Time Series

By- Shreya Shukla & Saumya Shukla

CONTENTS-

What is a Time Series?

Time Series is generally data which is collected over time and is dependent on it. A series of data points collected in time order (at constant time intervals) is known as a time series.

These are analysed to determine the long-term trend so as to forecast the future or perform some other form of analysis.

Most of business work on time series data is to analyse sales number for the next year, website traffic, count of traffic, number of calls received, etc.

Time series analysis comprises methods for analysing time series data in order to extract meaningful statistics and other characteristics of the data.

Time series forecasting is the use of a model to predict future values based on previously observed values.

Components of a Time Series

- **Systematic**: Components of the time series that have consistency or recurrence and can be described and modelled.
- **Non-Systematic**: Components of the time series that cannot be directly modelled.

A given time series is thought to consist of three systematic components including level, trend, seasonality, and one non-systematic component called noise.

1) **Level**: The baseline value for the series if it were a straight line.

2) **Trend**: The optional and often linear increasing or decreasing behaviour of the series over time.

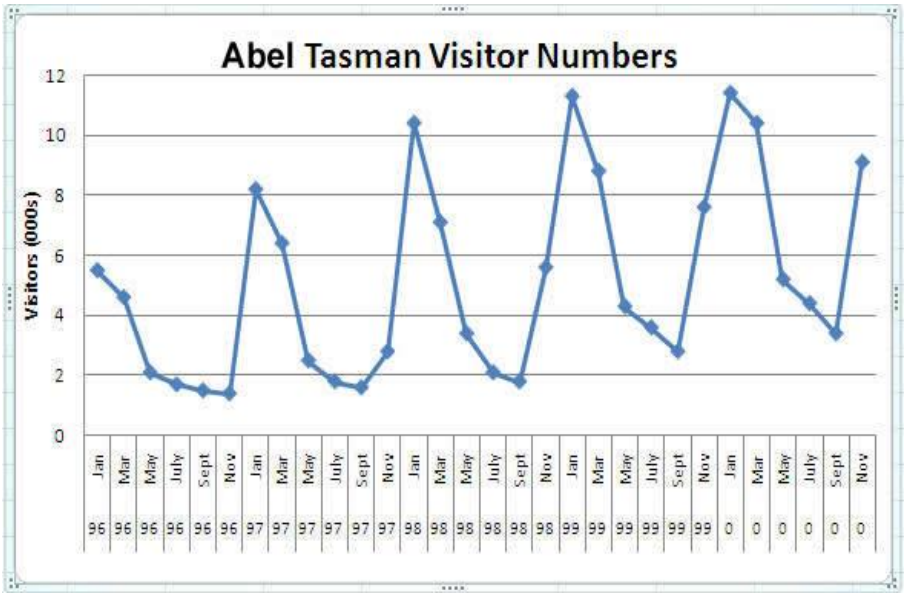
3) **Seasonality**: The optional repeating patterns or cycles of behaviour over time.

4) **Noise**: The optional variability in the observations that cannot be explained by the model.

TREND-

Here the red line represents an increasing trend of the time.

SEASONALITY-



We can see that the time series is repeating its pattern after every 12 months i.e. there is a peak every year during the month of January and a trough every year in the month of September, hence this time series has a seasonality of 12 months.

Difference between a time series and regression problem

- The main difference is that a time series is time dependent. So, the basic assumption of a linear regression model that the observations are independent doesn't hold in this case.
- Along with an increasing or decreasing trend, most Time Series have some form of seasonality trends, i.e. variations specific to a particular time frame.

So, predicting a time series using regression techniques is not a good approach.

Combining Time Series Components

A series is thought to be an aggregate or combination of these four components. All series have a level and noise. The trend and seasonality components are optional. These components can be combined either additively or multiplicatively**.**

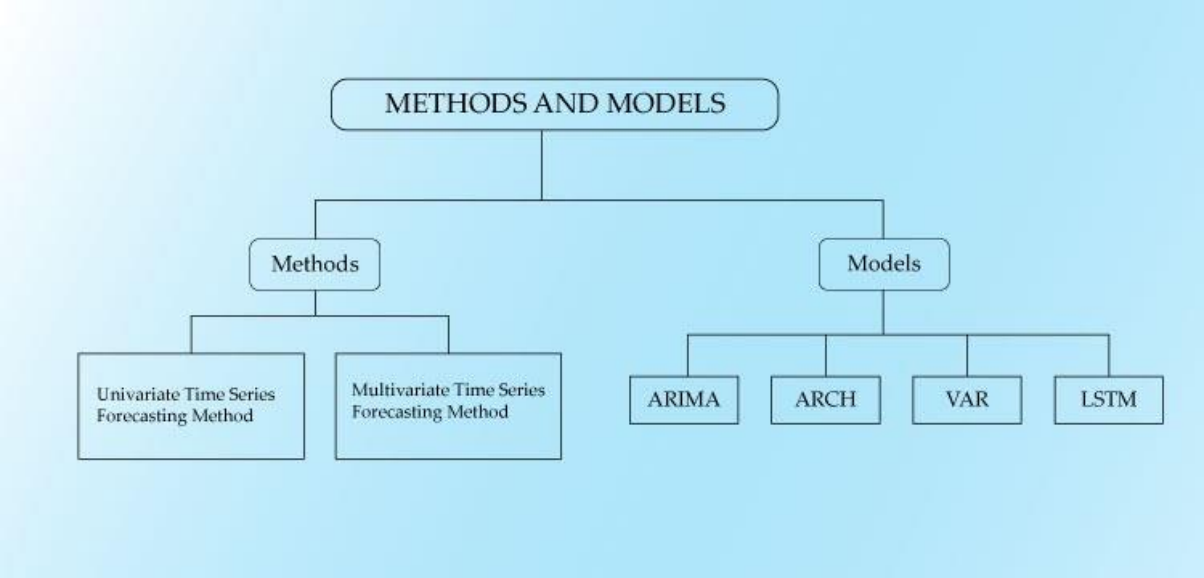
Additive Model

- An additive model suggests that the components are added together as follows:
- *y(t) = Level + Trend + Seasonality + Noise*
- An additive model is linear where changes over time are consistently made by the same amount.
- A linear trend is a straight line.
- A linear seasonality has the same frequency (width of cycles) and amplitude (height of cycles).

Multiplicative Model

- A multiplicative model suggests that the components are multiplied together as follows:
- *y(t) = Level * Trend * Seasonality * Noise*
- A multiplicative model is nonlinear, such as quadratic or exponential. Changes increase or decrease over time.
- A nonlinear trend is a curved line.
- A non-linear seasonality has an increasing or decreasing frequency and/or amplitude over time.

METHODS AND MODELS FOR TIMESERIES-



Methods

In the *Univariate Time-series* Forecasting method, forecasting problems contain only two variables in which one is time and the other is the field we are looking to forecast. For example, if you want to predict the mean temperature of a city for the coming week, now one parameter is time (week) and the other is a temperature.

On the other hand, in the *Multivariate Time-series* Forecasting method, forecasting problems contain multiple variables keeping one variable as time fixed and others will be multiple in parameters. Consider the same example, predicting the temperature of a city for the coming week, the only difference would come here now temperature will consider impacting factors such as rainfall and time duration of raining, humidity, wind speed, precipitation, atmospheric pressure, etc, and then the temperature of the city will be predicted accordingly. All these factors are related to temperature and impact it vigorously.

