Energy Data Analysis with R

Reto Marek

2020-11-10

Contents

Preface

This document gives you a short overview of the statistical software R and its ability to analyze and visualize time series in the context of building energy and comfort.

This book is aimed at R beginners as well as advanced R users and is strongly inspired by the R Graphics Cookbook. The goal of this book is to additionaly provide specific recipes for energy and comfort related tasks.

The recipes in this book will show you how to complete certain specific tasks. Examples are shown so that you can understand the basic principle and reproduce the analysis or visualization with your own data.

0.1 Why R and RStudio?

Spreadsheet programs like Excel quickly reach their limits when working with large data sets or creating complex graphics. Also the interactive ability of the graphics is limited. The open source programming language R and its graphical user interface RStudio offer many more possibilities for data analysis and data visualization.

Chapter 1

Getting started

1.1 Installing R and R Studio

- Before we can start the first analysis, we have to install "R" and "RStudio".
- "R" is a programming language used for statistical computing while "RStudio" provides a graphical user interface.
- "R" may be used without "RStudio", but "RStudio" may not be used without "R". Both, "R" and "RStudio" are free of charge and there are no licencse fees.

1.1.1 Download and Install R

1.1.1.1 Windows

1. Open https://cran.r-project.org/bin/windows/base/ and press the link "Download R..."
2. Run the downloaded installer file and follow the installation wizard

The wizard will install R into your "Program Files" folders and add a shortcut in your Start menu. Note that you will need to have all necessary administration rights to install new software on your machine.

1.1.1.2 Mac OSX

1. Open https://cran.r-project.org/bin/macosx/ and download the latest *.pkg file
2. Run the downloaded installer file and follow the installation wizard

The installer allows you to customize your installation. However the default values will be suitable for most users.

1.1.1.3 Linux

R is part of many Linux distributions, therefore you should check with your Linux package management system if it's already installed.

The CRAN website provides files to build R from source on Debian, Redhat, SUSE, and Ubuntu systems under the link "Download R for Linux"

- Open https://cran.r-project.org/bin/linux/ and then follow the directory trail to the version of Linux you wish to install R on top of

The exact installation procedure will vary depending on your Linux operating system. CRAN supports the process by grouping each set of source files with documentation or README files that explain how to install on your system.

1.1.2 Download and Install RStudio

R Studio is a development environment for R.

1. Open https://rstudio.com/products/rstudio/download/ and download "RStudio Desktop Open Source"
2. Follow the on-screen instructions
3. Once you have installed R Studio, you can run it like any other application via

1.1.3 Open RStudio

Now that you have both R and RStudio on your computer, you can begin using R by opening the RStudio program. Open RStudio just as you would any program, by clicking on its icon or by typing "RStudio" at the Windows Run prompt.

1.2 Create your first R Script

blabla

1.3 Whats next?

blabla

Chapter 2

R Basics

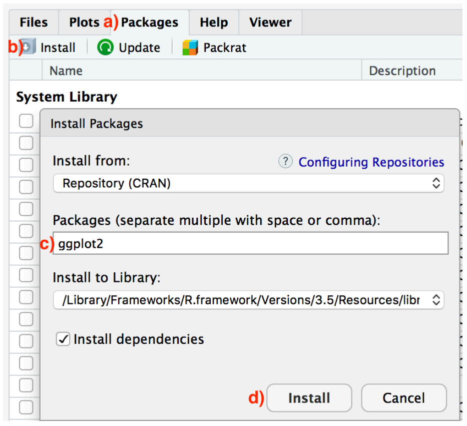
2.1 Packages in R

Many functions of R are not pre-installed and must be loaded manually. R packages are similar to libraries in C, Python etc. An R package bundles useful functions, help files and data sets. You can use these functions within your own R code once you load the package.

The following chapters describe how to install, load, update and use packages.

2.1.1 Installing a Package

The easiest way to install an R Package is to use the RStudio tab "Packages":



Install from: Repository (CRAN)

Packages (separate multiple with space or comma): c) ggplot2

Install to Library: /Library/Frameworks/R.framework/Versions/3.5/Resources/libr

✓ Install dependencies

Figure 2.1: Install packages via RStudio GUI

a) Click on the “Packages” tab

b) Click on “Install” next to Update

c) Type the name of the package under “Packages, in this case type ggplot2

d) Click “Install”

This will search for the package “ggplot” specified on a server (the so-called CRAN website). If the package exists, it will be downloaded to a library folder on your computer. Here R can access the package in future R sessions without having to reinstall it.

An other way is to use the install.packages function. Open R (if already opened please close all projects) and type the following at the command line:

```
install.packages("ggplot2")
```

If you want to install a package directly from github, the package “devtools” must be installed first:

```
install.packages("devtools")
library(devtools)
install_github("hslu-ige-laes/redutils")
```

2.1.2 Loading a Package

If you have installed a package, its functions are not yet available in your R project. To use an R package in your sript, you must load it with the following command:

```
install.packages("ggplot2")
```

2.1.3 Upgrading Packages

R packages are often constantly updated on CRAN or GitHub, so you may want to update them once in a while with:

```
update.packages(ask = FALSE)
```

