Exponential Smoothing

1. Characteristics of time series data

1.1. Time series components

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.

The trend component can also sometimes additionally reflect a cycle.

- A cyclic pattern is usually longer than a trend.
- In other words, a trend can be a pattern within a longer cycle. Hence, the trend component is sometimes also called "trend-cycle" component.

Seasonality:

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.
- It is also possible to have multiple seasonalities.

Error or Remainder:

Error or remainder is the uncategorized component that is part of time series data and is not described by trend and seasonality.

1.2 Time series decomposition

Time series decomposition describes the process of decomposing time series data into its components: (1) Trend, (2) Seasonality and (3) Error.

- It is used to better understand data and builds a basis for some time series forecasting approaches.

The data we will use in this blog post is real-world organic traffic data.

- Before we get to the time series decomposition, we will start with some required prework and plot the data for exploration.
- The first column displays the month of the year and the second column displays the volume of website traffic from organic search (e.g. Google) measured in Google Analytics sessions.

```
data <- read.csv("organic-traffic.csv", header = T)
head(data)
```

```
##    Month Organic.Sessions
## 1 1/1/14           144217
## 2 2/1/14           156374
## 3 3/1/14           176416
## 4 4/1/14           182978
```

```
## 5 5/1/14           174032
## 6 6/1/14           174129
```

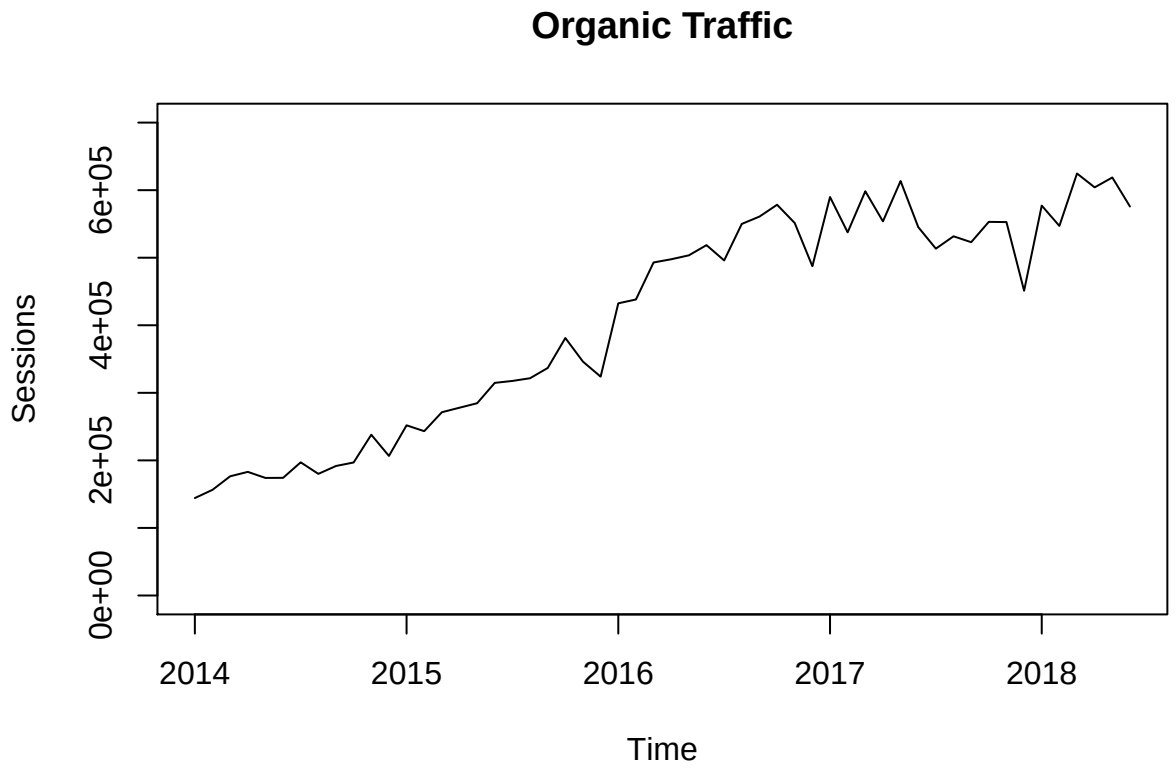
The following R code converts the `Month` column in the appropriate format as well as the `Organic Sessions` column into a time series object of monthly data (`frequency = 12`) and plots the data as a line graph.

```r
#Convert "factor" to "Date" according to cell formatting in CSV file
data[,1] <- as.Date(data[,1], format = "%m/%d/%y")

#Convert vector of "Organic Sessions" into time series object
Organic_Traffic <- ts(data[,2], start = c(2014,1), end = c(2018,6), frequency = 12)

#Avoid scientific notation for y-axis values
#options(scipen=999)

#Plot "Organic Traffic" data
plot(Organic_Traffic, main = "Organic Traffic", ylab = "Sessions", ylim = c(0, 700000))
```



The first step for the time series decomposition is to determine whether the underlying seasonality in the time series data is additive or multiplicative.

- Additive seasonality means that the magnitude of the seasonal amplitude swings remains approximately the same throughout independently of the level of the time series.
- Multiplicative seasonality means that the magnitude of the seasonal amplitude changes in proportion to the level of the time series.

The difference can be observed from the two plots in the following image.

- Also, note that the two time series [...] also have an additive and multiplicative trend, which may appear as the more visible feature at first glance.

In the case of the "Organic_Traffic" data, the magnitude of seasonality increases proportionally with level suggesting multiplicative seasonality as can be seen in the following image.

Knowing that the seasonality of the "Organic_Traffic" data is multiplicative, we can proceed to the actual time series decomposition.

