BTP seminar report Spring 2020

Karanam Tejendhar

April 22, 2020

Introduction
Date: 24/1/2020

Things I have already worked on:

1. I have studied Time Series analysis from the book : Introduction to Time Series Analysis and Forecasting by Douglas C. Montgomery, which uses a statistical approach.
2. I send the notes I used to learn statistical approach in Whatsapp.
3. Did some work on Fourier transforms and Fast Fourier transforms, work which is generally statistical and has no application of Machine Learning.

How not to use Machine Learning on time series:

1. There exists some data like"Stock index" where the future is completely random as in a "stochastic process". In these cases it is meaningless to say "predicted from past data" as the future data is completely independent of past data.

2. However, the interesting thing comes here. Even as mentioned in the above point, we get good predictions. Why? This is because, there are two components helping us in prediction.
(a) input features containing useful information, and
(b) changes in output features over time.
This part (b) makes our prediction task difficult. Here, (when there is no information we get from input features as in part (a)), the changes in output features give a term called "auto correlation", which means value "f(t+a)" will be equal to value at "f(t)" where "a" is some offset. This is clearly wrong.????(because it has nothing to do with input feature analysis. Just using past output values in future which we can't tell.)

3. Common error metrics such as mean percentage error or R2 accuracy will show good high prediction for all cases even sometimes in the cases mentioned above, which is incorrect.

MAIN POINT: ACCURACY METRICS CAN BE VERY MISLEADING WHEN USED INCORRECTLY.

5. Defining the model to predict the difference in values between time steps rather than the value itself, is a much stronger test of the model's predictive powers. (Copied this statement from internet).

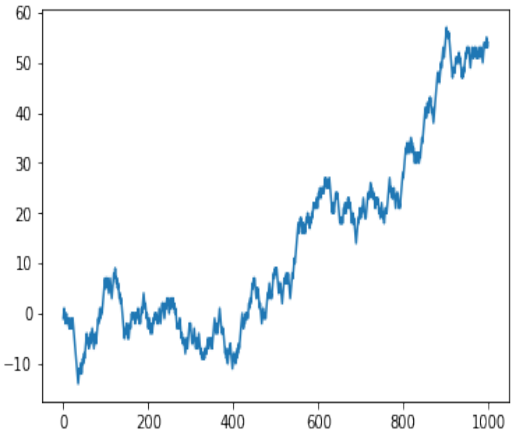
6. We should not make a non-stationary data to stationary data while applying models because this may give some learn able structure which is absent in the original non-stationary data. ????

CONCLUSION: One can be easily fooled by predicting future outcomes of a random process, where predicting future is impossible but still getting high accuracy from common accuracy metrics.??? Because this accuracy is due to repetition of previous values with some buffer but not from input features.?? EXPLAIN mathematically, logically.

Explanation:

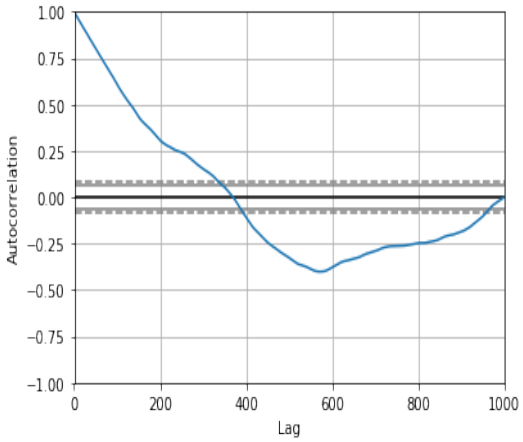
Suppose lets assume a random process.

$f(0) = 0$ and $f(i+1) = f(i) + R$ where $i >= 0$ and R belongs to $-1, 1$ and R=1 or -1 is completely random.
now



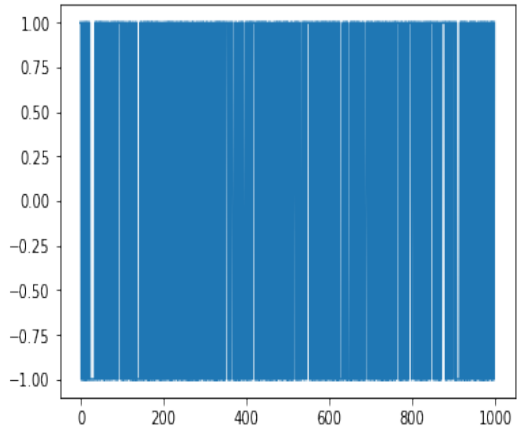
used pandas for graph generation
where y axis is f values and x axis have respective i or t
The process used to generate the series depends from one-time step to the next. This dependence provides some consistency from step-to-step , rather than the large jumps that a series of independent, random numbers.

Lets see the auto-correlation diagram (correlogram) of the above values in diagram obtained.



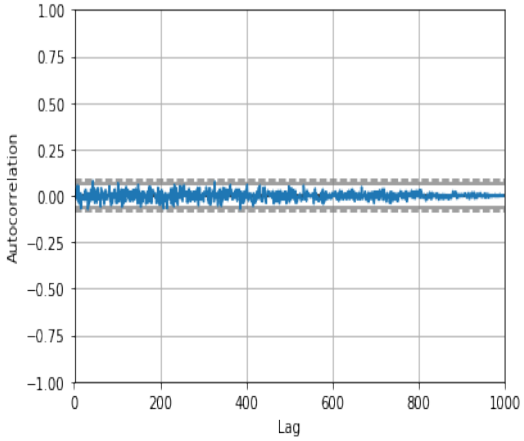
used pandas for graph generation
where y axis is correlation values and x axis respective lag.

Now lets replace every observation with the difference between itself and its next value.



used pandas for graph generation
where y axis is difference values and x axis respective i or t.
We can clearly see there is no clear structure to learn.

Now the updated auto correlation graph is



used pandas for graph generation
where y axis is correlation values and x axis respective lag.

All correlations are small and close to zero.

The variation from one step to another step is always one , either in positive way or negative. So the mean square of this expected error is always 1.

Time Series and some statistical techniques

1. A time series is a sequential set of data points, measured typically over successive times.

2. It is mathematically defined as a set of vectors x(t), t = 0,1,2,... where t represents the time elapsed.

3. The variable x(t) is treated as a random variable. The measurements taken during an event in a time series are arranged in a proper chronological order.

4. A time series containing records of a single variable is termed as uni variate, and more than one variable a multivariate.

Stationarity

1. A Time Series is said to be stationary if its statistical properties such as mean, variance remain constant over time.

2. Most of the Time Series models work on the assumption that the TS is stationary. Major reason for this is that there are many ways in which a series

can be non-stationary, but only one way for stationarity.

3. Intuitively, we can say that if a Time Series has a particular behaviour over time, there is a very high probability that it will follow the same in the future.

We can check stationarity using the following:

1. ACF and PACF plots: If the time series is stationary, the ACF/PACF plots will show a quick drop-off in correlation after a small amount of lag between points.

2. Plotting Rolling Statistics: We can plot the moving average or moving variance and see if it varies with time. Moving average/variance is for any instant 't'.

If the rolling mean and Standard deviation are not constant with respect to time , then time series is hence not stationary

3. Augmented Dickey-Fuller Test: This is one of the statistical tests for checking stationarity. Here the null hypothesis is that the TS is non-stationary. The test results comprise of a Test Statistic and some Critical Values for difference confidence levels. If the 'Test Statistic' is less than the 'Critical Value', we can reject the null hypothesis and say that the series is stationary.

Making Time Series Stationary

There are 2 major reasons behind non-stationaruty of a TS:

1. Trend – varying mean over time. For eg, in this case we saw that on average, the number of passengers was growing over time.

2.Seasonality – variations at specific time-frames. eg people might have a tendency to buy cars in a particular month because of pay increment or festivals.

So , we use transformations. To penalize higher values more than smaller values.

possible transformations:

1. log,square root, cube root etc
2. Exponential tranformation
3. Box Cox transformation

Techniques to remove Trend - Smoothing

1. Moving average:

We take average of 'k' consecutive values depending on the frequency of time series. A drawback in this particular approach is that the time-period has to be strictly defined.

2. Exponentially weighted moving average:

To overcome the problem of choosing a defined window in moving average, we can use exponential weighted moving average

We take a 'weighted moving average' where more recent values are given a higher weight.

There can be many technique for assigning weights. A popular one is exponentially weighted moving average where weights are assigned to all the previous values with a decay factor.

We will convert our data into y_t according to a function defined .
$y_t = j(t; \beta) + \epsilon_t$

where β is the vector of unknown parameters and ϵ are the uncorrelated errors.

example: SECOND-ORDER EXPONENTIAL SMOOTHING:
$y_t = \beta_0 + \beta_1 * t + \epsilon_t$

AUTOREGRESSIONS

An autoregressive model has dynamics given by

$$y_t = \delta + \phi_1 y_{t-1} + \ldots + \phi_p y_{t-p} + \epsilon_t.$$

AutoReg also permits models with:

Deterministic terms (trend)
n: No deterministic term
c: Constant (default)
ct: Constant and time trend
t: Time trend only

Seasonal dummies (seasonal)
True includes $s - 1$ dummies where s is the period of the time series (e.g., 12 for monthly)

Exogenous variables (exog)
A DataFrame or array of exogenous variables to include in the model Omission of selected lags (lags)
If lags is an iterable of integers, then only these are included in the model.

The complete specification is

$$y_t = \delta_0 + \delta_1 t + \phi_1 y_{t-1} + \ldots + \phi_p y_{t-p} + \sum_{i=1}^{s-1} \gamma_i d_i + \sum_{j=1}^{m} \kappa_j x_{t,j} + \epsilon_t.$$

where:

d_i is a seasonal dummy that is 1 if $mod(t, period) = i$. Period 0 is excluded if the model contains a constant (c is in trend). t is a time trend $(1, 2, \ldots)$ that starts with 1 in the first observation.
$x_{t,j}$ are exogenous regressors. Note these are time-aligned to the left-hand-side variable when defining a model.

ϵ_t is assumed to be a white noise process.

Forecast quality scoring metrics

1. R squared, coefficient of determination (it can be interpreted as a percentage of variance explained by the model), (-inf, 1]

2. Mean Absolute Error, it is an interpretable metric because it has the same unit of measurement as the initial series, [0, +inf)

3.Median Absolute Error, again an interpretable metric, particularly interesting because it is robust to outliers, [0, +inf)

4.Mean Squared Error, most commonly used, gives higher penalty to big mistakes and vise versa, [0, +inf)

5.Mean Squared Logarithmic Error, practically the same as MSE but we initially take logarithm of the series, as a result we give attention to small mistakes as well, usually is used when data has exponential trends, [0, +inf)

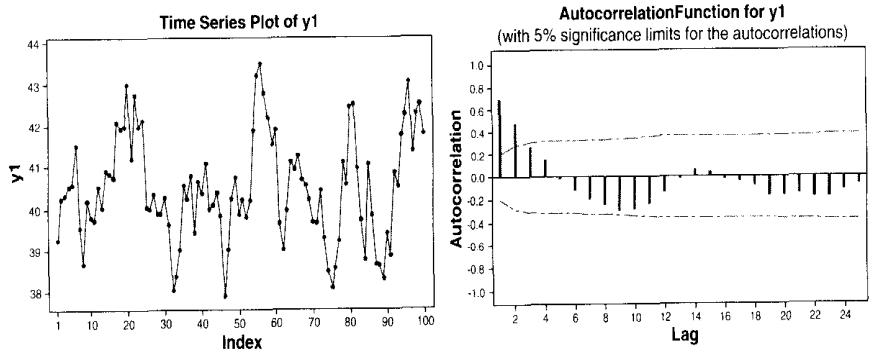
6.Mean Absolute Percentage Error, same as MAE but percentage, very convenient when you want to explain the quality of the model to your management, [0, +inf)

Steps to do time series analysis with a given list or sequence of data.Also maths behind them.

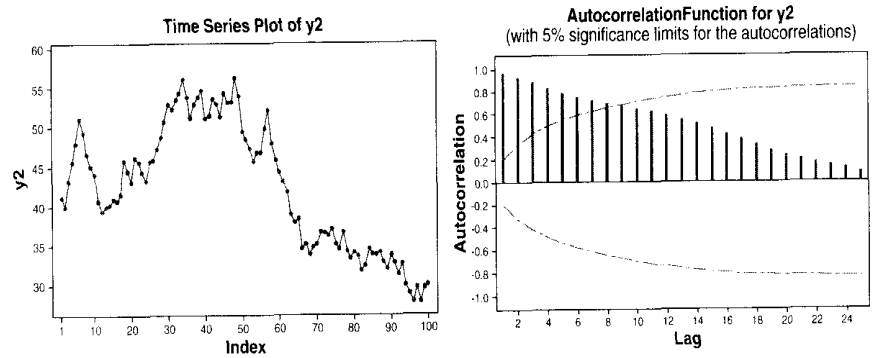
1 Data preprocessing and visualization

1.1 Stationary

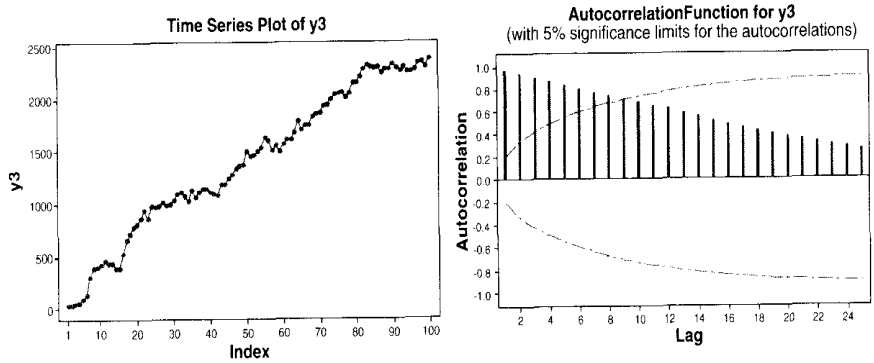
The stationary property of a time series is related to its statistical properties in time. It is enough to say a sequence is stationary or not depends on the first two moments. The expected value of the time series should not depend on time. The auto co-variance function defined as $Cov(y_t, y_{t+k})$ for any lag k is only a function of k and not of time. A strong and slowly dying auto-correlation function (ACF) will also suggest deviations from stationary. Some examples below



(a) $y_{1,t} = 10 + 0.75y_{1,t-1} + \varepsilon_t$



(b) $y_{2,t} = 2 + 0.95y_{2,t-1} + \varepsilon_t$



(c) $y_{3,t} = 20 + y_{3,t-1} + \varepsilon_t$

FIGURE 5.1 Realizations of (a) stationary, (b) near nonstationary, and (c) nonstationary processes.

Source: Douglas C Montgomery ,Cheryl L.Jennings, Murat Kulahsi. Introduction to time series analysis and forecasting

1.1.1 ACF and PACF plots

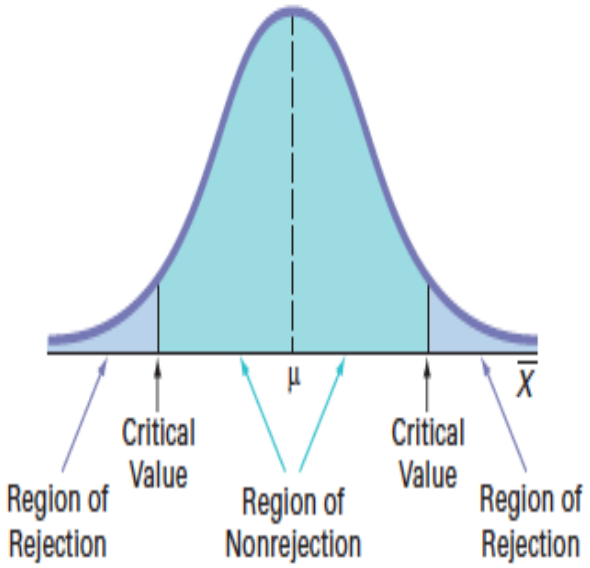
If the time series is stationary, the ACF/PACF plots will show a quick drop-off in correlation after a small amount of lag between points. We will see the proofs in future. Confidence intervals are drawn as a cone, which helps in finding the lag after which the plot cuts the upper confidence interval.