2.2 Loading Data

2.2.1 Csv File

```r
df <- read.csv("datafile.csv")
df <- read.csv("datafile.csv", header=FALSE, stringsAsFactors=FALSE)
```

```r
df <- read.csv("https://github.com/retomarek/r/raw/master/datasets/buildingMonitoringTestDataSet.csv",
               stringsAsFactors=FALSE,
               sep ="," )
```

Attention: By default, strings in the data are treated as factors. read.csv() is a convenience wrapper function around read.table(). If you need more control over the input, see ?read.table

2.2.2 Excel File

```r
# Only need to install once
install.packages("xlsx")

library(xslx)

df <- read.xlsx("datafile.xlsx", 1)
df <- read.xlsx("datafile.xls", sheetIndex=2)
df <- read.xlsx("datafile.xls", sheetName="Revenues")
```

For reading older Excel files in the .xls format, the gdata package has the function read.xls():

```r
# Only need to install once
install.packages("gdata")

library(gdata)
# Read first sheet
df <- read.xls("datafile.xls")
df <- read.xls("datafile.xls", sheet=2)
```

Both the xlsx and gdata packages require other software to be installed on your computer. For xlsx, you need to install Java on your machine. For gdata, you need Perl, which comes as standard on Linux and Mac OS X, but not Windows. On Windows, you'll need ActiveState Perl. The Community Edition can be obtained for free.

Chapter 3

Data Wrangling

3.1 Add Metadata for later filtering

Firstly we have to load a dataset into a dataframe:

```r
# load data set
df <- read.csv("https://github.com/hslu-ige-laes/edar/raw/master/sampleData/centralOutsideTemp.csv",
               stringsAsFactors=FALSE,
               sep =";")
```

3.1.1 Year, Month, Day, Day of Week

To group, filter and aggregate data we need to have a the date splitted up in day, month and year separately:

```r
library(dplyr)
library(lubridate)

df$time <- parse_date_time(df$time, "YmdHMS", tz = "Europe/Zurich")
df$year <- as.Date(cut(df$time, breaks = "year"))
df$month <- as.Date(cut(df$time, breaks = "month"))
df$day <- as.Date(cut(df$time, breaks = "day"))
df$weekday <- weekdays(df$time)
```

This code first parses the timestamp with a specific timezone. Then three columns are added.

Please note that the month also contains the year and a day. This is useful for a later step where you can group the series afterwards.

```
head(df,2)
```

```
##                  time centralOutsideTemp       year      month        day
## 1 2018-03-21 11:00:00                5.2 2018-01-01 2018-03-01 2018-03-21
## 2 2018-03-21 12:00:00                6.7 2018-01-01 2018-03-01 2018-03-21
##    weekday
## 1 Mittwoch
## 2 Mittwoch
```

```
tail(df,2)
```

```
##                      time centralOutsideTemp       year      month        day
## 21864 2020-09-17 10:00:00              26.65 2020-01-01 2020-09-01 2020-09-17
## 21865 2020-09-17 11:00:00              28.10 2020-01-01 2020-09-01 2020-09-17
##          weekday
## 21864 Donnerstag
## 21865 Donnerstag
```

3.1.2 Season of Year

For some analyses it is useful to color single points of a scatterplot according to the season. For this we need to have the season in a separate column:

```
# install redutils library
# devtools::install_github("retomarek/redutils", ref = "master")

# get season from a date
redutils::season(as.Date("2019-04-01"))
```

```
## [1] "Spring"
```

If you want to change the language, you can give the function dedicated names for the season:

```
redutils::season(as.Date("2019-04-01"),
                 c("Winter","Frühling","Sommer","Herbst"))
```

```
## [1] "Frühling"
```

To apply this function to a whole dataframe we can use the dplyr mutate function. The code below creates a new column named “season”:

```
# apply it for a data frame
df <- dplyr::mutate(df, season = redutils::season(df$time))
```

```r
head(df,2)
```

```
##                  time centralOutsideTemp       year      month        day
## 1 2018-03-21 11:00:00                5.2 2018-01-01 2018-03-01 2018-03-21
## 2 2018-03-21 12:00:00                6.7 2018-01-01 2018-03-01 2018-03-21
##    weekday season
## 1 Mittwoch Spring
## 2 Mittwoch Spring
```

```r
tail(df,2)
```

```
##                      time centralOutsideTemp       year      month        day
## 21864 2020-09-17 10:00:00              26.65 2020-01-01 2020-09-01 2020-09-17
## 21865 2020-09-17 11:00:00              28.10 2020-01-01 2020-09-01 2020-09-17
##          weekday season
## 21864 Donnerstag   Fall
## 21865 Donnerstag   Fall
```

3.2 Data Frames

Firstly we have to load a dataset into a dataframe:

```r
# load data set
df <- read.csv("https://github.com/hslu-ige-laes/edar/raw/master/sampleData/flatTempHum.csv",
               stringsAsFactors=FALSE,
               sep =";")
```

3.2.1 Change Row Names

```r
# Print the header and the first line
head(df, 1)
```

```
##                  time FlatA_Hum FlatA_Temp FlatB_Hum FlatB_Temp FlatC_Hum
## 1 2018-10-03 00:00:00        53      24.43      38.8       22.4        44
##   FlatC_Temp FlatD_Hum FlatD_Temp
## 1       24.5        49      24.43
```

```r
# rename columns and print the header and the first line
names(df) <- c("timestamp","Hum_A","Temp_A","Hum_B","Temp_B","Hum_C","Temp_C","Hum_D", "Temp_D")
head(df, 1)
```

```
##             timestamp Hum_A Temp_A Hum_B Temp_B Hum_C Temp_C Hum_D Temp_D
## 1 2018-10-03 00:00:00    53  24.43  38.8   22.4    44   24.5    49  24.43
```

