Energy Data Analysis with R

Reto Marek

2021-07-30

Contents

Chapter 1

Preface

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

The aim of this book is to provide additional specific recipes for energy and comfort related tasks and to make your entry into R smooth and easy.

It is aimed at R beginners as well as exerienced R users and is inspired by the R Graphics Cookbook and Engineering Data Analysis in R.

1.1 Content

The book is structured in the following main parts:

1. R Basics
2. Data Visualizations

The part "R Basics" covers general data analysis tasks like data loading, wrangling and aggregation. Mostly this part deals with number juggling and table viewing.

> "Numerical quantities focus on expected values, graphical summaries on unexpected values." - John W. Tukey

Such so called "graphical summaries" are covered in the part "Data Visualizations" which brings the calculated numbers to life. Visualizations are good to identify patterns, changes over time, unusual readings, and relationships between variables. The presented code makes the creation of common and useful

plots for energy and comfort data fast and easy. The recipes in this part 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. Simply copy the code into your R-script, run it and if you like the visualization replace the sample data with your own.

1.2 Why R and RStudio?

In the EVISUstudy of the Lucerne University of Applied Sciences and Arts, experts from the field were asked how and where they perform energy analyses and create visualizations. The result was that many people today either need Excel or use a building monitoring software to execute analysis and as well for making visualizations.

Excel users are pushing the program to its limits with the ever-increasing data sets. Also the interactive ability of the graphics there is limited. Switching to an analysis environment like "R" seems unavoidable for more complex tasks, but apparently causes problems for many people. In the market there are numerous books which make the start in "R" easier. There are also packages for various fields which support the discipline-specific analysis and visualization tasks.

However, experts from the energy and building services engineering industry lack a corresponding work and corresponding packages. The present book is intended to close the first gap, the second gets closed by the package redutils - Energy Data Utilities for R.

You might also wonder why R and not Python. Well, this is a question of faith and the author seems to prefer R.

1.3 Further Reading

A really good source is R for Data Science by Garrett Grolemund and Hadley Wickham. The entire book is freely available online through the same format of this book.

There are a number of other recommendable and free online books available, including:

- R Graphics Cookbook by Winston Chang
- Introduction to Data Science - Data Analysis and Prediction Algorithms with R by Rafael A. Irizarry
- Hands-On Programming with R by Garrett Grolemund
- Engineering Data Analysis in R by John Volckens and Kathleen E. Wendt
- Forecasting: Principles and Practice by Rob J Hyndman and George Athanasopoulos

1.4 Acknowledgements

The author would like to express his sincere thank to the Swiss Federal Office of Energy, as the launch of this book was part of a project of the research program "Buildings and Cities 2018".

This book was developed using Yihui Xie's bookdown framework. The book is built using code that combines R code, data, and text to create a book for which R code and examples can be re-executed every time the book is re-built.

The online book is hosted using GitHub's free GitHub Pages. All material for this book is available and can be explored at the book's GitHub repository.

Chapter 2

Getting started

Before we can start with the analysis described in the later chapters, we have to install "R". Therefore the first two parts of this chapter get you up and running with downloading and installing the relevant software and packages.

This may seem laborious, but it is necessary and easier than it appears at first glance.

If you already have "R" and "R Studio" installed, please take a look at the redutils package on github, which is frequently used throughout this book. Chapter 2.2 shows you how to install it.

2.1 Install R and R Studio

- "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
- When you later make an analysis and visualizations, you only work in the graphical user interface "RStudio"

2.1.1 Download and Install R

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 adds a shortcut in your Start menu. Note that you will need to have all necessary administration rights to install new software on your machine.

Mac OSX 1. Open https://cran.r-project.org/bin/macosx/ and download the latest `*.pkg` file 1. 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.

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.

2.1.2 Download and Install RStudio

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 by clicking the program icon

2.2 Install required packages

Appendix A gives you an introduction to what a package is and how to install it.

Fo.low these instructions to install the packages used in this book:

- Open "RStudio" just as you would any program, by clicking on its icon
- Copy the following code and paste it into your console (on the bottom left, right of the symbol `>`):

```
install.packages("devtools", "tidyverse", "plotly", "lubridate", "r2d3")
install_github("hslu-ige-laes/redutils")
```

- Press `Enter` or `Return`

The installation of the packages is now in progress and this may take a while, please be patient. In the meantime you can read in appendix A what packages are in general and how they can be installed and later loaded into scripts.

2.3 Create your first R Script

Finally, you have installed "R" and "RStudio" with some packages on your computer. Good, hopefully everything worked fine up to now.

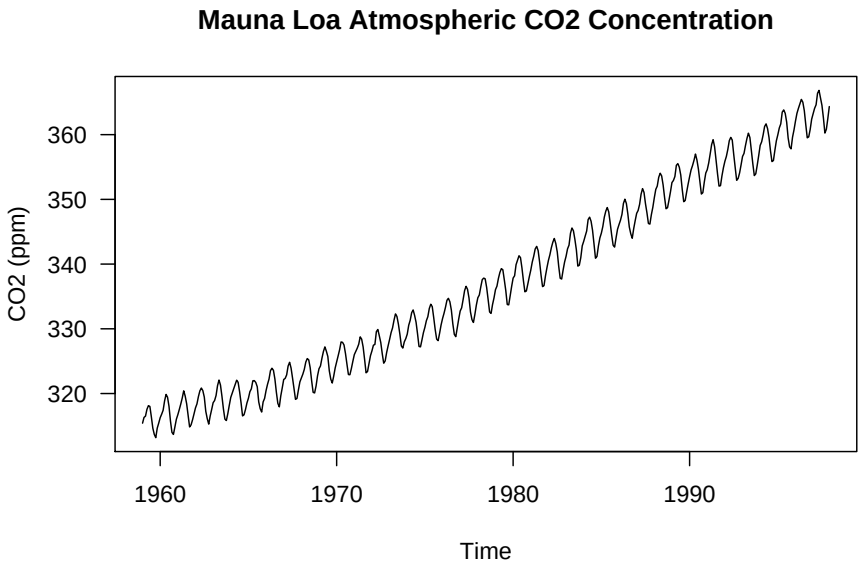
Let's create the first script with a visualization:

- Open "RStudio" just as you would any program, by clicking on its icon
- Go to the menu on the top left and click to `File / New File / R Script`
- Copy the following code and paste it into your script:

```
library(graphics)
plot(co2, ylab = "CO2 (ppm)", las = 1)
title(main = "Mauna Loa Atmospheric CO2 Concentration")
```

- select all by pressing `Ctrl + A`
- Thren run the code by pressing the `Run` Button or `Ctrl + Enter`

You should now get your first visualization:



As you probably noticed, we did not load any data. The basic installation of "R" and some packages come with test data. So that is an easy way to test something. The R Dataset Package provides some preinstalled datasets, including the used "Mauna Loa Atmospheric CO2 Concentration" dataset.

Part I

R Basics

Chapter 3

Introduction to R Basics

The following chapters of the part "R-Basics" give an overview of the basic principles for understanding the recipes in the part "Data Visualizations". It focuses on practical examples of how data can be loaded, transformed and analyzed.

It is not the goal of this book to introduce beginners completely to the programming language "R" and the environment in general. There are many really good sources where you can familiarize yourself with "R" if necessary.

Two are recommended to facilitate the introduction to "R" and to learn the relevant basics. Choose one that suits you and go through the recommended chapter:

- Introduction to Data Science - Chapter 2 "R Basics"
- Engineering Data Analysis in R - Chapter 2 "The R Programming Environment"

Throughout the book, reference is made to these two continuative sources and you only need to read the one that appeals to you more. In the referenced chapters, these cover similar content.

Chapter 4

Loading Data

This chapter introduces two functions that can be used to load data from csv- and Excel-files.

Experienced readers will find more information about data imports here:

- Introduction to Data Science - Chapter 5 “Importing data”
- Engineering Data Analysis in R - Chapter 3 “Getting and Cleaning Data”

csv-Files

Load data from comma separated files

```
# read data from current folder
df <- read.csv("datafile.csv")

# read data from a specific folder
df <- read.csv("C:/Desktop/datafile.csv")

# read data from a file in the internet
df <- read.csv2("https://github.com/hslu-ige-laes/edar/raw/master/sampleData/flatElectricity.csv")

# add arguments on how to parse the content
df <- read.csv("datafile.csv",
               header=FALSE,
               stringsAsFactors=FALSE,
               sep =",",
               na.strings = c("", "NA"))
```

> Attention: By default, strings in the data are treated as factors, so add `stringsAsFactors=FALSE`

Set your cursor to the function name `read.csv()` and press F1. Then you get in R Studio bottom right in the tab `Help` information about other function arguments you can use.

Note that the function read.csv() has the default `sep =","` and read.csv2() has the default `sep =";"`

Excel-Files

Load data from `*.xlsx` Excel files:

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

library(xslx)

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

# show the first lines of the so called "data frame"
head(df)
```

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

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

library(gdata)

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

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 5

Data Wrangling

This chapter gives an overview of the most important data manipulation functions used throughout this book.

Experienced readers will find more information about data imports here

- Introduction to Data Science - Chapter 4 “The tidyverse”
- Engineering Data Analysis in R - Chapter 3.5 “Data Cleaning”
- Engineering Data Analysis in R - Chapter 3.6 “Piping”

It is a fact that the data import and manipulation part of analyses often takes more time than the actual analysis or visualization itself. This is because the exchange data formats are not standardized and meters and sensors record at different time intervals. Data quality, missing data, data imputation and data cleansing also play a major role.

5.1 Add Metadata for later filtering

Firstly we have to load a dataset into a dataframe:

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

5.1.1 Year, Month, Day, Day of Week

To e.g. group, filter and aggregate data we need eventually the date splitted up in day, month and year:

```
library(dplyr)
library(lubridate)

 %>% %>% %>% %>% 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$weekday <- wday(df$time,
                   label = TRUE,
                   locale = "English",
                   abbr = TRUE,
                   week_start = getOption("lubridate.week.start", 1))
```

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
## 1 2018-03-21 11:00:00                5.2
## 2 2018-03-21 12:00:00                6.7
```

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

```
##                      time centralOutsideTemp
## 21864 2020-09-17 10:00:00              26.65
## 21865 2020-09-17 11:00:00              28.10
```

5.1.2 Hour, Minute, Second

```
df$hour <- as.POSIXct(lubridate::floor_date(df$time,"hours"), tz = "Europe/Zurich")
df$minute <- as.POSIXct(lubridate::floor_date(df$time,"minutes"), tz = "Europe/Zurich")
df$second <- as.POSIXct(lubridate::floor_date(df$time,"seconds"), tz = "Europe/Zurich")
```

5.1.3 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:

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

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

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

```
getSeason(as.Date("2019-04-01"),
                  seasonlab = 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":

```
df <- dplyr::mutate(df, season = getSeason(df$time))
```

```
head(df,1)
```

```
##                  time centralOutsideTemp season
## 1 2018-03-21 11:00:00                5.2 Spring
```

```
tail(df,1)
```

```
##                      time centralOutsideTemp season
## 21865 2020-09-17 11:00:00               28.1 Summer
```

5.2 Manipulating Data Frames

The `dplyr` package from the `tidyverse` introduces functions that perform some of the most common operations when working with data frames and uses names for these functions that are relatively easy to remember.

5.2.1 Change Row Names

```
# rename columns
names(df) <- c("timestamp","humidity","temp_c")
```

5.2.2 Adding a column with `mutate()`

```
library(dplyr)

# add new column with temperature conversion from celsius to fahrenheit
df <- dplyr::mutate(df, temp_f = temp_c * 1.8 + 32)

# This code does the same, but only with a pipe
df <- df %>%
  dplyr::mutate(df, temp_f = temp_c * 1.8 + 32)
```

5.2.3 Subsetting with `filter()`

```
library(dplyr)

# add new column with temperature conversion from celsius to fahrenheit
df <- filter(df,
             timestamp >= "2020-01-01 00:00:00",
             timestamp <= "2020-12-31 23:45:00")

df.high <- filter(df, temp_c > 30)
```

> Note: Whether you put the whole code on one line or split it after a comma does not have an effect on the computation, it is only more readable when the lines aren't too wide.

5.2.4 Add/remove columns with `select()`

```
library(dplyr)

# Based on the upper example we remove the celsius column after calculation
df <- select(df, -temp_c)

# Create new data frame
df.new <- select(df, timestamp, temp_f)

# use select() in a so called dplyr pipe
df <- df %>%
  dplyr::mutate(df, temp_f = temp_c * 1.8 + 32) %>%
  select(-temp_c)
```

5.2.5 The pipe %>%

With the package `dplyr` we can perform a series of operations, for example select and then filter, by sending the results of one function to another using what is called the pipe operator: `%>%`.

Details can be found in Introduction to Data Science - Chapter 4.5

5.2.6 Wide to Long

```
library(tidyr)

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

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

```
##                  time FlatA_Hum FlatA_Temp FlatB_Hum FlatB_Temp FlatC_Hum
## 1 2018-10-03 00:00:00      53.0      24.43      38.8      22.40      44.0
## 2 2018-10-03 01:00:00      53.0      24.40      38.8      22.40      44.0
## 3 2018-10-03 02:00:00      53.0      24.40      39.3      22.40      44.7
## 4 2018-10-03 03:00:00      53.0      24.40      40.3      22.40      45.0
## 5 2018-10-03 04:00:00      53.3      24.40      41.0      22.37      45.2
##   FlatC_Temp FlatD_Hum FlatD_Temp
## 1       24.5      49.0      24.43
## 2       24.5      49.0      24.40
## 3       24.5      48.3      24.38
## 4       24.5      48.0      24.33
## 5       24.5      47.7      24.30
```

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

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

```
##                   time     sensor value
## 1  2018-10-03 00:00:00  FlatA_Hum 53.00
## 2  2018-10-03 00:00:00 FlatA_Temp 24.43
## 3  2018-10-03 00:00:00  FlatB_Hum 38.80
## 4  2018-10-03 00:00:00 FlatB_Temp 22.40
## 5  2018-10-03 00:00:00  FlatC_Hum 44.00
## 6  2018-10-03 00:00:00 FlatC_Temp 24.50
## 7  2018-10-03 00:00:00  FlatD_Hum 49.00
## 8  2018-10-03 00:00:00 FlatD_Temp 24.43
## 9  2018-10-03 01:00:00  FlatA_Hum 53.00
## 10 2018-10-03 01:00:00 FlatA_Temp 24.40
## 11 2018-10-03 01:00:00  FlatB_Hum 38.80
## 12 2018-10-03 01:00:00 FlatB_Temp 22.40
## 13 2018-10-03 01:00:00  FlatC_Hum 44.00
## 14 2018-10-03 01:00:00 FlatC_Temp 24.50
## 15 2018-10-03 01:00:00  FlatD_Hum 49.00
## 16 2018-10-03 01:00:00 FlatD_Temp 24.40
```

5.2.7 Long to Wide

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

```
##                  time     sensor value
## 1 2018-10-03 00:00:00  FlatA_Hum 53.00
## 2 2018-10-03 00:00:00 FlatA_Temp 24.43
## 3 2018-10-03 00:00:00  FlatB_Hum 38.80
## 4 2018-10-03 00:00:00 FlatB_Temp 22.40
## 5 2018-10-03 00:00:00  FlatC_Hum 44.00
## 6 2018-10-03 00:00:00 FlatC_Temp 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)
```

```
##                  time FlatA_Hum FlatA_Temp FlatB_Hum FlatB_Temp FlatC_Hum
## 1 2018-10-03 00:00:00      53.0      24.43      38.8      22.40      44.0
## 2 2018-10-03 01:00:00      53.0      24.40      38.8      22.40      44.0
## 3 2018-10-03 02:00:00      53.0      24.40      39.3      22.40      44.7
## 4 2018-10-03 03:00:00      53.0      24.40      40.3      22.40      45.0
## 5 2018-10-03 04:00:00      53.3      24.40      41.0      22.37      45.2
## 6 2018-10-03 05:00:00      53.7      24.40      41.2      22.30      47.2
##   FlatC_Temp FlatD_Hum FlatD_Temp
## 1      24.50      49.0      24.43
## 2      24.50      49.0      24.40
## 3      24.50      48.3      24.38
## 4      24.50      48.0      24.33
## 5      24.50      47.7      24.30
## 6      24.57      47.2      24.30
```

