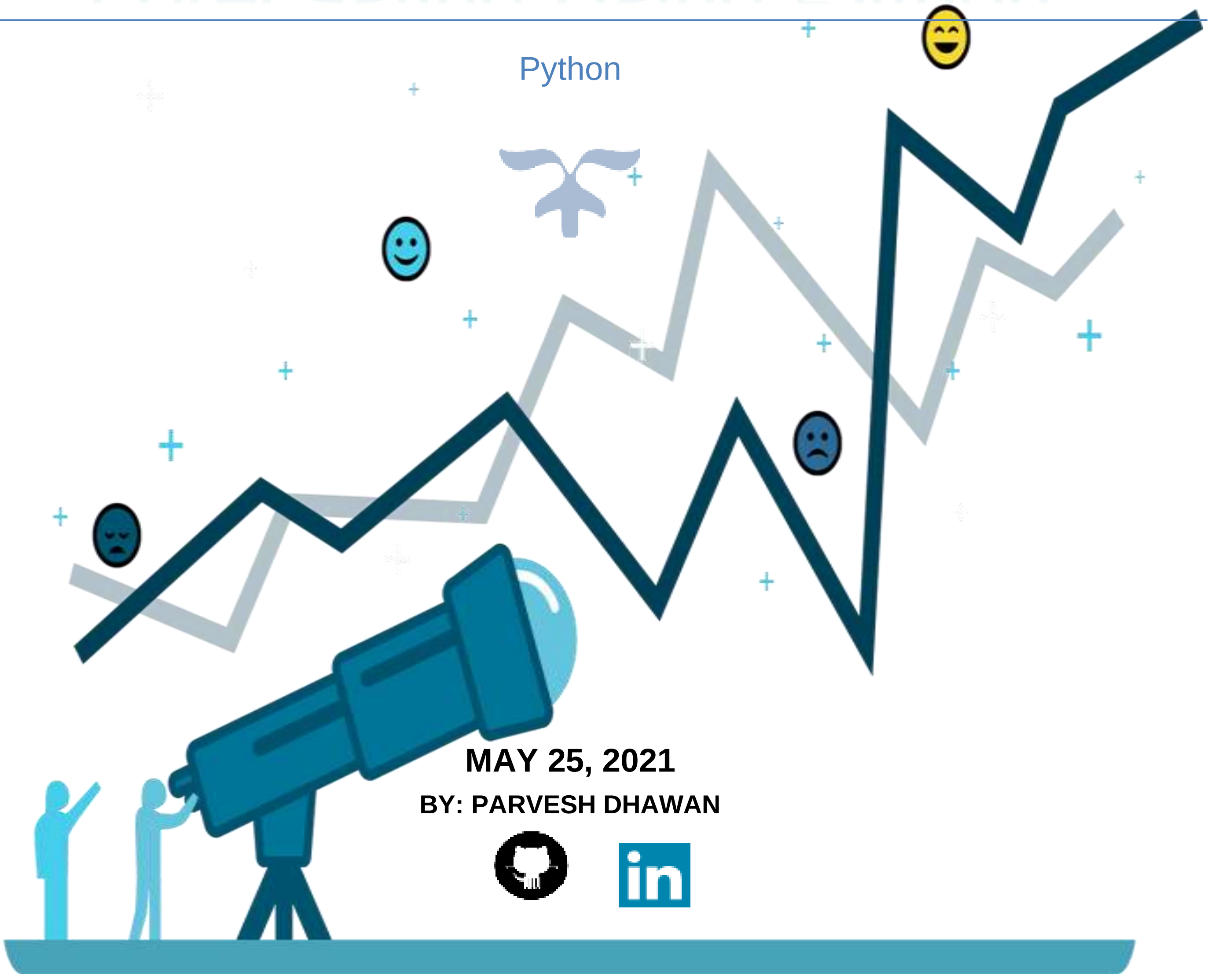
Time Series Analysis & Forecasting Using Python

Python



MAY 25, 2021

BY: PARVESH DHAWAN

Contents

Time Series

Time Series is simply a series of data points that has a time component involved in it. Every data point present in the dataset is associated directly with the date and time component.

Forecasting:

Forecasting is the process of making predictions of the future based on the past and present data and most commonly by analysis of trends.

Two types of Forecasting:

Quantitative Forecasting	**Qualitative Forecasting**
●Based on past data which used numerical features. ●Data Driven and Lesser bias. ●Time series analysis and statistics.	●Done when we do not have past data to analyze. ●It uses the Delphi method which involves subject matter experts.

Regression Analysis: To find patterns in data and using those patterns to predict dependent variable

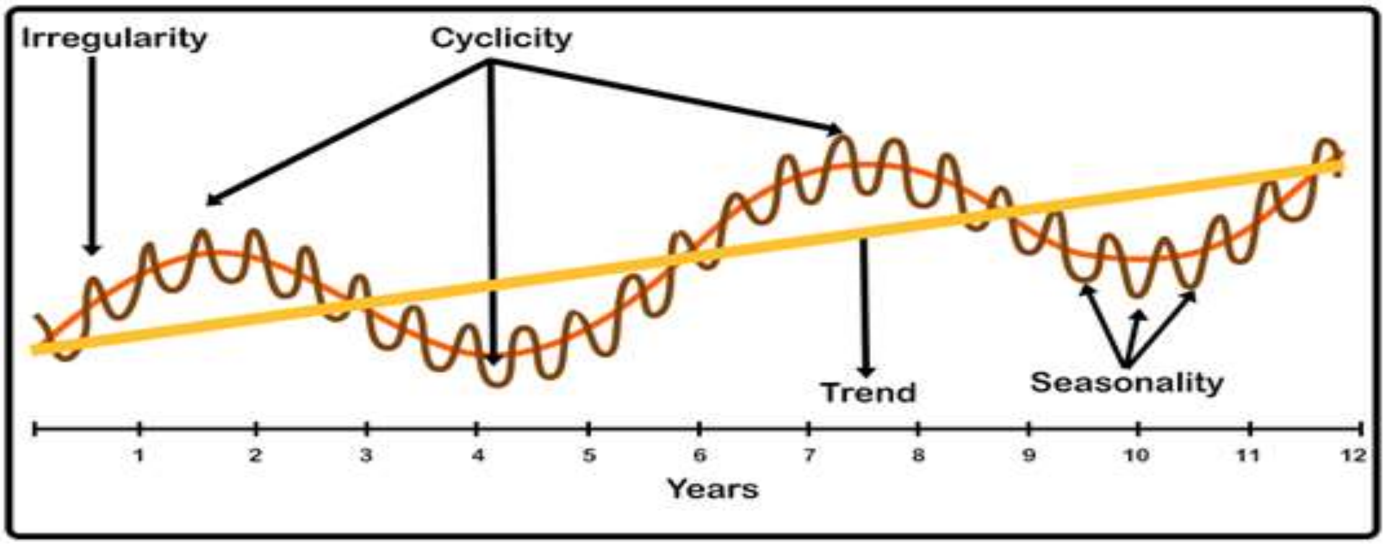
Time Series Analysis: To identify trends in data and use trends to forecast future events.

Example: Suppose you want to predict the price of a product based on some of its properties.

● The **Regression technique** would find the pattern in the data and would predict the price of the new product based on the patterns present in the training data.

● While using **Time series analysis** we forecast the future sales demand, temperature, number of customers/passengers etc.

Component of Time Series



Level:

Level is the baseline for the entire time series. It is the average of the time series and the baseline to which we add different other components.

Trend:

The Trend is the indication of whether the time series has moved higher or lower over the time period.

Seasonality:

Seasonality is the pattern in time series which repeats after a fixed interval of time.

Cyclicity:

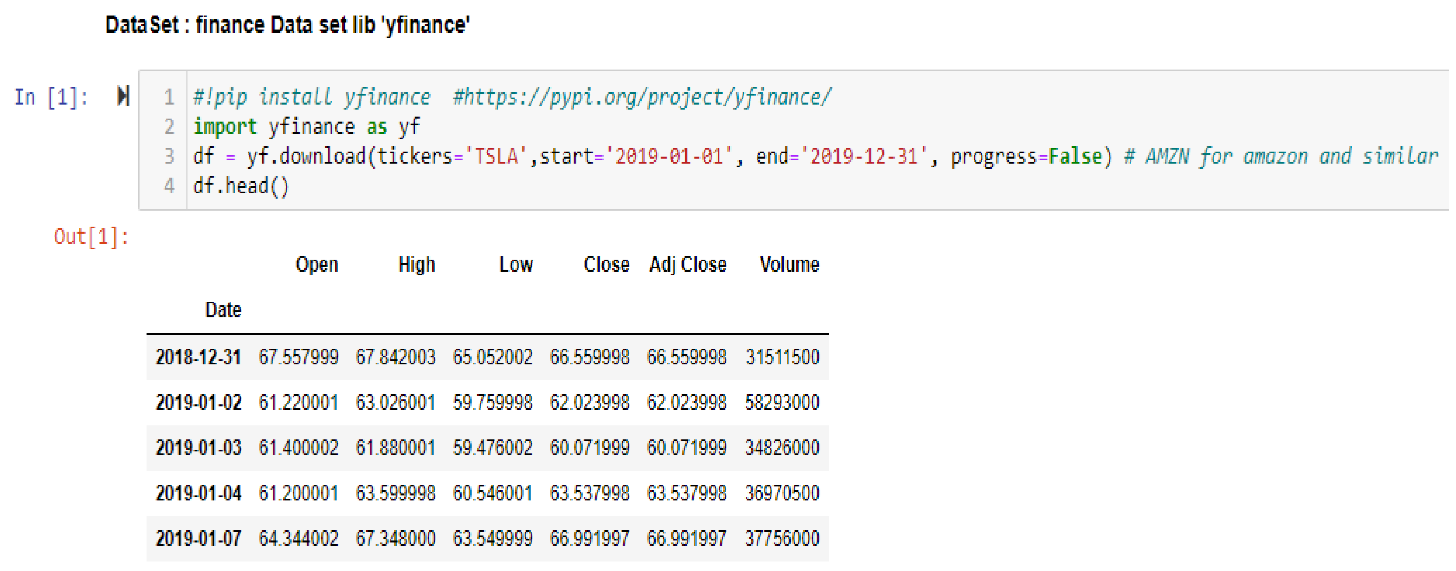
Cyclicity is the pattern in the time series which repeats itself after some interval of time but the interval of time is not fixed in the case of cyclicity unlike the seasonality.

Noise:

Noise is the random variation in the time series. We cannot use noise to forecast the future. Noise is just random fluctuations in our data and does not have any Pattern.

Popular Data Set Sources:

- ❖ Kaggle
- ❖ Google Trend
- ❖ ***yfinance***
  - ➢ A python library to get the historical financial data.



Data Set : finance Data set lib 'yfinance'

In [1]:
```python
#!pip install yfinance  #https://pypi.org/project/yfinance/
import yfinance as yf
df = yf.download(tickers='TSLA',start='2019-01-01', end='2019-12-31', progress=False) # AMZN for amazon and similar
df.head()
```

Out[1]:

	Open	High	Low	Close	Adj Close	Volume
Date						
2018-12-31	67.557999	67.842003	65.052002	66.559998	66.559998	31511500
2019-01-02	61.220001	63.026001	59.759998	62.023998	62.023998	58293000
2019-01-03	61.400002	61.880001	59.476002	60.071999	60.071999	34826000
2019-01-04	61.200001	63.599998	60.546001	63.537998	63.537998	36970500
2019-01-07	64.344002	67.348000	63.549999	66.991997	66.991997	37756000

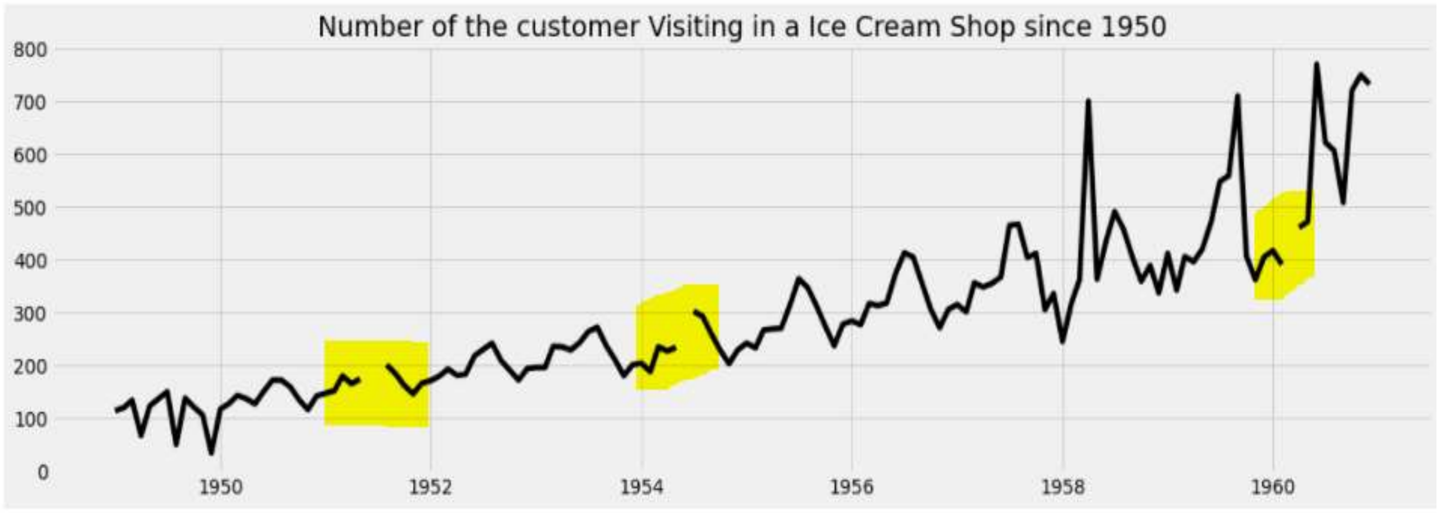
Handling Missing Value

Time series is a form of moving and sequential data so in general missing value imputation as Mean Model or Median won't be a good Choice in missing value imputation.