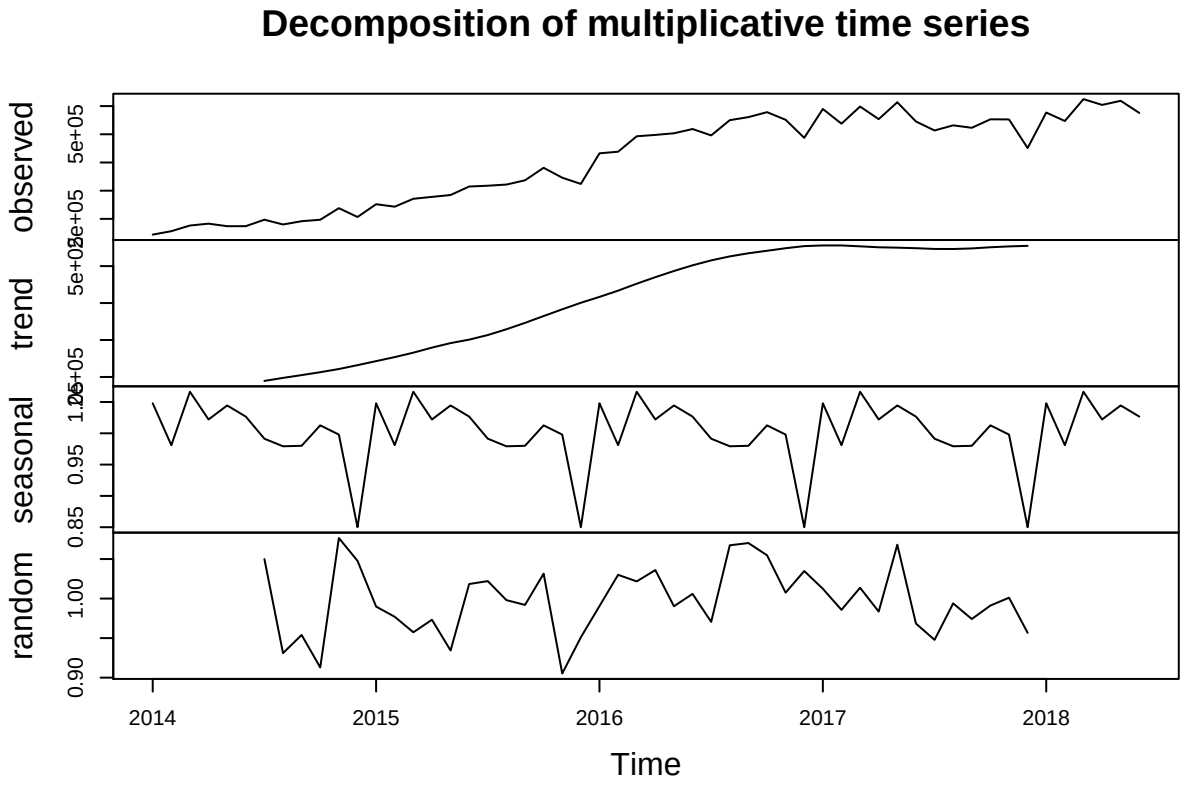
- There are various different methodologies of time series decomposition including classical, X11, SEATS and STL that come with varied advantages and disadvantages as well as requirements for the time unit of the data.
- In this case, we will use the classical time series decomposition as it is widely used while STL (Seasonal Decomposition of Time Series by Loess) is more sophisticated.
  - It is executed in R by decompose requiring "additive" or "multiplicative" as input for the type argument, which refers to the seasonal component in the time series.

```
#Decompose data into seasonal, trend and irregular components using classical decomposition
fit_decompose <- decompose(Organic_Traffic, type = "multiplicative")
#Print component data of decomposition
fit_decompose
```

```
## $x
##         Jan    Feb    Mar    Apr    May    Jun    Jul    Aug    Sep    Oct
## 2014 144217 156374 176416 182978 174032 174129 197103 180033 191700 196871
## 2015 251927 243225 271287 277962 284606 314751 317597 321638 336796 381121
## 2016 432491 438148 492964 497714 503518 518628 495984 550032 560869 578403
## 2017 589791 537567 598358 553789 613503 545310 513433 531702 522966 553064
## 2018 577146 547037 624663 604150 618746 575822
##         Nov    Dec
## 2014 237982 206609
## 2015 346087 323950
## 2016 551371 487392
## 2017 552756 451144
## 2018
##
## $seasonal
##            Jan       Feb       Mar       Apr       May       Jun       Jul
## 2014 1.0479508 0.9810111 1.0663775 1.0220808 1.0444953 1.0267025 0.9913709
## 2015 1.0479508 0.9810111 1.0663775 1.0220808 1.0444953 1.0267025 0.9913709
## 2016 1.0479508 0.9810111 1.0663775 1.0220808 1.0444953 1.0267025 0.9913709
## 2017 1.0479508 0.9810111 1.0663775 1.0220808 1.0444953 1.0267025 0.9913709
## 2018 1.0479508 0.9810111 1.0663775 1.0220808 1.0444953 1.0267025
##            Aug       Sep       Oct       Nov       Dec
## 2014 0.9792796 0.9800322 1.0127299 0.9979517 0.8500177
## 2015 0.9792796 0.9800322 1.0127299 0.9979517 0.8500177
## 2016 0.9792796 0.9800322 1.0127299 0.9979517 0.8500177
## 2017 0.9792796 0.9800322 1.0127299 0.9979517 0.8500177
## 2018
##
## $trend
##           Jan      Feb      Mar      Apr      May      Jun      Jul
## 2014       NA       NA       NA       NA       NA       NA 189358.2
## 2015 242858.6 253779.4 265725.2 279448.0 291629.5 301023.0 313435.7
## 2016 416653.8 433603.0 452455.8 470012.2 486785.8 502149.4 515513.7
```

```
## 2017 555924.5 555887.7 553544.7 550909.6 549911.5 548458.9 546421.7
## 2018       NA       NA       NA       NA       NA       NA
##           Aug      Sep      Oct      Nov      Dec
## 2014 197465.0 205036.7 212947.3 221512.2 231978.7
## 2015 329081.0 346439.4 364832.3 383109.9 400726.1
## 2016 526210.3 534744.2 541472.0 548391.2 554085.7
## 2017 546289.4 547780.0 550974.5 553291.3 554781.1
## 2018
##
## $random
##            Jan       Feb       Mar       Apr       May       Jun       Jul
## 2014        NA        NA        NA        NA        NA        NA 1.0499601
## 2015 0.9898751 0.9769627 0.9573818 0.9731935 0.9343427 1.0184103 1.0220960
## 2016 0.9905146 1.0300413 1.0217110 1.0360613 0.9903087 1.0059546 0.9704905
## 2017 1.0123752 0.9857609 1.0136719 0.9835099 1.0681134 0.9683999 0.9478064
## 2018        NA        NA        NA        NA        NA        NA
##            Aug       Sep       Oct       Nov       Dec
## 2014 0.9310122 0.9540039 0.9128846 1.0765565 1.0477871
## 2015 0.9980626 0.9919718 1.0315161 0.9052163 0.9510478
## 2016 1.0673870 1.0702249 1.0547775 1.0074973 1.0348407
## 2017 0.9938911 0.9741524 0.9911749 1.0010830 0.9566775
## 2018
##
## $figure
##  [1] 1.0479508 0.9810111 1.0663775 1.0220808 1.0444953 1.0267025 0.9913709
##  [8] 0.9792796 0.9800322 1.0127299 0.9979517 0.8500177
##
## $type
## [1] "multiplicative"
##
## attr(,"class")
## [1] "decomposed.ts"
```

```
#Plot component data of decomposition
plot(fit_decompose)
```



Since the applied time series decomposition was multiplicative, the individual values for a month can be multiplied to yield the month-respective organic sessions count.