Models

ARIMA Model: It is a combination of three different models itself, AR, MA and I, where "AR" reflects the evolving variable of interest is regressed on its own prior values, "MA" infers that the regression error is the linear combination of error terms values happened at various stages of time priorly, and "I" shows the data values are replaced by the difference between their values and the previous values. Combinedly "ARIMA" tries to fit the data into the model, and also ARIMA depends on the accuracy over a broad width of time series.

ARCH/GARCH Model: Being the extended model of its common version GARCH, Autoregressive Conditional Heteroscedasticity (ARCH) is the most volatile model for time series forecasting, and are well trained for catching dynamic variations of volatility from time series.

Vector Autoregressive Model or VAR model: It gives the independencies between various time-series data which as a generalization of the Univariate Autoregression Model.

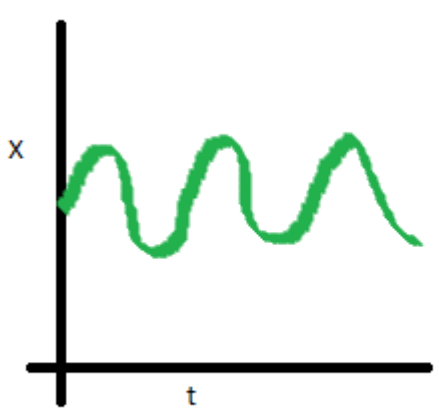
LSTM: Long-short term memory (LSTM) is a deep learning model, it is a kind of Recurrent Neural Network (RNN) to read the sequence dependencies. It enables us to handle long structures during training the dataset and creates predictions according to previous data.

<u>Multiple applications of the Time-Series Analysis</u>

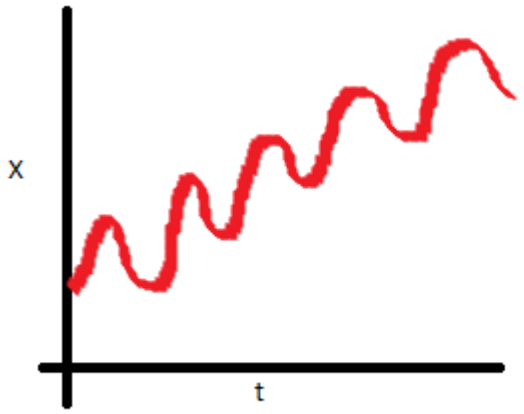
Stationary Series

There are three basic criteria for a series to be classified as stationary series:

1. The mean of the series should not be a function of time rather should be a constant.

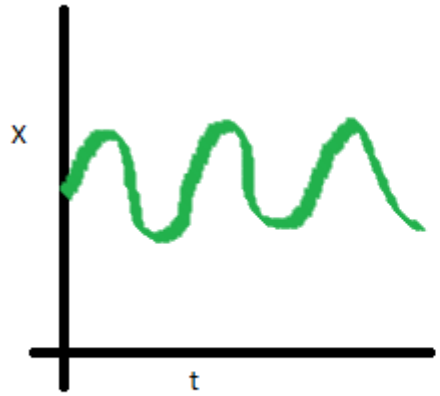


Stationary series

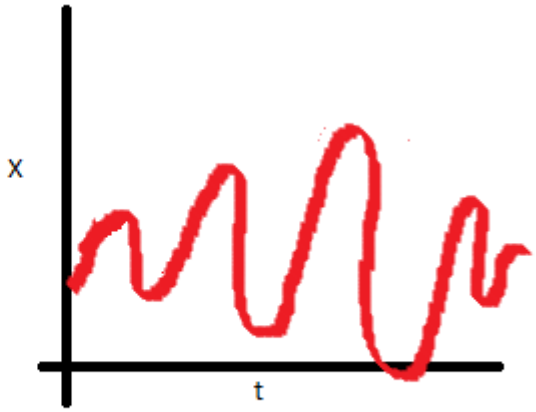


Non-Stationary series

2. The variance of the series should not a be a function of time. This property is known as homoscedasticity.

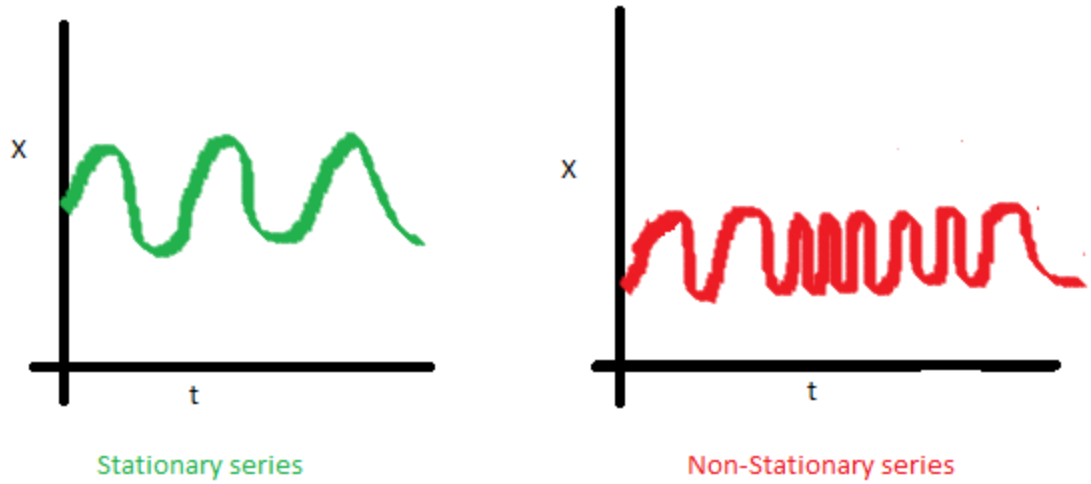


Stationary series



Non-Stationary series

3. The covariance of the i th term and the (i + m) th term should not be a function of time.



Stationary series

Non-Stationary series

Why do we have to make the time series stationary?

We make the series stationary to make the variables independent. Variables can be dependent in various ways, but can only be independent in one way. So, we will get more information when they are independent. Hence the time series must be stationary.

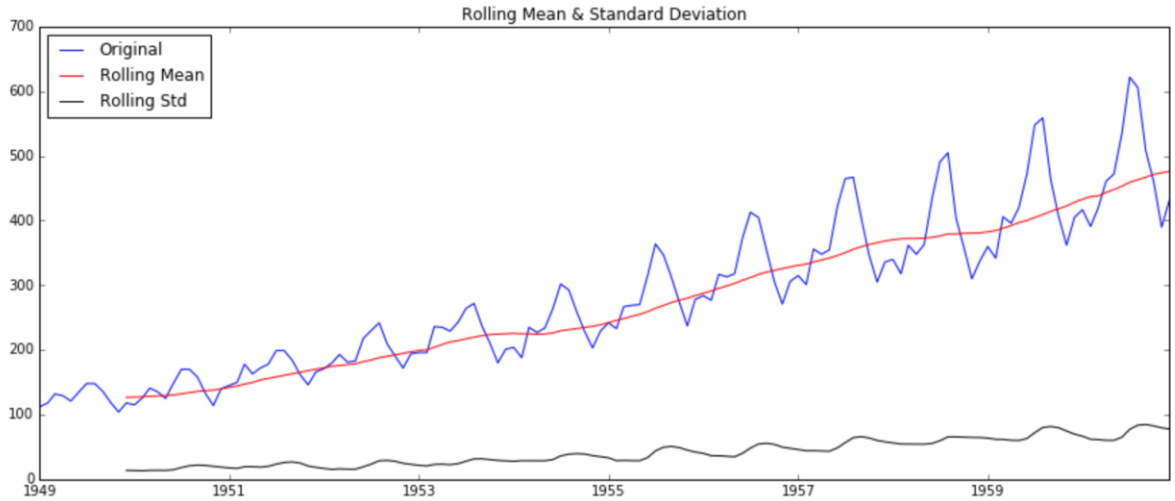
We can check stationarity using the following:

1. **Plotting Rolling Statistics:** We can plot the moving average or moving variance and see if it varies with time. By moving average/variance I mean that at any instant 't', we'll take the average/variance of the last year, i.e. last 12 months. But this is a visual technique.
2. **Dickey-Fuller Test:** The intuition behind this test is that it determines how strongly a time series is defined by a trend.