Let's load a sample data & Check the missing value using visualization technique:

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')

import warnings
warnings.filterwarnings('ignore')

## Reading dataset

data = pd.read_csv('dataset/Customers_in_a_Shop.csv',header=None)
data.columns = ['Date','Customers']
data['Date'] = pd.to_datetime(data['Date'],format="%Y-%m")
data = data.set_index('Date')
print(data.head())

# Checking Missing Values by Visualizing data as we have just 144 datapoints
plt.rcParams['figure.figsize'] = (17,5)
plt.plot(data,color='black')
plt.title("Number of the customer Visiting in a Ice Cream Shop since 1950")
plt.show()
```

```
            Customers
Date
1949-01-01      114.0
1949-02-01      120.0
1949-03-01      134.0
1949-04-01       67.0
1949-05-01      123.0
```



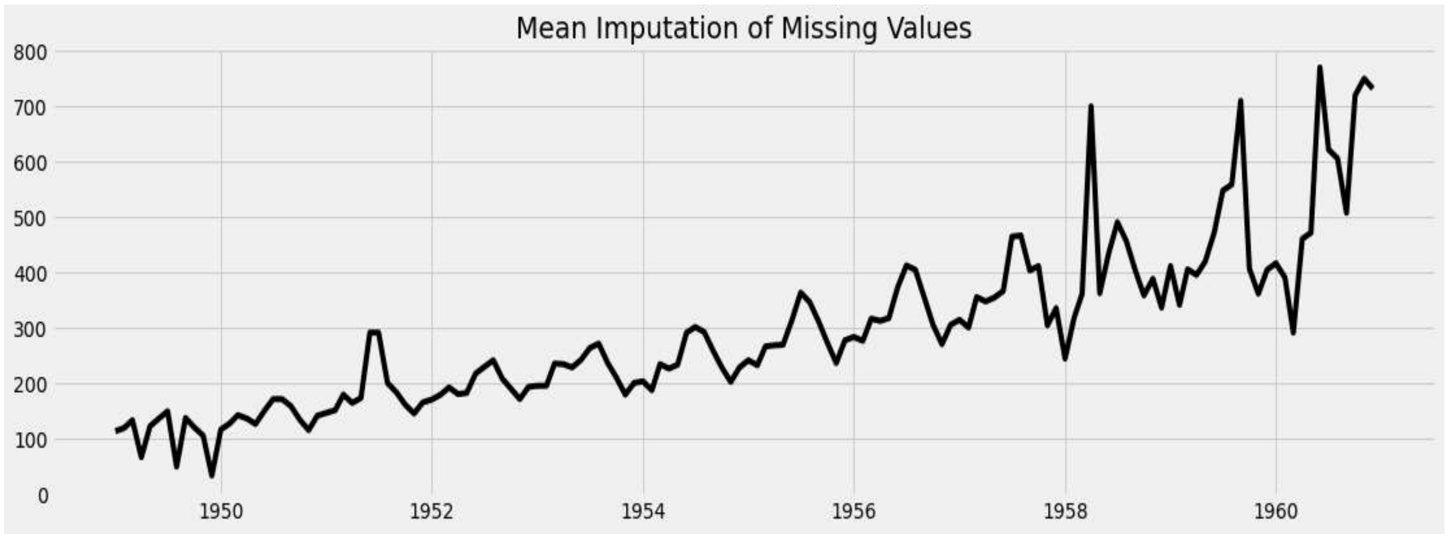
In General, there are 4 different ways of dealing with missing value in time series¶

1. Central Tendency (Mean, Median, Mode)
2. LOCF (Last Observation Carried Forward)
3. Linear Interpolation
4. Seasonal Interpolation Method

Mean Imputation

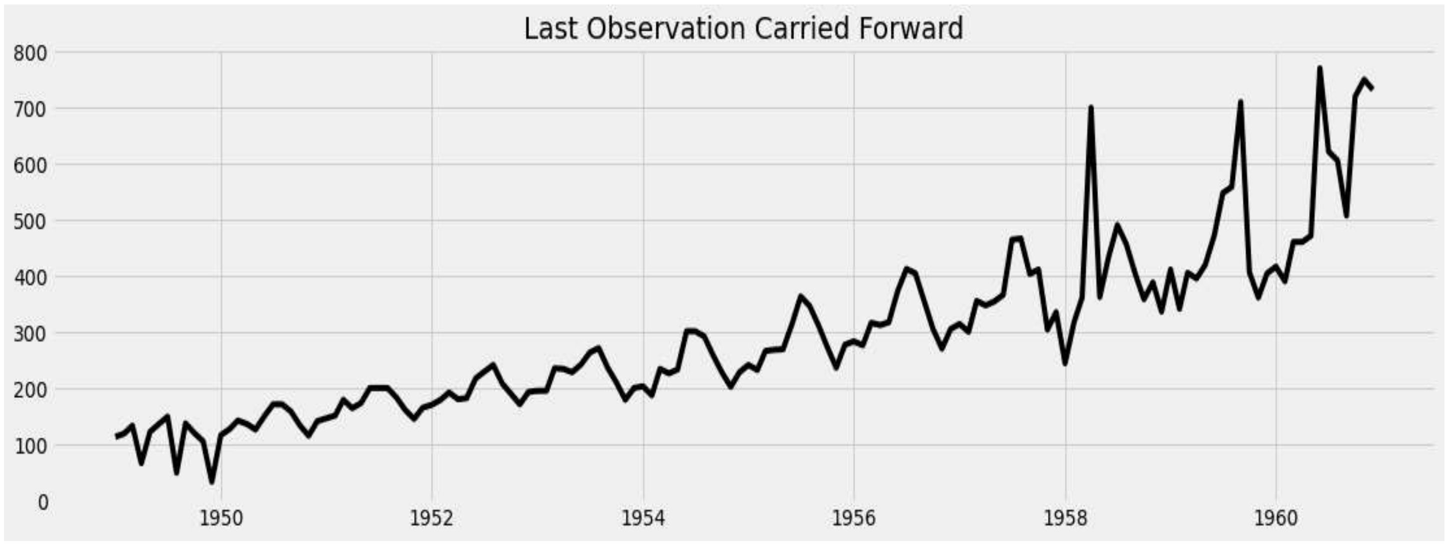
Find the mean and replace the missing value with mean.

Note: Imputation of every missing value with the mean of the data can distort the seasonality of the data and also it won't take into the consideration the nature of the data



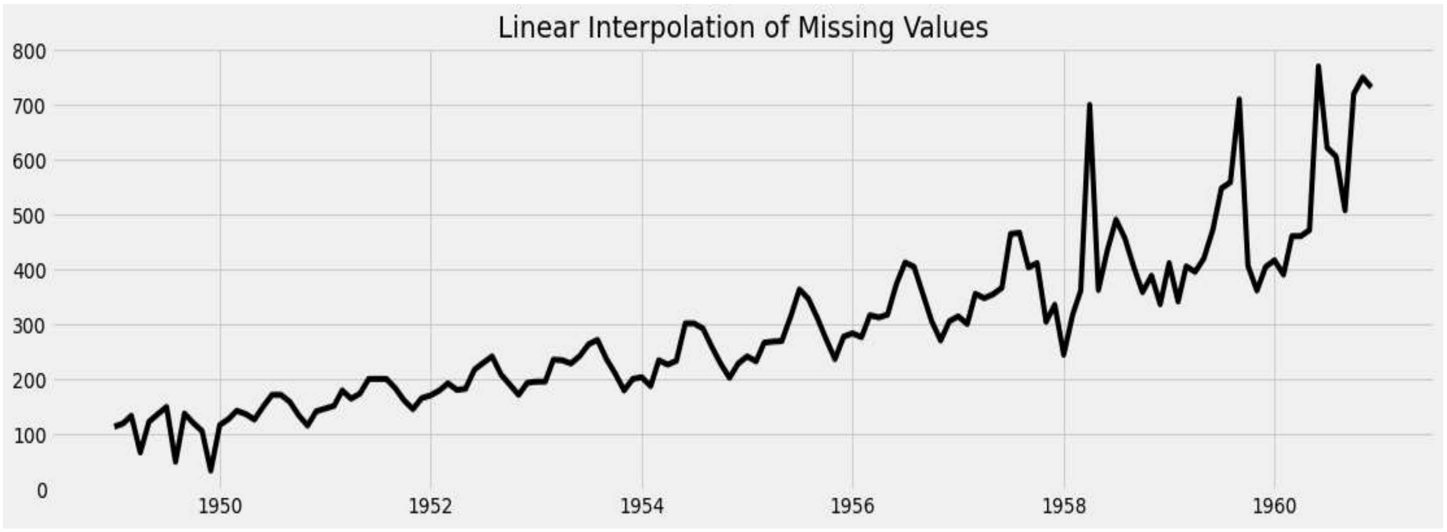
LOCF - Last Observation Carried Forward

In this method we Impute missing value with the last observation available
bfill() - Function is used to implement this.



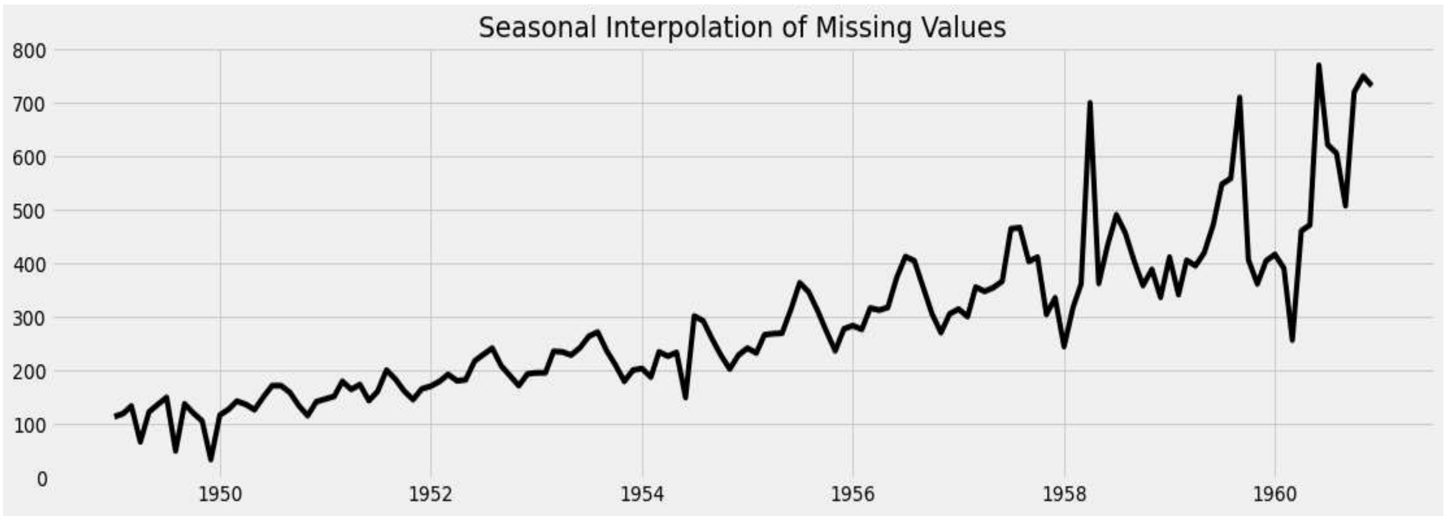
Linear Interpolation

We draw a straight line joining the previous and the next point of the missing values.



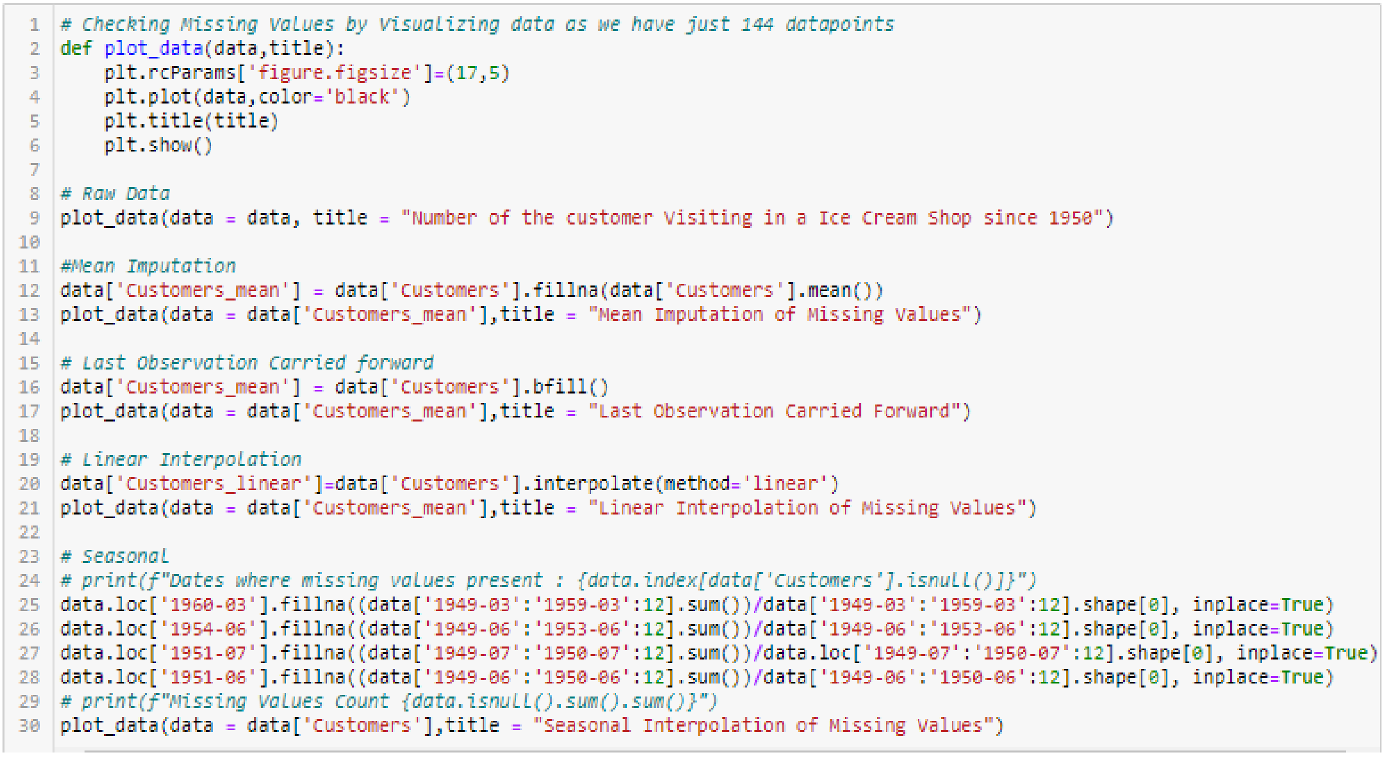
Seasonal Interpolation

We impute the missing values with the average of corresponding data points from the previous seasonal data and next seasonal data.



Based on all the imputation we can see that Linear Interpolation methods work better than rest of the methods.

Python Code:



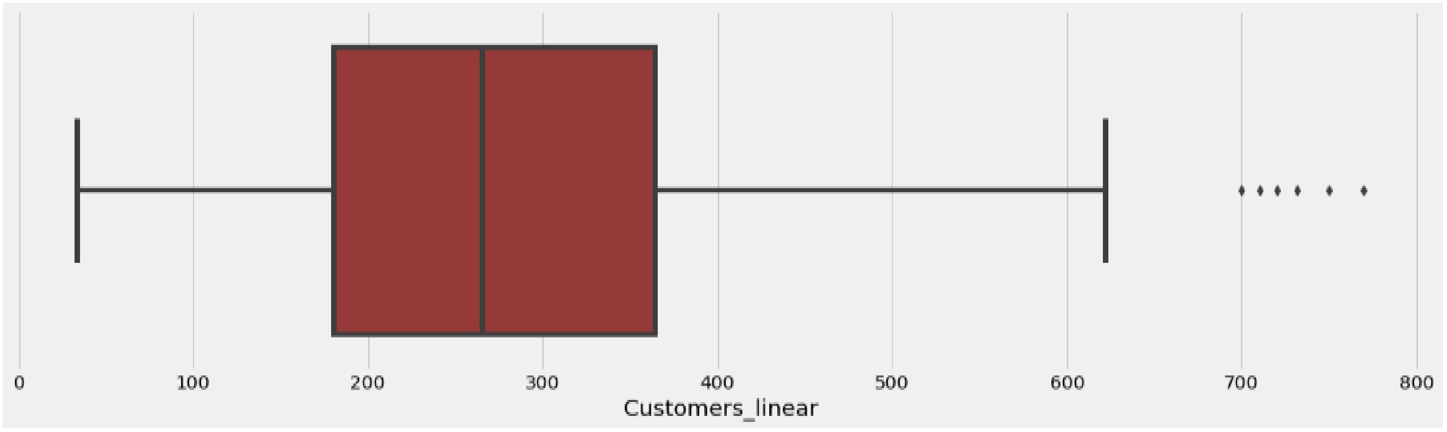
```
# Checking Missing Values by Visualizing data as we have just 144 datapoints
def plot_data(data,title):
    plt.rcParams['figure.figsize']=(17,5)
    plt.plot(data,color='black')
    plt.title(title)
    plt.show()

