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
```

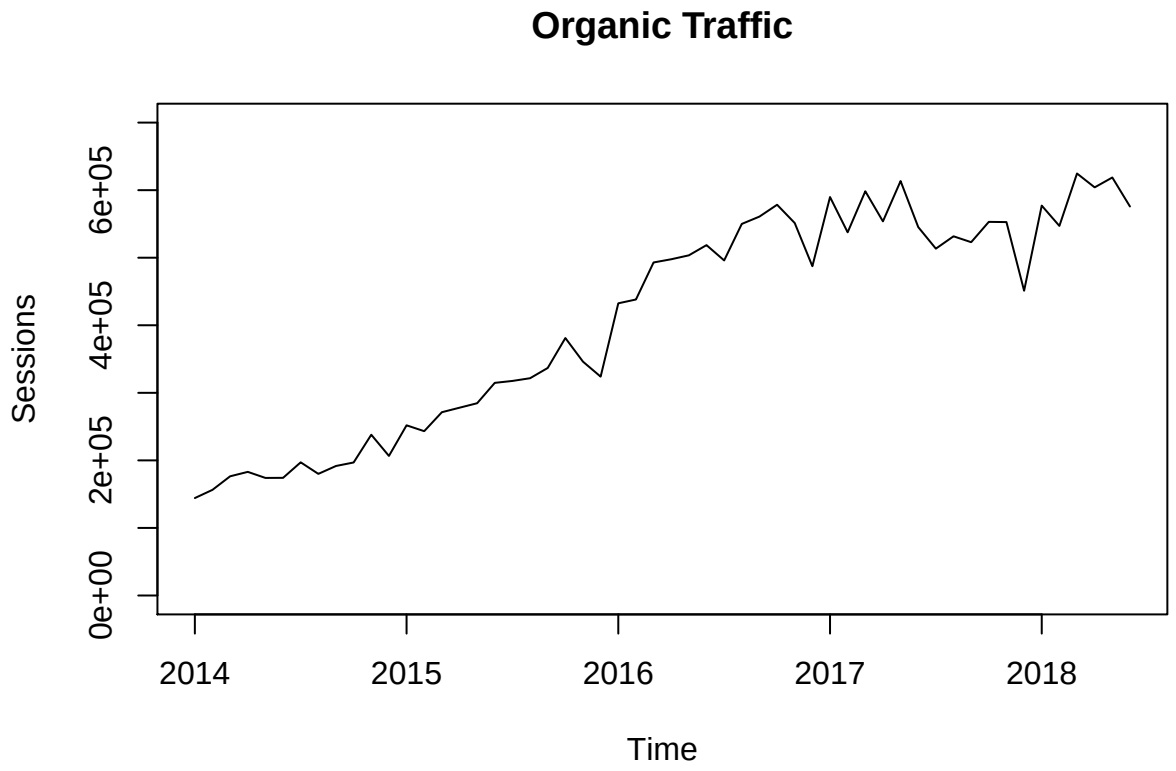
The following R code converts the `Month` column in the appropriate format as well as the `Organic Sessions` column into a time series object of monthly data (`frequency = 12`) and plots the data as a line graph.

```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.

