Software Engineering Project

Supplemental Report 2020

Name Nadia Abouelleil

Student Number 18340116

Time Series Analysis & Forecast on Irish Daily Covid Case Infection Rates

For this project, I wrote my code in Python using Jupyter Notebook. I looked up many youtube tutorials on how to create Time Series Forecasting models, as it wasn't something I was quite familiar with and I wanted to see the necessary steps and methods needed to create the forecast. All the tutorials I could find used Jupyter Notebook. The main steps used in this project:

- Data Cleaning and Analysis
- Checking Stationarity
- Transformation
- Differencing
- Time Series Components
- Finding ACF & PACF
- ARIMA Modeling
- Forecast

I sourced my data from https://data.gov.ie/dataset/covidstatisticsprofilehpscirelandopendata/resource/7b2f557c-1f08-4c3d-bb04-79d2f4aed949. However, this data file contained a lot of information that I didn't find necessary for the project so I took out all the columns, except the "Date", "ConfirmedCovidCases" and "TotalConfirmedCovidCases", using microsoft excel. The data file uses values starting from 2/29/2020 up until 26/8/2020.

I created 2 forecasting graphs for the following datasets:

1) **Confirmed Covid Cases** - this used the data given to predict the future values of the cases that occurred per day
2) **Total Confirmed Covid Cases** - this used the data given to predict the future values of the total number of cases

Both forecasts have the same code. The only real difference is labelling the graphs and reading in the different files since they are 2 different datasets.

Here is the explanation of my code using the Total Confirmed Covid Cases code:

First, necessary libraries are imported.

```
import pandas as pd
import numpy as np

import matplotlib.pylab as plt
%matplotlib inline
from matplotlib.pylab import rcParams
from datetime import datetime
import warnings
from statsmodels.tsa.arima_model import ARMA
import statsmodels.api as sm
warnings.filterwarnings('ignore')
```

Then a function to read in the data files and another function to print out the size of the data file is used (in this case it was 178 rows and 2 columns).

```
data=pd.read_csv("CovidCasesTotal.csv")
data.shape
```

```
(178, 2)
```

The 'Date' is created as an index for data and then the 1st 5 values of the dataset are printed.

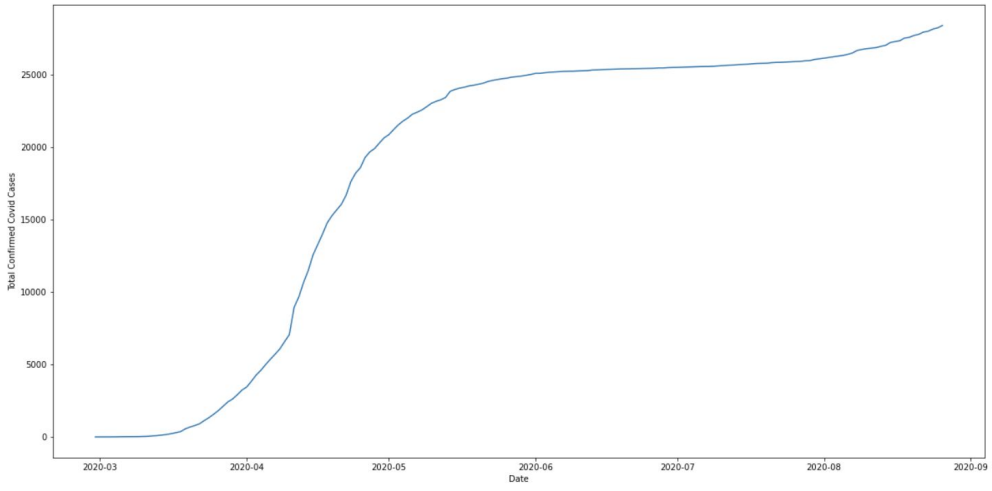
```
data['Date']=pd.to_datetime(data['Date'], infer_datetime_format=True)
data=data.set_index(['Date'])

data.head()
```

Date	TotalConfirmedCovidCases
2020-02-29	1
2020-03-03	2
2020-03-04	6
2020-03-05	13
2020-03-06	18

The 1st graph for the Total Confirmed Covid Cases dataset are printed. The graph shows the exponential growth of the virus infection rates.

```
plt.figure(figsize=(20,10))
plt.xlabel("Date")
plt.ylabel("Total Confirmed Covid Cases")
plt.plot(data)
```

```
[<matplotlib.lines.Line2D at 0x7fa2145ca820>]
```



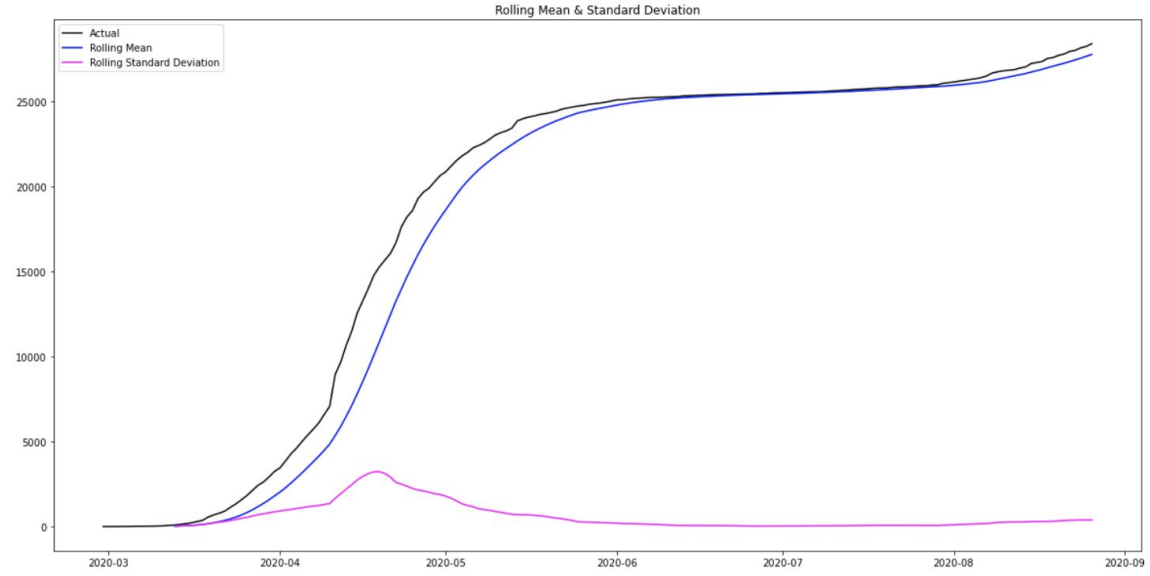
Here, the rolling mean and rolling standard deviation of the time series are calculated so to check the stationarity of the graph. A Time Series is said to be stationary if the mean and variance are constant (does not change over time).

```
rolmean=data.rolling(window=12).mean()
rolstd=data.rolling(window=12).std()
print(rolmean.head(15))
print(rolstd.head(15))
```

```
            TotalConfirmedCovidCases
Date
2020-02-29                       NaN
2020-03-03                       NaN
2020-03-04                       NaN
2020-03-05                       NaN
2020-03-06                       NaN
2020-03-07                       NaN
2020-03-08                       NaN
2020-03-09                       NaN
2020-03-10                       NaN
2020-03-11                       NaN
2020-03-12                       NaN
2020-03-13                 28.416667
2020-03-14                 39.083333
2020-03-15                 53.000000
2020-03-16                 71.083333
            TotalConfirmedCovidCases
Date
2020-02-29                       NaN
2020-03-03                       NaN
2020-03-04                       NaN
2020-03-05                       NaN
2020-03-06                       NaN
2020-03-07                       NaN
2020-03-08                       NaN
2020-03-09                       NaN
2020-03-10                       NaN
2020-03-11                       NaN
2020-03-12                       NaN
2020-03-13                 27.347788
2020-03-14                 38.407997
2020-03-15                 51.703701
2020-03-16                 68.869387
```