# Raw Data
plot_data(data = data, title = "Number of the customer Visiting in a Ice Cream Shop since 1950")

#Mean Imputation
data['Customers_mean'] = data['Customers'].fillna(data['Customers'].mean())
plot_data(data = data['Customers_mean'],title = "Mean Imputation of Missing Values")

# Last Observation Carried forward
data['Customers_mean'] = data['Customers'].bfill()
plot_data(data = data['Customers_mean'],title = "Last Observation Carried Forward")

# Linear Interpolation
data['Customers_linear']=data['Customers'].interpolate(method='linear')
plot_data(data = data['Customers_mean'],title = "Linear Interpolation of Missing Values")

# Seasonal
# print(f"Dates where missing values present : {data.index[data['Customers'].isnull()]}")
data.loc['1960-03'].fillna((data['1949-03':'1959-03':12].sum())/data['1949-03':'1959-03':12].shape[0], inplace=True)
data.loc['1954-06'].fillna((data['1949-06':'1953-06':12].sum())/data['1949-06':'1953-06':12].shape[0], inplace=True)
data.loc['1951-07'].fillna((data['1949-07':'1950-07':12].sum())/data.loc['1949-07':'1950-07':12].shape[0], inplace=True)
data.loc['1951-06'].fillna((data['1949-06':'1950-06':12].sum())/data['1949-06':'1950-06':12].shape[0], inplace=True)
# print(f"Missing Values Count {data.isnull().sum().sum()}")
plot_data(data = data['Customers'],title = "Seasonal Interpolation of Missing Values")
```

Handling Outliers

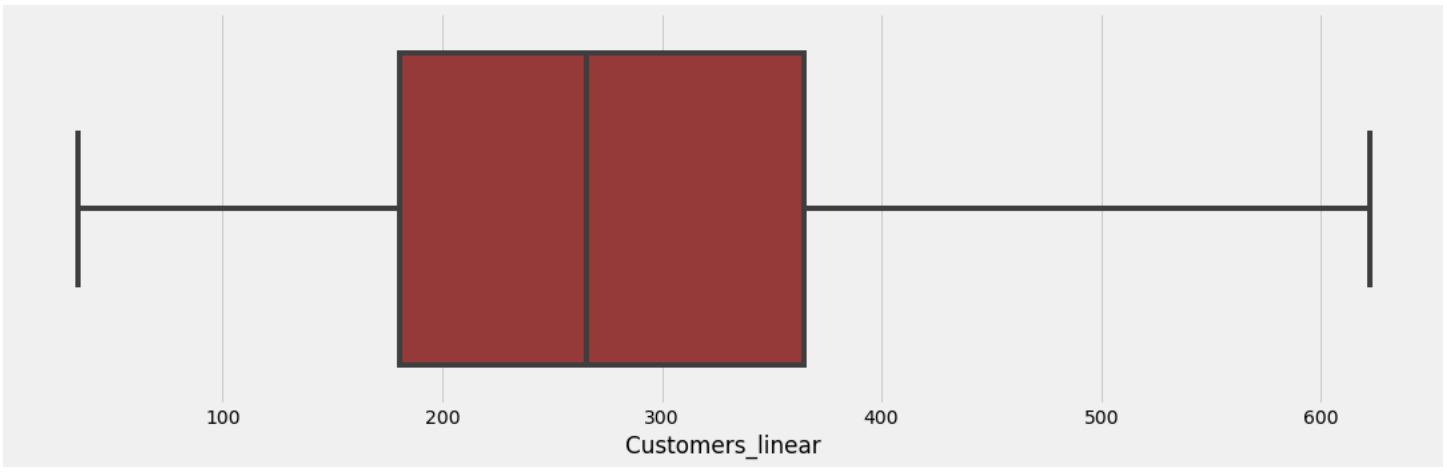
Outliers are extreme values that fall a long way outside of the other observations. If we feed a data set having outliers, then our predictive model will get highly confused and will start producing biased results.

1. Detect Outliers -- Most common method Is **Box Plot**
2. Deal with outliers
    a. Capping,
    b. other method of normalizing

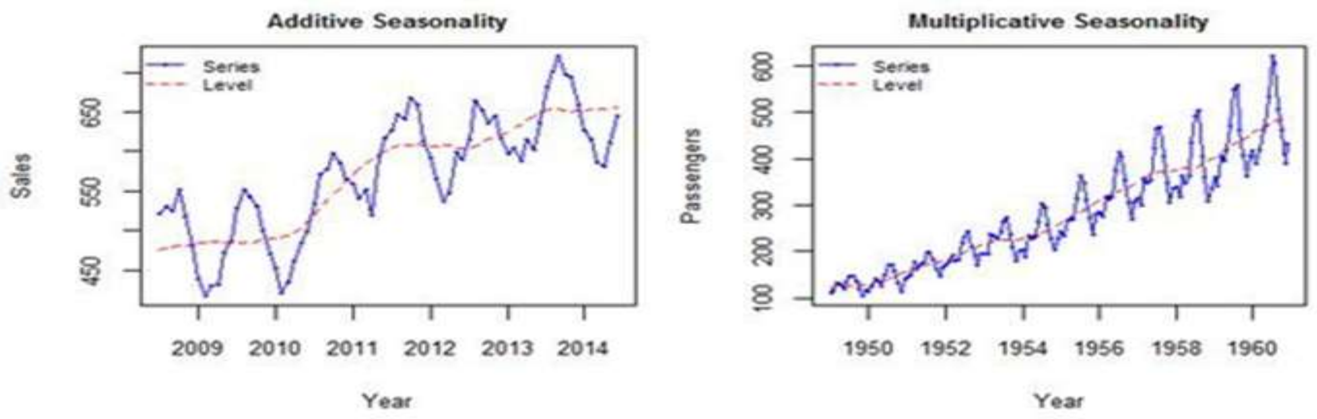
Box Plot: It is a standardized way of displaying the distribution of data. Any data points which are less than Q1-1.5IQR or greater than Q3+1.5IQR are considered to be outliers.



Capping the values greater than or equal to 700 as they are far away from the distribution of data.



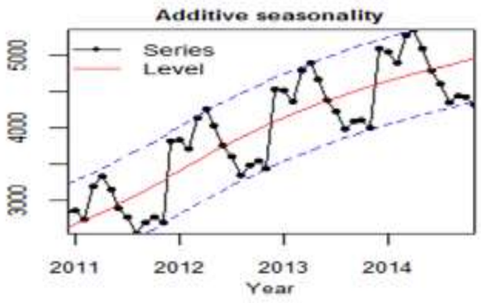
Time Series Decomposition



Why do we need to decompose the time series data?
To find the underlying pattern like trend, Seasonality, Cyclicity, Noise we need to decompose the Time series data. As it also improves our understanding about the time series data.

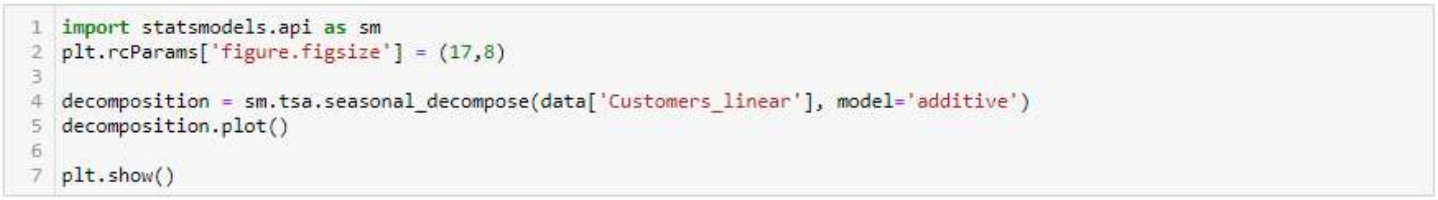
If we notice the previous line plot of our data there, we can see that seasonal components are growing as the level of time is increasing & for this, we prefer Multiplicative Seasonal Decomposition.
Let’s have a look at both of the decomposition type in details:

Additive Seasonal Decomposition

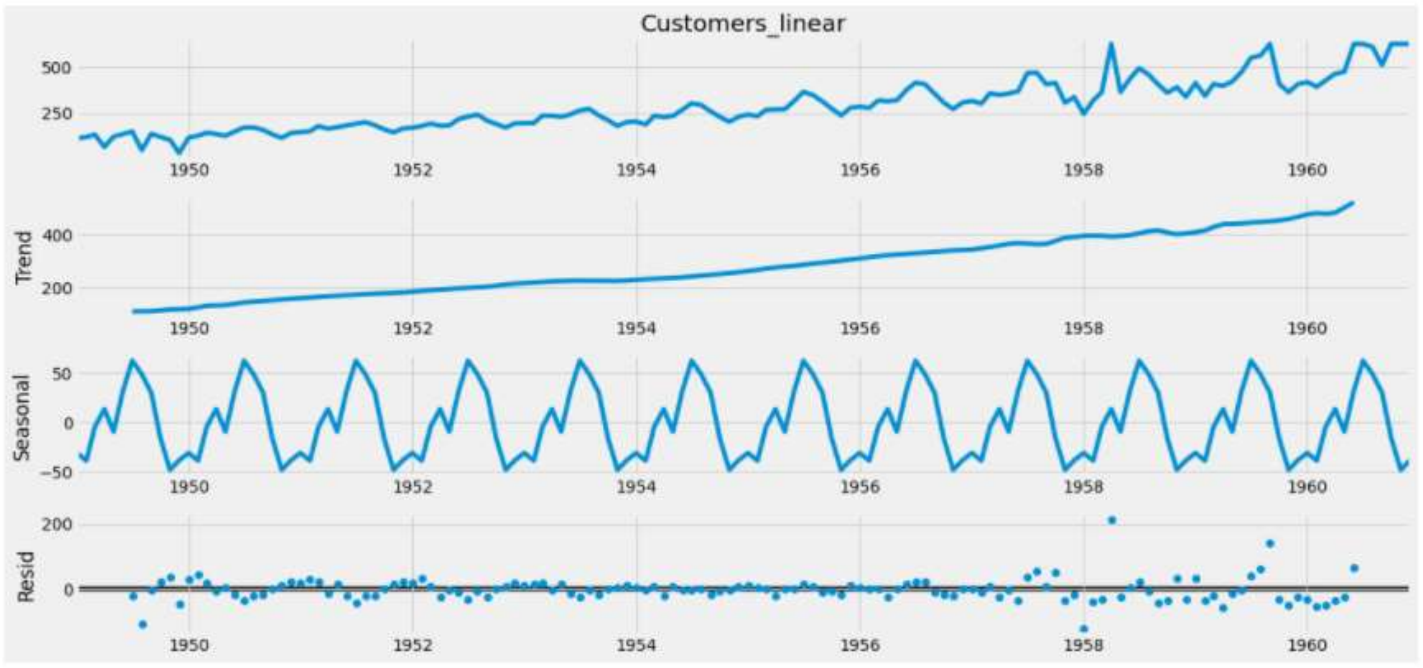


Additive Seasonal decomposition is when we add the individual components to get the time series data.
y(t) = Level + Trend + Seasonality + Noise

We use an additive model; the magnitude of seasonality does not change in relation to time.



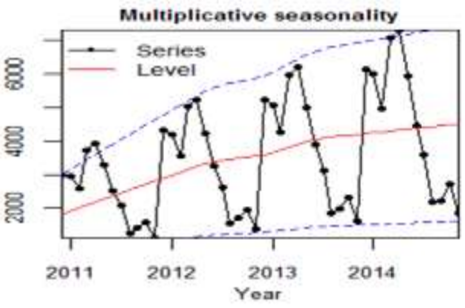
```python
import statsmodels.api as sm
plt.rcParams['figure.figsize'] = (17,8)

