# Advanced Research Methods - E7004

## Day 4 - Time-Series Analysis

Dr. Fabio Veronesi

22 March 2018

## Summary

- Introduction
- Loading time-series data in R
- Subsetting time-series
- Trend
- Functions applied to moving windows
- Detecting Anomalies and Outliers

---

## Introduction

After discussing about inferential statistics, today we will change topic and talk about time-series analysis. As the word suggests, time-series are data that have been measured at fixed time intervals, e.g. data coming from sensors. The main characteristic of time-series is that they feature some form of time wise variation, which may have different scales. For example, we can image that a sensor recording outdoor air temperature will present different values for nights as compared to days. It will also present differences between temperatures recorded in winter, compared to summer temperature. It may also show years of higher (or lower) than average temperature, again compared to previous years. These are sources of variation at different time scales, from hours to days and years. The main objective of a time-series analysis is describe these different scales of variation and use them to better understand how data changes with time and detect anomalies.

---

## Loading time-serie data in R

Time-series can be loaded into R using the function `ts` included in the `base` package that is automatically loaded when we open the software. However, some of the functions we are testing in this lecture are available in the package `xts`, which needs to be installed and loaded:

```r
install.packages("xts")
install.packages("forecast")
```

```r
library(xts)
library(forecast)
```

I prepared two examples of time-series, which I downloaded from the website of EPA (Environmental Protection Agency) using a custom function I developed for an experiment I presented on my Blog. The first dataset records temperature hourly values for the whole 2015, while the second features pm10 (air particle matter of size 10$\mu m$) hourly values for the same year. Please download both datasets, place them in a folder of your choice and load them in R using the function `read.csv`:

```r
PM10 = read.csv("PM10.csv")
TEMP = read.csv("Temperature.csv")

str(TEMP)
```

---

Again, both datasets are available from my GitHub and can be imported as follows:

```r
library(RCurl)

PM10.URL =
getURL("https://raw.githubusercontent.com/fveronesi/AdvancedResearchMethods/m
aster/PM10.csv")
PM10 = read.csv(text=PM10.URL)

TEMP.URL =
getURL("https://raw.githubusercontent.com/fveronesi/AdvancedResearchMethods/m
aster/Temperature.csv")
TEMP = read.csv(text=TEMP.URL)

str(TEMP)

## 'data.frame':    8755 obs. of  3 variables:
##  $ Date.Local        : Factor w/ 365 levels "01/01/2015","01/02/2015",..:
1 1 1 1 1 1 1 1 1 1 ...
##  $ Time.Local        : Factor w/ 24 levels "00:00","01:00",..: 1 2 3 4 5 6
7 8 9 10 ...
##  $ Sample.Measurement: int  30 31 31 31 31 32 31 32 35 37 ...
```

---

As mentioned these datasets were downloaded from EPA, which provides free data from all their measuring stations. I then extracted from the entire dataset only records coming from a particular station, selected randomly (the same for both). Since these data come from the US, temperature is recorded in Fahrenheit, so we need to convert it into Celsius:

```r
TEMP$TempC = (TEMP$Sample.Measurement - 32) * 5/9
```

Before we can proceed with our analysis there is one crucial step to take, namely convert date and time information into something R can work with. First of all, let's look at the columns holding these information, with the function `head`, which shows only the first few rows of a `data.frame`:

```
head(PM10)
```

```
##   Date.Local Time.Local Sample.Measurement
## 1 01/01/2015      00:00                 12
## 2 01/01/2015      01:00                 17
## 3 01/01/2015      02:00                 49
## 4 01/01/2015      03:00                 16
## 5 01/01/2015      04:00                 29
## 6 01/01/2015      05:00                  5
```

In this dataset, dates are reported in the format YEAR/MONTH/DAY, while time in HOUR:MINUTE. We can convert both using the function `as.POSIXct`, which is a Date-Time class of objects. Let's see the code to do it:

```
DateTime.TEMP = as.POSIXct(paste(TEMP$Date.Local, TEMP$Time.Local),
format="%d/%m/%Y %H:%M", tz="GMT")
DateTime.PM10 = as.POSIXct(paste(PM10$Date.Local, PM10$Time.Local),
format="%d/%m/%Y %H:%M", tz="GMT")

head(DateTime.PM10)
```

```
## [1] "2015-01-01 00:00:00 GMT" "2015-01-01 01:00:00 GMT"
## [3] "2015-01-01 02:00:00 GMT" "2015-01-01 03:00:00 GMT"
## [5] "2015-01-01 04:00:00 GMT" "2015-01-01 05:00:00 GMT"
```

Let's analyse what is going on in the code above. We have several functions nested within each other. The innermost function is `paste`, which does exactly what the name suggests. In fact, the function `paste` is used to paste together elements in the vector `PM10$Date.Local` with elements of the vector `PM10$Time.Local`. This creates a new vector with both of strings with both dates and time. The next function is `as.POSIXct`, which takes the new vector created by `paste` and transforms it into a Date-Time object. A crucial option to include into a call to the function `as.POSIXct` is `format`, which allows us to tell R what format to expect from the vector. As you can see the option `format` takes a schematic representation of the date-time format of the vector. Here we have several options to choose from, depending on our data. For more info please look here and here.

