CAPSTONE PROJECT-2

Machine Learning – Python

Project Title :

TIME SERIES ON HDFCBANK STOCK PRICE.

Abstract :

Our main objective of the this idea is to predict the price of holdings on HDFCBANK share price using the given data of the share price from 2000 to 2020. The available dataset is amputee i.e. the not required columns has been removed and the same data is used for training and testing. Time Series model was developed based on the dataset and applied to the test dataset to find out the accuracy on the values predicted.

Submitted By:

TARUN M V

Table of Content:

1. Introduction on project

Stock price prediction is very helpful in real life, If the accuracy is good then it good enough to trust and invest, keeping this in view this model was build. The motive is to create model on which the prediction is done and then it will be usefull for future investments. The data set used is on HDFCBANK.

2. Importance of the project :

What is the need for stock price prediction?

If we want to invest we are not allowed to invest without any prior knowledge because if anything goes wong we end loosing huge amout of money. Stock price prediction is very helpful in real life, If the accuracy is good then it good enough to trust and invest and gain some profits out of it is the ultimate goal.

3. Dataset:

```
stock_raw_data.shape
```

```
(5204, 15)
```

The number of observations in data set are 5204.

The features in data set are 15, where 12 features are numeric features and 3 feature are objects.

```
stock_raw_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5204 entries, 0 to 5203
Data columns (total 15 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Date                5204 non-null   object
 1   Symbol              5204 non-null   object
 2   Series              5204 non-null   object
 3   Prev Close          5204 non-null   float64
 4   Open                5204 non-null   float64
 5   High                5204 non-null   float64
 6   Low                 5204 non-null   float64
 7   Last                5204 non-null   float64
 8   Close               5204 non-null   float64
 9   VWAP                5204 non-null   float64
 10  Volume              5204 non-null   int64
 11  Turnover            5204 non-null   float64
 12  Trades              2354 non-null   float64
 13  Deliverable Volume  4695 non-null   float64
 14  %Deliverble         4695 non-null   float64
dtypes: float64(11), int64(1), object(3)
memory usage: 610.0+ KB
```

4. Feature description:

1. **Date :** Date of the trading took place
2. **Symbol :** Stock name
3. **Series :** EQ, It stands for Equity
4. **Prev Close :** It represents previous day closing price
5. **Open :** Open price of stock
6. **High :** Highest value the current stock hit for that day
7. **Low :** Lowest value the current stock hit for that day
8. **Last :** is the price at which the last matched trade
9. **Close**: the end of a trading session in the financial markets when the markets close for the day.
10. **VWAP :** Volume weighted average price
11. **Volume:** measures the number of shares traded in a stock or contracts traded in futures or options.
12. **Turnover :** a measure of stock liquidity, calculated by dividing the total number of shares traded during some period by the average number of shares outstanding for the same period
13. **Trades:** buying and selling shares in companies in an effort to make money on daily changes in price.
14. **Deliverable Volume:** represents that portion of overall traded volume in which an investor has taken the delivery into a demat account or sell from a demat account.
15. **%Deliverable:** a very important indicator when you want to understand whether the stock is bought for long term or just for speculative trading.

```
stock_raw_data.describe()
```

	Prev Close	Open	High	Low	Last	Close	VWAP	Volume	Turnover	Trades	Deliverable Volume
count	5204.000000	5204.000000	5204.000000	5204.000000	5204.000000	5204.000000	5204.000000	5.204000e+03	5.204000e+03	2354.000000	4.695000e+03
mean	997.948328	998.267179	1010.719350	984.770417	998.208878	998.194956	997.778255	1.943375e+06	2.202864e+14	76090.364486	1.183619e+06
std	638.481104	638.133600	644.158911	632.202775	638.437227	638.404237	638.195463	3.806884e+06	4.427248e+14	88162.140713	1.999756e+06
min	157.400000	162.150000	167.900000	157.000000	163.000000	163.400000	161.400000	1.042000e+03	2.291142e+10	807.000000	4.631000e+03
25%	470.637500	470.000000	476.550000	463.750000	471.712500	471.212500	470.087500	2.931628e+05	1.534602e+13	26334.750000	2.564430e+05
50%	915.675000	919.575000	935.000000	901.575000	915.700000	915.875000	917.440000	9.140735e+05	1.115150e+14	42183.500000	6.122350e+05
75%	1389.912500	1390.225000	1412.150000	1360.837500	1391.175000	1389.987500	1390.080000	2.024575e+06	2.067470e+14	89444.250000	1.344856e+06
max	2565.800000	2566.000000	2583.300000	2553.700000	2563.000000	2565.800000	2570.700000	1.005650e+08	1.426400e+16	790631.000000	6.669683e+07

- ➢ As we can see the max count is 5204 which is the number of observation in the data set.
- ➢ We can also see some of the feature count is less than 5204 which means it has some missing data which will treated in future steps.
- ➢ We get to know the mean, standard deviation and quartile values using this command describe().
- ➢ We also get to know the minimum and miximum values in the particular features.