decomposition = sm.tsa.seasonal_decompose(data['Customers_linear'], model='additive')
decomposition.plot()

plt.show()
```



Below observation notice in our dataset:

- There is an increasing trend in our time series
- For seasonality there is a fixed pattern
- In residual there seems to have a pattern as well from 1943 to 1953 and after 1958

Multiplicative Seasonal Decomposition



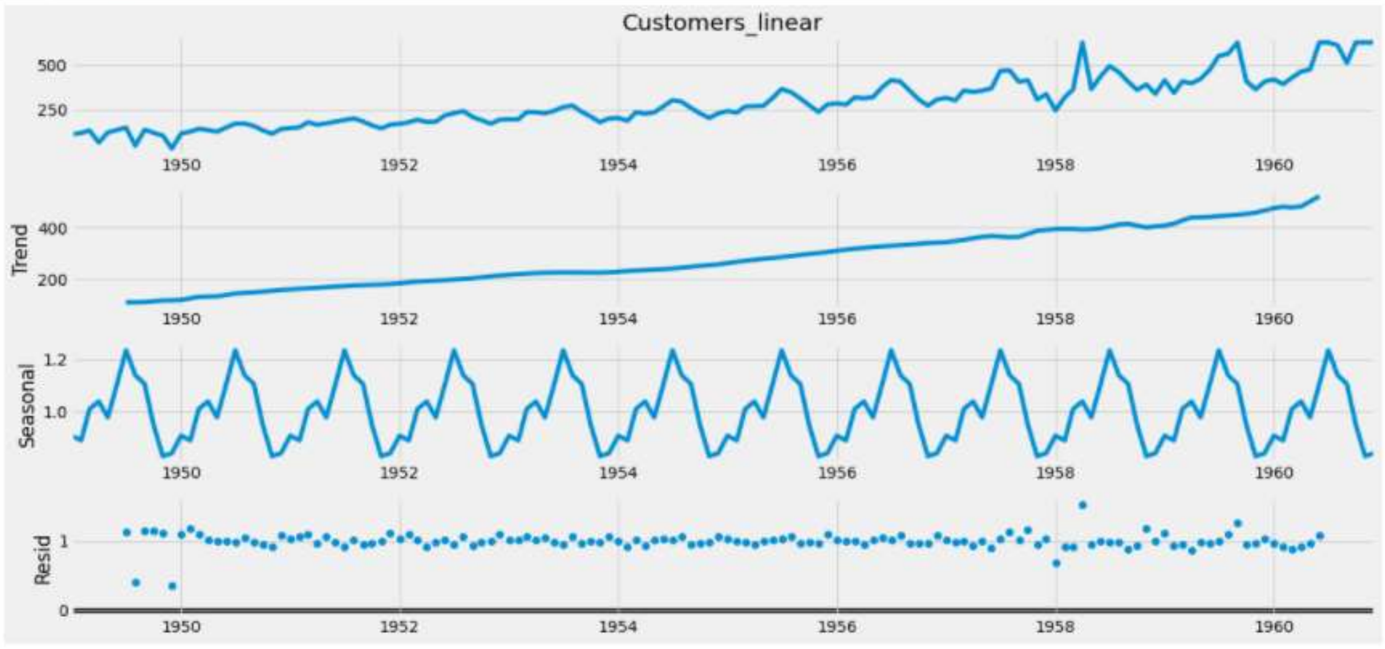
Multiplicative Seasonal decomposition is when we multiply the individual components to get the time series data.

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

On the other hand, we use multiplicative models when the magnitude of the seasonal pattern in the data depends on the magnitude of the data.

Multiplicative Seasonal decomposition

```python
plt.rcParams['figure.figsize'] = (17,8)