Next we need to include date-time information into the two `data.frame` as new columns:

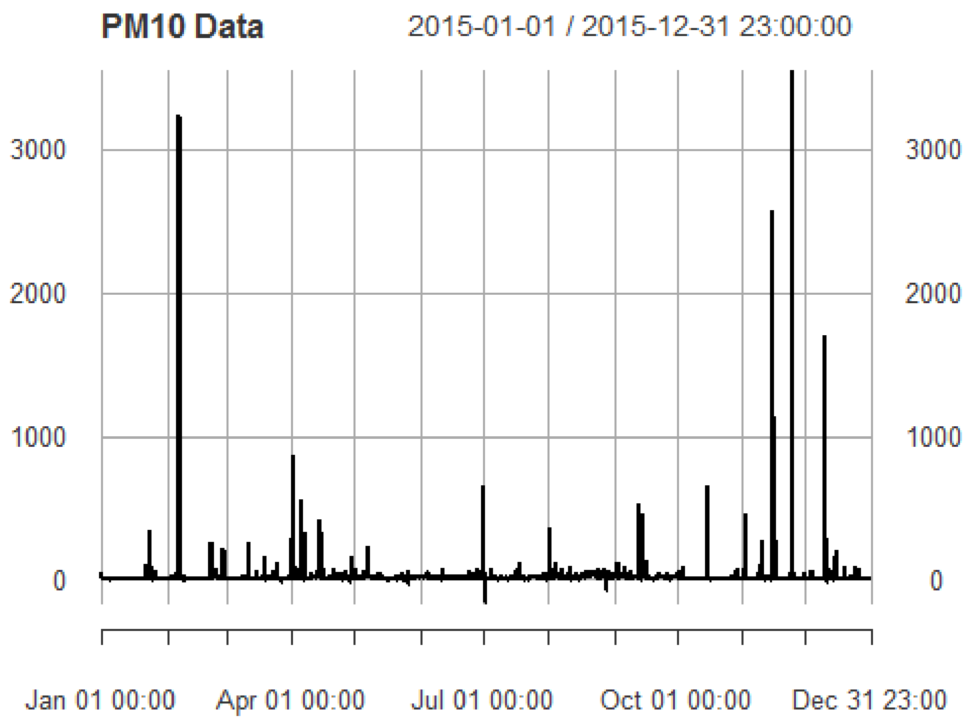
```
PM10$DateTime = DateTime.PM10
TEMP$Datetime = DateTime.TEMP
```

Now that we have the date-time information we can create a time-series object with the function `xts`:

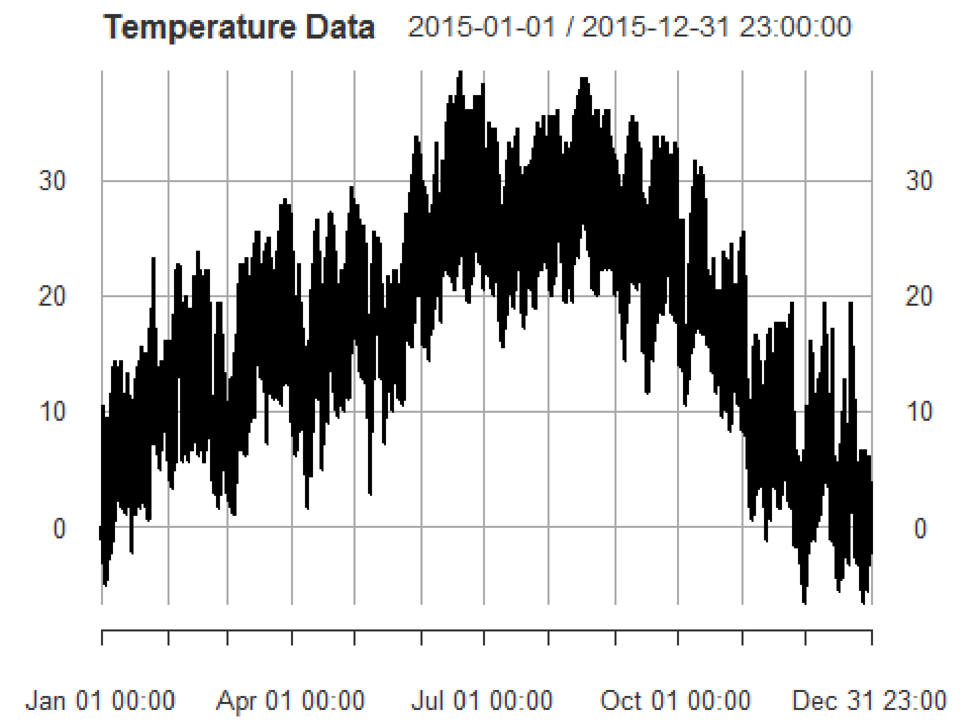
```
PM10.TS = xts(x=PM10$Sample.Measurement, order.by=PM10$DateTime)
TEMP.TS = xts(x=TEMP$TempC, order.by=TEMP$Datetime)
```

We can look at our two time-series by plotting them:

```r
plot(PM10.TS,main="PM10 Data")
```



```r
plot(TEMP.TS,main="Temperature Data")
```

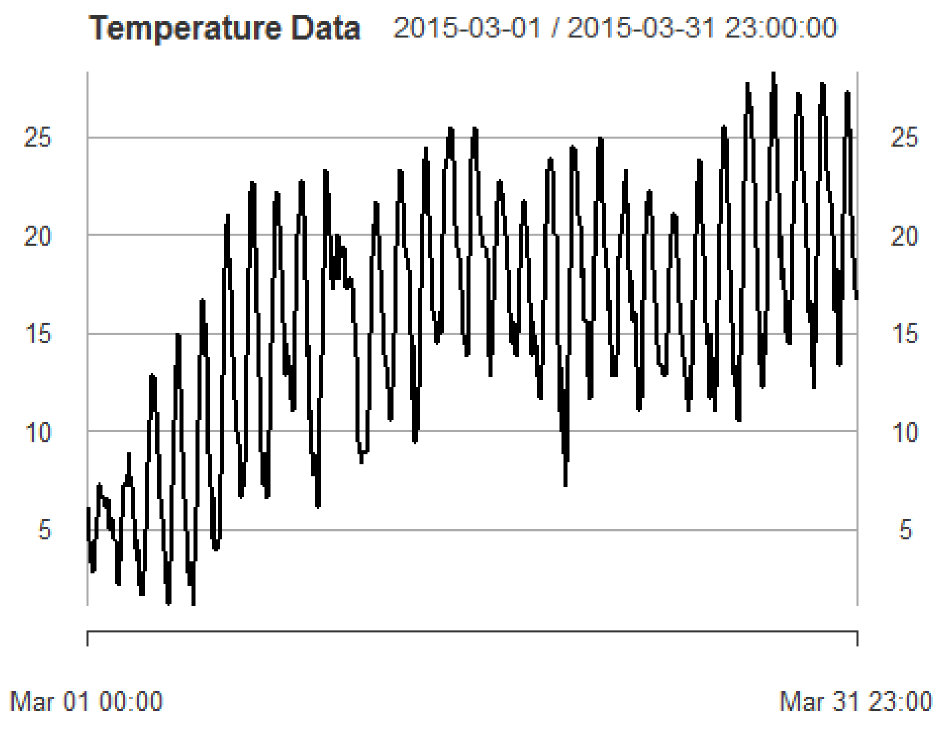


I decided to include both into this lecture because the questions we ask when dealing with time-series are very dependent upon the data we are dealing with. For example, for PM10 we are probably mostly interested in identifying peaks or values that are above a certain threshold (since high levels of PM10 can be very dangerous). On the contrary, while we may still be interested in looking at peaks or higher than average values of temperature, with this sort of time series we are mostly interested in identifying a general trend, or a seasonal trend.

