**Time Series Analysis**

Introduction:

Time series data is data that is collected at different points in time. This is opposed to cross-sectional data which observes individuals, companies, etc. at a single point in time. Because data points in time series are collected at adjacent time periods there is potential for correlation between observations. This is one of the features that distinguishes time series data from cross-sectional data.



Time series analysis is a statistical technique that deals with time series data, or trend analysis. Time series data means that data is in a series of particular time periods or intervals.

**Basics of Time Series Forecasting:**

Time series forecasting is the use of statistical methods to predict future behaviour based on historical data. Here we have some example of Time Series Forecasting:

We have some time wise Stock Price data like:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 10am | 11am | 12pm | 1pm | 2pm | 3pm | 4pm | 5pm |
| 239 | 456 | 546 | 867 | 498 | 643 | 876 | ? |



In this data we have shown stock prices of particular time interval and what will be the future Stok Price at 5pm?

For solve this problem we want to take average of 2 recent observations before and before that and put the wattage to before observation 0.8 and 0.2 to before that observation and calculate the average.

Like: (643\*0.2) + (876\*0.8) / 2 = 414.7

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 10am | 11am | 12pm | 1pm | 2pm | 3pm | 4pm | 5pm |
| 239 | 456 | 546 | 867 | 498 | 643 | 876 | 414.7 |

At 5pm future stock price will be 414.7

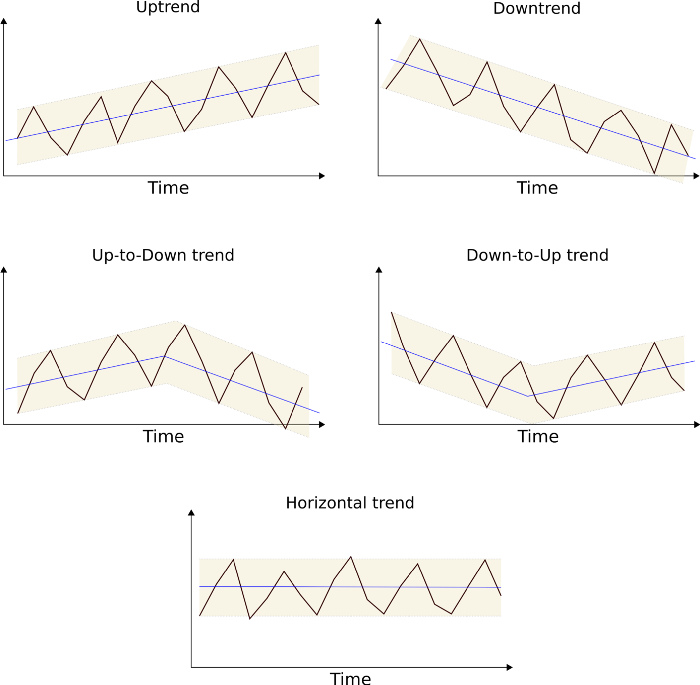


**Time Series Patterns**

Most time series data usually have at least one of these three kinds of patterns: trend, seasonality, and/or cycles. Let’s briefly describe each one.

## **Trend**

## The trend describes the general behaviour of a time series. If a time series manifests a positive long-term slope over time, it has an upward trend. If instead, it describes a general negative slope, it has a downtrend. The overall trend may also change direction. There can be an up-to-down trend or a down-to-up trend. Lastly, a stationary or horizontal trend defines a time series with neither positive nor negative long-term patterns.



## **Seasonality**

A seasonal pattern is any kind of fluctuation (change) in a time series that is caused by calendar-related events. These events can be the time of year (like winter or summer), or the time of day or the week. Seasonality always has fixed frequencies. That is, a seasonal pattern always starts and ends in the same period of a week, year, etc. Take a data centre as an example. If we consider the cooling system as the primary source of energy consumption, it is easy to imagine that in the summer, energy costs probably go up, while winter might show a decrease in energy consumption. Also, a clothing store that sells heavy coats might observe higher selling rates during winter, as opposed to the summer.

## **Cycle**

Lastly, a cyclical pattern in a time series is a kind of change that is not related to seasonal factors. These are rises and falls with non-fixed magnitudes that can last for more than a calendar year. Cyclical patterns are not repetitive. Usually, they result from external factors which make them much harder to predict.

# Time Series Smoothing in python

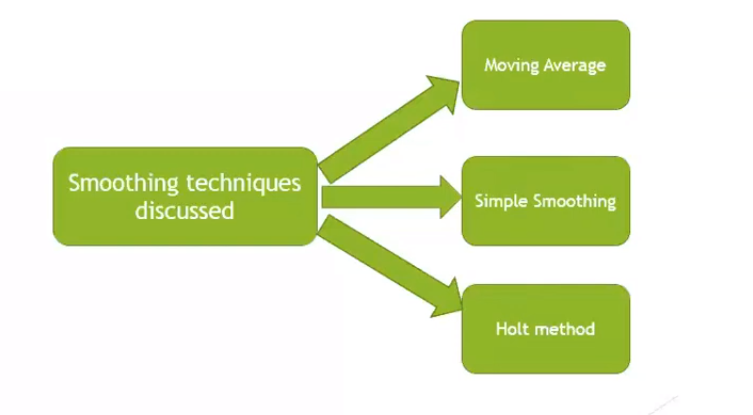
Simple Exponential Smoothing is used for time series prediction when the data particularly  does not follow any:

1. Trend: An upward or downward slope
2. Seasonality: Shows a particular pattern due to seasonal factors like Hours, days, Year, etc.