decomposition = sm.tsa.seasonal_decompose(data['Customers_linear'], model='multiplicative')
fig = decomposition.plot()
plt.show()
```



Below observation notice in our dataset:

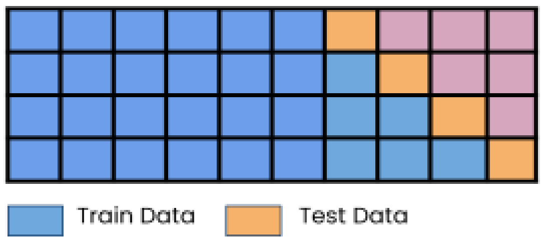
- The trends show an increasing nature of time series
- Seasonality captures overall pattern of the time series data
- There is no pattern in the residual this time, which means everything is captured by the trend and seasonality component of decomposition.

Finally, we can conclude that whenever the magnitude of seasonal pattern of the data is not directly proportional to the series data, we perform Additive Seasonal Decomposition. & In opposite to it we perform Multiplicative Seasonal Decomposition.

Splitting Time Series Data

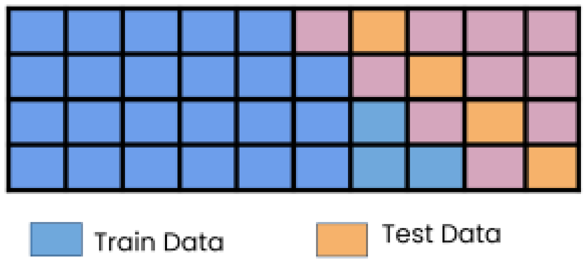
In most of the machine learning projects we perform the 80,30 splits as train & test split. But in time series data which is related to dates and time we cannot future data to test out the past data and due to that we cannot use the random splitting of data points. In time series test data should be the latest data produced and train data will be the past data available. We have two different types of cross validation techniques which we use for splitting of time series dataset.

One-Step Validation



In one-step validation, the test data is exactly after the train data.

Multi-Step Validation

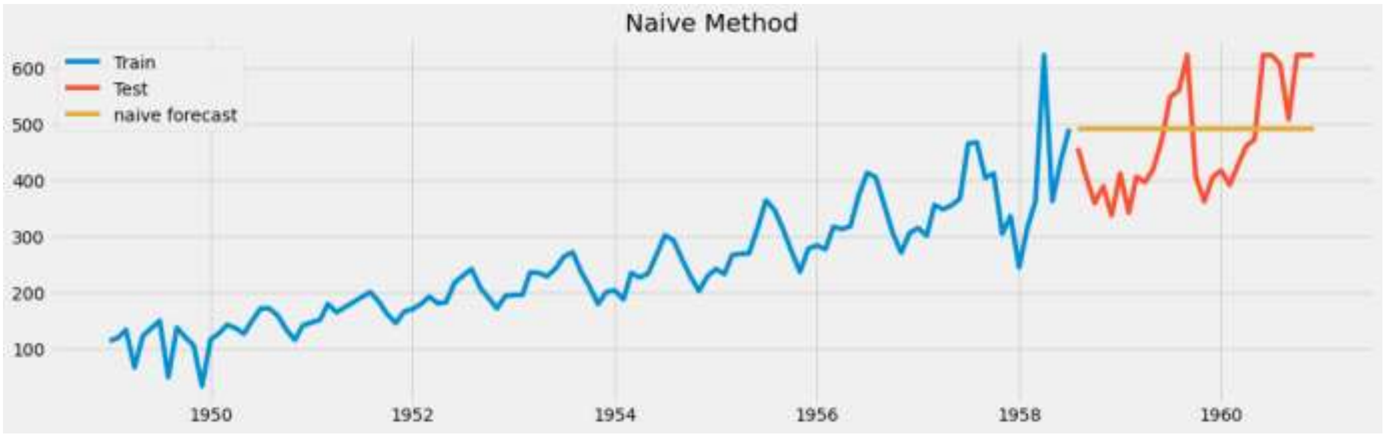


Multi-step validation is the same as one-step validation with the difference that in multi-step we do not consider the exact next point after train data points.

Smoothing Techniques in Time Series Data (for basic forecasting)

Naive Forecasting

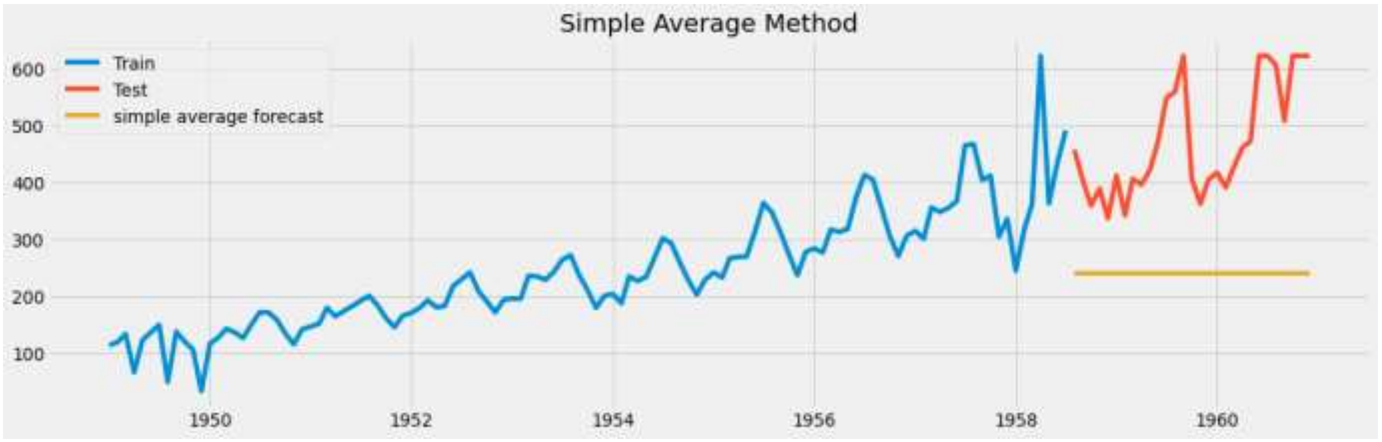
It looks at the last historical data and extrapolates it for all the future values without adjusting or attempting to establish causal factors. (In simple term it forecast the value which is the **latest one** in the series)



A few times this naive method works well in case of stock price prediction as some fluctuation appears to be random in this case. So, every time the fluctuations do not follow systematic patterns.

Simple Average Forecasting Method

In this method, we take the future predictions equal to the average of all the historical data

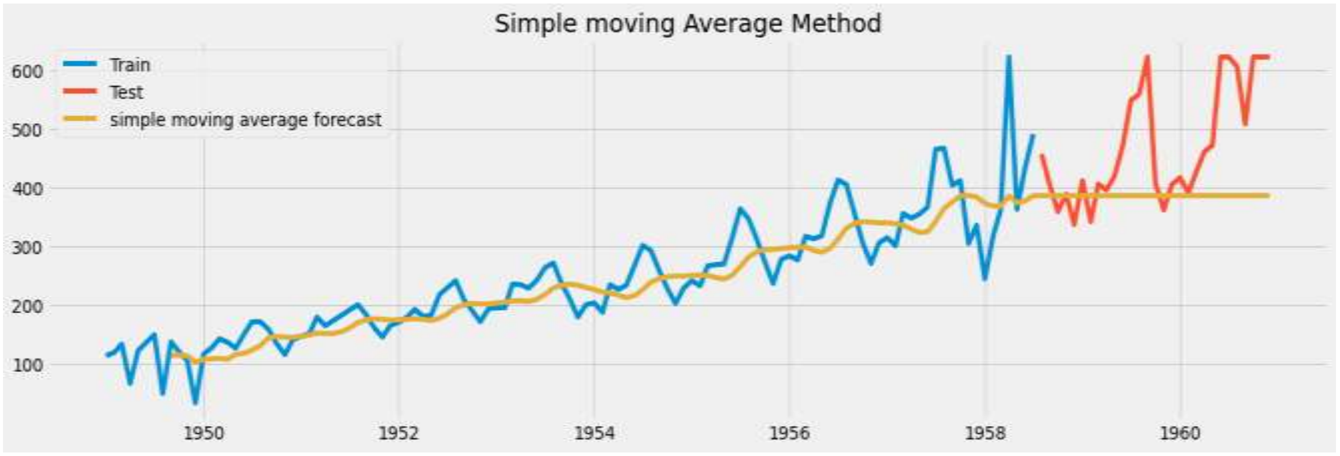


We can observe that this method of forecast doesn't capture any trend or seasonality.

Simple Moving Averages

To overcome the problem, we have faced in the simple moving average we can consider only the last few observations to forecast the value as their average. As the last few observations have more impact in the future rather than the first observation

Future predictions = Average of moving window
A window could be a time period of 3months or 6 or more depending on the data we have.
Window = 9; data[column].rolling(window).mean()



As we can see here, this method has captured the trend as well as the seasonality. If we reduce the windows size it may capture more seasonality from the data

Simple Exponential Smoothing

In simple Moving average we consider the last few values to forecast the future value and each of the past data points contributes equally towards the forecasting new value, **But intuitively shouldn't the most recent observation influence more than any other observation.** This is the main idea behind the working of Simple Exponential Smoothing.
Simple Exponential Smoothing is performed using the following formula

$$y(t+1) = \frac{a*y(1)+b*y(2)+\cdots+l*y(t)}{(a+b+c+\cdots l)}$$

Where a, b, c are the weights

a<b<c…. <l, l is the greatest of all.

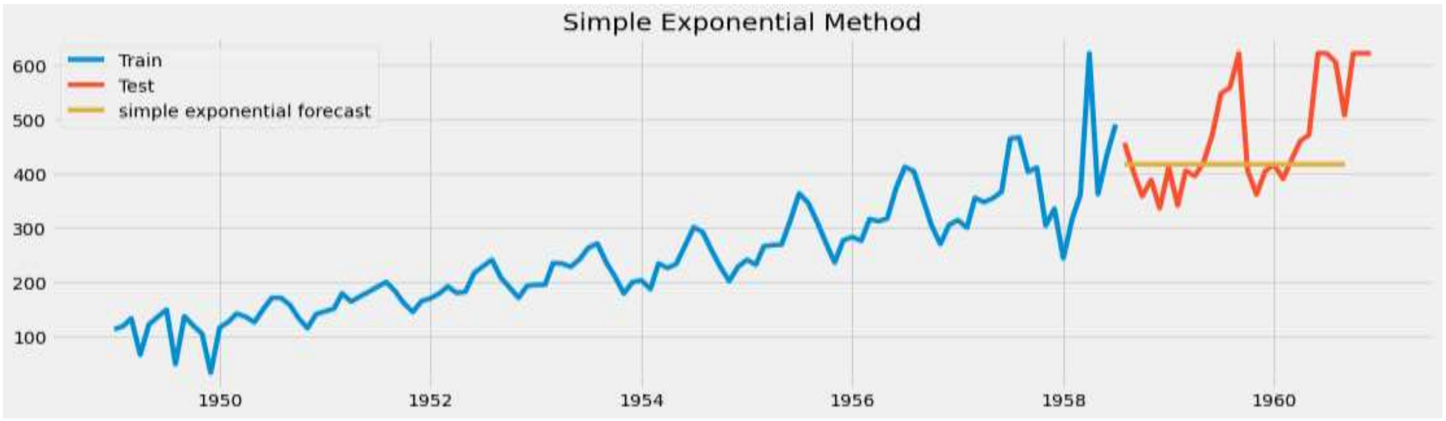
Simple moving exponential smoothing technique captures the level of the time series data.

Level at the point t is:

L(t) = α*y(t) + (1- α) *y(t-1)

y(t) denotes the recent observation

L(t-1) denotes the previous observation and **0< α< 1**



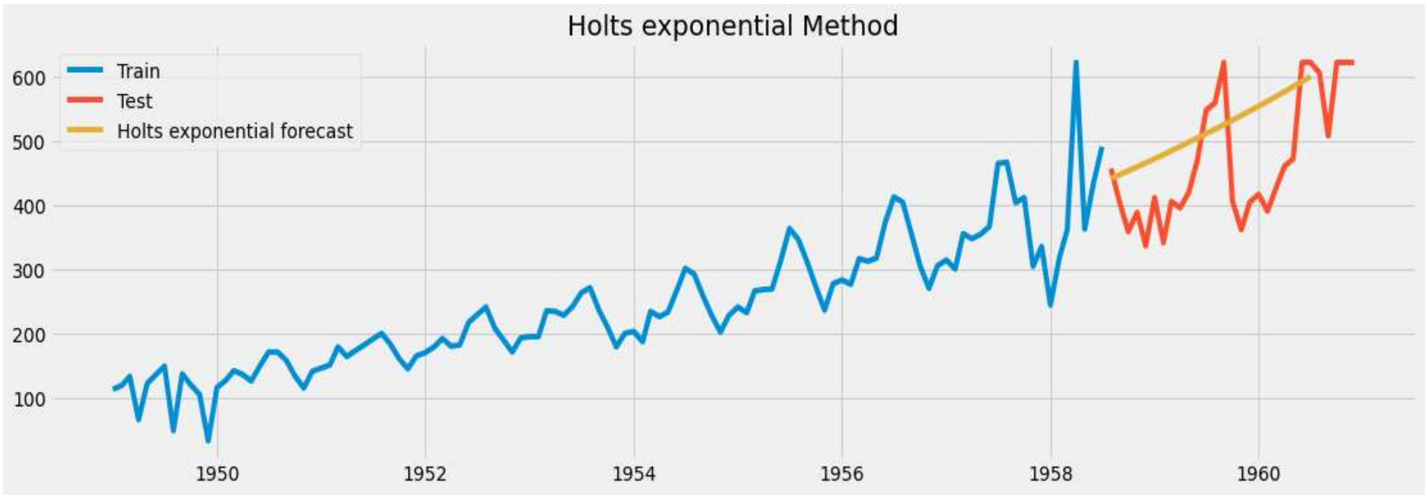
We can see here that results are getting better that the simple moving average method. As this has captured the level of the data.

Holt Exponential Smoothing