- The mathematical formula is: $y_t = S_t \cdot T_t \cdot R_t$.
  - S_t: seasonality component;
  - T_t: trend component;
  - R_t: remainder (or random).

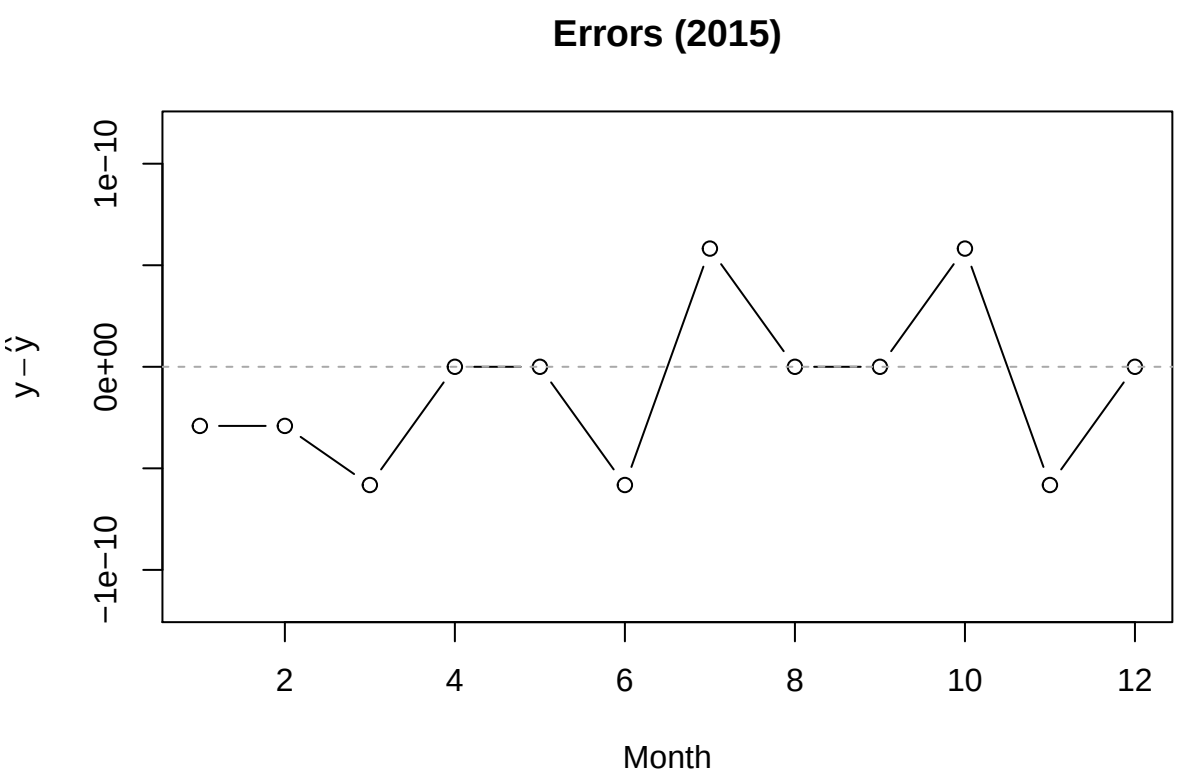
See below the differences between the real values (y_t) and the values reconstruced from the components ($\hat{y}_t = S_t \cdot T_t \cdot R_t$), for the 2015 data:

```
seasonal.2015 <- fit_decompose$seasonal[13:24]
trend.2015 <- fit_decompose$trend[13:24]
random.2015 <- fit_decompose$random[13:24]

reconst.2015 <- seasonal.2015 * trend.2015 * random.2015
original.2015 <- fit_decompose$x[13:24]
errors.2015 <- original.2015 - reconst.2015

ymax = max(abs(c(min(errors.2015), max(errors.2015))))

#options(scipen=-3)
plot(errors.2015,
     ylim = c(-2*ymax, 2*ymax), type = "b", col = "black",
     main = "Errors (2015)", xlab = "Month", ylab = expression(y - hat(y)))
abline(a = 0, b = 0, col="darkgray", lty=2)
```



More info on `plotmath` (mathematical annotations on plots) can be found here.

The steps of the `decompose` command that have led to those calculated values are as follows:

1. Calculate the trend with a centered moving average:

The moving average of order 3, for instance, calculates the average value of the 3 nearest neighboring values of the time series.

- If the moving average is centered, it means that the moving average is only calculated, if the neighboring values are organized symmetrically around the given observation of the time series.
  - In the case of order 3, this means that there is no calculated moving average for the first observation but for the second observation as the latter is centered between the first and third observation.
  - For order 5, the first calculated data point of the centered moving average will only exist for the third observation, which is centered among observations 1 to 5.
- The centering of the moving average causes it to not calculate data points for the beginning and ending of the underlying time series.
  - In our case, given that the first calculated data point of the trend line is in July 2014, 7th observation of the time series, we can conclude that decompose used an order of 13 (6 observations to the left of the centered moving average data point and 6 observations to its right).

2. Remove the trend from the time series:

In the second step, the trend line (based on the centered moving average with order 13 in this case) is removed from the time series.

- This is also called detrending.

- The remaining components in the time series are therefore seasonality and remainder.

3. Calculate the average for each time unit over all time periods:

With the trend removed from the time series, the average for each time unit over all time periods is taken.