SES works on weighted averages i.e. the average of the previous level and current observation. Largest weights are associated with the recent observations and the smallest weights are associated with the oldest observations.

The decrease in weight is controlled by the smoothing parameter which is known as 𝜶(alpha) here. 𝜶(alpha) value can be between 0 to 1:

* 𝜶(alpha)=0: Means that forecast for future value is the average of historical data.
* 𝜶(alpha)=1: Means that forecast for all future value is the value of the last observation



**Import Important Libraries**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import warnings

warnings. filterwarnings('ignore')

import os

from statsmodels.tsa.api import ExponentialSmoothing,SimpleExpSmoothing,Holt

Simple Exponential Smoothing (SES) is defined under the statsmodels library of python and like any other python library we can install statsmodel using pip install statsmodel.

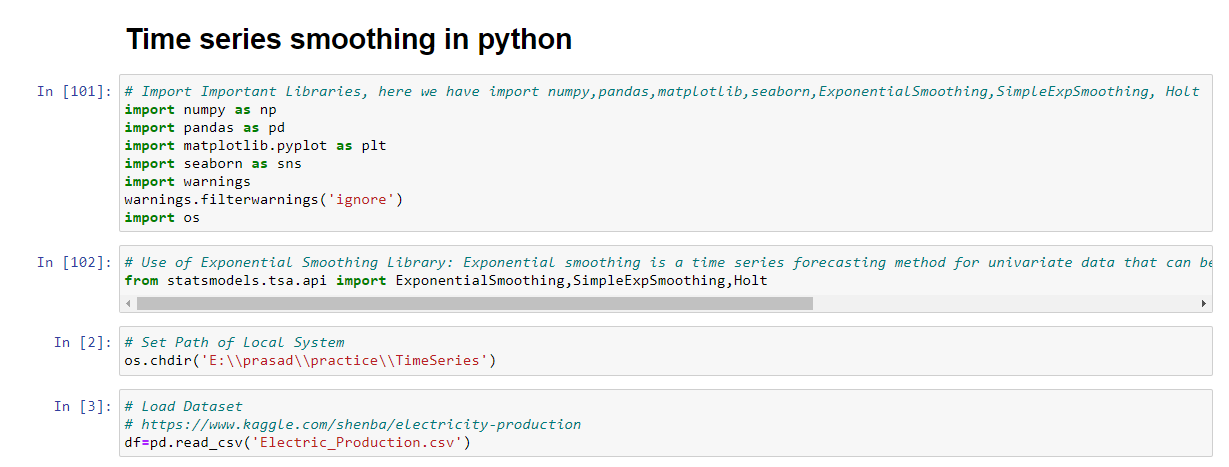
Use of Exponential Smoothing Library: Exponential smoothing is a time series forecasting method for univariate data that can be extended to support data with a systematic trend or seasonal component.

**Load Dataset:**

Here we use 'Electric\_Production.csv' from Kaggle.com. In this dataset having 397 rows and 2 columns. In this dataset null values are not present. First column is Date and second column name is Units. Dates Start from 01-01-1985 to 01-01-2018

Link: <https://www.kaggle.com/shenba/electricity-production>

df=pd.read\_csv('Electric\_Production.csv')



In this dataset two columns are available Date and Units. This two columns data types are different ‘Object’ and ‘float64’.

# Check the type of Date Column

type(df['DATE'])

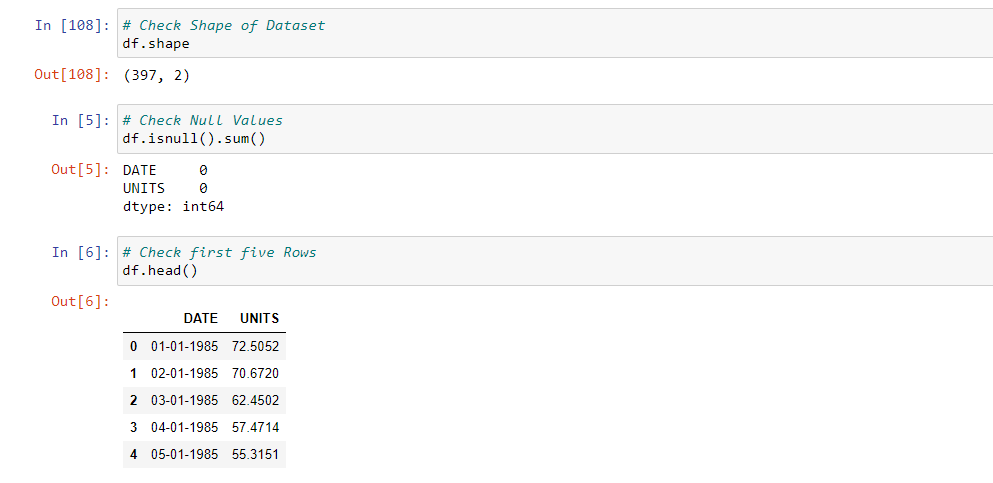
# Check Data types of Df

df.dtypes

we need to convert data type ‘Object’ into ‘datetime’ of ‘Date’ columns.

# Convert Date Column in datetime format

df['DATE']=pd.to\_datetime(df['DATE'])



**Creating Time Series of the Data:**

To visualize and analyse the data let us we create a time series of this dataset.