The null hypothesis of the test is that time series is not stationary (has some time-dependent structure). The alternate hypothesis (rejecting the null hypothesis) is that the time series is 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.

```
from statsmodels.tsa.stattools import adfuller
def test_stationarity(timeseries):

    #Determing rolling statistics
    rolmean = pd.rolling_mean(timeseries, window=12)
    rolstd = pd.rolling_std(timeseries, window=12)

    #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 Observations Used'])
    for key,value in dftest[4].items():
       dfoutput['Critical Value (%s)'%key] = value
    print dfoutput
```




```
Results of Dickey-Fuller Test:
Test Statistic                  0.815369
p-value                         0.991880
#Lags Used                     13.000000
Number of Observations Used    130.000000
Critical Value (5%)             -2.884042
Critical Value (1%)             -3.481682
Critical Value (10%)            -2.578770
dtype: float64
```

How to make a Time Series Stationary?

One of the first tricks to reduce trend can be **transformation**. For example, in this case we can clearly see that the there is a significant positive trend. So, we can apply transformation which penalize higher values more than smaller values. These can be taking a log, square root, cube root, etc.

```
Ex- log transform

ts_log = np.log(ts)
```

We will take rolling average here to remove the trend.

```
moving_avg = pd.rolling_mean(ts_log,12)

train_log_moving_avg_diff = Train_log - moving_avg
```

However, a drawback in this particular approach is that the time-period has to be strictly defined. So we take a 'weighted moving average' where more recent values are given a higher weight. There can be many techniques for assigning weights. A popular one is **exponentially weighted moving average** where weights are assigned to all the previous values with a decay factor.

```
expwighted_avg = pd.ewma(ts_log, halflife=12)

ts_log_ewma_diff = ts_log - expwighted_avg
```

Eliminating Trend and Seasonality-

The simple trend reduction techniques discussed before don't work in all cases, particularly the ones with high seasonality. Let's discuss two ways of removing trend and seasonality:

1. **Differencing** – taking the difference with a particular time lag
2. **Decomposition** – modelling both trend and seasonality and removing them from the model.

Differencing

In this technique, we take the difference of the observation at a particular instant with that at the previous instant. This mostly works well in improving stationarity. First order differencing can be done in Pandas as:

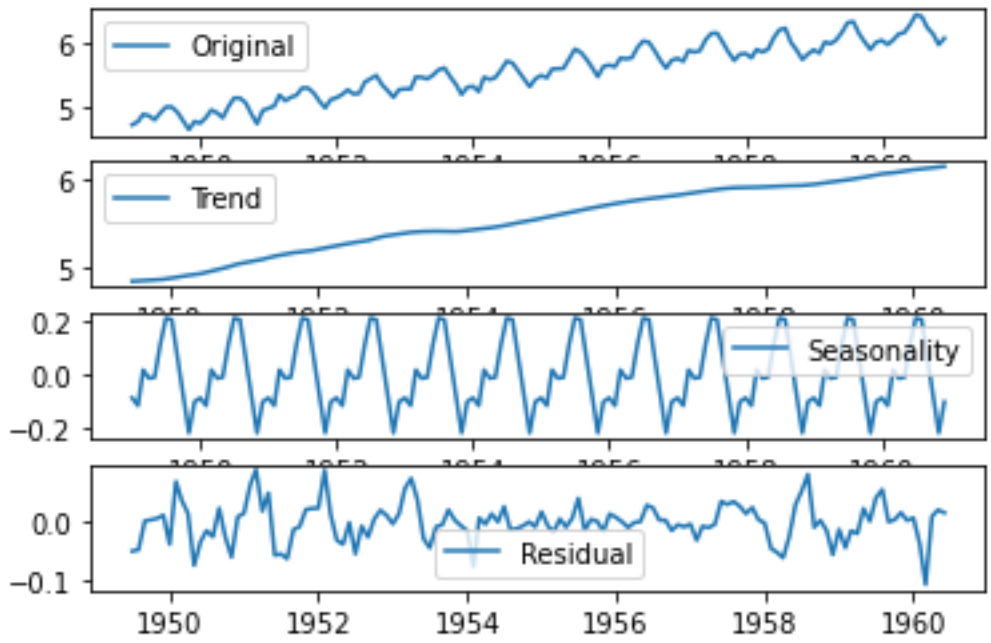
```
ts_log_diff = ts_log - ts_log.shift()
```

Decomposing

In this approach, both trend and seasonality are modelled separately and the remaining part of the series is returned. Seasonal decompose decomposes the time series into trend, seasonality and residuals.

```
from statsmodels.tsa.seasonal import seasonal_decompose

decomposition = seasonal_decompose(pd.DataFrame(Train_log).Count.values, freq = 24)


trend = decomposition.trend

seasonal = decomposition.seasonal

residual = decomposition.resid
```

Trend, Seasonality and Residual Plots



Residual is obtained by removing any trend or seasonality in the time series.

Forecasting a Time Series:

AR model

The autoregressive model specifies that the output variable depends linearly on its own previous values.

An auto regressive time series is one where the value at time ‘t’ depends on the values at times (t-1)….(t-h) superimposed on a white noise term. You define an auto regressive time series AR(h) of the order ‘h’ as a series

$$Xt = a + \sum_{i=1}^{h} bi\, Xt - i + Zt$$

where for some constants a and b. The coefficient b represents the influence (weight) of the value of the time series ‘i’ steps in the past, on the current value.

MA model

The moving-average model specifies that the output variable depends linearly on the current and various past values of a stochastic (imperfectly predictable) term.

A moving average time series is one where the influence of the noise at some time step ‘t’ carries over to the value at t+1, or possibly up to t+h for some fixed ‘h’. Formally, a moving average time series of the order ‘h’ (denoted MA(h)) is

$$Xt = a + Zt + \sum_{i=1}^{h} Zt - i$$

where Zt is the noise at present time stamp. The value at time ‘t’ in an MA(h) process is, therefore, the noise at the current time, t, superimposed on the cumulative weighted influence of the noise at ‘h’ previous time stamp.

