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 validation sets
- Smoothing techniques
  - Simple moving Average
  - Weighted Moving Average
  - Exponential smoothing
- Building the following time series models
  - Holt-Winters model
- 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 right price is 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.
data_daily <- 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(data_daily)
data_daily$Date=as.Date(data_daily$Date,format="%Y-%m-%d")
str(data_daily)
```

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

```
head(data_daily)
```

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(data_daily$Date,format="%Y-%m-%d"))
# To find the maximum of the dates
maxDate =max(as.Date(data_daily$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.
data_daily2= merge(seq,data_daily, by.x="dateRange",by.y="Date",all.x=T)
head(data_daily2)
```

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

data_daily2$MIN_PRICE=(na.locf(RtData2.Day2$MIN_PRICE) +
              rev(na.locf(rev(RtData2.Day2$MIN_PRICE))))/2
```

```
# Let us verify this before we move on.
head(data_daily2)
```

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.

```
data_daily2$WEEK <- as.numeric(format(data_daily2$dateRange,
                               format="%Y.%W"))
 head(data_daily2)
# Now aggregating to weekly data
data_weekly<- sqldf("select WEEK as WEEK,min(MIN_PRICE) as MIN_PRICE from data_daily2
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 validation. 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 & validation
Train=data_weekly[which(data_weekly$WEEK<=2013.47),]
validation=data_weekly[which(data_weeklyWEEK>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.

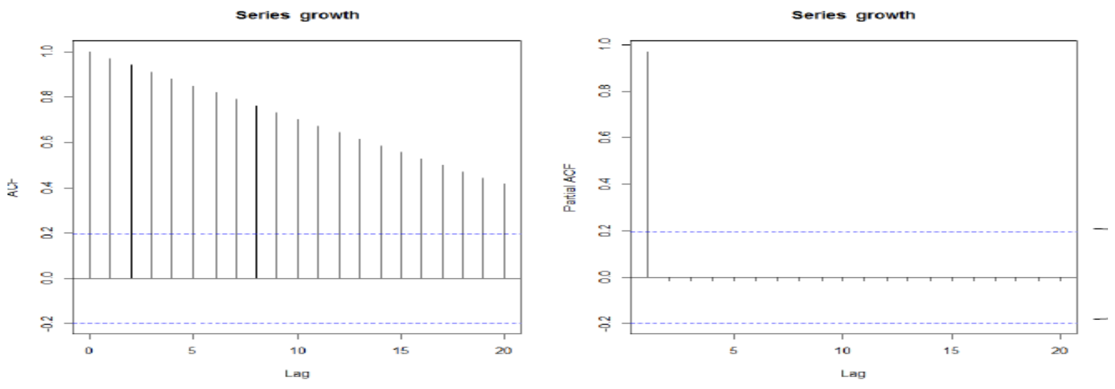
```
par(mfrow=c(1,2))
acf(Price,lag=30)
pacf(Price,lag=30)
```

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)

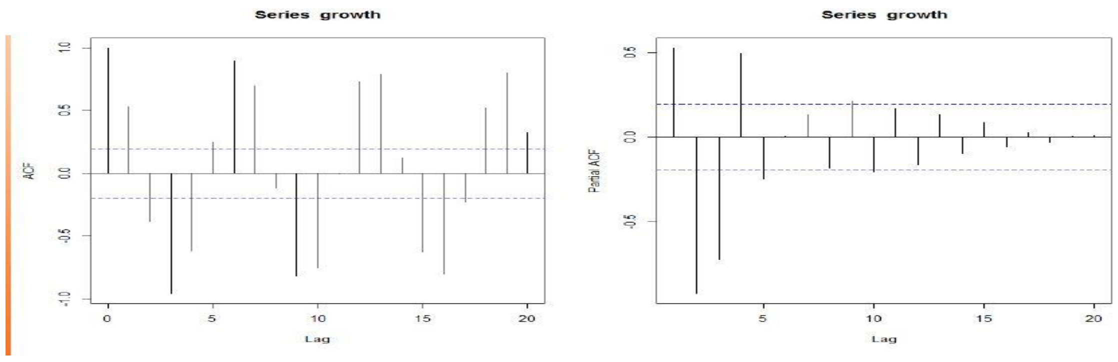
```
Price1 <- ts(Train$MIN_PRICE, frequency =1)
par(mfrow=c(1,2))
acf(Price1,lag=30)
pacf(Price1,lag=30)
```

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

ACF and PACF – Idealized Trend



ACF and PACF – Idealized Seasonality



SMOOTHING TECHNIQUES – MOVING AVERAGES METHODS (SMA, WMA AND EMA)

18. **Smoothing Techniques:** 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)
```

19. Now fit weighted moving average

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

20. Build exponential moving average model

```
fitEma <- EMA(Price, n = 2)
```

21. 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

22. 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.

23. 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 & Checking model
holtforecastTrain <- data.frame(priceholtforecast$fitted)
holtforecastTrainpredictions <- holtforecastTrain$xhat
head(holtforecastTrainpredictions)

# To get the predictions on validation Data, you can use function "forecast". "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(priceholtforecast, h=8)

# Forecasting on validation data
hw_price <- HoltWinters(Price , beta=TRUE, gamma=FALSE)

#hw_price_gamma <- HoltWinters(Price[1:260], beta=TRUE, gamma=TRUE,
seasonal="additive")
hw_price$fitted
train_actuals <- Price[3:261]
train_pred <- data.frame(hw_price$fitted)[1]
DMwR::regr.eval(train_actuals,train_pred)

library("forecast")
hw_price_forecasts = forecast(hw_price,h=1)
validation_preds <- data.frame(hw_price_forecasts)$Point.Forecast
#forecast.HoltWinters(hw_price,h=1)
validation_actuals <- validation$MIN_PRICE
validation_actuals
validation_preds
DMwR::regr.eval(validation_actuals,validation_preds)
```

Problem 1:

Do the following:

1. Use the following code to get stock data of Google from yahoo
2. Consider only the closing price, aggregate the data weekly
3. Implement model and forecast for the next week
4. Report the error

```
install.packages("quantmod")
library(quantmod)
start <- as.Date("2017-01-01")
end <- as.Date("2019-03-30")
getSymbols("GOOGL", src = "yahoo", from = start, to = end)
```

Problem 2:

A leading super market company wants to understand the performance of a particular online orders delivery services unit. In that process, computed on-time delivery rate in last 3 years by monthly and wants to forecast coming months expected patterns. Use 'DeliveryRate.csv' and implement time series analysis on the data.

Resources for more practice problems and material:
https://otexts.com/fpp2/appendix-for-instructors.html
https://otexts.com/fpp2/arima-exercises.html