Holt's exponential smoothing **captures the level and trend** of time series in the forecast.
The Forecast Equation is: **y(t+1) = l(t) +b(t)**
where **l(t) is the level** component and **b(t) is the trend** component.

The trend component is calculated as: **b(t) = β(l(t) - l(t-1)) + (1-β) b(t-1)**

Where, β is the smoothing parameter for trend.

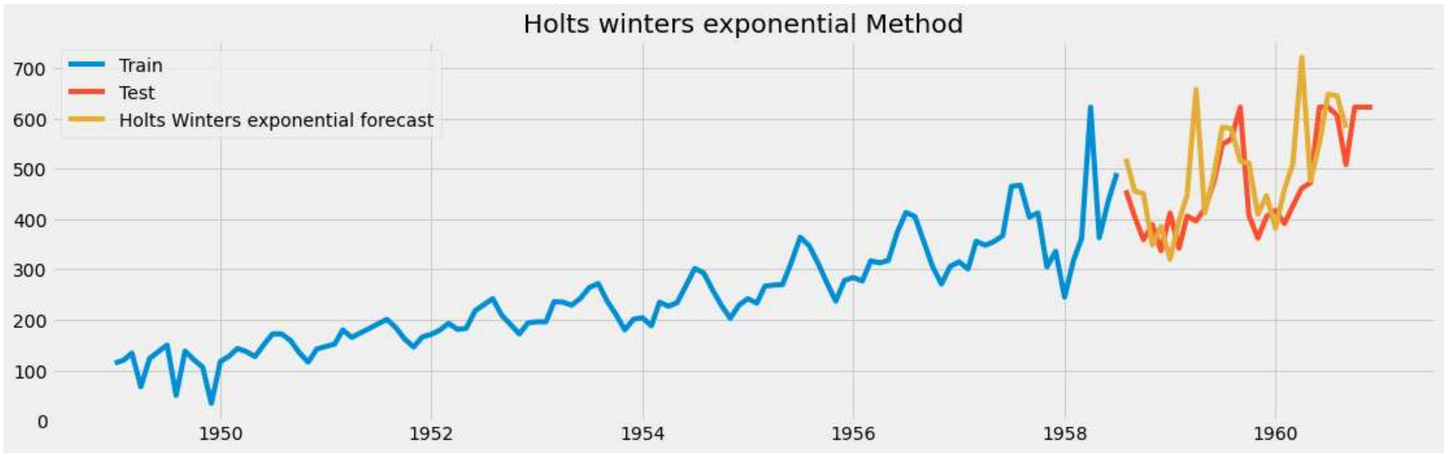


Holt Winter Exponential Smoothing

Holt's Winter Exponential Smoothing **captures** all **level, trend and seasonality.**

The forecast equation is: **y(t+1) = l(t)+b(t)+s(t+1-m)**

Where m is the number of times a season repeats in a time period.



This method has captured all the three components of a time series data that are level, trends and seasonality.

Evaluation Metrics for Time Series Data

Mean Forecast Error (MFE)

- Mean Forecast Error shows deviation of the forecast from the actual demand.
- This is the mean of the sum of all the differences between the actual values and forecasted values.
- *Let's consider 10 data points*
    1. -2, -4, 3, 8, -3, -8, 1, -1, 5, -5
    2. **Sum of errors = 0**
- According to this metrics error is 0 but we can see that there is a huge deviation in b/w prediction and actual values.

Mean Absolute Error (MAE)

Mean Absolute Error takes the absolute values of the differences between actual values and forecasted values.

$$\text{Mean absolute error} = \frac{\text{Sum of all absolute errors}}{\text{No of errors } (n)}$$

$$= \sum_{i=1}^{n} \frac{X_i - X}{n}$$

The main problem about MAE is that it doesn't provide any clear information about the error.
The error value calculated does not have anything to compare with.

Mean Absolute Percentage Error (MAPE)

$$MAPE = \frac{100\%}{n} \sum \left| \frac{y - \widehat{y}}{y} \right|$$

Mean Absolute Percentage Error calculates the percentage of mean absolute error to get a clear idea of how much the forecasted values deviates from the actual values.

Mean Square Error (MSE)

$$MSE = \frac{1}{n} \Sigma \underbrace{\left(y - \widehat{y} \right)^2}_{\text{The square of the difference between actual and predicted}}$$

In order to capture the deviations, we square the differences, sum them up and take the average.

Root Mean Square Error (RMSE)

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y})^2}$$

Root Mean Square Error is equal to the root of the Mean Square error in order to bring error value to the same dimension.

Forecast Models

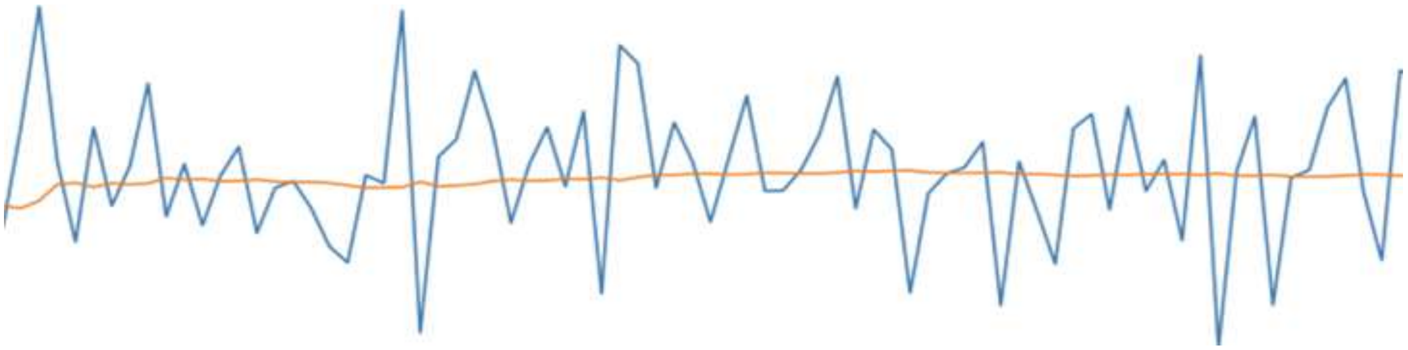
Auto Regressive Models (AR)

- It predicts future behavior based on past behavior. It is mainly used when there is some correlation between values in a time series and the values that precede and succeed them.
- In Autoregression we use a Regression model to formulate a time series problem. We make linear combinations of past observations and use them to forecast future observations.
- **Assumptions:**
  i. **Stationary (Must be)**
  ii. **Autocorrelation (Must have)**
- We can apply AR models only onto those time series data which are stationary in nature. And if the time series is not stationary then we have to convert a non-stationary into stationary using a few statistical tests.

Stationarity

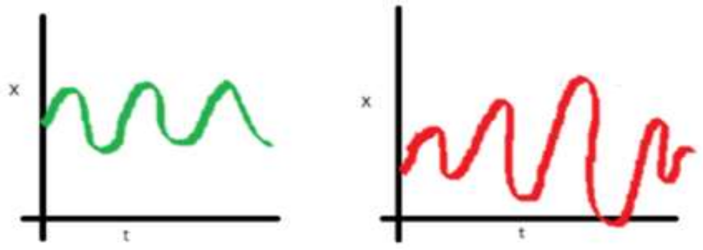
It means that statistical properties of a process generating a time series do not change over time.
Statistical properties generally considered in this case are **Mean, Variance, Covariance**. These all should remain the same irrespective of the time at which we observe.

Time Series with constant mean



Here, the Blue line is the time series and the Orange line is represented as Mean.

Time Series with Constant Variance and Covariance (Fluctuations)



●The **Green** Line, Represents a Chart with **Constant Variance.**

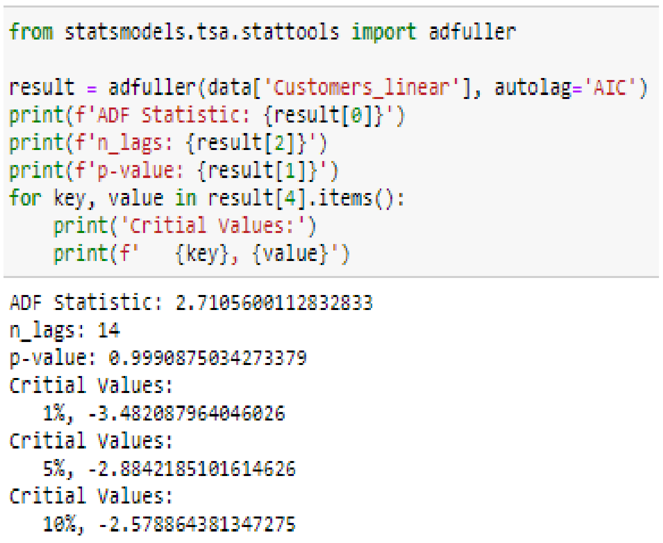
●The **Red** Line Represents a Chart with **non-constant variance.**

By looking both the above plot we can say that this time series stationary in nature

Stationarity Statistical Test

Augmented Dickey-Fuller (ADF) test

Null Hypothesis: The time series is not stationary
Alternate Hypothesis: The time series is stationary.

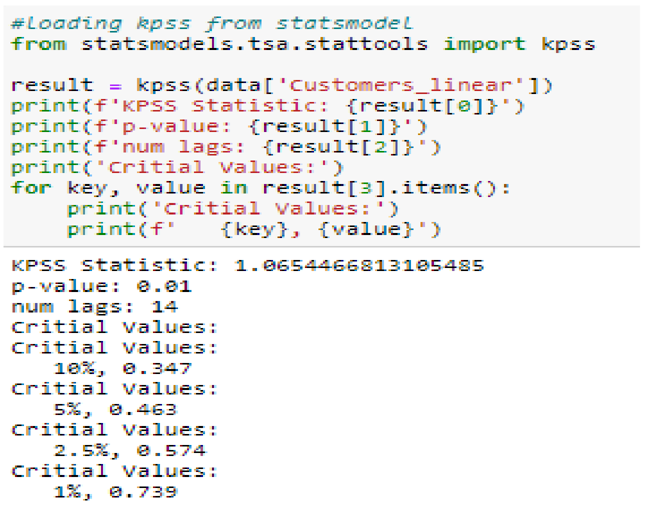