Auto regressive Moving Average (ARMA) model:

A time series that exhibits the characteristics of AR(p) and MA(q) process can be modelled using ARMA (p, q). It is also called Classical decomposition method.
Where p is the number of time stamp of AR and q is the number of time stamps of MA model.

$$Xt = a + \sum_{i=1}^{h} bi\, Xt - i + Zt + \sum_{i=1}^{h} Zt - i$$

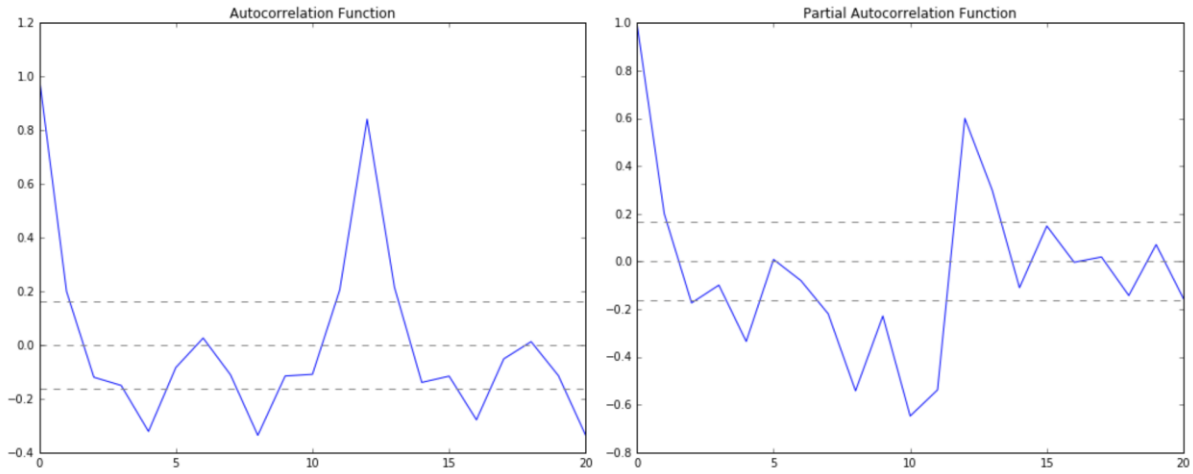
ARIMA (Auto regressive Integrated Moving Average) model-

- ARIMA stands for Auto Regression Integrated Moving Average.
- It combines both Autoregression (AR) and Moving Average (MA) models as well as a differencing pre-processing step of the sequence to make the sequence stationary, called integration (I).
- The notation for the model involves specifying the order for the AR(p), I(d), and MA(q) models as parameters to an ARIMA function, e.g. ARIMA (p, d, q).
- An ARIMA model can also be used to develop AR, MA, and ARMA models.
- Here p is the order of the autoregressive model (number of time lags) For instance if p is 5, the predictors for x(t) will be x(t-1) to x(t-5).
- d is the degree of differencing (number of times the data have had past values subtracted)
- q is the order of moving average model. q is the lagged forecast errors in prediction equation. For instance, if q is 5, the predictors for x(t) will be e(t-1) to e(t-5) where e(i) is the difference between the moving average at i^{th} instant and actual value.
- The ARIMA forecasting for a stationary time series is a linear (like a linear regression) equation.

To find the optimized values of these parameters (p and q) , we will use ACF(Autocorrelation Function) and PACF(Partial Autocorrelation Function) graph.

1. **Autocorrelation Function (ACF):** It is a measure of the correlation between the timeseries with a lagged version of itself. For instance, at lag 5, ACF would compare series at time instant 't1'…'t2' with series at instant 't1-5'…'t2-5' (t1-5 and t2 being end points).
2. **Partial Autocorrelation Function (PACF):** This measures the correlation between the TS with a lagged version of itself but after eliminating the variations already explained by the intervening comparisons. E.g. at lag 5, it will check the correlation but remove the effects already explained by lags 1 to 4.

```
#ACF and PACF plots:
from statsmodels.tsa.stattools import acf, pacf
lag_acf = acf(ts_log_diff, nlags=20)
lag_pacf = pacf(ts_log_diff, nlags=20, method='ols')
#Plot ACF:
plt.subplot(121)
plt.plot(lag_acf)
plt.axhline(y=0,linestyle='--',color='gray')
plt.axhline(y=-1.96/np.sqrt(len(ts_log_diff)),linestyle='--',color='gray')
plt.axhline(y=1.96/np.sqrt(len(ts_log_diff)),linestyle='--',color='gray')
plt.title('Autocorrelation Function')
#Plot PACF:
plt.subplot(122)
plt.plot(lag_pacf)
plt.axhline(y=0,linestyle='--',color='gray')
plt.axhline(y=-1.96/np.sqrt(len(ts_log_diff)),linestyle='--',color='gray')
plt.axhline(y=1.96/np.sqrt(len(ts_log_diff)),linestyle='--',color='gray')
plt.title('Partial Autocorrelation Function')
plt.tight_layout()
```



In this plot, the two dotted lines on either side of 0 are the confidence intervals.

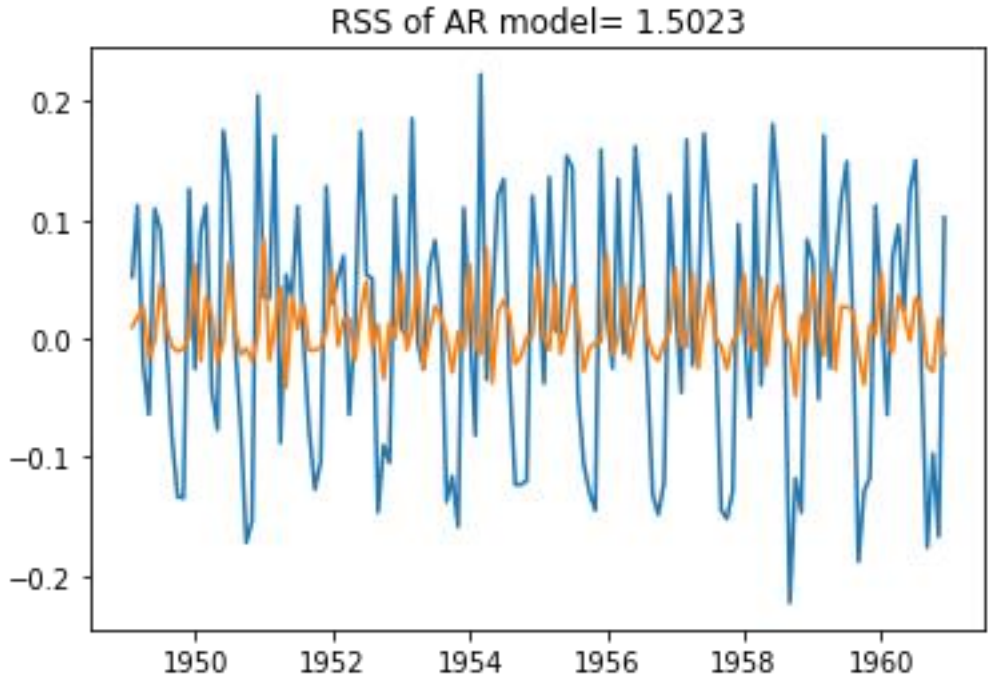
1. **p** = The lag value where the **PACF** chart crosses the upper confidence interval for the first time. In this case p=2.
2. **q** = The lag value where the **ACF** chart crosses the upper confidence interval for the first time. In this case q=2.

from statsmodels.tsa.arima_model import ARIMA

AR -

model = ARIMA (Train_log, order=(2, 1, 0)) # here the q value is zero since it is just the AR model

