Time series analysis

1. Introduction

A time series is a sequence of observations recorded in discrete time points.
They can be recorded in hourly (e.g. air temperature), daily (e.g. DJI Average), monthly (e.g. sales) or yearly (e.g. GDP) time intervals.
Real-world time series data commonly contains several components or recognizable patterns.
Those time series components can be divided into

1. **Systematic components** - are those that allow modeling as they have some consistent or recurring pattern. Examples for systematic time series components are
    a. **Trend** - A trend is a consistent increase or decrease in units of the underlying data. Consistent in this context means that there are consecutively increasing peaks during a so-called uptrend and consecutively decreasing lows during a so-called downtrend. Also, if there is no trend, this can be called **stationary**.
    b. **Seasonality** - describes cyclical effects due to the time of the year. Daily data may show valleys on the weekend, weekly data may show peaks for the first week of every or some month(s) or monthly data may show lower levels for the winter or summer months respectively.
2. **Non-systematic** components - any other observation in the data is categorized as unsystematic and is called error or remainder.

The general magnitude of the time series is also referred to with “level”.

In our example, we will be looking wholesale confectionery products data. Our main focus is net revenue sales dataset from year 2017, until end of first quarter of 2020 (3 years and 3 months) by month interval.
We use one dimensional time series analysis techniques and machine learning algorithms, and compare them.

2. Loading libraries

All necessary libraries are loaded and written all together at the beginning of the project. This way we know what packages we used, and all possible changes are made easier.
In **fpp2** are all auto model and plotting functions.
RODBC is used to create connection using ODBC driver, the most popular and basic database connection driver. **Uroot** is used for later in chapter for changing model parameters. **Tsfknn** is package for using knn algorithm on time series data. **Nnfor** is neural network package for time series data.

```
#install.packages("fpp2")
library(fpp2)
#install.packages("uroot")
library(uroot)
#install.packages("RODBC")
library(RODBC)
#install.packages("tsfknn")
library(tsfknn)
#install.packages("nnfor")
library(nnfor)
```

3. Importing data

We will use SQL Server to import data from view object named **TEST_ProdajaPoMesecima**.
Firstly, we create connection object by giving them connection string, contained of **driver's name**, **server's name**, **database**, **uid** - username and **pwd** - password.
Then, we call stored view from server, and get raw data inside data variable.

```
connection <- odbcDriverConnect(connection = "Driver={SQL Server Native Client 11.0};server=sqlrep.algrosso.com;database=AlGrosso Analitycs;uid=ReadOnly;pwd=BBRO.124578;")
?odbcDriverConnect
#getting data from View
data <- sqlFetch(connection, "dbo.Test_ProdajaPoMesecima", colnames = FALSE, rownames = TRUE)
```

Also, **for MAC users**, I provided **data.csv** file. Set file path location to the file data.csv inside **path** variable (keep same structure using \\ marks).

Structured data is needed before procced into next step. For that we will remove last month's incomplete data, present sales in thousand's values, and declare our data as time series data, by defining start period and frequency of our data, which, for month, is 12.

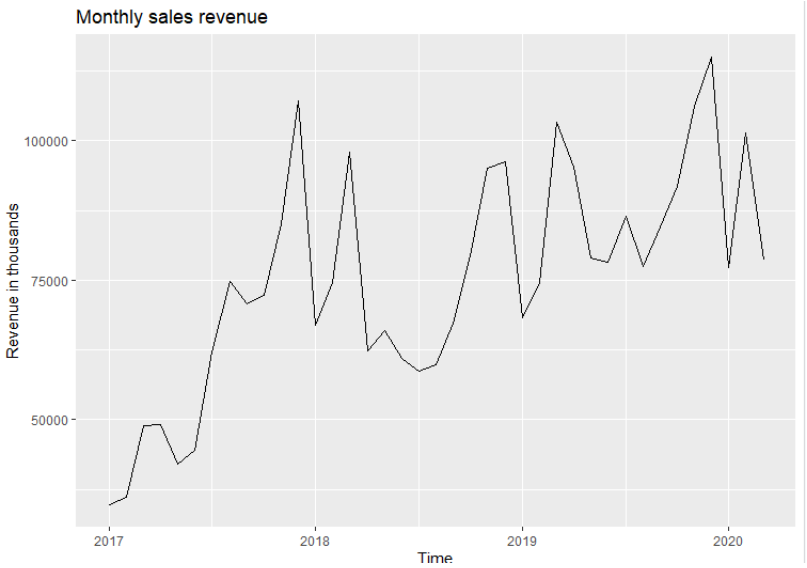
```
#removing last month from data frame, which is incomplete
data <- head(data, -1)

#dividing revenue by 1000 for tracking convinience in plots
data[,3] <- round(data[,3]/1000, digits = 0)

#declare data as time series data
tdata <- ts(data[,3], start = c(2017,1), frequency = 12)
```

4. Exploratory analysis

Our time series looks like this:

