Objective:

In this session, you will learn to perform analysis on time series data and build multiple models and , validate your forecasted values.

Key takeaways:

- Understanding time series data
- Required preprocessing steps with respective to time series models
    - Identifying time variant
    - Aggregating or preparing the data
    - Handling missing values
    - Split the data into train and test
- Building the following time series models
    - Simple moving Average
    - Weighted Moving Average
    - Exponential smoothing
    - ARIMA models
- Model diagnostics and Model evaluations

Problem Statement:

An e-commerce company offers a specific class of products and the prices of those products varies periodically. They want to understand what is right price to quote on a given period for each product which will help them to increase the sales without effecting on their revenue. The given data contains historical data of a product. Perform time series analysis and forecast the price.

DATA EXPLORATION AND PREPARATION

1. The raw data received in the format '. RData' (i.e. Data file or R image file). Reading and Saving the Data sets as R image file (. RData files) saves lot of time. Else, you may end up executing the same code multiple times to generate the data set for model building.

```
#Import the ".RDdata" files into R
setwd("……")
load("Data.RData")
```

2. Quick review of data

```
dim(data)
head(data)
names(data)
str(data)
tail(data)
```

3. Look at the results of R commands: "head" and "tail". Have you noticed that there are multiple price points on the same day (Because, they are e-commerce company they can change price dynamically!). However, in order to build the time series models, it is required to have only one data point per unit time reference (Say Day, Week or Month, etc.,). Hence you need to aggregate the data day wise. Use the following R code to do so.

```
library(sqldf) # to write SQL like commands in R to aggregate the data.
RtData2.Day <- sqldf("select Date,min(Price) as MIN_PRICE from data group by Date")
```

Note: We are aggregating the data based on minimum price per day. You are free to choose either maximum or average to aggregate the data.

4. Please note that by default, R will read date variables into its environment as factor or character variables. So, to change the variable type into Date format, you could run the code shown below.

```
str(RtData2.Day)
RtData2.Day$Date=as.Date(RtData2.Day$Date,format="%Y-%m-%d")
str(RtData2.Day)
```

5. Let us look the head of the data set now and try to find out if there are any missing values.

```
head(RtData2.Day)
```

```
> head(RtData2.Day)
        Date MIN_PRICE
1 2009-01-01        71
2 2009-01-03        71
3 2009-01-04        71
4 2009-01-05        71
5 2009-01-07        71
6 2009-01-08        71
```

6. Missing values: You will not see “NA” in the data sets. The tricky part is, the missing values in the time series are quite not obvious. You can’t see “NA” directly, however, if you please look at the date field, do you notice that data for few dates are missing (say, Jan 2nd and Jan 6th).
7. Now let us learn to handle the missing values. In this case, we are not ignoring the missing values but wants to replace them. To do so, first create a date field which consists of continuous sequence of dates. We then check against this with the current price data and find out the missing dates.

```
# To find the minimum of the dates
minDate=min(as.Date(RtData2.Day$Date,format="%Y-%m-%d"))
# To find the maximum of the dates
maxDate =max(as.Date(RtData2.Day$Date,format="%Y-%m-%d"))
# generating the series of dates
seq <- data.frame("dateRange"=seq(minDate,maxDate,by="days"))
```

8. Now we join this variable to the current data to see the missing dates

```
# left joining to see the missing values for the dates. all.x will do the left join.”all.y” will do right join.
RtData2.Day2= merge(seq,RtData2.Day,by.x="dateRange",by.y="Date",all.x=T)
head(RtData2.Day2)
```

9. Our goal is to replace the missing values with it’s either proceeding value or succeeding or both. Here is an example to understand how R function na.locf(), from “zoo” library. Not sure why named it as library: Zoo. A zoo without animals but with R functions!

```
#          Here is the example to understand how “na.locf()”  function works
library(zoo)
x <- c(1,2,3,4,5,NA,7,8)
# na.locf function is used to replace the missing values. This will replace the missing value with the it’s immediate preceding value.
na.locf(x)
# This function reverses the sequence
rev(x)
# if you want to replace the missing value with its immediate neighbors, here is the R code. This code is to show that missing value is replaced with it’s preceding and succeeding values
na.locf(x)
rev(na.locf(rev(x)))
(na.locf(x) + rev(na.locf(rev(x))))/2
# Use the above code to replace the missing values in the Price variable

RtData2.Day2$MIN_PRICE=(na.locf(RtData2.Day2$MIN_PRICE) +
```

```
                    rev(na.locf(rev(RtData2.Day2$MIN_PRICE))))/2

# Let us verify this before we move on.
head(RtData2.Day2)
```