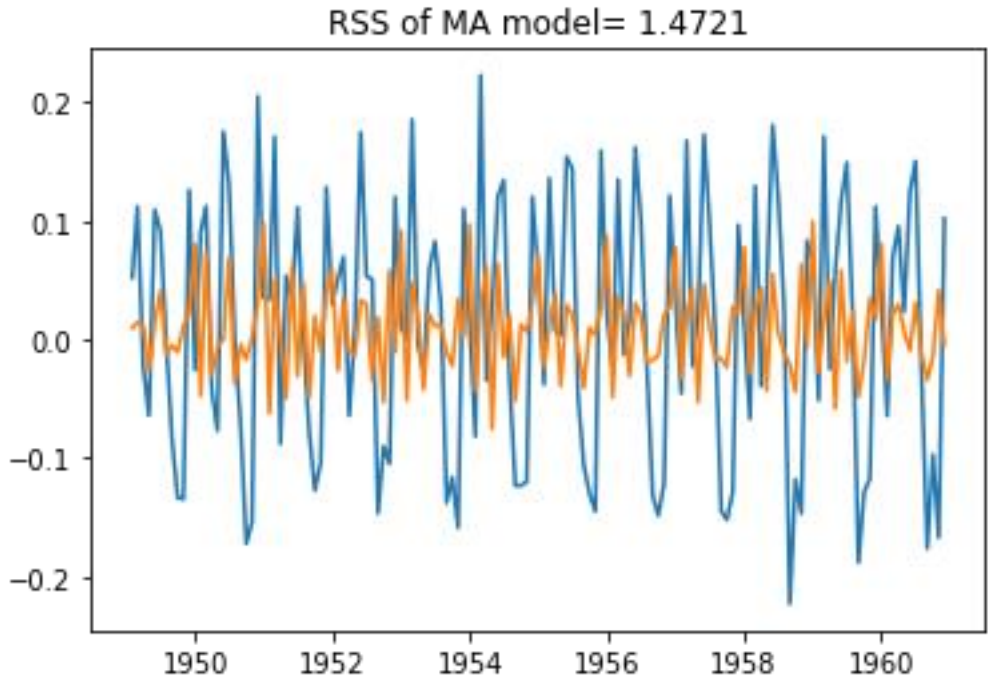
results_AR = model.fit(disp=-1)



MA-

model = ARIMA (Train_log, order=(0, 1, 2)) # here the p value is zero since it is just the MA model

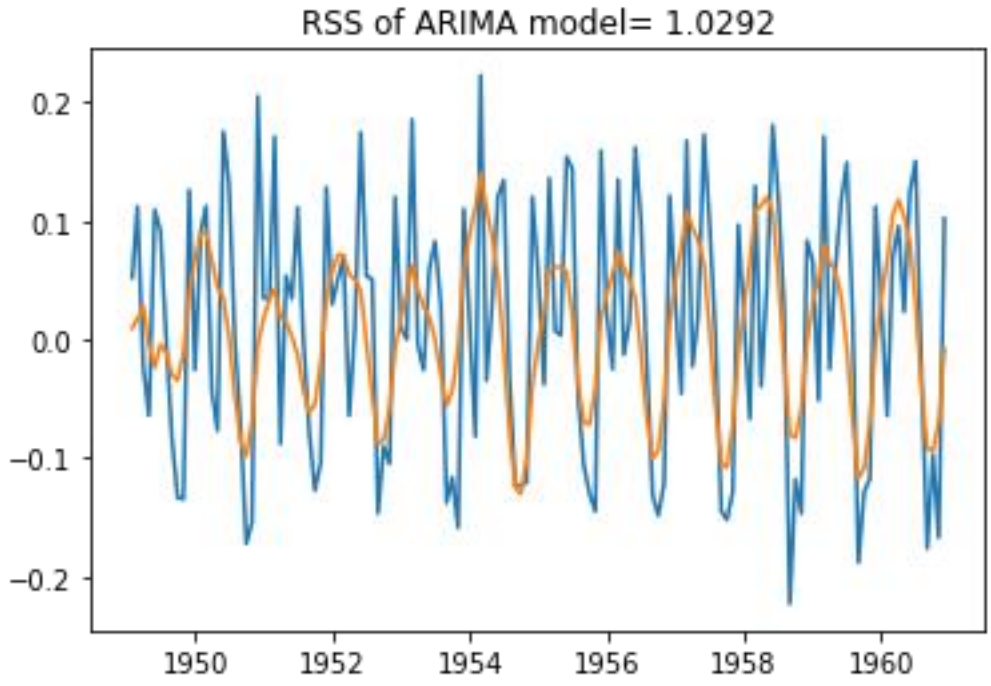
results_MA = model.fit(disp=-1)



ARIMA (Combined model)-

model = ARIMA (Train_log, order=(2, 1, 2))

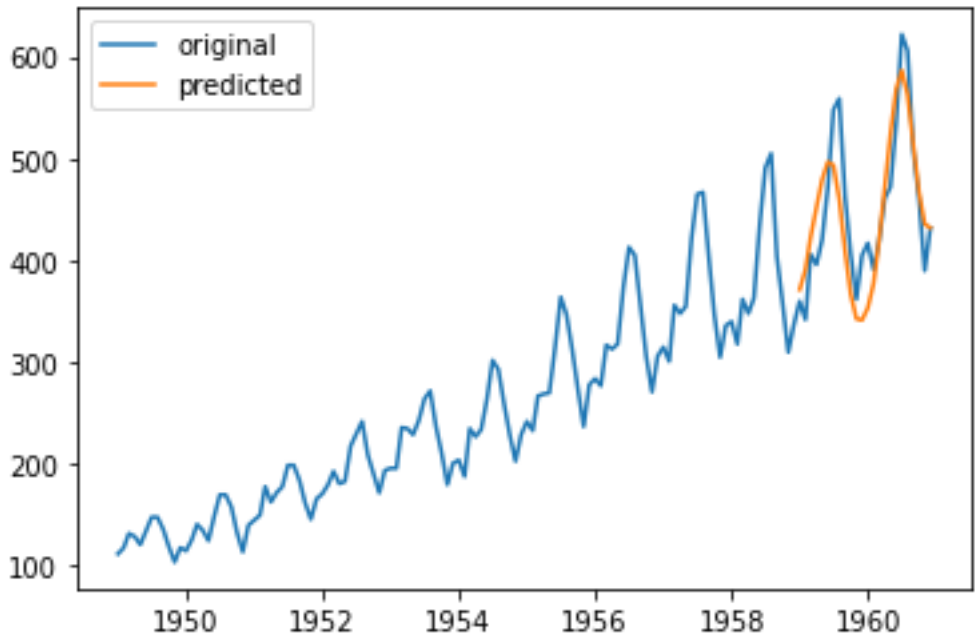
results_ARIMA = model.fit(disp=-1)