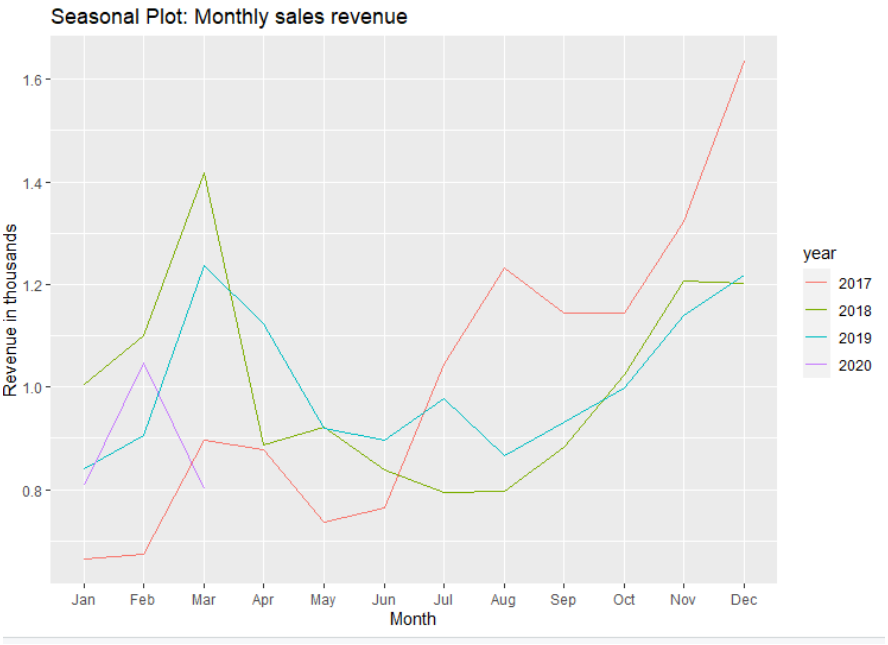


From graph, we can fill there is good positive trend. There are also variations in our values, but in this analysis, we will not focus on them.

To inspect seasonality we need to remove trend from series.
There are two most popular ways:

1. For additive trend - Finding first difference and
2. For multiplicative trend - Dividing time series with trend function. We will use this one.

Then we can use **ggseasonplot** function on transformed series.



We call see that in months of December, in mid-March we have strong seasonality and weak in July-August.
We need to be concern of this and use appropriate models, but I will show all models, regards of these components.

3. Splitting data

I choose proportion of 85:15 because of lack of data records. For each data partition, we need to recreate time series copy.

```
ttest <- tail(tdata, 6)
ttrain <- head(tdata, 33)
```

4. Model evaluation

Some of the most popular evaluation metrics are:

1. **Residual Standard Deviation** of training data **(SD res, or √sigma^2)** – lower values are good sign, but not enough
2. **Mean Absolute Error (MAE)** of test data – absolute prediction error, lower is better.

3. **Root Mean Squared Error (RMSE)** of test data– prediction square deviation error, lower is better.

Indicator number 2. and 3. are crucial for model grading.

Also, for visualization, we will be using these evaluation charts:

1. **Graphical presentation of predicted vs real value**
2. **For ARIMA models, Correlogram, also known as ACF plot - ACF** is an auto-correlation function which gives us values of auto-correlation of any series with its lagged values. In simple terms, it describes how well the present value of the series is related with its past values.

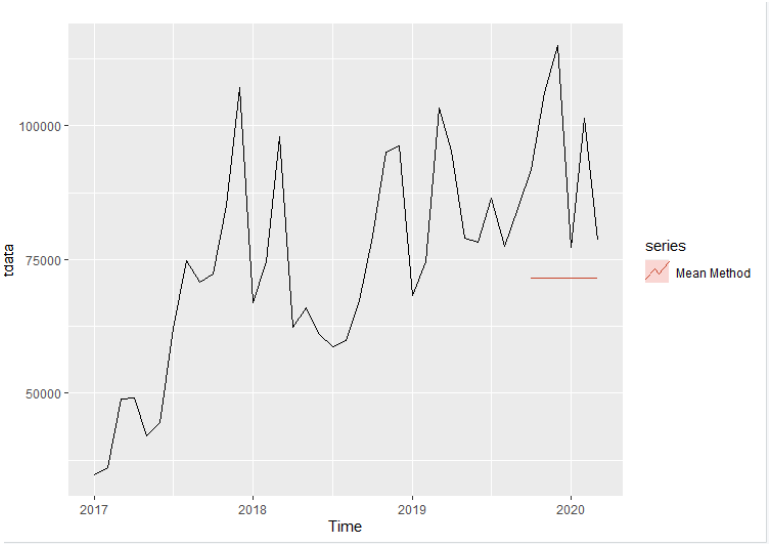
5. Modelling – simple techniques

Simple forecasting techniques are used as benchmarks. They provide a general understanding of historical data and to build intuition upon which to add additional layers complexity and are good to test basic nature about time series.
Several such techniques are common in literature such as: **mean model, naive forecast - random walk, drift method and seasonal naïve.**

7.1 Mean model

A mean model takes the mean of previous observations and uses that for forecasting.
Formula: $\mathbf{y_t = mean(y_{t-x})}$ x- observation of model for whole period
Model: **meanf <- meanf(ttrain, h=6)** h- period for prediction

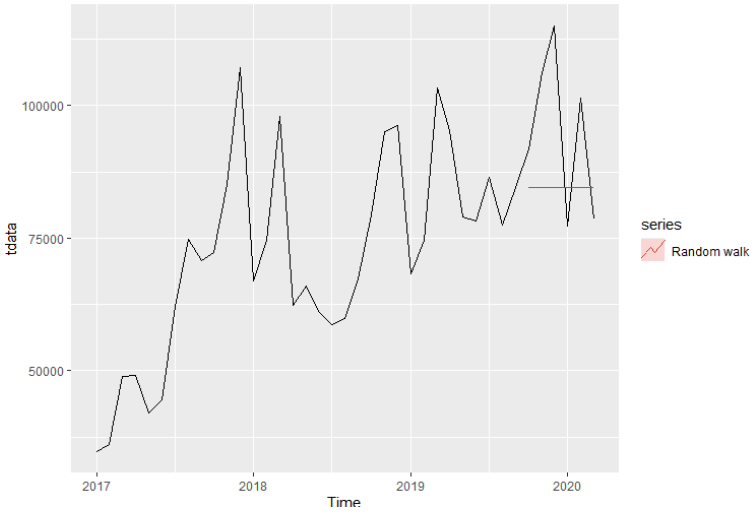
SD RES	MAE	RMSE
18876.63	23513.24	27318.80