## Subsetting time-series objects

We have loaded a full year of hourly data. However, let's say we want to extract only a certain month or a particular range. This is how we can do that in R. We will first extract one month of data:

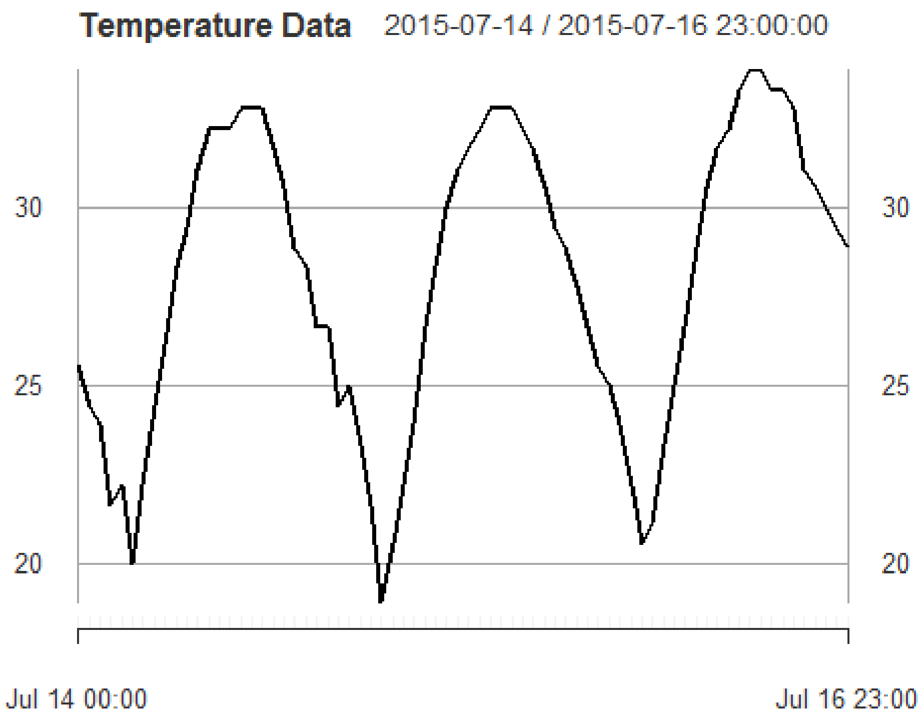
```
TEMP.March = TEMP.TS['2015-03']

plot(TEMP.March,main="Temperature Data")
```



The subsetting is done very similarly to what we have done for `data.frame`, i.e. using square brackets. Within the brackets we just need to specify which month to extract, with the correct format for the date.

For extracting a particular range we can use the following code:

```r
TEMP.MidJuly = TEMP.TS['2015-07-14/2015-07-16']

plot(TEMP.MidJuly,main="Temperature Data")
```



For more info on subsetting `xts` objects please take a look at: XTS Vignette

---

## Trend

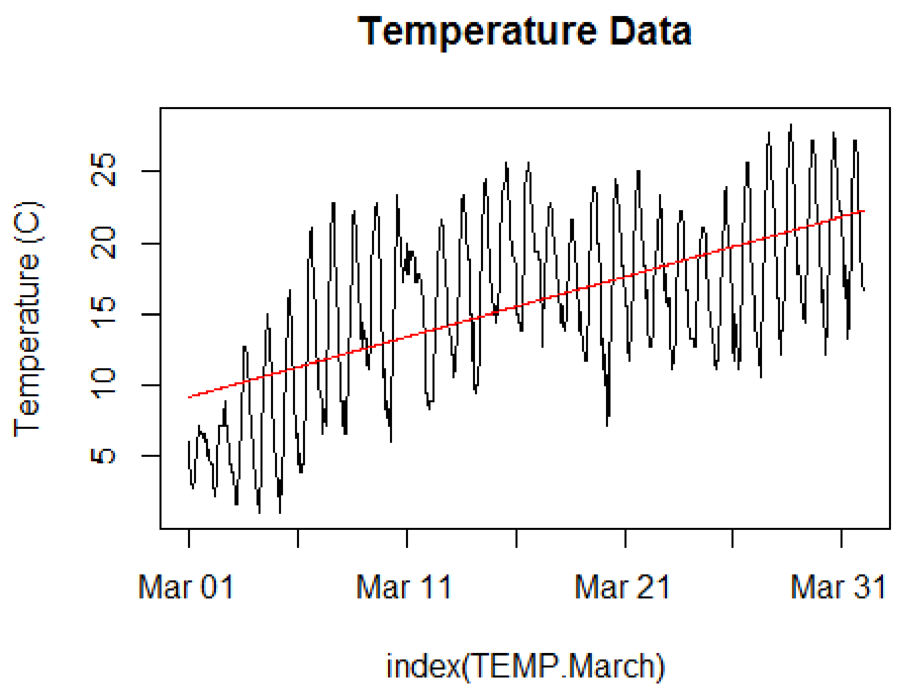
For the large majority of environmental data our interest lies in finding the general and seasonal trends. For trend we mean the general direction of the time-series. We can better understand this concept by looking at the plot for the month of March above. As you can see the time-series present some daily fluctuations, but in general the signal is increasing with time. This is what the general trend is, and it can be simply computed with a linear regression:

```r
l.trend = lm(coredata(TEMP.March) ~ index(TEMP.March))
```

Here we can see two new functions: `coredata`, which extract the variable from the time-series, and `index`, which extracts the time component.

