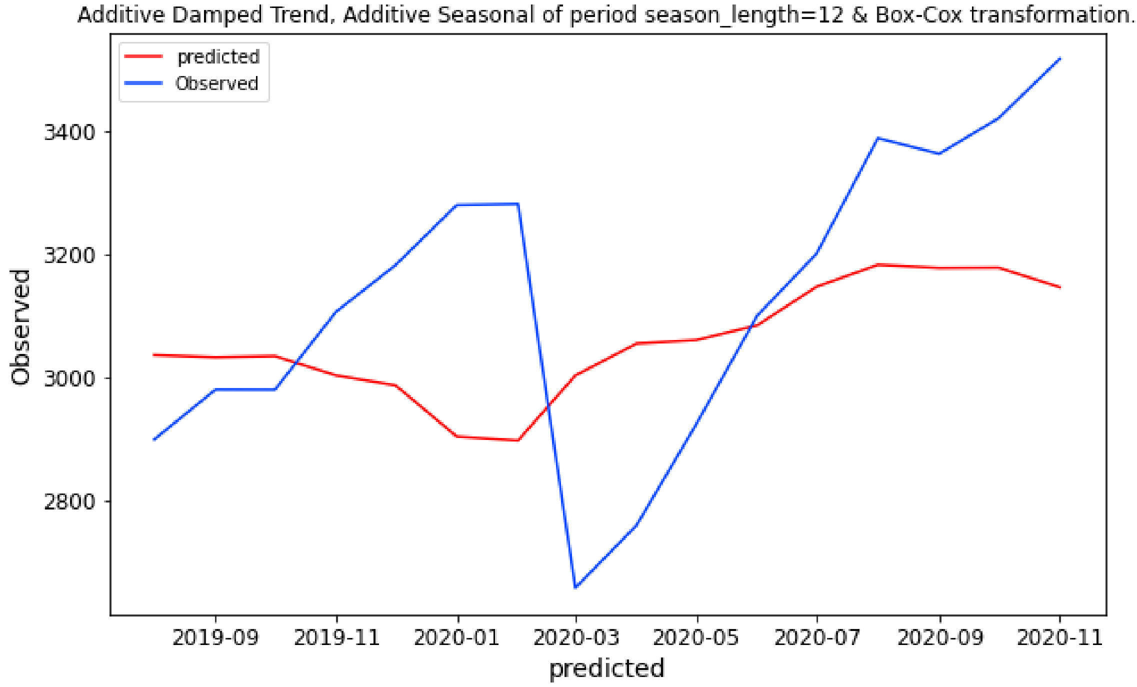
Fit 11

In [60]:

```
for i in range(1,17):
    fit_11= ExponentialSmoothing(train_df_1, seasonal_periods=12, trend=
    forecast_11= fit_11.forecast(i)
    third_forecast= pd.DataFrame(forecast_11, index= test_df_1.index, co

    third_forecast = third_forecast.join(test_df_1)
    third_forecast['RMSE']=np.sqrt(((third_forecast.Predicted_values-thi
print(third_forecast)

plt.plot(third_forecast.Predicted_values,color='red',label='predicted')
plt.plot(third_forecast.Close,label='Observed')
plt.xlabel('predicted')
plt.ylabel('Observed')
plt.title("Additive Damped Trend, Additive Seasonal of period season_len
plt.legend()
plt.show()
```

```
                Predicted_values          Close          RMSE
Date
2019-08-01     3035.49693612 2898.17257592 137.32436020
2019-09-01     3031.70666542 2979.19866536  52.50800005
```

```
2019-10-01     3033.70933303 2978.98676128  54.72257175
2019-11-01     3002.41163301 3105.80598958 103.39435658
2019-12-01     2986.27495970 3181.54837135 195.27341165
2020-01-01     2903.07093405 3279.13679751 376.06586345
2020-02-01     2896.45157474 3280.88376381 384.43218906
2020-03-01     3002.32716438 2656.98193359 345.34523079
2020-04-01     3054.19781575 2759.02799479 295.16982096
2020-05-01     3059.70121112 2922.43740549 137.26380563
2020-06-01     3083.72104750 3099.55335286  15.83230537
2020-07-01     3146.53222568 3200.27257907  53.74035339
2020-08-01     3182.09376943 3387.95970105 205.86593161
2020-09-01     3176.97073808 3362.48432617 185.51358809
2020-10-01     3177.48344035 3420.35094821 242.86750786
2020-11-01     3145.68087118 3516.81796875 371.13709757
```



In [61]:

```
prediction_third=third_forecast['Predicted_values']
mape_third = np.mean(np.abs(prediction_third - third_forecast['Close'])/
print(f"Mean Absolute Percentage Error for third forecast : {mape_third}
print(f"Average RMSE value of Fit 11 : {np.mean(third_forecast.RMSE)}")
```

```
Mean Absolute Percentage Error for third forecast : 0.0630128813460985
Average RMSE value of Fit 11 : 197.27852462561344
```