1.1.2 Plotting Rolling Statistics

We take a window and slide over the data. On this window path we simultaneously take data from window and calculate mean and variance and plot them. If the plotted mean and variance are constant then we can say sequence is stationary else non stationary. There is a drawback here. We don't know the size of window we need to use.

1.1.3 Augmented Dickey-Fuller Test

We get two values from this test. One is test statistic and the other is critical value. If the test value value is less than the critical value, we can say that sequence is stationary with respective confidence .



1.2 Making Time Series Stationary

1.2.1 Transformations

Non constant variance is quite common in time series data.
A very famous way of taking care about this non constant variance is the **power family** transformations, given by:

$$y^{(\lambda)} = \frac{y^{\lambda}-1}{\lambda k^{\lambda-1}} \qquad if \lambda \neq 0$$
$$y^{(\lambda)} = k \log y \qquad if \lambda = 0$$

where $k = exp[(1/T)\sum_{t=1}^{T} \log y_t]$ is the geometric mean of the observations.
The values ofλ and their transformations are:

1. $\lambda = 0.5$ (A square root transformation)
2. $\lambda = 0$ (A log transformation)
3. $\lambda = -0.5$(A reciprocal square root transformation)
4. $\lambda = -1$(A inverse transformation)

The log transformation is used frequently in situations where the variability in original time series increases with the average level of the series.The application of a natural logarithm transformation to a sequence , tends to stabilize the variance and leaves just a few unusual values.
Note: when doing this transformation we are giving more penalty to large values than small values and bringing all values to same level.

1.2.2 Techniques to remove Trend - Smoothing

1.2.2.1 Moving Average

Data set as consisting of two distinct components: signal and noise. We focus to separate these two components.

$$y_t = \mu + \epsilon_t$$

where μ represents the underlying constant level of system response and ϵ_t is the noise at time t. The ϵ_t are often assumed to be uncorrelated with mean 0 and constant variance ${\sigma_\epsilon}^2$

An obvious best choice to find moving average is to replace the current observation with the average of the observations at T, T - 1, ... , 1
Proof:

from the above equation, Sum of square errors is

$$SS_E = \sum_{t=1}^{T}(y_t - \mu)^2$$

The least squares estimate of μ can be found by setting the derivative of SS with respect to μ to 0, which results

$$\mu_{new} = \frac{1}{T}\sum_{t-1}^{T} y_t.$$

Hence proved.

Now subtracting the obtained moving average from original data set results a sequence which is stationary. We can get original data again by just reversing the process.

1.2.2.2 Exponentially weighted moving average

We will limit our discussion with first order exponential smoothing.

An exponentially weighted smoother is obtained by introducing a discount factor θ as

$$\sum_{t=0}^{T-1} \theta^t y_{T-t} = y_T + \theta y_{T-1} + \theta^2 y_{T-2} + ... + \theta^{T-1} y_1$$

and observe theta $|\theta| < 1$

We see that the previous observations are penalised or discounted in a geometrically decreased manner.

We need to take care of weights average term.

$$\sum_{t=0}^{T-1} \theta^t = \frac{1-\theta^T}{1-\theta}$$

So, $y_n ew_t = (1-\theta)(y_T + \theta y_{T-1} + \theta^2 y_{T-2} + ... + \theta^{T-1} y_1)$

as θ^T tends to zero for larger values of T.

Now subtract the new y_t values from old values of data to get stationary data. We can get original data again by just reversing the process.

1.2.3.1.Differencing :

Transformations such as logarithms can help to stabilise the variance of a time series.But what about mean?

Differencing will help in stabilising the mean.

The differencing in between consecutive observations from original series of T observation can be written as

$$y'_t = y_t - y_{t-1}$$

So the differed values are only T-1 values. If the new obtained values y'_t are white noise(Stationary) then

$$y_t - y_{t-1} = \epsilon_t$$

then we can get our original values again by reversing i.e

$$y_t = y_{t-1} + \epsilon_t$$

This is called **first order differencing.**

If the obtained values are still not stationary, then we go for differencing again called as **Second order differencing.** Here we will remain only T-1 values. In general going beyond second order differencing is not needed.

Here we use 'd' as a variable to show how many differences we did to our data.

Sometimes the given data might be seasonal. In these cases we will use **sea-**

sonal differencing.

$$y'_t = y_t - y_{t-m}$$

Where m is number of observations in a season.

Simple example on differencing: let the observations results $1, 2, 3, 4, 5, 6$. Then we can clearly see their mean is also increasing . So non stationary. But if we do differencing we get $1, 1, 1, 1, 1$. Which has constant mean i.e nothing to do with time, which is Stationary.

Till now we have learned how to make a Non stationary data to stationary. Because we will apply our models to only these stationary data. Lets dive into models of time series.

2 Time Series forecasting

2.1 Auto regression (AR)

One approach to modeling this time series is to assume that the contributions of the disturbances that are way in the past should be small compared to the more recent disturbances that the process has experienced.
In this notation, the weights on the disturbances starting from the current disturbance and going back in past will be $1, \phi, \phi^2, \phi^3, ...$ and $|\phi| < 1$
so we get y_t value as

$$\begin{aligned} y_t &= \mu + \epsilon_t + \phi\epsilon_{t-1} + \phi^2\epsilon_{t-2} + ... \\ &= \mu + \textstyle\sum_{i=0}^{\infty} \phi^i \epsilon_{t-i} \end{aligned}$$

we can write this as :

$$\begin{aligned} y_t &= \mu + \epsilon_t + \phi(\epsilon_{t-1} + \phi\epsilon_{t-1} + ...) \\ y_t &= \delta + \phi y_{t-1} + \epsilon_t \end{aligned}$$

where $\delta = (1 - \phi)\mu$.

The process is called **first-order autoregressive process**
Now its mean

$$E(y_t) = \mu = \frac{\delta}{1-\phi}$$

the autocovariance function

$$\gamma(k) = \sigma^2 \phi^k \frac{1}{1-\phi^2}$$

its variance :

$$\gamma(0) = \sigma^2 \frac{1}{1-\phi^2}$$

Correspondingly, the autocorrelation function for a stationary AR(1) process is given as

$$\rho(k) = \frac{\gamma(k)}{\gamma(0)} = \phi^k$$

as $|\phi| < 1$ e ca see ACF is exponentially decreasing. This is what we observe in ACF graphs. It should decrease exponentially for a stationary data.

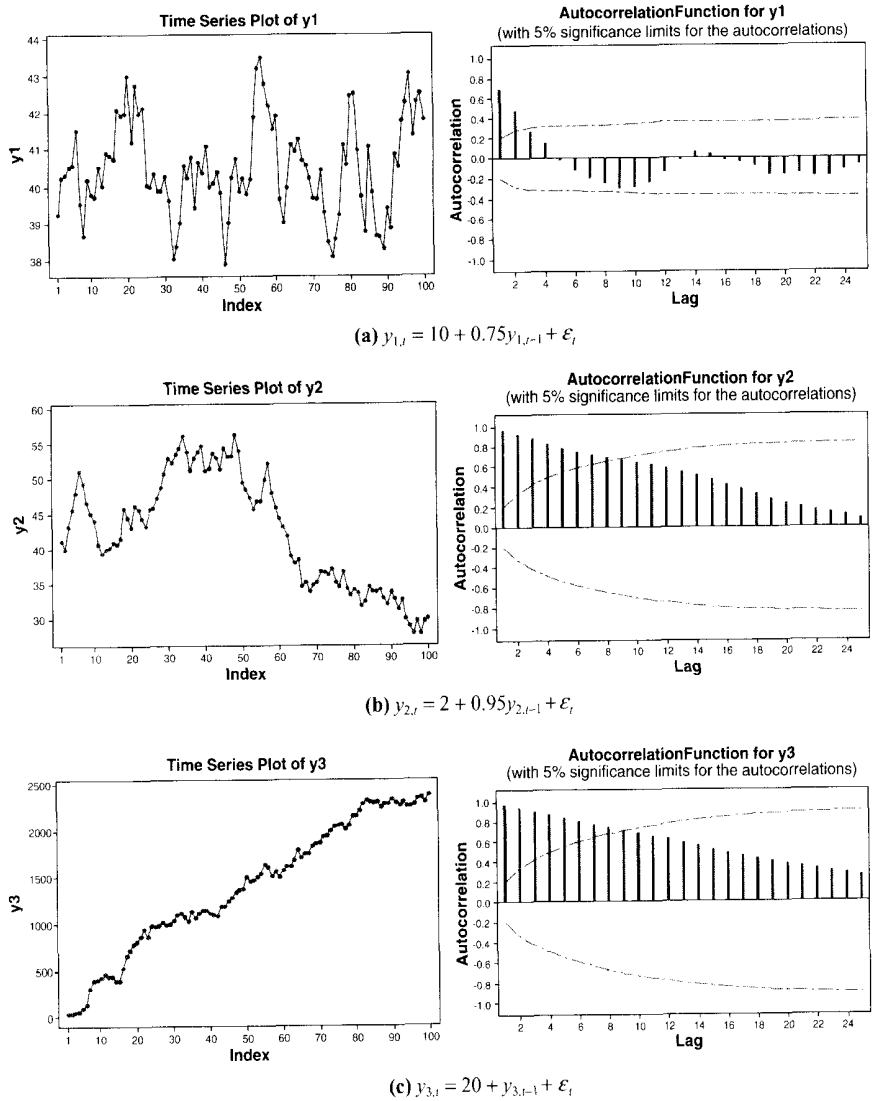


FIGURE 5.1 Realizations of (a) stationary, (b) near nonstationary, and (c) nonstationary processes.

Source: Douglas C Montgomery ,Cheryl L.Jennings, Murat Kulahsi. Introduction to time series analysis and forecasting

Similarly for **second-order autoregressive process** similarly we get

$y_t = \delta + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \epsilon_t$

and we get ACF after solving in similar way

$\rho(k) = c_1 {m_1}^k + c_2 {m_2}^k$ for k= 0,1,2,...and $|m_1| and |m_2| < 1$

it is where $\delta = (1 - \phi)\mu$.

It should also decrease exponentially but of two exponential functions.

2.2 Moving Average (MA)

A moving average process of order q (MA(q)) is given as

$$y_t = \mu + \epsilon_t - \theta_1 \epsilon_{t-1} - ...\theta_q \epsilon_{t-q}$$

where ϵ_t is white noise. White noise means their mean is zero and have a variance(σ^2).

Statement :This MA(q) is always stationary regardless the values of weights.

Proof:

its mean

$E(y_t) = E(\mu + \epsilon_t - \theta_1 \epsilon_{t-1} - ...\theta_q \epsilon_{t-q})$

$E(y_t) = \mu$ as white noise mean is zero.// and its variance

$Var(y_t) = Var(\mu + \epsilon_t - \theta_1 \epsilon_{t-1} - ...\theta_q \epsilon_{t-q})$

$Var(y_t) = \sigma^2(1 + {\theta_1}^2 + ... + {\theta_q}^2)$

Autocovariance at lag k is

$$\begin{aligned} \gamma_y(k) &= Cov(y_t, y_{t+k}) \\ &= E[(\epsilon_t - \theta_1 \epsilon_{t-1} - ...\theta_q \epsilon_{t-q})(\epsilon_{t+k} - \theta_1 \epsilon_{t+k-1} - ...\theta_q \epsilon_{t+k-q})] \\ &= \sigma^2(-\theta_k + \theta_1 \theta k + 1 + ... + \theta_{q-k} \theta q) \qquad for k = 1, 2, 3, 4, ...q \\ &= 0 \qquad for \qquad k > q \end{aligned}$$

Hence from above two calculated values we can find autocovariance function(ACF) of MA(q)

$$\begin{aligned} \rho_y(k) &= \frac{\gamma_y(k)}{\gamma_y(0)} \\ &= \frac{-\theta_k + \theta_1 \theta k + 1 + ... + \theta_{q-k} \theta q}{1 + {\theta_1}^2 + ... + {\theta_q}^2} \qquad for \qquad k = 1, 2, 3...q \\ &= 0 \qquad for \qquad k > q \end{aligned}$$

Now we will implement first order Moving average process. which has q=1

$y_t = \mu + \epsilon_t - \theta_1 \epsilon_{t-1}$

Every thing is same as calculated for MA(q).

So Autocovariance function here will be

$\rho_y(1) = \frac{-\theta_1}{1 + {\theta_1}^2}$

and $= 0$ for $k > 1$

So $|\rho|$ value is always less than or equal to 1/2. hence and the auto-correlation

function cuts off after lag 1 for sure.
Similarly for any value of q we can calculate lag value where ACF function will get intersect with X-axis.
We will calculate the parameters θ by minimising error.

2.3 Auto regressive Moving Average (ARMA)

Generally data we get may have both change in mean and variance with time
So we implement both Moving average and Auto Regression in same equation and need to check whether the equation y_t is stable or not.
We have two variables p,q indicates the no of terms of AR and MA .
Here

$$y_t = \delta + \phi_1 y_{t-1} + \phi_2 y_{t-2} + .. + \phi_p y_{t-p} + \epsilon_t - \theta_1 \epsilon_{t-1} - ...\theta_q \epsilon_{t-q}$$

where ϵ_t is white noise.

Stationarity of ARMA (p, q) Process:
The stationarity of an ARMA process is related to the AR component in the model and can be checked through the roots of the associated polynomial

$$m^p - \phi_1 m^{p-1} - \phi_2 m^{p-2} - .... - \phi_p = 0$$

If the absolute value of all the roots of the above equation is less than 1, then the sequence is stationary.

Invertibility of ARMA (p, q) Process :
Similar to the stationarity condition. the invertibility of an ARMA process is related to the MA component and can be checked through the roots of the associated polynomiall

$$m^q - \theta_1 m^{q-1} - \theta_2 m^{q-2} - .... - \theta_q = 0$$

If the absolute value of all the roots of the above equation is less than 1, then the sequence is Invertible and has an infinite AR representation.

A small table to compare the behavior of theoretical ACF and PACF for Stationary Processes

TABLE 5.1 Behavior of Theoretical ACF and PACF for Stationary Processes

Model	ACF	PACF
MA(q)	Cuts off after lag q	Exponential decay and/or damped sinusoid
AR(p)	Exponential decay and/or damped sinusoid	Cuts off after lag p
ARMA(p,q)	Exponential decay and/or damped sinusoid	Exponential decay and/or damped sinusoid

Source: Douglas C Montgomery ,Cheryl L.Jennings, Murat Kulahsi. Introduction to time series analysis and forecasting

2.4 Auto regressive Integrated Moving Average (ARIMA)(p,d,q)

It is similar to ARMA method. But we add another value d to the equation. Here 'd' is used for differencing discussed above. It make life easier to combine both steps 1)converting to stationary data and 2) implementing ARMA ,into one single process.
It is represented by (ARIMA)(p,d,q)
here

$y_t = \alpha + \sum_{i=1}^{\infty} \pi_i y_{t-i} + \epsilon_t.$

3 An example of implementing a model using all the above steps one by one

Problem statement.

From World Health Organization - On 31 December 2019, WHO was alerted to several cases of pneumonia in Wuhan City, Hubei Province of China. The virus did not match any other known virus. This raised concern because when a virus is new, we do not know how it affects people.

So daily level information on the affected people can give some interesting insights when it is made available to the broader data science community. data is available from 1/22/2020 to 2/17/2020.