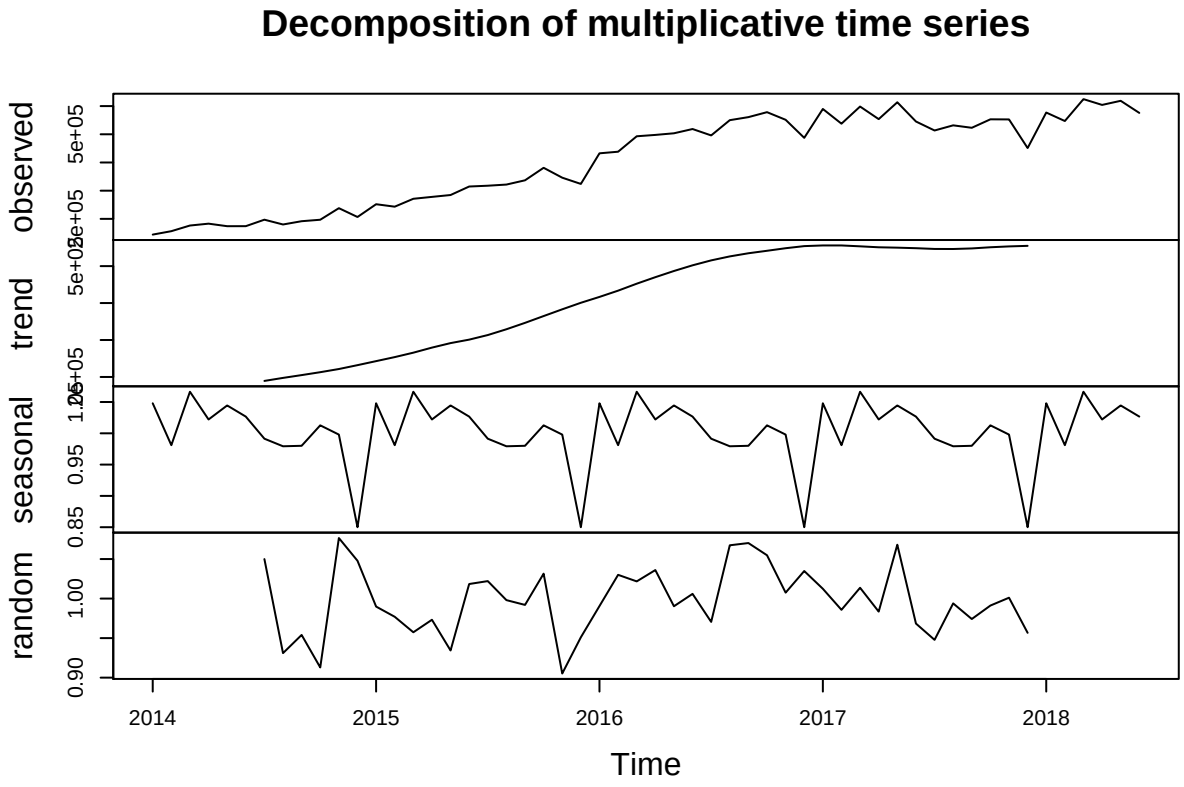
- There are various different methodologies of time series decomposition including classical, X11, SEATS and STL that come with varied advantages and disadvantages as well as requirements for the time unit of the data.
- In this case, we will use the classical time series decomposition as it is widely 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).