10. By looking at the summary, it looks like the price is not changing much over the days. So, let us try to aggregate the price by week instead of days. To do this, add the week column corresponding to each date. The format" %Y.%W" adds another column to our data showing the week number in which each date falls into.

```
RtData2.Day2$WEEK <- as.numeric(format(RtData2.Day2$dateRange,
                        format="%Y.%W"))
 head(RtData2.Day2)
# Now aggregating to weekly data
RtData2Day2 <- RtData2.Day2
head(RtData2Day2)
library(sqldf)
RtData2.Week <- sqldf("select WEEK as WEEK,min(MIN_PRICE) as MIN_PRICE from
RtData2Day2 group by WEEK")
```

Note that we consider the minimum price per day to continue the analysis. You are free to use any of the following aggregations such as Maximum or Average.

11. Great! now you have successfully completed data pre-processing. Let us divide the data into Training and Testing. Please note that unlike other models, the input to the Time series models shouldn't be random. If we split the data randomly, you might miss the records that defines trend and seasonality aspects of the data. Also, you may get the pure random data and building the models on such data makes the model obsolete. So, only alternative is to divide the data sequentially.

```
#Dividing data as Train & Test
Train=RtData2.Week[which(RtData2.Week$WEEK<=2013.47),]
Test=RtData2.Week[which(RtData2.Week$WEEK>2013.47),]
```

TIME SERIES DATA ANALYSIS

12. Note that our target variable "price" is a numeric vector. We are now using this column only to build the models. However, we need to communicate to R that this is a weekly data. The "ts" function gets this job done. The parameter "frequency" 52 indicates the weeks (because we are reading weekly data). If you are reading monthly data, you can change it to 12, for day-wise data you need to specify 365 and 4 to read the quarterly data

```
Price <- ts(Train$MIN_PRICE, frequency =52)
```

13. Now let us visualize the data

```
plot(Price,type="l",lwd=3,col="blue",xlab="week",ylab="Price", main="Time series plot")
```

14. Before we building the models, let us focus on visualization. We first Decompose the data in to Trend and seasonality. Looking at the graph, this data consists of trend and seasonality. Therefore, this time series said be non-stationary. You can also understand the strength of both trend and seasonality by looking at plot ACF and PACF graphs.

```
pricedecomp <- decompose(Price)
plot(pricedecomp)
```

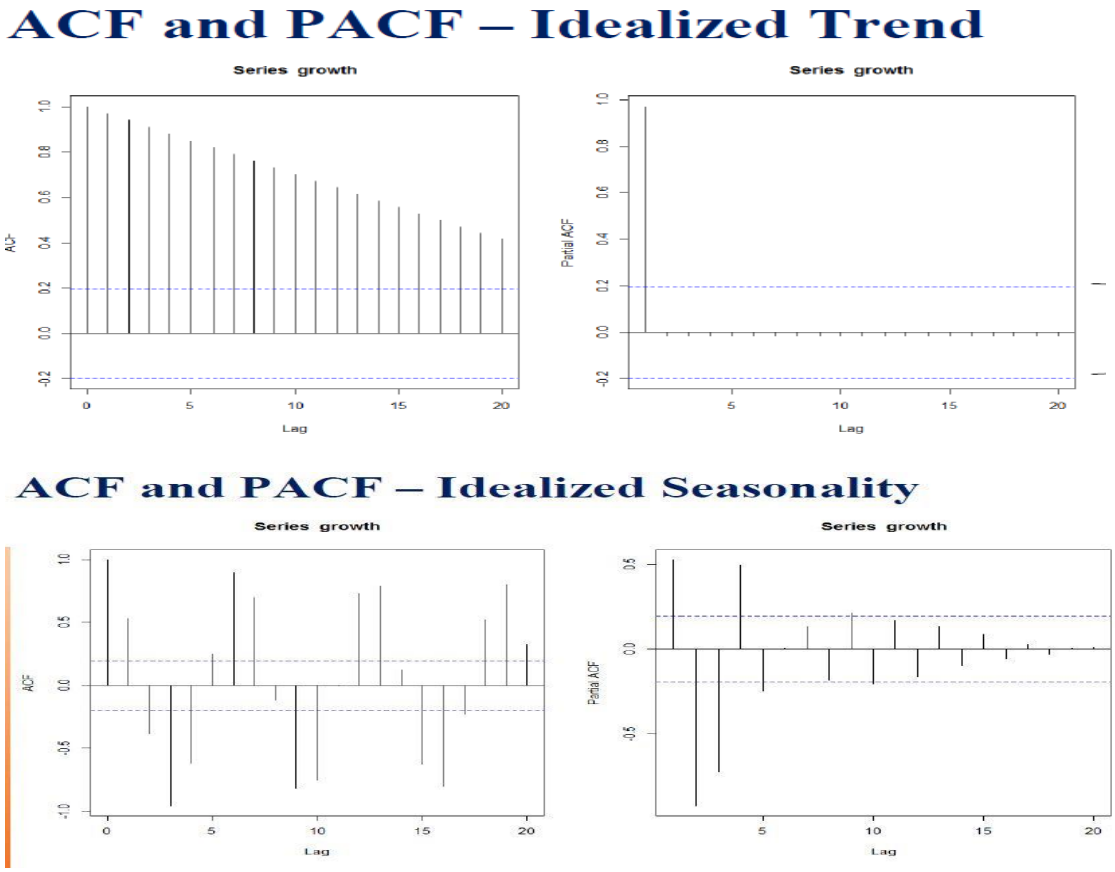
15. Here is the R code to draw the ACF and PACF graphs.
    # par mfrow is to show 2 graphs side by side. You could change it to say c(2,4) to see 4 graphs in 2 rows. C(1,3) to see the 3 graphs in a single row.

```
par(mfrow=c(1,2))
acf(Price,lag=30)
pacf(Price,lag=30)
```