3.2.2 Wide to Long

```
# create a copy of the dataframe and print the header and the first five line
head(df, 5)
```

```
##             timestamp Hum_A Temp_A Hum_B Temp_B Hum_C Temp_C Hum_D Temp_D
## 1 2018-10-03 00:00:00  53.0  24.43  38.8  22.40  44.0   24.5  49.0  24.43
## 2 2018-10-03 01:00:00  53.0  24.40  38.8  22.40  44.0   24.5  49.0  24.40
## 3 2018-10-03 02:00:00  53.0  24.40  39.3  22.40  44.7   24.5  48.3  24.38
## 4 2018-10-03 03:00:00  53.0  24.40  40.3  22.40  45.0   24.5  48.0  24.33
## 5 2018-10-03 04:00:00  53.3  24.40  41.0  22.37  45.2   24.5  47.7  24.30
```

```
# convert wide to long format
df.long <- as.data.frame(tidyr::pivot_longer(df,
                                             cols = -timestamp,
                                             names_to = "sensor",
                                             values_to = "value",
                                             values_drop_na = TRUE)
                         )

# long format
head(df.long, 16)
```

```
##              timestamp sensor value
## 1  2018-10-03 00:00:00  Hum_A 53.00
## 2  2018-10-03 00:00:00 Temp_A 24.43
## 3  2018-10-03 00:00:00  Hum_B 38.80
## 4  2018-10-03 00:00:00 Temp_B 22.40
## 5  2018-10-03 00:00:00  Hum_C 44.00
## 6  2018-10-03 00:00:00 Temp_C 24.50
## 7  2018-10-03 00:00:00  Hum_D 49.00
## 8  2018-10-03 00:00:00 Temp_D 24.43
## 9  2018-10-03 01:00:00  Hum_A 53.00
## 10 2018-10-03 01:00:00 Temp_A 24.40
## 11 2018-10-03 01:00:00  Hum_B 38.80
## 12 2018-10-03 01:00:00 Temp_B 22.40
## 13 2018-10-03 01:00:00  Hum_C 44.00
## 14 2018-10-03 01:00:00 Temp_C 24.50
## 15 2018-10-03 01:00:00  Hum_D 49.00
## 16 2018-10-03 01:00:00 Temp_D 24.40
```

3.2.3 Long to Wide

```
# long format
head(df.long)
```

```
##             timestamp sensor value
## 1 2018-10-03 00:00:00  Hum_A 53.00
## 2 2018-10-03 00:00:00 Temp_A 24.43
## 3 2018-10-03 00:00:00  Hum_B 38.80
## 4 2018-10-03 00:00:00 Temp_B 22.40
## 5 2018-10-03 00:00:00  Hum_C 44.00
## 6 2018-10-03 00:00:00 Temp_C 24.50
```

```
# convert long table into wide table
df.wide <- as.data.frame(tidyr::pivot_wider(df.long,
                                            names_from = "sensor",
                                            values_from = "value")
                         )

# wide format
head(df.wide)
```

```
##             timestamp Hum_A Temp_A Hum_B Temp_B Hum_C Temp_C Hum_D Temp_D
## 1 2018-10-03 00:00:00  53.0  24.43  38.8  22.40  44.0  24.50  49.0  24.43
## 2 2018-10-03 01:00:00  53.0  24.40  38.8  22.40  44.0  24.50  49.0  24.40
## 3 2018-10-03 02:00:00  53.0  24.40  39.3  22.40  44.7  24.50  48.3  24.38
## 4 2018-10-03 03:00:00  53.0  24.40  40.3  22.40  45.0  24.50  48.0  24.33
## 5 2018-10-03 04:00:00  53.3  24.40  41.0  22.37  45.2  24.50  47.7  24.30
## 6 2018-10-03 05:00:00  53.7  24.40  41.2  22.30  47.2  24.57  47.2  24.30
```

3.2.4 Merge two Dataframes

```
library(dplyr)
library(lubridate)

# read file one and parse dates
dfOutsideTemp <- read.csv("https://github.com/hslu-ige-laes/edar/raw/master/sampleData/centralOutsideTemp.csv",
                          stringsAsFactors=FALSE,
                          sep =";")

dfOutsideTemp$time <- parse_date_time(dfOutsideTemp$time,
                                      orders = "YmdHMS",
                                      tz = "Europe/Zurich")

# read file two and parse dates
dfFlatTempHum <- read.csv("https://github.com/hslu-ige-laes/edar/raw/master/sampleData/flatTempHum.csv",
                          stringsAsFactors=FALSE, sep =";")

dfFlatTempHum$time <- parse_date_time(dfFlatTempHum$time,
                                      order = "YmdHMS",
                                      tz = "Europe/Zurich")

# merge the two files into a new data frame and keep only rows where all values are available
df <- merge(dfOutsideTemp, dfFlatTempHum, by = "time") %>% na.omit()
```