- For the monthly "Organic_Traffic" data, this means that the average is calculated for all January observations, February observations and so on over all time periods.
  - All time periods are 4.5 years in this case, which equates 5 observations for January to June and 4 observations for July to December.
  - These average values for each time period are then centered to have a baseline of the time series level without seasonality.
  - In the case of a multiplicative time series, the effect of seasonality for each time period is expressed by a factor.
    - In the case of a multiplicative time series, the effect of seasonality for each time period is expressed by a factor.
  - You can observe that the values of the seasonal component do repeat every 12 months as the seasonality considered exhibits the same annual effect for every year of the dataset.
    - The calculated effect of the seasonality is therefore sensitive to the time window spanned by the dataset.
    - Longer time spans of data will accordingly yield a better model of the seasonality effect.

You can deseasonalize your traffic using the seasonality factors.

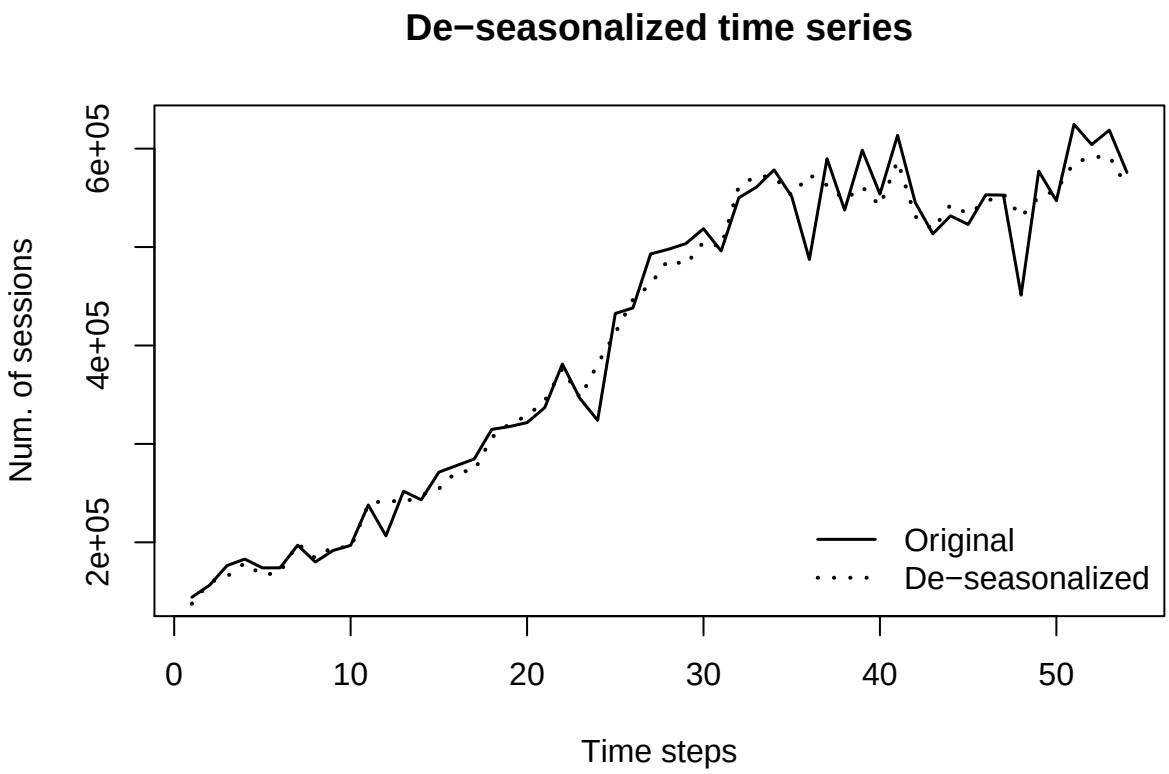
- For December 2014, this looks like this: $\text{Dec}_{2014,\text{deseasonalized}} = \frac{206,609}{0.8500177} = 243,064.4$.
- Accordingly, for November 2014 this looks like this: $\text{Nov}_{2014,\text{deseasonalized}} = \frac{237,982}{0.9979517} = 238,470.5$.
  - If you looked at the "Organic_Traffic" numbers, you will see that December 2014 had $206,609$ organic sessions while November 2014 had $237,982$.
  - So, you would conclude that December must have been an overall weaker month but if you factor out seasonality, December was actually a stronger month than November in terms of number of organic sessions.

```
len <- length(fit_decompose$x)
deseason.all <- fit_decompose$x / fit_decompose$seasonal

plot(1:len, fit_decompose$x,
     main = "De-seasonalized time series", xlab = "Time steps", ylab = "Num. of sessions",
     lty = 1, lwd = 1.5, type="l", col="black")

lines(1:len, deseason.all,
      lty = 3, lwd=2.0, col="black")

legend("bottomright", lty = c(1,3), lwd = c(1.5, 2.0), col = c("black", "black"), bty = "n",
       legend = c("Original", "De-seasonalized"))
```