16. If the lag values (on x axis) in the above graphs are difficult to interpret, you can re-read the "time series" data using "ts" function, changing the period to say 1 instead of 52, and generate the graphs. Please use the following R code. Note that this change is just to make the graph better readable and has no other significance. We will continue building the models taking the weekly data only (period is :4)

```
Price1 <- ts(Train$MIN_PRICE, frequency =1)
par(mfrow=c(1,2))
acf(Price1,lag=30)
pacf(Price1,lag=30)
```

17. Understanding ACF and PACF plots
    - **To understand how ACF and PACF graphs are constructed, please refer to the activity in the excel file: ComputingACFvalues.xlsx file**.
    - Look at the idealized graph for trends and seasonality



18. We all now under agreement that this data is non-stationary. To apply models such as moving average models, time series must be converted into Stationary data. We can difference the time series data with certain lag or apply log transformations. Let us start with the differencing the time series data. The next question that comes to our mind is what will be the lag? We need to refer to the ACF and PACF graphs to get some clues. Look at the PACF graph, at lag 1 itself the correlations are fall to near zero levels.
    So, we can start differencing the data using with lag=1 and lag=2 and visualize to see if this makes the data stationary.

```
par(mfrow=c(1,2))
plot(diff(Price1,lag=1),type="l")
plot(diff(Price1,lag=2),type="l")
```

MODEL BUILDING – MOVING AVERAGES METHODS (SMA, WMA AND EMA)

19. Since we understand that the time series has trend, let us build the simple moving average model.

```
# The library TTR stands for Technical trading rules.
library(TTR)
fitsma <- SMA(Price,n=2)
length(fitsma)
length(Price)

Let us see how this model performs. You could choose any of the error metrics. Here we used MAPE to compute the error.
smaMape <- mean(abs((Price[2:length(Price)]-fitsma[2:length(Price)])/Price[2:length(Price)]))
smaMape
```

Play with n =3 and report the error metric on both Train and Test data.

20. Now fit weighted moving average and compute the error metrics.

```
fitwma<- WMA(Price,n=2,1:2)
```

21. Build exponential moving average model and evaluate it. Compare this value with value you obtained for SMA model.

```
fitEma <- EMA(Price, n = 2)
emaMape <- mean(abs((Price[2:length(Price)]-fitEma[2:length(Price)])
                /Price[2:length(Price)]))
emaMape
```

If you want to understand how these models are computed. Please refer to excel file : "movingAverages.xlsx" file.

22. Visualize the models built so far, please use following reference code.

```
par(mfrow=c(2,2))
plot(Train$MIN_PRICE, type="l", col="black")
plot(fitsma, type="l", col="red")
plot(fitwma, type="l", col="blue")
plot(fitEma, type="l", col="brown")

par(mfrow=c(1,1))
plot(Train$MIN_PRICE, type="l", col="black")
lines(fitsma,col="red",)
lines(fitwma, col="blue")
lines(fitEma, col="brown")
```

MODEL BUILDING – HOLTS WINTER METHOD

23. Let's move on to build the Holt winter's model. This is the first model to capture the seasonality. This method involves three smoothing equations (one for level and one for trend other for seasonality).