5.2.8 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()

head(df)
```

```
##                  time centralOutsideTemp FlatA_Hum FlatA_Temp FlatB_Hum
## 1 2018-10-03 00:00:00              11.80      53.0      24.43      38.8
## 2 2018-10-03 01:00:00              11.25      53.0      24.40      38.8
## 3 2018-10-03 02:00:00              11.45      53.0      24.40      39.3
## 4 2018-10-03 03:00:00              11.40      53.0      24.40      40.3
## 5 2018-10-03 04:00:00              11.10      53.3      24.40      41.0
## 6 2018-10-03 05:00:00              11.05      53.7      24.40      41.2
##   FlatB_Temp FlatC_Hum FlatC_Temp FlatD_Hum FlatD_Temp
## 1      22.40      44.0      24.50      49.0      24.43
## 2      22.40      44.0      24.50      49.0      24.40
## 3      22.40      44.7      24.50      48.3      24.38
## 4      22.40      45.0      24.50      48.0      24.33
## 5      22.37      45.2      24.50      47.7      24.30
## 6      22.30      47.2      24.57      47.2      24.30
```

Chapter 6

Tidy data

The 'Tidyverse' is a collection of R packages designed for data science. See tidyverse.org/packages/ for an up-to-date list of included packages. Beside these core packages, a lot of other packages base on the same philosopy, e.g. 'tidyverts' - a collection of tidy tools for time series.

What's so special about that? Well, all packages have the same design philosophy, grammar and data structures. This means that once the data is in this data structure, it is relatively easy and straightforward to perform analysis and visualization.

6.1 Philosopy/Principles

There are three interrelated rules which make a dataset according to this philosophy tidy:

1. Each variable must have its own column.
2. Each observation must have its own row.
3. Each value must have its own cell.

For more information, see the 'Tidy Data' chapter of the book 'R for Data Science'" and the 'Tidy Data' paper published in the 'Journal of Statistical Software'".

6.2 tsibble

The 'tsibble' package provides a data infrastructure for tidy temporal data with wrangling tools. Adapting the forementioned tidy data principles, tsibble is a data- and model-oriented object.

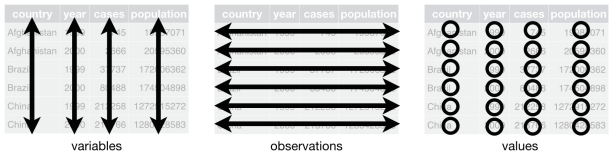


Figure 6.1: Three rules make a dataset tidy: variables are in columns, observations are in rows, and values are in cells

In `tsibble`:

1. Index is a variable with inherent ordering from past to present.
2. Key is a set of variables that define observational units over time.
3. Each observation should be uniquely identified by index and key.
4. Each observational unit should be measured at a common interval, if regularly spaced.

6.3 Examples

The following example show how these concepts can be applied in practice.

6.3.1 tidy up flatTempHum.csv

6.3.1.1 Loading data and parsing timestamp

```
library(tidyverse)
library(lubridate)

# load test data set
df <- read.csv("https://github.com/hslu-ige-laes/edar/raw/master/sampleData/flatTempHum.csv",
               stringsAsFactors=FALSE,
               sep =";")
df$time <- parse_date_time(df$time, "YmdHMS", tz = "Europe/Zurich")

head(df, 5)
```

```
##                  time FlatA_Hum FlatA_Temp FlatB_Hum FlatB_Temp FlatC_Hum
## 1 2018-10-03 00:00:00      53.0      24.43      38.8      22.40      44.0
## 2 2018-10-03 01:00:00      53.0      24.40      38.8      22.40      44.0
## 3 2018-10-03 02:00:00      53.0      24.40      39.3      22.40      44.7
## 4 2018-10-03 03:00:00      53.0      24.40      40.3      22.40      45.0
## 5 2018-10-03 04:00:00      53.3      24.40      41.0      22.37      45.2
##   FlatC_Temp FlatD_Hum FlatD_Temp
```

```
## 1       24.5      49.0      24.43
## 2       24.5      49.0      24.40
## 3       24.5      48.3      24.38
## 4       24.5      48.0      24.33
## 5       24.5      47.7      24.30
```

6.3.1.2 tidy up

What needs to be done here to get this data set into a tidy format?

- Firstly each observation should have its own row… To fulfill that, we have to convert from wide to long.
- Secondly, the variables are `flat` and `sensor` which are currently merged together in the column name. That must be separated.

```
# convert wide to long format
df.long <- as.data.frame(tidyr::pivot_longer(df,
                                             cols = -time,
                                             names_to = "datapoint",
                                             values_to = "value",
                                             values_drop_na = TRUE))
head(df.long, 5)
```

```
##         time  datapoint value
## 1 2018-10-03  FlatA_Hum 53.00
## 2 2018-10-03 FlatA_Temp 24.43
## 3 2018-10-03  FlatB_Hum 38.80
## 4 2018-10-03 FlatB_Temp 22.40
## 5 2018-10-03  FlatC_Hum 44.00
```

```
# separate datapoint into two columns
df.separated <- df.long %>%
  separate(col = datapoint, into = c("flat", "sensor"), sep = "_") %>%
  mutate_at("flat", str_replace, "Flat", "") %>%
  na.omit()

head(df.separated, 10)
```

```
##                   time flat sensor value
## 1  2018-10-03 00:00:00    A    Hum 53.00
## 2  2018-10-03 00:00:00    A   Temp 24.43
## 3  2018-10-03 00:00:00    B    Hum 38.80
## 4  2018-10-03 00:00:00    B   Temp 22.40
## 5  2018-10-03 00:00:00    C    Hum 44.00
## 6  2018-10-03 00:00:00    C   Temp 24.50
## 7  2018-10-03 00:00:00    D    Hum 49.00
## 8  2018-10-03 00:00:00    D   Temp 24.43
## 9  2018-10-03 01:00:00    A    Hum 53.00
## 10 2018-10-03 01:00:00    A   Temp 24.40
```

This dataset can now be considered tidy and is ready for further processing.

6.3.1.3 convert to 'tsibble'

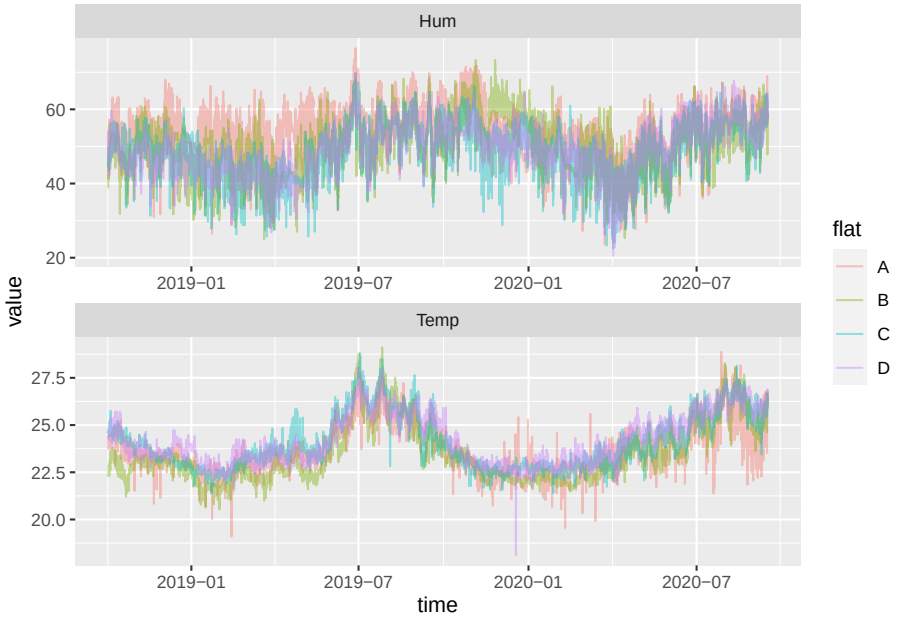
```
library(tsibble)

# convert the data frame in a tsibble object
tsbl <- as_tsibble(df.separated, key = c(flat, sensor), index = time)
tsbl
```

```
## # A tsibble: 131,602 x 4 [1h] <Europe/Zurich>
## # Key:       flat, sensor [8]
##    time                flat  sensor value
##    <dttm>              <chr> <chr>  <dbl>
##  1 2018-10-03 00:00:00 A     Hum     53
##  2 2018-10-03 01:00:00 A     Hum     53
##  3 2018-10-03 02:00:00 A     Hum     53
##  4 2018-10-03 03:00:00 A     Hum     53
##  5 2018-10-03 04:00:00 A     Hum     53.3
##  6 2018-10-03 05:00:00 A     Hum     53.7
##  7 2018-10-03 06:00:00 A     Hum     50.8
##  8 2018-10-03 08:00:00 A     Hum     42.7
##  9 2018-10-03 09:00:00 A     Hum     46
## 10 2018-10-03 10:00:00 A     Hum     41.3
## # ... with 131,592 more rows
```

6.3.1.4 Visualization

It's now relatively easy and straightforward to make a plot using the key-columns `flat` and `sensor`:

```
library(ggplot2)

ggplot(tsbl, aes(x= time, y= value, colour = flat)) +
  geom_line(alpha=0.4) +
  facet_wrap(~ sensor,  scales="free", ncol = 1)
```


Hum
60
40
20
Temp
27.5
25.0
22.5
20.0
2019−01
2019−07
2020−01
2020−07
value
time
flat
A
B
C
D

Chapter 7

Explorative Data Analysis

Explorative Data Analysis (EDA) is a technique based on the human characteristic of visual pattern recognition.

The purpose of EDA is simple: learn more about data by visualizing it in different ways.

> "Exploratory data analysis is graphical detective work." - John W. Tukey, considered the founder of EDA

7.1 Get Overview of Data

A first step is getting an overview of the whole data set and specific series of it.

7.1.1 Load data

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

```
## Warning: 3 failed to parse.
```

7.1.2 Names

Show the column headers of a data frame:

```
names(df)
```

```
## [1] "time"              "energyHeatingMeter" "supplyTempHeating"
```

7.1.3 Structure

Show the structure of the data frame:

```
str(df)
```

```
## 'data.frame':    22317 obs. of  3 variables:
##  $ time              : POSIXct, format: "2018-03-03 00:00:00" "2018-03-03 01:00:00" ...
##  $ energyHeatingMeter: num  45020 NA NA NA NA ...
##  $ supplyTempHeating : num  24.7 24.1 23.7 23.4 31.6 ...
```

7.1.4 Head/Tail

Show the first and last values:

```
head(df)
```

```
##                 time energyHeatingMeter supplyTempHeating
## 1 2018-03-03 00:00:00           45019.81             24.73
## 2 2018-03-03 01:00:00                 NA             24.06
## 3 2018-03-03 02:00:00                 NA             23.73
## 4 2018-03-03 03:00:00                 NA             23.45
## 5 2018-03-03 04:00:00                 NA             31.59
## 6 2018-03-03 05:00:00                 NA             29.14
```

```
tail(df)
```

```
##                      time energyHeatingMeter supplyTempHeating
## 22312 2020-09-17 15:00:00                 NA                NA
## 22313 2020-09-17 16:00:00                 NA                NA
## 22314 2020-09-17 17:00:00                 NA                NA
## 22315 2020-09-17 18:00:00                 NA                NA
## 22316 2020-09-17 19:00:00                 NA                NA
## 22317 2020-09-17 20:00:00                 NA                NA
```

> Note: if you want to show only the first three entries, you can type `head(df,3)`

7.1.5 Five number summary

Reveals details about the distribution of the data:

```
summary(df$supplyTempHeating)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##   18.37   23.35   24.61   25.72   27.49   44.68     362
```

7.2 Quick Data Visualizations

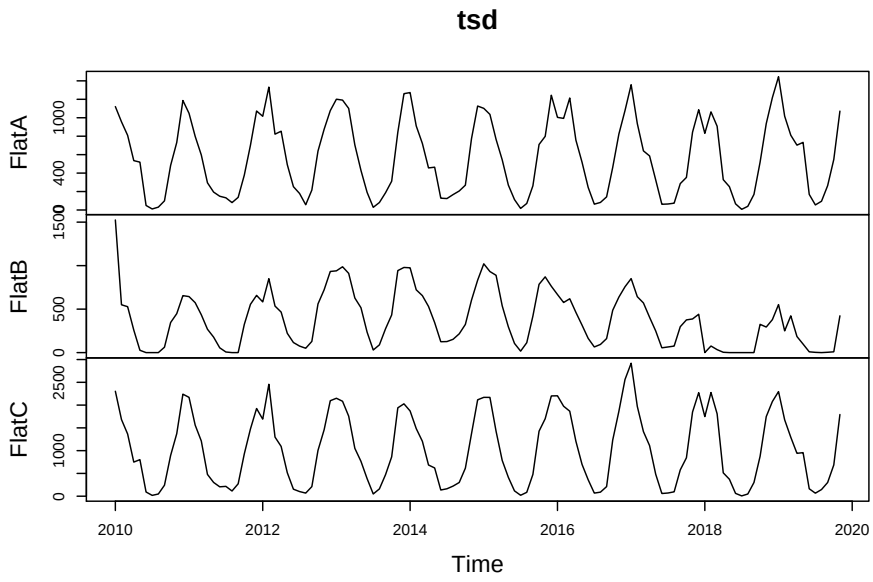
Please refer to the R Graphics Cookbook from Winston Chang - Chapter 2 “Quickly Exploring Data”.

This is a great resource where you can find everything you need to know about creating basic plots. Some plots of the part “Data Visualizations” are as well easy to create and can get used in the context of EDA.

```
library(dplyr)
library(lubridate)
# load csv file
df <- read.csv2("https://github.com/hslu-ige-laes/edar/raw/master/sampleData/flatHeatAndHotWater.csv",
                stringsAsFactors=FALSE)
# filter flat
df <- df %>% select(timestamp, Adr01_energyHeat, Adr02_energyHeat, Adr03_energyHeat)
#df <- df %>% filter(timestamp > "2014-12-01")

df <- df %>% dplyr::mutate(FlatA = lead(Adr01_energyHeat) - Adr01_energyHeat)
df <- df %>% dplyr::mutate(FlatB = lead(Adr02_energyHeat) - Adr02_energyHeat)
df <- df %>% dplyr::mutate(FlatC = lead(Adr03_energyHeat) - Adr03_energyHeat)

# remove counter value column
df <- df %>% select(-Adr01_energyHeat, -Adr02_energyHeat, -Adr03_energyHeat) %>% na.omit()
df$timestamp <- parse_date_time(df$timestamp,
                                orders = "YmdHMS",
                                tz = "Europe/Zurich")

tsd <- ts(df %>% select(-timestamp), frequency = 12, start = min(year(df$timestamp)))
```

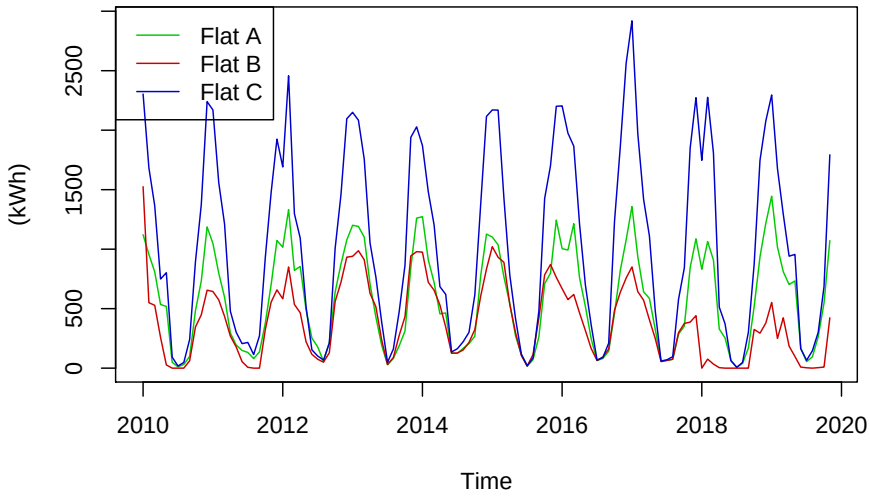
7.2.1 Multiple time series plot

```
# plot FlatA_Ele
plot(tsd)
```



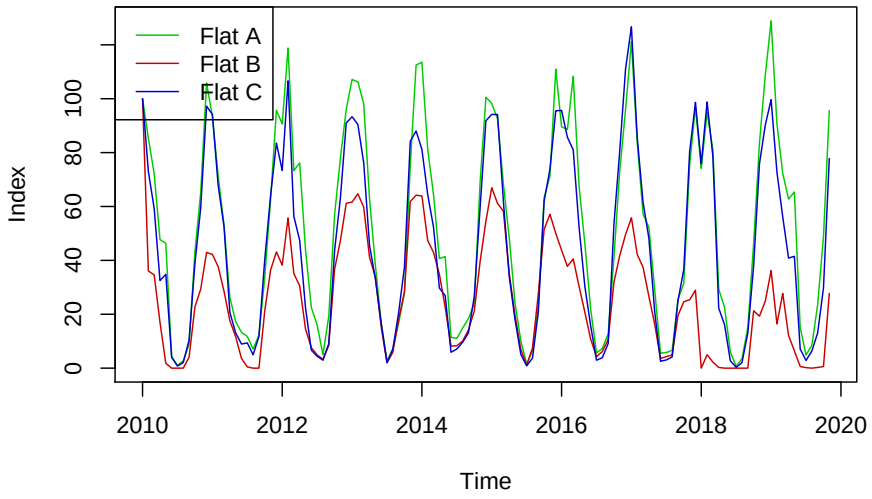
7.2.2 Multiple time series plot in one plot