This notebook contain the model fitting of no of people who got corona virus and proved to be confirmed

And then predict the future possible number of patients.

Source of dataset: https://www.kaggle.com/sudalairajkumar/novel-corona-virus-2019-dataset.

First i made some visualizations and then stationarity checking and then models.

In [1]:

```
# importing some important libraries

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

import warnings
import itertools
import statsmodels.api as sm
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.graphics.tsaplots import plot_pacf
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.ar_model import AR
from statsmodels.tsa.arima_model import ARMA, ARIMA
from pmdarima import auto_arima
from statsmodels.tsa.statespace.sarimax import SARIMAX

from math import sqrt

import matplotlib
```

In [2]:

```
df = pd.read_csv('./novel-corona-virus-2019-dataset/2019_nCoV_data.csv',parse_dates=['Last Update'
])
```

In [3]:

```
#printing first 5 values of dataset.
df.head()
```

Out[3]:

	Sno	Date	Province/State	Country	Last Update	Confirmed	Deaths	Recovered
0	1	01/22/2020 12:00:00	Anhui	China	2020-01-22 12:00:00	1.0	0.0	0.0
1	2	01/22/2020 12:00:00	Beijing	China	2020-01-22 12:00:00	14.0	0.0	0.0
2	3	01/22/2020 12:00:00	Chongqing	China	2020-01-22 12:00:00	6.0	0.0	0.0
3	4	01/22/2020 12:00:00	Fujian	China	2020-01-22 12:00:00	1.0	0.0	0.0
4	5	01/22/2020 12:00:00	Gansu	China	2020-01-22 12:00:00	0.0	0.0	0.0

In [4]:

```
#printing last 5 values of dataset.
df.tail()
```

```
df.tail()
```

Out[4]:

	Sno	Date	Province/State	Country	Last Update	Confirmed	Deaths	Recovered
1714	1715	02/17/2020 22:00:00	Madison, WI	US	2020-02-05 21:53:02	1.0	0.0	0.0
1715	1716	02/17/2020 22:00:00	Orange, CA	US	2020-02-01 19:53:03	1.0	0.0	0.0
1716	1717	02/17/2020 22:00:00	San Antonio, TX	US	2020-02-13 18:53:02	1.0	0.0	0.0
1717	1718	02/17/2020 22:00:00	Seattle, WA	US	2020-02-09 07:03:04	1.0	0.0	1.0
1718	1719	02/17/2020 22:00:00	Tempe, AZ	US	2020-02-01 19:43:03	1.0	0.0	0.0

In [5]:

```
df2 = df.groupby(["Date", "Country"])[['Date', 'Country', 'Confirmed', 'Deaths', 'Recovered']].sum(
).reset_index()
```

COUNTRIES EFFECTED

A total of 34 countries affected.

In [6]:

```
all_countries = df['Country'].unique()
print("Number of countries with cases: " + str(len(all_countries)))
print("Countries with cases: ")
for i in all_countries:
    print("    " + str(i))
```

```
Number of countries with cases: 34
Countries with cases:
    China
    US
    Japan
    Thailand
    South Korea
    Mainland China
    Hong Kong
    Macau
    Taiwan
    Singapore
    Philippines
    Malaysia
    Vietnam
    Australia
    Mexico
    Brazil
    France
    Nepal
    Canada
    Cambodia
    Sri Lanka
    Ivory Coast
    Germany
    Finland
    United Arab Emirates
    India
    Italy
    Sweden
    Russia
    Spain
    UK
    Belgium
    Others
    Egypt
```

In [7]:

```
confirmed = df.groupby('Date').sum()['Confirmed'].reset_index()
confirmed.columns = ['Date','Confirmed']
confirmed['Date'] = pd.to_datetime(confirmed['Date'])
confirmed=confirmed.set_index("Date")

deaths = df.groupby('Date').sum()['Deaths'].reset_index()
deaths.columns = ['Date',"Deaths"]
deaths['Date'] = pd.to_datetime(deaths['Date'])
deaths=deaths.set_index("Date")

recovered= df.groupby('Date').sum()['Recovered'].reset_index()
recovered.columns = ['Date',"Recovered"]
recovered['Date'] = pd.to_datetime(recovered['Date'])
recovered=recovered.set_index("Date")
```

In [8]:

```
print(confirmed.isnull().sum())
print(deaths.isnull().sum())
print(recovered.isnull().sum())
```

```
Confirmed    0
dtype: int64
Deaths    0
dtype: int64
Recovered    0
dtype: int64
```

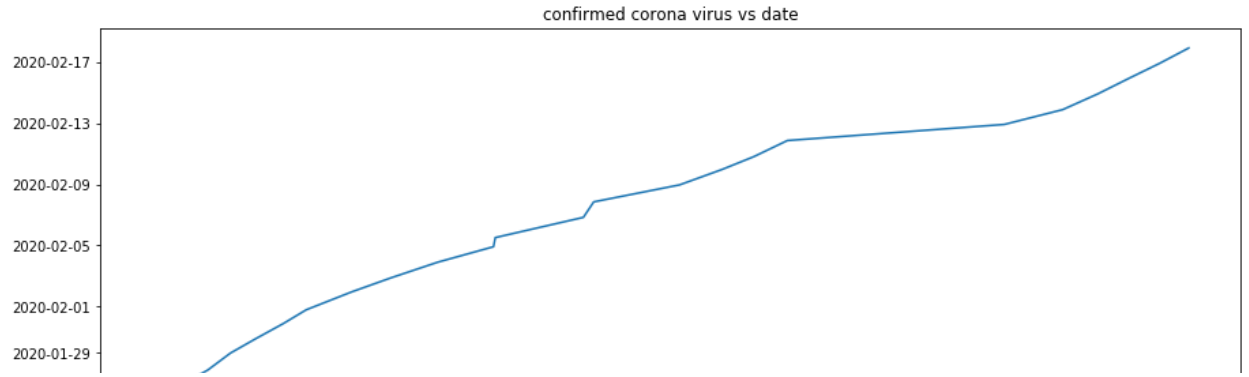
NOTE:(for entire notebook)

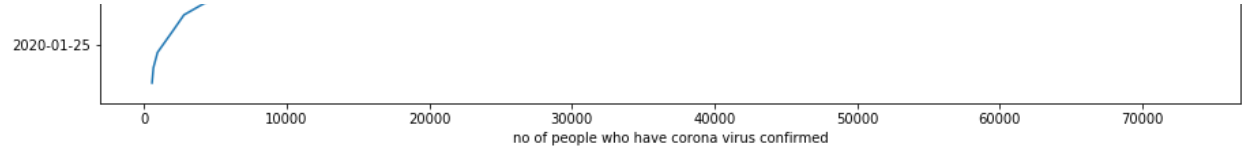
- #### blue colour line in graph indicates original data.
- #### red colour line in graph indicates mean in respective method.
- #### black colour line in graph indicates Standard deviation in respective method.
- #### orange colour line in graph indicates predicted data

In [9]:

```
x=confirmed['Confirmed']
y=confirmed.index
plt.figure(figsize=(15,6))
from pandas.plotting import register_matplotlib_converters
plt.title('confirmed corona virus vs date')
plt.xlabel('no of people who have corona virus confirmed')
plt.plot(x,y);
```

```
/home/tejendhar/anaconda3/lib/python3.7/site-
packages/pandas/plotting/_matplotlib/converter.py:103: FutureWarning: Using an implicitly
registered datetime converter for a matplotlib plotting method. The converter was registered by pa
ndas on import. Future versions of pandas will require you to explicitly register matplotlib conve
rters.

To register the converters:
 >>> from pandas.plotting import register_matplotlib_converters
 >>> register_matplotlib_converters()
  warnings.warn(msg, FutureWarning)
```





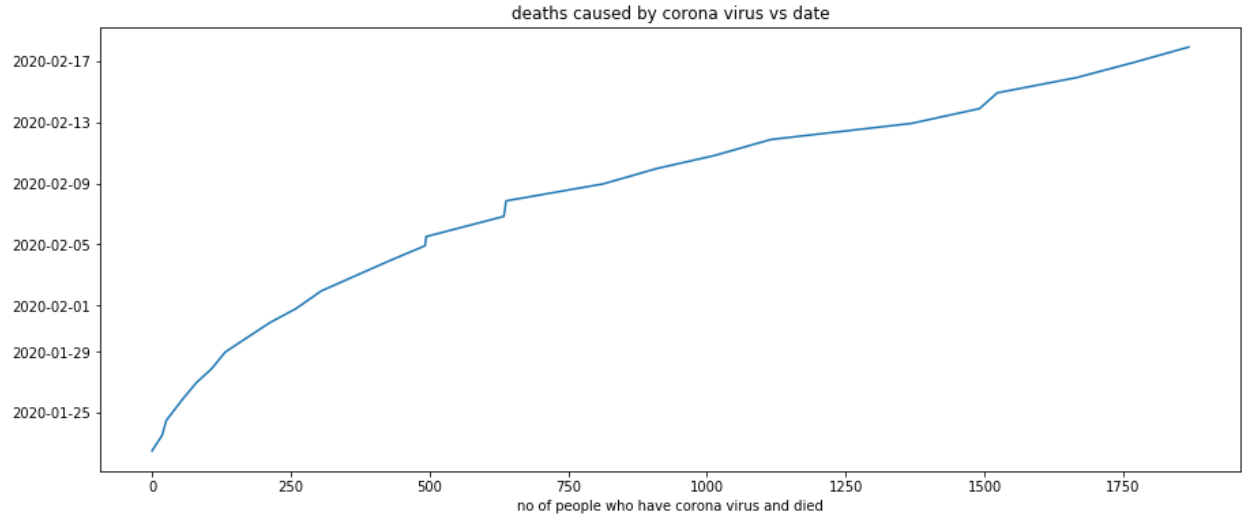
From the above graph we can say that the data is keep on increasing and hence non stationary data.

It has some dependency on time.

So we need to remove this non stationarity to apply model.

In [10]:

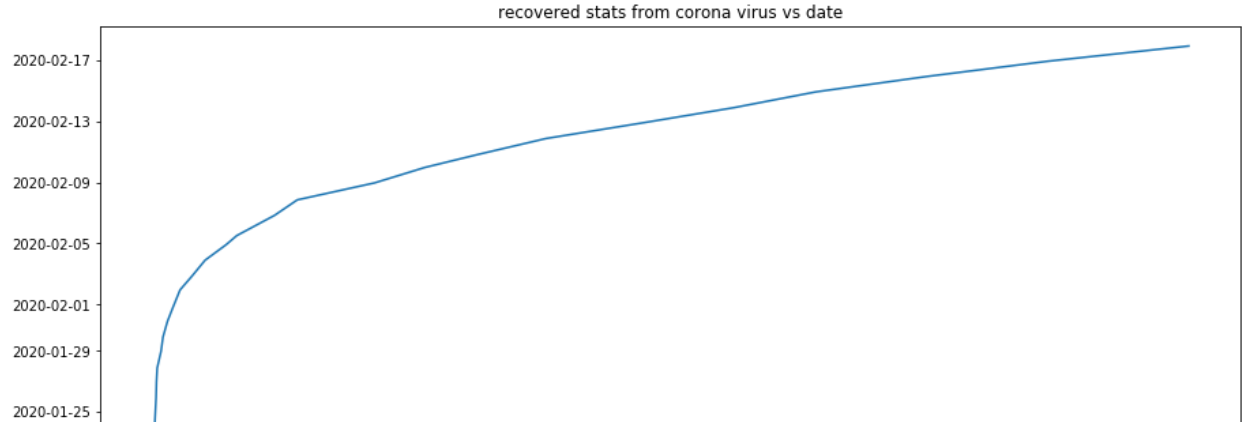
```
x=deaths['Deaths']
y=deaths.index
plt.figure(figsize=(15,6))
plt.title('deaths caused by corona virus vs date')
plt.xlabel('no of people who have corona virus and died')
plt.plot(x,y);
```

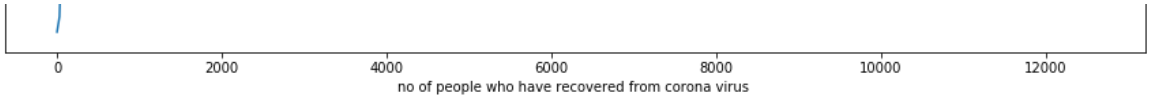


The above graph is only for information. Nothing to do with prediction .

In [11]:

```
x=recovered['Recovered']
y=recovered.index
plt.figure(figsize=(15,6))
plt.title('recovered stats from corona virus vs date')
plt.xlabel('no of people who have recovered from corona virus ')
plt.plot(x,y);
```



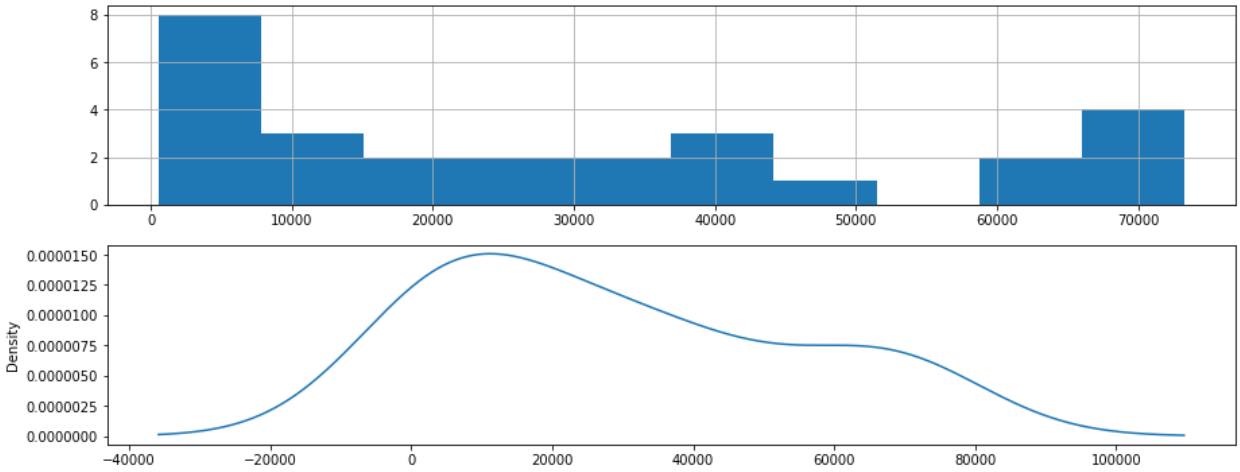


The above graph is only for information. Nothing to do with prediction .

In [12]:

```
from pandas import Series
from matplotlib import pyplot
pyplot.figure(1)
plt.figure(figsize=(15,6))

pyplot.subplot(211)
confirmed.Confirmed.hist()
pyplot.subplot(212)
confirmed.Confirmed.plot(kind='kde')
pyplot.show()
```

```
<Figure size 432x288 with 0 Axes>
```



The above Graph shows us the density of the given data. A lot of dates have number of people got infected is in range (1,10000).

This above graph gives us the view the variance of data.

Stationarity

---A Time Series is said to be stationary if its statistical properties such as mean, variance remain constant over time.

---Most of the Time Series models work on the assumption that the TS is stationary. Major reason for this is that there are many ways in which a series can be non-stationary, but only one way for stationarity.

---Intuitively, we can say that if a Time Series has a particular behaviour over time, there is a very high probability that it will follow the same in the future.

---Also, the theories related to stationary series are more mature and easier to implement as compared to non-stationary series.

---We can use statsmodels to perform a decomposition of this time series.