5. Data Preparation

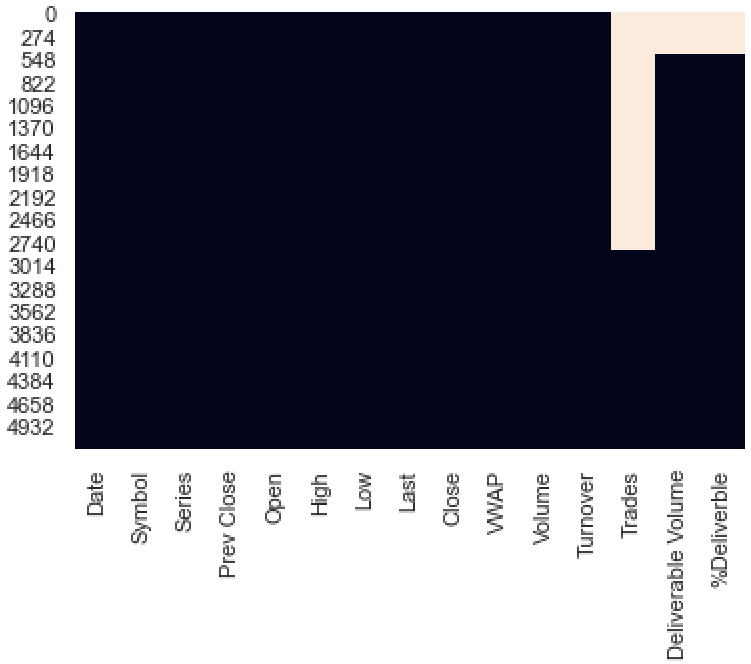
We remove unwanted columns that is not needed and check missing values. Aggregate sales data by date and finally index it with the time series data.

Heat map to check null values

```
stock_raw_data.isnull().sum()
```

```
Date                     0
Symbol                   0
Series                   0
Prev Close               0
Open                     0
High                     0
Low                      0
Last                     0
Close                    0
VWAP                     0
Volume                   0
Turnover                 0
Trades                2850
Deliverable Volume     509
%Deliverble            509
dtype: int64
```

The above command is used to check the number of null values present in the particular features in the dataset.



From the above heat map we can see the null values present in the features (Trade, Deliverable volume and % Deliverable)

How to treat null values in the data?

- Find th out the null values using code as well as using visualization
- Once the features are found with null values. Then we use imputation.
- **Imputation** is the process of replacing missing data with substituted values.
- Here in this case we are not going to replace the null values of the above 3 features with the mean of the those particular features as it is irrelavent to the objective.

Changing data format to datetime:

Here we are copying the data to new dataframe and changing the format of date to datetime format for better view of stock price view

```
stock_data['Date'] = stock_data['Date'].apply(lambda x:pd.to_datetime(x))
```

```
stock_data.head()
```

	Date	Symbol	Series	Prev Close	Open	High	Low	Last	Close	VWAP	Volume	Turnover	Trades	Deliverable Volume	%Deliverble
0	2000-03-01	HDFCBANK	EQ	157.40	166.00	170.00	166.00	170.00	170.00	169.52	33259	5.638120e+11	NaN	NaN	NaN
1	2000-04-01	HDFCBANK	EQ	170.00	182.00	183.45	171.00	174.00	173.80	174.99	168710	2.952260e+12	NaN	NaN	NaN
2	2000-05-01	HDFCBANK	EQ	173.80	170.00	173.90	165.00	168.00	166.95	169.20	159820	2.704090e+12	NaN	NaN	NaN
3	2000-06-01	HDFCBANK	EQ	166.95	168.00	170.00	165.30	168.95	168.30	168.44	85026	1.432170e+12	NaN	NaN	NaN
4	2000-07-01	HDFCBANK	EQ	168.30	162.15	171.00	162.15	170.75	168.35	166.79	85144	1.420160e+12	NaN	NaN	NaN

Before format:

```
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Date                  5204 non-null   object
 1   Symbol                5204 non-null   object
 2   Series                5204 non-null   object
 3   Prev Close            5204 non-null   float64
 4   Open                  5204 non-null   float64
```

After format:

```
stock_data.dtypes
```

```
Date                  datetime64[ns]
Symbol                        object
Series                        object
Prev Close                   float64
Open                         float64
High                         float64
Low                          float64
Last                         float64
Close                        float64
VWAP                         float64
Volume                         int64
Turnover                     float64
Trades                       float64
Deliverable Volume           float64
%Deliverble                  float64
dtype: object
```

Removing the features not required for my prediction:

Remove columns that we do not need

```
cols = ['Symbol','Series','Prev Close','Open','High','Low','Last','VWAP','Volume','Turnover','Trades',
        'Deliverable Volume','%Deliverble']
stock_data_1 = stock_data.drop(cols, axis=1, inplace=True)
stock_data_1 = stock_data.sort_values('Date')
```

Two features left and no null values:

```
: stock_data_1.isnull().sum()
```

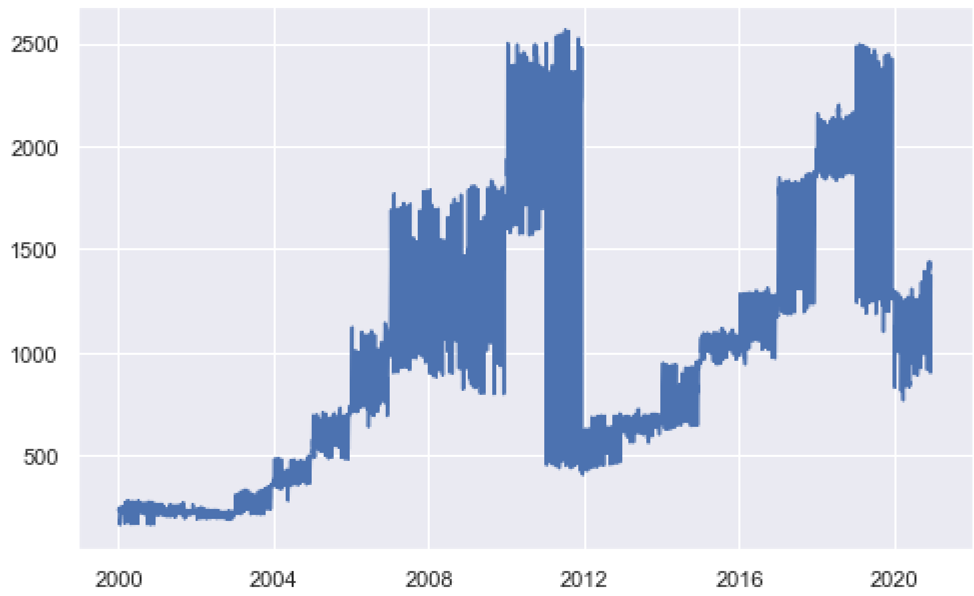
```
: Date     0
  Close    0
  dtype: int64
```

Data indexing:

It is a data structure technique which is used to quickly locate and access the data in a database. Indexes are created using a few database columns.

The data has been indexed and then the plot was drawn for the data.

An example of index is **to put data in ascending order**.



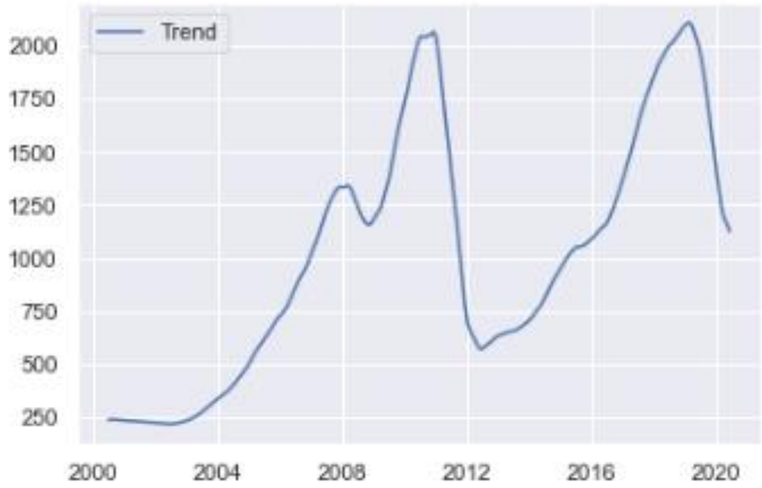
Since the data is not that clear to understand we use the sampling technique here to understand the data much more comfortably.

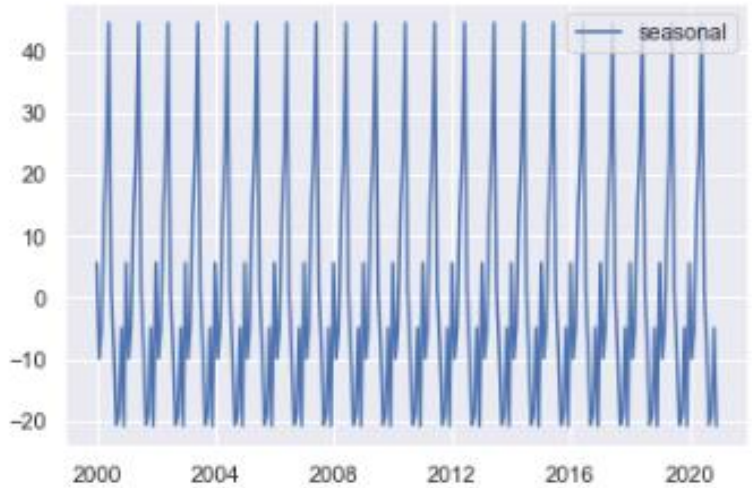
Hence we are sampling the date to month wise so that it will have far better view and easy to understand.