    There are there parameters and below are the descriptions of them

    1. Alpha to smooth the time series data
    2. Beta to handle the trend
    3. Gamma to allow the model to fit the seasonality

```
#Building the Holt winter's model taking only Trend component.
holtpriceforecast <- HoltWinters(Train$MIN_PRICE, beta=TRUE, gamma=FALSE)
```

```
# Look the fitted or forecasted values
head(holtpriceforecast$fitted)
```

Build another Holt winter's model taking both Trend component and Seasonality (additive). Want to understand more about the additive and multiplicative seasonality's, please refer to the graphs below. The volume of the seasonality is adding up in the additive seasonality while it's getting multiplied in the multiplicative seasonality.

```
priceholtforecast <-  HoltWinters(Price, beta=TRUE, gamma=TRUE,
                        seasonal="additive")
# Look the fitted or forecasted values . Did you  notice the
head(priceholtforecast$fitted)
```

Note: Look the fitted or forecasted values. You will see the values for smoothing parameter and Trend coefficient and the seasonality components. Since you are building the models on weekly data, you will get 52 seasonal components. If you are reading the monthly data, you will get 12 seasonal components.

24. Let us evaluate the algorithm on the Train data. You need to compute MAPE. Convert the output you got from "priceholtforecast$fitted" and store in a data frame and then take the "xhat" column. This column contains the predictions on the training data

```
# Getting the predictions on Training data
holtforecastTrain <- data.frame(priceholtforecast$fitted)
holtforecastTrainpredictions <- holtforecastTrain$xhat
head(holtforecastTrainpredictions)
# To get the predictions on Test Data, you can use function "forecast.Holt". "h" indicates the number of future weeks (or whatever be your reference time period, say months, quarters, etc.,) for which you want to get the predictions
priceforecast <-  forecast.HoltWinters(priceholtforecast, h=8)
```

MODEL BUILDING – ARIMA

25. Let us finally build the ARIMA models.
Let us try to manually build the ARIMA time series models manually and look at the residual plots. Inspecting residual plots and error metrics will help in choosing the p,d and q values. You could apply the function "Arima" as well.

```
# Model with no trend and no seasonality.
model1 <- arima(Price,c(0,0,0))
model1
library("forecast")
forecast.Arima(model1, h=4)

model2 <- arima(Price,c(0,1,0))
model2

model3 <- arima(Price,c(0,2,0))
model3

model4 <- arima(Price,c(1,1,1))
model4

par(mfrow=c(1,4))
plot(model1$residuals,ylim=c(-50,50))
plot(model2$residuals,ylim=c(-50,50))
plot(model3$residuals,ylim=c(-50,50))
plot(model4$residuals,ylim=c(-50,50))
```

Now it' easy. You Can apply auto.arima.

```
library("forecast")
MODEL_ARIMA <- auto.arima(Price, ic='aic')
summary(MODEL_ARIMA)
```

Model diagnostics. We can check how good model is by checking the model residuals using BOX Jenkins test. This statistic can be used to examine residuals from a time series are not correlated in order to see if all underlying population autocorrelations for the errors may be 0.

Null hypothesis: error is not correlated
Alt hypothesis: error is correlated

```
# Lest us look at the acf and Pacf graphs to check if
# there are patterns
acf(as.numeric(MODEL_ARIMA$residuals) ,lag.max = 20,
   main = "Residuals ACF plot")
pacf(as.numeric(MODEL_ARIMA$residuals) ,lag.max = 20,
   main = "Residuals PACF plot")
```

Let us get the predictions on the Test data using different ARIMA models we built. Look at the prediction results from model 1. Remember that the model 1 was constructed with no trend and no seasonality. You will notice that all future predictions are the same. Let us try to understand the reasoning behind this. Compute the mean of the Train data. This is exactly the same as the result of all future predictions. Do You understand that a model1 is nothing but an intercept model without trend and no seasonality? The intercept is nothing but the average on the training data? In model 2 and model3, we are adding the Trend and Seasonality components.

```
# getting the predictions on the Test Data from model.
pricearimaforecasts <- forecast.Arima(MODEL_ARIMA, h=4)
```