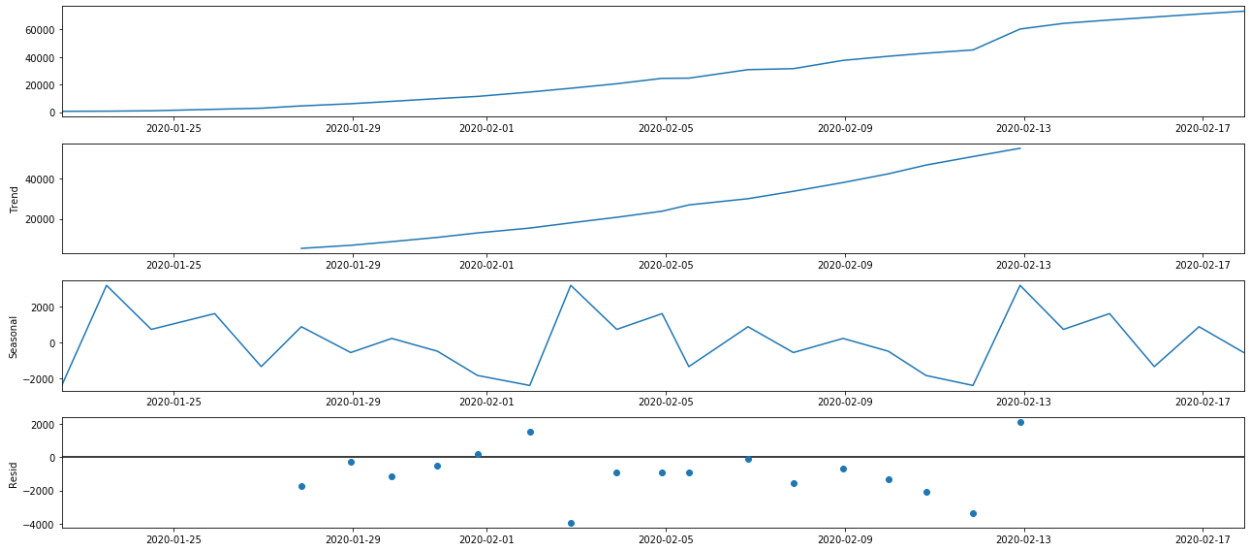
---The decomposition of time series is a statistical task that deconstructs a time series into several components, each representing one of the underlying categories of patterns.

---With statsmodels we will be able to see the trend, seasonal, and residual components of our data.

we will see some methods to find stationarity.

In [13]:

```
# Decomposition process is very important for every problem. It gives usa lot of inference.
from pylab import rcParams
rcParams['figure.figsize'] = (18, 8)
confirmed.index = pd.to_datetime(confirmed.index)
decomposition = sm.tsa.seasonal_decompose(confirmed,period=10)
fig = decomposition.plot()
plt.show()
```



From the above graph we can see that mean increases with time. Hence from this we can confirm that the given data is stationary. Seasonal parameter is not so varying . So no need to consider seasonal parameters here.

ACF and PACF plots

---Let's review the Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF) plots

---If the time series is stationary, the ACF/PACF plots will show a quick drop-off in correlation after a small amount of lag between points.

---The data is non-stationary if a high number of previous observations are correlated with future values.

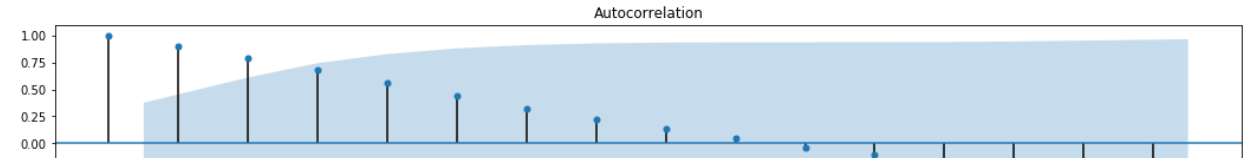
---Confidence intervals are drawn as a cone.

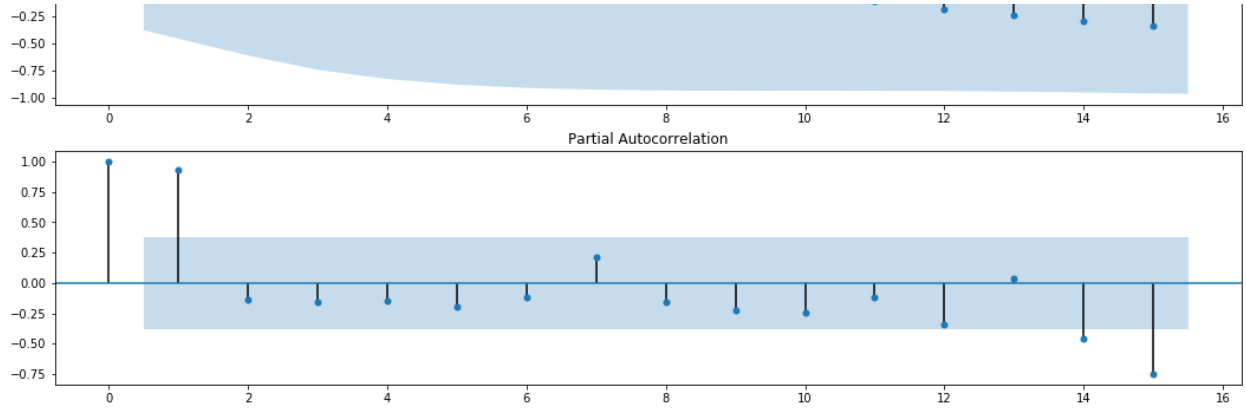
---By default, this is set to a 95% confidence interval, suggesting that correlation values outside of this code are very likely a correlation and not a statistical fluke.

---The partial autocorrelation at lag k is the correlation that results after removing the effect of any correlations due to the terms at shorter lags.

In [14]:

```
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.graphics.tsaplots import plot_pacf

pyplot.figure()
pyplot.subplot(211)
plot_acf(confirmed.Confirmed, ax=pyplot.gca(), lags = 15)
pyplot.subplot(212)
plot_pacf(confirmed.Confirmed, ax=pyplot.gca(), lags = 15)
pyplot.show()
```



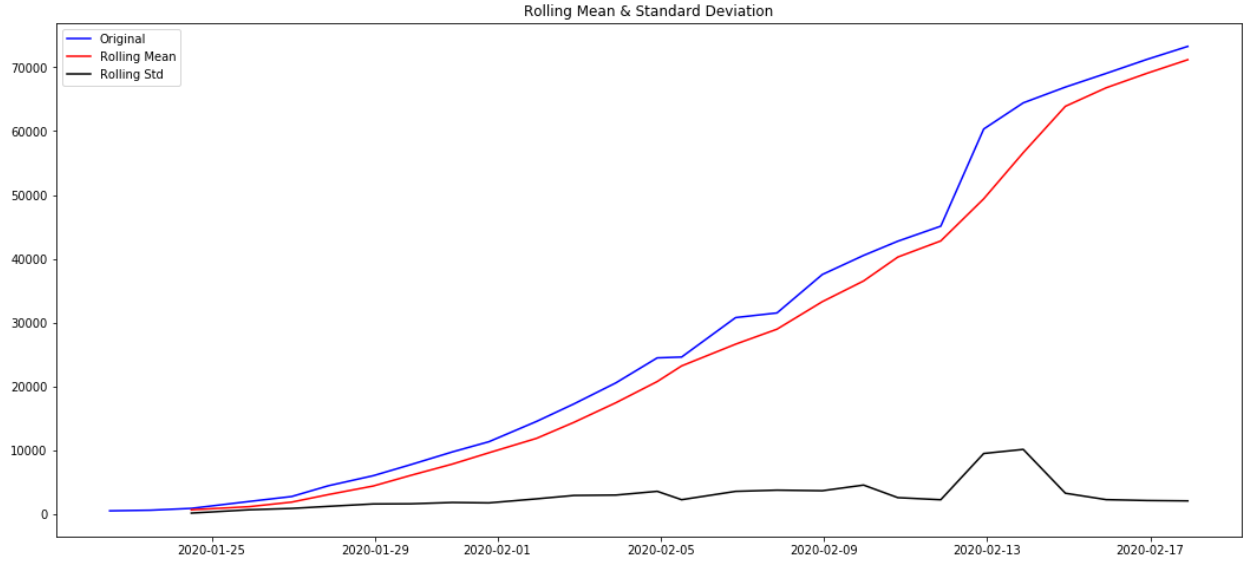


from the above ACF graph we can say that, the graph doesn't converge to zero before reaching confidence interval, which indicates non stationarity.

Plotting Rolling Statistics

In [15]:

```
#rolmean = pd.rolling_mean(confirmed, window=12)
#rolstd = pd.rolling_std(confirmed, window=12)
rolmean = confirmed.rolling(3).mean()
rolstd = confirmed.rolling(3).std()


#Plot rolling statistics:
orig = plt.plot(confirmed, color='blue',label='Original')
mean = plt.plot(rolmean, color='red', label='Rolling Mean')
std = plt.plot(rolstd, color='black', label = 'Rolling Std')
plt.legend(loc='best')
plt.title('Rolling Mean & Standard Deviation')
plt.show(block=False)
```



Again from the above graph we can infer rolling average is increaing and moving statndard deviation not changes much.

Augmented Dickey-Fuller Test

The intuition behind the test is that if the series is integrated then the lagged level of the series y(t-1) will provide no relevant information in predicting the change in y(t).

Null hypothesis: The time series is not stationary

Rejecting the null hypothesis (i.e. a very low p-value) will indicate stationarity

In [16]:

```
from statsmodels.tsa.stattools import adfuller
#Perform Dickey-Fuller test:
print ('Results of Dickey-Fuller Test:')
dftest = adfuller(confirmed.Confirmed, autolag='AIC')
dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','#Lags Used','Number of Observa
tions Used'])
for key,value in dftest[4].items():
    dfoutput['Critical Value (%s)'%key] = value
print (dfoutput)
```

```
Results of Dickey-Fuller Test:
Test Statistic                  2.516581
p-value                         0.999055
#Lags Used                      5.000000
Number of Observations Used    21.000000
Critical Value (1%)            -3.788386
Critical Value (5%)            -3.013098
Critical Value (10%)           -2.646397
dtype: float64
```

Low value of p-value indicates stationarity . But it is quite high which infers non stationarity.

In [17]:

```
### AN OVERALL FUNCTION TO CHECK STATIONARITY.

def test_stationarity(timeseries):

    #Determing rolling statistics
    rolmean = timeseries.rolling(5).mean()
    rolstd = timeseries.rolling(5).std()

    #Plot rolling statistics:
    orig = plt.plot(timeseries, color='blue',label='Original')
    mean = plt.plot(rolmean, color='red', label='Rolling Mean')
    std = plt.plot(rolstd, color='black', label = 'Rolling Std')
    plt.legend(loc='best')
    plt.title('Rolling Mean & Standard Deviation')
    plt.show(block=False)

    #Perform Dickey-Fuller test:
    print ('Results of Dickey-Fuller Test:')
    dftest = adfuller(timeseries, autolag='AIC')
    dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','#Lags Used','Number of Obs
ervations Used'])
    for key,value in dftest[4].items():
        dfoutput['Critical Value (%s)'%key] = value
    print (dfoutput)
```

From the above methods we can say that the given data have varying mean. So we need to remove that moving average and make data stationary to apply models.

Also here for this data set a very little change in variance. So we may neglect removing variance.

Techniques to remove Trend - Smoothing¶

1.log scale transformation

We can apply transformation which penalize higher values more than smaller values. These can be taking a log, square root, cube root, etc

2.Exponential tranformation

3.Box Cox transformation

4.Square root transformation

Lets apply exponential transformation

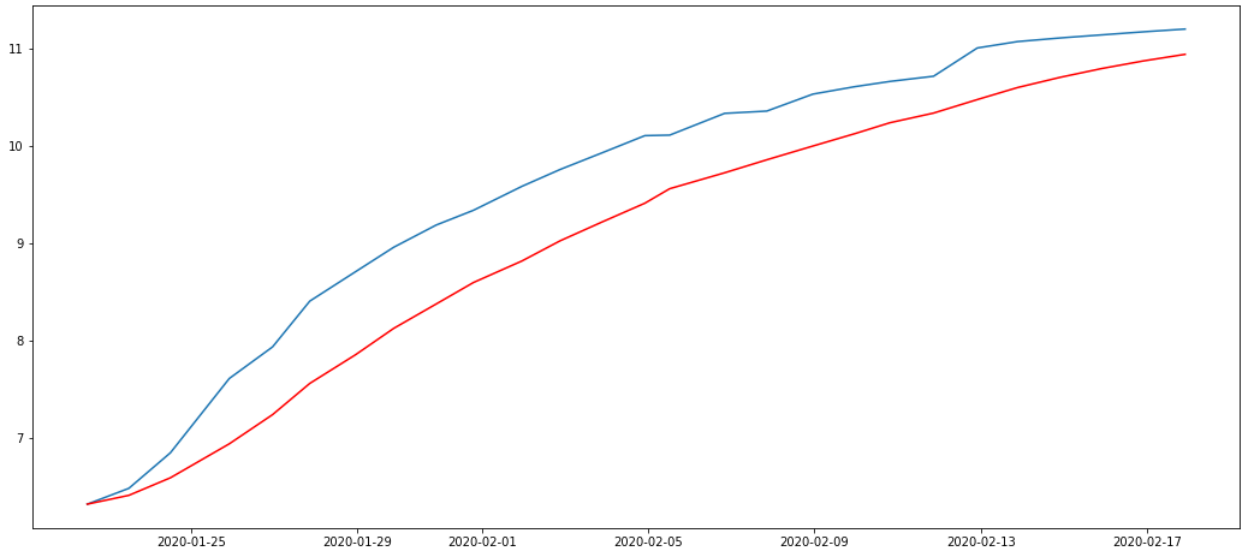
An exponentially weighted smoother is obtained by introducing a discount factor. We see that the previous observations are penalised or discounted in a geometrically decreased manner. We need to take care of weights average term.

In [18]:

```
ts_log = np.log(confirmed)

expwighted_avg = pd.DataFrame.ewm(ts_log, halflife=3).mean()
plt.plot(ts_log)
plt.plot(expwighted_avg, color='red')
```

Out[18]:

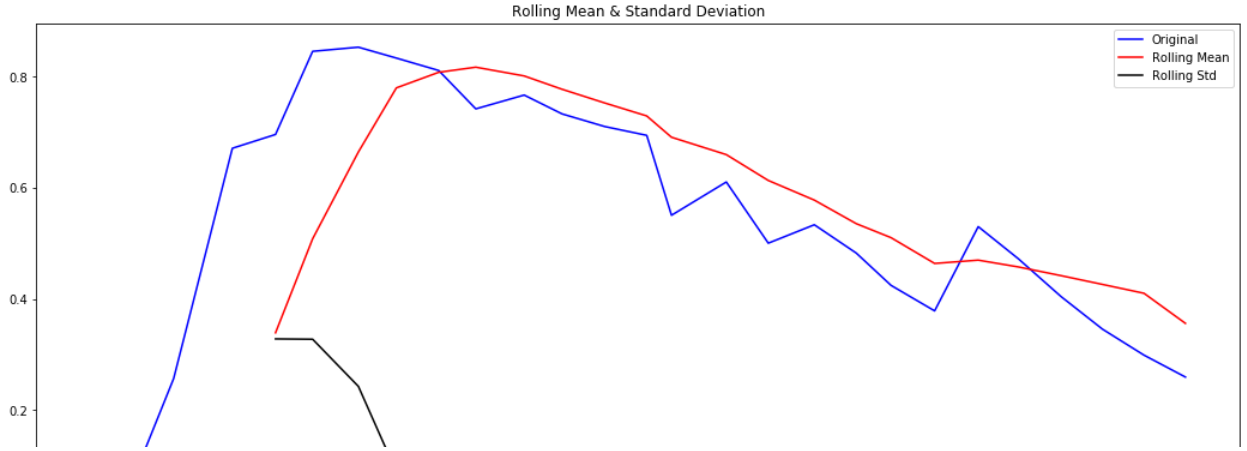
```
[<matplotlib.lines.Line2D at 0x7f5baa926510>]
```

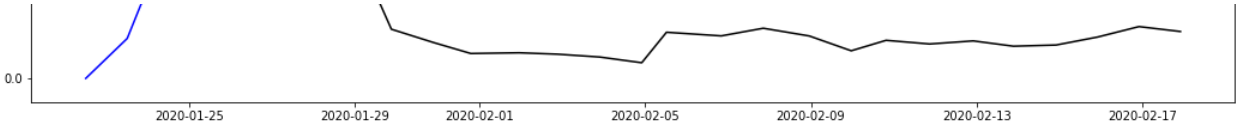


we have applied exponential transformation here . then below i calculated resultant values and checked for if any non stationarity remained or not.