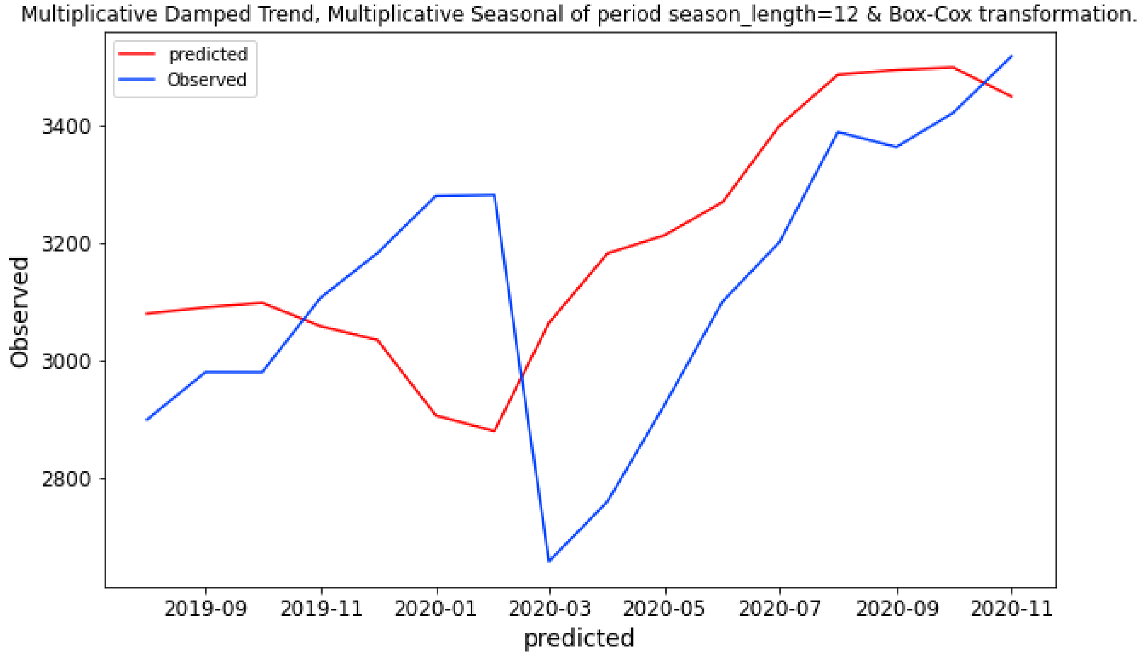
Fit 12

In [62]:

```
for i in range(1,17):
    fit_12= ExponentialSmoothing(train_df_1, seasonal_periods=12, trend=
    forecast_12= fit_12.forecast(i)
    fourth_forecast= pd.DataFrame(forecast_12, index= test_df_1.index, c

    fourth_forecast = fourth_forecast.join(test_df_1)
    fourth_forecast['RMSE']=np.sqrt(((fourth_forecast.Predicted_values-f
print(fourth_forecast)

plt.plot(fourth_forecast.Predicted_values,color='red',label='predicted')
plt.plot(fourth_forecast.Close,label='Observed')
plt.xlabel('predicted')
plt.ylabel('Observed')
plt.title("Multiplicative Damped Trend, Multiplicative Seasonal of perio
```

```
plt.legend()
plt.show()
```

```
                Predicted_values        Close         RMSE
Date
2019-08-01      3078.72318675 2898.17257592 180.55061083
2019-09-01      3089.44353553 2979.19866536 110.24487017
2019-10-01      3097.20377473 2978.98676128 118.21701346
2019-11-01      3057.18273013 3105.80598958  48.62325946
2019-12-01      3033.86333143 3181.54837135 147.68503992
2020-01-01      2905.27977803 3279.13679751 373.85701947
2020-02-01      2878.63546633 3280.88376381 402.24829747
2020-03-01      3063.12135989 2656.98193359 406.13942629
2020-04-01      3181.30694612 2759.02799479 422.27895133
2020-05-01      3211.84464466 2922.43740549 289.40723916
2020-06-01      3268.72350880 3099.55335286 169.17015594
2020-07-01      3398.26824266 3200.27257907 197.99566359
2020-08-01      3485.57005713 3387.95970105  97.61035609
2020-09-01      3493.36853754 3362.48432617 130.88421137
2020-10-01      3497.84274804 3420.35094821  77.49179983
2020-11-01      3448.44728716 3516.81796875  68.37068159
```



In [63]:

```
prediction_fourth=fourth_forecast['Predicted_values']
mape = np.mean(np.abs(prediction_fourth - fourth_forecast['Close'])/np.a
print(f"Mean Absolute Percentage Error for fourth forecast : {mape}")
print(f"Average RMSE value for Fit 12 : {np.mean(fourth_forecast.RMSE)}"
```

```
Mean Absolute Percentage Error for fourth forecast : 0.06680602847237493
Average RMSE value for Fit 12 : 202.54841224794438
```

we can easily see that, the Holt-winter method with Additive trend and seasonality is giving us lowest RMSE(175.23). Therefore, we get the most accurate forecasted values for the testing data of our stock price dataset by using this method

MAPE for this method is around 5.2% implies the model is about 94.8% accurate in predicting the test set observations.