We need to set the index as date column. In time series analysis our index must be date & time format.



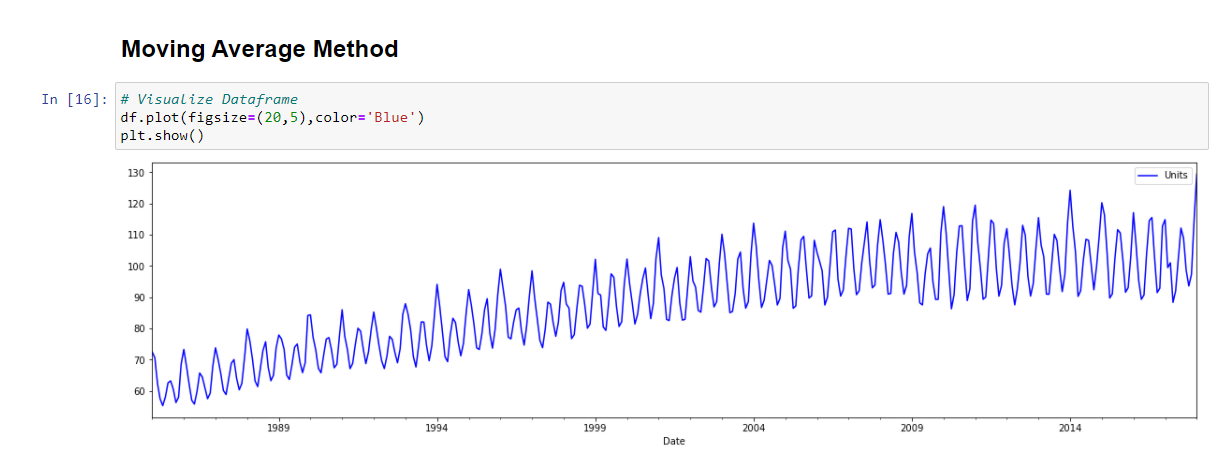
**Visualize the Data**

Now we analyse this data by using Matplotlib.pyplot as plt library.

# Visualize Dataframe

df.plot(figsize=(20,5))

plt.show()



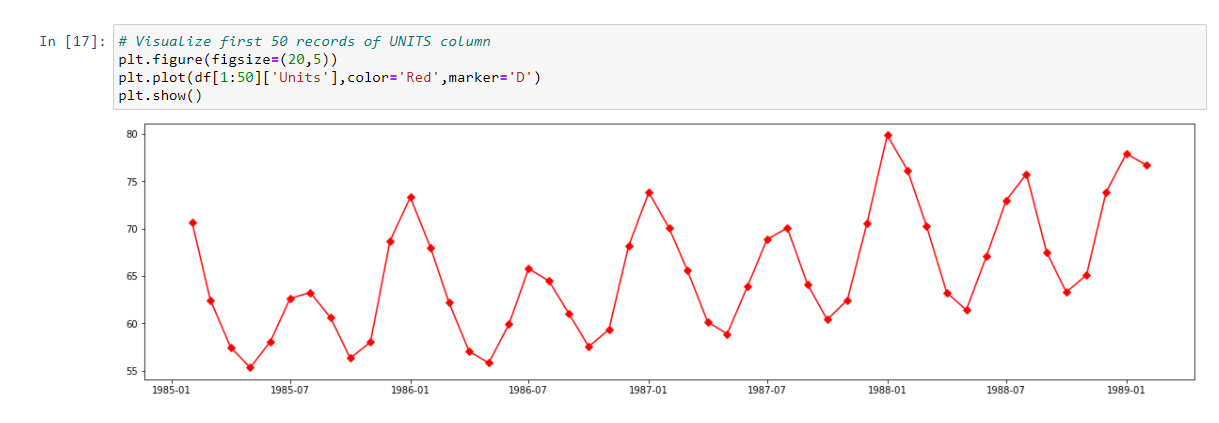
As per this visualization index ‘Date’ on X axis and Units are present on y-axis. Data clearly shown there is a particular trend and seasonality in units. For better understanding we plot only first 50 units.

# Visualize first 50 records of UNITS column

plt.figure(figsize=(20,5))

plt.plot(df[1:50]['Units'],color='Red')

plt.show()

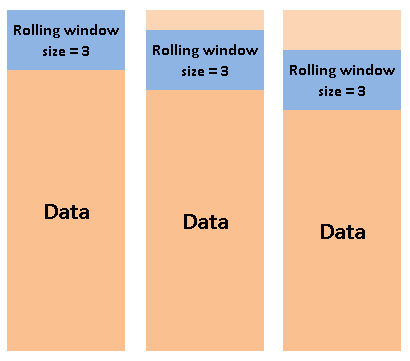


Seasonality and trend clearly shown in this visualization.

## **Moving Average Method**

### **Rolling Average Transform**

Rolling is a very useful operation for time series data. Rolling means creating a rolling window with a specified size and perform calculations on the data in this window which, of course, rolls through the data. The figure below explains the concept of rolling.



It is worth noting that the calculation starts when the whole window is in the data. In other words, if the size of the window is three, the first aggregation is done at the third row. Let’s apply rolling with window size=5 to the dataframe we created in the previous part: If we set the window size is 5 that time we get the average of first five units on fifth row.