Now we can plot it onto the plot above:

```r
plot(index(TEMP.March), coredata(TEMP.March), main="Temperature Data",
type="l", ylab="Temperature (C)")
lines(index(TEMP.March), l.trend$fitted.values, col="red")
```

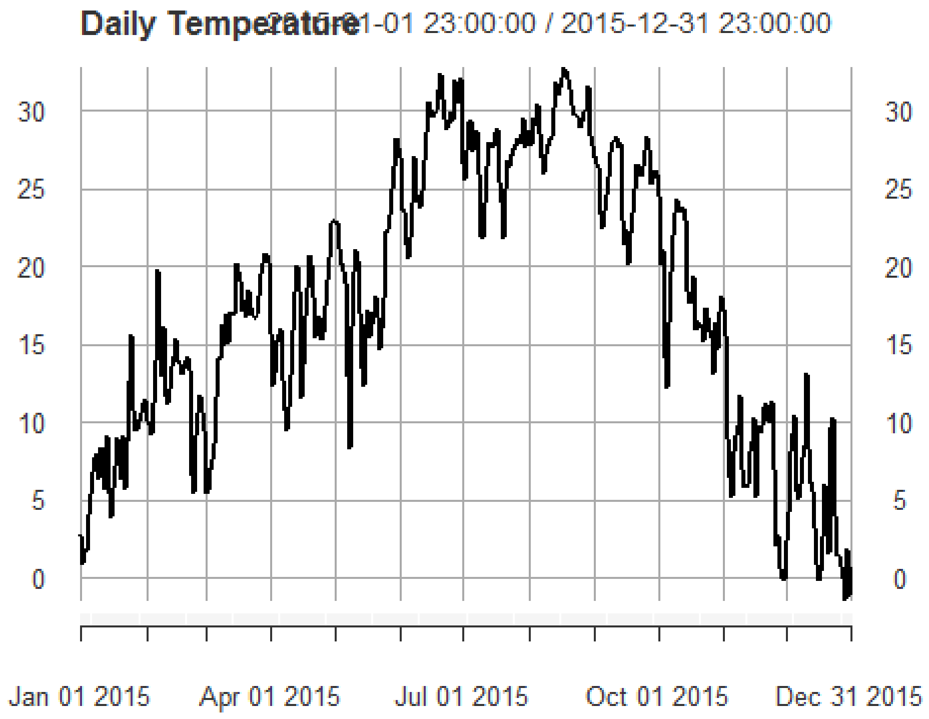


We can also apply particular functions to the time-series, for example to calculate mean values of a moving daily window:

```
Daily.Temperature = apply.daily(x=TEMP.TS, FUN=mean)
```

This allows us to have a better idea of the general trend across the full year:

```
plot(Daily.Temperature, main="Daily Temperature")
```



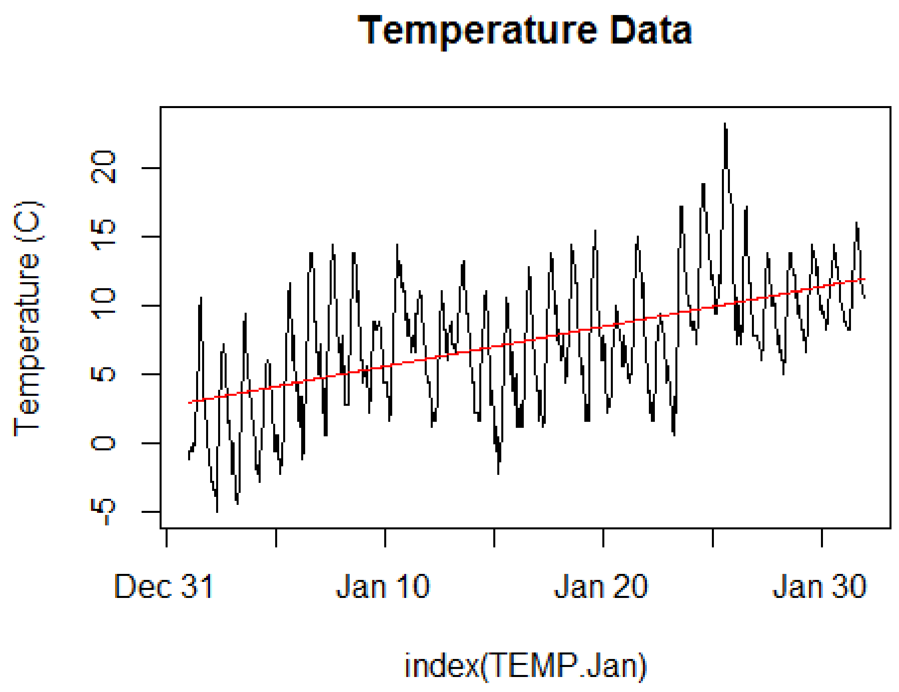
We can also increase the size of the moving windows with the functions `apply.weekly`, `apply.monthly`, `apply.quarterly`, and `apply.yearly`, if we had more than one year of data. There is also the function `period.apply`, which allows us to define a custom period of time when to fit our function.

These functions could be used to detect differences in the general trend, which may indicate changes from the "normality" that we would expect. We could for example include in `apply.monthly` a custom function to fit a line through the data and extract its slope:

```
Monthly.slope = apply.monthly(x=TEMP.TS, FUN=function(x){
  LM = lm(coredata(x) ~ index(x))
  coef(LM)[2]
})

Monthly.slope

##                                [,1]
## 2015-01-31 23:00:00  3.366464e-06
## 2015-02-28 23:00:00 -1.169485e-06
## 2015-03-31 23:00:00  4.893317e-06
## 2015-04-30 23:00:00  3.102771e-06
## 2015-05-31 23:00:00  2.194786e-06
## 2015-06-30 23:00:00  3.841056e-06
## 2015-07-31 23:00:00  4.862285e-07
## 2015-08-31 23:00:00  7.871123e-07
## 2015-09-30 23:00:00  3.055973e-07
## 2015-10-31 23:00:00 -2.112511e-06
## 2015-11-30 23:00:00 -2.914003e-06
## 2015-12-31 23:00:00 -2.467235e-06
```

In general, we would expect temperature to increase in spring and summer months, and decrease in autumn and winter. However, these results suggest a more complex picture. January for example presents a relatively high positive slope. Let's give a closer look:

```
TEMP.Jan = TEMP.TS['2015-01']

l.trend = lm(coredata(TEMP.Jan) ~ index(TEMP.Jan))

plot(index(TEMP.Jan), coredata(TEMP.Jan), main="Temperature Data", type="l",
ylab="Temperature (C)")
lines(index(TEMP.Jan), l.trend$fitted.values, col="red")
```



Is this normal to record an average temperature of 10 degrees Celsius at the end of January? This is a simple way to detect anomalies.