7.2 Naive method – Random walk

A random walk, on the other hand, would predict the next value, $\hat{Y}(t)$, that equals he average of previous values.
Formula: y_t = mean(y_{t-1})
Model: **rwf <- rwf(ttrain, h=6)** or **naive(ttrain, h=6)** h- period for prediction

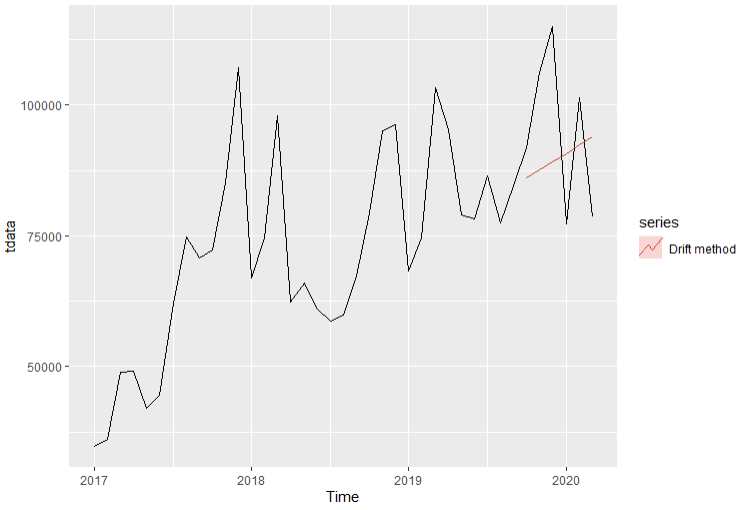
SD RES	MAE	RMSE
15495.68	14908.00	17422.21



7.3 Drift method

Equivalent to extrapolating a line drawn between first and last observations. Forecast equal to last value plus average change. It is first dynamic model.
Formula: **y_t = y_{t-1} + mean(rest data)**
Model: **rwd <- rwf(ttrain, h=6, drift=TRUE)** h- period for prediction

SD RES	MAE	RMSE
15495.68	14648.55	16021.92



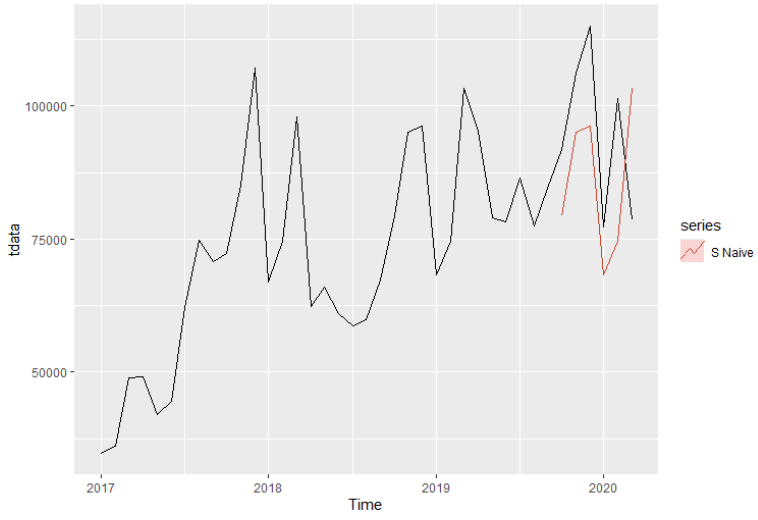
7.4 Seasonal naïve method

Forecasts are equal to last value from same season. Good for isolating and testing seasonal effects on time series.

Formula: $y_t = y_{t-s}$

Model: **snaive <- snaive(ttrain, h=6)** h- period for prediction

SD RES	MAE	RMSE
16453.78	17185.17	18487.01



6. Exponential smoothing models – ETS

Exponential smoothing is a popular forecasting method for short-term predictions. Such forecasts of future values are based on past data whereby the most recent observations are weighted more than less recent observations. As part of this weighting, constants are being ***smoothed***. Exponential smoothing introduces the idea of building a forecasted value as the average figure from differently weighted data points for the average calculation.

There are different exponential smoothing methods that differ from each other in the components of the time series that are modeled.

8.1 Simple ETS method

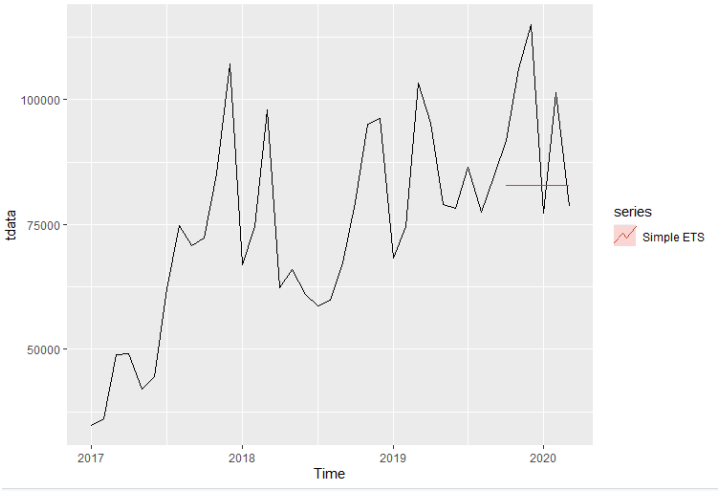
Simple exponential smoothing assumes that the time series data has only a level and some error (or remainder) but no trend or seasonality. It uses only one smoothing constant. The smoothing parameter α determines the distribution of weights of past observations and with that how heavily a given time period is factored into the forecasted value. If the smoothing parameter is close to 1, recent observations carry more weight and if the smoothing parameter is closer to 0, weights of older and recent observations are more balanced.