The rolling mean and standard deviation are plotted on the graph with the original data. The value of the standard deviation at any point on the graph represents the increase of growth in number of cases on a single day compared to the history of growth on that day.

```
plt.figure(figsize=(20,10))
actual=plt.plot(data, color='black', label='Actual')
mean_6=plt.plot(rolmean, color='blue', label='Rolling Mean')
std_6=plt.plot(rolstd, color='magenta', label='Rolling Standard Deviation')
plt.legend(loc='best')
plt.title('Rolling Mean & Standard Deviation')
plt.show(block=False)
```



The Dickey-Fuller (ADF) Test is used as another method of checking stationarity. The null hypothesis of the ADF test is that the Time seris is nonstationary. We reject the null hypothesis if the p-value is less than 0.05 (significance level). In this case the P-value is 0.139653, which is greater than 0.05. Therefore, we fail to reject the null hypothesis meaning the Time series is nonstationary.

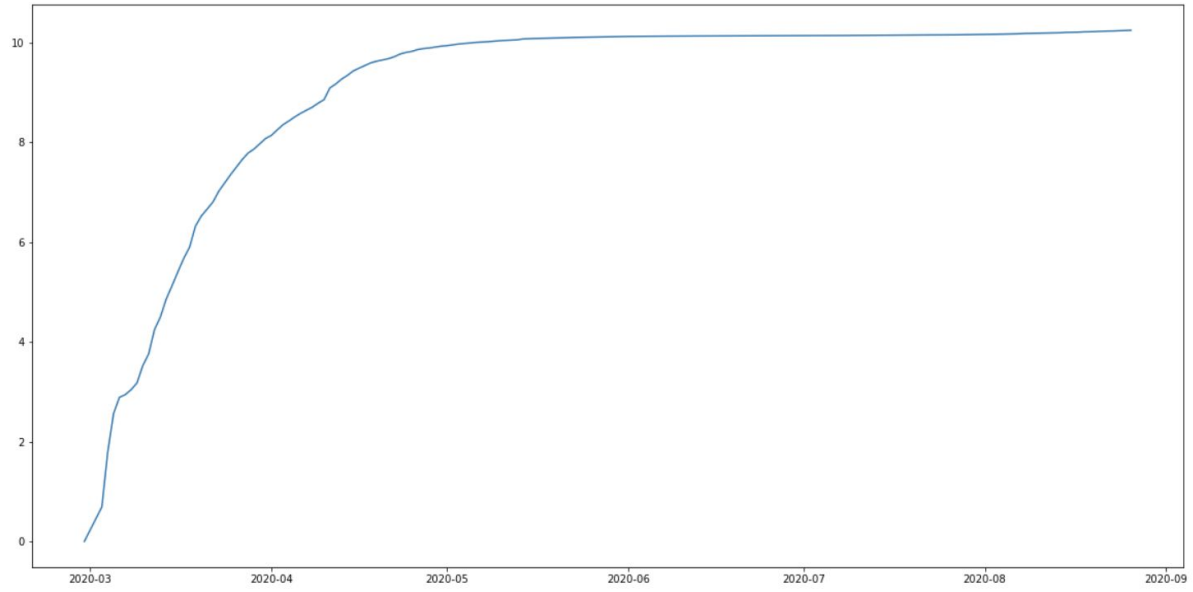
```
from statsmodels.tsa.stattools import adfuller
print('Dickey-Fuller Test: ')
dftest=adfuller(data['TotalConfirmedCovidCases'], autolag='AIC')
dfoutput=pd.Series(dftest[0:4], index=['Test Statistic','P-value','Lags Used','No. of Observations Used'])
for key,value in dftest[4].items():
    dfoutput['Critical Value (%s)'%key] = value
print(dfoutput)
```

```
Dickey-Fuller Test:
Test Statistic                -2.407420
P-value                        0.139653
Lags Used                      4.000000
No. of Observations Used     173.000000
Critical Value (1%)           -3.468726
Critical Value (5%)           -2.878396
Critical Value (10%)          -2.575756
dtype: float64
```

To make the time series stationary, the data is logged first. The result is plotted below.

```
plt.figure(figsize=(20,10))
data_log=np.log(data)
plt.plot(data_log)
```