```
# Plotting all in one frame
colours<-c("green3","red3","blue3")

plot(tsd,plot.type="single",ylab="(kWh)",col=colours)

# Legend
ltxt<-c("Flat A", "Flat B", "Flat C")
legend("topleft",lty=1,col=colours, legend=ltxt)
```



7.2.3 Indexed time series plot

```
tsdPlot <- tsd

# Index timeseries
tsdPlot[,1]<-tsdPlot[,1]/tsdPlot[1,1]*100
tsdPlot[,2]<-tsdPlot[,2]/tsdPlot[1,2]*100
tsdPlot[,3]<-tsdPlot[,3]/tsdPlot[1,3]*100

# Plotting all in one frame
colours<-c("green3","red3","blue3")

plot(tsdPlot,plot.type="single",ylab="Index",col=colours)

# Legend
ltxt<-c("Flat A", "Flat B", "Flat C")
legend("topleft",lty=1,col=colours, legend=ltxt)
```


Flat A
Flat B
Flat C
Index
0
20
40
60
80
100
2010
2012
2014
2016
2018
2020
Time

Part II

Data Visualizations

Chapter 8

Introduction to Data Visualizations

In the following chapters of the "Data Visualizations" section, visualizations are presented that are either already frequently used in the field of building monitoring, are difficult to create or seem to be recommendable.

Sometimes not the final creation of the lines is challenging, but the preceding data preparation of the raw data. Therefore all parts of the data preparation from the raw-files up to the final visualization are shown as recipes.

The focus is on visualizations for reports in print quality. However, since some functions are implemented in the redutils package and are thus easy to call up, they are therefore also suitable for the fast visualization of data in the field of explorative data analysis.

Chapter 9

Statistical Characteristics

The first step of a data analysis is to become familiar with the data. For this purpose some visualizations of statistical parameters are presented hereafter.

9.1 Boxplot

9.1.1 Goal

You want to create a boxplot:

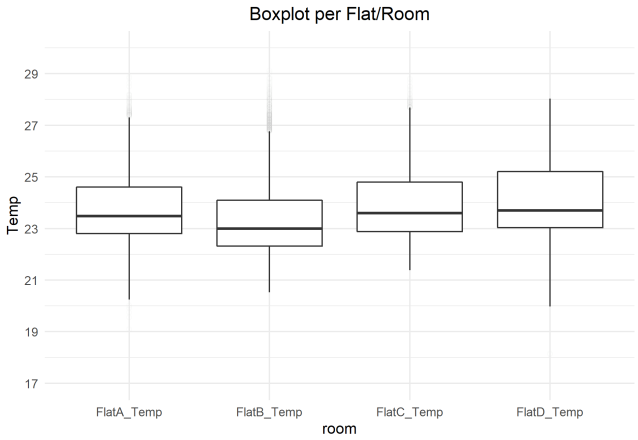


Figure 9.1: Building Energy Signature Plot

9.1.2 Data Basis

A csv file with room temperature and humidity time series of four rooms. Below are only the room temperatures visualized.

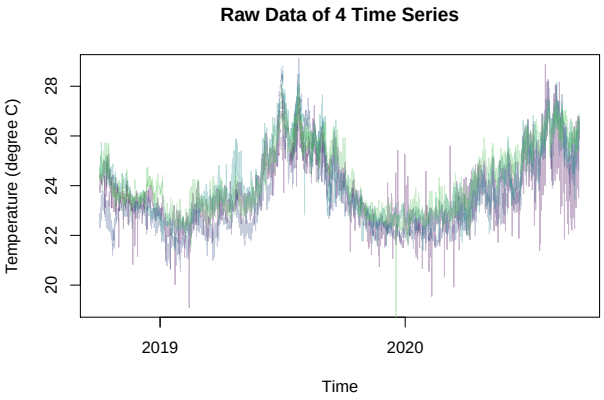
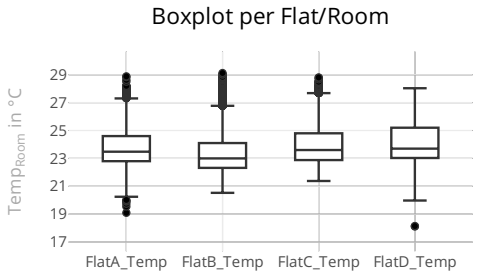


Figure 9.2: Room Temperature Raw Data for Boxplot

9.1.3 Solution

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

```
library(ggplot2)
library(plotly)
library(dplyr)
library(tidyr)
library(redutils)
library(lubridate)
# load time series data and aggregate daily mean values
df <- read.csv("https://github.com/hslu-ige-laes/edar/raw/master/sampleData/flatTempHum.csv",
                           stringsAsFactors=FALSE,
                           sep =";")

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

df <- df %>%
  select(time, FlatA_Temp, FlatB_Temp, FlatC_Temp, FlatD_Temp)

df <- as.data.frame(tidyr::pivot_longer(df,
                                        cols = -time,
                                        names_to = "room",
                                        values_to = "Temp",
                                        values_drop_na = TRUE))

minY <- round(min(df %>% select(Temp), na.rm = TRUE) - 1, digits = 0)
maxY <- round(max(df %>% select(Temp), na.rm = TRUE) + 1, digits = 0)


plot <- ggplot(df, aes(x = room, y = Temp)) +
  geom_boxplot(outlier.alpha = 0.01, outlier.shape = 1, outlier.colour = "darkgrey") +
  scale_y_continuous(limits=c(minY,maxY), breaks = seq(minY, maxY, by = 2)) +
  ggtitle("Boxplot per Flat/Room") +
  theme_minimal() +
  theme(
    legend.position="none",
    plot.title = element_text(hjust = 0.5)
  )

yaxis <- list(
  title = "Temp<sub>Room</sub> in \u00B0C\n",
  automargin = TRUE,
  titlefont = list(size = 14, color = "darkgrey")
)

# create interactive plot
ggplotly(plot + ylab(" ") + xlab(" ")) %>%
  plotly::config(modeBarButtons = list(list("toImage")),
                 displaylogo = FALSE,
                 toImageButtonOptions = list(
                   format = "svg"
                 )
  ) %>%
  layout(yaxis = yaxis)
```




```
# save static plot as png (optional)
ggsave("images/boxplot.png", plot)
```

9.2 Cross-Correlation

9.2.1 Goal

You want to create a cross-correlation plot:

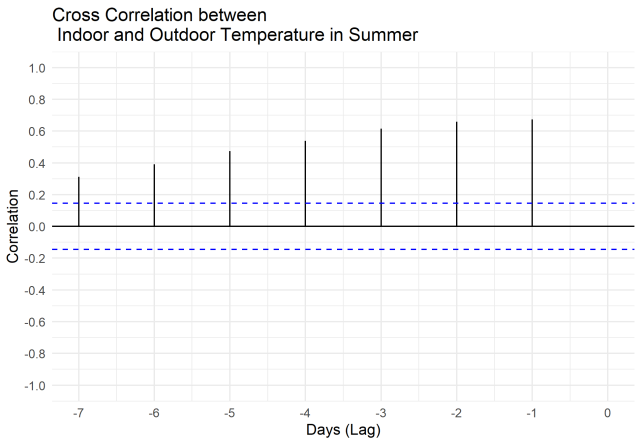


Figure 9.3: Building Energy Signature Plot

9.2.2 Data Basis

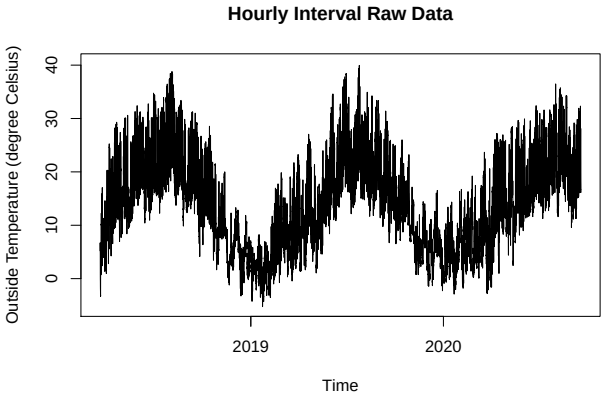


Figure 9.4: Raw Data Room and Outdoor Temperature for Cross Correlation Plot

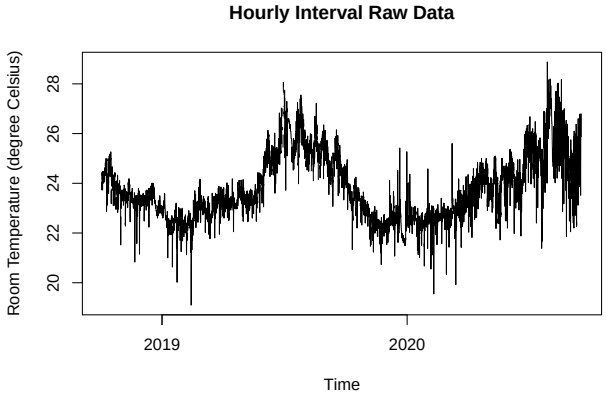


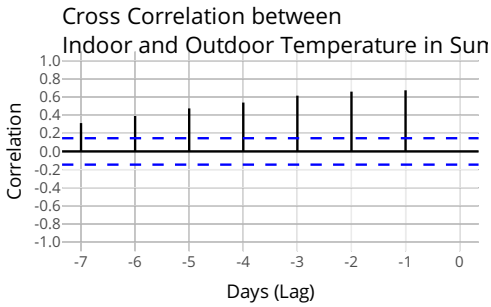
Figure 9.5: Raw Data Room and Outdoor Temperature for Cross Correlation Plot

9.2.3 Solution

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

```
library(redutils)
library(dplyr)
library(lubridate)
library(zoo)
library(plotly)
library(forecast)

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

dfTempOa$time <- parse_date_time(dfTempOa$time,
                                       order = "YmdHMS",
                                       tz = "UTC")

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

dfTempOa <- dfTempOa %>%
  group_by(day) %>%
  dplyr::summarize(tempOa = mean(centralOutsideTemp, na.rm = TRUE)) %>%
  ungroup()

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

dfTempR$time <- parse_date_time(dfTempR$time,
                                      order = "YmdHMS",
```

```
                                    tz = "UTC")

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

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

dfTempR <- dfTempR %>%
  group_by(day) %>%
  dplyr::summarize(tempR = mean(FlatA_Temp, na.rm = TRUE)) %>%
  ungroup()

data <- merge(dfTempR, dfTempOa, all = TRUE) %>% unique() %>% na.omit()

data$season <- redutils::getSeason(data$day)

data <- data %>%
  filter(season == "Summer")

# plot diagram
plot <- ggCcf(data$tempOa,
              data$tempR,
              lag.max = 7) +
  theme_minimal() +
  scale_x_continuous(limits = c(-7, 0), breaks = seq(-7,0,1)) +
  scale_y_continuous(limits = c(-1, 1), breaks = seq(-1,1,0.2)) +
  labs(title="Cross Correlation between \n Indoor and Outdoor Temperature in Summer",
       x ="Days (Lag)", y = "Correlation")

# make plot interactive (optional)
ggplotly(plot)
```




```
# save static plot as png (optional)
ggsave("images/crossCorrelation.png", plot)
```

9.3 Density Plot

9.3.1 Goal

You want to create a density plot of a temperature series with the mean value as vertical line:

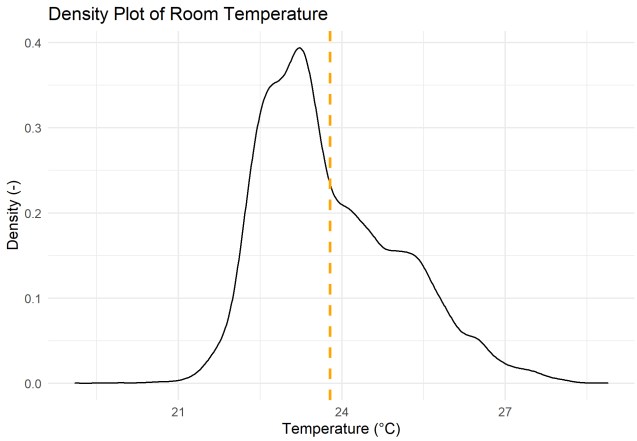


Figure 9.6: Density Plot Temperature

9.3.2 Data Basis

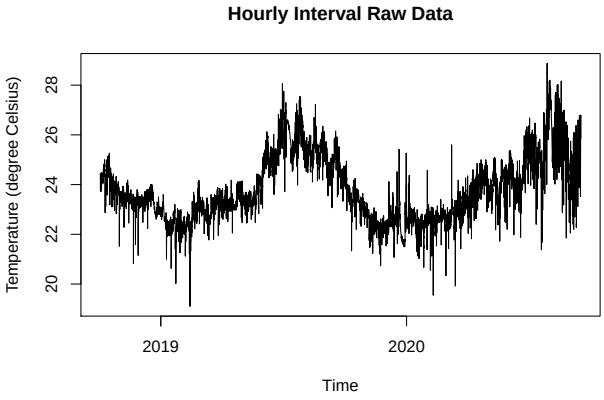
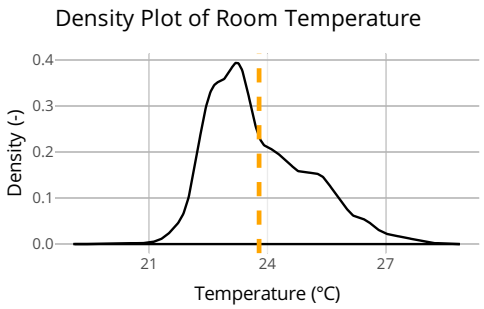


Figure 9.7: Raw Data Temperature for Density Plot

9.3.3 Solution

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

```
library(ggplot2)
library(plotly)
library(dplyr)

# load time series data and aggregate daily mean values
library(dplyr)
library(lubridate)
# 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()

colnames(df) <- c("time", "value")

# static chart with ggplot
plot <- ggplot(df, aes(x = value)) +
  geom_density() +
  geom_vline(aes(xintercept = mean(value, na.rm = TRUE)),
             color = "orange",
             linetype = "dashed",
             size = 1,
             label = "Mean")+
  ggtitle("Density Plot of Room Temperature") +
  labs(x = "Temperature (\u00B0C)",
       y = "Density (-)") +
  theme_minimal()

# interactive chart
plotly::ggplotly(plot)
```




```
# save static plot as png
ggsave("images/plotDensity.png", plot)
```

9.4 Density Plot Season

9.4.1 Goal

You want to create a density plot of a temperature series for each season of the year:

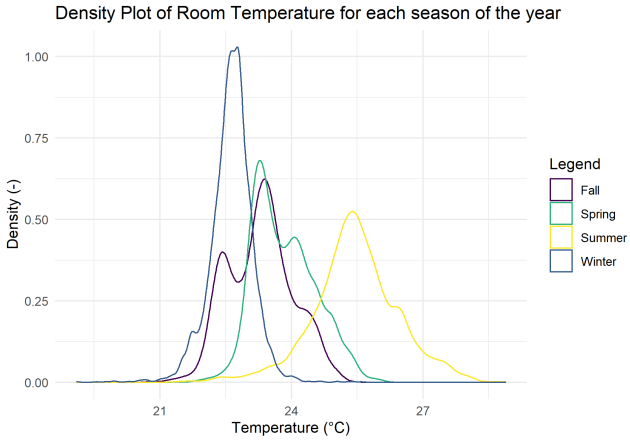


Figure 9.8: Density Plot Temperature for each season of the year

9.4.2 Data Basis

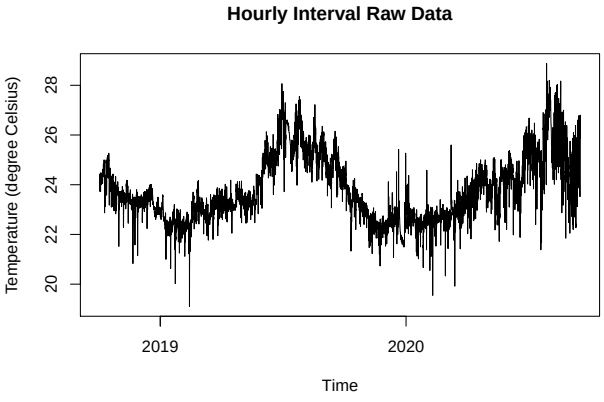
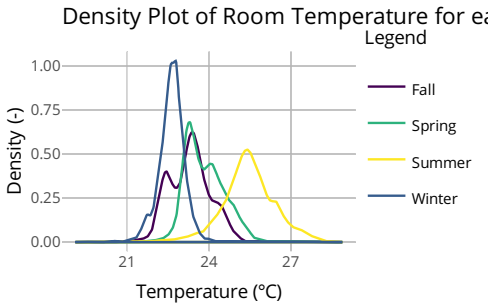


Figure 9.9: Raw Data Temperature for Density Plot

9.4.3 Solution

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