Chapter 4

Explorative Data Analysis

4.1 Get overview

Get an overview of the whole data set and specific series of it

4.1.1 Load data

Load test data set in a data frame (e.g. from a csv-file)

```
df <- read.csv("https://github.com/retomarek/r/raw/master/datasets/buildingMonitoringTestDataSet.csv",
               stringsAsFactors=FALSE,
               sep ="," )
```

4.1.2 Names

show the column headers of the data frame

```
names(df)
```

```
##  [1] "time"                  "WthStnPress"           "WthStnHum"
##  [4] "WthStnRain"            "WthStnSolRad"          "WthStnTemp"
##  [7] "WthStnWindDir"         "WthStnWindSpd"         "BldgEnergyHotwater"
## [10] "BldgEnergyHeating"     "FlatHum"               "FlatTemp"
## [13] "FlatVolFlowColdwater"  "FlatVolFlowHotwater"
```

4.1.3 Structure

show the structure of the data frame

```
str(df)
```

```
## 'data.frame':    16394 obs. of  14 variables:
##  $ time                : chr  "2018-09-30T22:00:00.000Z" "2018-09-30T23:00:00.000Z" "2018-10-01T00:00:00
##  $ WthStnPress         : num  1012 1012 1011 1011 1011 ...
##  $ WthStnHum           : num  87 87.5 87.5 86.5 88 89 86.5 81 78 80.5 ...
##  $ WthStnRain          : num  0.8 1.1 0.5 0.5 0.6 0.1 0.2 0 0 0 ...
##  $ WthStnSolRad        : num  0 0 0 0 0 0 0 0 3 24.5 ...
##  $ WthStnTemp          : num  12.8 12.4 11.9 11.9 11.6 ...
##  $ WthStnWindDir       : num  157.5 11.2 146.2 157.5 146.2 ...
##  $ WthStnWindSpd       : num  3.2 1.6 2.4 0.8 2.4 0.8 0.8 3.2 4 3.2 ...
##  $ BldgEnergyHotwater  : num  0 19 0 0 0 ...
##  $ BldgEnergyHeating   : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ FlatHum             : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ FlatTemp            : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ FlatVolFlowColdwater: num  0.006 0 0 0 0.006 ...
##  $ FlatVolFlowHotwater : num  0 0 0 0 0 ...
```

4.1.4 Head/Tail

The head and tail functions are generic, so they will work whether your data is stored in a simple data frane, a zoo object, or an xts object.

```
head(df)
```

```
##                       time WthStnPress WthStnHum WthStnRain WthStnSolRad
## 1 2018-09-30T22:00:00.000Z     1012.30      87.0        0.8            0
## 2 2018-09-30T23:00:00.000Z     1011.90      87.5        1.1            0
## 3 2018-10-01T00:00:00.000Z     1011.45      87.5        0.5            0
## 4 2018-10-01T01:00:00.000Z     1010.90      86.5        0.5            0
## 5 2018-10-01T02:00:00.000Z     1010.55      88.0        0.6            0
## 6 2018-10-01T03:00:00.000Z     1010.20      89.0        0.1            0
##   WthStnTemp WthStnWindDir WthStnWindSpd BldgEnergyHotwater BldgEnergyHeating
## 1      12.80        157.50           3.2                  0                 0
## 2      12.35         11.25           1.6                 19                 0
## 3      11.90        146.25           2.4                  0                 0
## 4      11.90        157.50           0.8                  0                 0
## 5      11.60        146.25           2.4                  0                 0
## 6      11.75         22.50           0.8                  0                 0
##   FlatHum FlatTemp FlatVolFlowColdwater FlatVolFlowHotwater
## 1      NA       NA                0.006                   0
## 2      NA       NA                0.000                   0
## 3      NA       NA                0.000                   0
## 4      NA       NA                0.000                   0
## 5      NA       NA                0.006                   0
## 6      NA       NA                0.000                   0
```

```
tail(df)
```

```
##                           time WthStnPress WthStnHum WthStnRain WthStnSolRad
## 16389 2020-08-13T18:00:00.000Z    1011.650     74.75    2.19964            9
## 16390 2020-08-13T19:00:00.000Z    1012.000     79.00    2.19964            0
## 16391 2020-08-13T20:00:00.000Z    1011.950     78.25    2.19964            0
## 16392 2020-08-13T21:00:00.000Z    1012.025     76.50    2.19964            0
## 16393 2020-08-13T22:00:00.000Z    1012.250     73.00    0.00000            0
## 16394 2020-08-13T23:00:00.000Z          NA        NA         NA           NA
##       WthStnTemp WthStnWindDir WthStnWindSpd BldgEnergyHotwater
## 16389     22.000        162.00      0.000000                 NA
## 16390     20.175        124.25      1.609340                 NA
## 16391     19.350        125.00      0.402335                 NA
## 16392     19.900         93.00      1.609340                 NA
## 16393     20.625        116.25      2.414010                 NA
## 16394         NA            NA            NA                 NA
##       BldgEnergyHeating FlatHum FlatTemp FlatVolFlowColdwater
## 16389                NA      NA       NA                   NA
## 16390                NA      NA       NA                   NA
## 16391                NA      NA       NA                   NA
## 16392                NA      NA       NA                   NA
## 16393                NA      NA       NA                   NA
## 16394                NA      NA       NA                   NA
##       FlatVolFlowHotwater
## 16389                  NA
## 16390                  NA
## 16391                  NA
## 16392                  NA
## 16393                  NA
## 16394                  NA
```

