**Time series:** Timestamp and corresponding data

**Forecast units**: With the increase in forecast units the number of points to be predicted increase.

**Fraction points**

2016.5: Algorithms extracts the start and end date and divides

0.25 3 months

0.5 6 months

**Forecast Settings:**

**Split Point: It determines the amount of actual data to be used**

Eg: 0.5 means 50% of data to be used for prediction.

Default value used is 0.75 i.e 75%.

**Forecast Unit:** It determines how many unit of data to be (predicted)forecasted.

**Lambda:**

1. When value of lambda is NULL it is set to NULL.
2. When value of lambda is set to auto, its value is calculated using BoxCox.lambda function.
3. When lambda is set to auto user can specify lambda value ranging from -5 to 5

**Algorithm**: There are 4 algorithms used to calculate forecast values.

1. **Exponential Smoothing**

Link referred: <https://otexts.org/fpp2/ses.html>

Forecasts are calculated using weighted averages, where the weights decrease exponentially as observations go from further in the past — the smallest weights are associated with the oldest observations

For any α between 0 and 1, the weights attached to the observations decrease exponentially as we go back in time, hence the name “exponential smoothing”. If α is small (i.e., close to 0), more weight is given to observations from the more distant past. If α is large (i.e., close to 1), more weight is given to the more recent observations.

This method is suitable for forecasting data with no clear trend or seasonal pattern.

Alpha is smoothing constant

Forecast for current period is obtained by adding the forecast in last period to the error in last period

**Model Tuning:**

1. When model tuning set to **auto** then parameters are set as given below:
2. es\_errorType = Z
3. es\_trendType = Z
4. es\_sWindow = periodic
5. es\_tWindow = NULL

For more information regarding above model tuning parameters go to this [link](https://www.rdocumentation.org/packages/forecast/versions/8.4/topics/forecast.stl)

1. **Linear Regression**

Linear regression is used to predict the value of an outcome variable *Y* based on one or more input predictor variables *X*. The aim is to establish a linear relationship (a mathematical formula) between the predictor variable(s) and the response variable, so that, we can use this formula to estimate the value of the response *Y*, when only the predictors (*Xs*) values are known.

**tslm method** is used to build tsModel which is used to get forecast values, following are the default values of parameters passed in this method

* formula = tsSeries ~ (0+trend+season)
* lambda = Value calculated using BoxCox.lambda function
* biasadj = false

For more information of tslm go to this [link](https://www.rdocumentation.org/packages/forecast/versions/8.4/topics/tslm)

Forecast values are calculated by passing tsModel in forecast function, to know more about forecast function have a look on [this](https://www.rdocumentation.org/packages/forecast/versions/8.4/topics/forecast.stl).

1. **Neural Networks**

A neural network can be thought of as a network of “neurons” which are organized in layers. The predictors (or inputs) form the bottom layer, and the forecasts (or outputs) form the top layer. There may also be intermediate layers containing “hidden neurons”.

The simplest networks contain no hidden layers and are equivalent to linear regressions. The neural network version of a linear regression with four predictors. The coefficients attached to these predictors are called “weights”. The forecasts are obtained by a linear combination of the inputs. The weights are selected in the neural network framework using a “learning algorithm” that minimizes a “cost function” such as the MSE. Of course, in this simple example, we can use linear regression which is a much more efficient method of training the model.

Once we add an intermediate layer with hidden neurons, the neural network becomes non-linear. This is known as a *multilayer feed-forward network*, where each layer of nodes receives inputs from the previous layers. The outputs of the nodes in one layer are inputs to the next layer. The inputs to each node are combined using a weighted linear combination. The result is then modified by a nonlinear function before being output.

**nnetar function** is used to build the model with the single hidden layer, this model is further passed as parameter to calculate forecast values. Following are the parameters passed while defining a model.

1. When model tuning is set to “auto” then

* lambda = Value calculated using BoxCox.lambda function
* biasadj = false

1. When model tuning is set to “manual” then

* nn\_decay = user can enter values between 0 and 1. Default value will be 0.0009
* nn\_maxit = user can enter values greater than 0. Default value will be 200
* nn\_size = user can enter values greater than 0. Default value will be 20
* nn\_repeats = user can enter values greater than 0. Default value will be 20
* lambda = User can enter values between -5 to +5
* biasadj = Toggle can be changed to true

The above build model is then passed in forecast function to calculate forecasted values.

1. **ARIMA**

ARIMA is an acronym that stands for AutoRegressive Integrated Moving Average. It is a generalization of the simpler AutoRegressive Moving Average and adds the notion of integration.

The parameters of the ARIMA model are defined as follows:

* p: The number of lag observations included in the model, also called the lag order.
* d: The number of times that the raw observations are differenced, also called the degree of differencing.
* q: The size of the moving average window, also called the order of moving average.

To know more about ARIMA go to this [link](https://machinelearningmastery.com/arima-for-time-series-forecasting-with-python/)

**auto.arima** function is used to build the model, which is passed as parameter to calculate forecast value. Following are the parameters passed while defining a model.

1. When model tuning is set to “auto” then

* lambda = Value calculated using BoxCox.lambda function
* biasadj = false

1. When model tuning is set to “manual” then

* max.p = User can enter integer value. Default value is 3.
* max.q = User can enter integer value. Default value is 3.
* max.d = User can enter integer value. Default value is 2.
* max.P = User can enter integer value. Default value is 2.
* max.Q = User can enter integer value. Default value is 2.
* max.D = User can enter integer value. Default value is 1.
* seasonal = Toggle for ON and OFF. Default value is True/ON.
* allowdrift = Toggle for ON and OFF. Default value is False/OFF.
* allowmean = Toggle for ON and OFF. Default value is False/OFF.
* stepwise = Toggle for ON and OFF. Default value is True/ON.
* lambda = User can enter values between -5 to +5
* biasadj = Toggle can be changed to true

To get more information about the above parameters go to this [link](https://www.rdocumentation.org/packages/forecast/versions/8.4/topics/auto.arima)

The above build model is then passed in forecast function to calculate forecasted values.