Formula: $\mathbf{F_t = \alpha y_{t-1} + (1-\alpha)F_{t-1}}$ where

F_t, F_{t-1} - forecast values for time t, t-1; y_{t-1} actual value for time t-1; α - smoothing constant

Model: **sets <- ses(ttrain, h=6)** h- period for prediction, α- here is estimated by model itself to be best fitted one

SD RES	MAE	RMSE
14205.04	15507.66	18561.26



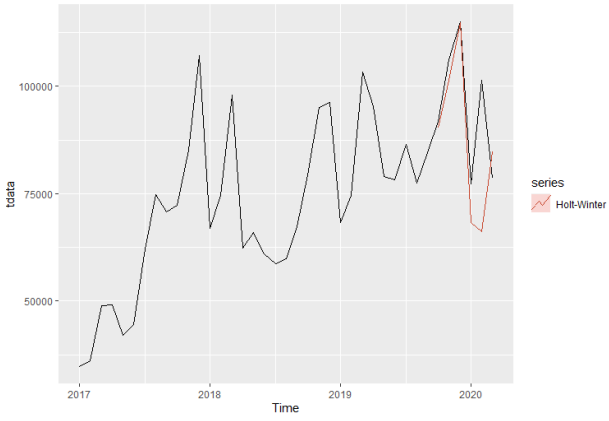
8.2 Holt-Winters exponential smoothing – HW

Holt-Winters exponential smoothing is a time series forecasting approach that takes the overall level, trend and seasonality of the underlying dataset into account for its forecast.

Hence, the Holt-Winters model has three smoothing parameters indicating the exponential decay from most recent to older observations:
α (alpha) for the level component,
β (beta) for the trend component,
γ (gamma) for the seasonality component.
Model: **hw <- hw(ttrain, h = 6, seasonal = "additive")** h- period for prediction

***Note* - Our Holt Winter model demonstrate better result in using additive seasonality instead of multiplicative, even though seasonality visually seems to be multiplicative. This characteristic is well known for short time series, so we will be cautious in using this model – thanks professor Jelena Jovanovic for this excellent observation.**

SD RES	MAE	RMSE
14830.10	9482.57	15179.86



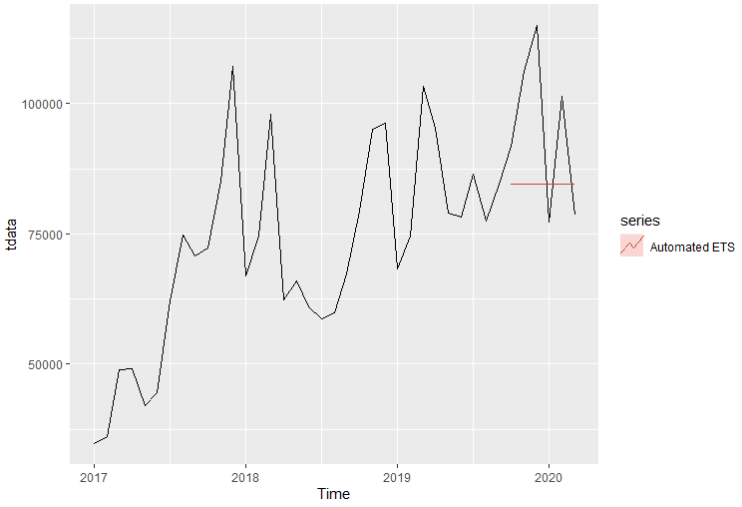
8.3 Automated exponential smoothing

There is also an general automated exponential smoothing forecast. In this case, it is automatically determined whether a time series component is additive or multiplicative.
The ets function takes several arguments including model. The model argument takes three letters as input.
The first letter denotes the error type (A, M or Z),
the second letter denotes the trend type (N, A, M or Z),
and the third letter denotes the seasonality type (N, A, M or Z).
The letters stand for N = none, A = additive, M = multiplicative and Z = automatically selected.

Model: **aets <- ets(ttrain)**
Note(thank's to my professor)* In this particular example, if we use aets function, it will be suggesting us **ETS (M,N,N)**, which is obviously not ideal model, considering the observed trend in data. In this case we must add model parameter in ets function like this **(ttrain, model = ("AAA"))**

I will use these default values, only because my previous HW model is practically same as **ETS (A,A,A)**, but be concerned if you are using only this function.

SD RES	MAE	RMSE
0.1913	14908.23	17422.64



7. ARIMA family models

Auto Regressive Integrated Moving Average (ARIMA) is arguably the most popular and widely used statistical forecasting technique. As the name suggests, this family of techniques has 3 components:
a) an "**autoregression**" component that models the relationship between the series and it's lagged observations;
b) a "**moving average**" model that models the forecast as a function of lagged forecast errors; and
c) an "**integrated**" component that makes the series stationary.