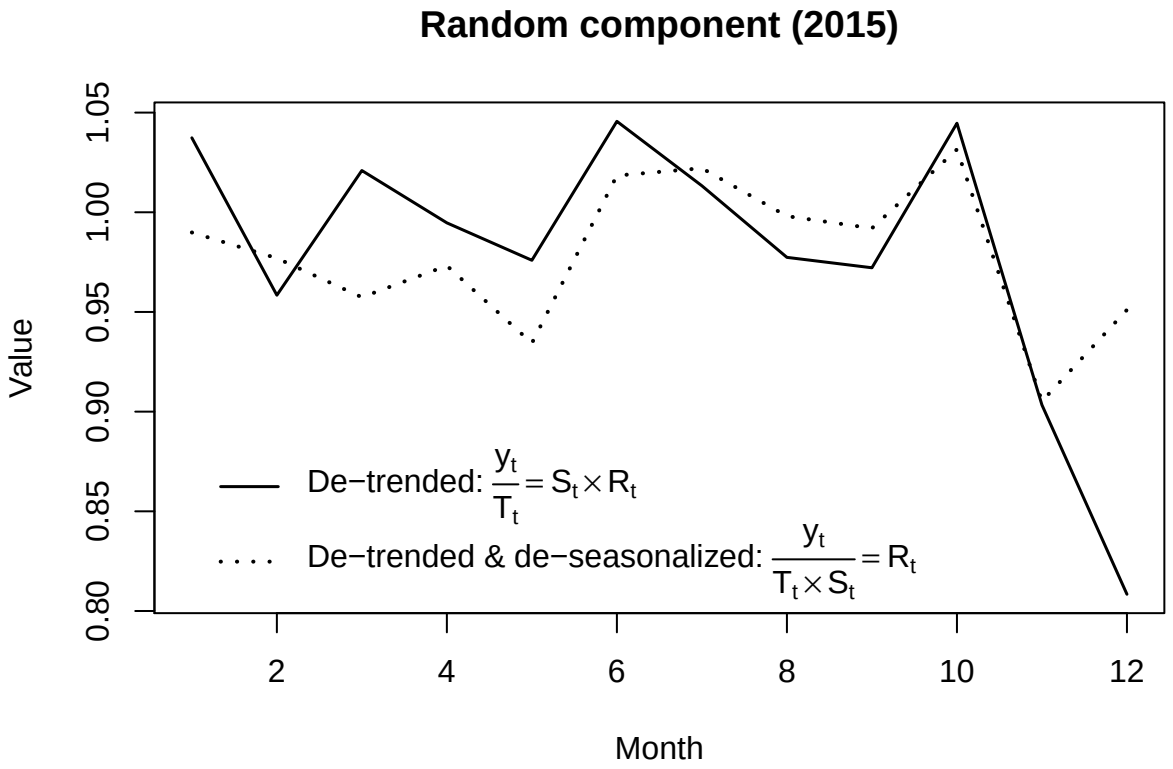
4. Calculate the error component:

In the fourth step, the trend and seasonality are removed from the original time series to yield the error component.

```
plot(1:12, original.2015 / trend.2015,
     main = "Random component (2015)", xlab = "Month", ylab = "Value",
     lty = 1, lwd=1.5, type="l", col="black")

lines(1:12, original.2015 / (trend.2015 * seasonal.2015),
      lty = 3, lwd=2.0, col="black")

legend(x = 1, y = 0.90,
       lty = c(1,3), lwd = c(1.5, 2.0), col = c("black", "black"), bty = "n",
       legend = c(expression("De-trended: " * frac(y[t],T[t]) == S[t] %*% R[t]),
                  expression("De-trended & de-seasonalized: " * frac(y[t],T[t] %*% S[t]) == R[t])))
```



In conclusion, time series decomposition allows you to understand your time series data at a deeper level. For instance, you can quickly identify the underlying trend and take it as the "growth rate" of the organic traffic or you can de-seasonalize data points to identify whether traffic really went down in a given month or whether it was due to seasonality only.

---

2. Exponential smoothing for time series forecasts

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. This is different from the simple moving average method, in which every data point has equal weight in the average calculation.
- 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.

- For instance, simple exponential smoothing (SES) uses only one smoothing constant, double exponential smoothing or Holt exponential smoothing uses two smoothing constants and triple exponential smoothing or Holt-Winters exponential smoothing accordingly uses three smoothing constants.

2.1 Simple exponential smoothing

Simple exponential smoothing assumes that the time series data has only a level and some error (or remainder) but no trend or seasonality.

It is therefore not applicable to the "Organic_Traffic" data as that data has underlying seasonality and trend.

- Nevertheless, for illustration purposes, we will apply simple exponential smoothing to our organic traffic data in this section purely to explore its mechanics.

For exponential smoothing, all past observations are part of the calculation for the forecasted value.

- 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.
  - For example, in the case of $\alpha = 0.5$, the weight halves with every next older observation: 0.5, 0.25, 0.125, 0.0625, etc.
- As you can see in the following image, the forecasted value of the next period in the future is the sum of the respective products of Weight and past Organic Sessions for every previous time period.

Here is also the mathematical notation for columns D and E respectively:

$$\text{Weight}_t = \alpha \cdot (1-\alpha)^t \text{Forecast} = \sum_{t=0}^{n} \Big(\text{Organic sessions}_t \cdot \text{Weight}_t \Big)$$

In this case, our freely chosen smoothing parameter $\alpha = 0.5$ yields a forecast of $591,069$ Organic Sessions for the next future period.

If we fit a simple exponential smoothing model to the "Organic_Traffic" data in R, then the smoothing parameter α is automatically determined by optimizing for minimum forecasting error.

- The following R code fits a simple exponential smoothing model to the "Organic_Traffic" data and prints a summary including the calculated smoothing parameter α and forecasting error measures.
- Note that instead of the `ses(Organic_Traffic)` function, you can also use `ets(Organic_Traffic, model = "ANN")` from the `forecast` package but in this section we go with the simpler `ses` function.

```
#Install and load forecast package if not installed
if(!require(forecast)) {
  install.packages("forecast", dependencies = T, repos = "http://cran.us.r-project.org")
  library(forecast) }

#Fit simple exponential smoothing model to data and show summary
fit_ses <- ses(Organic_Traffic)
summary(fit_ses)
```

```
##
## Forecast method: Simple exponential smoothing
##
## Model Information:
## Simple exponential smoothing
##
## Call:
##  ses(y = Organic_Traffic)
##
##   Smoothing parameters:
##     alpha = 0.6347
```

```
##
##   Initial states:
##     l = 151553.5283
##
##   sigma:  40114
##
##      AIC     AICc      BIC
## 1364.111 1364.591 1370.078
##
## Error measures:
##                     ME     RMSE      MAE      MPE     MAPE      MASE
## Training set 12767.87 39364.14 30369.84 3.353772 7.486932 0.2705201
##                     ACF1
## Training set -0.2443821
##
## Forecasts:
##          Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
## Jul 2018       589153.8 537745.6 640561.9 510531.8 667775.8
## Aug 2018       589153.8 528265.2 650042.4 496032.7 682274.8
## Sep 2018       589153.8 520073.8 658233.7 483505.1 694802.4
## Oct 2018       589153.8 512755.7 665551.8 472313.0 705994.5
## Nov 2018       589153.8 506079.8 672227.7 462103.1 716204.4
## Dec 2018       589153.8 499901.9 678405.7 452654.8 725652.8
## Jan 2019       589153.8 494124.7 684182.8 443819.4 734488.2
## Feb 2019       589153.8 488679.2 689628.4 435491.1 742816.4
## Mar 2019       589153.8 483514.0 694793.6 427591.7 750715.9
## Apr 2019       589153.8 478589.8 699717.7 420060.8 758246.7
```

More on the `ses()` function here.