```
library(ggplot2)
library(plotly)
library(dplyr)

# load time series data and aggregate daily mean values
library(dplyr)
library(lubridate)
library(redutils)

# 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()

colnames(df) <- c("time", "value")

df$season = getSeason(df$time)

# static chart with ggplot
plot <- ggplot(df) +
  geom_density(aes(x = value, colour = season)) +
  scale_color_manual(values=c("#440154", "#2db27d", "#fde725", "#365c8d")) +
  ggtitle("Density Plot of Room Temperature for each season of the year") +
  labs(x = "Temperature (\u00B0C)",
       y = "Density (-)",
       colour = "Legend") +
  theme_minimal()

# interactive chart
plotly::ggplotly(plot)
```




```
# save static plot as png
ggsave("images/plotDensitySeasons.png", plot)
```

Chapter 10

Time Series Decomposition

Typical time series result from the interaction of regular deterministic and random causes. The deterministic regular causes can vary periodically (seasonally) and/or contain long-term trends. Random causes are often also called noise.

Decomposition breaks down time series into their components “trend”, “seasonality” and “randomness”. For further data analysis it is often necessary to decompose the trend and the seasonal component from the raw data. Therefore, a large part of the time series analyses deals with corresponding procedures. Sometimes the remaining Randomness is of interest and sometimes a detrended series as in chapter 13.4.

Trend Component The trend component of a time series refers to the general direction in which the time series moves in the long term. Time series can have a positive or negative trend, or no trend, but they can also have no trend. A trend is present when the data show a continuously rising and/or falling direction.

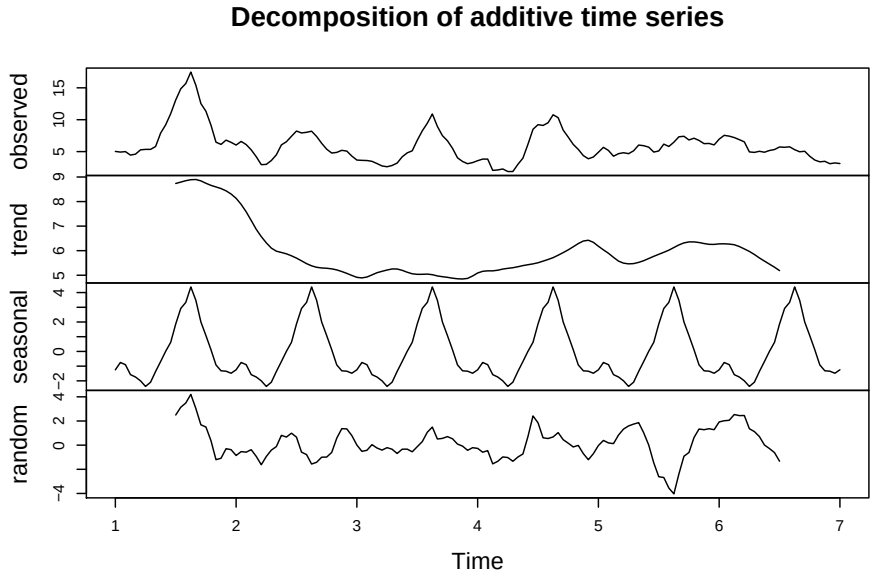
Seasonal Component The seasonal component for time series data refers to their tendency to rise and fall with constant frequency. Seasonality occurs over a fixed and known period of time (e.g. the quarter of the year, the month or the day of the week).

Random/Remainder/Irregular Component The rest is what remains of the time series data after its trend and seasonal components have been removed. It is the random fluctuation in the time series data that cannot be explained by the above components.

For energy data analysis, each of the three components mentioned above may be of interest, depending on the case.

This chapter shows how a time series can be decomposed and presented interactively.

An exemplary decomposition of an outside temperature time series over 7 days:



10.1 Long term

10.1.1 Goal

Decompose a long term time series of ten years monthly data:

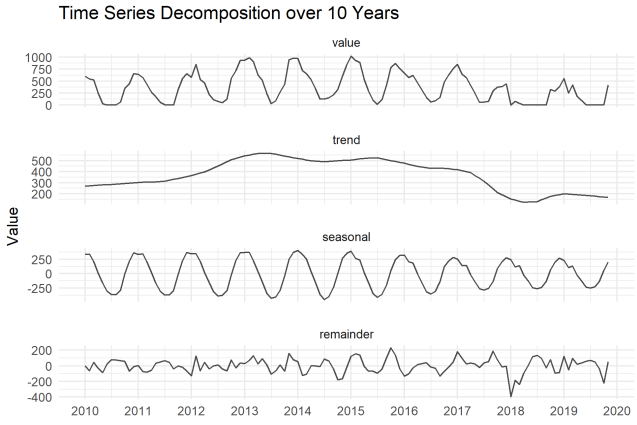


Figure 10.1: Decomposition of long time series over 10 Years

10.1.2 Data Basis

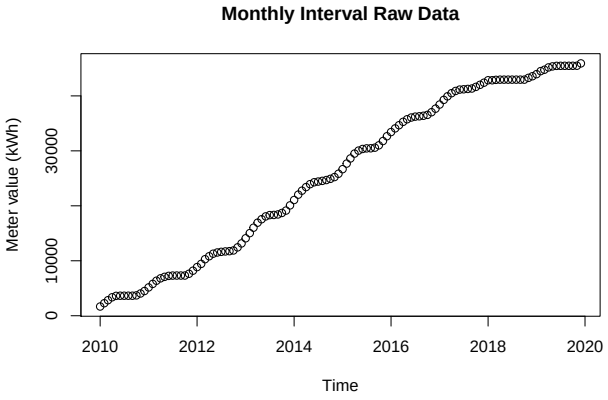


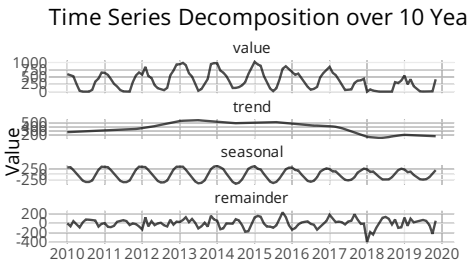
Figure 10.2: Raw Data for Decomposition Plot Long Term

10.1.3 Solution

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

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

# load csv file
df <- read.csv2("https://github.com/hslu-ige-laes/edar/raw/master/sampleData/flatHeatAndHotWater.csv",
                stringsAsFactors=FALSE)

# filter flat
df <- df %>% select(timestamp, Adr02_energyHeat)

colnames(df) <- c("Time", "meterValue")

df$Time <- parse_date_time(df$Time,
                                orders = "YmdHMS",
                                tz = "Europe/Zurich")

# calculate consumption value per month
# pay attention, the value of 2010-02-01 00:00:00 represents the meter reading on february first,
# so the consumption for february first is value(march) - value(february)!
df <- df %>% dplyr::mutate(value = lead(meterValue) - meterValue)

# remove counter value column
df <- df %>% select(-meterValue) %>% na.omit()
df[1,2] <- 600

df.ts <- ts(df %>% select(value) %>% na.omit(), frequency = 12, start = min(year(df$Time)))

df.decompose <- df.ts[,1] %>%
  stl(s.window = 7)

df.decompose <- df.decompose$time.series

df.decompose <- as.data.frame(df.decompose)

df.decompose <- cbind(df, df.decompose)

data <- as.data.frame(tidyr::pivot_longer(df.decompose,
                                          cols = -Time,
                                          names_to = "Component",
                                          values_to = "Value",
                                          values_drop_na = TRUE)
)
data$component <- as.factor(data$Component)
data$component <- factor(data$Component, c("value",
                                           "trend",
                                           "seasonal",
                                           "remainder"))
data$Value <- round(data$Value, digits = 1)

plot <- ggplot(data) +
```

```
  geom_path(aes(x = Time,
                y = Value
  ),
  color = "black",
  alpha = 0.7) +
  facet_wrap(~component, ncol = 1, scales = "free_y") +
  scale_x_datetime(date_breaks = "years" , date_labels = "%Y") +
  theme_minimal() +
  theme(panel.spacing = unit(1, "lines"),
        legend.position = "none") +
  labs(x = "") +
  ggtitle("Time Series Decomposition over 10 Years")

ggplotly(plot)
```




```
# save static plot as png (optional)
ggsave("images/plotDecompositionLong.png", plot)
```

10.1.4 Discussion

- Energy optimization in mid-2017 is clearly visible in the trend and also in the magnitude of the seasonal pattern
- And as well in the remainder the too low setting of January 2018 where the thermostat of the flat got deactivated

- The trend shows as well an higher consumption in June 2013

10.2 Short term

10.2.1 Goal

Decompose a short term time series of e.g. 5 days 15min data:

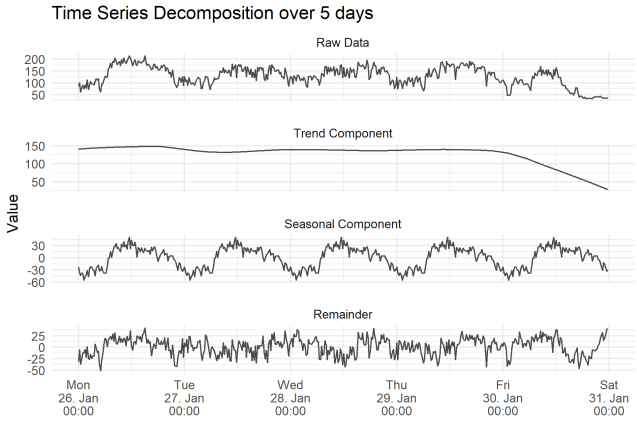


Figure 10.3: Seasonal Plot Overlapping per Month over 10 Years

10.2.2 Data Basis

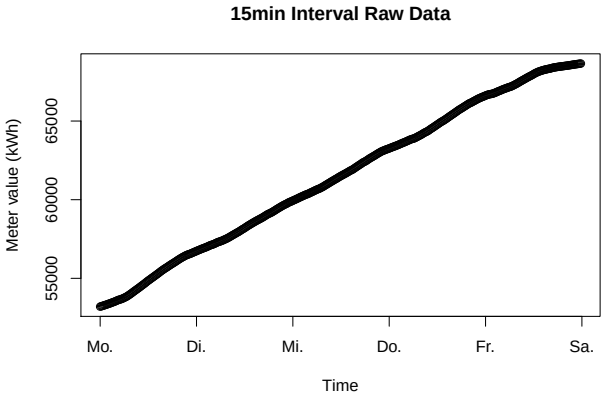


Figure 10.4: Raw Data for Decomposition Plot Short Term

10.2.3 Solution

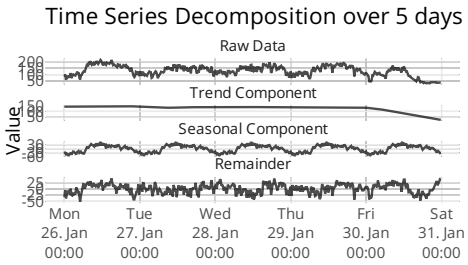
Create a new script, copy/paste the following code and run it:

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

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

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

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

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

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

# uncomment to filter time range if necessary
#df <- df %>% filter(Time > "2015-03-01 00:00:00", Time < "2015-04-01 00:00:00")

# Fill missing values with NA
grid.df <- data.frame(time = seq(min(df$time, na.rm = TRUE),
                                 max(df$time, na.rm = TRUE),
                                 by = "15 mins"))
df <- merge(df, grid.df, all = TRUE)

# convert steadily counting energy meter value from kWh to power in kW
df <- df %>%
  dplyr::mutate(value = (meterValue - lag(meterValue))*4) %>%
  select(-meterValue) %>%
  na.omit()

# remove negative values which occur beause of change summer/winter time
df <- df %>% filter(value >= 0)

# select time range
df <- df %>% filter(time >= as.POSIXct("2015-01-26 00:00:00", tz = "UTC"),
                    time < as.POSIXct("2015-01-31 00:00:00", tz = "UTC"))

# ========== Start of Code ================
df.ts <- ts(df %>% select(value) %>% na.omit(),
            frequency = 96)

df.decompose <- df.ts[,1] %>%
```

```
  stl(s.window = 193)

df.decompose <- df.decompose$time.series

df.decompose <- as.data.frame(df.decompose)

df.decompose <- cbind(df, df.decompose)

data <- as.data.frame(tidyr::pivot_longer(df.decompose,
                                          cols = -time,
                                          names_to = "component",
                                          values_to = "value",
                                          values_drop_na = TRUE)
)
data$component <- as.factor(data$component)
data$component <- factor(data$component, c("value",
                                           "trend",
                                           "seasonal",
                                           "remainder"))
# prepare data for plot

componentTitles = c("Raw Data","Trend Component", "Seasonal Component", "Remainder")

data <- data %>%
  dplyr::mutate(component = recode(component,
                                   value = componentTitles[1],
                                   trend = componentTitles[2],
                                   seasonal = componentTitles[3],
                                   remainder = componentTitles[4]),
         value = round(data$value, digits = 1)) %>%
  rename(Value = value,
         Time = time)

plot <- ggplot(data) +
  geom_path(aes(x = Time,
                y = Value
               ),
            color = "black",
            alpha = 0.7) +
  facet_wrap(~component, ncol = 1, scales = "free_y") +
  scale_x_datetime(date_breaks = "days" , date_labels = "%a\n%d. %b\n%H:%M") +
  theme_minimal() +
  theme(panel.spacing = unit(1, "lines"),
        legend.position = "none") +
  labs(x = "") +
  ggtitle("Time Series Decomposition over 5 days")

ggplotly(plot)
```




```
# save static plot as png (optional)
ggsave("images/plotDecompositionShort.png", plot)
```

Chapter 11

Seasonal Plots

Seasonal plots are a graphical tool to visualize and detect seasonality in a time series. Based on a selected periodicity it emphasizes the seasonal patterns and also shows the changes in seasonality over time. Especially, it allows to detect changes between different seasons.

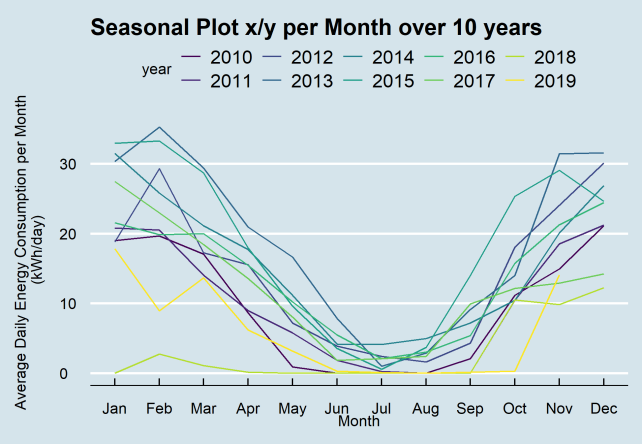


Figure 11.1: Seasonal Plot Overlapping per Month over 10 Years

This is like a standard time series plot except that the data are plotted against the "seasons" for each year and are overlapping. Be aware that seasons in this context don't correlate with the seasons of the year. For example in 11.1 the seasons are months.

When displaying the data in months, it is important that the consumption is calculated down to a daily value, otherwise there will be unnecessary distortions due to the different number of days per month.

However, such plots are only useful if the period of the seasonality is already known. In many cases, this will in fact be known. For example, monthly data typically has a period of 12. If the period is not known, an autocorrelation plot or spectral plot can be used to determine it.

This chapter shows some useful types of seasonal plots.

11.1 Overlapping

11.1.1 Goal

Plot a seasonal plot as described in Hyndman and Athanasopoulos (2014, chapter 2.4):

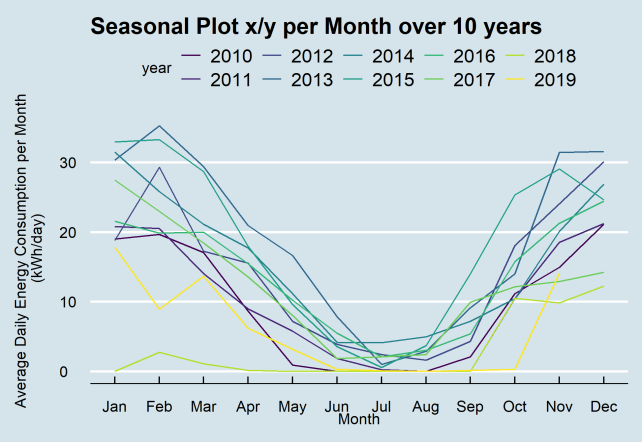


Figure 11.2: Seasonal Plot Overlapping per Month over 10 Years

11.1.2 Data Basis

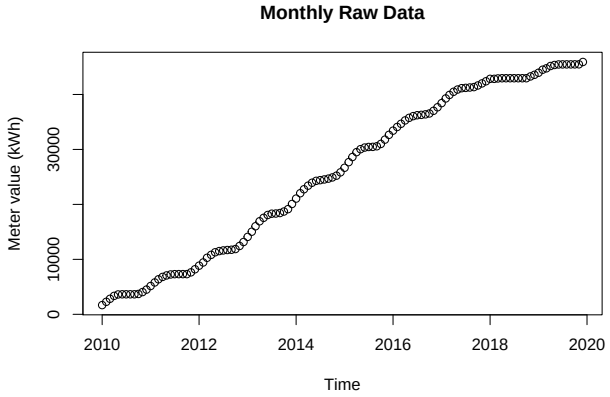


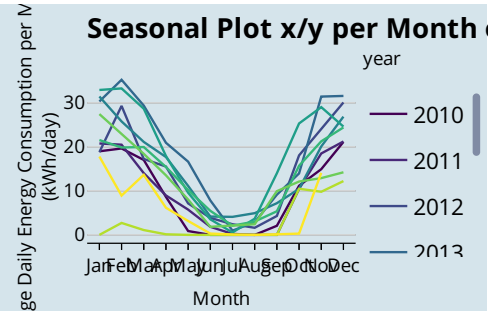
Figure 11.3: Raw Data for Seasonal Plot Overlapping

11.1.3 Solution

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