```
from statsmodels.tsa.stattools import adfuller

result = adfuller(data['Customers_linear'], autolag='AIC')
print(f'ADF Statistic: {result[0]}')
print(f'n_lags: {result[2]}')
print(f'p-value: {result[1]}')
for key, value in result[4].items():
    print('Critial Values:')
    print(f'   {key}, {value}')
```

```
ADF Statistic: 2.7105600112832833
n_lags: 14
p-value: 0.9990875034273379
Critial Values:
   1%, -3.482087964046026
Critial Values:
   5%, -2.8842185101614626
Critial Values:
   10%, -2.578864381347275
```

Kwiatkowski-Phillips-Schmidt-Shin (KPSS) test

Null Hypothesis: The time series is stationary.
Alternate Hypothesis: The time series is not stationary.



```
#Loading kpss from statsmodel
from statsmodels.tsa.stattools import kpss

result = kpss(data['Customers_linear'])
print(f'KPSS Statistic: {result[0]}')
print(f'p-value: {result[1]}')
print(f'num lags: {result[2]}')
print('Critial Values:')
for key, value in result[3].items():
    print('Critial Values:')
    print(f'   {key}, {value}')
```

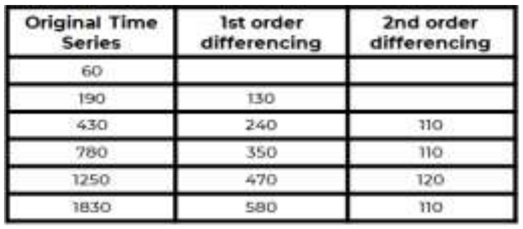
```
KPSS Statistic: 1.0654466813105485
p-value: 0.01
num lags: 14
Critial Values:
Critial Values:
   10%, 0.347
Critial Values:
   5%, 0.463
Critial Values:
   2.5%, 0.574
Critial Values:
   1%, 0.739
```

As from above both the test it is concluded that the series is not Stationary. Let's make the series stationary then.

Converting Non-Stationarity to Stationary

As the series is not stationary, Let see how we convert a non-stationary series into stationary.

Differencing (Mean Constant)



Original Time Series	1st order differencing	2nd order differencing
60		
190	130	
430	240	110
780	350	110
1250	470	120
1830	580	110

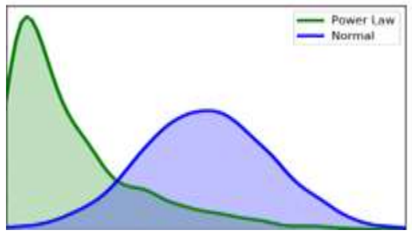
It is a technique to make a mean constant for a time series, that means differencing removes trends from a time series. It means to calculate the difference b/w two consecutive observations.
As here in the table we can see that the series follows an increasing trend, so we perform n-level differencing until it removes trends from the series.

Box Cox Transformation (Variance Constant)

There are many but most commonly used - **Box Cox Transformation**

$$y_i^{(\lambda)} = \begin{cases} \dfrac{y_i^{\lambda} - 1}{\lambda} & \text{if } \lambda \neq 0, \\ \ln(y_i) & \text{if } \lambda = 0, \end{cases}$$

Box Cox transformation transforms non-normal dependent variables into normal distribution.



y is the original time series. (Green)

y(λ) is the transformed series. (Normal)

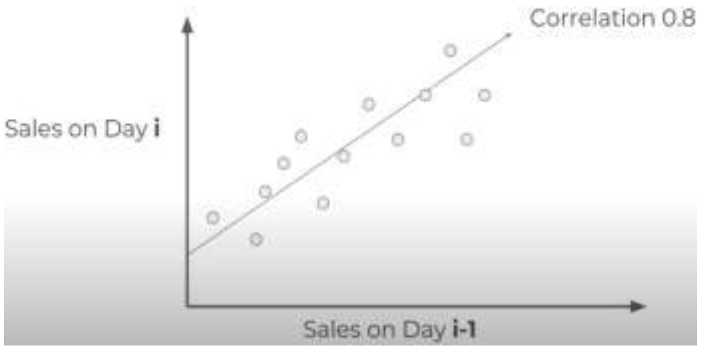
To find the optimal value of lambda bet -5 and +5 is to minimize the variance of the time series.

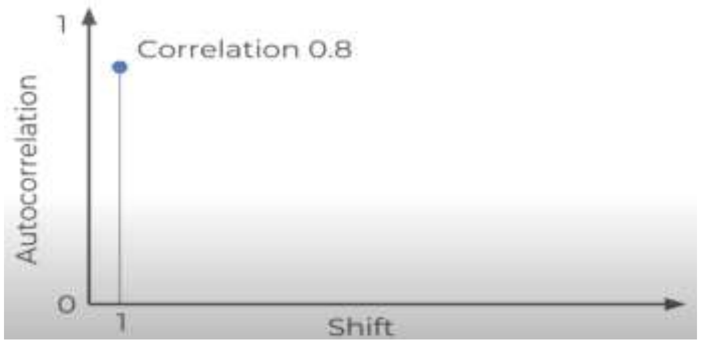
Autocorrelation Function (ACF)

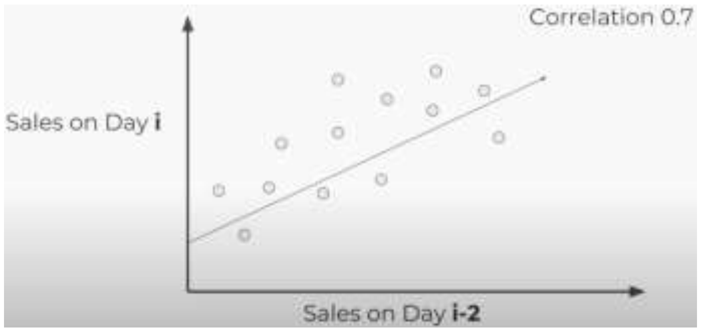
In simple terms, it describes how well the present value of a series is related with its past values. A time series can have components like trend, seasonality, cyclic and residual. ACF considers all these components while finding correlations. An autocorrelation plot shows the correlation of the series with itself, lagged by x time units. So, the y axis in correlation and the x axis is the number of time units of lag. Imagine we had some sales data. We can compare the standard sales data against the sales data shifted by 1 time step. This answers the question:

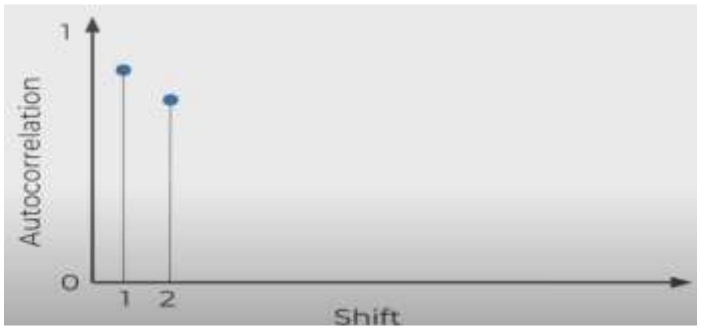
"How correlated are today's sales to yesterday's sales?"

Code: ***from statsmodels.graphics.tsaplots import plot_acf***

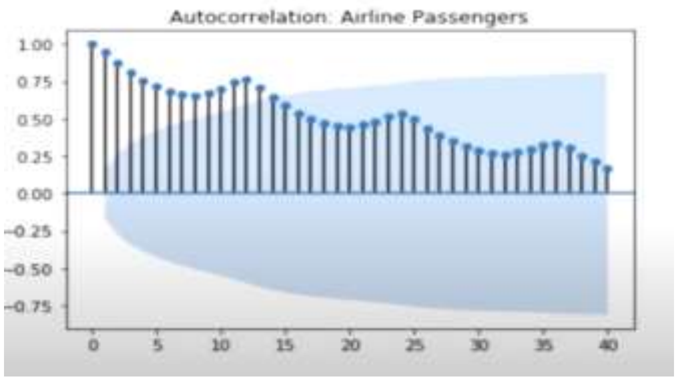






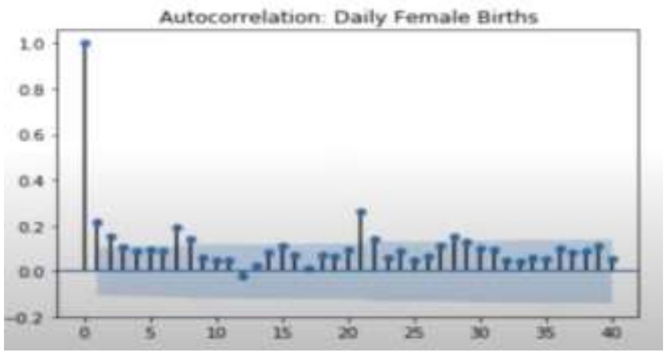


So, on the lagged as sales on day will be decreasing and we will find the corresponding autocorrelation values.
Some typical examples:



Gradual Decline:

This is very common for seasonal data. If you notice there are peaks after a certain number of shifts and increase in the autocorrelation. And this high autocorrelation is after every 12 months, which is a seasonality effect like a year.
Also, as you shift more and more you will be less likely to be correlated.



Sharp Drop-Off

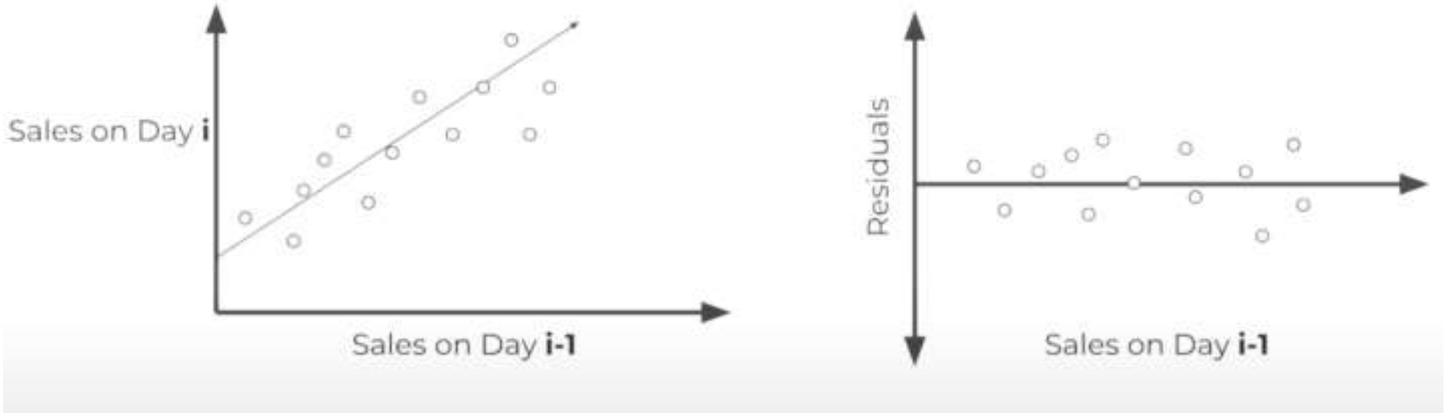
Maybe someday you have high correlation and looking for 2 or 3 days maybe you don't have a high correlation

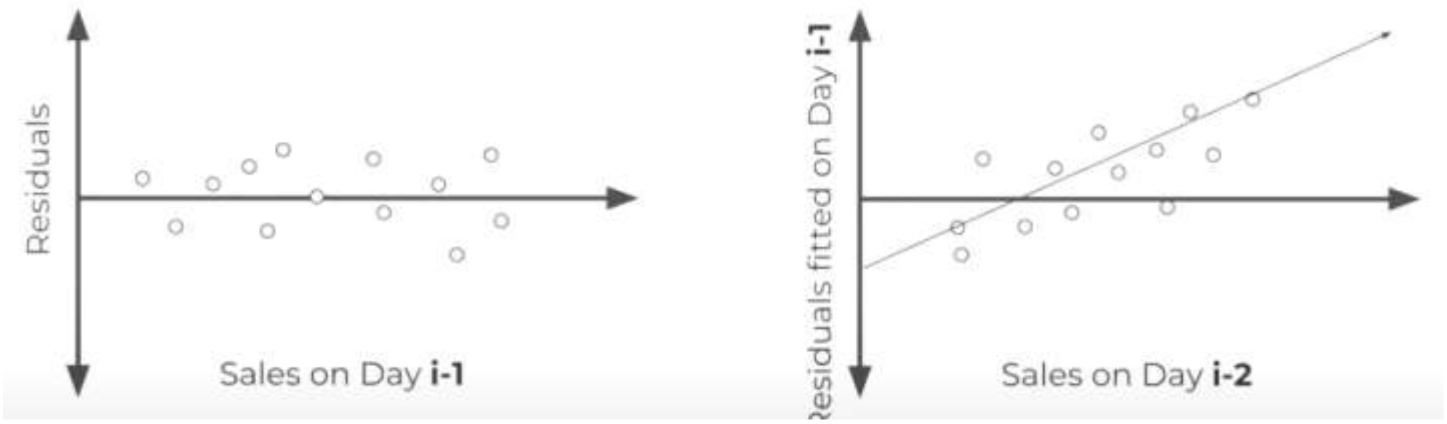
It makes sense that in general there is a decline of some sort, the further away you get with the shift, the less likely the time series would be correlated with itself.

Partial Autocorrelation Function (PACF)

Basically, instead of finding correlations of present with lags like ACF, it finds correlation of the residuals with the next lag value. So, if there is any hidden information in the residual which can be modeled by the next lag, we might get a good correlation and we will keep that next lag as a feature while modeling. Remember while modeling we don't want to keep too many features which are correlated as that can create multicollinearity issues. Hence, we need to retain only the relevant features.

Code: ***from statsmodels.graphics.tsaplots import plot_pacf***





Similarly, we will keep finding the correlation among the residuals for i-3, i-4.and so on.

It's a bit complicated because we're introducing this third step of calculating the residuals and then plotting the residuals fitted on the day before against the actual data for the day before that one.

We essentially plot out the relationship between the previous day's residuals versus the real values of the current day. In general, we expect the partial autocorrelation to drop off quite quickly.

ACF VS PACF

In autocorrelation we find out the direct and indirect correlation between the present time series and lagged series.
y(t) --is directly correlated with → y(t-1) --is directly correlated with → y(t-2)
y(t) -- is indirectly correlated with → y (t-2

In case if we don't want the indirect correlation, we will go for PACF. **As the PACF only describes the direct relationship between an observation and its lag.**

Simple Auto Regressive Model

Autoregressive model is used for forecasting future observation as a linear regression of one or more past observations. This model has a parameter **p**

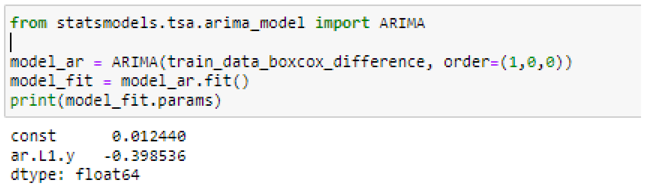
"p is the maximum number of lags.**"**

Step 1: Plot the partial autocorrelation function and check for the past observations which actually influence the future observations.
Step 2: Figure out the observations that have a significant correlation value.
Step 3: These significant partial correlations will be used to build a regressive model similar to a linear regression model.

- Let the highest order lag is 10.
- 2nd,4th and 6th have significant correlation with future observations.
- Model Equation would become: ***y = β0 + β1*y(t-2) + β2*y(t-4) + β3*y(t-6)***
- Here y(t-2), y(t-4) and y(t-6) will be the only independent variables.

Implementation:

- ***"from statsmodels.tsa.arima_model import ARIMA"***
- **ARIMA** has three components**: Auto Regressive, Integrated and Moving Averages**
- Similarly, it has three parameters: **p, d, q**
- Here we are building an Autoregressive model so we only need to define the value of **p** (p = lag order).
- lag order is the maximum number of lags used to build p numbers on past data points to predict future data points.



```
from statsmodels.tsa.arima_model import ARIMA

model_ar = ARIMA(train_data_boxcox_difference, order=(1,0,0))
model_fit = model_ar.fit()
print(model_fit.params)
```