4.1.5 Five number summary

reveals details of a specific series

```
summary(df$WthStnTemp)
```

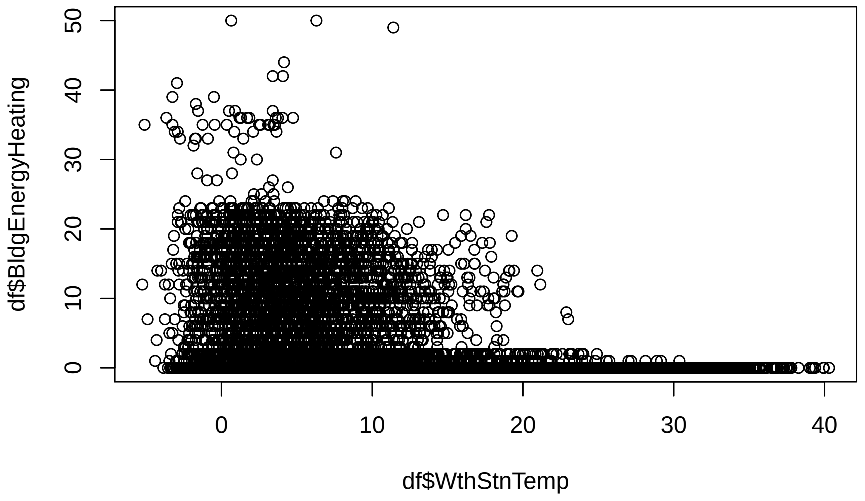
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's 
##   -5.25    5.50   11.25   11.99   17.35   40.30      12
```

4.2 Basic plots

4.2.1 Scatterplot

4.2.1.1 plot()

```
# load data set
df <- read.csv("https://github.com/retomarek/r/raw/master/datasets/buildingMonitoringTestDataSet.csv",
               stringsAsFactors=FALSE,
               sep =",")

# crate simple scatterplot
plot(df$WthStnTemp, df$BldgEnergyHeating)
```



df$BldgEnergyHeating
0
10
20
30
40
50
0
10
20
30
40
df$WthStnTemp

Chapter 5

Data Visualizations

5.1 Room Temperature Reduction

5.1.1 Task

As part of an energy optimization, you lower the room temperatures in a room and would now like to show the reduction effect using the time series of the room temperature sensor. In the example below you make two optimizations at different dates.

You want to create a time series plot with

- the daily median, min and max value
- the overall median of each period
- the desired setpoint

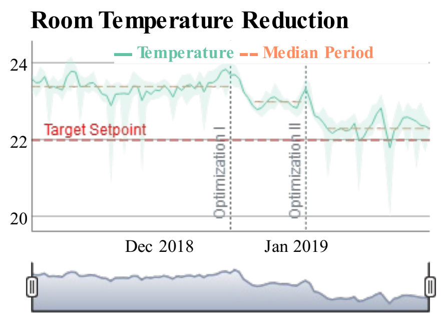
5.1.2 Basis

- Time series data from e.g. a temperature sensor with unaligned time intervals

5.1.3 Solution

```
library(dplyr)
library(lubridate)
library(dygraphs)
```

```r
library(xts)
library(redutils)
library(RColorBrewer)

# Settings
tempSetpoint = 22.0

startDate = "2018-11-01"
endDate = "2019-02-01"

optiDate1 = "2018-12-17"
optiLabel1 = "Optimization I"

optiDate2 = "2019-01-03"
optiLabel2 = "Optimization II"

optiDelayDays = 5

# read and print data
df <- read.csv("https://github.com/hslu-ige-laes/edar/raw/master/sampleData/flatTempHum.csv",
                 stringsAsFactors=FALSE,
                 sep =";")

# select temperature and remove empty cells
df <- df %>% select(time, FlatA_Temp) %>% na.omit()

# create column with day for later grouping
df$time <- parse_date_time(df$time, "YmdHMS", tz = "Europe/Zurich")
df$day <- as.Date(cut(df$time, breaks = "day"))
df$day <- as.Date(as.character(df$day,"%Y-%m-%d"))

# filter time range
df <- df %>% filter(day > startDate, day < endDate)

# calculate daily median, min and max of temperature
df <- df %>%
  group_by(day) %>%
  mutate(minDay = min(as.numeric(FlatA_Temp)),
         medianDay = median(as.numeric(FlatA_Temp)),
         maxDay = max(as.numeric(FlatA_Temp))
         ) %>%
  ungroup()

# shrink down to daily values and remove rows with empty values
df <- df %>% select(day, medianDay, minDay, maxDay) %>% unique() %>% na.omit()

# calculate medians for time ranges
df <- df %>%
  mutate(period = ifelse(day >= startDate & day <= optiDate1,
                         "Baseline",
                         ifelse((day >= (as.Date(optiDate1) + optiDelayDays))
                                & (day <= optiDate2),
                                "Opti1",
                                ifelse((day >= (as.Date(optiDate2) + optiDelayDays))
                                & (day <= endDate),
                                "Opti2",
                                NA)
                         )))

df <- df %>%
  group_by(period) %>%
  mutate(medianPeriod = ifelse(is.na(period), NA, median(medianDay))) %>%
  ungroup() %>%
  select(-period)