**Code:**

rolling\_series=df[1:50].rolling(window=5)

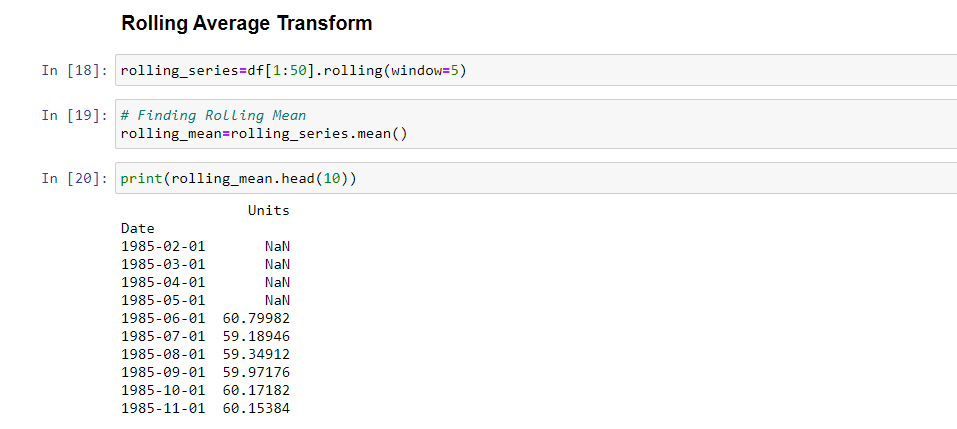
# Finding Rolling Mean

rolling\_mean=rolling\_series.mean()

print(rolling\_mean.head(10))

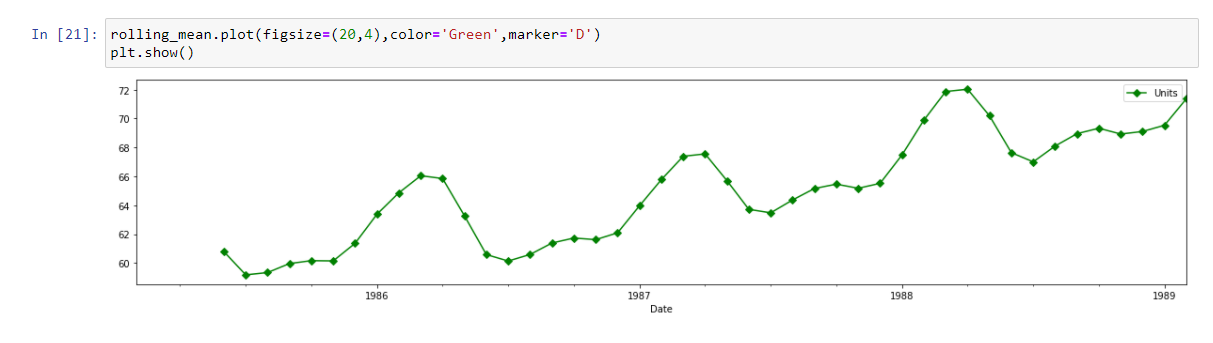
rolling\_mean.plot(figsize=(20,4),color='Green',marker='D')

plt.show()



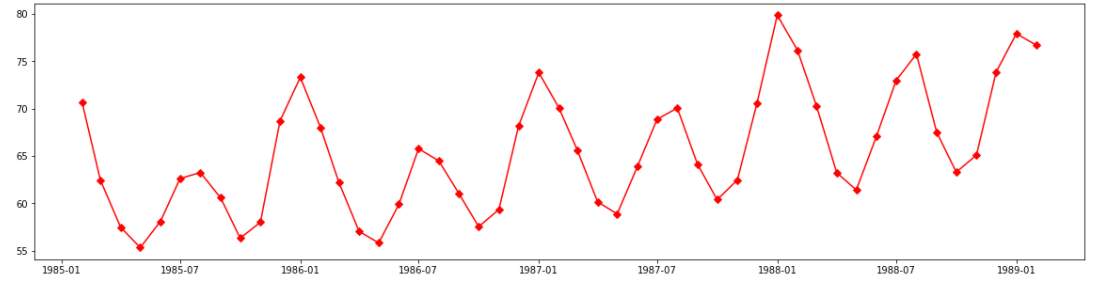
The value on 1985–06–01 after rolling is the mean of the first 5 values in the original data. The value on 2013–07–01 is the mean of the values from the second to sixth in the original data and so on.

**Visualize Data after Rolling**

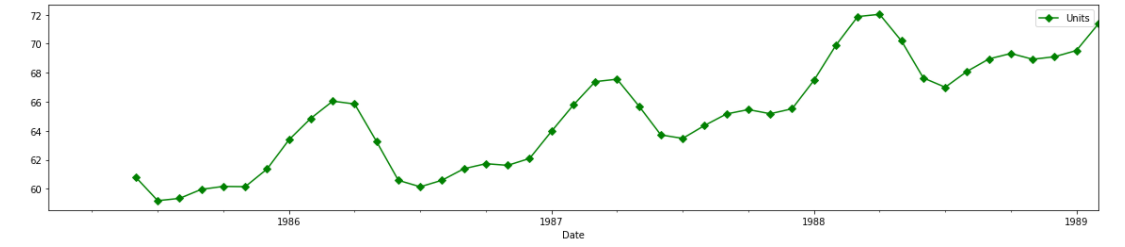


In this figure data visualization is some Smooth as compare the before Rolling visualization. If we need to create smooth visualization and data smoothing so that time we need to apply rolling technique and set the window size as per requirement.