With a simple line of code we were able to instantly detect months that behave differently compared to the general trend of the time-series. This is a very powerful technique for example to create early warning systems, which can detect anomalies in real-time.

## Decomposition

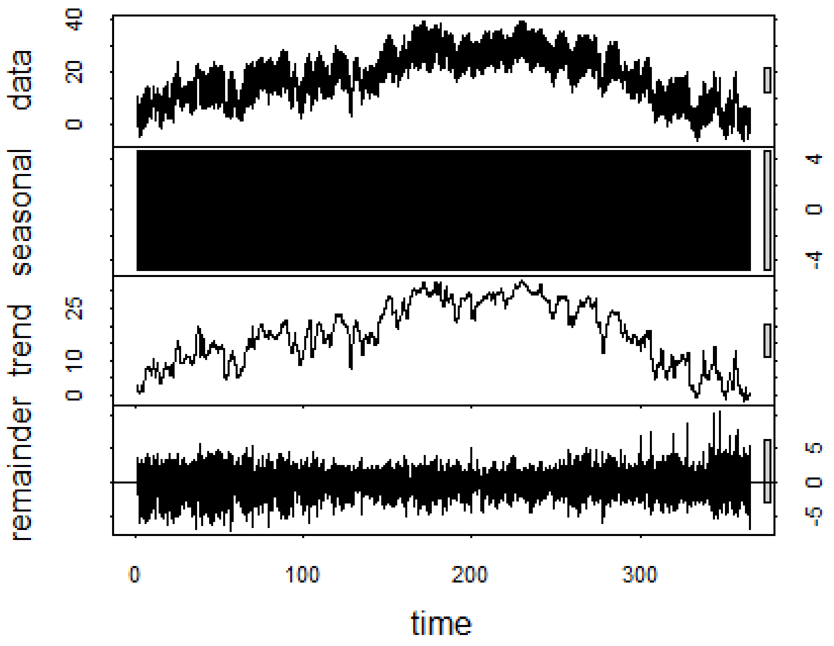
Particularly with environmental data, we are often interested in decomposing the time-series, meaning extracting trend at various frequencies. Before we can do that we need to specify the frequency, defined as the number of observations per cycle. Usually, for time-series with less observations than once per week, the frequency is just the number of observations in a year. So if we had a monthly time-series we would set up a frequency of 12 (since there are 12 months in a year). The matter is a bit more complicated in cases of time-series with more than one observations per week (such as in our case where we have one observation per hour). In such cases we need to specify the frequency of the seasonality.

As mentioned, for environmental data we expect to find more than one scale of cyclicity. In our case we have a yearly cycle, with higher temperature during summer months. We may also have a monthly cycle and we certainly have daily cycles, with higher temperatures during the day and lower during the night. The number we input for `frequency` depends on which cycle we want to detect (more info here). For daily values we would input `24` (24h per day), for monthly we would input 2474 (24h times 7 days times 4 weeks). We can for example look at daily seasonality:

```r
attr(TEMP.TS, 'frequency') <- 24
```

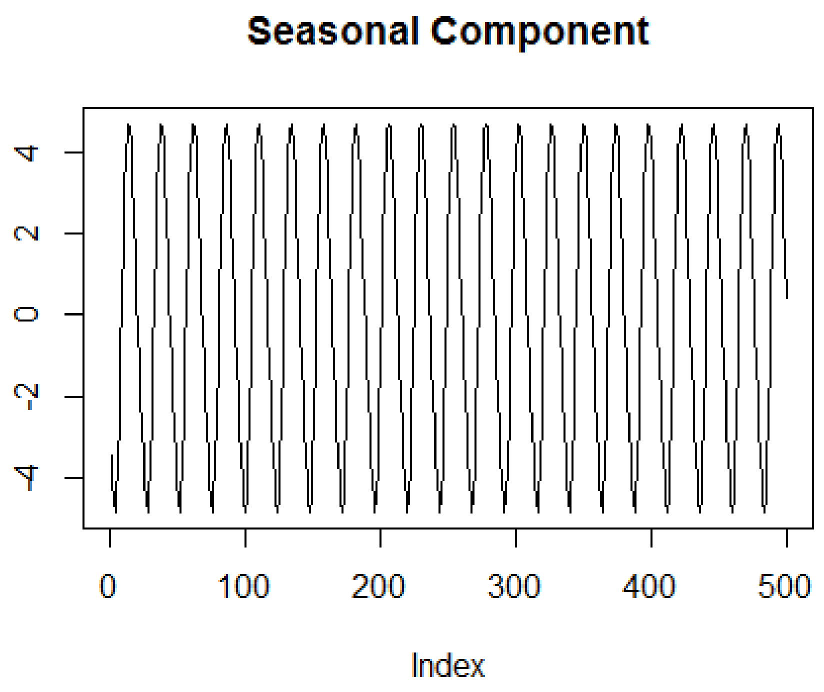
Once the frequency is set correctly we can use the function `stl` to compute the local trends. This function uses a local polynomial regression loess:

```r
STL <- stl(TEMP.TS,"periodic")

plot(STL)
```