6. Decomposing

In this approach, both trend and seasonality are modelled separately and the remaining part of the series is returned.

We have used seasonal_decompose feature for better understanding the data






```
<matplotlib.legend.Legend at 0x171f5088850>
```

7. Checking Stationarity

Time-series analysis should be to check whether there is any evidence of a trend or seasonal effects and, if there is, remove them.

Augmented Dickey-Fuller (ADF) statistic is one of the more widely used statistic test to check whether your time series is stationary or non-stationary. It uses an **autoregressive model and optimizes an information criterion** across multiple different lag values.

The null hypothesis of the test is that the time series can be represented by a unit root, that it is not stationary (**has some time-dependent structure**). The alternate hypothesis (**rejecting the null hypothesis**) is that the time series is stationary.

a. Performing the Dickey Fuller Test

Fuller test tests **the null hypothesis that a unit root is present in an autoregressive time series model**. The alternative hypothesis is different depending on which version of the test is used, but is usually stationarity or trend-stationarity.

```
Results of Dickey Fuller Test:
Test statistics                 -2.366739
p-value                          0.151345
#Lags Used                       6.000000
Number of Observations used    245.000000
Critical value (1%)             -3.457326
Critical value (5%)             -2.873410
Critical value (10%)            -2.573096
dtype: float64
```

The p-value is 0.15, which is beyond the threshold (0.05). Hence the null-hypothesis is not rejected.

Test statistics > critical value, hence we reject the null hypothesis which implies it is not stationary.

b. KPSS Test

KPSS test is a statistical test to **check for stationarity of a series around a deterministic trend.** The p-value is 0.01, which is way below the threshold (0.05). Hence the null-hypothesis is rejected.

```
Results of KPSS Test
Test Statistics           0.803629
p-value                   0.010000
Lags Used                16.000000
Critical value (10%)      0.347000
Critical value (5%)       0.463000
Critical value (2.5%)     0.574000
Critical value (1%)       0.739000
dtype: float64
```

8. Differencing to remove the seasonality

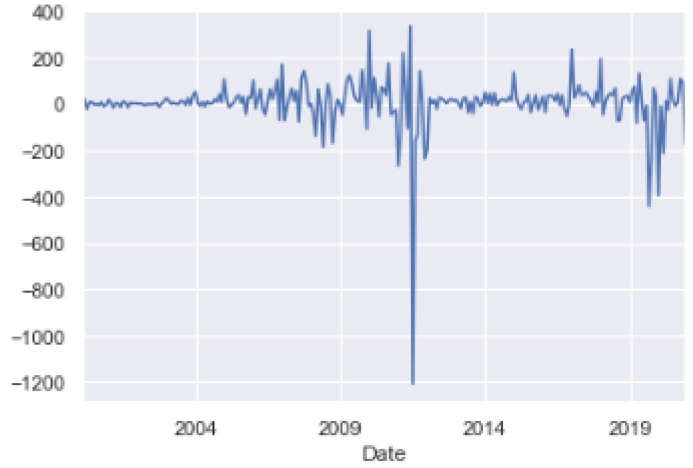
Differencing is performed by subtracting the previous observation from the current observation.

Differencing can help stablize the mean of the time series by removing changes in the level of a time series and so eliminating trends and seasonality

data = data - data.shift(1)

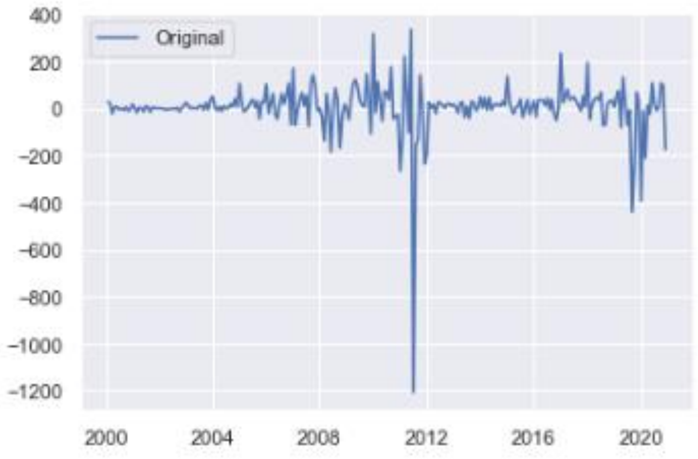
The shift() function is used to shift index by desired number of periods with an optional time freq. When frequency is not passed, shift the index without realigning the data.