You can see in this output under Smoothing parameters that $\alpha = 0.6347$.

- This means that 63.47% of the forecast are based on the most recent observation.

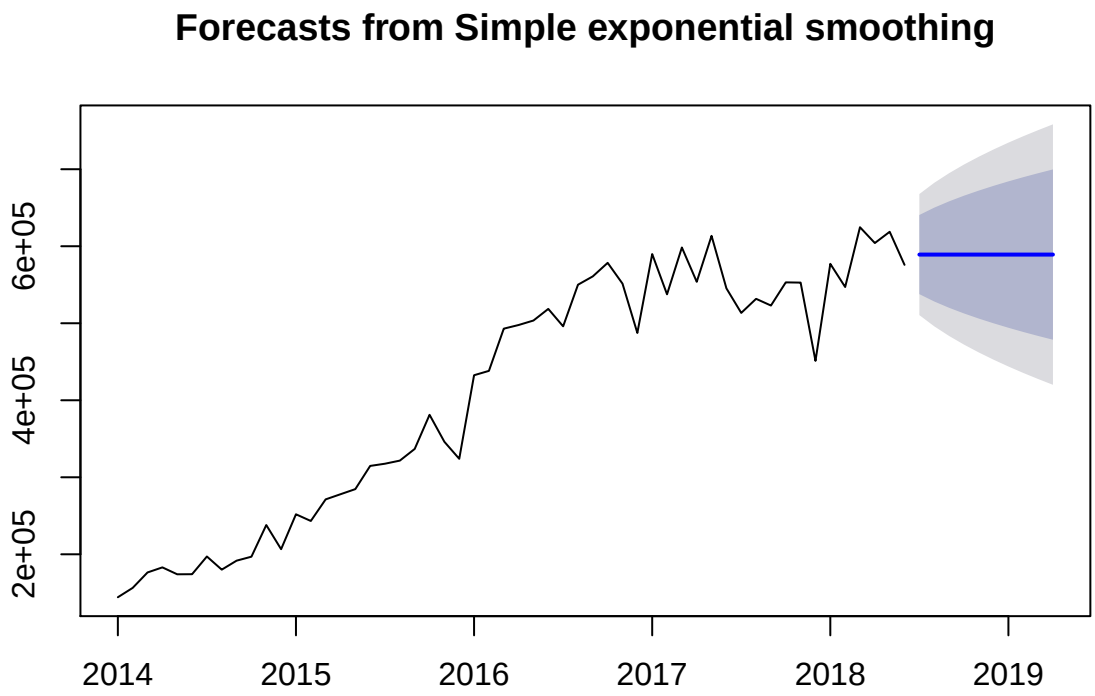
Below that various Error measures - that will be discussed later - are shown and the summary ends with Forecasts.

Since simple exponential smoothing assumes there is no trend or seasonality the forecasts for future time periods is at a certain level but flat in its development over time.

- `Lo 80` and `Hi 80` are the boundaries of an 80% confidence interval and accordingly, the wider bounds of `Lo 95` and `Hi 95` represent the 95% confidence interval (the higher the confidence, the wider the bounds).
- Also, note that with increasing distance into the future the confidence intervals widen.

The plot function produces the `plot` below which represents the forecasted level for future time periods and visualizes the 80% and 95% confidence intervals.

```
#Plot the forecasted values
plot(fit_ses)
```



Component form:

$$\begin{cases} \text{(Forecast eq.)} & \hat{y}_{t+h|t} = l_t \\ \text{(Level eq.)} & l_t = \alpha y_t + (1-\alpha) l_{t-1} \end{cases}$$

- Source here.

2.2 Holt exponential smoothing

Holt exponential smoothing is a time series forecasting approach that fits time series data with an overall level as well as a trend.

- Additionally to simple exponential smoothing, which uses smoothing parameter α only there is also a β smoothing parameter for the exponential decay of the modeled trend component.
  - This β smoothing parameter ranges between 0 and 1, with higher values indicating more weight to recent observations.
  - A β value of 0.5 means that the most recent observation's trend component is weighted with 50% in the modeled trend slope.
- The mechanics of the β smoothing parameter are the same as described for the α smoothing parameter previously. The difference is that they refer to different time series components, e.g. trend and level, respectively.

```
#Fit Holt exponential smoothing model to data and show summary
fit_holt <- holt(Organic_Traffic)
summary(fit_holt)
```

```
##
```

```
## Forecast method: Holt's method
##
## Model Information:
## Holt's method
##
## Call:
##  holt(y = Organic_Traffic)
##
##   Smoothing parameters:
##     alpha = 0.4917
##     beta  = 1e-04
##
##   Initial states:
##     l = 151328.0675
##     b = 9587.0359
##
##   sigma:  38131.24
##
##      AIC     AICc      BIC
## 1360.518 1361.768 1370.463
##
## Error measures:
##                       ME    RMSE      MAE      MPE     MAPE      MASE
## Training set -2562.151 36691.8 27499.14 -1.20321 6.821003 0.2449493
##                       ACF1
## Training set -0.09848366
##
## Forecasts:
##          Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
## Jul 2018       610544.0 561676.8 659411.1 535808.1 685279.8
## Aug 2018       620117.2 565660.9 674573.5 536833.4 703400.9
## Sep 2018       629690.4 570165.5 689215.3 538654.9 720725.8
## Oct 2018       639263.6 575067.2 703459.9 541083.7 737443.4
## Nov 2018       648836.8 580284.8 717388.7 543995.6 753677.9
## Dec 2018       658410.0 585761.5 731058.4 547303.7 769516.2
## Jan 2019       667983.2 591455.6 744510.7 550944.4 785021.9
## Feb 2019       677556.4 597335.6 757777.1 554869.3 800243.4
## Mar 2019       687129.6 603376.9 770882.2 559041.0 815218.1
## Apr 2019       696702.8 609559.9 783845.6 563429.3 829976.2
```