# create xts object for plotting
plotdata <- xts( x=df[,-1], order.by=df$day)

# plot graph
dygraph(plotdata, main = "Room Temperature Reduction") %>%
  dyAxis("x", drawGrid = FALSE) %>%
  dySeries(c("minDay", "medianDay", "maxDay"),
           label = "Temperature") %>%
  dySeries(c("medianPeriod"),
           label = "Median Period",
           strokePattern = "dashed") %>%
  dyOptions(colors = RColorBrewer::brewer.pal(3, "Set2")) %>%
  dyEvent(x = optiDate1,
          label = optiLabel1,
          labelLoc = "bottom",
          color = "slategray",
          strokePattern = "dotted") %>%
  dyEvent(x = optiDate2,
          label = optiLabel2,
          labelLoc = "bottom",
          color = "slategray",
          strokePattern = "dotted") %>%
  dyLimit(tempSetpoint,
          color = "red",
          label = "Target Setpoint") %>%
  dyRangeSelector() %>%
  dyLegend(show = "always")
```



5.1.4 Discussion

In this example we used the dygraph package to create the graph. This package is fast and allows to show a rangeslider on the bottom of the graph. The exact same graph but without a slider is as well possible with ggplot.

Please note that the calculation of the periodic median after optimization I and II starts delayed because it takes time until the building has cooled down.

5.2 Building Energy Signature

5.2.1 Task

You want to create a scatter plot with

- the daily mean outside temperature on the x-axis
- the daily energy consumption on the y-axis
- points colored according to season

5.2.2 Basis

- Two separate csv files with time series data from the outside temperature and the energy data with unaligned time intervals
- Energy consumption time series from a energy meter with steadily increasing meter values

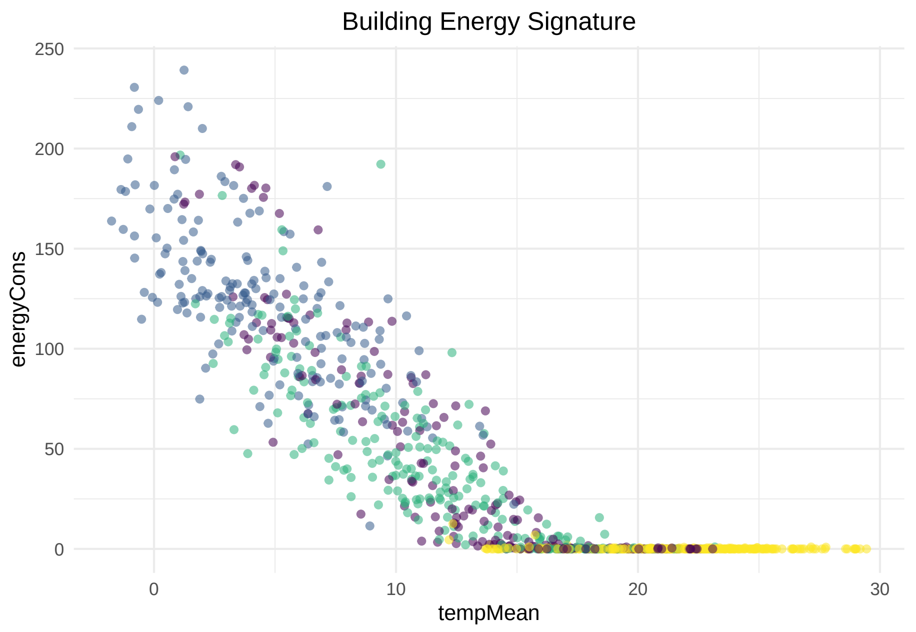
5.2.3 Solution

After reading in the two time series the data has to get aggregated per day and then merged. Note that during the aggregation of the energy data you have to calculate the daily conspumption from the steadiliy increasing meter values as well.

Create a new script, copy/paste the following code and run it:

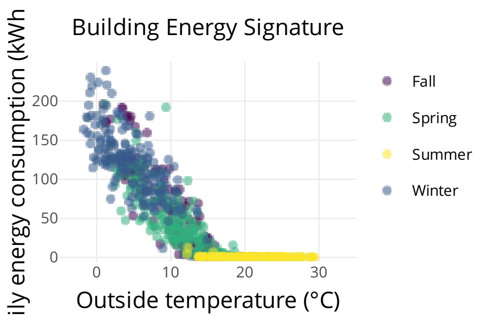
```
library(ggplot2)
library(plotly)
library(dplyr)
library(redutils)
library(lubridate)