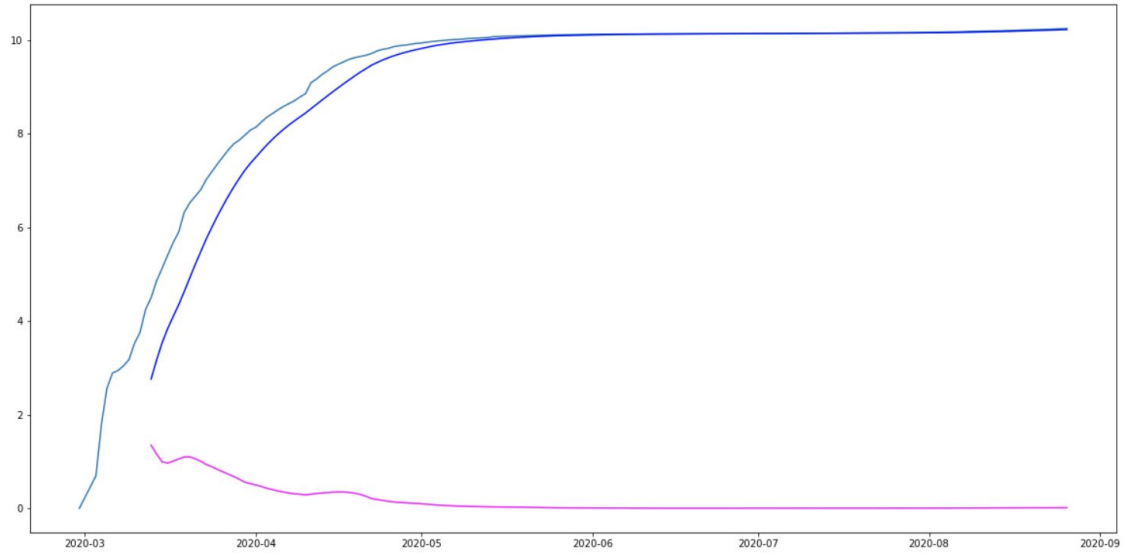
```
[<matplotlib.lines.Line2D at 0x7fa1fa4c0760>]
```



The rolling mean and standard deviation of the now logged time series is calculated to check the stationarity. As you can see, it still is not stationary.

```
plt.figure(figsize=(20,10))
MAvg=data_log.rolling(window=12).mean()
MStd=data_log.rolling(window=12).std()
plt.plot(data_log)
plt.plot(MAvg, color='blue')
plt.plot(MStd, color='magenta')
```

```
[<matplotlib.lines.Line2D at 0x7fa2136462e0>]
```



The 2nd method used to make the time series stationary is the differencing method. In this case, it is the logged data minus the moving average. The 1st 5 values of the result are printed.

```
data_log_diff=data_log-MAvg
data_log_diff=data_log_diff.dropna()
data_log_diff.head()
```

Date	TotalConfirmedCovidCases
2020-03-13	1.737884
2020-03-14	1.692902
2020-03-15	1.593259
2020-03-16	1.569248
2020-03-17	1.579513

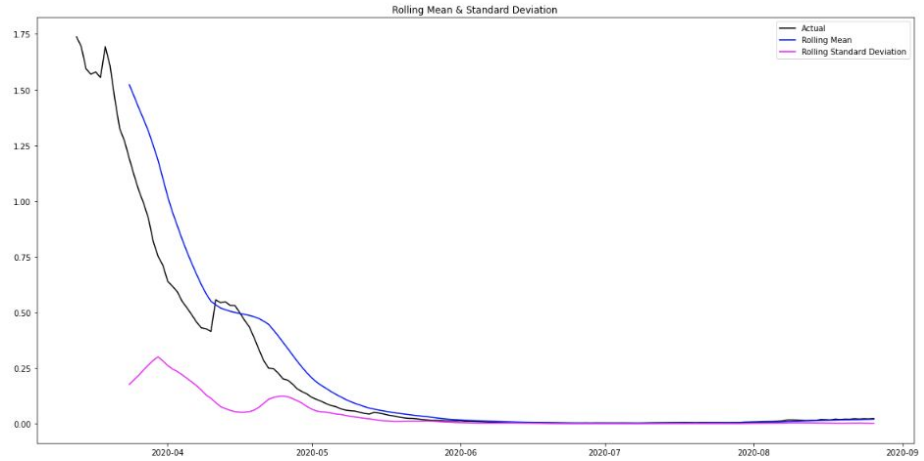
A function is defined to check the stationarity of the tests. This includes the rolling mean and standard deviation methods, as well as the Dickey-Fuller Test.

```
stationarity(timeseries):

rolmean=timeseries.rolling(window=12).mean()
rolstd=timeseries.rolling(window=12).std()

plt.figure(figsize=(20,10))
actual=plt.plot(timeseries, color='black', label='Actual')
mean_6=plt.plot(rolmean, color='blue', label='Rolling Mean')
std_6=plt.plot(rolstd, color='magenta', label='Rolling Standard Deviation')
plt.legend(loc='best')
plt.title('Rolling Mean & Standard Deviation')
plt.show(block=False)

print('Dickey-Fuller Test: ')
dftest=adfuller(timeseries['TotalConfirmedCovidCases'], autolag='AIC')
dfoutput=pd.Series(dftest[0:4], index=['Test Statistic','p-value','Lags Used','No. of Observations Used'])
for key,value in dftest[4].items():
    dfoutput['Critical Value (%s)'%key] = value
print(dfoutput)
```

Result: Our data should now be stationary

The stationarity of the differenced data is then checked. Using the Rolling method, we can see that the mean and standard deviation vary at the start than stop varying at 2020-06. The ADF method shows the p-value is now 0.000006 rather than 0.139653, which is what it was. This means we reject the null hypothesis since the p-value is less than 0.5. Therefore, the data should now be stationary.

```
1 stationarity(data_log_diff)
```

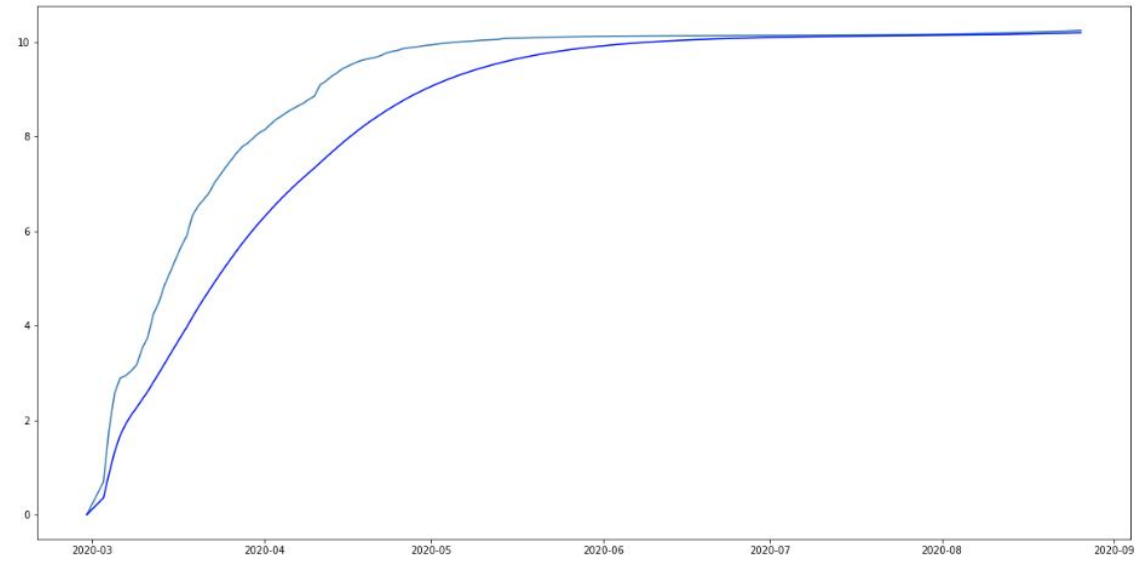



```
Dickey-Fuller Test:
Test Statistic                -5.277426
p-value                        0.000006
Lags Used                     13.000000
No. of Observations Used     153.000000
Critical Value (1%)           -3.473830
Critical Value (5%)           -2.880623
Critical Value (10%)          -2.576945
dtype: float64
```

Trend stationarity is then checked and an exponential transformation on the data is performed.