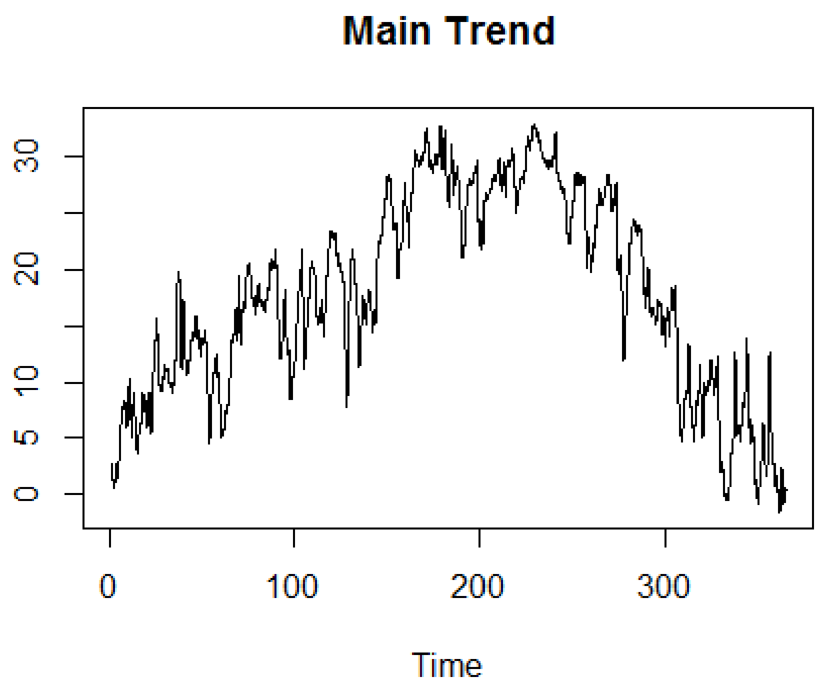
The plot above shows the main trend, and the very complex seasonal component. We can zoom into the seasonal component by plotting only the first 500 time-steps (around 21 days):

```
plot(STL$time.series[1:500,1], main="Seasonal Component", ylab="", type="l")
```



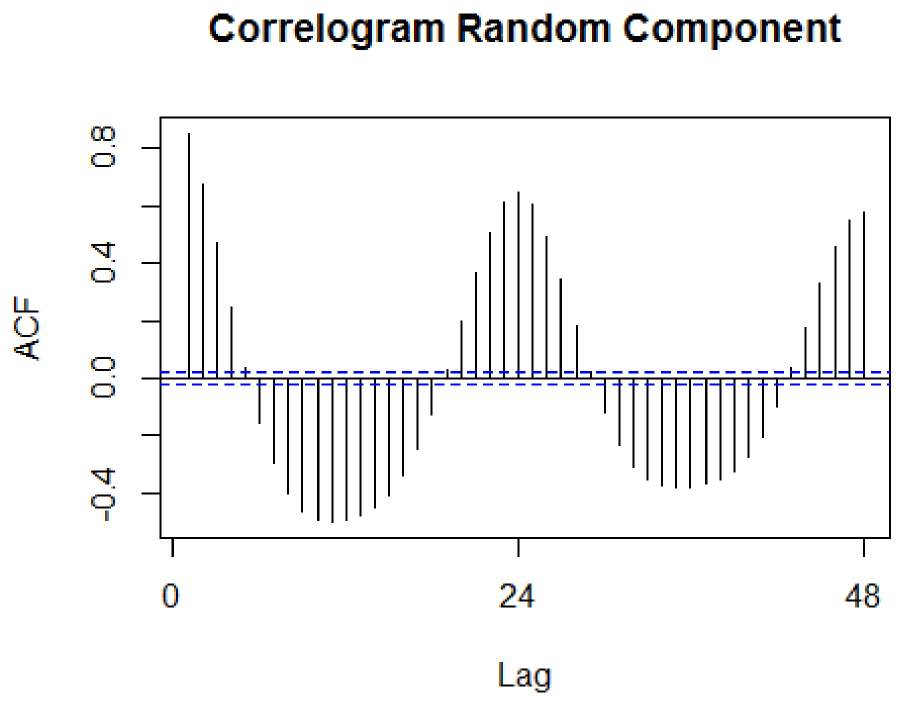
As you can see we clearly identify 21 peaks, meaning the function is doing what we asked, thus removing the daily variation from the data. By doing so we can better appreciate the yearly trend, which we can plot with the following line:

```
plot(STL$time.series[,2], main="Main Trend", ylab="")
```



Now that we have a way of removing all the cyclic components we can also produce the correlogram of the random part of the time-series. The correlogram is a plot of correlation between values at time $t$ with values at time $t \pm x$. The function to compute the correlogram is `Acf` in the package `forecast`. In the function we need to include the random component of the time-series we just decomposed, which can be done as follows:

```
Acf(STL$time.series[,3], main="Correlogram Random Component")
```



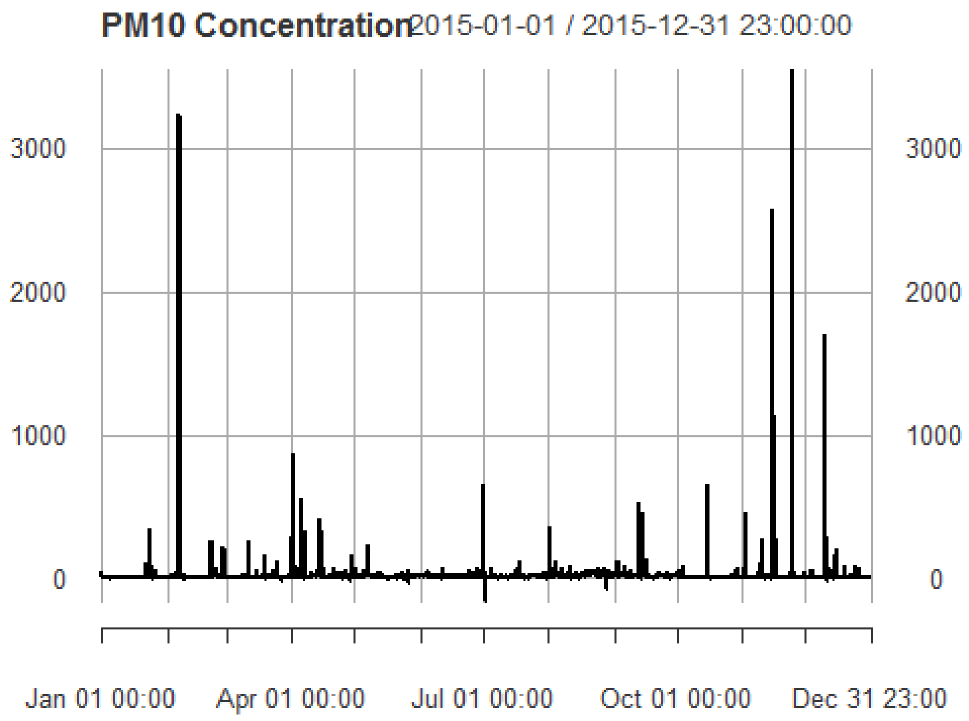
This correlogram seems to show that after removing the daily variations, data still present a 12 hour cycle.