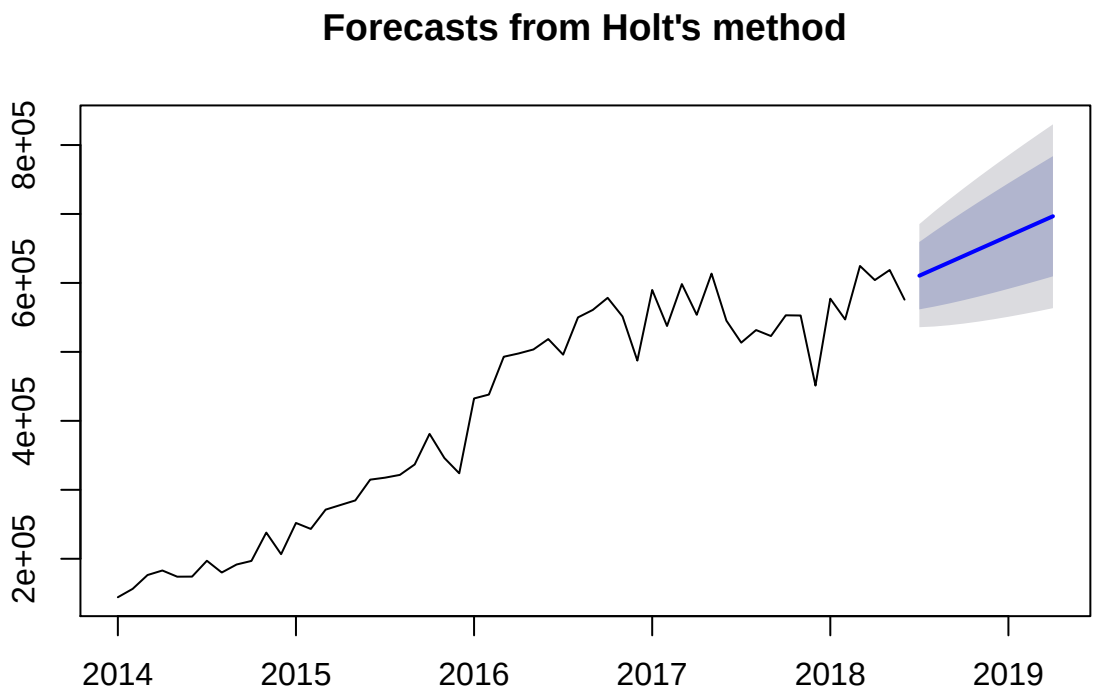
More on the `holt()` function here.

The output of the `summary` function will this time detect values for both the α and β smoothing parameters by optimizing for minimum forecasting error.

- The forecasted values take trend into account and hence, you can see the increase in the values over time.

The `plot` function produces the plot below based on the forecasted values and the confidence intervals.

```
#Plot the forecasted values
plot(fit_holt)
```



The inclusion of the trend component in the model produces a forecast that takes the uptrend of the "Organic_Traffic" data accordingly into account. However, the seasonal fluctuations in the data are not part of the model.

Component form:

$$\begin{cases} \text{(Forecast eq.)} & \hat{y}_{t+h|t} = l_t + hb_t \\ \text{(Level eq.)} & l_t = \alpha y_t + (1-\alpha)(l_{t-1} + b_{t-1}) \\ \text{(Trend eq.)} & b_t = \beta(l_t - l_{t-1}) + (1-\beta)b_{t-1} \end{cases}$$

- Source here.

2.3 Holt-Winters exponential smoothing

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 for the level component, β for the trend component, and γ for the seasonality component.

The following R code fits the Holt-Winters model to the time series data.

- The argument `h = 10` sets the forecasted time periods to 10 (its default is 24), while the default for the previously used functions ses and holt is 10.
- Also, the argument `seasonal = "multiplicative"` indicates that the seasonal component in the data increases in proportion to the level.

```r
#Fit Holt-Winters exponential smoothing model to data and show summary
fit_hw <- hw(Organic_Traffic, h = 10, seasonal = "multiplicative")
summary(fit_hw)
```

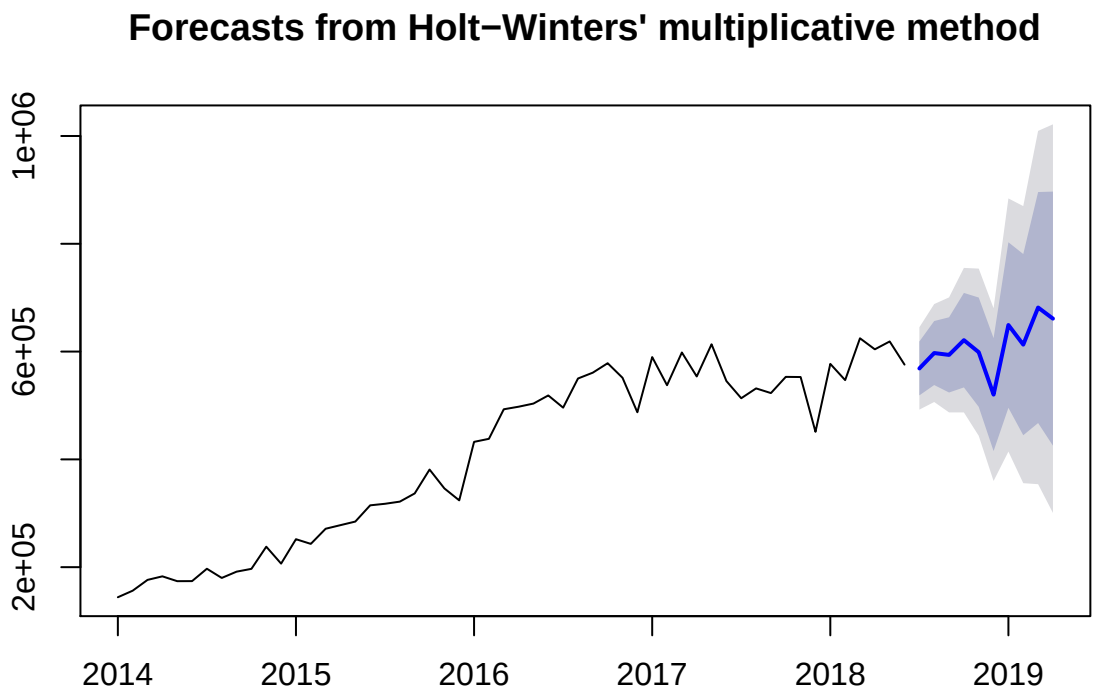
```
##
## Forecast method: Holt-Winters' multiplicative method
##
## Model Information:
## Holt-Winters' multiplicative method
##
## Call:
##  hw(y = Organic_Traffic, h = 10, seasonal = "multiplicative")
##
##   Smoothing parameters:
##     alpha = 0.3463
##     beta  = 0.1877
##     gamma = 1e-04
##
##   Initial states:
##     l = 130521.6671
##     b = 10247.8985
##     s = 0.8433 0.9785 1.0232 0.9861 0.9999 0.9599
##            1.0058 1.062 1.039 1.0796 0.9782 1.0444
##
##   sigma:  0.0685
##
##      AIC     AICc      BIC
## 1326.566 1343.566 1360.379
##
## Error measures:
##                      ME     RMSE      MAE        MPE     MAPE      MASE
## Training set -596.4343 18566.24 14609.47 -0.2138233 4.092297 0.1301342
##                   ACF1
## Training set 0.1731074
##
## Forecasts:
##          Point Forecast    Lo 80    Hi 80    Lo 95     Hi 95
## Jul 2018       568707.6 518767.9 618647.3 492331.4  645083.8
## Aug 2018       597294.8 537909.9 656679.7 506473.5  688116.1
## Sep 2018       593840.8 524093.4 663588.2 487171.3  700510.3
## Oct 2018       621173.1 533522.8 708823.4 487123.6  755222.6
## Nov 2018       598818.9 497403.4 700234.3 443717.3  753920.4
## Dec 2018       520170.6 415430.4 624910.8 359984.3  680356.9
## Jan 2019       649277.3 495745.0 802809.6 414470.0  884084.6
## Feb 2019       612884.7 444811.4 780957.9 355838.8  869930.5
## Mar 2019       681702.6 467423.1 895982.1 353990.4 1009414.8
## Apr 2019       661114.1 425416.8 896811.3 300646.2 1021581.9
```