**Before Apply Rolling**

****

**After Apply Rolling**

****

## As per both visualizations there is more difference between two plots. Here ‘Red’ is an original time series data and ‘Green’ is a Moving Average time series data. In first visualization there is more zigzag because that is an original data but after applying rolling technique on second plot there is zigzag and movement has been smoothing that is the main use of moving average rolling technique smooth the data from zigzag. If we increase the size of window from 5 to 10 that time visualization will be the smoother as compere to window 5.

## **Simple Exponential Smoothing:**

For creating a prediction model using SES we should have a 𝜶(alpha) value which we discussed in the beginning. Here we will create three instances in which we will take three different 𝜶(alpha) values as:

1. 𝜶(alpha) = 0.2
2. 𝜶(alpha) = 0.8
3. 𝜶(alpha) value automatically optimized by statsmodel which is the recommended one.

We will pass the data into Simple Exponential Smoothing and fit the data with different values of the Smoothing Level.

Here we take first 50 records from dataset. And create fit1 and fit2 variables including SimpleExpSmoothing.

In fit1 variable we have set the alpha value is 0.2 and fit2 variable we have set alpha value is 0.8

**Code:**

warnings.filterwarnings('ignore')

data=df[1:50]

fit1=SimpleExpSmoothing(data).fit(smoothing\_level=0.2,optimized=False)

fit2=SimpleExpSmoothing(data).fit(smoothing\_level=0.8,optimized=False)

plt.figure(figsize=(20,5))

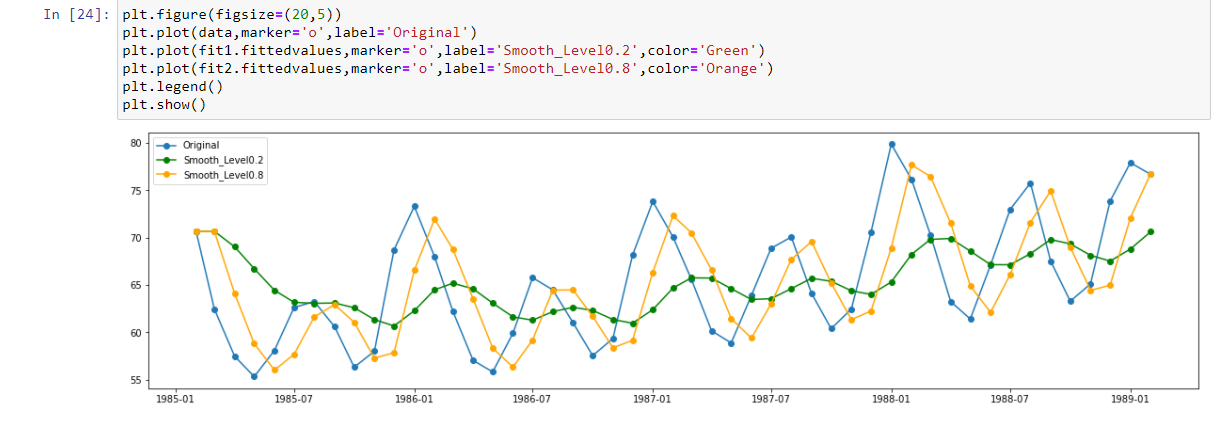
plt.plot(data,marker='o',label='Original')

plt.plot(fit1.fittedvalues,marker='o',label='Smooth\_Level0.2',color=Green)

plt.plot(fit2.fittedvalues,marker='o',label='Smooth\_Level0.8',color=Orange)

plt.legend()

plt.show()



In this plot, we can see that the ‘Blue’ curve is the actual distribution of the data, other than that the ‘Orange’ curve is the most accurate as it is plotted according to the optimized value determined by the statsmodel itself. As per smooth level 0.2 is pretty much smoother than original and alpha 0.2 that colour is ‘Green’. If we give more weightage (0.8) on recent observation that time curve will be follow our original observation here our original curve is ‘Blue’ and ‘Orange’ is more weightage of exponential smoothing curve. And ‘Green’ is less weightage (0.2) of recent observation curve.

Other than Simple Exponential Smoothing there are many other Exponential Smoothing models that work for time series prediction, namely:

* Holt’s Method: This method is used when the data shows a particular trend like an upward or a downward slope. Here we have two smoothing equations one for level and the other one for trend.
* Holt’s Winter Method:  This method is used when data has a certain trend and seasonality also for eg. data shows an upward trend and that too in a particular month. Similar to the other exponential smoothing methods it has smoothing parameters like level, Trend. Also, a third parameter seasonality is also added in this model.

## **Holt Method for Exponential Smoothing**

Holt's Linear Trend Method This is accomplished by adding a second single exponential smoothing model to capture the trend (either upwards or downwards).