---

## Outliers

Many times with time-series we are asked to provide methods to detect anomalies. These may be values that do not fit with the average trend, or may be values that are above/below a certain safety threshold. We can start by talking about how to detect values that are outside a safety threshold, by looking at the PM10 time-series (measurement units are $\frac{\mu g}{m^3}$). Here we can clearly see some spikes, which may suggest particularly high level of pollution.

```
plot(PM10.TS, main="PM10 Concentration")
```



The question is, were these values outside the legal limit?

According to the EPA PM10 have a limit of 150 $\frac{\mu g}{m^3}$ over 24h. So theoretically we just need to compute the daily average and check whether it is above 150 $\frac{\mu g}{m^3}$. Let's see the code to compute how many days PM10 are above the limit:

```
sum(apply.daily(PM10.TS, FUN=mean)>150)

## [1] 6
```

Here we are first applying the function `apply.daily` to compute daily averages. Then we are including a logical statement `>150` to check which values, within the vector created by `apply.daily`, are above 150. Then we are using the function `sum` that computes the sum of all values that resulted TRUE after the logical statement (so sum of the values above 150).

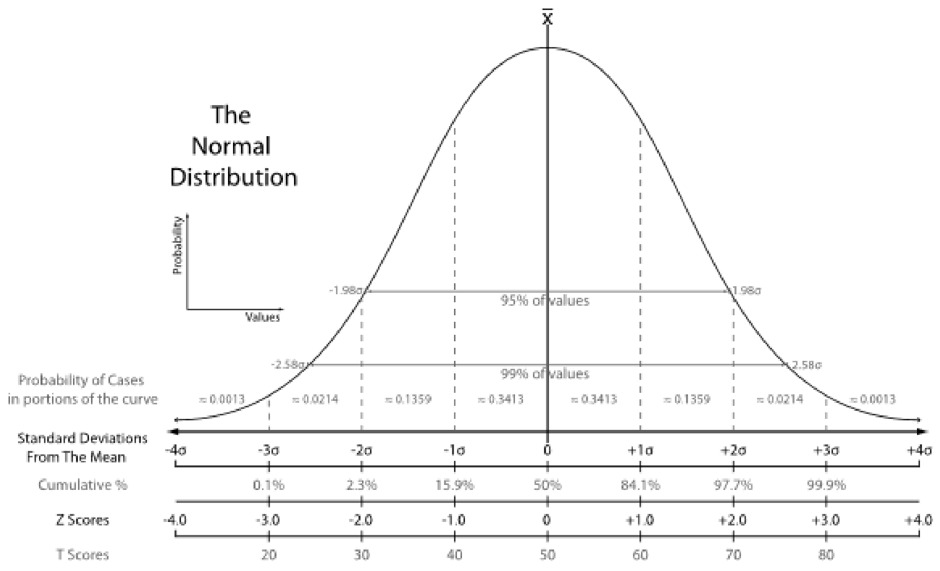
To check where these anomalies are along the time-series we can use the following line:

```
index(PM10.TS)[which(apply.daily(PM10.TS, FUN=mean)>150)]

## [1] "2015-01-02 12:00:00 GMT" "2015-01-02 13:00:00 GMT"
## [3] "2015-01-14 06:00:00 GMT" "2015-01-14 07:00:00 GMT"
## [5] "2015-01-14 16:00:00 GMT" "2015-01-15 08:00:00 GMT"
```

Here we are using the function `which` that identifies the indexes (i.e. the locations along the vector `index(PM10.TS)`) where the logical statement is TRUE.

Now we can explore the other point, namely how to find anomalies, or extreme values. This concept is very close to the concept of outliers in statistics. An outlier is defined as an observation that has an abnormal distance from the mean. This does not necessarily mean it is a value to exclude, but it may be wise to be able to detect these improbable values.

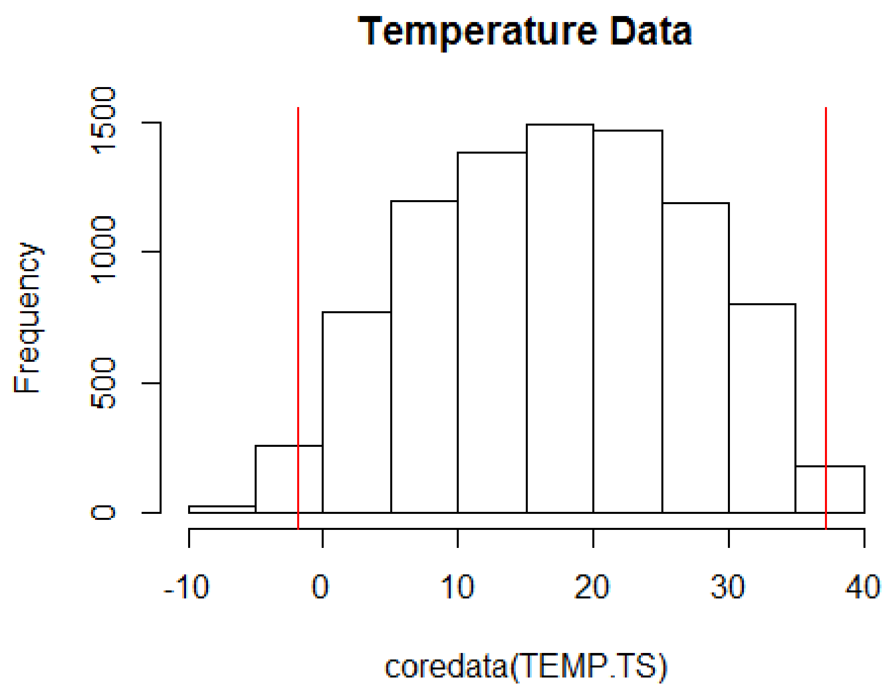
We can detect outliers using the mean and standard deviation. The standard deviation measures the average distance of all observations from the mean. Please look at the image below:



*Normal Distribution*

This image clearly shows how to extract extreme values. One way is taking the interval outside the range $mean \pm (2 \times SD)$; this should include the 5% most extreme observations. We could also use the range $mean \pm (3 \times SD)$, to be more conservative and include only 1% of extreme observations. The code is clearly very simple:

```
hist(coredata(TEMP.TS), main="Temperature Data")
abline(v=mean(coredata(TEMP.TS)) + (2 * sd(coredata(TEMP.TS))), col="red")
abline(v=mean(coredata(TEMP.TS)) - (2 * sd(coredata(TEMP.TS))), col="red")
```



These lines first plot the histogram of the temperature values, then add two vertical lines (function `abline`, option v for vertical), one at $mean + (2 \times SD)$ and the other at $mean - (2 \times SD)$.

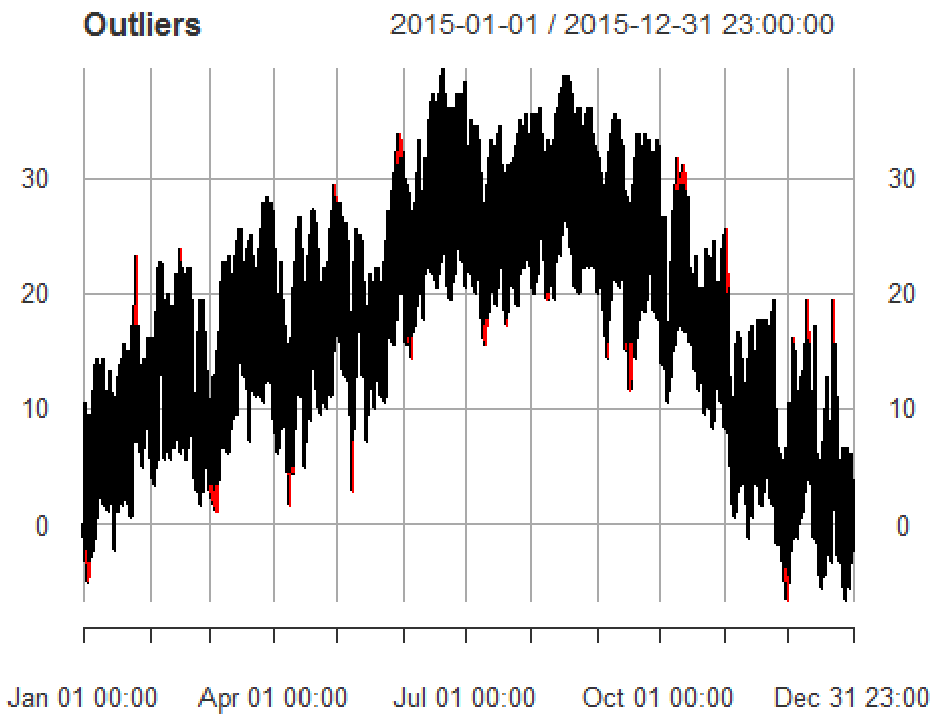
The problem with this approach is that it will take into account the whole time-series and will only exclude very high values in the summer and very low values in winter. However, these may be normal for the time of year and geographical location. We need to be more specific with our search for outliers. This require a manual monthly search, which we can do with a `for` loop:

```
OUT.TS = TEMP.TS
for(MONTH in 1:12){
  sub = OUT.TS[paste0("2015-",MONTH)]

  M = mean(sub)
  SD = sd(sub)

  V = apply(sub, 1, FUN=function(x){if(x > (M + (2 * SD)) | x < (M - (2 *
SD))){x} else {NA}})

  coredata(OUT.TS[paste0("2015-",MONTH)]) = V
}
```

We are starting by creating the object `OUT.TS`, which for the time being is simply the `TEMP.TS` time-series. We are going to use this new object to replace some of its values with `NA` in case they will be not be considered outliers. Then we start the `for` loop, where we iterate by month. Within the loop we start by subsetting the time-series (using the code we already saw at the beginning of the lecture). Then we compute both the mean and standard deviation on the subset, and store them into two objects: `M` and `SD`. At this point we can use the function apply to iterate through the subset and check whether each of the elements can be considered an outlier. As you can see, we are applying a test for which if the value is an outlier (above of below $mean \pm (2 \times SD)$) it gets removed (so it is replaced with `NA`) from the time-series.

We can check the results by plotting the new time-series (`OUT.TS`) on top of the original time-series (`TEMP.TS`):

```
plot(TEMP.TS, main="Outliers")

lines(OUT.TS, col="red")
```



Red values indicate potential outliers for each month.

---

## Conclusions

In this lecture we looked at time-series analysis. We first learned how to load time-series in R with the package `xts` and then how to perform a series of operations. In particular, we looked at trend and how to detect changes in time. Then we focused on methods for detecting anomalies. In the next and final lecture we will look at machine learning and how to do predictive analysis in R.

## References

- Time Series Analysis
- Forecasting: principles and practice - Free Book
- Metcalfe, A.V. and Cowpertwait, P.S., (2009). Introductory time series with R.
- Little Book of R for Time Series
- Time Series Analysis with R

## Homework

- Using the function `apply.monthly` try to create a plot with mean, minimum and maximum values

- Try to create a plot, using `PM10.TS`, with a horizontal line that identifies the legal limit of 150 $\frac{\mu g}{m^3}$