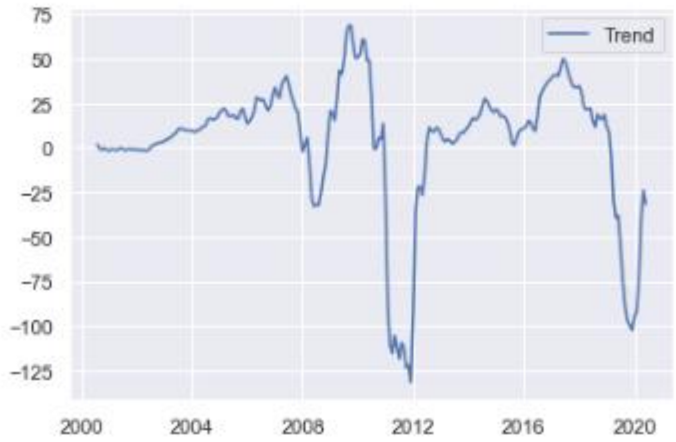
Plot after removing the seasonality

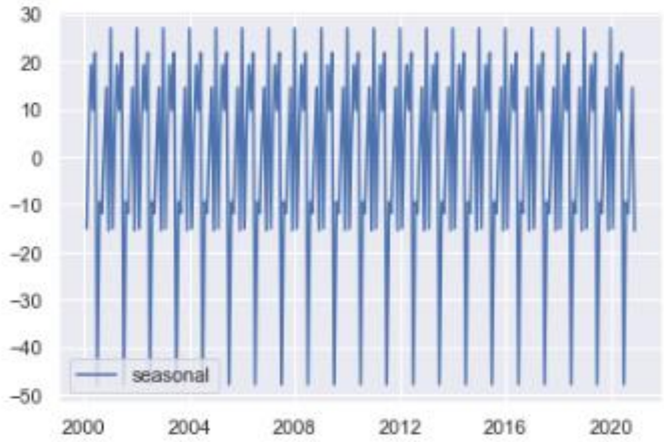


9. Decomposing on new differenced data

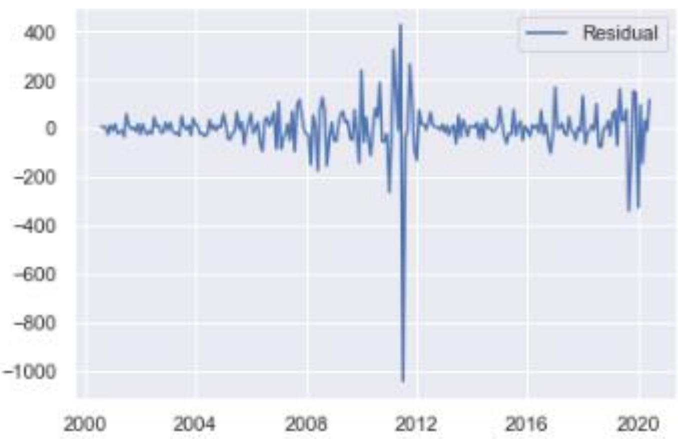
In this approach, both trend and seasonality are modelled separately and the remaining part of the series is returned.





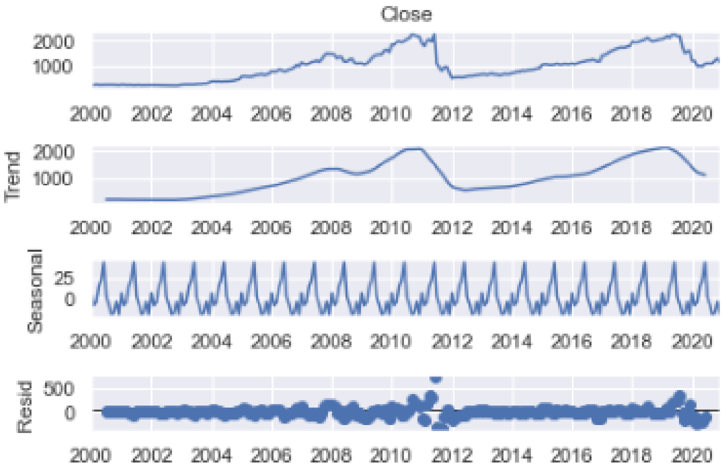



```
<matplotlib.legend.Legend at 0x171f50d7340>
```



After decomposing now again we can observe a lot of difference in the graph now and before differencing the data (removing seasonality from data)using shift() function.