**Code:**

fit3=Holt(data).fit() #Linear Trend means No Exponential Trend

fit4=Holt(data,exponential=True).fit() # exponential Trend

plt.figure(figsize=(20,5))

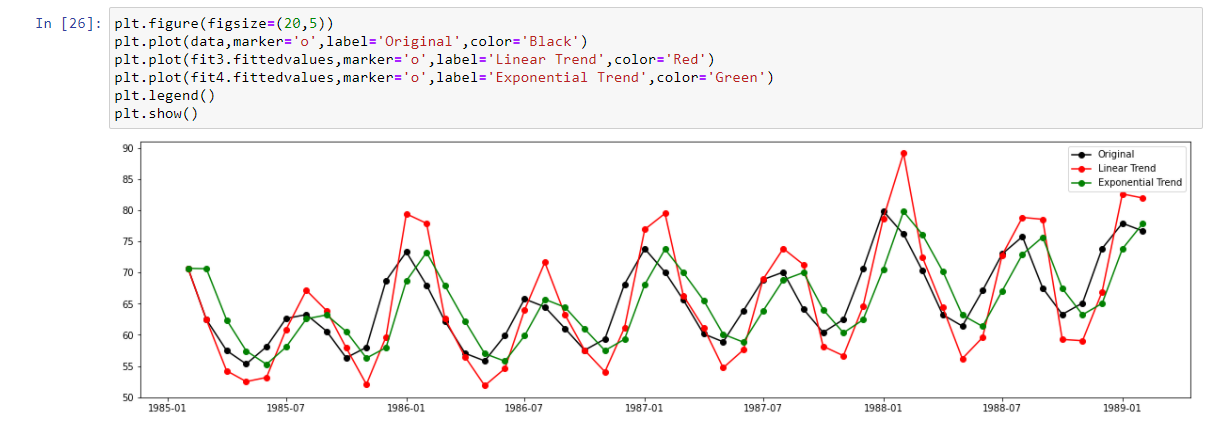
plt.plot(data,marker='o',label='Original',color='Black')

plt.plot(fit3.fittedvalues,marker='o',label='Linear Trend',color='Red')

plt.plot(fit4.fittedvalues,marker='o',label='Exponential Trend',color='Green')

plt.legend()

plt.show()



As per this above visualization here ‘Black’ is original curve. ‘Red’ Linear Trend means No Exponential Trend. And ‘Green’ is Exponential Trend.

# Time Series Decomposition and stationarity Check

We already discussed three types of time series patterns: trend, seasonality and cycles. When we decompose a time series into components, we usually combine the trend and cycle into a single trend-cycle component (sometimes called the trend for simplicity). Thus we think of a time series as comprising three components: a trend-cycle component, a seasonal component, and a remainder component (containing anything else in the time series).

Here we consider some common methods for extracting these components from a time series. Often this is done to help improve understanding of the time series, but it can also be used to improve forecast accuracy.

# E:\prasad\SMIIT\Working Project\Time Series Analysis\blog pic\13.png

If we assume an additive decomposition, then we can write.

yt=St+Tt+Rt

where yt is the data, St is the seasonal component, Tt is the trend-cycle component, and Rt is the remainder component, all at period t. Alternatively, a multiplicative decomposition would be written as

yt=St×Tt×Rt.

The additive decomposition is the most appropriate if the magnitude of the seasonal fluctuations, or the variation around the trend-cycle, does not vary with the level of the time series. When the variation in the seasonal pattern, or the variation around the trend-cycle, appears to be proportional to the level of the time series, then a multiplicative decomposition is more appropriate. Multiplicative decompositions are common with economic time series.

An alternative to using a multiplicative decomposition is to first transform the data until the variation in the series appears to be stable over time, then use an additive decomposition. When a log transformation has been used, this is equivalent to using a multiplicative decomposition becauseyt=St×Tt×Rtis equivalent tologyt=logSt+logTt+logRt.

**Load Dataset**

# Here we Loaded Australian Drug Sales data. CSV file name is TimeSeries.csv

# This dataset we already uploaded on our GitHub account.

# Link: <https://github.com/SMIIT-Projects/Time-Series-Analysis>

# E:\prasad\SMIIT\Working Project\Time Series Analysis\blog pic\14.png

In this dataset having a 204 rows and 2 columns. First column name is ‘Date’ and second one is ‘Value’. As per the Time Series first of all we need to convert ‘Date’ column in to datetime format and once it will convert then set index as a ‘Date’ column. After set index as date then we have only 1 column in our dataset that name is value. Now our shape of data is 204 rows and 1 column. In this dataset our index starts from 01-07-1991 to 01-06-2008.

**Code:**

# Import Important Libraries Seasonal and Seasonal\_decompose

from statsmodels.tsa.seasonal import seasonal\_decompose

from dateutil.parser import parse

# Load Dataset Australian Drug Sales

df=pd.read\_csv('TimeSeries.csv', parse\_dates=['Date'],index\_col='Date')

df.head()

# Check Shape of Data

df.shape

# Reset the index

df.reset\_index(inplace=True)

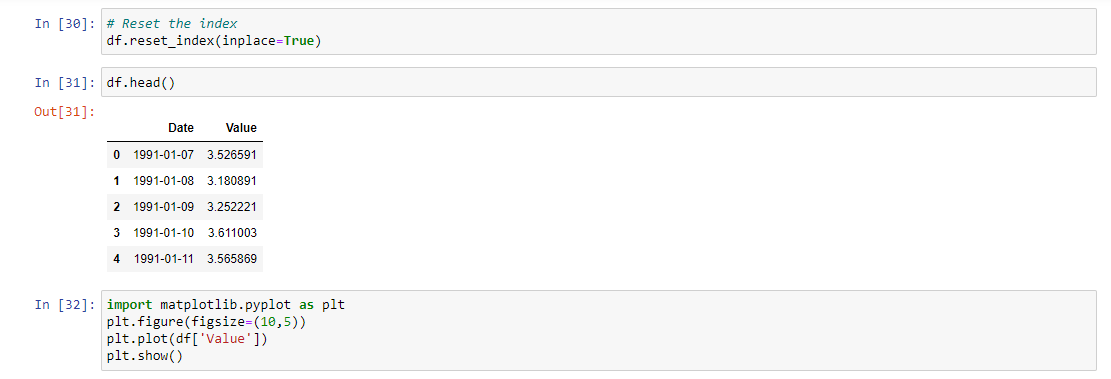
df.head()