The model takes in the following parameter values:

p that defines the number of lags;

d that specifies the number of differences used; and

q that defines the size of moving average window

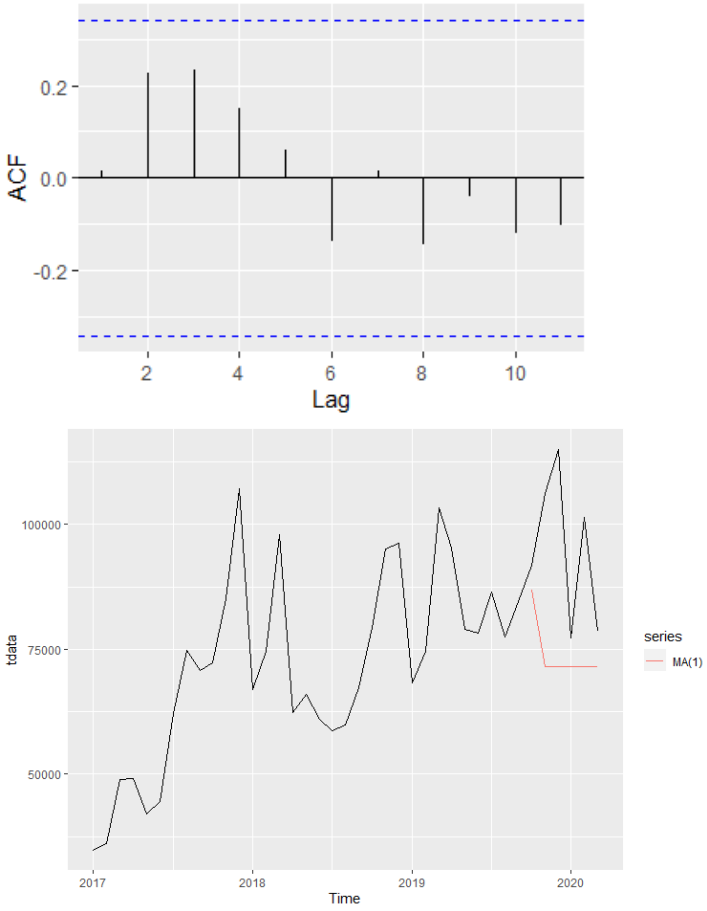
9.1 Moving average model

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

The moving-average model is a special case and key component of the more general ARMA and ARIMA (0, 0, q) models of time series, which have a more complicated stochastic structure.
Model: **MA(q): $X_t = \mu + \epsilon_t + \epsilon_{t-1}\theta_1 + \ldots + \epsilon_{t-q}\theta_q$** - where **$\mu$** is the mean of the series, the $\vartheta_1, \ldots, \vartheta_q$ are the parameters of the model and the $\varepsilon_t, \varepsilon_{t-1}, \ldots, \varepsilon_{t-q}$ are white noise error terms. The value of q is called the order of the MA model. **I will demonstrate use of simplest MA (1) model, and later I will add more complexity.**

SD RES	MAE	RMSE
12806.54	20977.35	26129.44

ACF function is inside dashed lines, there are no statistically significant autocorrelation, which is positive sign



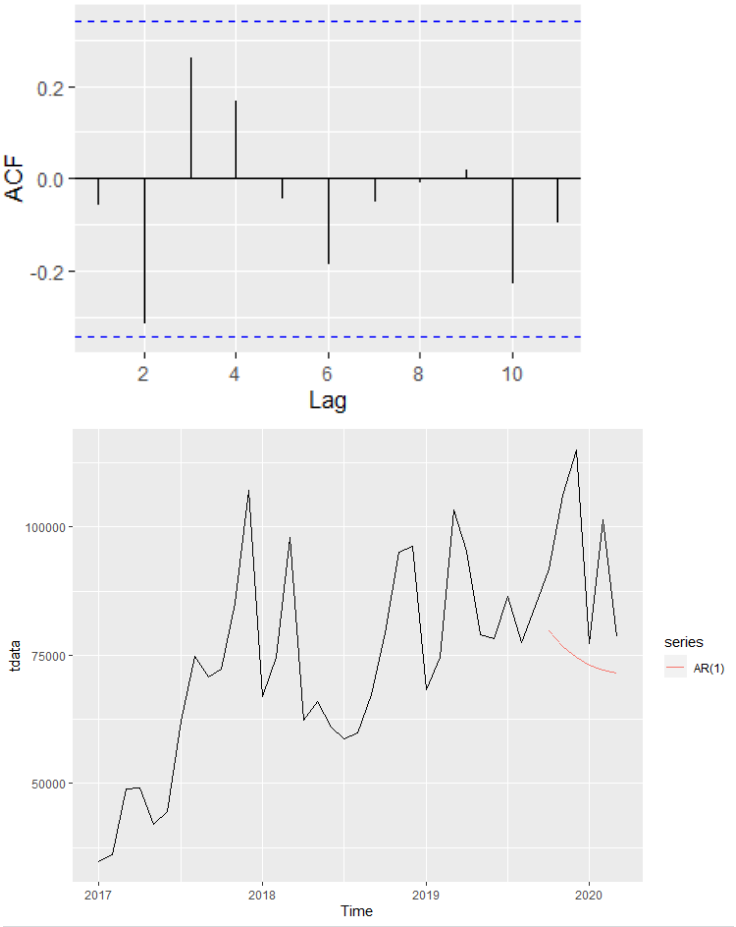
9.2 Autoregressive model

The autoregressive model specifies that the output variable depends linearly on its own previous values and on a stochastic element. Together with the moving-average (MA) model, it is a special case and key component of the more general autoregressive–moving-average (ARMA) and autoregressive integrated moving average (ARIMA(p,0,0)) models of time series. It is also a special case of the vector autoregressive model (VAR), which consists of a system of more than one interlocking stochastic difference equation in more than one evolving random variable.

Model: AR(p): $Y_t = \beta_0 + \beta_1 y_{t-1} + \ldots + \beta_p y_{t-p} + \epsilon_t$ - where β_0 is constant, the β_1, ..., β_p are the parameters of the model and the ε_t is white noise. The value of p is called the order of the AR model.
I will demonstrate use of simplest AR (1) model, and later I will add more complexity.

SD RES	MAE	RMSE
14152.43	20378.13	24385.42

ACF function is inside dashed lines, which is positive sign.