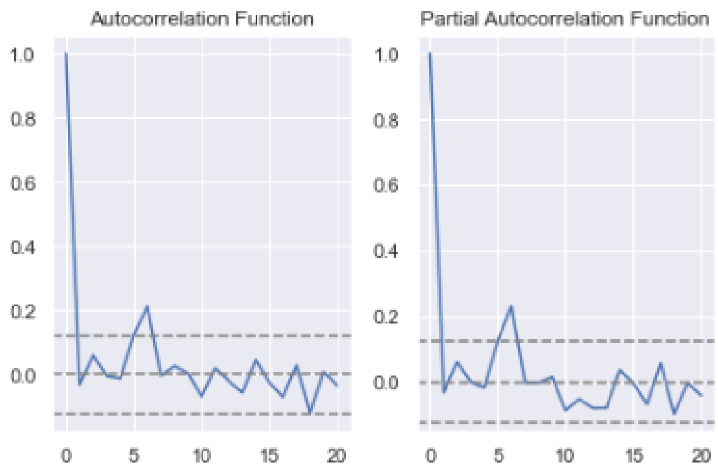
10. Before removing seasonality :



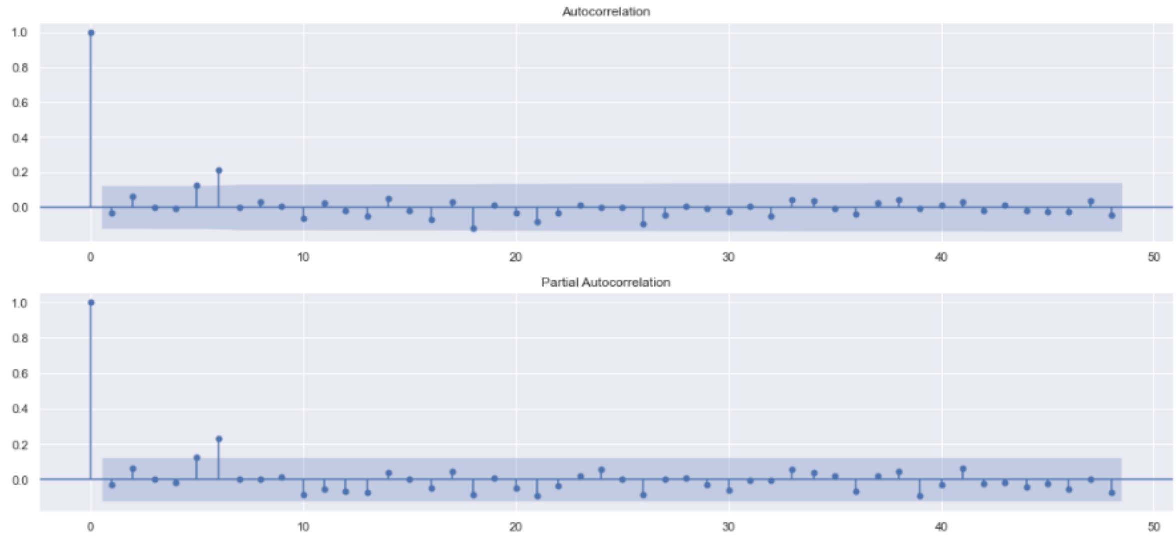
After removing seaonality:

11. ACF and PACF :

ACF and PACF plots indicate that an MA (1) model would be appropriate for the time series because the **ACF cuts after (20) lags while the PACF shows a slowly decreasing trend.**



Normal plot with 48 lags

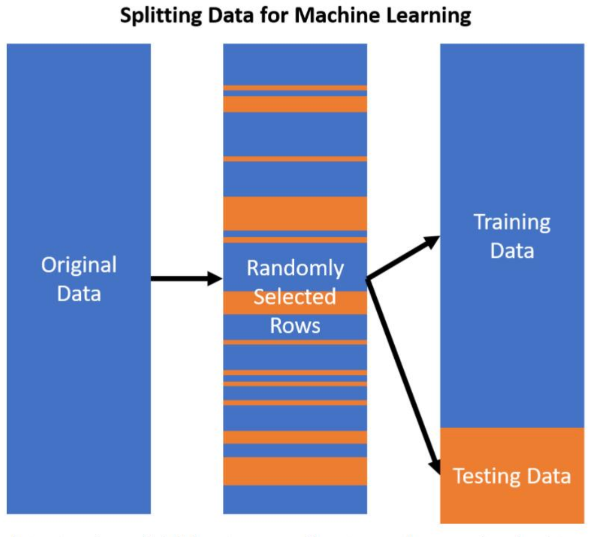


The values are same for both because **both measure the correlation between data points at time t with data points at time t – 1 and** it describes how well the present value of the series is related with its past values.

12. Creating Training and Test dataset

Creating a Training dataset with 95% of the original dataset and testing dataset with remaining 5% of original dataset.

```
size = int(len(data)*0.95)
train,test = data[0:size],data[size:len(data)]
```



13. Model building:

ARIMA MODEL :

An autoregressive integrated moving average, or ARIMA, is **a statistical analysis model that uses time series data to either better understand the data set or to predict future trends**. A statistical model is autoregressive if it predicts future values based on past values.