# load time series data and aggregate daily mean values
dfOutsideTemp <- read.csv("https://github.com/hslu-ige-laes/edar/raw/master/sampleData/centralOutsideTemp.csv",
                          stringsAsFactors=FALSE,
                          sep =";")

dfOutsideTemp$time <- parse_date_time(dfOutsideTemp$time,
                                      order = "YmdHMS",
                                      tz = "Europe/Zurich")

dfOutsideTemp$day <- as.Date(cut(dfOutsideTemp$time, breaks = "day"))

dfOutsideTemp <- dfOutsideTemp %>%
  group_by(day) %>%
  mutate(tempMean = mean(centralOutsideTemp)) %>%
  ungroup()

dfOutsideTemp <- dfOutsideTemp %>%
  select(day, tempMean) %>%
  unique() %>%
  na.omit()

dfHeatEnergy <- read.csv("https://github.com/hslu-ige-laes/edar/raw/master/sampleData/centralHeating.csv",
                         stringsAsFactors=FALSE,
                         sep =";")

dfHeatEnergy <- dfHeatEnergy %>%
  select(time, energyHeatingMeter) %>%
  na.omit()

dfHeatEnergy$time <- parse_date_time(dfHeatEnergy$time,
```

```r
                                      orders = "YmdHMS",
                                      tz = "Europe/Zurich")

dfHeatEnergy$day <- as.Date(cut(dfHeatEnergy$time, breaks = "day"))

dfHeatEnergy <- dfHeatEnergy %>%
  group_by(day) %>%
  mutate(energyMax = max(energyHeatingMeter)) %>%
  ungroup()

dfHeatEnergy <- dfHeatEnergy %>%
  select(day, energyMax) %>%
  unique() %>%
  na.omit()

dfHeatEnergy <- dfHeatEnergy %>%
  mutate(energyCons = energyMax - lag(energyMax)) %>%
  select(-energyMax) %>%
  na.omit()

# merge the data in a tidy format
df <- merge(dfOutsideTemp, dfHeatEnergy, by = "day")

# calculate season
df <- df %>% mutate(season = redutils::season(df$day))

# static chart with ggplot
p <- ggplot2::ggplot(df) +
  ggplot2::geom_point(aes(x = tempMean,
                          y = energyCons,
                          color = season,
                          alpha = 0.1,
                          text = paste("</br>Date:  ", as.Date(df$day),
                                       "</br>Temp: ", round(df$tempMean, digits = 1), "\u00B0C",
                                       "</br>Energy: ", round(df$energyCons, digits = 0), "kWh/d",
                                       "</br>Season: ", df$season))
                       ) +
  scale_color_manual(values=c("#440154", "#2db27d", "#fde725", "#365c8d")) +
  ggtitle("Building Energy Signature") +
        theme_minimal() +
        theme(
          legend.position="none",
          plot.title = element_text(hjust = 0.5)
        )
p
```



Add the following part to your script to make the chart above interactive:

```
# continuation from upper ggplot code section
plotly::ggplotly(p, tooltip = c("text")) %>%
  layout(xaxis = list(title = "Outside temperature (\u00B0C)",
                      range = c(min(-5,min(df$tempMean)), max(35,max(df$tempMean))), zeroline = F),
         yaxis = list(title = "Daily energy consumption (kWh/d)",
                      range = c(-5, max(df$energyCons) + 10)),
         showlegend = TRUE
         ) %>%
  plotly::config(displayModeBar = FALSE, displaylogo = FALSE)
```



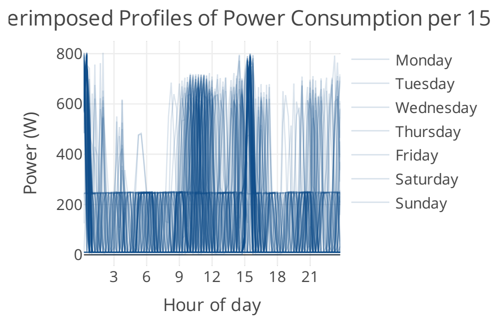
5.3 Superimposed Daily Mean Profiles

```
library(plotly)
library(dplyr)
library(lubridate)

# load time series data
df <- read.csv("https://github.com/hslu-ige-laes/edar/raw/master/sampleData/heatPumpElectricity.csv",
               stringsAsFactors=FALSE,
               sep =";")

# rename column names
colnames(df) <- c("timestamp", "meterValue")

df$timestamp <- parse_date_time(df$timestamp,
                                orders = "YmdHMS",
                                tz = "Europe/Zurich")
df$timestamp <- force_tz(df$timestamp, tzone = "UTC")

# filter time range if necessary
df <- df %>% filter(timestamp > "2019-11-01 00:00:00", timestamp < "2020-04-01 00:00:00")

# Fill missing values with NA
grid.df <- data.frame(timestamp = seq(df[1, 1], df[nrow(df), 1], by = "15 mins"))
df <- merge(df, grid.df, all = TRUE)
```

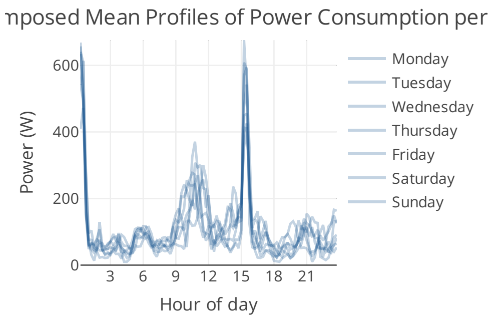
```r
# convert steadily counting energy meter value in kWh to power in Watts
df <- df %>%
  mutate(value = (meterValue - lag(meterValue))/4*1000) %>%
  select(-meterValue) %>%
  na.omit()