```
const      0.012440
ar.L1.y   -0.398536
dtype: float64
```

●β0 = Constant

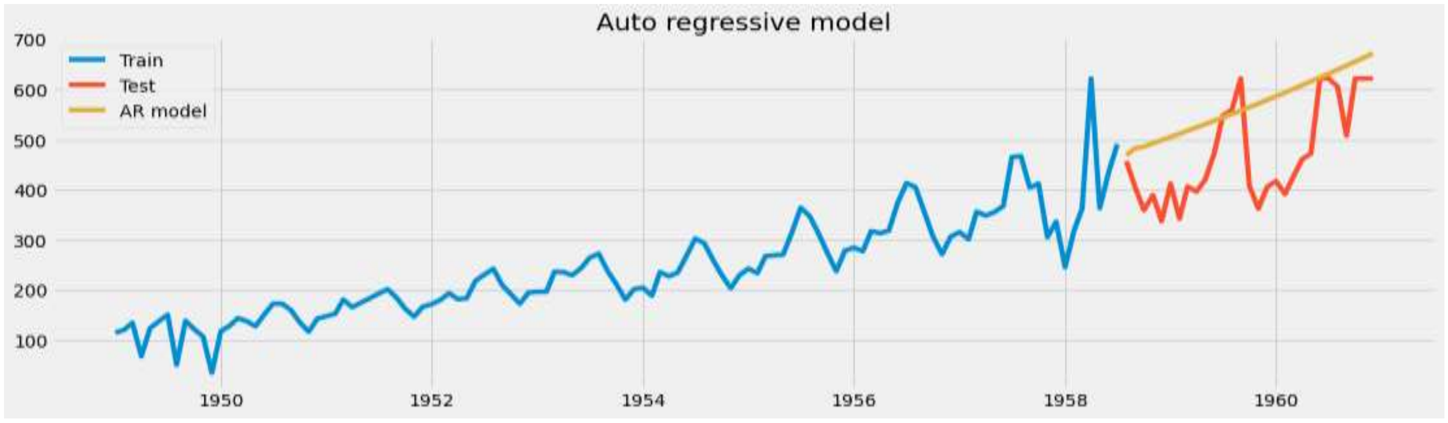
●*β1* = ar.L1.y

Order =(p,d,q)

As here we have fit the train dataset in our model. But We have transformed our series using Differencing & Box Cox transformation. As under prediction we need to get output as in our original series. Let's do it

Step 1 : We used the differencing method so we will take out the cumulative sum using the cumsum function
"cumulative sum is [a, a+b, a+b+c, a+b+c+d]"
Step 2 : We will add the first row as well into the series (As during Differencing we remove the top row)
Step 3 : We will re-transform the series using the exponentiation function.
" np.exp(predicted_value)"

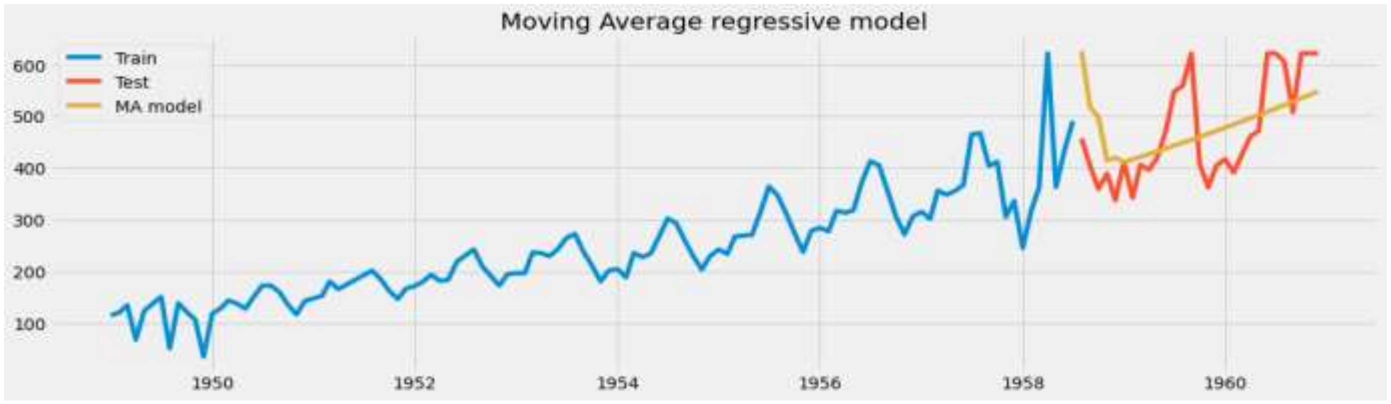
As the box Cox transformation does the log transformation to our series, so to re-transform back to normal form we will do the exponentiation of the series. As in visualization we can see clearly that the AR model has successfully captured the trend in data but not the seasonality from the time series.



The method is suitable for time series without trend and seasonal components.

Moving Average Model

- In the previous Model of AR, we forecast the future observation based on the linear combination of past observations. But in the **Moving Average model,** we consider the past forecasted errors to forecast the future values.
- The Moving average model has a parameter "**q**" which is the size of the Moving average window over which linear combinations of errors are calculated.
- Equations is: **y(t) = μ + φ(k)*ε(t-k)**
- **μ** is the mean of the series (Model output constant value will be this)
- **ε(t-k)** is the past forecasted value
- **φ(k)** is the weight associated with error value (Model output weight value will be this)



The method is suitable for time series without trend and seasonal components.

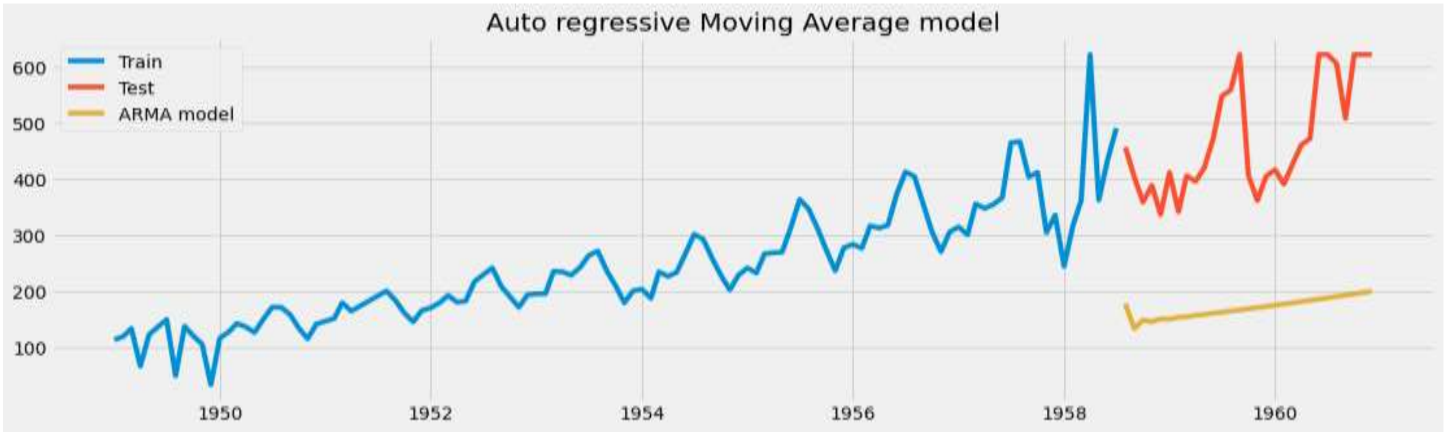
Auto Regressive Moving Average Model (ARMA)

As the name suggests it works the same way, ARMA combines both the AR Model and the MA Model. It describes the time series in two polynomials One for the AR Model and another for the MR Model. It exhibits characteristics of both the Models.

- ARMA combines both AR and MR
- It takes into account one or more past observations as well as the past errors
- It contains two parameters **"p"** and **"q"** where **p = Highest lag** and **q = numbers of past error included**
- Equations: **y(t) = ß0 + ß1*y(t-1) + Ψ1ε(t-1)**
- **ß0** is the Intercept
- **y(t-1)** is the lag value. **ß1** is the weight associated

- **Ψ1** is the weight associated with error, **ε(t-1)** is the error Value.
  Note: This equation is for the case when both p and q are equal to 1.

ARMA equations change with the change in the value of "**p**" and **"q"**. We can determine the value of p and q using autocorrelation & partial autocorrelation plots.



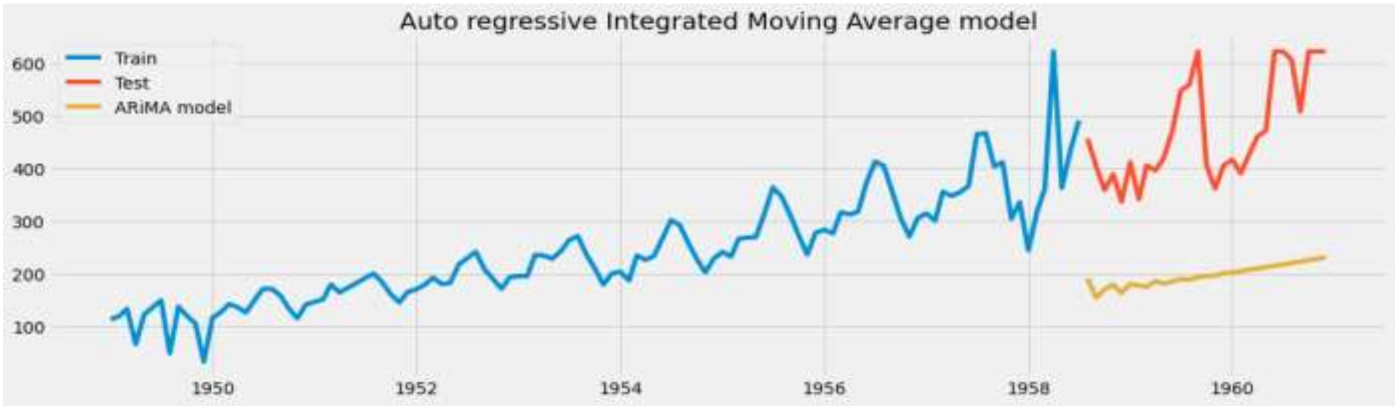
The method is suitable for time series without trend and seasonal components.

Auto Regressive Integrated Moving Average Model (ARIMA)

ARIMA model works similar to the ARMA model, but in ARMA model we first transform the data using Box Cox transformation and then perform differencing in order to convert the Time series stationary. While in the case of ARIMA Model, we only need to transform the time series using Box Cox and then the Model will itself take care of the differencing and remove the trend from the time series.

- It transforms the time series using Box Cox transformation.
- We do not need to do the differencing, AEIMA takes care of differencing itself.
- It removes trends from the time series.
- Parameters: **p - highest lag, q - number of past errors included, d - degree of differencing** to make series stationary
- Equations: **▲y(t) = θ1 *▲y(t-1) + φ1*ε(t-1) + ε(t)**
- Where, ▲y(t-1) = ▲y(t+1) - y(t)
- **θ1** is the weight associated
- **ε(t-1)** is the error term, **φ1** is the weight associated with error term
- **p & q** values can be determined using **ACF & PACF, d -** can be determined using **ADF & KPSS test**

As both the ARMA & ARIMA model fails to capture the Seasonality.

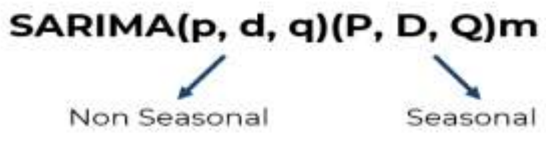


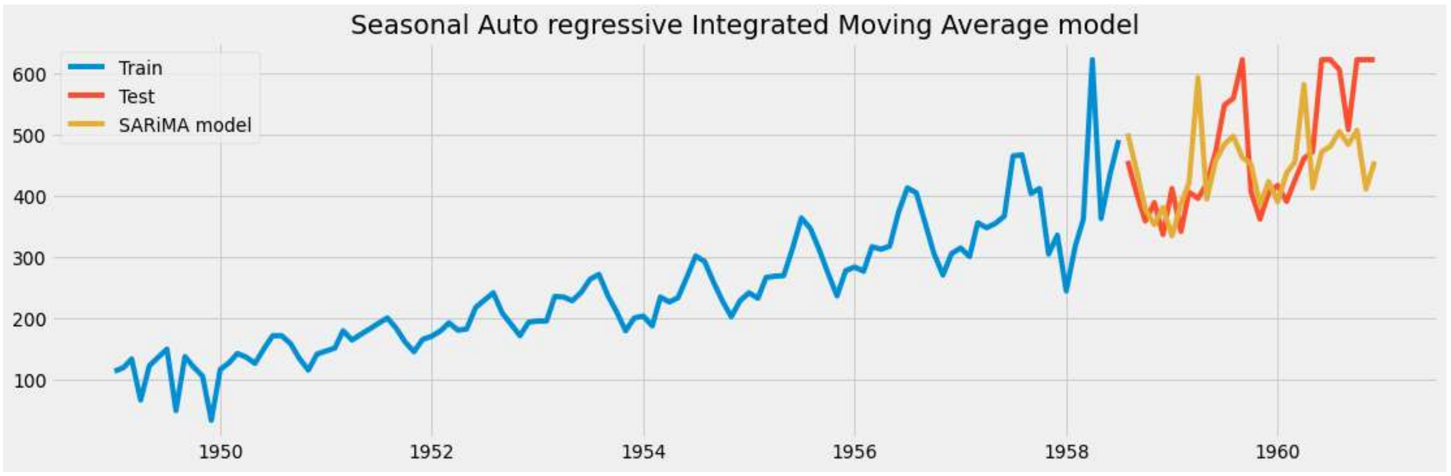
The method is suitable for time series with trend and without seasonal components.

Seasonality Auto Regressive Integrated Moving Average Model

SARIMA model brings all the features of ARIMA model along with the seasonality.

- The Time series is differenced to make it stationary.
- The SARIMA equation is a linear combination of past observations and past errors.
- **Seasonal differencing is performed on the time series.**
- SARIMA models future seasonality as a linear combination of past seasonality observations and past seasonality errors.
- Parameters **(p, d, q), (P, D, Q), m**
- **m = number of time steps for a single seasonal period**
- **P = the seasonal auto aggressive → 0 <= p <= 4**
- **D = Seasonal differencing order → 0, 1, 2**
- **Q = seasonal moving average order → 0 <= q <= 4**
- **p = can get the value using ACF**
- **q = PACF**
- **d = order of differencing - via ADF & KPSS test**
- For determining values of P, Q, D need to perform grid search. As grid search performs goodness of Fit.





The method is suitable for time series with trend and/or seasonal components.

ARIMAX Model (Exogenous Variable)

ARIMAX and ARIMA model works in the same fashion using the default **p, d, q** parameters, but an extra variable in case of the ARIMAX model.

- These extra variables will help us to forecast the time series based on some external factors also.
- **p, q, d** remains the same in case of the ARIMAX model.
- **Addition of an external variable to the model**
  - E.g. - Earlier we were predicting based on the date only as the time series and here we will add a new independent variable to it as well.
  - E.g.: Suppose a company sale went high whenever they launch a new campaign, so adding a new campaign will add as a new independent variable
- ***ARIMA(train['meantemp'], order=(1,0,3), exog= train['humidity'])***
- Rest all the preprocessing steps remain the same
- **Model future observations as a linear regression of the external variable.**

Exogenous Variable: is one whose value is determined outside the model and is imposed on the model.

Endogenous Variable: is a variable whose value is determined by the model

SARIMAX Model (Exogenous Variable + along with seasonal and non-seasonal components)

It has all the features of the SARIMA model.

- The SARIMAX model is the model which, along with the seasonal and non-seasonal components, also models an external variable. **SARIMAX (p, d, q) (P, D, Q, s)**
- Performing differencing to make the time series stationary.
- Model future observations as the linear regression of past observations and past errors.
- Performs seasonal differencing to make data stationary over the season.
- Models' future observations as a linear regression of past observations and past seasonal errors.
- **Model future observations as a linear regression of the external variable.**
- The parameters to be used in SARIMAX model are: **- p, q, d, m, P, D, Q**
- The SARIMAX model has an extra term **αx(t)** which indicates the external variable and **α** denotes the weight associated with it.

Model Selection

Model selection is generally performed based on the strength of the dataset.

- **Smaller Dataset** → values less than 10 data points
- **Larger Dataset** → values greater than 10 data points

Smaller Dataset (datapoints < 10)

- **If Noisy → prefer to go Simple Moving Average Method**
  - As this method helps to cancel out noise to some extent
  - Method will not work well if data has seasonal components or more than 10 obs.
  - E.g.: Daily forecasting of stock pricing -- because the number of observations is fewer and data is noisy. Simple moving average predicts better since it takes variation of very few data points.
- **If neither Noisy nor Seasonal Pattern**
  - The Naive Method works well - as it will forecast value based on the previous trained data.
  - In case of observations more than 10 the Naive forecast trends to underpredict or overpredict the values.
  - E.g.: Forecasting the sales of a grocery store that opened recently, in that case the sale is not dependent on any seasonal component
- Also, in case of not noisy data and seasonality with data points fewer than 10, the seasonal naive method works well.

Larger Dataset (datapoints > 10)

In the case of a large data set we generally perform Smoothing or ARIMA family models to forecast the values.

- Capture → **Level → Simple Exponential Smoothing technique**
- Capture → **Level, Trend → Holt Exponential Smoothing / ARIMA**
- Capture → **Level, Trend, Seasonality → Holt Winter Exponential Smoothing / SARIMA**
- Capture → **Exogenous Variable (Multi variable) → Level, Trend, Seasonality → ARIMAX / SARIMAX**

Some Important factors while working on Time Series

- **Get rid of Missing Values:** prefer either linear or seasonal interpolation method
- **Remove or Cap outliers**: they will impact and forecast biased values
- **Check stationary whenever apply AR models**
- **Perform Differencing → to remove trend**
- **Perform Box Cox transformation → to remove seasonality**

Code is available on GitHub and free to use or update