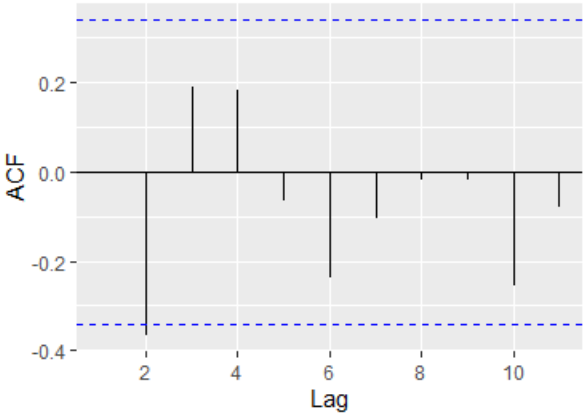


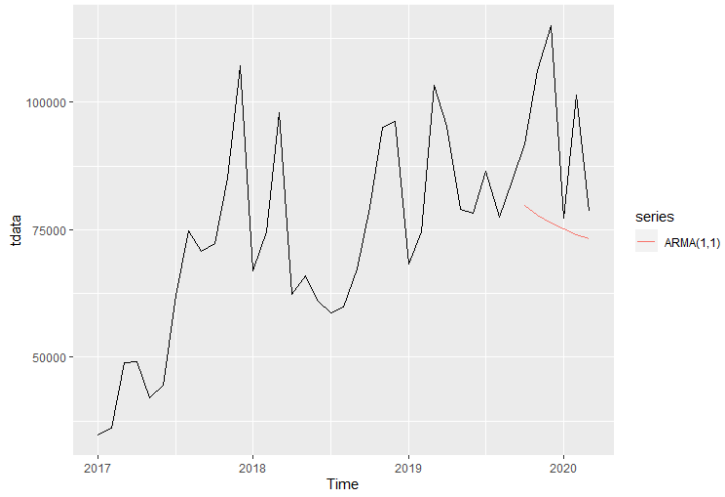
9.3 Autoregressive–moving-average model - ARMA

The ARMA model consist of two parts. The AR part involves regressing the variable on its own lagged (i.e., past) values. The MA part involves modeling the error term as a linear combination of error terms occurring contemporaneously and at various times in the past.
Model: ARMA(p, q): $Y_t = \mu + \beta_0 + \beta_1 y_{t-1} + \ldots + \beta_p y_{t-p} + \epsilon_t + \epsilon_{t-1}\,\theta_1 + \ldots + \epsilon_{t-q}\,\theta_q$
I will demonstrate use of simplest ARMA (1,1) model, which is composition of AR (1) and MA (1), to build, usually better, hybrid model. This is not final model.
I will add more model complexity and also providing reasons for choosing particular parameters values.

SD RES	MAE	RMSE
14078.28	18957.66	23145.57

ACF function is not fully inside dashed lines, which is sign there are probably better models





9.4 Autoregressive integrated moving average- ARIMA

ARIMA models are applied in some cases where data show evidence of non-stationarity, where an initial differencing step (corresponding to the "integrated" part of the model) can be applied one or more times to eliminate the non-stationarity.
The I (for "integrated") indicates that the data values have been replaced with the difference between their values and the previous values (and this differencing process may have been performed more than once). The purpose of each of these features is to make the model fit the data as well as possible. There are two types of models:

1. Non-seasonal ARIMA models are generally denoted ARIMA(p,d,q) where parameters p, d, and q are non-negative integers, p is the order (number of time lags) of the autoregressive model, d is the degree of differencing (the number of times the data have had past values subtracted), and q is the order of the moving-average model;

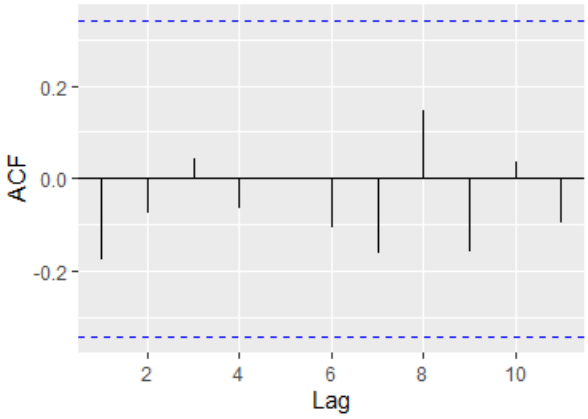
2. Seasonal ARIMA models are usually denoted ARIMA(p,d,q)(P,D,Q)m, where m refers to the number of periods in each season, and the uppercase P,D,Q refer to the autoregressive, differencing, and moving average terms for the seasonal part of the ARIMA model.

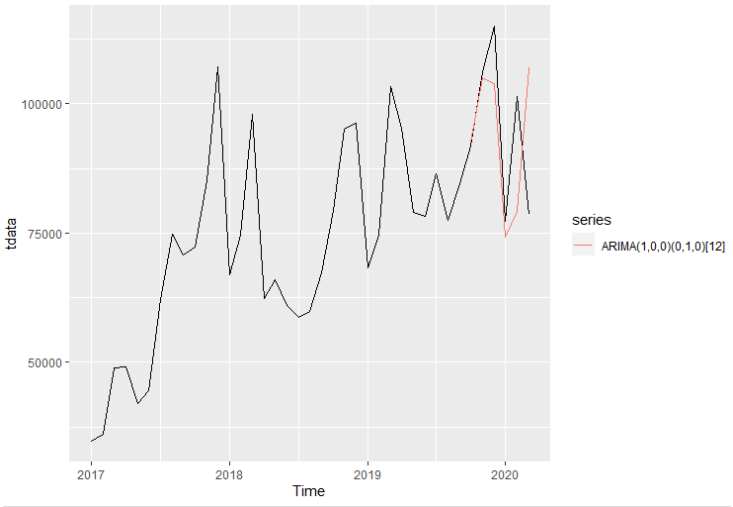
9.4.1 ARIMA (1,0,0)(0,1,0)[12]

I used auto.arima function from forecast package, which is good function for exploring various types of fitted ARIMA models for particular dataset. It will try all possible combinations of ARIMA models by changing parameter values. We will start by first model, that was suggested, seasonal ARIMA.