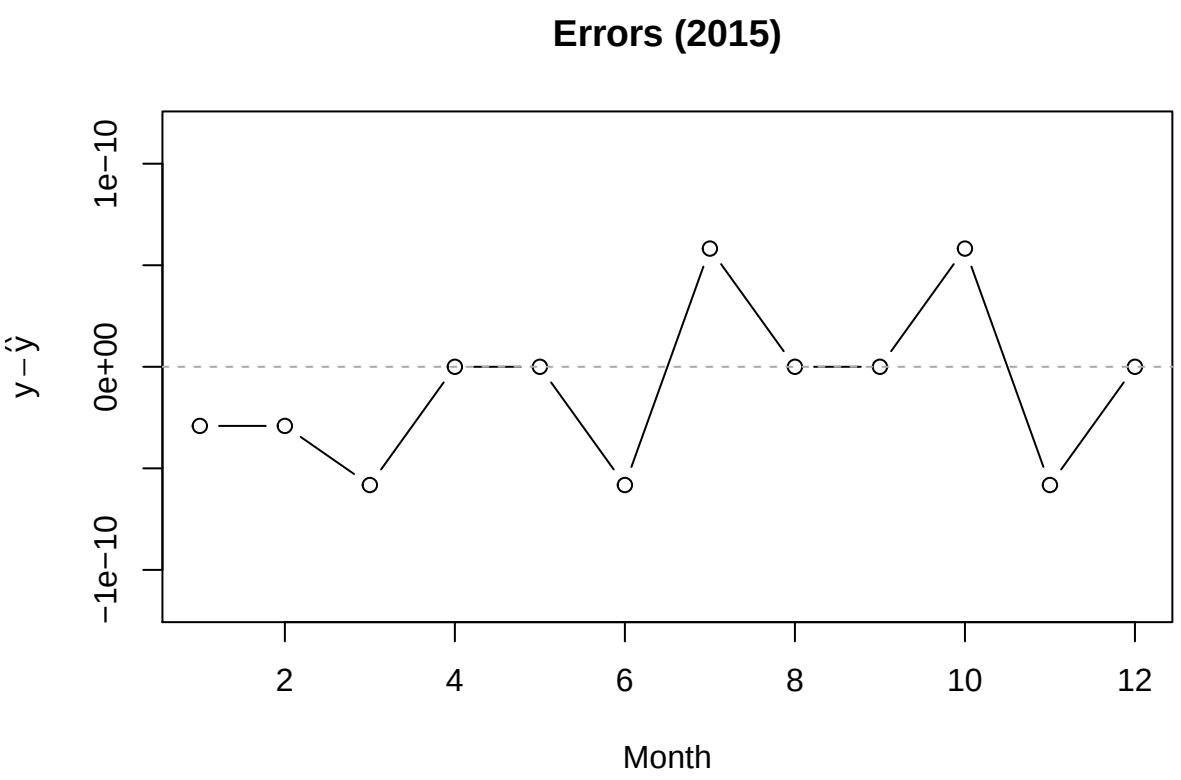
See below the differences between the real values (y_t) and the values reconstruced from the components ($\hat{y}_t = S_t \cdot T_t \cdot R_t$), for the 2015 data:

```
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.

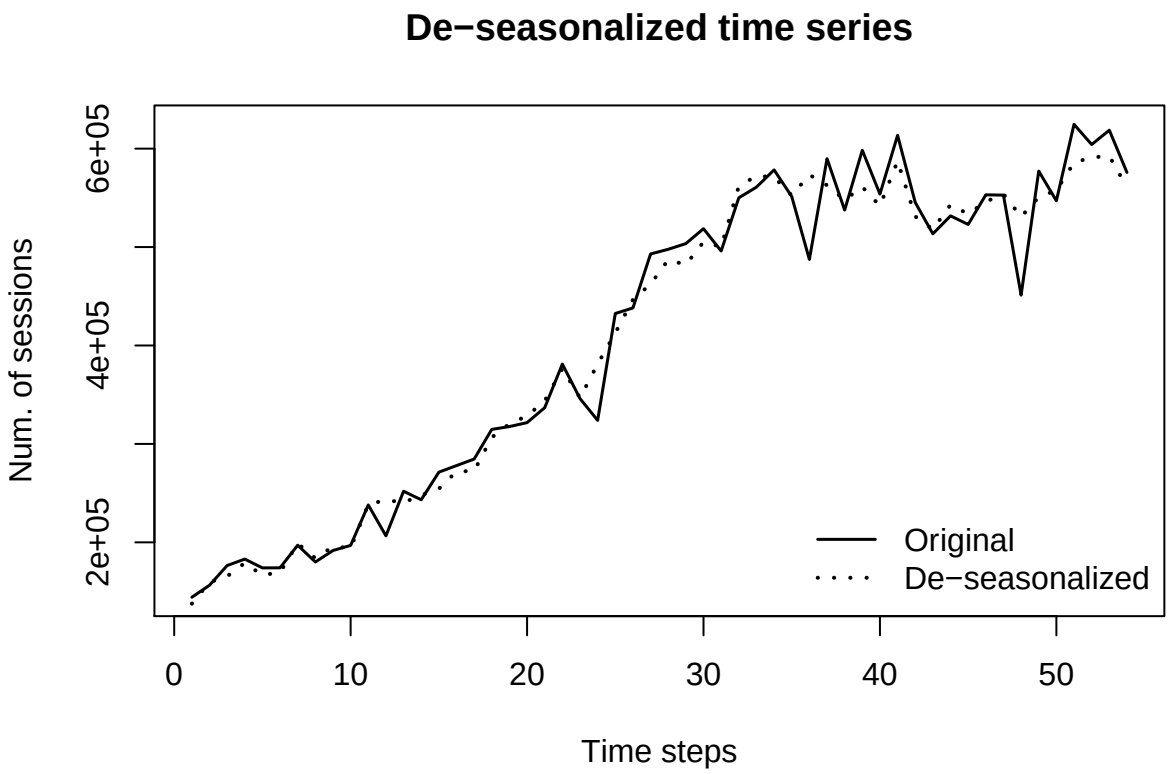
- For December 2014, this looks like this: $\text{Dec}_{2014,\text{deseasonalized}} = \frac{206,609}{0.8500177} = 243,064.4$.