Time Series Forecasting using ARIMA. We will use ARIMA for forecasting our time series. ARIMA is also denoted as ARIMA (p, d, q)

p is the number of autoregressive terms,

d is the number of non- seasonal differences needed for stationarity.

q is the number of lagged forecast errors in the prediction equation.

Parameter selection (ARIMA) :

We are tryong to select the best fit parameters using rcParams where we got around 64 combinations of parameters and its values, we generated it using the data and seasonal_pdq list generated a while ago.

```
ARIMA(0, 0, 0)x(0, 0, 0, 12)12 - AIC:2901.939764746572
ARIMA(0, 0, 0)x(0, 0, 1, 12)12 - AIC:2768.6727783307833
ARIMA(0, 0, 0)x(0, 1, 0, 12)12 - AIC:2911.2573317236775
ARIMA(0, 0, 0)x(0, 1, 1, 12)12 - AIC:2650.3396465851565
ARIMA(0, 0, 0)x(1, 0, 0, 12)12 - AIC:2779.9582291574134
ARIMA(0, 0, 0)x(1, 0, 1, 12)12 - AIC:2770.672322499722
ARIMA(0, 0, 0)x(1, 1, 0, 12)12 - AIC:2718.4776470367838
ARIMA(0, 0, 0)x(1, 1, 1, 12)12 - AIC:2652.173934055331
ARIMA(0, 0, 1)x(0, 0, 0, 12)12 - AIC:2892.496614238038
ARIMA(0, 0, 1)x(0, 0, 1, 12)12 - AIC:2759.1978928773337
ARIMA(0, 0, 1)x(0, 1, 0, 12)12 - AIC:2901.092707730828
ARIMA(0, 0, 1)x(0, 1, 1, 12)12 - AIC:2640.9486194925335
ARIMA(0, 0, 1)x(1, 0, 0, 12)12 - AIC:2781.793907814168
```

These are few examples from the data generated.

Now we are supposed to find the parameters with min value and then we are going to use those for stock prediction.

```
order = order_list[min_val]
seasonal_order = param_seasonal_list[min_val]
```

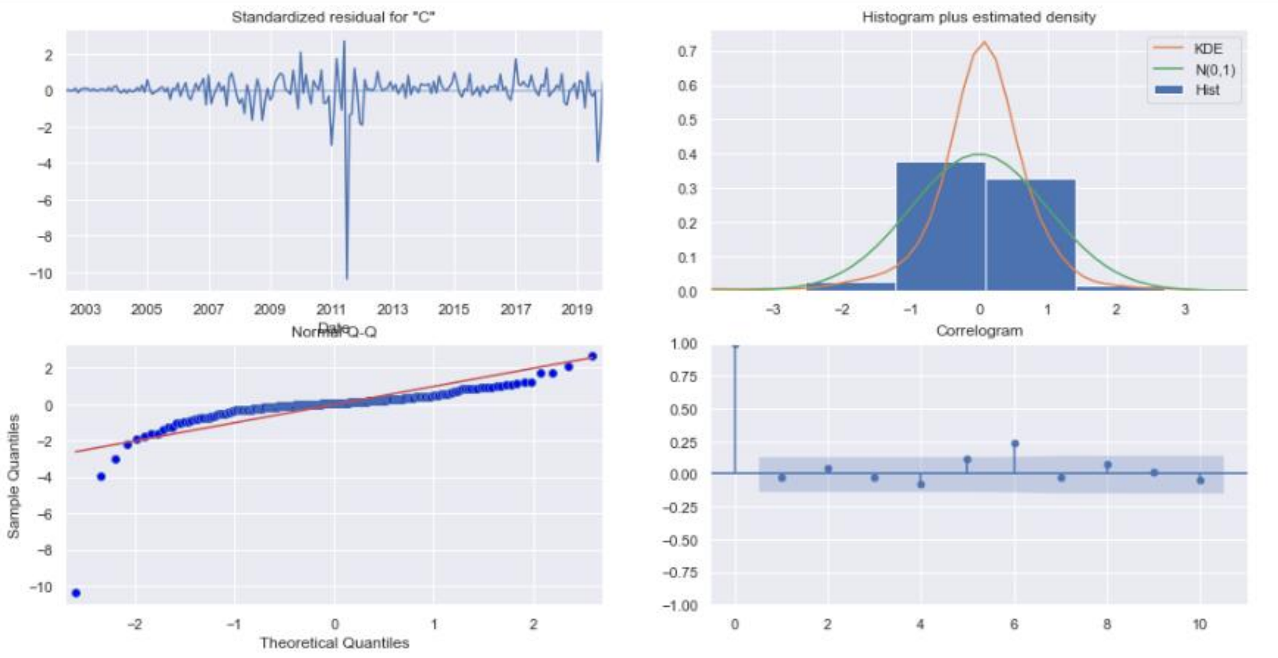
Fitting ARIMA model

```
mod = sm.tsa.statespace.SARIMAX(train,
                                order=order,
                                seasonal_order=seasonal_order,
                                enforce_stationarity=False,
                                enforce_invertibility=False)

results = mod.fit()
print(results.summary().tables[1])
```

```
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ma.L1         -1.0000     86.591     -0.012      0.991    -170.716     168.716
ma.S.L12      -1.0000     86.600     -0.012      0.991    -170.732     168.732
sigma2      1.293e+04      0.006   2.05e+06      0.000    1.29e+04    1.29e+04
==============================================================================
```