SD RES	MAE	RMSE
13981.54	11087.77	15449.20

ACF function is inside dashed lines, which is positive sign.



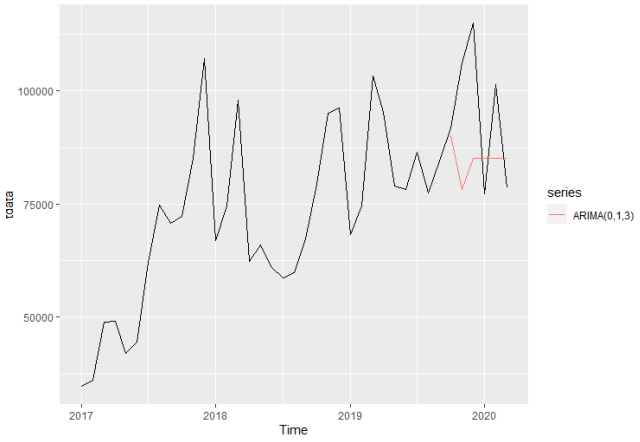


9.4.2 ARIMA (0,1,3)

Last model was suggested with few error messages. That errors pointed that chosen seasonal unit root test encountered an error when testing for the second difference (insufficient number of data after 1st difference). By default, it used one seasonal difference.

We will consider using a different seasonal unit root test (for example Canova and Hansen (CH) test statistic)

SD RES	MAE	RMSE
12806.54	15033.53	18463.70



8. Machine learning models

In this last few sections i presented few most commonly used statistical models for forecasting univariate time series data. There are also machine learning models.
This problem topic is relatively new for machine learning algorithms, but recently they are gaining in popularity in almost every field aspect, and they are very good competitors to our traditional models.
I will demonstrate two types of models.

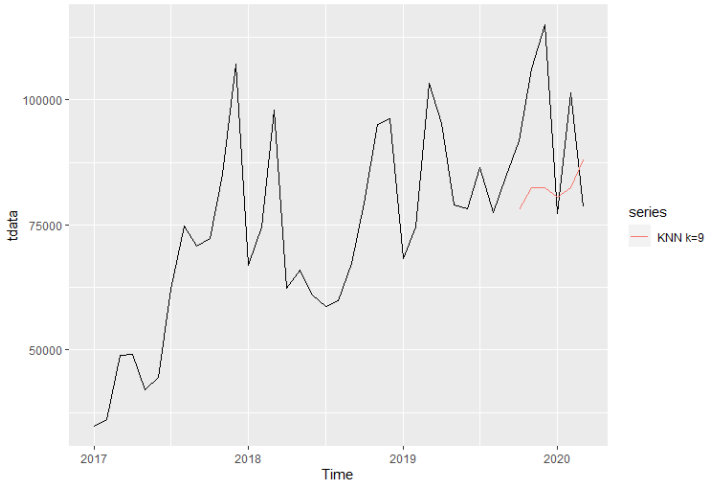
10.1 KNN model

KNN is a very popular algorithm used in classification and regression. Given a new example, KNN finds its k most similar examples, called nearest neighbors, according to a distance metric such as the Euclidean distance, and predicts its value as an aggregation of the target values associated with its nearest neighbors.
In this project I will use KNN regression.
Model: **knn <- knn_forecasting(ttrain, h = 6, lags = 1:12, k = c(9), cf = c("mean"))** where:
h - prediction horizon;
lags - determine the lagged values used as features or autoregressive explanatory variables. A good choice for the lags used as features would be 1:12, when we are working with monthly data. Also, after testing with different time frames, 12 was the best fit;
k - the number of nearest neighbors used in the prediction. Unfortunately, cross validation tests are not implemented yet in time series analysis.
cf - combination function used to aggregate the targets-default is mean.

Using manual comparing tests, optimal result is for k=9.

	SD Residuals	MAE	RMSE
KNN9	0	16924.30	19443.02
KNN2	0	19408.11	21499.52
KNN3	0	21118.04	24113.34
KNN4	0	24117.50	26686.69
KNN5	0	20140.59	22376.14
KNN6	0	20873.07	23273.85
KNN7	0	20995.55	23658.19
KNN8	0	19207.46	21286.83
KNN23	0	22577.10	25338.70
KNN34	0	17996.15	20261.13
KNN45	0	22565.46	24843.77
KNN56	0	20466.41	22917.19
KNN67	0	19006.81	21106.26
KNN78	0	18803.69	20911.16
KNN234	0	22055.88	24542.85
KNN345	0	21567.94	24566.54
KNN456	0	19762.66	21937.25
KNN567	0	21013.42	23052.47
KNN678	0	18501.48	20659.36