In [19]:

```
ts_log_ewma_diff = ts_log.Confirmed - expwighted_avg.Confirmed
test_stationarity(ts_log_ewma_diff)
```






```
Results of Dickey-Fuller Test:
Test Statistic                  -2.353709
p-value                          0.155230
#Lags Used                       1.000000
Number of Observations Used     25.000000
Critical Value (1%)             -3.723863
Critical Value (5%)             -2.986489
Critical Value (10%)            -2.632800
dtype: float64
```

Now we have p-value as very less. So we can go further with model implementation as data became stationary.

We have many methods to remove stationarity . we can go with any of them . But we need to check the stationarity of resultant data.

Autoregression (AR)

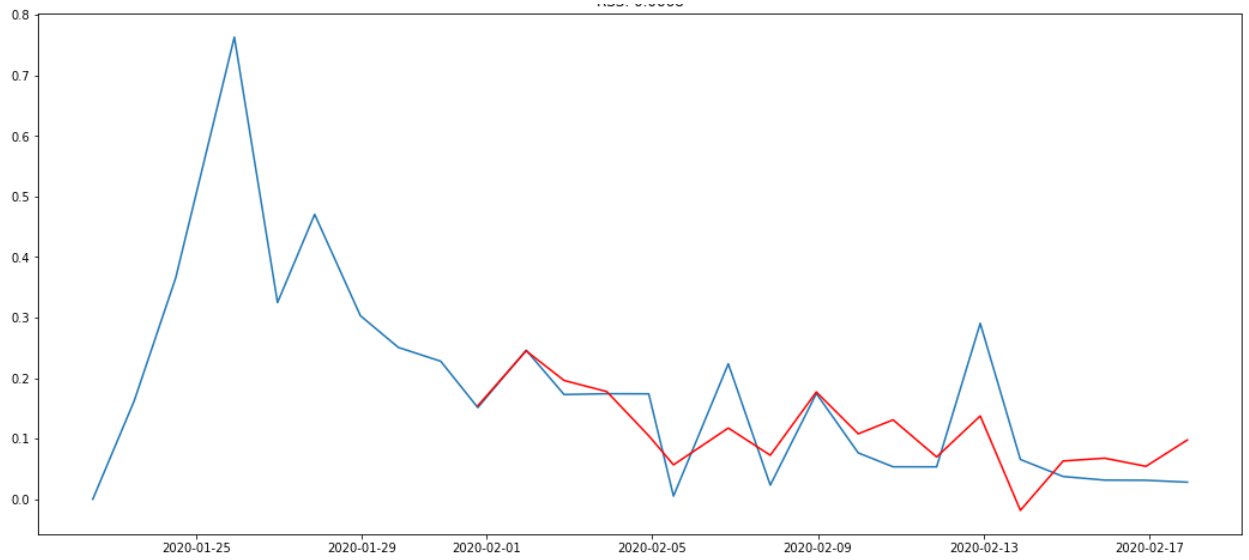
The autoregression (AR) method models the next step in the sequence as a linear function of the observations at prior time steps.

Number of AR (Auto-Regressive) terms (p): p is the parameter associated with the auto-regressive aspect of the model, which incorporates past values i.e lags of dependent variable. For instance if p is 5, the predictors for x(t) will be x(t-1)….x(t-5).

In [20]:

```
import warnings
warnings.filterwarnings('ignore', 'statsmodels.tsa.ar_model.AR', FutureWarning)

ts_log_diff = ts_log.Confirmed - ts_log.Confirmed.shift()
ts_log_diff = ts_log_diff.fillna(0)
import datetime as dt
from statsmodels.tsa.ar_model import AR
from random import random

z= pd.date_range(start='1/22/2018', end='2/17/2018')
model = AR(ts_log_diff, dates=z,freq='D')
model_fit = model.fit()

plt.plot(ts_log_diff)
plt.plot(model_fit.fittedvalues, color='red')
plt.title('RSS: %.4f'% np.nansum((model_fit.fittedvalues-ts_log_diff)**2))
plt.show()
```

```
/home/tejendhar/anaconda3/lib/python3.7/site-packages/statsmodels/tsa/ar_model.py:691:
FutureWarning:
statsmodels.tsa.AR has been deprecated in favor of statsmodels.tsa.AutoReg and
statsmodels.tsa.SARIMAX.

AutoReg adds the ability to specify exogenous variables, include time trends,
and add seasonal dummies. The AutoReg API differs from AR since the model is
treated as immutable, and so the entire specification including the lag
length must be specified when creating the model. This change is too
substantial to incorporate into the existing AR api. The function
ar_select_order performs lag length selection for AutoReg models.

AutoReg only estimates parameters using conditional MLE (OLS). Use SARIMAX to
estimate ARX and related models using full MLE via the Kalman Filter.

To silence this warning and continue using AR until it is removed, use:

import warnings
warnings.filterwarnings('ignore', 'statsmodels.tsa.ar_model.AR', FutureWarning)

  warnings.warn(AR_DEPRECATION_WARN, FutureWarning)
```



Reversing the transformations

In [21]:

```
predictions_ARIMA_diff = pd.Series(model_fit.fittedvalues, copy=True)
```

In [22]:

```
predictions_ARIMA_diff_cumsum = predictions_ARIMA_diff.cumsum()
```

In [23]:

```
predictions_ARIMA_log = pd.Series(ts_log.Confirmed.iloc[0], index=ts_log.index)
predictions_ARIMA_log = predictions_ARIMA_log.add(predictions_ARIMA_diff_cumsum,fill_value=0)
```

In [24]:

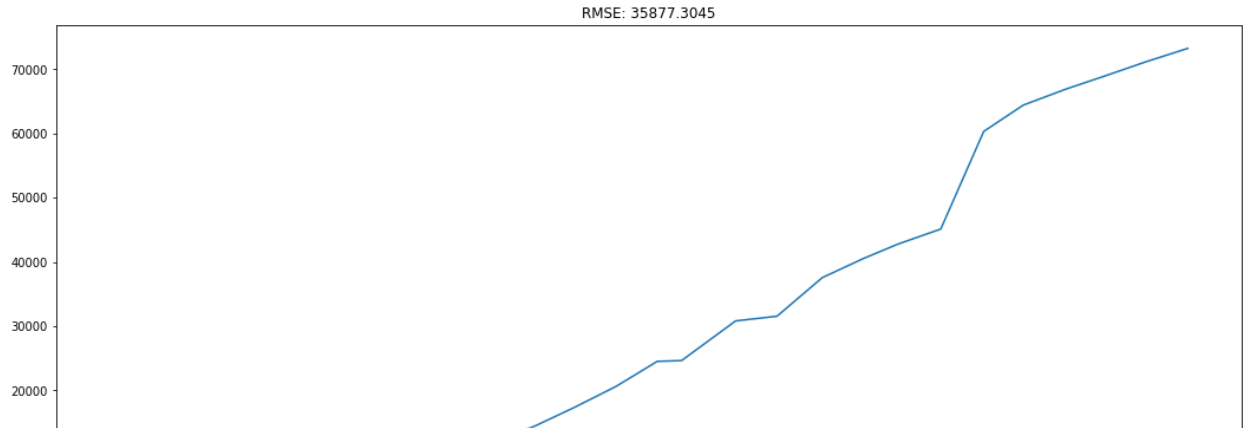
```
predictions_ARIMA = np.exp(predictions_ARIMA_log)
```

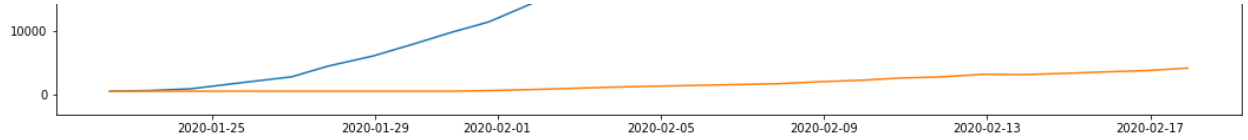
In [25]:

```
plt.plot(confirmed.Confirmed)
plt.plot(predictions_ARIMA)
plt.title('RMSE: %.4f'%
np.sqrt(np.nansum((predictions_ARIMA-confirmed.Confirmed)**2)/len(confirmed.Confirmed)))
```

Out[25]:

```
Text(0.5, 1.0, 'RMSE: 35877.3045')
```





We already know that our data has not much variance. Hence we got a nearly constant newly obtained values. i.e Nothing to do with regression. Our root mean square error is also so high.

In [26]:

```
from statsmodels.tsa.arima_model import ARMA
from random import random

# fit model
ts_log_diff.index = z
model = ARMA(ts_log_diff, order=(1, 0))
model_fit = model.fit(disp=False)
```

In [27]:

```
model_fit.summary()
```

Out[27]:

ARMA Model Results

Dep. Variable:	Confirmed	**No. Observations:**	27
Model:	ARMA(1, 0)	**Log Likelihood**	13.502
Method:	css-mle	**S.D. of innovations**	0.146
Date:	Fri, 06 Mar 2020	**AIC**	-21.004
Time:	01:51:19	**BIC**	-17.116
Sample:	01-22-2018 - 02-17-2018	**HQIC**	-19.848

	coef	std err	z	P>|z|	[0.025	0.975]
const	0.1699	0.053	3.184	0.001	0.065	0.274
ar.L1.Confirmed	0.4866	0.170	2.865	0.004	0.154	0.819

Roots

	Real	Imaginary	Modulus	Frequency
AR.1	2.0552	+0.0000j	2.0552	0.0000

Above block is summary of the model AR .which is nothing but ARMA(1,0).

Note:

We need to change the parameters of model(?,?) according to the result table.(Example: AR(), ARMA() etc..)

The model summary reveals a lot of information. The table in the middle is the coefficients table where the values under 'coef' are the weights of the respective terms.

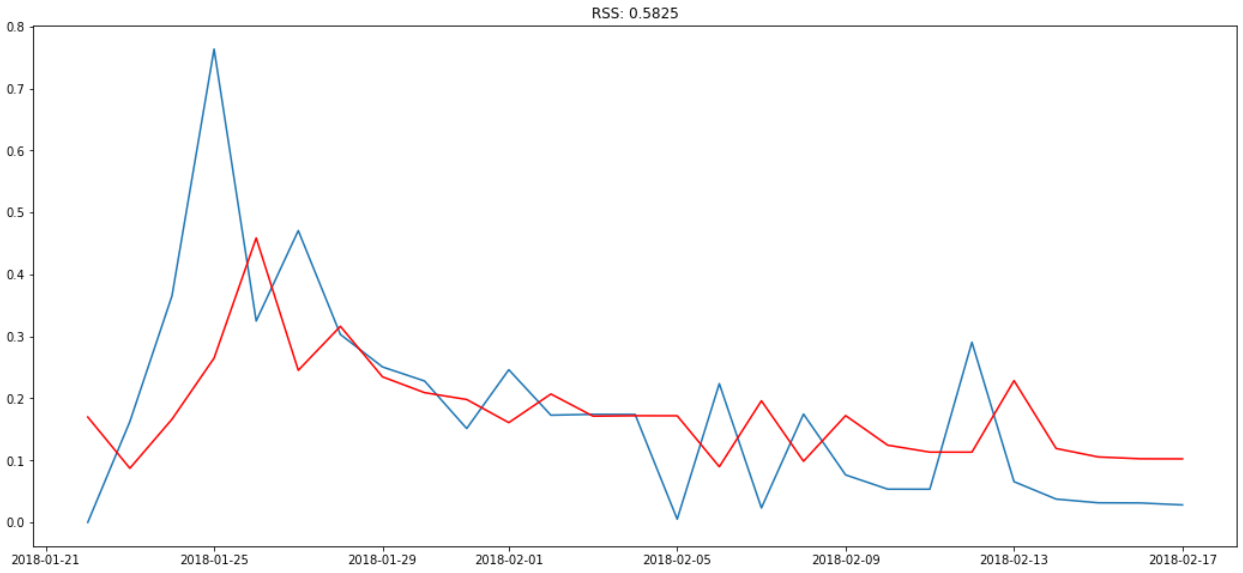
The P-Value in 'P>|z|' column should ideally be less than 0.05 for the respective X to be significant.

In [28]:

```
plt.plot(ts_log_diff)
plt.plot(model_fit.fittedvalues, color='red')
plt.title('RSS: %.4f'% np.nansum((model_fit.fittedvalues-ts_log_diff)**2))
```

Out[28]:

```
Text(0.5, 1.0, 'RSS: 0.5825')
```



from the above image we can see RSS value is less. Hence this model is good.

Autoregressive Moving Average (ARMA)

Number of AR (Auto-Regressive) terms (p): p is the parameter associated with the auto-regressive aspect of the model, which incorporates past values i.e lags of dependent variable. For instance if p is 5, the predictors for x(t) will be x(t-1)….x(t-5).

Number of MA (Moving Average) terms (q): q is size of the moving average part window of the model i.e. lagged forecast errors in prediction equation. For instance if q is 5, the predictors for x(t) will be e(t-1)….e(t-5) where e(i) is the difference between the moving average at ith instant and actual value.