```
library(forecast)
library(dplyr)
library(plotly)
library(htmlwidgets)
library(ggthemes)
library(viridis)
library(lubridate)

# load csv file
df <- read.csv2("https://github.com/hslu-ige-laes/edar/raw/master/sampleData/flatHeatAndHotWater.csv",
                stringsAsFactors=FALSE)

# filter flat
df <- df %>% select(timestamp, Adr02_energyHeat)

colnames(df) <- c("timestamp", "meterValue")

# calculate consumption value per month
# pay attention, the value of 2010-02-01 00:00:00 represents the meter reading on february first,
# so the consumption for february first is value(march) - value(february)!
df <- df %>% dplyr::mutate(value = lead(meterValue) - meterValue)

# remove counter value column and calculate consumption per day instead of month
df <- df %>%
  select(-meterValue) %>%
  mutate(value = value /lubridate::days_in_month(timestamp))

# value correction (outlier because of commissioning)
df[1,2] <- 19

# create time series object for ggseanplot function
```

```
df.ts <- ts(df %>% select(value) %>% na.omit(), frequency = 12, start = min(year(df$timestamp)))

# create x/y plot
numYears = length(unique(year(df$timestamp))) # used for colours

plot <- ggseasonplot(df.ts,
                     col = viridis(numYears),
                     main = "Seasonal Plot x/y per Month over 10 years",
                     ylab = "Average Daily Energy Consumption per Month \n(kWh/day)\n"
                     )

# show static plot (uncomment it if you want a static plot)
#plot

# change theme (optional)
plot <- plot + ggthemes::theme_economist()

# make plot interactive (optional)
plotly <- plotly::ggplotly(plot)

# show plot interactive plot (optional)
plotly
```




```
# save static plot as png (optional)
ggsave("images/plotSeasonalXY.png", plot)
```

```
# save interactive plot as html (optional)
library(htmlwidgets)
htmlwidgets::saveWidget(plotly, "plotlySeasonalXY.html")
```

11.1.4 Discussion

A seasonal plot allows the underlying seasonal pattern to be seen more clearly, and is especially useful in identifying years in which the pattern changes.

Hints:

- Double click on a specific year in the legend to display only that year
- Click once to activate/deactivate an element

11.2 Mini Plots

11.2.1 Goal

Plot a seasonal month plot as described in Hyndman and Athanasopoulos (2014, chapter 2.5):

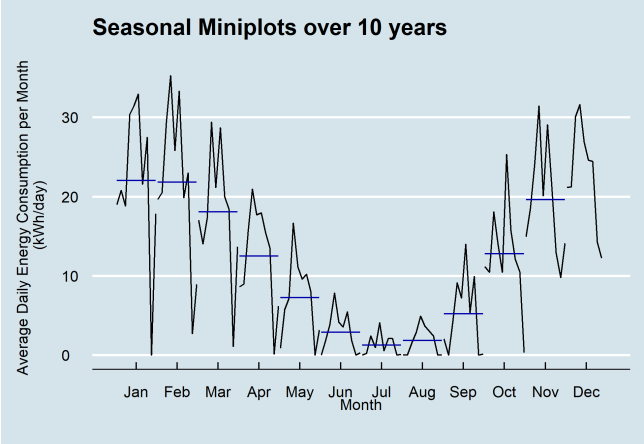


Figure 11.4: Seasonal Plot with mini Time Plots over 10 Years

Here the seasonal patterns for each season are collected together in seprate mini time plots.

11.2.2 Data Basis

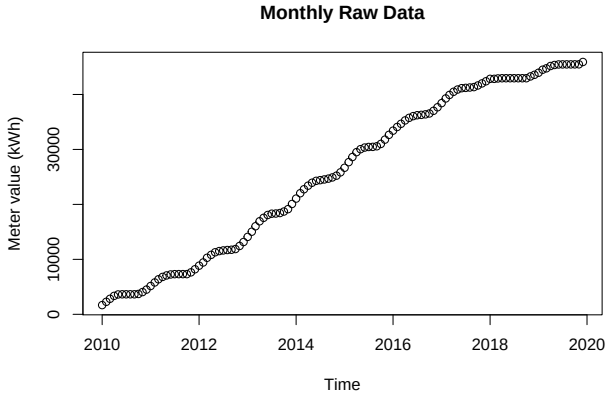


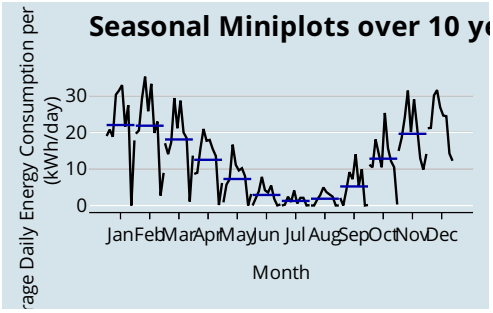
Figure 11.5: Raw Data for Seasonal Miniplots

11.2.3 Solution

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

```
library(forecast)
library(dplyr)
library(plotly)
library(htmlwidgets)
library(ggthemes)
library(viridis)
library(lubridate)

# load csv file
df <- read.csv2("https://github.com/hslu-ige-laes/edar/raw/master/sampleData/flatHeatAndHotWater.csv",
                stringsAsFactors=FALSE)

# filter flat
df <- df %>% select(timestamp, Adr02_energyHeat)

colnames(df) <- c("timestamp", "meterValue")

# calculate consumption value per month
# pay attention, the value of 2010-02-01 00:00:00 represents the meter reading on february first,
# so the consumption for february first is value(march) - value(february)!
df <- df %>% dplyr::mutate(value = lead(meterValue) - meterValue)

# remove counter value column and calculate consumption per day instead of month
df <- df %>%
  select(-meterValue) %>%
  mutate(value = value /lubridate::days_in_month(timestamp))

# value correction (outlier because of commissioning)
df[1,2] <- 19

# create time series object for ggmonthplot function
```

```
df.ts <- ts(df[-1], frequency = 12, start = min(year(df$timestamp)))

# create x/y plot

numYears = length(unique(year(df$timestamp)))

plot <- ggmonthplot(df.ts,
                    col = viridis(numYears),
                    main = "Seasonal Miniplots over 10 years\n",
                    ylab = "Average Daily Energy Consumption per Month \n(kWh/day)\n",
                    xlab = "Month\n "
                    )

# change theme (optional)
plot <- plot + ggthemes::theme_economist()

# make plot interactive (optional)
plotly <- plotly::ggplotly(plot)

# show plot
plotly
```




```
# save static plot as png
ggsave("images/plotSeasonalMiniplots.png", plot)
```

```
# save interactive plot as html
library(htmlwidgets)
htmlwidgets::saveWidget(plotly, "plotlySeasonalMiniplots.html")
```

11.2.4 Discussion

- This type of seasonal plot shows the mean value of each month and therefore emphasises on the monthly comparison
- It revelas as well the mean seasonal pattern with the blue lines

11.3 Polar

11.3.1 Goal

Plot a seasonal plot as described in Hyndman and Athanasopoulos (2014, chapter 2.4):

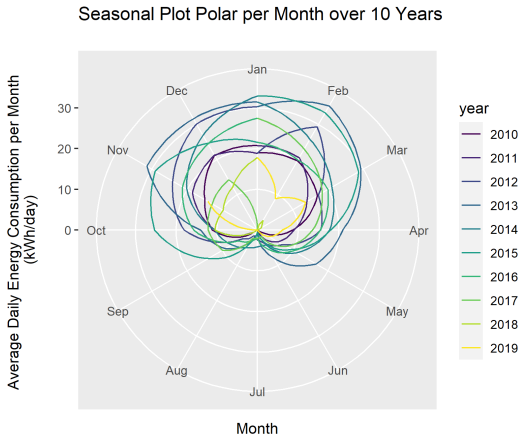


Figure 11.6: Seasonal Plot Polar per Month over 10 Years

This is like an overlapping time series plot which uses polar coordinates. Be aware that seasons in this context don't correlate with the seasons of the year.

11.3.2 Data Basis

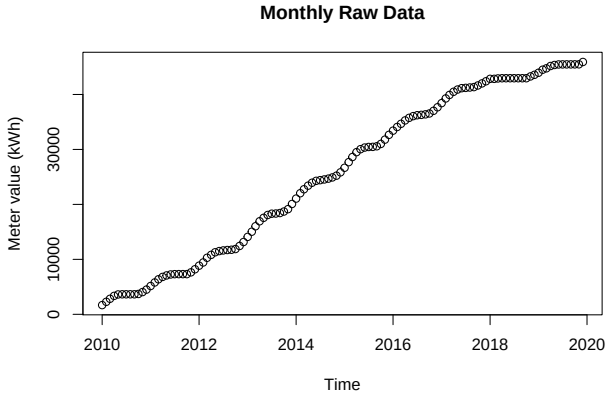


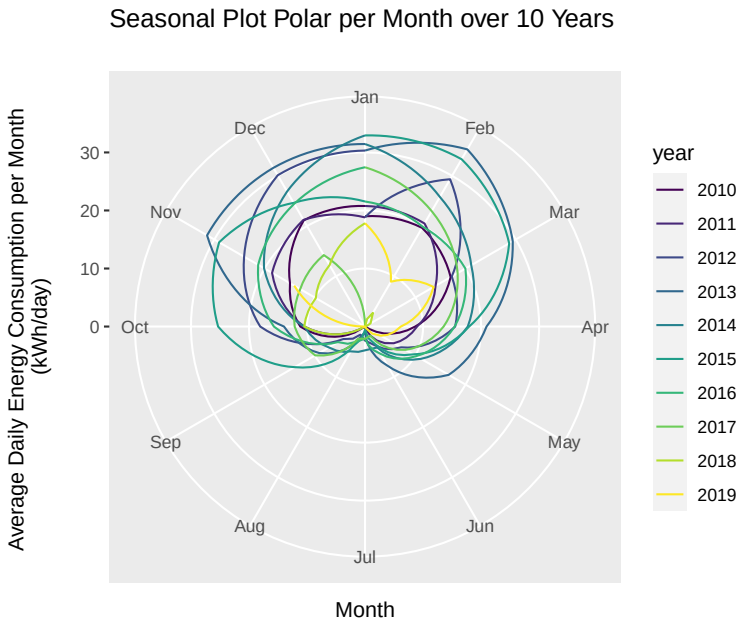
Figure 11.7: Raw Data for Seasonal Plot Polar

11.3.3 Solution

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

```
library(forecast)
library(dplyr)
library(plotly)
library(htmlwidgets)
library(ggthemes)
library(viridis)
library(lubridate)

# load csv file
df <- read.csv2("https://github.com/hslu-ige-laes/edar/raw/master/sampleData/flatHeatAndHotWater.csv",
                stringsAsFactors=FALSE)

# filter flat
df <- df %>% select(timestamp, Adr02_energyHeat)

colnames(df) <- c("timestamp", "meterValue")

# calculate consumption value per month
# pay attention, the value of 2010-02-01 00:00:00 represents the meter reading on february first,
# so the consumption for february first is value(march) - value(february)!
df <- df %>% dplyr::mutate(value = lead(meterValue) - meterValue)

# remove counter value column and calculate consumption per day instead of month
df <- df %>%
  select(-meterValue) %>%
  mutate(value = value /lubridate::days_in_month(timestamp))

# value correction (outlier because of commissioning)
df[1,2] <- 19

df.ts <- ts(df %>% select(value) %>% na.omit(), frequency = 12, start = min(year(df$timestamp)))
```

```
# create polar plot

numYears = length(unique(year(df$timestamp)))

plot <- ggseasonplot(df.ts,
                     col = viridis(numYears),
                     main = "Seasonal Plot Polar per Month over 10 Years\n",
                     ylab = "Average Daily Energy Consumption per Month \n(kWh/day)\n",
                     polar = TRUE
                     )

# show plot (interactive version with plotly unfortunately not possible)
plot
```




```
# save static plot as png (optional)
ggsave("images/plotSeasonalPolar.png", plot)
```

11.3.4 Discussion

This representation emphasizes the high consumption in summer very well, which could undoubtedly be reduced in a residential building. The Years 2018 and 2019 show, that this optimization was done.

To emphasize this optimization please refer to chapter 11.4.

11.4 Before/After Optimization

11.4.1 Goal

To highlight an energy optimization in a season diagram, we can gray out the seasons before the optimization and only highlight the monthly values after the optimization. To better quantify the success, we can calculate and display the confidence interval of the years before.

We will create the following plot:

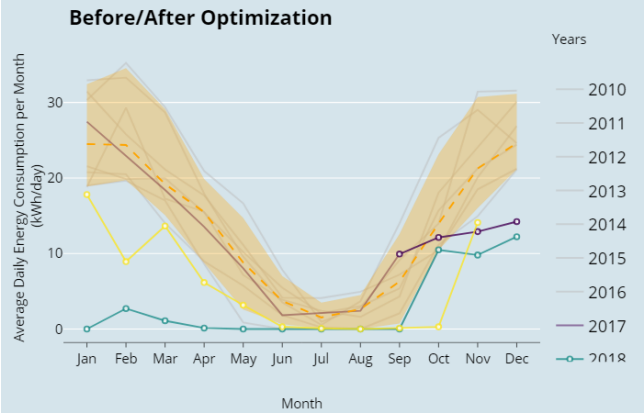


Figure 11.8: Seasonal Plot Overlapping Before/After

11.4.2 Data Basis

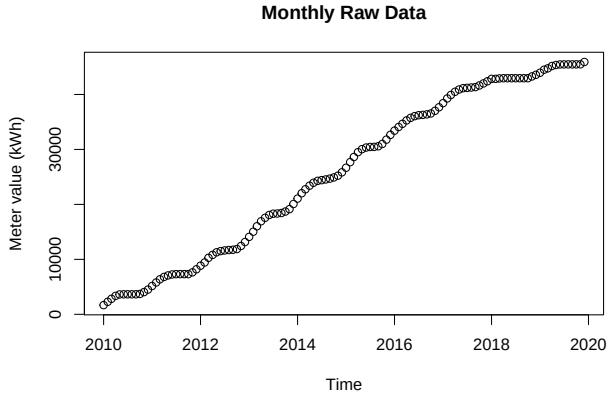


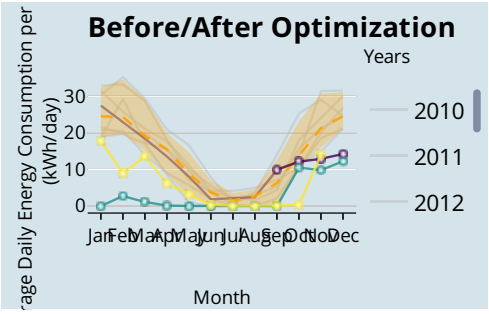
Figure 11.9: Raw Data for Seasonal Plot Overlapping Before/After Optimization

11.4.3 Solution

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

```
library(redutils)
library(dplyr)
library(plotly)
library(htmlwidgets)
library(ggthemes)