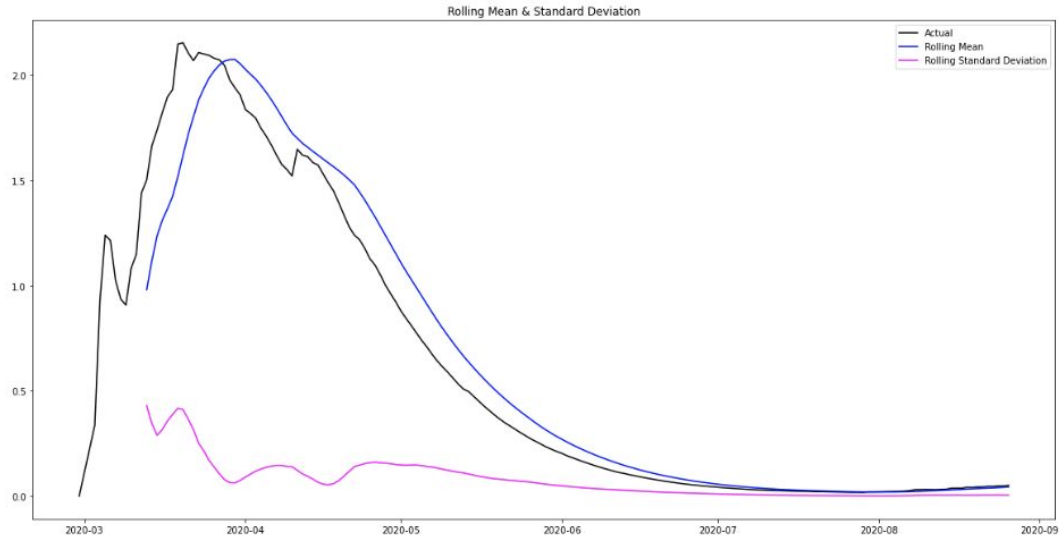
```
1 plt.figure(figsize=(20,10))
2 exp_data=data_log.ewm(halflife=12, min_periods=0, adjust=True).mean()
3 plt.plot(data_log)
4 plt.plot(exp_data, color='blue')
```

```
[<matplotlib.lines.Line2D at 0x7fa1f97d9520>]
```



The differencing method is applied on the data again since log transformation is non-stationary. The result is plotted. We can now see the time series is stationary again.

```
exp_data_diff=data_log-exp_data
stationarity(exp_data_diff)
```

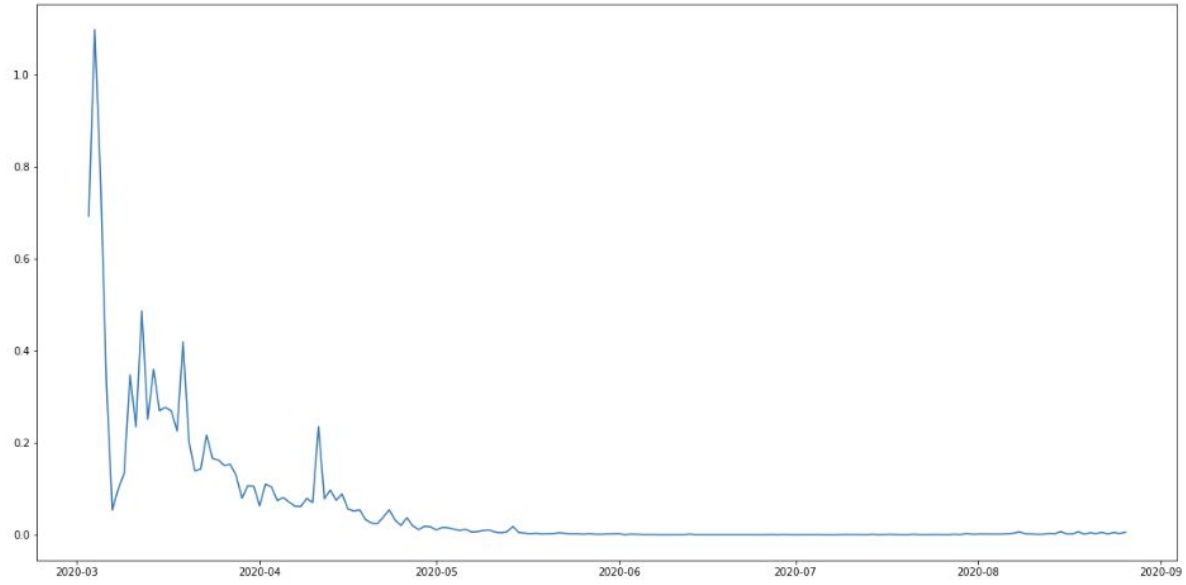



```
Dickey-Fuller Test:
Test Statistic                -4.945943
p-value                        0.000028
Lags Used                     14.000000
No. of Observations Used     163.000000
Critical Value (1%)           -3.471119
Critical Value (5%)           -2.879441
Critical Value (10%)          -2.576314
dtype: float64
```

The values of the time series must then be shifted in preparation of the forecast. The results are then plotted.