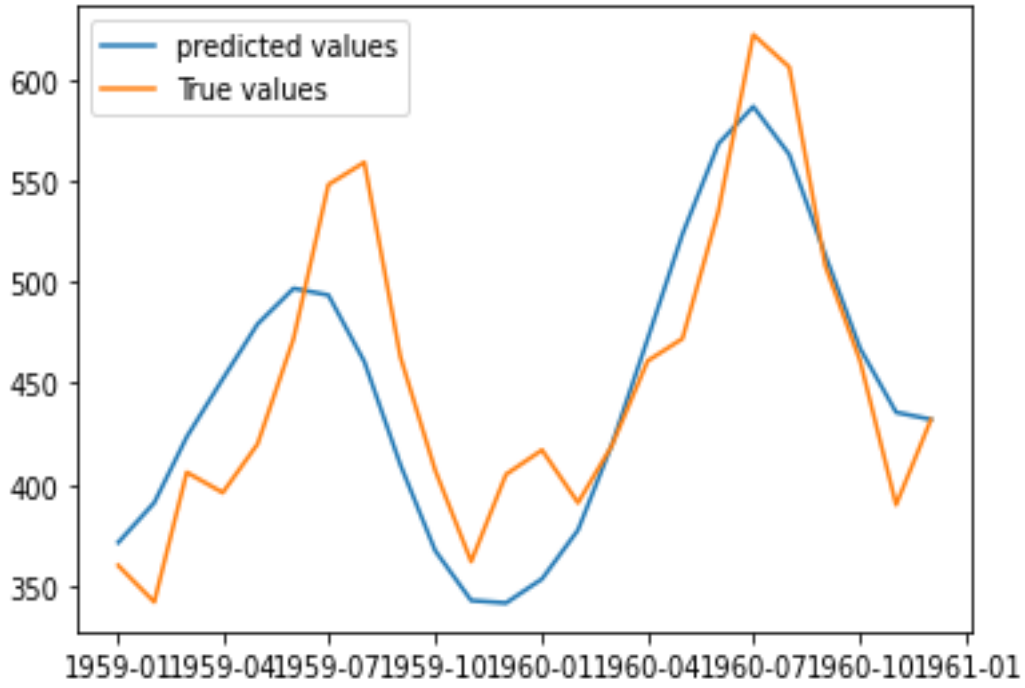
plt.plot(indexed_data,label='original')

plt.plot(pred,label='predicted')

plt.legend()

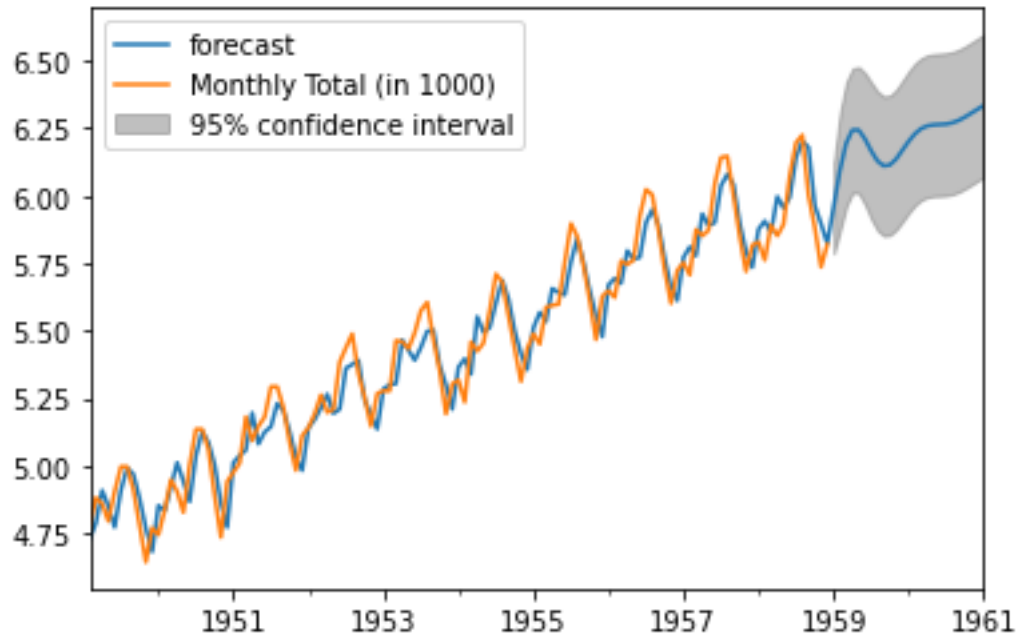


plt.plot(pred,label='predicted values')
plt.plot(TEST,label='True values')
plt.legend()



Predictions for last 2 year (i.e. 2*12=24 months) using arima model

arima.plot_predict(1,144) plt.show()



Modelling Techniques

Various other models to forecast the time series:

i) Naive Approach

ii) Moving Average

iii) Simple Exponential Smoothing

iv) Holt's Linear Trend Model

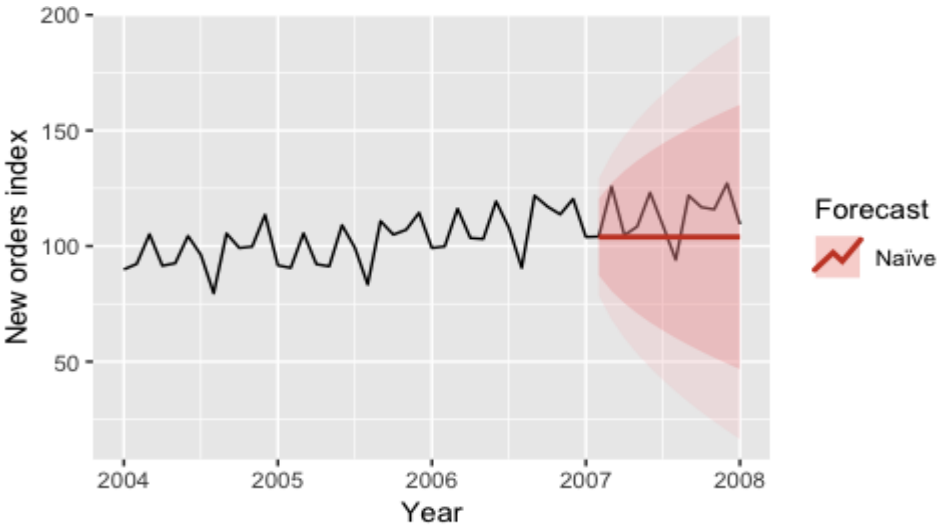
v) Holt winter's model

i) Naive Approach

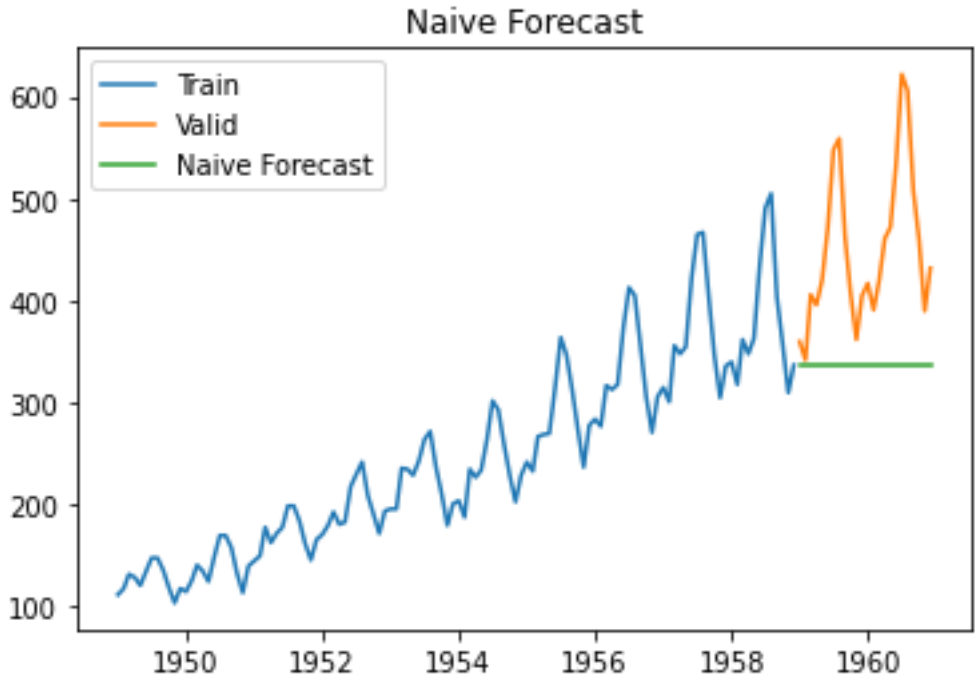
- In this forecasting technique, we assume that the next expected point is equal to the last observed point. So, we can expect a straight horizontal line as the prediction.