MAE	RMSE
16924.29	19443.02



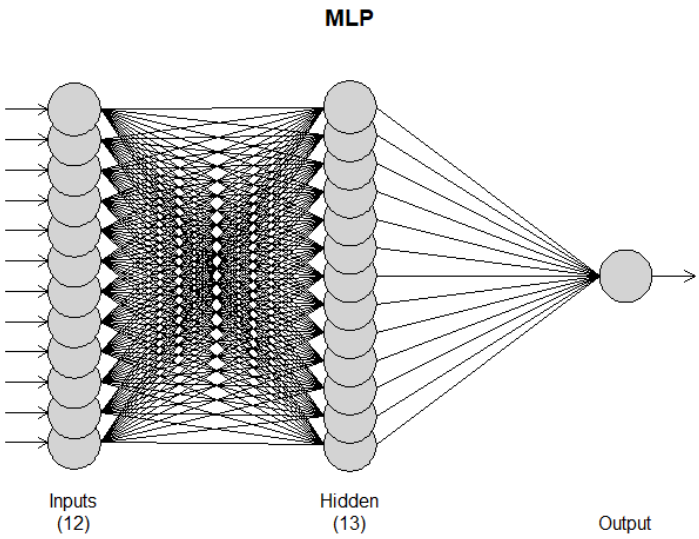
10.2 Neural networks - Multilayer Perceptron's (MLP)

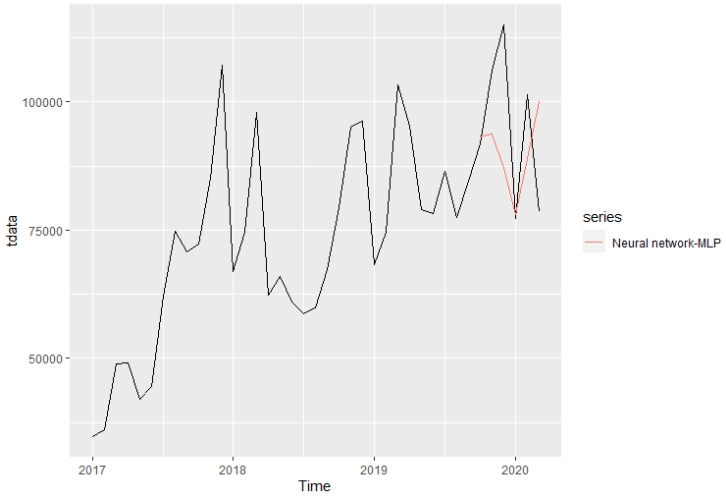
I chose **nnfor** package, because it differs from existing neural network implementations for R, in that it provides code to automatically design networks with reasonable forecasting performance, and also provide in-depth control to the experienced user. This increases the robustness of the resulting networks, but also helps reduce the training time.
Note that nnfor package relay on neuralnet library, and since neuralnet cannot tap on GPU processing, large networks tend to be very slow to train. Our data is not long, so we can create and train more networks.

Model: mlp <- mlp(ttrain, lags=1:12, sel.lag=FALSE, reps = 10000) where:
lags –allows you to select the autoregressive lags considered by the network. If this is not

provided then the network uses 1 : frequency(series);
reps - defines how many training repetitions are used. The default reps=20 is a compromise between training speed and performance, but the more repetitions you can afford the better. They help not only in the performance of the model, but also in the stability of the results, when the network is retrained. **I found using 10000 samples are quite good estimator, but have patience in mind;**
comb - How the different training repetitions are combined is controlled by the argument comb that accepts the options median, mean, and mode. **Default is median, and that is in my model;**
hd - defines a fixed number of hidden nodes. If it is a single number, then the neurons are arranged in a single hidden layer. If it is a vector, then these are arranged in multiple layers. **By default, it will chose optimal number, which in my case is 13 in one layer(For univariate time series, multiple layers are rarely used);**
difforder - it is useful to remove the trend from a time series prior to modelling it. This is handled by the argument difforder. If difforder=0 no differencing is performed. For diff=1, level differences are performed. Similarly, if difforder=12 then 12th order differences are performed. If the time series is seasonal with seasonal period 12, this would then be seasonal differences. You can do both with difforder=c(1,12) or any other set of difference orders. **In my case, difforder=NULL then the code decides automatically – chose D1 – first difference.**
In testing and modifying parameters of function, this model got me best results.

MAE	RMSE
~12626.29	~16259.02





9. Conclusion - choosing best fitting model for forecasting

	SD Residuals	MAE	RMSE
Mean	18876.6325	23513.242	27318.80
Random walk-Naive	15495.6773	14908.000	17422.21
Drift	15495.6773	14648.547	16021.92
S Naive	16453.7837	17185.167	18487.01
Simple ETS	14205.0381	15507.655	18561.26
HW	14830.1032	9482.577	15179.86
Automated ETS	0.1913	14908.236	17422.64
AR(1)	14152.4326	20378.126	24385.42
MA(1)	12806.5380	20977.351	26129.44
ARMA(1,1)	14078.2819	18957.655	23145.57
ARIMA(1,0,0)(0,1,0)[12]	13981.5383	11087.773	15449.20
ARIMA(0,1,3)	12806.5380	15033.525	18463.70
KNN k=9	0.0000	16924.296	19443.02
Neural networks-MLP	0.0000	12746.440	16113.69

>

First four benchmark models, are good to test certain things about time series, but are not used as final models. AR and MA models are good when there are non-seasonal effects.

HW is clear winner here, and there are two main reasons (against ARIMA and ML models):

1. **Short time frame** – 39 observation only in total;
2. **Existence of structural fracture movements** - (cause is Corona virus). Structural fracture is a series of observations that is inconsistent with the previous time series.
   Corona virus has influenced on bringing several government decisions (interventions), that begun from middle of March 2020., such as:
   a. **Some import products could not be imported into our country;**
   b. **Sales value got decreased because sales field people work from home, and, thus, their efficiency decline down**

In more general occasions, ARIMA models are preferred. They are superior models, and are widely used in practice.
Also, there are many more complex model structures that are not part of this research project, and for

more in depth problem-oriented area (VAR, GARCH, LSTM…). It would be pleasure for me to continue researching, working and learning them 😊.