In [29]:

```
from statsmodels.tsa.arima_model import ARMA
from random import random

model = ARMA(ts_log_diff, order=(1,0))
model_fit = model.fit(disp=False)
```

In [30]:

```
model_fit.summary()
```

Out[30]:

ARMA Model Results

Dep. Variable:	Confirmed	**No. Observations:**	27
Model:	ARMA(1, 0)	**Log Likelihood**	13.502
Method:	css-mle	**S.D. of innovations**	0.146
Date:	Fri, 06 Mar 2020	**AIC**	-21.004
Time:	01:51:20	**BIC**	-17.116
Sample:	01-22-2018	**HQIC**	-19.848
	- 02-17-2018		

	coef	std err	z	P>|z|	[0.025	0.975]
const	0.1699	0.053	3.184	0.001	0.065	0.274
ar.L1.Confirmed	0.4866	0.170	2.865	0.004	0.154	0.819

Roots

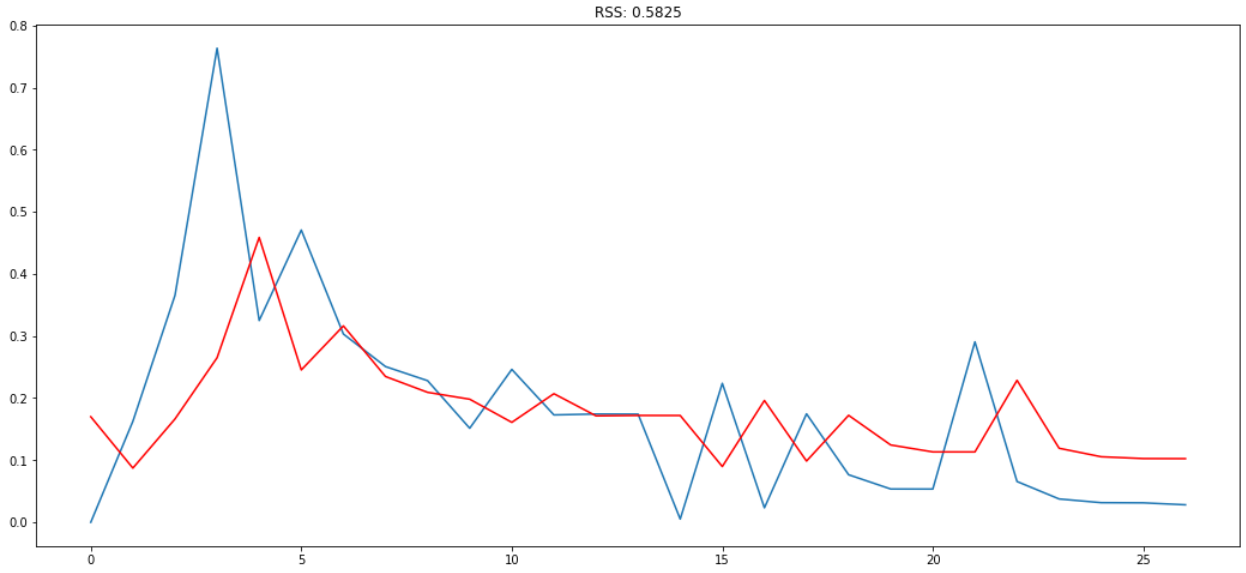
	Real	Imaginary	Modulus	Frequency
AR.1	2.0552	+0.0000j	2.0552	0.0000

In [31]:

```
plt.plot(ts_log_diff.values)
plt.plot(model_fit.fittedvalues.values, color='red')
plt.title('RSS: %.4f'% np.nansum((model_fit.fittedvalues-ts_log_diff)**2))
```

Out[31]:

```
Text(0.5, 1.0, 'RSS: 0.5825')
```



we can see that model summary is good for ARMA(1,0) . that is similar to auto regression.

Autoregressive Integrated Moving Average (ARIMA)¶

In an ARIMA model there are 3 parameters that are used to help model the major aspects of a times series: seasonality, trend, and noise. These parameters are labeled p,d,and q.

Number of AR (Auto-Regressive) terms (p): p is the parameter associated with the auto-regressive aspect of the model, which incorporates past values i.e lags of dependent variable. For instance if p is 5, the predictors for x(t) will be x(t-1)….x(t-5).

Number of Differences (d): d is the parameter associated with the integrated part of the model, which effects the amount of differencing to apply to a time series.

Number of MA (Moving Average) terms (q): q is size of the moving average part window of the model i.e. lagged forecast errors in prediction equation. For instance if q is 5, the predictors for x(t) will be e(t-1)….e(t-5) where e(i) is the difference between the moving average at ith instant and actual value.

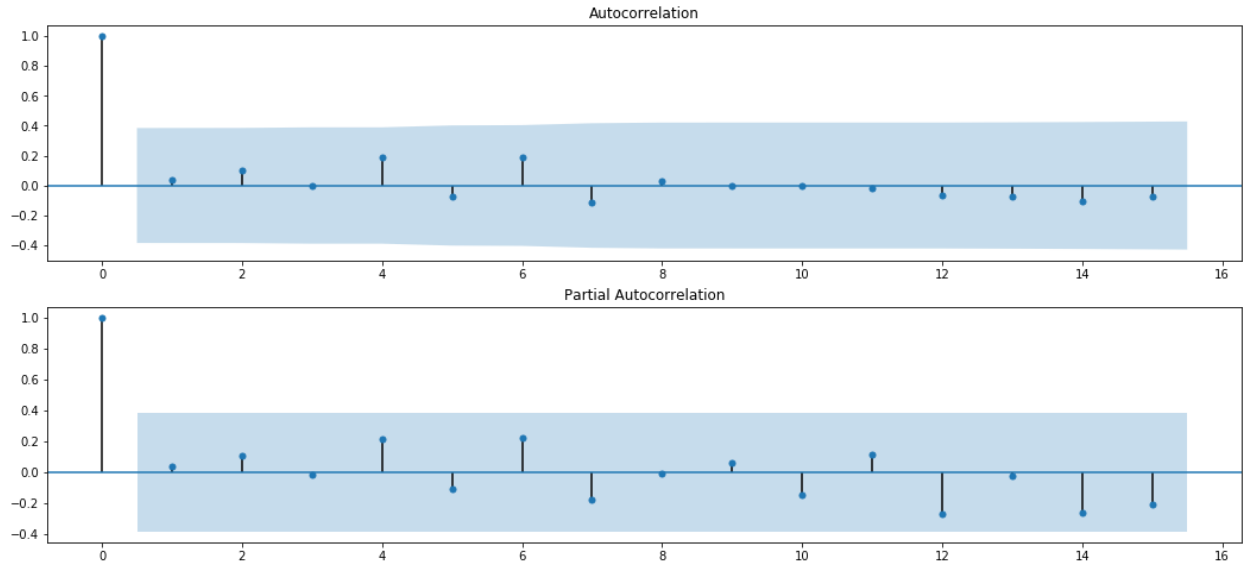
In [32]:

```
ts = confirmed.Confirmed - confirmed.Confirmed.shift()
ts.dropna(inplace=True)
```

In [33]:

```
pyplot.figure()
pyplot.subplot(211)
plot_acf(ts, ax=pyplot.gca(),lags=15)
pyplot.subplot(212)
```

```
plot_pacf(ts, ax=pyplot.gca(),lags=15)
pyplot.show()
```



Interpreting ACF plots

ACF Shape	Indicated Model
Exponential, decaying to zero	Autoregressive model. Use the partial autocorrelation plot to identify the order of the autoregressive model
---	---
Alternating positive and negative, decaying to zero Autoregressive model.	Use the partial autocorrelation plot to help identify the order.
---	---
One or more spikes, rest are essentially zero	Moving average model, order identified by where plot becomes zero.
---	---
Decay, starting after a few lags	Mixed autoregressive and moving average (ARMA) model.
---	---
All zero or close to zero	Data are essentially random.
---	---
High values at fixed intervals	Include seasonal autoregressive term.
---	---
No decay to zero	Series is not stationary

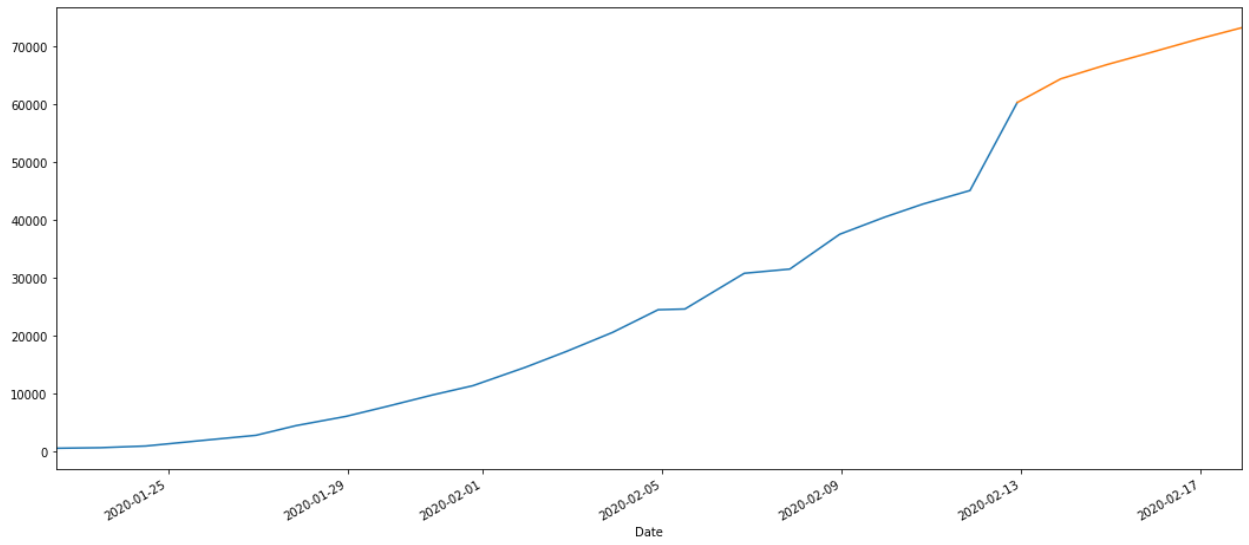
In [34]:

```
import datetime

train = confirmed[:int(0.75*(len(confirmed))+2)]
valid = confirmed[int(0.75*(len(confirmed)+1)):]

train['Confirmed'].plot()
valid['Confirmed'].plot()

#train.index = train.index.date
#valid.index = valid.index.date


#train.index = pd.to_datetime(train.index)
#valid.index = pd.to_datetime(valid.index)
#a=[]
#for i in range(len(train.index)):
#    a.append(datetime(train.index[i].year,train.index[i].month,train.index[i].day))
#print(type(a[0]))
#train = train.set_index([a])
#datetime(start_index_year,start_index_month,start_index_day)
```

```
train=train.reset_index()
valid=valid.reset_index()

train['Date'] = train['Date'].apply(lambda x: datetime.date(x.year,x.month,x.day))
valid['Date'] = valid['Date'].apply(lambda x: datetime.date(x.year,x.month,x.day))

train.set_index("Date", inplace = True)
valid.set_index("Date", inplace = True)


type(train.index[0])
```

Out[34]:

```
datetime.date
```



The graph of train and valid data set from the original series

IMPLEMENTING ARIMA.

Here we ahould keep the value of p=1 and d=1 and q=0 because we need d=1 for removing non stationarity . Ad we don't need q value because of significant change in variance of initial data.

In [35]:

```
from statsmodels.tsa.arima_model import ARIMA
from sklearn.metrics import mean_squared_error
from math import sqrt

model = ARIMA(train.values, order=(1, 1, 0))
model_fit = model.fit(disp=1)
```

In [36]:

```
model_fit.summary()
```

Out[36]:

ARIMA Model Results

Dep. Variable:	D.y	**No. Observations:**	21
Model:	ARIMA(1, 1, 0)	**Log Likelihood**	-199.275
Method:	css-mle	**S.D. of innovations**	3196.738
Date:	Fri, 06 Mar 2020	**AIC**	404.551

Time:	01:51:21	**BIC**	407.684
Sample:	1	**HQIC**	405.231

	coef	std err	z	P>|z|	[0.025	0.975]
const	2789.1108	632.165	4.412	0.000	1550.090	4028.132
ar.L1.D.y	-0.1411	0.406	-0.347	0.729	-0.938	0.655

Roots

	Real	Imaginary	Modulus	Frequency
AR.1	-7.0894	+0.0000j	7.0894	0.5000

Here we can change the value of p=1 to 0 because p>|z| is very high for ar parameter.

In [37]:

```
from sklearn.model_selection import TimeSeriesSplit

predictions = model_fit.predict()

predictions=predictions[len(predictions)-5:len(predictions)]
```

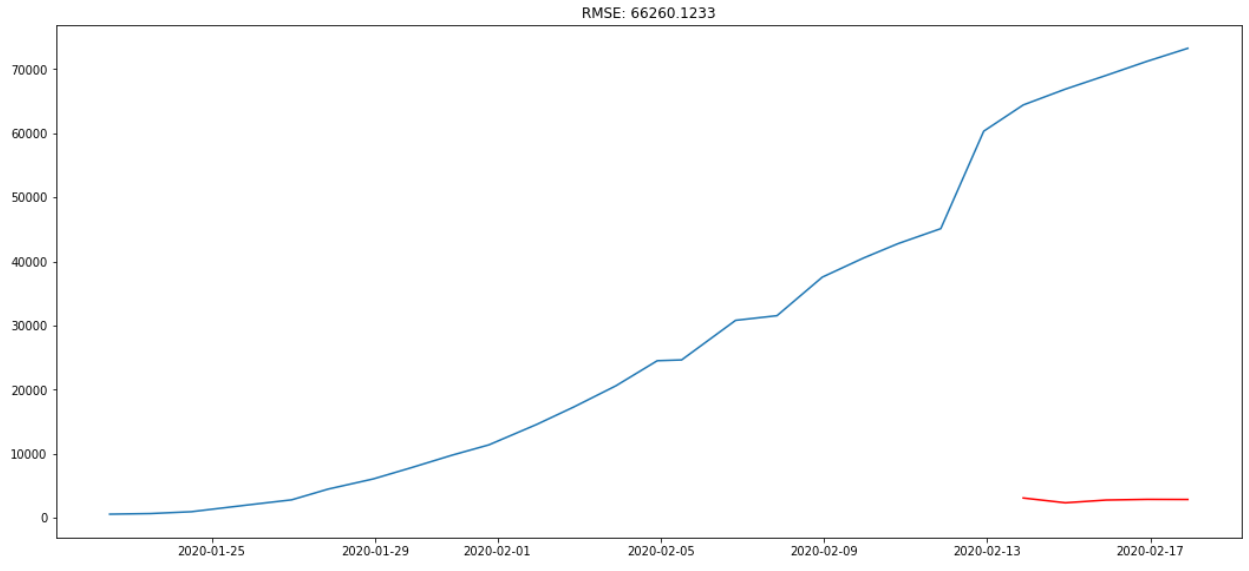
In [38]:

```
mse = mean_squared_error(confirmed[len(confirmed.Confirmed)-5:len(confirmed.Confirmed)],
predictions)
rmse = sqrt(mse)
print('RMSE: {}, MSE:{}'.format(rmse,mse))
```

```
RMSE: 66260.12330580161, MSE:4390403940.500034
```

In [39]:

```
l=confirmed.index[len(confirmed.Confirmed)-5:len(confirmed.Confirmed)]
a=pd.DataFrame()
a["predicted"]=predictions
a=a.set_index(l)
plt.plot(confirmed.Confirmed)
plt.plot(a.predicted, color='red')
plt.title('RMSE: %.4f'% rmse)
plt.show()
```



Reversing the calculated values.

In [40]:

```
predictions_ARIMA_diff = pd.Series(a.predicted, copy=True)
```

In [41]:

```
predictions_ARIMA_diff_cumsum = predictions_ARIMA_diff.cumsum()
```

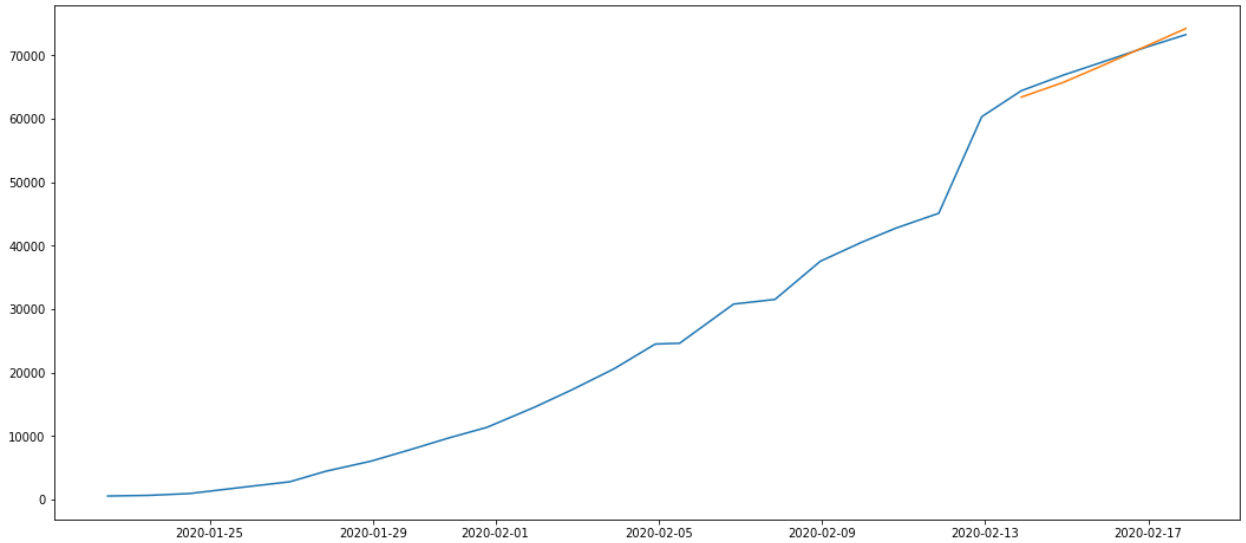
In [42]:

```
predictions_ARIMA_log = pd.Series(valid.Confirmed.iloc[0], index=valid.index)[1:len(predictions_ARI
MA_log )]
predictions_ARIMA_log = predictions_ARIMA_log[predictions_ARIMA_log.values==valid.Confirmed.iloc[0
]]
predictions_ARIMA_log = np.add(predictions_ARIMA_log.values,predictions_ARIMA_diff_cumsum.values)
predicted_log=pd.DataFrame()
predicted_log['predicted']=predictions_ARIMA_log
predicted_log=predicted_log.set_index(l)
```

In [43]:

```
plt.plot(confirmed.Confirmed)
plt.plot(predicted_log)

#plt.title('RMSE: %.4f'% np.sqrt(np.nansum(k)**2)/len(k))
o=(predicted_log).values
j=confirmed.Confirmed[len(confirmed.Confirmed)-5:len(confirmed.Confirmed)].values
k=[]
e=0
for i in range(len(o)):
    k.append(abs(o[i]-j[i])/j[i])
    e=e+k[i]*k[i]
print("RMS error = ", e)
```

```
RMS error =  [0.00077604]
```



Orange colour line in graph is predicted value. Blue colour line is original value.

we can see that the final predicted curve is almost as the given curve.And the error we got is very less.