E.g. Suppose we have passenger count for 5 days and we have to predict the passenger count for next 2 days. Naive approach will assign the 5th day's passenger count to the 6th and 7th day.

$$\hat{y}_{t+1} = y_t$$



Applying Naive Approach on airline data, gives RMS of 137.328-



ii) Moving Average

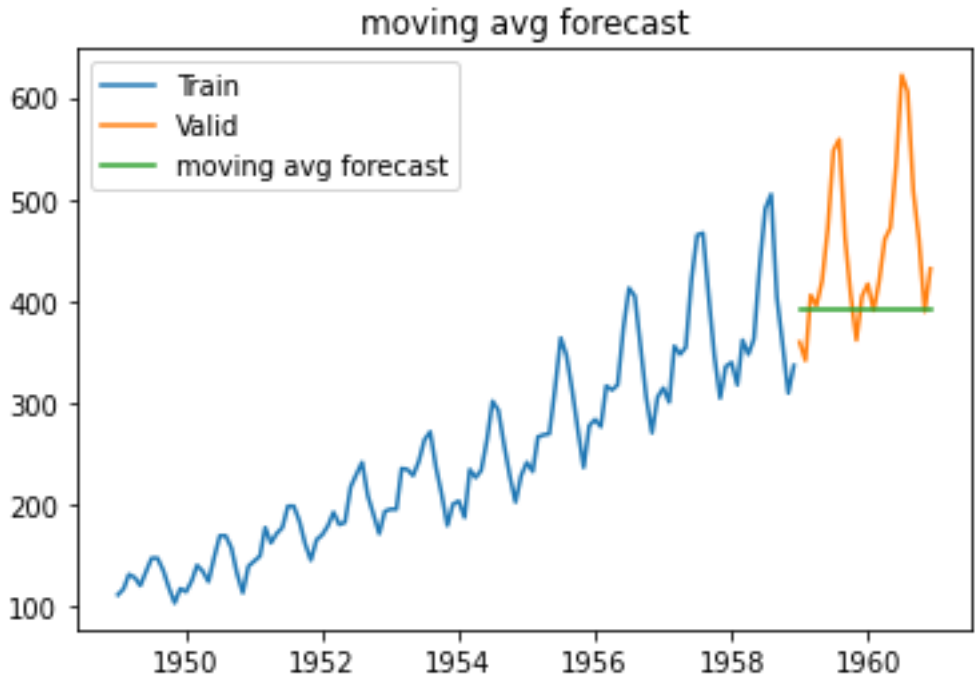
- In this technique we take the average of the passenger counts for last few time periods only.

Example-

Here the predictions are made on the basis of the average of last few points instead of taking all the previously known values.

$$\hat{y}_i = \frac{1}{p}(y_{i-1} + y_{i-2} + y_{i-3} \ldots\ldots + y_{i-p})$$

Applying moving average on airline data, gives RMS of 96.3296-



iii) Simple Exponential Smoothing

- In this technique, we assign larger weights to more recent observations than to observations from the distant past.
- The weights decrease exponentially as observations come from further in the past, the smallest weights are associated with the oldest observations.

NOTE - If we give the entire weight to the last observed value only, this method will be similar to the naive approach. So, we can say that naive approach is also a simple exponential smoothing technique where the entire weight is given to the last observed value.

Example-

Here the predictions are made by assigning larger weight to the recent values and lesser weight to the old values.

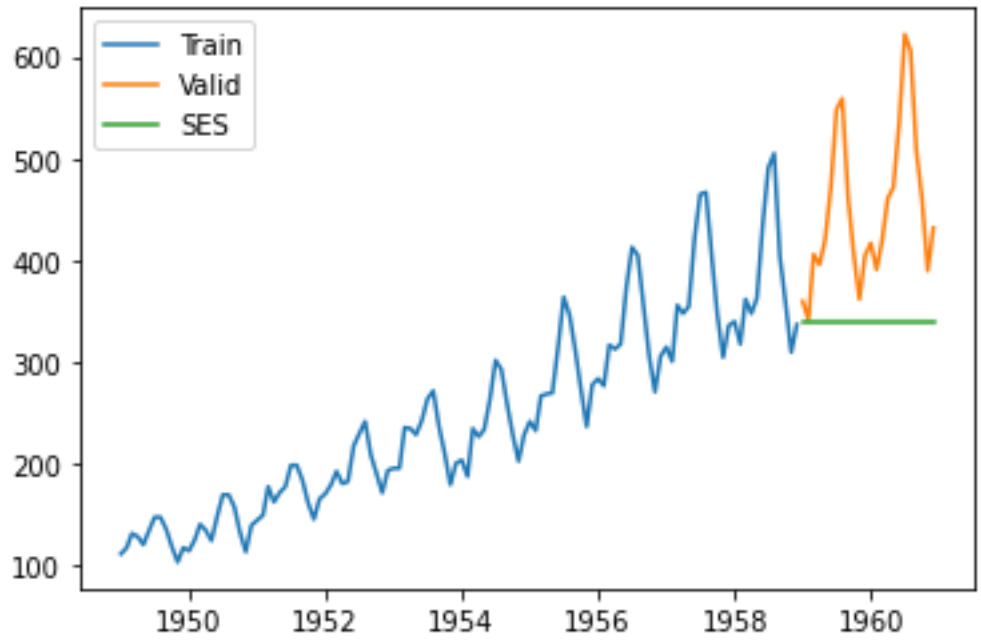
$$\hat{y}_{T+1|T} = \alpha y_T + \alpha(1-\alpha)y_{T-1} + \alpha(1-\alpha)^2 y_{T-2} + \cdots$$

It can also be written as:

$$\hat{y}_{t+1|t} = \alpha * y_t + (1-\alpha) * \hat{y}_{t|t-1}$$

where $0 \leq \alpha \leq 1$ is the **smoothing** parameter.