The Best Forecasting model is "Fit 9"

In [ ]:

Chapter V. Discussions:

After conducting the project, it became evident that ARIMA model can be used for forecasting purposes, however it has some limitations. The main requirement for the accurate prediction is a presence of a time series with a small volatility, therefore, the best forecasting results in the research were produced for the stock index and the worst for stocks. Indexes are less volatile due to a big number of companies that they include and this gives an effect of diversification.

In order to improve forecasting results for a stock price, the Exponential Smoothing with the help of Holt- Winter's Trend and Seasonality Method for Additive Model and Box-Cox Transformation is more appropriate for volatile stock prices than ARIMA model which is suitable for relatively stable broad stock indexes.

In general, ARIMA models provided more precise forecasts over one, two and three days than over the ten-day horizon. These findings agree with the conclusions made by other researchers that ARIMA model is capable to fully utilize time series patterns in order to make accurate short-term forecasts. (L.C. Kyungjoo, Y. Sehwan and J. John, 2007)

In addition, it was determined that while ARMA model can sometimes compete with ARIMA model to provide the most accurate short-term forecasts, it does not stand a chance in long-term forecasting (more than a week). Thus, the conclusion can be made from all discovered facts that ARMA models, produce less precise forecasting results for stock price and index predictions than ARIMA models do.

Moreover, after extending the time series from the one year to the five-year time series, precision of FTSE All-Share index`s and Barclay's stock value forecasts worsened for the short-term periods and improved for the ten day period while forecast accuracy of the GSK stock deteriorated among all periods. Thus, it can be assumed that a one-year time series is sufficient to conduct a forecast for the next one-three days while a longer time series can be considered for longer term forecast. However, the drawback of using long time series is that it has higher probability of containing periods with high volatility that is not relevant to the current time and can distort the forecast.

Chapter VII. Conclusion:

In this project it was determined that ARIMA model can be effectively applied for forecasting values of stock indexes or diversified equity portfolios that do not possess company specific risk. However, ARIMA model is sometime not suitable for predicting stock prices because of their highly volatile nature and embedded unsystematic risk. These factors make ARIMA forecast for stock prices highly deviated from the actual results. Nevertheless, ARIMA model provides more precise forecasting results than ARMA.

It was found that in order to increase accuracy of a stock price forecast, the Exponential Smoothing with the help of Holt- Winter's Trend and Seasonality Method for Additive Model and Box-Cox Transformation should be used. ARMA model is usually based on an non-stationary time series of a stock price that possesses heteroscedasticity which is treated by a Box-Cox transformation as volatility to be modeled. Subsequently, Exponential Smoothing with the help of Holt- Winter's Trend and Seasonality Method for Additive Model and Box-Cox Transformation can be used to forecast stock prices while taking into account the volatility forecasts.

The inverse relationship between the length of the forecast and the forecast precision was determined. However, this relationship was not perfect among the parts of the short-term forecast (one two- and three-day forecasts), meaning that on a number of occasions this relationship was violated by forecasts that were situated closely in time. Nevertheless, the general tendency was unambiguous – the precision of the short-term forecast was higher than the accuracy of the long-term predictions. The interdependence between a time length of a forecast and its forecasting accuracy was not linear, suggesting that forecasting precision decreased more slowly than the time length of the forecast increased.

In addition, it was discovered that a forecasting model based on a one-year time series provides relatively precise forecasting results for one-three-day periods, while a five-year time series is more appropriate for longer term forecasting.

This project was restricted to one stock index and two stocks, therefore, in order to confirm the conclusions made, further works should be conducted and more financial securities should be examined. Moreover, it would be useful to check ARIMA forecasting precision over other future periods, apart from one-, two-, three- and ten-day horizons.

List of References

Adebiyi, A. and O. Adewumi, A. (2014). Stock Price Prediction Using the ARIMA Model. *2014 UKSim-AMSS 16th International Conference on Computer Modelling and Simulation, At Cambridge University, United Kingdom*. [online] Available at: http://www.researchgate.net/publication/261179224_Stock_Price_Prediction_Using_the_ARIMA_Model [Accessed 1 Jun. 2015].

Box, G., Jenkins, G. and Reinsel, G. (2013). *Time Series Analysis: Forecasting and Control*. 4th ed. United States: John Wiley & Sons Inc.

John E Hanke, Dean W. Wicheran, Arthur G. Reitsch. (2013). *Business Forecasting*. (8th Edition), Pearson, Prentice Hall, New Jersey ()

Fama, E. (1965). The Behavior of Stock-Market Prices. *The Journal of Business*, [online] 38(1), pp.34-105. Available at: http://www.jstor.org/stable/2350752 [Accessed 16 May 2015].