# load csv file
df <- read.csv2("https://github.com/hslu-ige-laes/edar/raw/master/sampleData/flatHeatAndHotWater.csv",
                stringsAsFactors=FALSE)

# filter flat
df <- df %>% select(timestamp, Adr02_energyHeat)

colnames(df) <- c("timestamp", "meterValue")

# calculate consumption value per month
# pay attention, the value of 2010-02-01 00:00:00 represents the meter reading on february first,
# so the consumption for february first is value(march) - value(february)!
df <- df %>% dplyr::mutate(value = lead(meterValue) - meterValue)

# remove counter value column and calculate consumption per day instead of month
df <- df %>%
  select(-meterValue) %>%
  mutate(value = value /lubridate::days_in_month(timestamp))

# value correction (outlier because of commissioning)
df[1,2] <- 19

# create plot
```

```
plot <- plotSeasonalXYBeforeAfter(df,
                                  dateOptimization = "2017-09-01",
                                  locTimeZone = "Europe/Zurich",
                                  main = "Before/After Optimization",
                                  ylab = "Average Daily Energy Consumption per Month \n(kWh/day)\n"
                                  )

# change theme (optional)
plot <- plot + ggthemes::theme_economist()

# make plot interactive (optional)
plotly <- plotly::ggplotly(plot)

# show plot
plotly
```




```
# save interactive plot as html (optional)
library(htmlwidgets)
htmlwidgets::saveWidget(plotly, "plotEnergyConsBeforeAfter.png")
```

11.4.4 Discussion

- One can clearly see the impact of the optimization in mid-2017
- And as well the too low setting of January 2018 where the thermostat of the flat got deactivated

- The confidence band shows as well the year 2013 which had an unusual high consumption from February to June

Chapter 12

Heat Maps

Temporal patterns can show at what time certain devices are exposed to how much load. Heatmaps show quantitative values based on two axes. These can be day and hours, month and day, etc. The cell at the intersection contains the corresponding value. Depending on the context, this can be an average value, a maximum/minimum value, etc.

Heatmaps allow the viewer to see short-term patterns and long-term trends in the data with a higher temporal granularity.

This chapter shows different types of heat maps which are useful for energy data analysis and pattern recognition.

12.1 Calendar

12.1.1 Goal

Create a calendar heat map with daily energy consumption values:

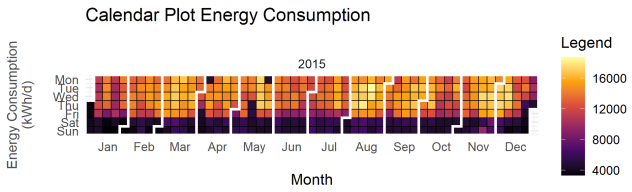


Figure 12.1: Calendar Heat Map

12.1.2 Data Basis

Daily energy consumption values of one whole year in an interval of 15mins.

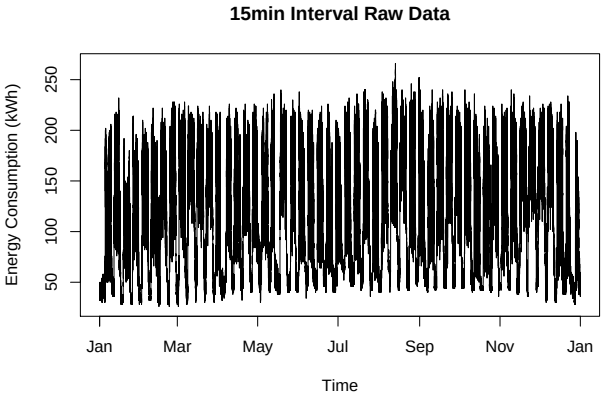
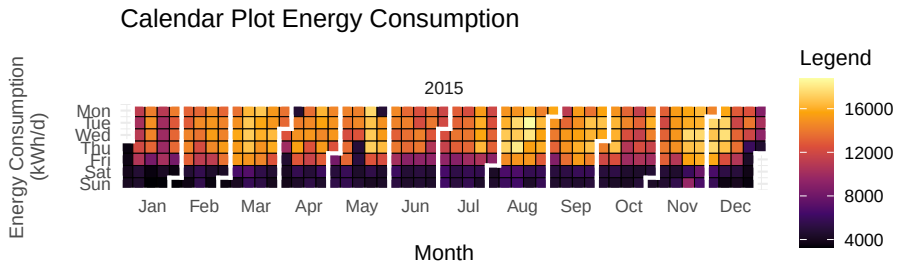


Figure 12.2: Raw Data for Decomposition Plot Short Term

12.1.3 Solution

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

```
library(ggplot2)
library(ggTimeSeries)
library(plotly)
library(lubridate)
library(dplyr)
library(tidyquant)

data <- readRDS(system.file("sampleData/eboBookEleMeter.rds", package = "redutils"))

data <- data[-nrow(data),]

data$timestamp <- parse_date_time(data$timestamp,
                                  order = "YmdHMS",
                                  tz = "UTC")

data$day <- as.Date(lubridate::floor_date(data$timestamp,"day"))

data <- data %>%
  select(-timestamp)

data.plot <- data %>%
  dplyr::group_by(day) %>%
  dplyr::mutate(calcVal = sum(value, na.rm = TRUE)) %>%
  ungroup() %>%
  select(-value) %>%
  unique()

plot <- ggplot_calendar_heatmap(data.plot,
                            "day",
                            "calcVal",
                            monthBorderSize = 1,
                            monthBorderColour = "white",
                            monthBorderLineEnd = "square") +
  scale_fill_viridis_c(option = "B") +
  theme_minimal() +
  theme(axis.title.y = element_text(colour = "grey30", size = 10, face = "plain"),
                  )+
  labs(x = "\nMonth",
       y = "Energy Consumption\n(kWh/d)\n ",
       fill = "Legend") +
  facet_wrap(~Year, ncol = 1) +
  ggtitle("Calendar Plot Energy Consumption\n")

plot
```




```
# create the interactive plot (optional, uncomment line)
#ggplotly(plot)
```

```
# save static plot as png (optional)
ggsave("images/plotHeatMapCalendar.png", plot)
```

12.1.4 Discussion

Some findings:

- first two days in year minimal consumption
- 6th of April: Easter Monday
- 25th of May: Whitmonday (de: Pfingstmontag)
- More usage in August
- In November one Sunday with unusual high consumption
- On Fridays in general less consumption

12.2 Median-Weeks

12.2.1 Goal

Create an energy consumption heat map of a median week depending on the season of the year:

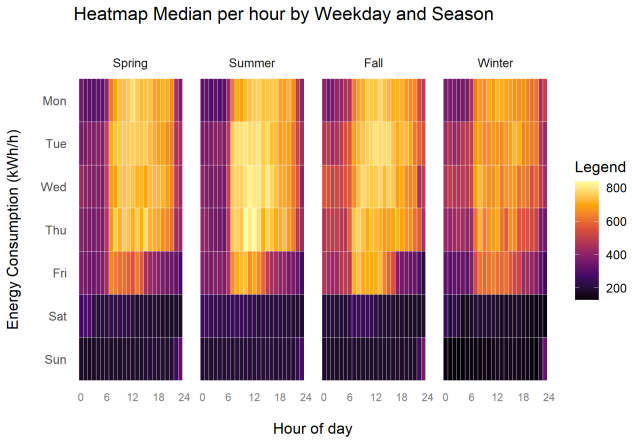


Figure 12.3: Heat Map of Median Weeks per Season of Year

12.2.2 Data Basis

Daily energy consumption values of one whole year in an interval of 15mins.

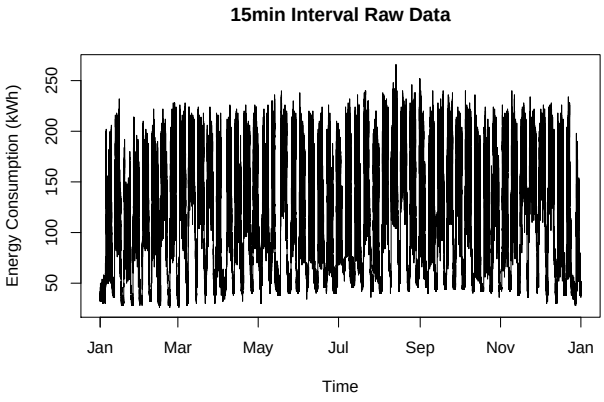
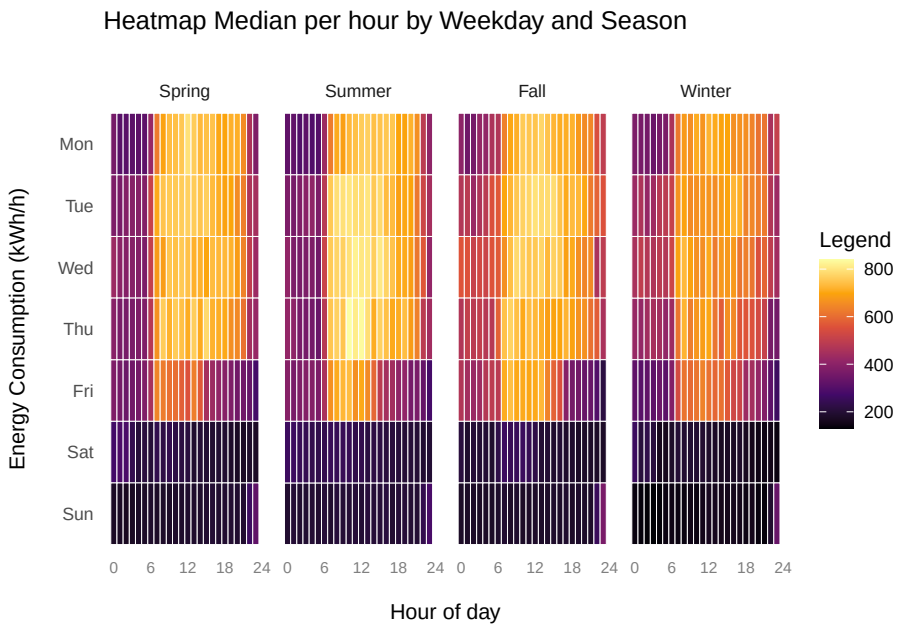


Figure 12.4: Raw Data for Decomposition Plot Short Term

12.2.3 Solution

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

```
library(redutils)
library(plotly)

data <- readRDS(system.file("sampleData/eboBookEleMeter.rds", package = "redutils"))
plot <- plotHeatmapMedianWeeks(data, locTimeZone = "Europe/Zurich")

# show the static plot
plot
```




```
# create the interactive plot (optional, uncomment line)
#ggplotly(plot)
```

```
# save static plot as png (optional)
ggsave("images/plotHeatMapMedianWeeks.png", plot)
```

12.2.4 Discussion

Some findings:

- Increased consumption at midnight, but not visible in Summer, probably heating affected

- Clearly less consumption at weekends, starting already friday afternoon
- High peaks on Tuesdays and Thursdays in summer

Chapter 13

Typical Daily Profiles

An important step in energy data analysis is the recognition of patterns. For this purpose, the data are first aggregated to hourly values, averaged and displayed as typical daily patterns. The daily profiles can be displayed separately for the weekdays and the seasons.

This chapter shows some useful visualizations to accommodate this recognition.

13.1 Overview

13.1.1 Goal

Create an overview of typical daily profiles per weekday and season of year with a confidence band where most of the values lie:

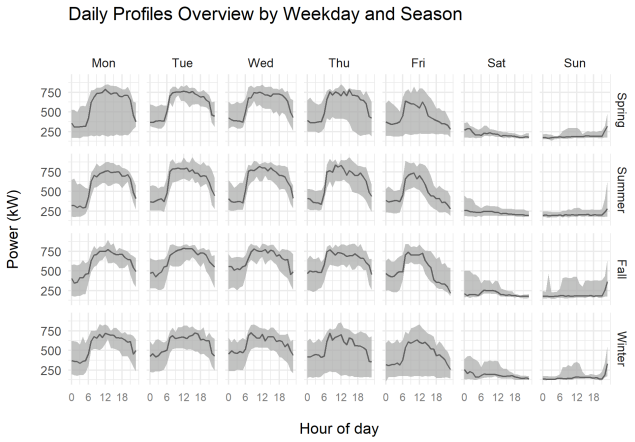


Figure 13.1: Overview of Daily Profiles by Weekday and Season

13.1.2 Data Basis

Energy consumption values of one whole year in an interval of 15mins.

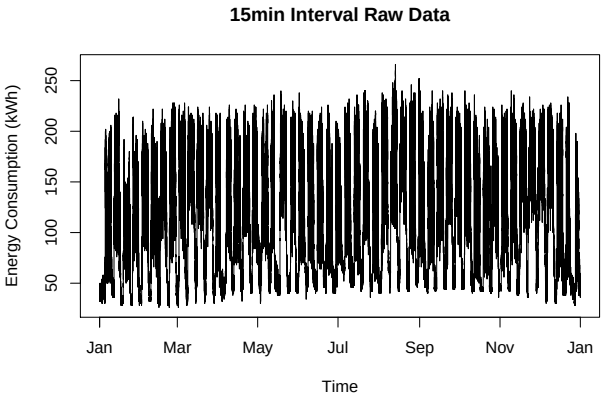
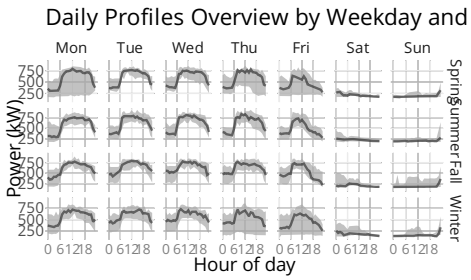


Figure 13.2: Raw Data for Decomposition Plot Short Term

13.1.3 Solution

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

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

# load time series data
df <- readRDS(system.file("sampleData/eboBookEleMeter.rds", package = "redutils"))

plot <- plotDailyProfilesOverview(df,
                                  locTimeZone = "Europe/Zurich",
                                  main = "Daily Profiles Overview by Weekday and Season",
                                  ylab = "Power (kW)",
                                  col = "black",
                                  confidence = 95.0)

ggplotly(plot)
```




```
# save static plot as png (optional)
ggsave("images/plotDailyProfOverview.png", plot)
```

13.2 Overlayed

13.2.1 Goal

Create a plot of all data per week by season of the year:

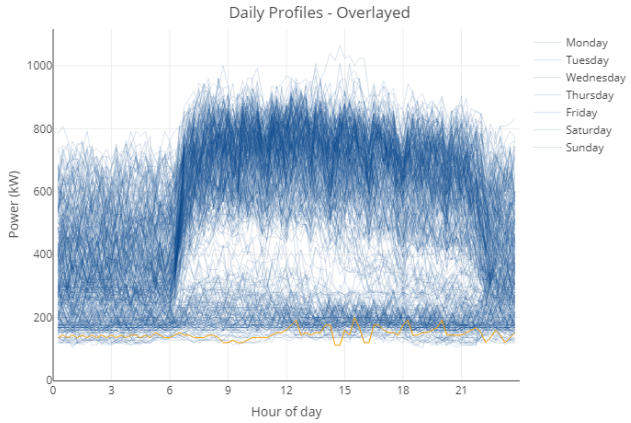


Figure 13.3: Overlayed Daily Profiles

13.2.2 Data Basis

Energy consumption values of one whole year in an interval of 15mins.