import matplotlib.pyplot as plt

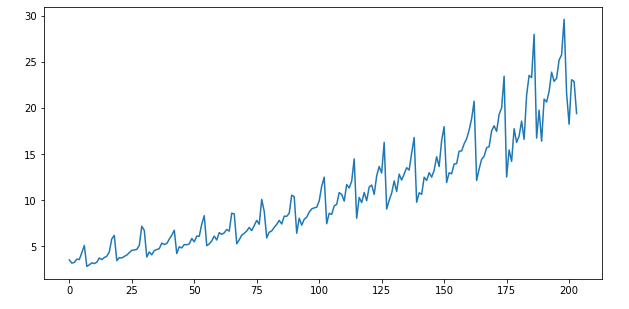
plt.figure(figsize=(10,5))

plt.plot(df['Value'])

plt.show()



For this visualization first column is an index column so we not use as of now. Here we reset the index one more time and only visualize data as df.value.



After visualizing this plot this time series look like this. Here we see Seasonality, Trend and Cyclic behaviour.

Now we will apply Multiplicative Decomposition technique.

**Multiplicative Decomposition**

# Definition of multiplicative time Series:

# Value = Base Level \* Trend \* Seasonality \* Error

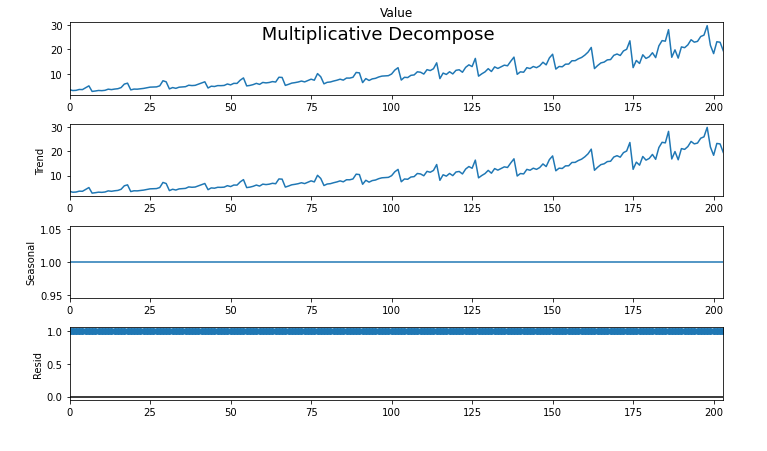
# Code:

# # Multiplicative Decomposition

# mul\_result = seasonal\_decompose(df['Value'], model='multiplicative',period=1) plt.rcParams.update({'figure.figsize': (10,6)})

# mul\_result.plot().suptitle('\n Multiplicative Decompose',fontsize=18)

# plt.show()



As per above visualization 4 chart are there. First chart is Actual curve, second chat is Trend, third is Seasonal and last one is Residual. As you can see time series actual curve look like more or less similar to the trend component. Which means that in this time series trend component is more prominent if we decompose multiplicatively. And Seasonal component is at 1 there is not much of seasonality in this data and Residual is also 1. That’s all of the different parts of the time series when we decompose using Seasonal Decompose.

**Additive Decomposition:**

# Definition of Additive Time Series:

# Value = Base Level + Trend + Seasonality + Error

# Code:

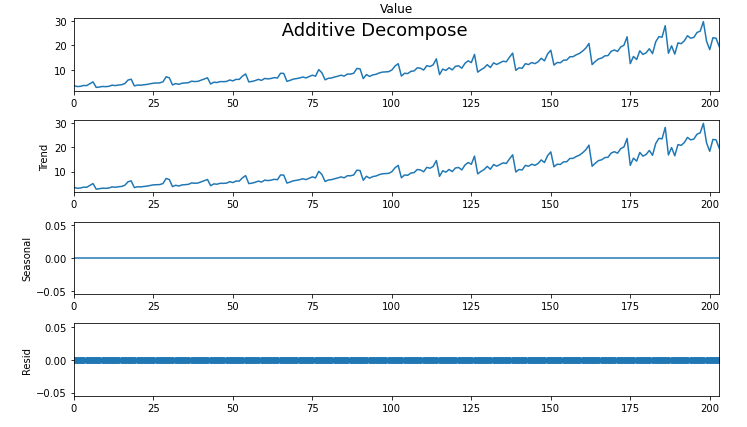
# # Additive Decomposition

# add\_result= seasonal\_decompose(df['Value'], model='additive',period=1)

# plt.rcParams.update({'figure.figsize': (10,6)})

# add\_result.plot().suptitle('\n Additive Decompose',fontsize=18)

# plt.show()



In this visualization also 4 chart are available. First chart is Actual curve, second chat is Trend, third is Seasonal and last one is Residual. As you can see time series actual curve look like more or less similar to the trend component. Which means that in this time series trend component is more prominent if we decompose Additively. And Seasonal component is at 0 there is not much of seasonality in this data and Residual is also 0. That’s all of the different parts of the time series when we decompose using Seasonal Decompose.