Now using this less error model we can predict confirmed number of people those will be effected by corona virus in future.

hence a model is built by checking all the parameters that may effect our prediction

4 Summary

1. We are mainly concentrating on building a perfect time series model, given a sequence of data.
2. We have discussed what happening in every step, while building a model.
3. First step we should do is visualise the given sequence and see what type of data it is and also we need to check if the given sequence is seasonal or not.
4. Second step is Data preprocessing, where we check the stationarity of data. (refer here : 1)
5. If the data is stationary then no problem else we need to make the data stationary. (refer here : 1.1)
6. Non stationary data means its mean or variance or both depends on time, which in turn makes prediction difficult. So we make any data stationary first.(refer here : 1.2)
7. We have discussed some methods to check whether the data is Stationary or not. Also some techniques to make data stationary.(refer here : 1.2)
8.After getting the stationary data we will start building our models. They can be

    1. Auto regression (AR).
    2. Moving Average (MA) .
    3. Auto regressive Moving Average (ARMA).
    4. Auto regressive Integrated Moving Average (ARIMA).

9. We have discussed how Auto regression and moving average first .Both are important for a model to be good.
10. The moving average(MA) model does not uses the past forecasts to predict the future values whereas it uses the errors from the past forecasts. While, the autoregressive model(AR) uses the past forecasts to predict future values. (refer here : 2.2 and 2.1)
11. Later we discussed how these AR model and MA model can be combined to produce a simpler model known as Auto regressive Moving Average (ARMA). This is just the combination of above two models. contains two variables (p,q) which controls the AR and MA models here. We discussed how p,q values effect our model. (refer here : 2.3)
12. Why can't we add something to our ARMA model, to insert a variable(d) that can take care of non stationarity. Just to make life easier. Then Auto regressive Integrated Moving Average (ARIMA) was created. This ARIMA full form itself explains that. (refer here : 2.4)
13. Auto ARIMA is something where we just use ARIMA for different possible values of (p,d,q) and check which triple results less and error and good model.
14. Finally we got a good model that helps us in predicting the future values.
15. Also discussed an example of corona virus , where we are applying the above discussed format in building a time series model .(refer here : 3)

5 Fourier Transformation for time series analysis

Statement : Any continuous signal in the time domain can be represented uniquely and unambiguously by an infinite series of sinusoids.

The statement says that we can find a function like $f(t)$ which is a sum of an infinite series of sinusoids that will actually represent the sequence exactly.

$f(t) = \frac{a_0}{2} + \sum_{k=1}^{\infty}(a_k cos(2\pi kt) + b_k sin(2\pi kt))$

Here, for finding a_k and b_k coefficients we use Fourier transformation and get a group of coefficients.

In practise, we deal with values that are discretely sampled, usually at constant duration.

A classical fourier transform can be expressed as a Discrete Fourier transform (DFT),which converts a finite sequence of equally spaced samples of a function into a same length sequence of equally spaced sampled of the discrete time fourier transform.

$X_k = \sum_{n=0}^{N-1} x_n.e^{-2i\pi kn/N}$

where $e^{i\theta} = cos(\theta) + sin(\theta)$

if we plug this in above equation we get

$X_k = \sum_{n=0}^{N-1} x_n.[cos(2\pi kn/N) - isin(2\pi kn/N)]$

which will be of form $a + ib$ or $a - ib$.
hence we get

$a_k = \sum_{n=0}^{N-1} x_n.cos(2\pi kn/N)$

$b_k = -\sum_{n=0}^{N-1} x_n.sin(2\pi kn/N)$

Now substitute these coefficients in above f(t) gives our final function for discrete time series.Put your values of future time t and get the predicted value. Check error for validation set to verify your accuracy.
Note : Make sure your sequence to be an stationary or weak stationary.

Problem statement.

From World Health Organization - On 31 December 2019, WHO was alerted to several cases of pneumonia in Wuhan City, HubeiProvince of China. The virus did not match any other known virus. This raised concern because when a virus is new, we do not knowhow it affects people.So daily level information on the affected people can give some interesting insights when it is made available to the broader datascience community. data is available from 1/22/2020 to 2/17/2020.This notebook contain the model fitting of no of people who got corona virus and proved to be confirmedAnd then predict the future possible number of patients.Source of dataset: https://www.kaggle.com/sudalairajkumar/novel-corona-virus-2019-dataset

In [1]:

```
import numpy as np
import pylab as pl
from numpy import fft

import pandas as pd

df=pd.read_csv('./novel-corona-virus-2019-dataset_new/covid_19_data.csv',parse_dates=['Last
Update'])
df.columns = ['Sno','Date','Province/State','Country','Last Update','Confirmed','Deaths','Recovered
']
```

In [2]:

```
confirmed=df.groupby('Date').sum()['Confirmed'].reset_index()
confirmed.columns=['Date','Confirmed']
confirmed['Date']=pd.to_datetime(confirmed['Date'])
confirmed=confirmed.set_index("Date")
```

In [3]:

```
confirmed.head()
```

Out[3]:

Date	Confirmed
2020-01-22	555.0
2020-01-23	653.0
2020-01-24	941.0
2020-01-25	1438.0
2020-01-26	2118.0

extracted information as above format. All indicates deaths and recoverd cases too

In [4]:

```
a=pd.date_range(start="2020-01-22",end="2020-04-08")
p= [date_obj.strftime('%Y/%m/%d') for date_obj in a]
```

Function to find the fourier function and predicted values . We need two parameters as input. One the sequence we need to analyse. The other is number of predicted values we need to find. I am giving number of harmonics here as 10 . which indicates i am using most 10 frequencies that influence the function. Later the same stuff as detrending and shifting from time domain to frequency domain .

Here we write one functions depend on trend. This removes trend with 3rd degree polynomial and add to predicted values at the end of the function while returning. This can be linear or quadratic if it returns a good value of predictions. But 3rd degree covers both linear ad quadratic, So i used 3rd degree here. So we are detrending in frequecy domain using fft.fft() function.

here n_predict variable is number of values we need to predict.

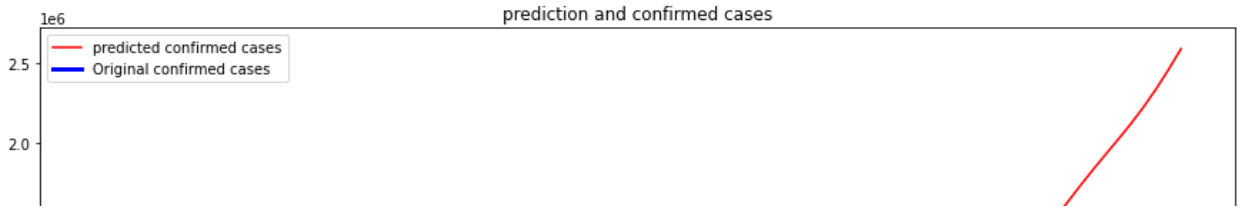
In [5]:

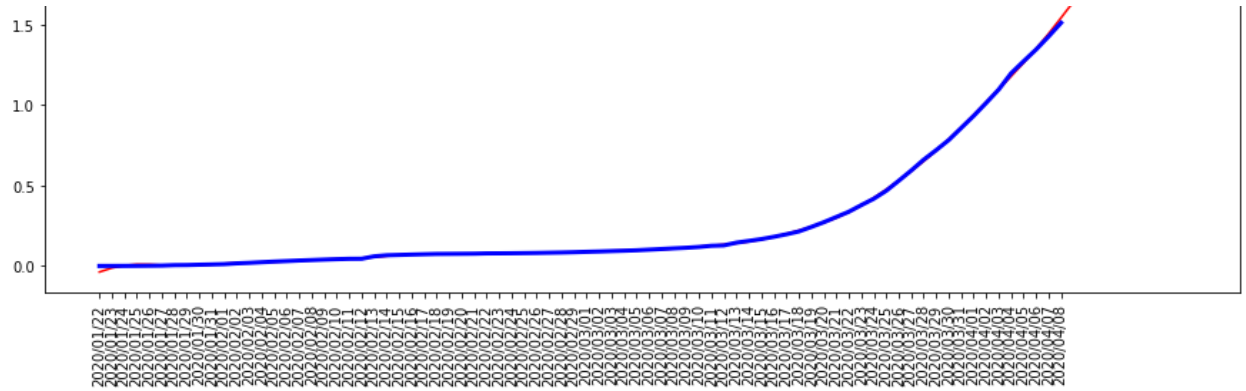
```
def fourierT(x, n_predict):
    n = x.size
    n_harm = 10   # number of harmonics in model

    #taking t as n integers from 1 to n
    t = np.arange(0, n)
    #find trend using polyfit , i adopted degree 3
    p = np.polyfit(t, x, 3)
    #Subtract the trend found above
    x_notrend = x - ((p[0] * (t*t*t)) + (p[1]*(t**2)) + (p[2]*t)+p[3])         # detrended x
    # detrended x in frequency domain
    x_freqdom = fft.fft(x_notrend)  #Return the Discrete Fourier Transform sample frequencies.
    f = fft.fftfreq(n)              # Return the Discrete Fourier Transform sample frequencies.


    indexes = list(range(n))
    indexes = np.array(range(0,n))
    indexes=sorted(indexes,key = lambda i: np.absolute(f[i]))

    t = np.arange(0, n + n_predict)
    restored_sig = np.zeros(t.size)  #initialize to zero
    for i in indexes[:1 + n_harm * 2]:
        ampli = np.absolute(x_freqdom[i]) / n   # amplitude calculation
        phase = np.angle(x_freqdom[i])          # phase calculation
        restored_sig += ampli * np.cos(2 * np.pi * f[i] * t + phase)  # restored sig is Acos(2*pi*f*
t + phase)
    return restored_sig + ((p[0] * (t*t*t)) + (p[1]*(t**2)) + (p[2]*t)+p[3])  # add the trend and re
turn
```

main function to import data set and extract required dataframe form. And then plotting the sequence and actual values.

In [6]:

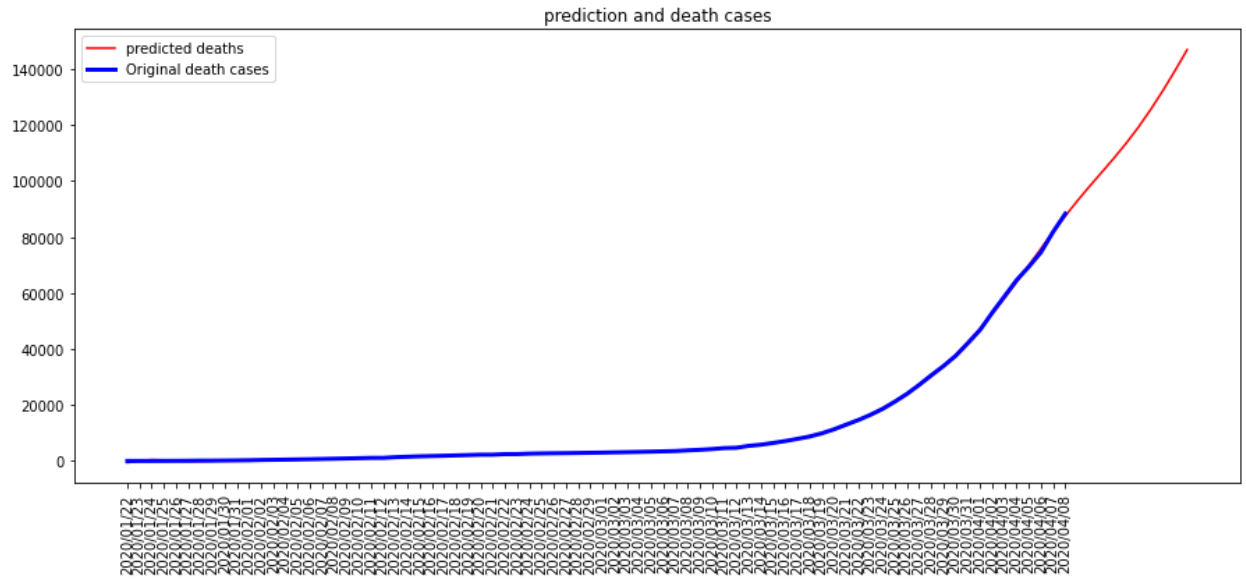
```
def main():
    df=pd.read_csv('./novel-corona-virus-2019-dataset_new/covid_19_data.csv',parse_dates=['Last
Update'])
    df.columns = ['Sno','Date','Province/State','Country','Last Update','Confirmed','Deaths','Recov
ered']
    confirmed=df.groupby('Date').sum()['Confirmed'].reset_index()
    confirmed.columns=['Date','Confirmed']
    confirmed['Date']=pd.to_datetime(confirmed['Date'])
    confirmed=confirmed.set_index("Date")
    x=confirmed['Confirmed']
    x = confirmed['Confirmed'].tolist()

    x=np.array(x)
    n_predict = 10
    predictions = fourierT(x, n_predict)

    fig = pl.figure(figsize=(15,6))
    ax1 = fig.add_subplot()
    ax1.tick_params(axis ='x', rotation = 90)
    pl.title("prediction and confirmed cases")
    pl.plot(np.arange(0, predictions.size), predictions, 'r', label = 'predicted confirmed cases')
    pl.plot(p, x, 'b', label = 'Original confirmed cases', linewidth = 3)
    pl.legend()
    pl.show()

if __name__ == "__main__":
    main()
```





In [7]:

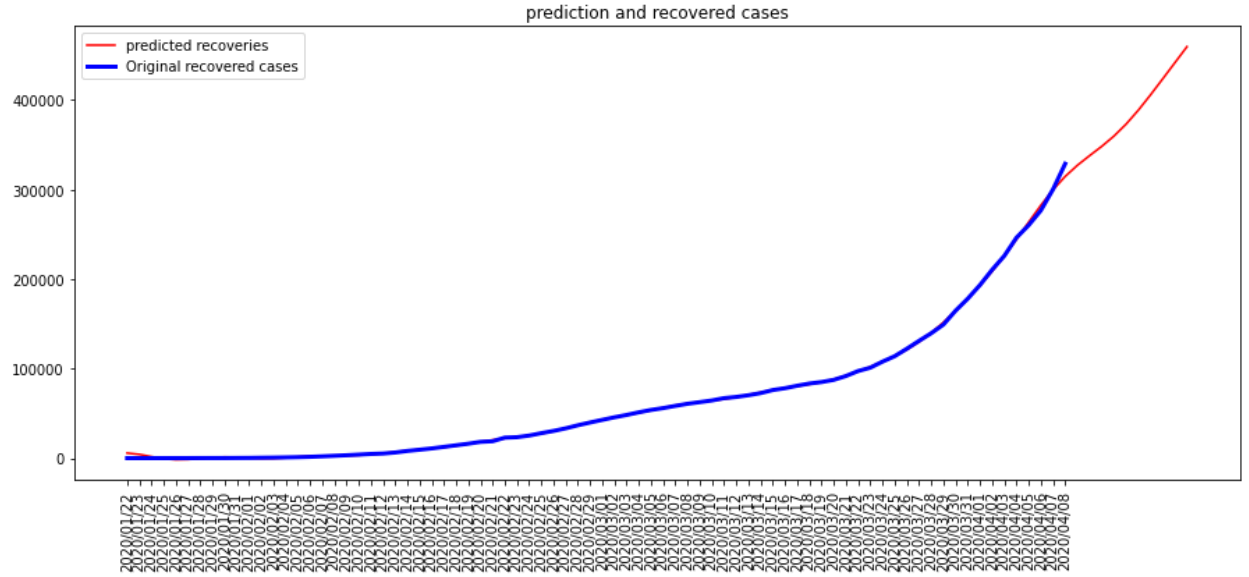
```
def main():
    df=pd.read_csv('./novel-corona-virus-2019-dataset_new/covid_19_data.csv',parse_dates=['Last
Update'])
    df.columns = ['Sno','Date','Province/State','Country','Last Update','Confirmed','Deaths','Recov
ered']
    deaths=df.groupby('Date').sum()['Deaths'].reset_index()
    deaths.columns=['Date','Deaths']
    deaths['Date']=pd.to_datetime(deaths['Date'])
    deaths=deaths.set_index("Date")
    x=deaths['Deaths']
    x = deaths['Deaths'].tolist()

    x=np.array(x)
    n_predict = 10
    predictions = fourierT(x, n_predict)

    fig = pl.figure(figsize=(15,6))
    ax1 = fig.add_subplot()
    ax1.tick_params(axis ='x', rotation = 90)
    pl.title("prediction and death cases")
    pl.plot(np.arange(0, predictions.size), predictions, 'r', label = 'predicted deaths')
    pl.plot(p, x, 'b', label = 'Original death cases', linewidth = 3)
    pl.legend()
    pl.show()

if __name__ == "__main__":
    main()
```



In [8]:

```
def main():
    df=pd.read_csv('./novel-corona-virus-2019-dataset_new/covid_19_data.csv',parse_dates=['Last
Update'])
```

```
update '])
    df.columns = ['Sno','Date','Province/State','Country','Last Update','Confirmed','Deaths','Recov
ered']
    recovered=df.groupby('Date').sum()['Recovered'].reset_index()
    recovered.columns=['Date','Recovered']
    recovered['Date']=pd.to_datetime(recovered['Date'])
    recovered=recovered.set_index("Date")
    x=recovered['Recovered']
    x = recovered['Recovered'].tolist()

    x=np.array(x)
    n_predict = 10
    predictions = fourierT(x, n_predict)


    fig = pl.figure(figsize=(15,6))
    ax1 = fig.add_subplot()
    ax1.tick_params(axis ='x', rotation = 90)
    pl.title("prediction and recovered cases")
    pl.plot(np.arange(0, predictions.size), predictions, 'r', label = 'predicted recoveries')
    pl.plot(p, x, 'b', label = 'Original recovered cases', linewidth = 3)
    pl.legend()
    pl.show()

if __name__ == "__main__":
    main()
```



6 Summary

MAIN IDEA: main idea here is converting our given sequence(x(n) into frequency domain (X(n)),where the values are expressed using frequencies and phase. Find the next possible value using frequencies and phase. And the reconvert them into normal space domain again, which is our required time series analysis.

1. Any sequence of data can be represented as sum of set of sinusoids with different amplitudes, phases and frequencies.

2.Find the polynomial that nearly fits the data given using np.polyfit (refer here : 6).

3.After finding the polynomial that fits the given data (Can be of any degree, as I used 3rd degree polynomial here) and subtract it from data results a detrended sequence.(For making the data stationary)

4.numpy.fft.fft is a function which computes the one-dimensional n-point discrete Fourier Transform (DFT) with the efficient Fast Fourier Transform (FFT) algorithm [CT].(refer here 7)

5.Using this fft.fft we calculate the values of amplitudes in frequency domain.
Let $X(t)$ be the frequency domain values and $x(t)$ is the spacial domain values , then we are calculating

$$X(k) = \sum_{n=0}^{N-1} x(n)e^{-j2\pi nk/N} \qquad \text{where k=0,1,2..N-1.}$$

6. fft.fftfreq is a function which returns Discrete Fourier Transform sample frequencies. These are helpful to adjust them with the number of harmonics we are using.

7. Now restore the values of spacial domain again using the for loop iterates over the given sequence and the future values we need to find (By calculating the values of Amplitude and phase) and substitute those amplitudes and phase in $Acos(2\pi ft + \phi)$.

This is nothing but reversing those frequency domain values again into spacial domain values.

$$x(k) = \frac{1}{N} * \sum_{n=0}^{N-1} X(k)e^{j2\pi nk/N} \qquad \text{for } 0 <= n <= N-1$$

8. Add the detrended component that we removed in from point 3 And plot those new values gives the predicted values.

Example for fft.fftfreq:
let $array([8.+0.j, 0.+0.j, 0.-4.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+4.j, 0.+0.j])$ this be the returned value of fft.fft()
the result has nonzero values at indices 0, 2 and 6. There are 8 elements. This means
$y = \frac{8e^{2\pi it/8*0} - 4ie^{2\pi it/8*2} + 4ie^{2\pi it/8*6}}{8}$

SIMPLE EXAMPLE OF FOURIER TRANSFORM ANALYSIS:

In [1]:

```
import numpy as np
import pylab as pl
from numpy import fft
import pandas as pd
```

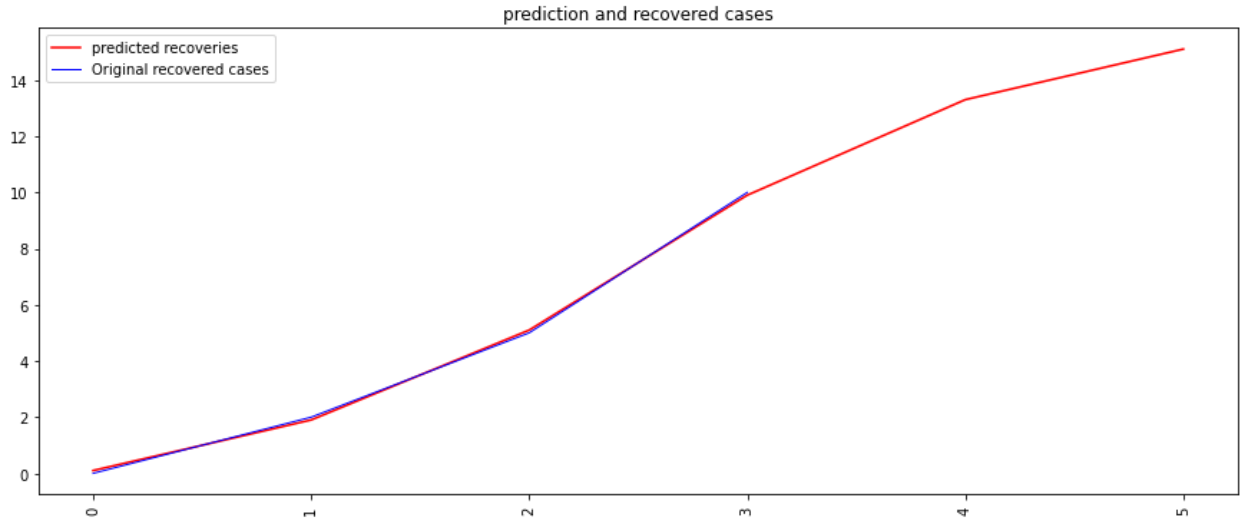
In [2]:

```
def fourierT_test(x, n_predict):
    n = x.size
    n_harm = 1   # number of harmonics in model

    #taking t as n integers from 1 to n
    t = np.arange(0, n)
    #find trend using polyfit , i adopted degree 3
    p = np.polyfit(t, x, 1)
    print("the coefficients of nearly fitted polynomial :",p)
    #Subtract the trend found above
    x_notrend = x - ((p[0] * (t)) + (p[1]))        # detrended x
    print("no trend data : " ,x_notrend)
    # detrended x in frequency domain
    x_freqdom = fft.fft(x_notrend)  #Return the Discrete Fourier Transform sample frequencies.
    print("frequency domain values : " ,x_freqdom)

    f = fft.fftfreq(n)              # Return the Discrete Fourier Transform sample frequencies.
    print("sample frequency values : ",f)

    indexes = list(range(n))
    indexes = np.array(range(0,n))
    indexes = sorted(indexes,key = lambda i: np.absolute(f[i]))
    print("sorted frequencies indexes :", indexes)
    t = np.arange(0, n + n_predict)
    restored_sig = np.zeros(t.size)  #initialize to zero
    print("These frequencies are choosen as our main frequecies which are useful to convert to mai
n space domain sequence",indexes[:1 + n_harm * 2])
    for i in indexes[:1 + n_harm * 2]:
        ampli = np.absolute(x_freqdom[i]) / n   # amplitude calculation
        print("amplitude : ",ampli)
        print()
        phase = np.angle(x_freqdom[i])          # phase calculation
        print("phase : ",phase)
        print()
        restored_sig += ampli * np.cos(2 * np.pi * f[i] * t + phase)  # restored sig is A*cos(2*pi*f
*t + phase)
        print()
    print("final restored-sig values are :" , restored_sig)
    print("final values are ",restored_sig + ((p[0]*t)+p[1]))
    return restored_sig + ((p[0]*t)+p[1])  # add the trend and return
```

In [3]:

```
def main():
    x=[0,2,5,10]

    x=np.array(x)
    n_predict = 2
    predictions = fourierT_test(x, n_predict)


    fig = pl.figure(figsize=(15,6))
    ax1 = fig.add_subplot()
    ax1.tick_params(axis ='x', rotation = 90)
    pl.title("prediction and recovered cases")
    pl.plot(np.arange(0, predictions.size), predictions, 'r', label = 'predicted recoveries')
    pl.plot([0,1,2,3], x, 'b', label = 'Original recovered cases', linewidth = 1)
    pl.legend()
    pl.show()

if __name__ == "__main__":
    main()
```

```
the coefficients of nearly fitted polynomial : [ 3.3 -0.7]
no trend data :  [ 0.7 -0.6 -0.9  0.8]
frequency domain values :  [-1.66533454e-15+0.j   1.60000000e+00+1.4j -4.00000000e-01+0.j
  1.60000000e+00-1.4j]
sample frequency values :  [ 0.    0.25 -0.5  -0.25]
sorted frequencies indexes : [0, 1, 3, 2]
These frequencies are choosen as our main frequecies which are useful to convert to main space dom
ain sequence [0, 1, 3]
amplitude :  4.163336342344337e-16

phase :  3.141592653589793


amplitude :  0.5315072906367326

phase :  0.7188299996216253


amplitude :  0.5315072906367326

phase :  -0.7188299996216253


final restored-sig values are : [ 0.8 -0.7 -0.8  0.7  0.8 -0.7]
final values are  [ 0.1  1.9  5.1  9.9 13.3 15.1]
```



In [ ]:

I have taken our input as $[0, 2, 5, 10]$ and we need to find the next two elements.
For this we use the above algorithm and explaining what happens in above algorithm.
A Best linear line which passes through the four points is $[3.3x - 0.7]$ which is calculated using the function np.polyfit()
Next we subtract this from $[0, 2, 5, 10]$ which gives detrended values.

$$[0, 2, 5, 10] - [-0.7, 2.6, 5.9, 9.2] = [0.7, -0.6, -0.9, 0.8]$$

Next we calculate Frequency domain values using

$$X(k) = \sum_{n=0}^{N-1} x(n)e^{-j2\pi nk/N} \qquad \text{where k=0,1,2..N-1.}$$

which gives frequency domain values as $X[i]$. for $i = 0, 1, 2, 3$
For example lets calculate $X[1]$.

$$X[1] = \sum_{0}^{3} x[n].e^{-j(1)2\pi n/N}$$
$$X[1] = x[0]e^{0} + x[1]e^{-2\pi j/4} + x[2]e^{-2\pi j2/4} + x[3]e^{-2\pi j3/4}$$

Substituting the values gives

$$x[0] + x[1].(-j) + x[2].[-1] + x[3][j]$$
$$0.7 + 0.9 + j(0.8 + 0.6)$$
$$1.6 + j1.4$$

Which is same as we obtained from the code.

Now calculating the sample frequencies associated with above calculated X[] values using fft.fftfreq()
The possible sample frequencies for $n = 4$ are $[0, 0.25, -0.5, -0.25]$.
Now I assigned numbers to them in increasing order of their absolute values as $[0, 1, 3, 2]$
Now depends on number of harmonics we are using we choose our frequencies as 2*no.of.harmonics+1. from the above sample frequencies.
Now take an array t, which contains $[0, 1, 2, 3, 4, 5]$ where 0,1,2,3 are from inputs and 4,5 are for the values we are going to predict. Here this helps in predicting the future values. We also take an array restored-sig =[0,0,0,0,0,0] , same as t size and contains all zeroes. We update this restore-sig for every iterate of those 2*no.of.harmonics+1 no of frequencies and update the value of this. Finally we get our predicted values.
restored-sig updation : We convert frequency domain values into space domain using $x(k) = \frac{1}{N} * \sum_{n=0}^{N-1} X(k)e^{j2\pi nk/N}$ for $0 <= n <= N - 1$ but we only need the real part. So calculate the only real value $A\cos(2\pi ft + phase)$ and go on adding these terms over all the iterations.

amplitude1 : 4.163336342344337e-16
phase1 : 3.141592653589793

amplitude2: 0.5315072906367326

phase2 : 0.7188299996216253

amplitude3 : 0.5315072906367326
phase3 : -0.7188299996216253

Final restored-sig is our predicted value. Which we got in our code as [0.8 -0.7 -0.8 0.7 0.8 -0.7] by adding all the above obtained Acos() values.

Finally we add our detrended term again and this will be our predicted value.
$[0.8, -0.7, -0.8, 0.7, 0.8, -0.7]+[-0.7, 2.6, 5.9, 9.2, 12.5, 15.8] = [0.1, 1.9, 5.1, 9.9, 13.3, 15.1]$

Hence Predicted

References

[1] Douglas C Montgomery ,Cheryl L.Jennings, Murat Kulahsi. Introduction to time series analysis and forecasting .

[2] Robert H.Shamway , David S.Stoffer. Time Seres analysis and Its applications

[3] https://github.com/advaitsave/Introduction-to-Time-Series-forecasting-Python

[4] https://towardsdatascience.com/fourier-transformation-and-its-mathematics-fff54a6f6659

[5] https://docs.scipy.org/doc/numpy/reference/generated/numpy.fft.fftfreq.html

[6] https://docs.scipy.org/doc/numpy/reference/generated/numpy.polyfit.html

[7] https://docs.scipy.org/doc/numpy/reference/generated/numpy.fft.fft.html