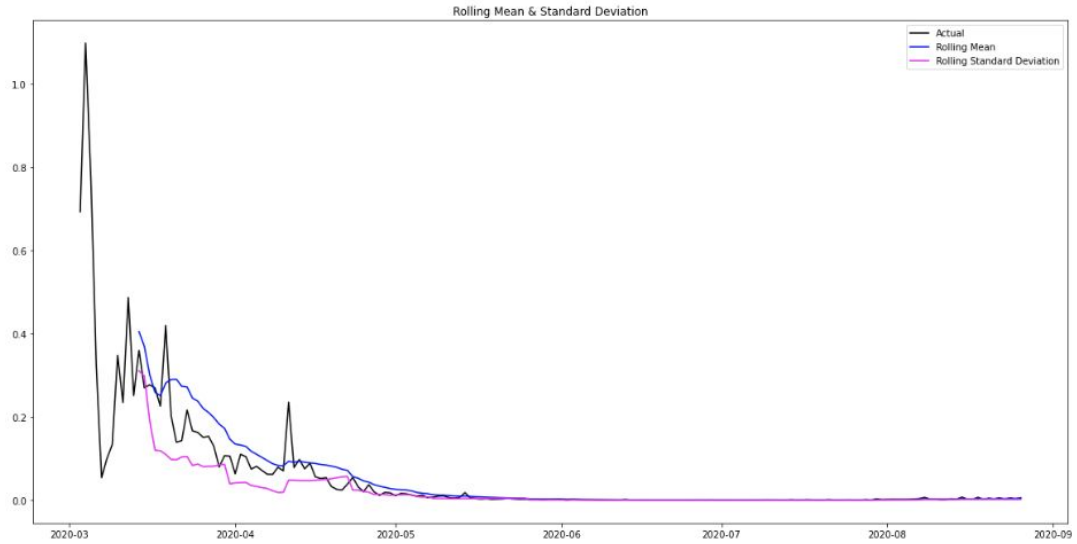
```
plt.figure(figsize=(20,10))
data_shift=data_log-data_log.shift()
plt.plot(data_shift)
```

```
[<matplotlib.lines.Line2D at 0x7fa1f603ac40>]
```



The stationarity of the result is then checked.

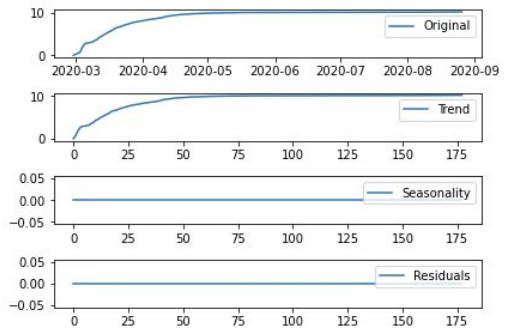
```
data_shift=data_shift.dropna()
stationarity(data_shift)
```




```
Dickey-Fuller Test:
Test Statistic                -8.573003e+00
p-value                        8.102153e-14
Lags Used                      1.400000e+01
No. of Observations Used       1.620000e+02
Critical Value (1%)           -3.471374e+00
Critical Value (5%)           -2.879552e+00
Critical Value (10%)          -2.576373e+00
dtype: float64
```

The time series is then decomposed into its components (Trend, Seasonality, Residual)

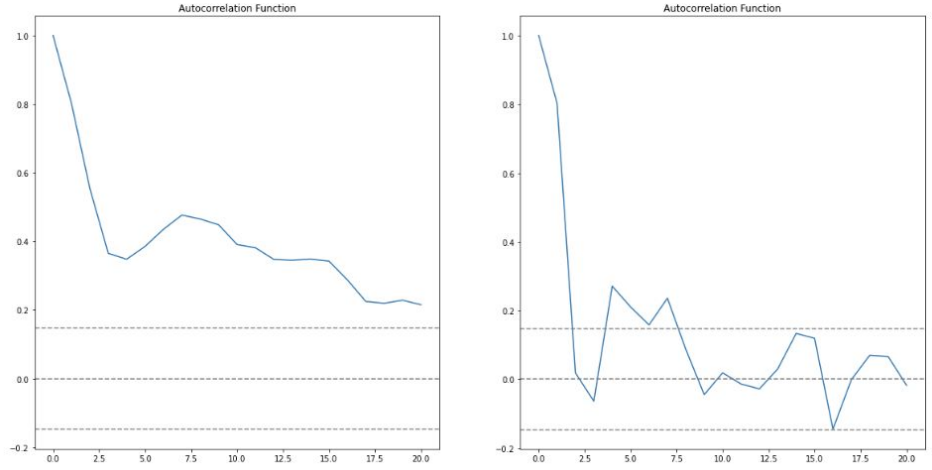
```
from statsmodels.tsa.seasonal import seasonal_decompose
decomp=seasonal_decompose(data_log.values, freq=1)

trend=decomp.trend
seasonal=decomp.seasonal
residual=decomp.resid

plt.subplot(411)
plt.plot(data_log, label='Original')
plt.legend(loc='best')
plt.subplot(412)
plt.plot(trend, label='Trend')
plt.legend(loc='best')
plt.subplot(413)
plt.plot(seasonal, label='Seasonality')
plt.legend(loc='best')
plt.subplot(414)
plt.plot(residual, label='Residuals')
plt.legend(loc='best')
plt.tight_layout()
```



To find the q and p values for the ARIMA model, the ACF (Auto-Correlation Function) and PACF (Partial Auto-Correlation Function) graphs must be plotted first.

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

lag_acf=acf(data_shift, nlags=20)
lag_pacf=pacf(data_shift, nlags=20, method='ols')

plt.figure(figsize=(20,10))
plt.subplot(121)
plt.plot(lag_acf)
plt.axhline(y=0,linestyle='--',color='grey')
plt.axhline(y=-1.96/np.sqrt(len(data_shift)),linestyle='--',color='grey')
plt.axhline(y=1.96/np.sqrt(len(data_shift)),linestyle='--',color='grey')
plt.title('Autocorrelation Function')

plt.subplot(122)
plt.plot(lag_pacf)
plt.axhline(y=0,linestyle='--',color='grey')
plt.axhline(y=-1.96/np.sqrt(len(data_shift)),linestyle='--',color='grey')
plt.axhline(y=1.96/np.sqrt(len(data_shift)),linestyle='--',color='grey')
plt.title('Autocorrelation Function')
```

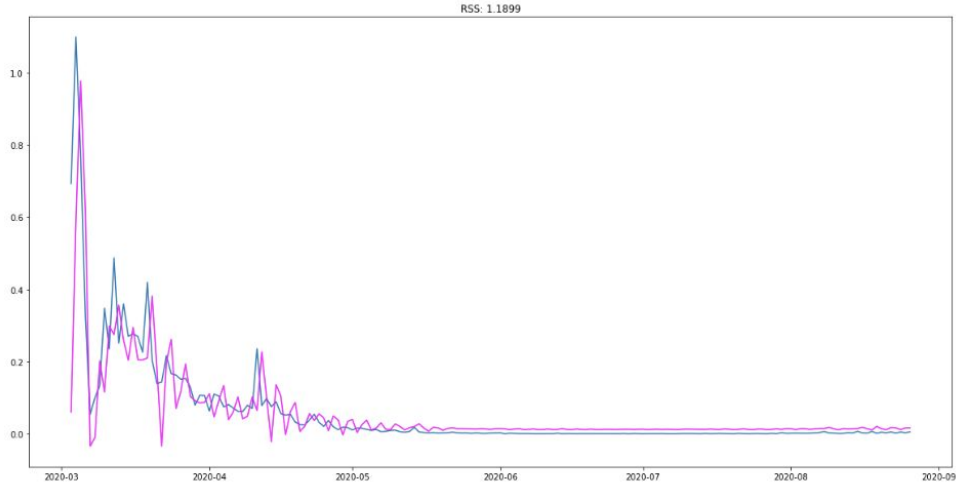
```
Text(0.5, 1.0, 'Autocorrelation Function')
```



The ARIMA model is created with p=2, q=2 and I=1.