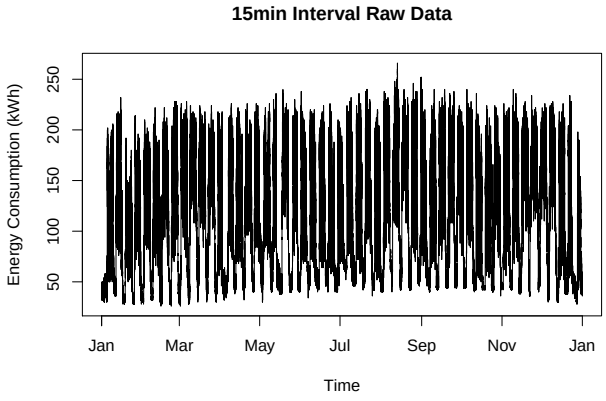


Figure 13.4: Raw Data for Decomposition Plot Short Term

13.2.3 Solution

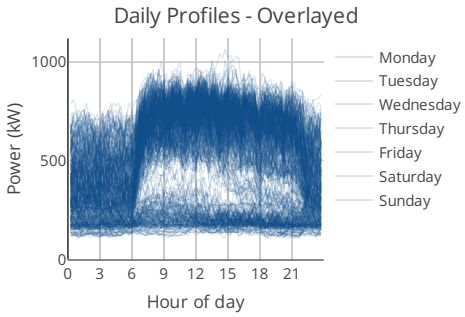
Create a new script, copy/paste the following code and run it:

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

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

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

# load time series data
df <- readRDS(system.file("sampleData/eboBookEleMeter.rds", package = "redutils"))
df <- dplyr::mutate(df, value = value * 4)

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

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

# plot graph with all time series
rangeX <- seq(0,24,0.25)
maxValue <- max(df$value, na.rm = TRUE)*1.05

plot <- 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$timestamp, "%H:%M"),
                                "<br>Date: ", format(df$timestamp, "%Y-%m-%d"),
                                "<br>Value: %{y:.0f}")) %>%
  # workaround with add_trace to have fixed y axis when selecting a dedicated day
  add_trace(x = 0, y = 0, type = "scatter", showlegend = FALSE, opacity=0) %>%
  add_trace(x = 24, y = maxValue, type = "scatter", showlegend = FALSE, opacity=0) %>%
  layout(title = "Daily Profiles - Overlayed",
         showlegend = TRUE,
         xaxis = list(
           title = "Hour of day",
           range = rangeX,
           tickvals = list(0, 3, 6, 9, 12, 15, 18, 21),
           showline=TRUE
           ),
           yaxis = list(
```

```
            title = "Power (kW)",
            range = c(0, maxValue)
          )
        ) %>%
  highlight(on = "plotly_hover",
            off = "plotly_doubleclick",
            color = "orange",
            opacityDim = 1.0,
            selected = attrs_selected(showlegend = FALSE)) %>% # this hides elements in the legend
  plotly::config(modeBarButtons = list(list("toImage")), displaylogo = FALSE)

# show plot
plot
```



13.3 Mean

13.3.1 Goal

Create a plot of mean data per week:

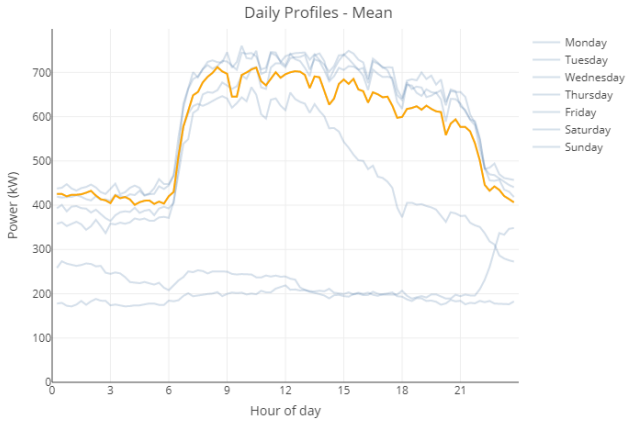


Figure 13.5: Mean Daily Profiles per Weekday

13.3.2 Data Basis

Energy consumption values of one whole year in an interval of 15mins.

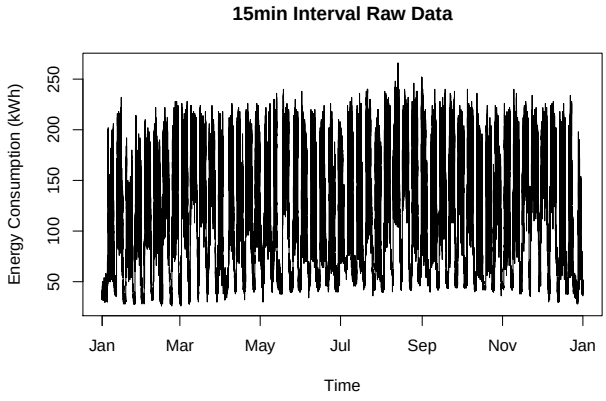


Figure 13.6: Raw Data for Decomposition Plot Short Term

13.3.3 Solution

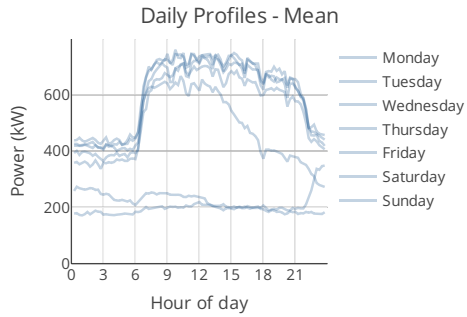
Create a new script, copy/paste the following code and run it:

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

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

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

# load time series data
df <- readRDS(system.file("sampleData/eboBookEleMeter.rds", package = "redutils"))
df <- dplyr::mutate(df, value = value * 4)

# add metadata for later grouping and visualization purposes
df$x <- hour(df$timestamp) + minute(df$timestamp)/60 + second(df$timestamp) / 3600
df$weekday <- weekdays(df$timestamp)
df$weekday <- factor(df$weekday, c("Monday","Tuesday","Wednesday","Thursday","Friday","Saturday", "Sunday")

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

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

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

# plot graph with mean values
maxValMean <- max(df$dayTimeMean, na.rm = TRUE)*1.05

plot <- 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$timestamp, "%H:%M"),
                                "<br>Mean: %{y:.0f}")) %>%
  # workaround with add_trace to have fixed y axis when selecting a dedicated day
  add_trace(x = 0, y = 0, type = "scatter", showlegend = FALSE, opacity=0) %>%
  add_trace(x = 24, y = maxValMean, type = "scatter", showlegend = FALSE, opacity=0) %>%
  layout(title = "Daily Profiles - Mean",
         showlegend = TRUE,
         xaxis = list(
```

```
          title = "Hour of day",
          tickvals = list(0, 3, 6, 9, 12, 15, 18, 21)
        ),
        yaxis = list(
            title = "Power (kW)",
            range = c(0, maxValMean)
        )
  ) %>%
  highlight(on = "plotly_hover",
            off = "plotly_doubleclick",
            color = "orange",
            opacityDim = 0.7,
            selected = attrs_selected(showlegend = FALSE)) %>% # this hides elements in the legend
  plotly::config(modeBarButtons = list(list("toImage")), displaylogo = FALSE)

# show plot
plot
```



13.4 Decomposed

13.4.1 Goal

Create a plot of detrended mean data per week as recommended in "Building electricity consumption: Data analytics of building operations with classical time series decomposition and case based subsetting", Pickering et al, 2018:

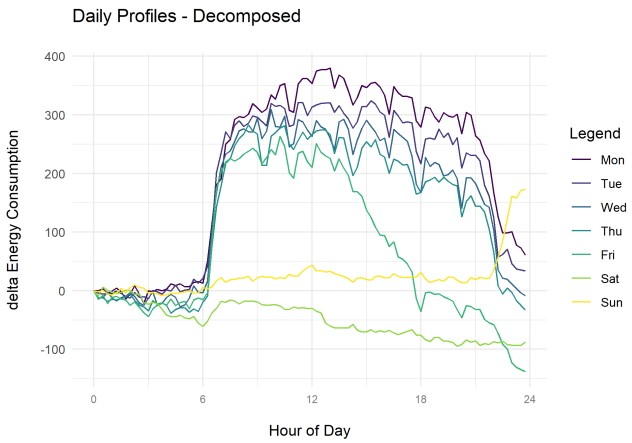


Figure 13.7: Mean Daily Profiles per Weekday

13.4.2 Data Basis

Energy consumption values of one whole year in an interval of 15mins.

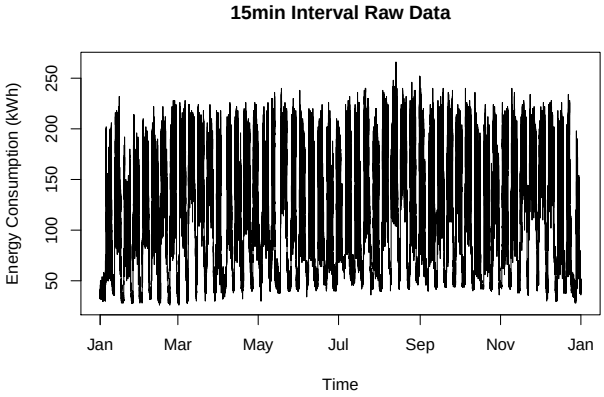
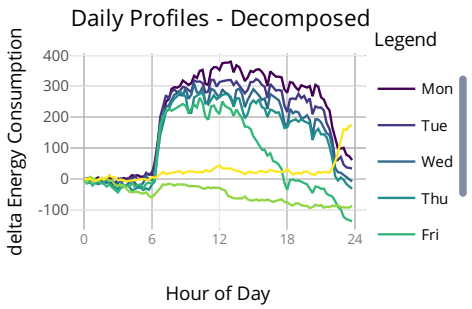


Figure 13.8: Raw Data for Decomposition Plot Short Term

13.4.3 Solution

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

```
library(plotly)
library(redutils)

# load time series data
df <- readRDS(system.file("sampleData/eboBookEleMeter.rds", package = "redutils"))
df <- dplyr::mutate(df, value = value * 4)

plot <- plotDailyProfilesDecomposed(df, locTimeZone = "Europe/Zurich")

ggplotly(plot)
```




```
# save static plot as png (optional)
ggsave("images/plotDailyProfDecomposed.png", plot)
```

Chapter 14

Sum Frequency Diagrams

This type of visualization has a long tradition in energy data analyses and various calculation methods are still based on it.

14.1 Sum Frequency Days

14.1.1 Goal

You want to create a sum frequency plot of a temperature series for each day in a year:

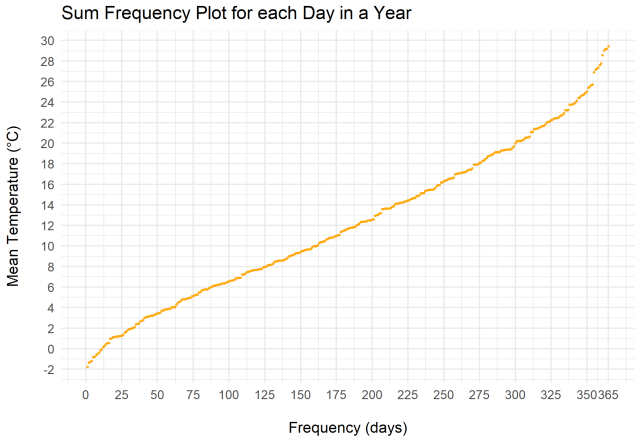


Figure 14.1: Sum Frequency Plot Temperature Days

14.1.2 Data Basis

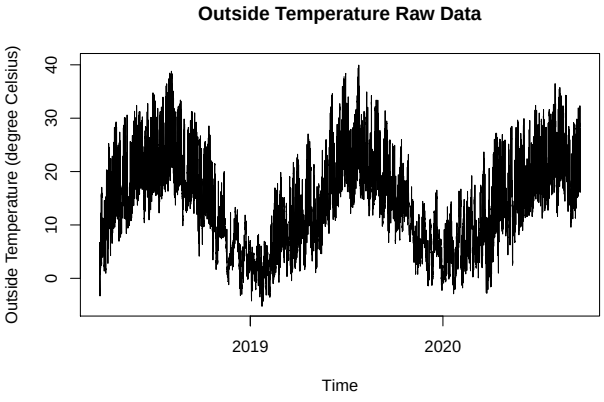


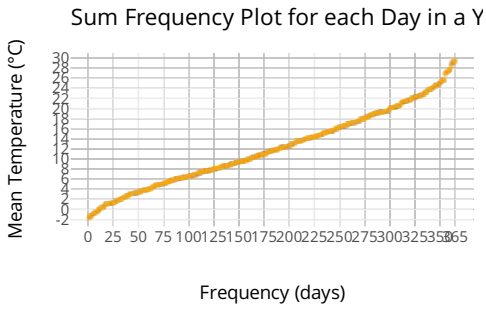
Figure 14.2: Outside Temperature Raw Data for Sum Frequency Days Plot

14.1.3 Solution

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

```
library(ggplot2)
library(plotly)
library(dplyr)
library(lubridate)
library(zoo)
library(mgcv)

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

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

# rename columns
colnames(df) <- c("time", "value")

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

# filter year
filterYear <- "2019"

df <- filter(df, year == filterYear)

df <- df %>%
  group_by(day) %>%
  dplyr::summarise(meanValue = mean(value, na.rm = TRUE)) %>%
  ungroup()

# Fill missing values with NA
grid.df <- data.frame(day = seq(as.Date(paste0(filterYear, "-01-01")),
                                as.Date(paste0(filterYear, "-12-31")),
                                 by = "days"))
df <- merge(df, grid.df, all = TRUE)

# replace NA with interpolation
df$meanValue <- na.approx(df$meanValue)

tempMin <- floor(min(df$meanValue, na.rm = TRUE))
tempMax <- ceiling(max(df$meanValue, na.rm = TRUE))

# create new data frame with sorted values
data <- data.frame(sort(df$meanValue))
data$day <- as.numeric(row.names(data))
colnames(data) <- c("meanValue", "day")

# static chart with ggplot
plot <- ggplot2::ggplot(data) +
  ggplot2::geom_point(aes(x = day,
                          y = meanValue),
```

```
                      colour = "orange",
                      alpha = 0.8,
                      size = 0.5
                      ) +
  ggtitle("Sum Frequency Plot for each Day in a Year") +
  ylab("Mean Temperature (\u00B0C)\n") +
  xlab("\nFrequency (days)") +
  theme_minimal() +
  scale_x_continuous(breaks = append(seq(0, 365, 25), 365)) +
  scale_y_continuous(breaks = seq(tempMin, tempMax, 2))

# interactive chart
plotly::ggplotly(plot)
```




```
# save static plot as png
ggsave("images/plotSumFrequencyDays.png", plot)
```

14.2 Sum Frequency Hours

14.2.1 Goal

You want to create a sum frequency plot of a temperature series for each hour in a year:

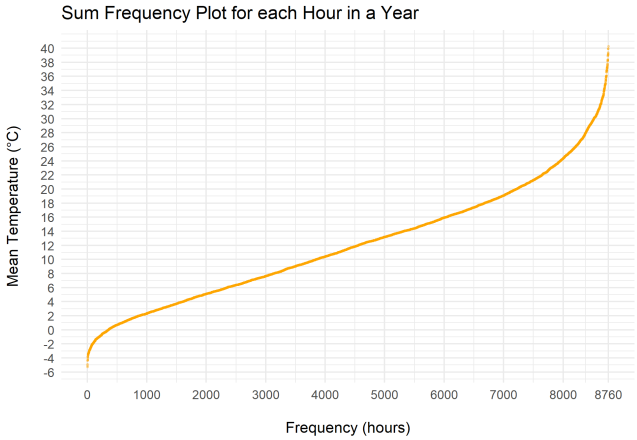


Figure 14.3: Sum Frequency Plot Temperature

14.2.2 Data Basis

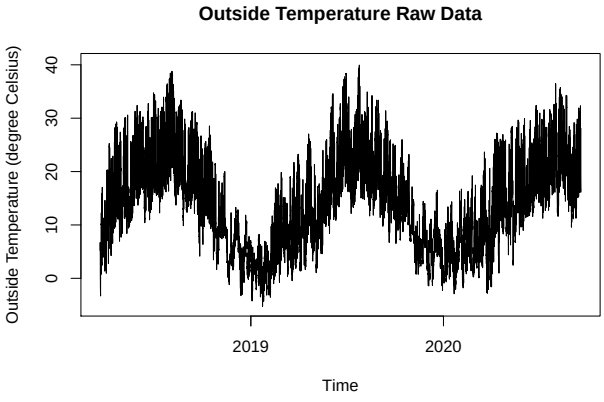


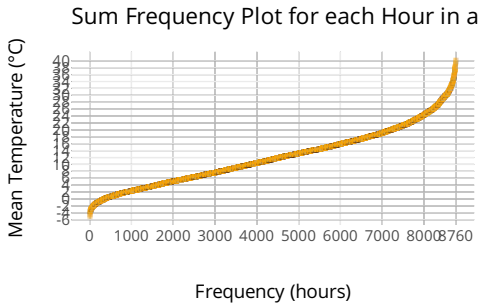
Figure 14.4: Outside Temperature Raw Data for Sum Frequency Hours Plot

14.2.3 Solution

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

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

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

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

# rename columns
colnames(df) <- c("time", "value")

df$day <- as.Date(cut(df$time, breaks = "day"))
df$hour <- as.POSIXct(cut(df$time, breaks = "hour"), tz = "Europe/Zurich")
df$year <- year(df$time)

# filter year

# edit year!!!
filterYear <- "2019"

df <- filter(df, year == filterYear)

df <- df %>%
  group_by(hour) %>%
  dplyr::summarise(meanValue = mean(value, na.rm = TRUE)) %>%
  ungroup()

# Fill missing values with NA
grid.df <- data.frame(hour = seq(as.POSIXct(paste0(filterYear, "-01-01 00:00:00"),
                                            tz = "Europe/Zurich"),
                                 as.POSIXct(paste0(filterYear, "-12-31 23:00:00"),
                                            tz = "Europe/Zurich"),
                                 by = "hours"))
df <- merge(df, grid.df, all = TRUE)