**Create Different Data Frame of Additive and Multiplicative:**

**Code:**

#Additive

new\_df\_add = pd.concat([add\_result.seasonal, add\_result.trend, add\_result.resid, add\_result.observed], axis=1)

new\_df\_add.columns = ['seasoanilty', 'trend', 'residual', 'actual\_values']

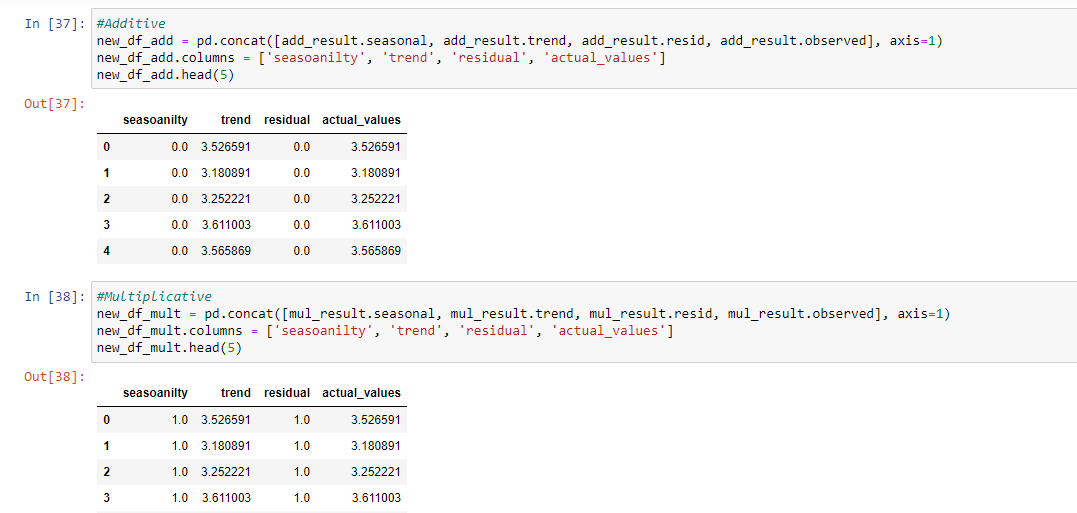
new\_df\_add.head(5)

#Multiplicative

new\_df\_mult = pd.concat([mul\_result.seasonal, mul\_result.trend, mul\_result.resid, mul\_result.observed], axis=1)

new\_df\_mult.columns = ['seasoanilty', 'trend', 'residual', 'actual\_values']

new\_df\_mult.head(5)



As per above both data frame in Additive data frame trend and actual\_values are similar and there is no seasonality and residual are available.

and In multiplicative data frame there is Trend and actual\_values is also similar and there is seasonality is 1 and residual is also 1.

# Stationarity

#### Definition of stationarity - constant mean and variance

# E:\prasad\SMIIT\Working Project\Time Series Analysis\stationary.png

A common assumption in many time series techniques is that the data are stationary. A stationary process has the property that the mean, variance and autocorrelation structure do not change over time. Stationarity can be defined in precise mathematical terms, but for our purpose we mean a flat looking series, without trend, constant variance over time, a constant autocorrelation structure over time and no periodic fluctuations seasonality. For practical purposes, stationarity can usually be determined from a run sequence plot.

# Augmented Dickey Fuller Test (ADF Test)

Augmented Dickey Fuller test (ADF Test) is a common statistical test used to test whether a given Time series is stationary or not. It is one of the most commonly used statistical test when it comes to analysing the stationary of a series.

***ADF Test - null hypothesis - non-stationary - if p-value < 5% reject null hypothesis***

Since testing the stationarity of a time series is a frequently performed activity in autoregressive models, the ADF test along with KPSS test is something that you need to be fluent in when performing time series analysis.

Another point to remember is the ADF test is fundamentally a statistical significance test. That means, there is a hypothesis testing involved with a null and alternate hypothesis and as a result a test statistic is computed and p-values get reported.

It is from the test statistic and the p-value, you can make an inference as to whether a given series is stationary or not.

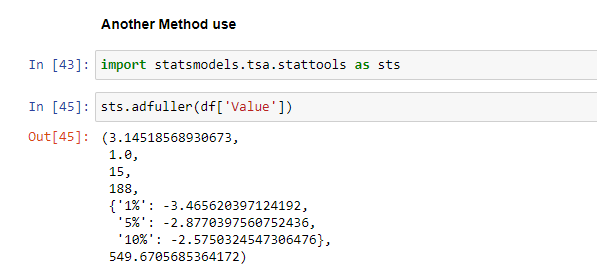


In ADF test when null hypothesis is rejected data will be Stationary. And null hypothesis is accepted data will be non-stationary.

As per ADF test if p-value < 5% that time null hypothesis will rejected.

If p-value is > 5% null hypothesis will have accepted.

As per above visualization p-value is 1.0 that’s way this data is non stationary.



GitHub Link: <https://github.com/SMIIT-Projects/Time-Series-Analysis>

## **Tools and Technologies:**

The Code is written in Python 3.8.5.

## **Used libraries:**

numpy==1.20.1

pandas==1.2.3

matplotlib==3.3.4

seaborn==0.11.1

statsmodels==0.12.2

Thank You!