```
smodels.tsa.arima_model import ARIMA

e(figsize=(20,10))
MA(data_log, order=(2,1,2))
odel.fit(disp=-1)
data_shift)
results.fittedvalues, color='magenta')
('RSS: %.4f'% sum((results.fittedvalues-data_shift['TotalConfirmedCovidCases'])**2))
otting ARIMA model')
```

```
plotting ARIMA model
```



The following functions are then used to get the predicted values for the forecast.

- Predicted values from ARIMA model in difference form

```
predictions=pd.Series(results.fittedvalues, copy=True)
print(predictions.head())
```

```
Date
2020-03-03    0.060115
2020-03-04    0.585762
2020-03-05    0.977139
2020-03-06    0.617274
2020-03-07   -0.034045
dtype: float64
```

- Inverse Transformation for differencing using cumulative sum

```
predictions_cum_sum=predictions.cumsum()
print(predictions_cum_sum.head())
```

```
Date
2020-03-03    0.060115
2020-03-04    0.645878
2020-03-05    1.623016
2020-03-06    2.240290
2020-03-07    2.206245
dtype: float64
```

- Predicted values in log scale

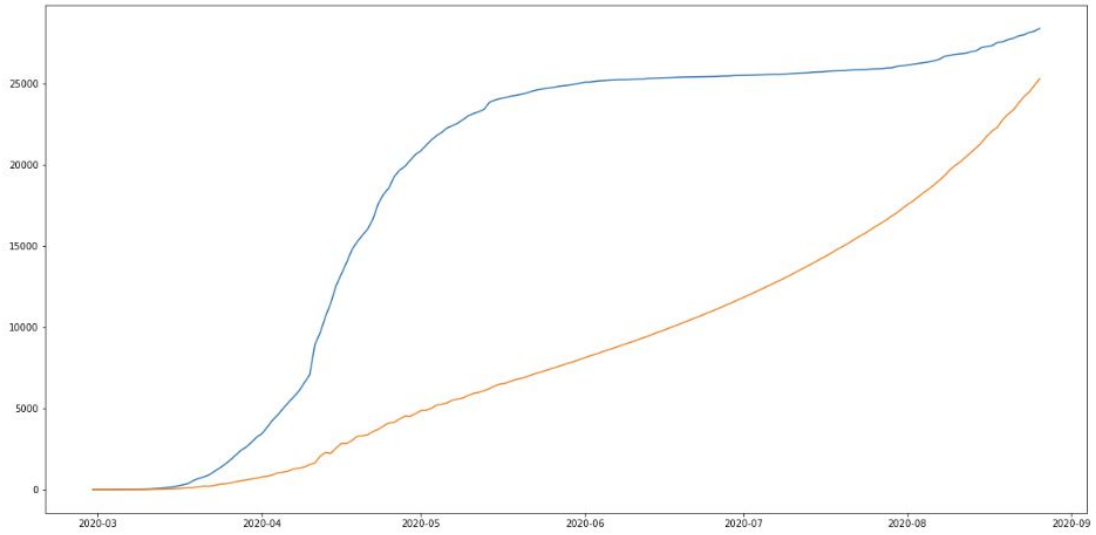
```
predictions_log=pd.Series(data_log['TotalConfirmedCovidCases'].iloc[0], index=data_
predictions_log=predictions_log.add(predictions_cum_sum,fill_value=0)
predictions_log.head()

predictions_log=pd.Series(data_log['TotalConfirmedCovidCases'].iloc[0], index=data_
predictions_log=predictions_log.add(predictions_cum_sum,fill_value=0)
predictions_log.head()
```

```
Date
2020-02-29    0.000000
2020-03-03    0.060115
2020-03-04    0.645878
2020-03-05    1.623016
2020-03-06    2.240290
dtype: float64
```

The log transformation is taken out to plot the actual vs predicted value graph for the forecast.

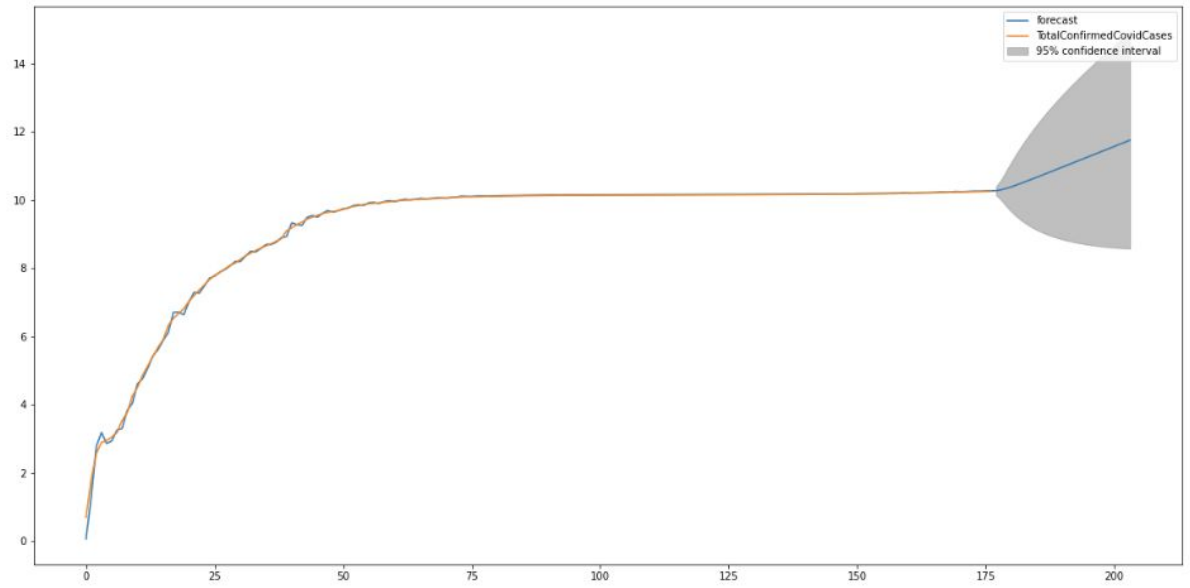
```
predictions_ARIMA=np.exp(predictions_log)
plt.figure(figsize=(20,10))
plt.plot(data)
plt.plot(predictions_ARIMA)
```

```
[<matplotlib.lines.Line2D at 0x7fa213fb2c10>]
```



The results of the Time Series forecast are then plotted with a 95% confidence interval.