# change language to English, otherwise weekdays are in local language
Sys.setlocale("LC_TIME", "English")
```

```
## [1] "English_United States.1252"
```

```r
# add metadata for later grouping and visualization purposes
df$x <- hour(df$timestamp) + minute(df$timestamp)/60 + second(df$timestamp) / 3600
# df$time <- format(df$timestamp, "%H:%M")
df$time <- format(as.POSIXct(strptime(df$timestamp, "%Y-%m-%d  %H:%M:%S",tz="")) ,format = "%H:%M")
df$time <- as.POSIXct(df$time, format = "%H:%M")
df$weekday <- weekdays(df$timestamp)
df$weekday <- factor(df$weekday, c("Monday","Tuesday","Wednesday","Thursday","Friday","Saturday", "Sunday"))
# df$day <- as.Date(cut(df$timestamp, breaks = "day"))
df$day <- as.Date(df$timestamp, format = "%Y-%m-%d  %H:%M:%S")

df <- df %>% mutate(value = ifelse(x == 0.00, NA, df$value))


# Test
# ==================================================================================
# plot graph with all time series
df %>%
  highlight_key(~day) %>%
  plot_ly(x=~x,
          y=~value,
          color=~weekday,
          type="scatter",
          mode="lines",
          line = list(width = 1),
          alpha = 0.15,
          colors = "dodgerblue4",
          text = ~day,
          hovertemplate = paste("Time: ", format(df$time, "%H:%M"),
                                "<br>Date: ", format(df$time, "%Y-%m-%d"),
                                "<br>Mean: %{y:.0f}")) %>%
  layout(title = "Superimposed Profiles of Power Consumption per 15 min",
         showlegend = TRUE,
         xaxis = list(
           title = "Hour of day",
           tickvals = list(0, 3, 6, 9, 12, 15, 18, 21)
           ),
           yaxis = list(
             title = "Power (W)",
             range = c(0, max(df$value))
           )
         ) %>%
  highlight(on = "plotly_hover",
            off = "plotly_doubleclick",
            color = "orange",
             opacityDim = 1.0) %>%
  plotly::config(modeBarButtons = list(list("toImage")), displaylogo = FALSE)
```



```
# Calculate Mean value for all 15 minutes for each weekday
df <- df %>% group_by(weekday, x) %>% mutate(dayTimeMean = mean(value)) %>% ungroup()

# shrink data frame
df <- df %>%
  select(x, weekday, time, dayTimeMean) %>%
  unique() %>%
  na.omit() %>%
  arrange(weekday, x)

# plot graph with mean values
df %>%
  highlight_key(~weekday) %>%
  plot_ly(x=~x,
          y=~dayTimeMean,
          color=~weekday,
          type="scatter",
          mode="lines",
          alpha = 0.25,
          colors = "dodgerblue4",
          text = ~weekday,
          hovertemplate = paste("Time: ", format(df$time, "%H:%M"),
                                "<br>Mean: %{y:.0f}")) %>%
  layout(title = "Superimposed Mean Profiles of Power Consumption per 15 min",
         showlegend = TRUE,
        xaxis = list(
          title = "Hour of day",
          tickvals = list(0, 3, 6, 9, 12, 15, 18, 21)
        ),
        yaxis = list(
            title = "Power (W)",
            range = c(0, max(df$dayTimeMean))
        )
) %>%
highlight(on = "plotly_hover",
          off = "plotly_doubleclick",
          color = "orange",
          opacityDim = 0.7) %>%
plotly::config(modeBarButtons = list(list("toImage")), displaylogo = FALSE)
```



5.4 Mollier hx Diagram

5.4.1 Task

You want to plot a mollier h-x diagram with

- scatter plot of temperature- and humidity sensor data (mean values per day)

- points colored according to season
- comfort zone

5.4.2 Basis

- A csv file with time series from multiple temperature and humidity sensors in °C and %rH

5.4.3 Solution

The sensor data is not in a constant intervall and not yet aggregated. So after reading in the time series the data has to get filtered and aggregated per day.

Finally use the plot function `mollierHxDiagram` from the `redutils` package (R Energy Data Utilities) which you can install as followed:

```r
install.packages("devtools")
library(devtools)
install_github("hslu-ige-laes/redutils")
```

Create a new script, copy/paste the following code and run it:

```r
library(redutils)
library(dplyr)
library(r2d3)
library(lubridate)

# read and print data
data <- read.csv("https://github.com/hslu-ige-laes/edar/raw/master/sampleData/flatTempHum.csv",
                 stringsAsFactors=FALSE,
                 sep =";")

# select temperature and humidity and remove empty cells
data <- data %>% select(time, FlatA_Temp, FlatA_Hum) %>% na.omit()

# create column with day for later grouping
data$time <- parse_date_time(data$time, "YmdHMS", tz = "Europe/Zurich")
data$day <- as.Date(cut(data$time, breaks = "day"))

# calculate daily mean of temperature and humidity
data <- data %>%
  group_by(day) %>%
  mutate(tempMean = mean(as.numeric(FlatA_Temp)),
         humMean = mean(as.numeric(FlatA_Hum))
         ) %>%
  ungroup()

# shrink down to daily values and remove rows with empty values
data <- data %>% select(day, tempMean, humMean) %>% unique() %>% na.omit()
```

```r
# plot mollier hx diagram
redutils::mollierHxDiagram(data)
```

5.4.4 Discussion

The diagram is based on D3 and packaged into the package `redutils`. The original D3 source with a html integration you can find here: https://github.com/hslu-ige-laes/d3-mollierhx

5.4.5 See Also

If your two time series are in separate files, you must first read them in separately and then merge them into one data frame. See chapter 3.2.4