This is the plot generated for the diagnostics

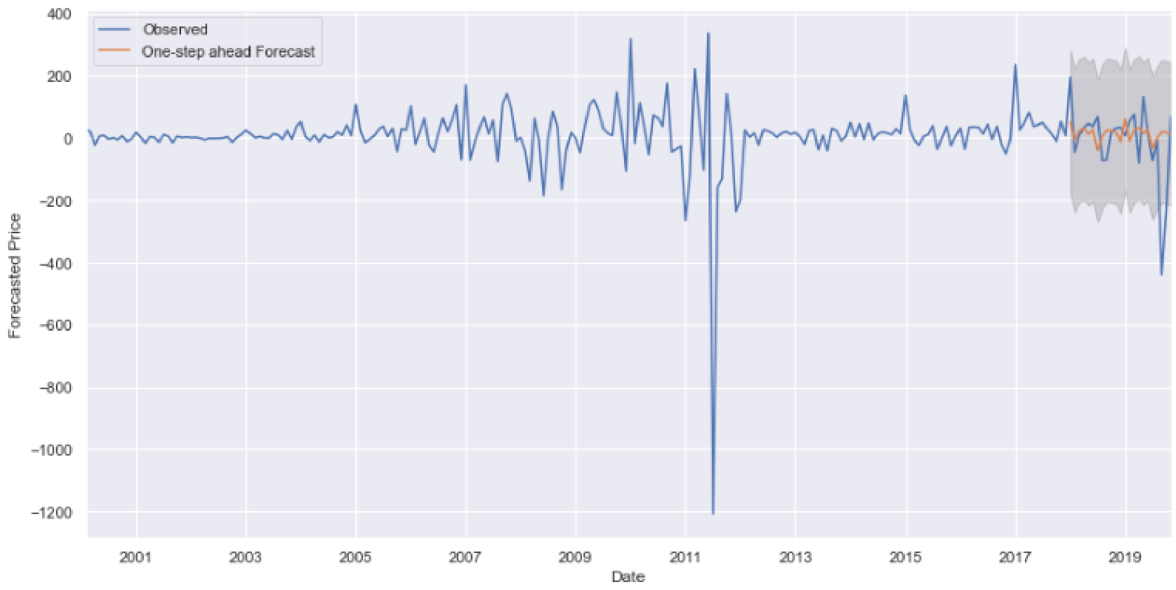


Standard residual is as same as before and in histogram we can find KDE and N(0,1) looks normal and correlogram is as same as the before when we calculated ACF and PACF.

14.Validating Forecasts :

Validating the prediction with the data present in the dataframe to check whether the observation is same as Forecast.

From 2018 to 2019 the prediction/Forecast looks precise where as we have to check where the prediction is not matching that is near to 2020. Lets forcast data for 2020.




```
The Mean squared Error of our forecasts is 38879.55
The Root Mean squared Error of our forecasts is 197.18
```

15.Visualizing the Forecast :

Here from the graph we can say that the prediction is not precise to Forecast from 2019 Dec to 2020 Dec.

```
The Mean squared Error of our forecasts is 93233.35
The Root Mean squared Error of our forecasts is 305.34
```

16. Comaparing predictions:

Date	Actual	Predicted
2019-12-01	29.211556	212.951371
2020-01-01	-394.142584	280.034199
2020-02-01	-9.792246	215.724528
2020-03-01	-211.316845	247.478577
2020-04-01	11.347727	248.441333
2020-05-01	-24.811842	242.568927
2020-06-01	107.934342	248.019998
2020-07-01	16.541630	186.402664
2020-08-01	-9.719130	221.685345
2020-09-01	8.287143	218.804557
2020-10-01	107.128571	232.217157
2020-11-01	93.647619	244.738264
2020-12-01	-174.126190	213.022427

17. Conclusion:

The stock price increases and decreases irrespective of season and in this case I believe this was because of COVID-19 pandemic the stock price was down and that is the reason the algorithm could not predict as expected as it was an unknown factor and does not depends on previous data.

References:

1. https://en.wikipedia.org/ [Theory information]
2. https://www.geeksforgeeks.org/ [Theory & Python packages]
3. https://towardsdatascience.com/ [Theory & Understanding]
4. https://stackoverflow.com/ [Theory & Understanding]
5. https://images.google.com/ [Concepts related images]
6. https://methods.sagepub.com/ [Theory & Understanding]