- Accordingly, for November 2014 this looks like this: $\text{Nov}_{2014,\text{deseasonalized}} = \frac{237,982}{0.9979517} = 238,470.5$.
  - If you looked at the "Organic_Traffic" 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"))
```



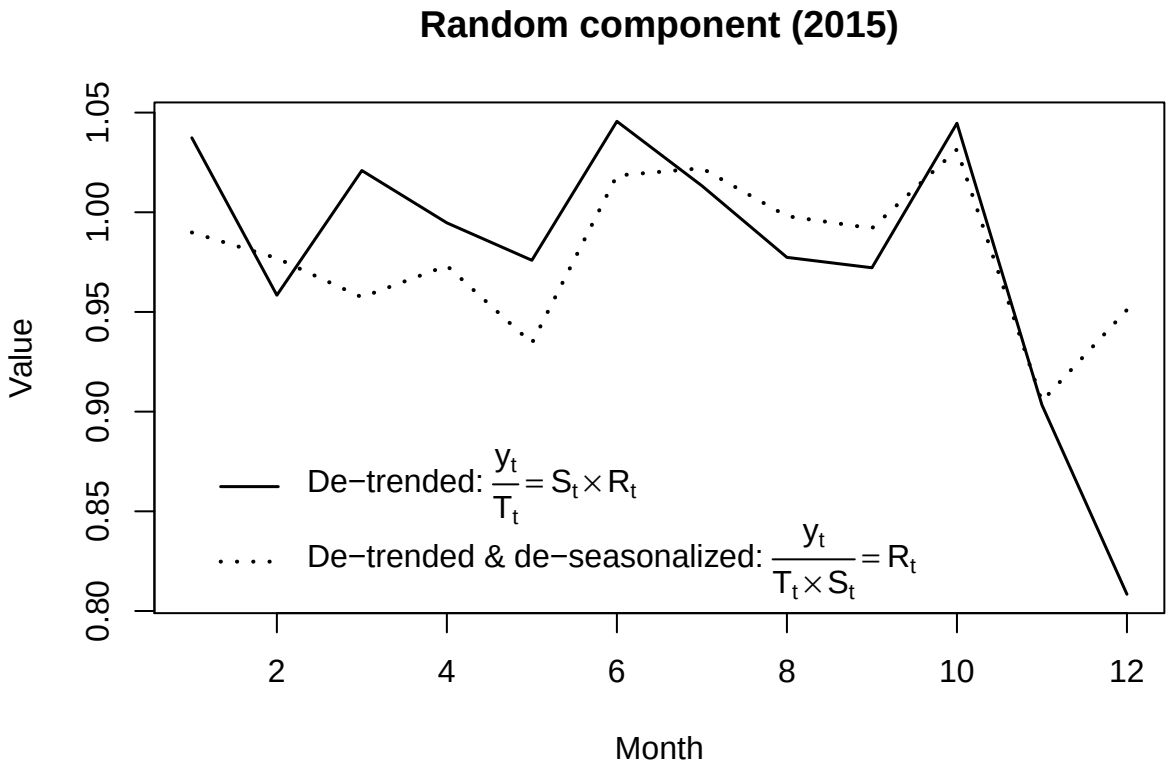
4. Calculate the error component:

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.