```
rcParams['figure.figsize']=20,10
results.plot_predict(1,204)
x=results.forecast(steps=120)
```



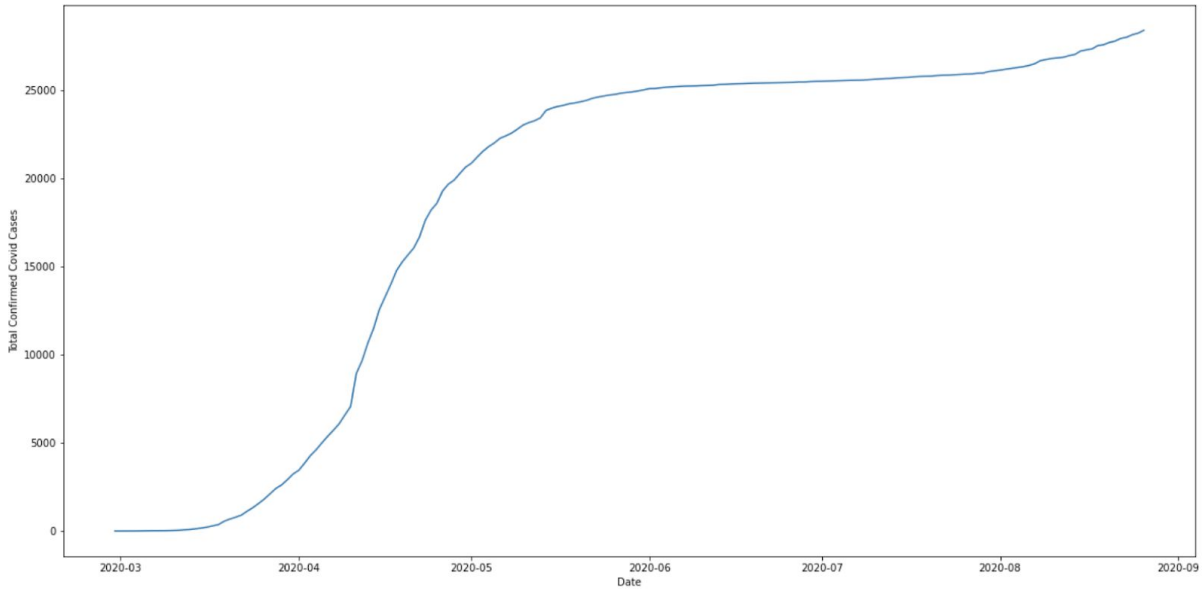
Here are the results of the forecasted data in numeric form:

```
results.forecast(steps=120)
```

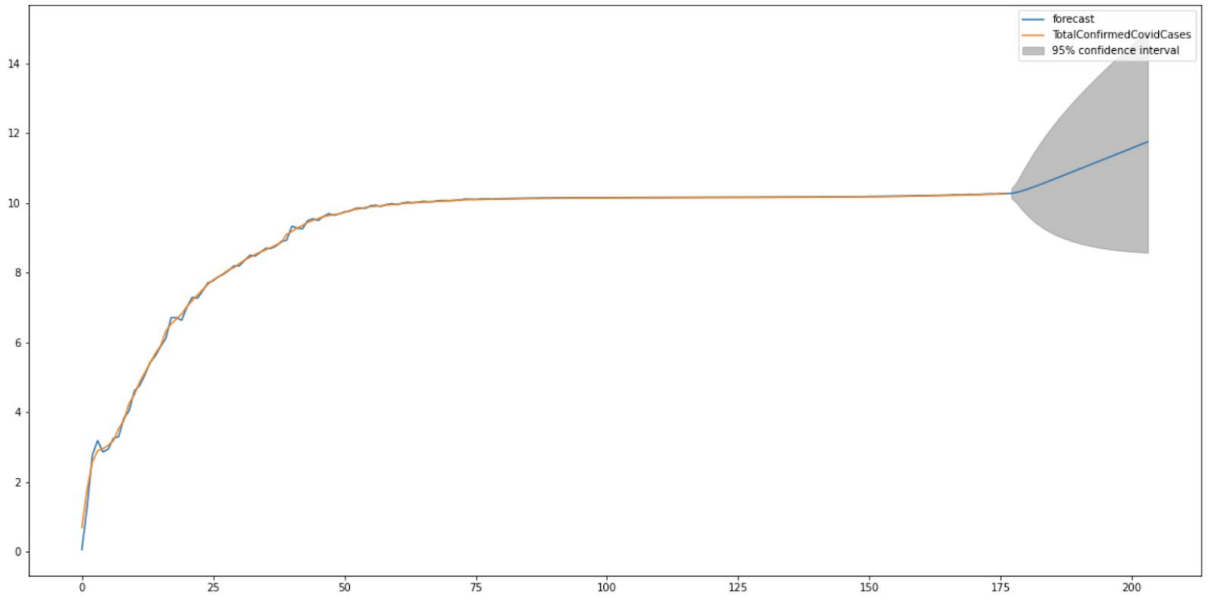
```
(array([10.26742312, 10.29509869, 10.33621472, 10.38440277, 10.4372197 ,
        10.49282733, 10.55016806, 10.60857344, 10.66763545, 10.72710186,
        10.78681744, 10.84668653, 10.90665021, 10.96667215, 11.02672999,
        11.08680995, 11.14690354, 11.20700552, 11.26711268, 11.32722302,
        11.38733532, 11.44744884, 11.50756309, 11.56767781, 11.62779281,
        11.68790799, 11.74802327, 11.80813862, 11.86825401, 11.92836943,
        11.98848486, 12.0486003 , 12.10871574, 12.16883119, 12.22894664,
        12.2890621 , 12.34917755, 12.409293  , 12.46940846, 12.52952391,
        12.58963937, 12.64975482, 12.70987028, 12.76998573, 12.83010119,
        12.89021664, 12.9503321 , 13.01044755, 13.07056301, 13.13067846,
        13.19079392, 13.25090937, 13.31102483, 13.37114029, 13.43125574,
        13.4913712 , 13.55148665, 13.61160211, 13.67171756, 13.73183302,
        13.79194847, 13.85206393, 13.91217938, 13.97229484, 14.03241029,
        14.09252575, 14.1526412 , 14.21275666, 14.27287211, 14.33298757,
        14.39310302, 14.45321848, 14.51333393, 14.57344939, 14.63356484,
        14.6936803 , 14.75379575, 14.81391121, 14.87402666, 14.93414212,
        14.99425757, 15.05437303, 15.11448848, 15.17460394, 15.2347194 ,
        15.29483485, 15.35495031, 15.41506576, 15.47518122, 15.53529667,
        15.59541213, 15.65552758, 15.71564304, 15.77575849, 15.83587395,
        15.8959894 , 15.95610486, 16.01622031, 16.07633577, 16.13645122,
        16.19656668, 16.25668213, 16.31679759, 16.37691304, 16.4370285 ,
```

Original vs Forecasted Time Series Graphs

Total Confirmed Covid Cases



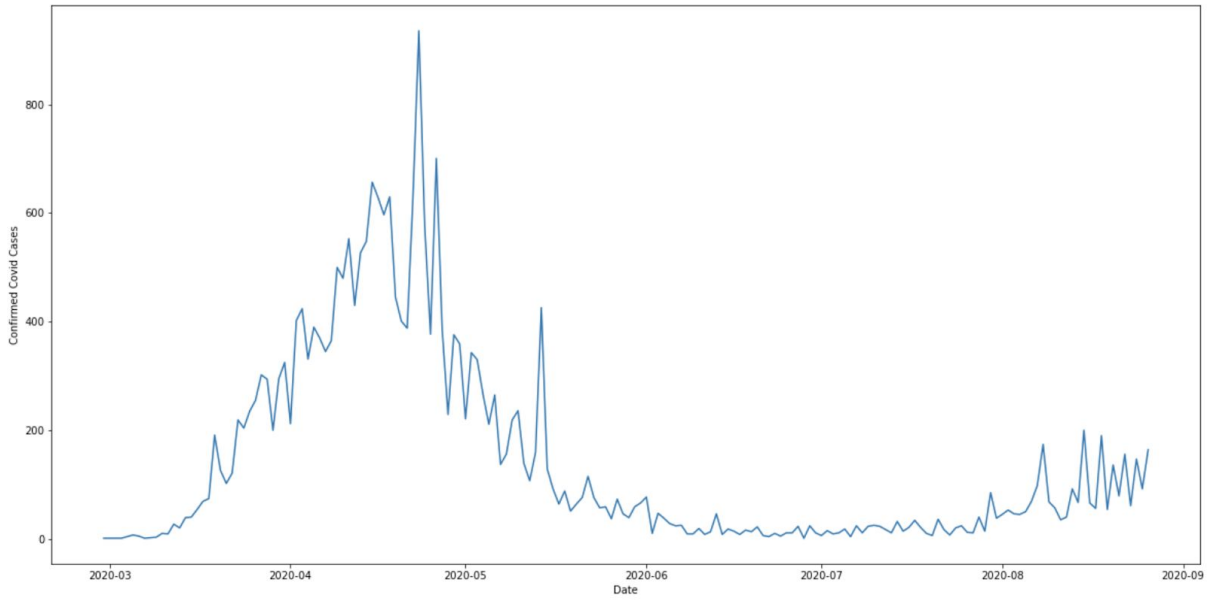
Original Time Series Graph



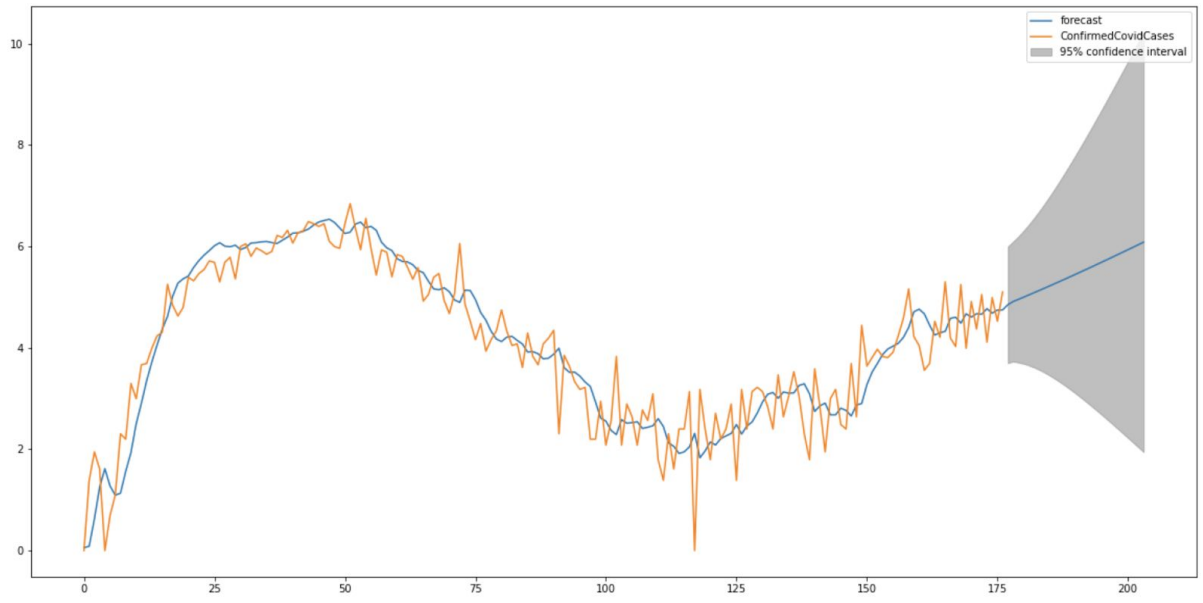
Forecasted Time Series Graph

We can see in the graphs how there's an exponential increase in the infection rates and then there is a stationary phase. However, the past month shows that there was a growth in infection rates and you can see in the forecast that the rate is more likely to spike up again rather than decrease/stay stationary.

Confirmed Covid Cases



Original Time Series Graph



Forecasted Time Series Graph

Here as well we can see how the 95% confidence interval suggests that the daily cases are most likely to rise than decrease. The angle of the forecast in the 2 graphs are practically identical, which is expected considering the datasets. And the 95% interval of the 2 are also a similar shape (the top half spikes up more while bottom halk leans more to a stationary phase.