More on the `hw()` function here.

The `summary` function shows the estimates for α, β, and also γ for the seasonality component.

The `plot` function visualizes the forecast including seasonality.

```r
#Plot the forecasted values
plot(fit_hw)
```



Component form:

$$\begin{cases} \text{(Forecast eq.)} & \hat{y}_{t+h|t} = l_t + hb_t + s_{t+h-m(k+1)} \\ \text{(Level eq.)} & l_t = \alpha(y_t - s_{t-m}) + (1-\alpha)(l_{t-1} + b_{t-1}) \\ \text{(Trend eq.)} & b_t = \beta(l_t - l_{t-1}) + (1-\beta)b_{t-1} \\ \text{(Seasonality eq.)} & s_t = \gamma(y_t - l_{t-1} - b_{t-1}) + (1-\gamma)s_{t-m} \end{cases}$$

- Source here.

2.4 Automated exponential smoothing forecasts

The previous three sections introduced simple, Holt and Holt-Winters exponential smoothing. These three exponential smoothing models use different (combinations of) time series components (e.g. level, trend, seasonality) for their respective forecasts.

- There is also an automated exponential smoothing forecast that can be accomplished with the `ets()` function from the `forecast` package.
  - In this case, it is automatically determined whether a time series component is additive or multiplicative.
  - The initial state variable and smoothing parameters [...] are optimized to yield the model by default with the least AICc (Akaike information criterion with a correction for small sample sizes) as chosen automated forecast model.
- The previously used functions `ses()`, `holt()` and `hw()` are convenience wrappers to the `ets()` function.
- The `ets()` function takes several arguments, including `model`, which 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.
  - This means that `ses(Organic_Traffic)` is, for instance, the convenience wrapper to `ets(Organic_Traffic, model = "ANN")`, which fits a model with additive error and without trend or seasonality.

- Also, the `ets()` function can be used with `Z` for all time series components or without the model argument to fit the automated exponential smoothing forecast to the data.
  - This means `ets(Organic_Traffic)` is the same like `ets(Organic_Traffic, model = "ZZZ")`.
  - One drawback of the `ets()` function is that no time horizon can be chosen.

```
#Fit automated exponential smoothing model to data and show summary
fit_auto <- forecast(Organic_Traffic, h = 10)
summary(fit_auto)
```

```
##
## Forecast method: ETS(M,A,M)
##
## Model Information:
## ETS(M,A,M)
##
## Call:
##  ets(y = object, lambda = lambda, biasadj = biasadj, allow.multiplicative.trend = allow.multiplicati
##
##   Smoothing parameters:
##     alpha = 0.3188
##     beta  = 0.127
##     gamma = 1e-04
##
##   Initial states:
##     l = 123847.8715
##     b = 10582.8208
##     s = 0.8525 1.0175 1.007 0.9661 0.973 0.9787
##            1.0042 1.0361 1.0393 1.0833 0.997 1.0453
##
##   sigma:  0.0652
##
##      AIC     AICc      BIC
## 1321.090 1338.090 1354.903
##
## Error measures:
##                       ME     RMSE      MAE        MPE     MAPE      MASE
## Training set -173.1986 22240.86 17499.12 -0.1353928 4.458985 0.1558738
##                    ACF1
## Training set 0.2723685
##
## Forecasts:
##          Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
## Jul 2018       583461.8 534722.7 632201.0 508921.8 658001.9
## Aug 2018       588494.1 534778.5 642209.8 506343.1 670645.1
## Sep 2018       592762.7 532009.3 653516.1 499848.5 685677.0
## Oct 2018       626618.1 553376.8 699859.5 514605.2 738631.1
## Nov 2018       642021.2 555994.4 728048.0 510454.6 773587.8
## Dec 2018       545289.8 461658.8 628920.9 417387.2 673192.5
```

```
## Jan 2019       677755.0 559398.1 796112.0 496743.7 858766.3
## Feb 2019       655091.9 525727.6 784456.2 457246.3 852937.6
## Mar 2019       721216.3 561344.7 881087.8 476713.8 965718.7
## Apr 2019       700996.2 527826.6 874165.8 436156.1 965836.3
```

From the output in the previous image, you can see an AICc value of 1338.090, which is lower than for all other presented exponential smoothing models.

- Also, the headline of the plot below contains `ETS(M,A,M)` showing that the automated model characterized error of the "Organic_Traffic" data as multiplicative (`M`), trend as additive (`A`) and seasonality as multiplicative (`M`).

```
#Plot the forecasted values
plot(forecast(fit_auto))
```