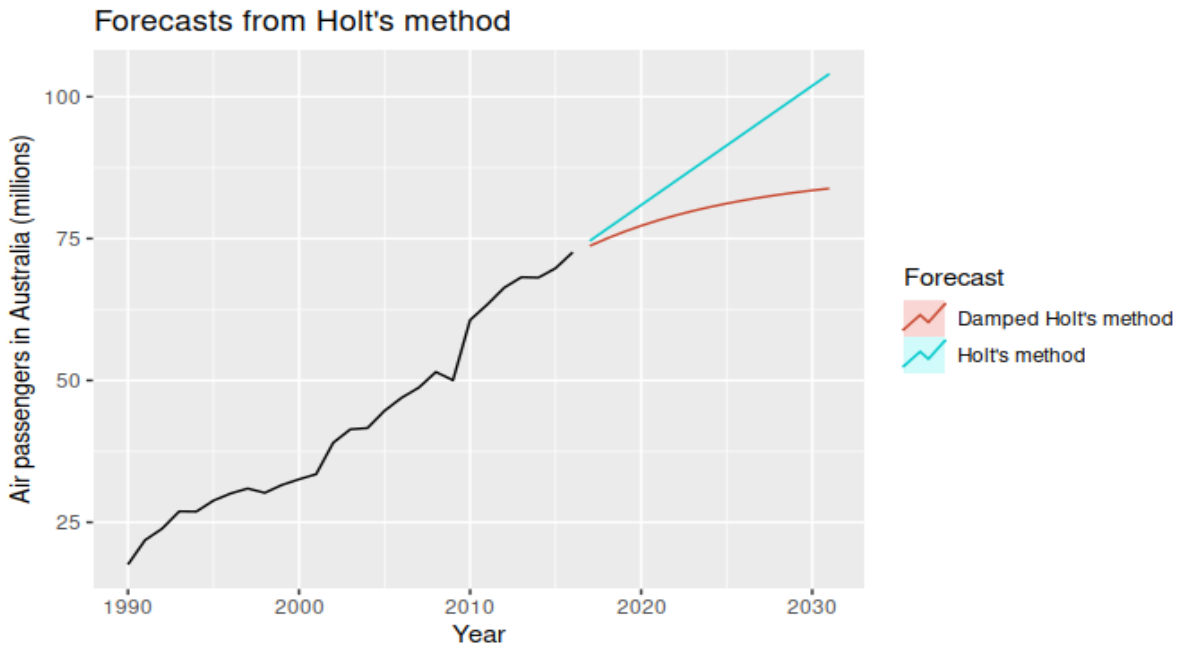
Applying Simple Exponential Smoothing on airline data, gives RMS of 135.650-



iv)Holt's Linear Trend Model

- It is an extension of simple exponential smoothing to allow forecasting of data with a trend.
- This method takes into account the trend of the dataset. The forecast function in this method is a function of level and trend.

Examples-



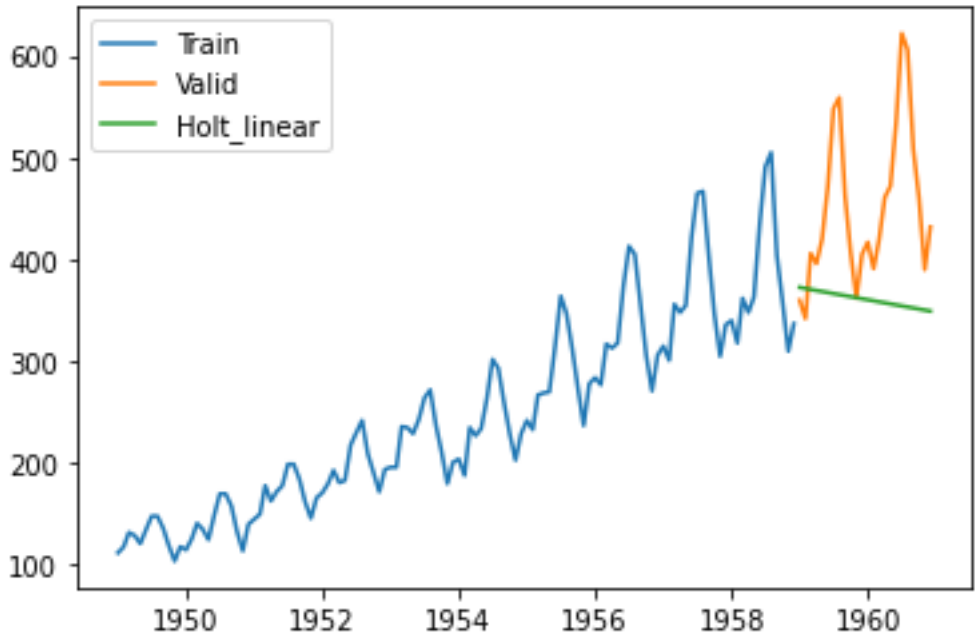
We can see an inclined line here as the model has taken into consideration the trend of the time series.

$$\text{Forecast equation : } \hat{y}_{t+h|t} = \ell_t + h\, b_t$$

$$\text{Level equation : } \ell_t = \alpha y_t + (1-\alpha)(\ell_{t-1}+b_{t-1})$$

$$\text{Trend equation : } b_t = \beta*(\ell_t - \ell_{t-1})+(1-\beta)b_{t-1}$$

Applying Holt's Linear Trend Model on airline data, gives RMS of 119.835-

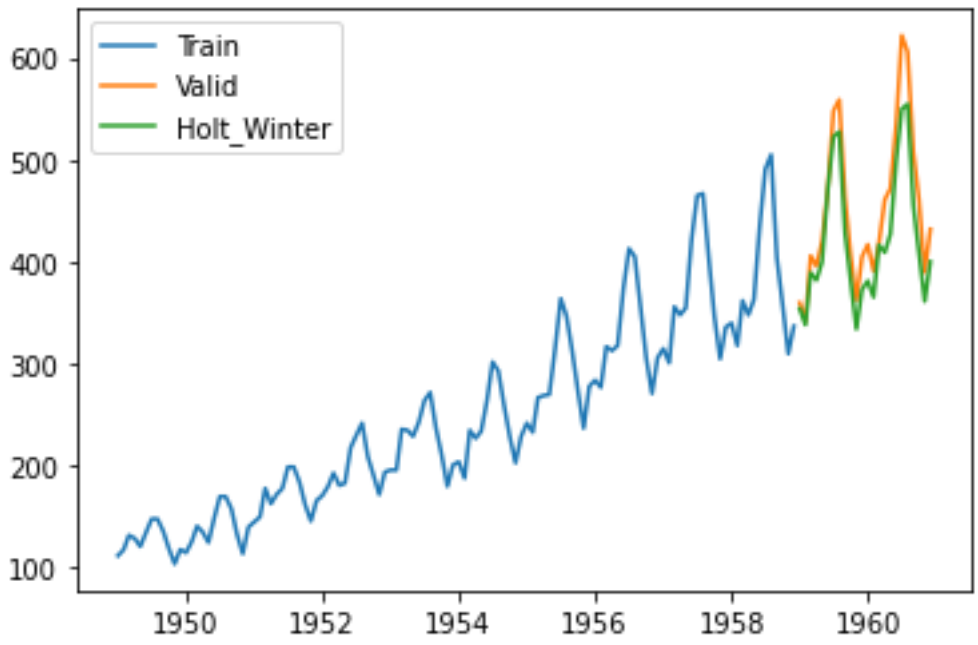


v) Holt winter's model

- Datasets which show a similar set of patterns after fixed intervals of a time period suffer from seasonality.
- The above-mentioned models don't take into account the seasonality of the dataset while forecasting. Hence, we need a method that takes into account both trend and seasonality to forecast future prices.
- One such algorithm that we can use in such a scenario is Holt's Winter method. The idea behind Holt's Winter is to apply exponential smoothing to the seasonal components in addition to level and trend.

$$
\begin{aligned}
\text{level} \quad L_t &= \alpha(y_t - S_{t-s}) + (1-\alpha)(L_{t-1} + b_{t-1}); \\
\text{trend} \quad b_t &= \beta(L_t - L_{t-1}) + (1-\beta)b_{t-1}, \\
\text{seasonal} \quad S_t &= \gamma(y_t - L_t) + (1-\gamma)S_{t-s} \\
\text{forecast } F_{t+k} &= L_t + kb_t + S_{t+k-s},
\end{aligned}
$$

Applying Holt winter Model on airline data, gives RMS of 34.897-



THE END