# replace NA with interpolation
df$meanValue <- na.approx(df$meanValue)

tempMin <- floor(min(df$meanValue, na.rm = TRUE))
tempMax <- ceiling(max(df$meanValue, na.rm = TRUE))

# create new data frame with sorted values
data <- data.frame(sort(df$meanValue))
data$hour <- as.numeric(row.names(data))
colnames(data) <- c("meanValue", "hour")
```

```
# static chart with ggplot
plot <- ggplot2::ggplot(data) +
  ggplot2::geom_point(aes(x = hour,
                          y = meanValue),
                      colour = "orange",
                      alpha = 0.3,
                      size = 0.5
                      ) +
  ggtitle("Sum Frequency Plot for each Hour in a Year") +
  ylab("Mean Temperature (\u00B0C)\n") +
  xlab("\nFrequency (hours)") +
  theme_minimal() +
  scale_x_continuous(breaks = append(seq(0, 8760, 1000), 8760)) +
  scale_y_continuous(breaks = seq(tempMin, tempMax, 2))

# interactive chart
plotly::ggplotly(plot)
```




```
# save static plot as png
ggsave("images/plotSumFrequencyHour.png", plot)
```

Chapter 15

Comfort Plots

Satisfied and healthy residents are important, so energetic optimization should improve comfort as much as possible and certainly not adversely affect it. For this reason, the so-called comfort monitoring is gaining more and more importance.

The visualizations in this part focus on some typical comfort related visualizations.

15.1 Mollier hx Diagram

15.1.1 Goal

You want to plot a mollier h-x diagram:

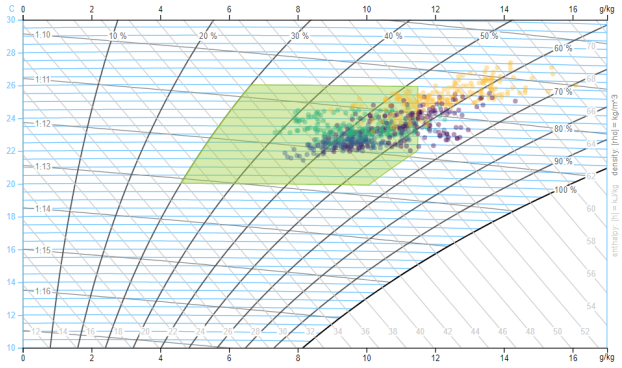


Figure 15.1: Mollier hx Diagram with comfort zone

15.1.2 Data Basis

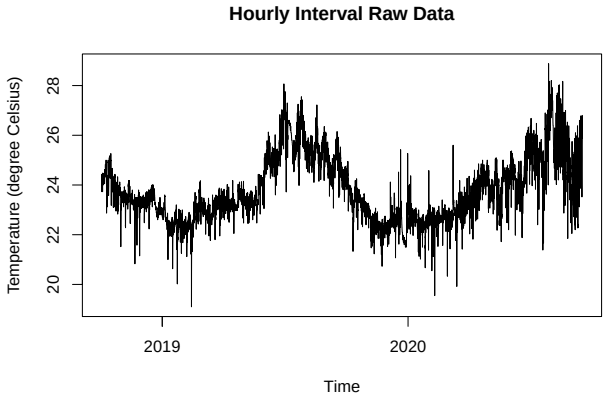


Figure 15.2: Raw Data Temperature and Humidity for Mollier hx Diagram

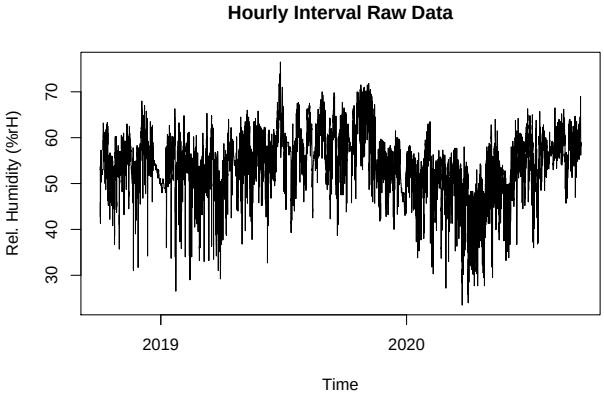


Figure 15.3: Raw Data Temperature and Humidity for Mollier hx Diagram

15.1.3 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). If you have not yet installed this package, proceed as follows:

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

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

```
library(redutils)
library(dplyr)
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) %>%
  summarize(tempMean = mean(as.numeric(FlatA_Temp)),
            humMean = mean(as.numeric(FlatA_Hum))
            ) %>%
  ungroup()

# plot mollier hx diagram
plot <- plotMollierHx(data)

# show plot
plot
```

15.1.4 See also

If you are interested in implementing this plot in your own dashboard you can check the free D3 implementation of it: https://github.com/hslu-ige-laes/d3-mollierhx

15.2 Temperature versus Humidity

15.2.1 Goal

You want to create a temperature versus humidity comfort plot:

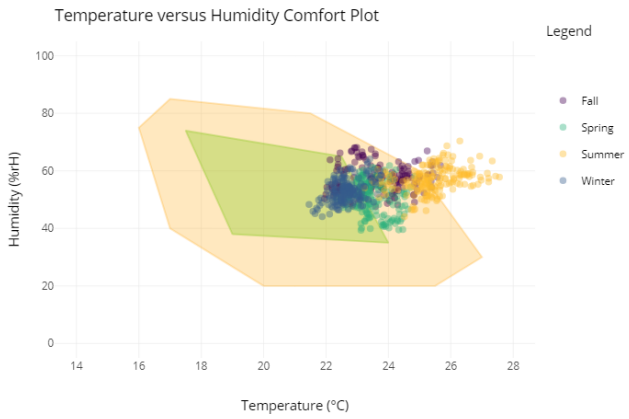


Figure 15.4: Temperature versus Humidity Comfort Plot

15.2.2 Data Basis

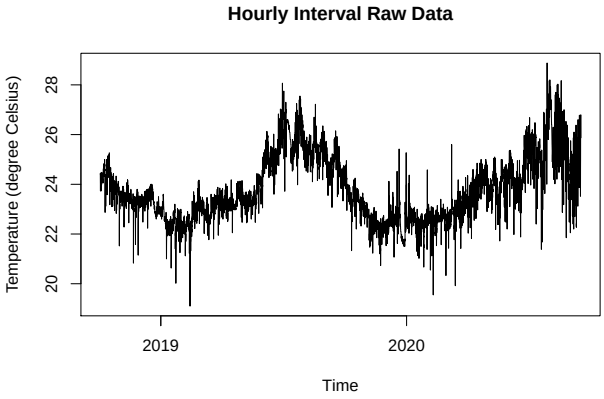


Figure 15.5: Raw Data Temperature and Humidity for Temp vs. Hum Comfort Plot

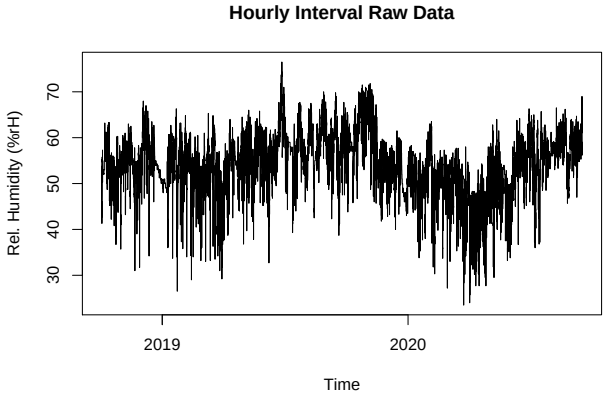


Figure 15.6: Raw Data Temperature and Humidity for Temp vs. Hum Comfort Plot

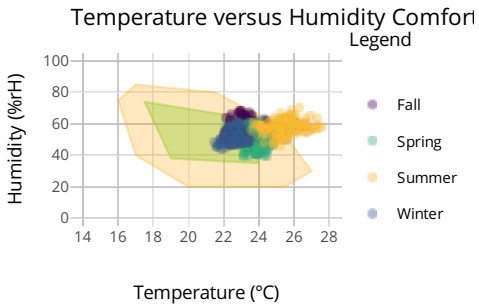
15.2.3 Solution

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

```
library(redutils)
library(dplyr)
library(lubridate)
library(plyr)

# 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()
colnames(data) <- c("time", "tempRaw", "humidityRaw")

# 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) %>%
  dplyr::summarize(temperature = mean(as.numeric(tempRaw), na.rm = TRUE),
                   humidity = mean(as.numeric(humidityRaw), na.rm = TRUE)
  ) %>%
  ungroup() %>%
  na.omit()

# calculate season
data$day <- parse_date_time(data$day, "Ymd", tz = "Europe/Zurich")
```

```
data$season <- getSeason(data$day)

# create plot
# axis properties
minx <- floor(min(14.0,min(data$temperature)))
maxx <- ceiling(max(28.0,max(data$temperature)))
miny <- 0.0
maxy <- 100.0

# comfort zones
df.zoneNotComfortable <- data.frame(Temp = c(minx,minx,maxx, maxx, minx),
                                    Hum = c(miny,maxy,maxy, miny, miny),
                                    Zones = "uncomfortable")
df.zoneStillComfortable <- data.frame(Temp = c(20,17,16,17,21.5,25,27,25.5,20),
                                      Hum = c(20,40,75,85,80,60,30,20,20),
                                      Zones = "Still comfortable")

df.zoneComfortable <- data.frame(Temp = c(19,17.5,22.5,24,19),
                                 Hum = c(38,74,65,35,38),
                                 Zones = "Comfortable")

df.zones <- rbind.fill(df.zoneNotComfortable, df.zoneStillComfortable)
df.zones <- rbind.fill(df.zones, df.zoneComfortable)

plot <- ggplot() +
  geom_polygon(data = df.zoneStillComfortable,
               aes(x = Temp,
                   y = Hum),
               alpha = 0.25,
               color = "orange",
               fill = "orange",
               name = "Comofort Zone") +
  geom_polygon(data = df.zoneComfortable,
               aes(x = Temp,
                   y = Hum),
               alpha = 0.4,
               color = "yellowgreen",
               fill = "yellowgreen") +
    geom_point(data = data,
              aes(x = temperature,
                  y = humidity,
                  fill = season,
                  colour = season,
                  text = paste0("Temp:    ", sprintf("%.1f \u00B0C", temperature),
                                "<br />Hum:     ", sprintf("%.1f %%rH", humidity),
                                "<br />Date:     ", day,
                                "<br />Season: ", season)
                  ),
              alpha = 0.4) +
  ggtitle("Temperature versus Humidity Comfort Plot") +
  labs(x = "\nTemperature (\u00B0C)",
       y = "Humidity (%rH)\n",
       fill = "",
       colour = "Legend") +
  scale_x_continuous(breaks = seq(minx, maxx, 2),
                     limits = c(minx, maxx)) +
  scale_y_continuous(breaks = seq(miny, maxy, 20),
```

```
                    limits = c(miny, maxy)) +
  scale_color_manual(values = c("#440154", "#2db27d", "#febc2b", "#365c8d")) +
  scale_fill_manual(values = c("#440154", "#2db27d", "#febc2b", "#365c8d")) +
  theme_minimal()

# interactive chart
ggplotly(plot, tooltip = "text")
```




```
# save static plot as png
#ggsave("images/comfortTempHum.png", plot)
```

15.3 SIA 180 Thermal Comfort

15.3.1 Goal

You want to plot a diagram like the one from the SIA 180:2014

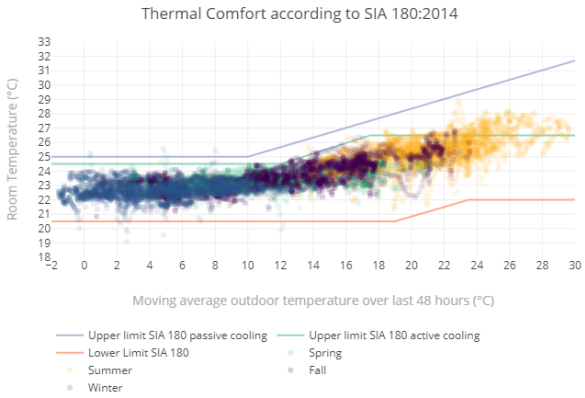


Figure 15.7: Thermal Comfort according to SIA 180:2014

15.3.2 Data Basis

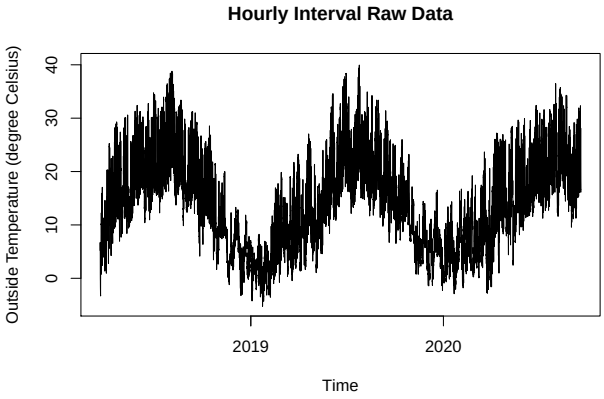


Figure 15.8: Raw Data Room and Outdoor Temperature for SIA180 Thermal Comfort Plot

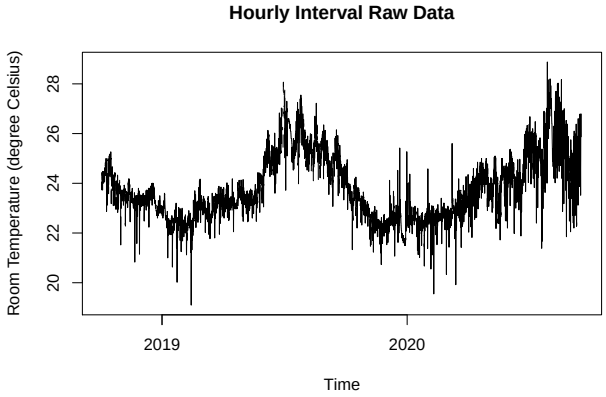


Figure 15.9: Raw Data Room and Outdoor Temperature for SIA180 Thermal Comfort Plot

15.3.3 Solution

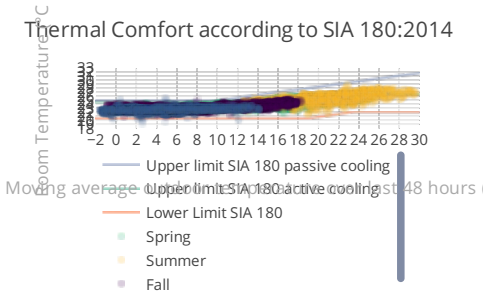
Create a new script, copy/paste the following code and run it:

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

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

dfTempOa$time <- parse_date_time(dfTempOa$time,
                                     order = "YmdHMS",
                                     tz = "UTC")

dfTempOa$hour <- cut(dfTempOa$time, breaks = "hour")

dfTempOa <- dfTempOa %>%
  group_by(hour) %>%
  dplyr::mutate(tempMean = mean(centralOutsideTemp)) %>%
  ungroup() %>%
  select(time, tempMean) %>%
  unique()

# Fill missing values with NA
grid.df <- data.frame(time = seq(min(dfTempOa$time, na.rm = TRUE),
                                 max(dfTempOa$time, na.rm = TRUE),
                                 by = "hour"))
dfTempOa <- merge(dfTempOa, grid.df, all = TRUE)
```

```r
dfTempOa <- dfTempOa %>%
  dplyr::mutate(tempOa = rollmean(tempMean, 48, fill = NA, align = "right"))

dfTempOa <- dfTempOa %>%
  select(time, tempOa) %>%
  unique() %>%
  na.omit()


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

dfTempR$time <- parse_date_time(dfTempR$time,
                                    order = "YmdHMS",
                                    tz = "UTC")

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

dfTempR$hour <- cut(dfTempR$time, breaks = "hour")

dfTempR <- dfTempR %>%
  group_by(hour) %>%
  dplyr::mutate(tempR = mean(FlatA_Temp)) %>%
  ungroup() %>%
  select(time, tempR) %>%
  unique()

# Fill missing values with NA
grid.df <- data.frame(time = seq(min(dfTempR$time, na.rm = TRUE),
                                 max(dfTempR$time, na.rm = TRUE),
                                 by = "hour"))
dfTempR <- merge(dfTempR, grid.df, all = TRUE)

data <- merge(dfTempR, dfTempOa, all = TRUE) %>% unique() %>% na.omit()

data$season <- redutils::getSeason(data$time)

# plot diagram

# axis properties
minx <- floor(min(0, min(data$tempOa)))
maxx <- ceiling(max(28, max(data$tempOa)))

miny <- floor(min(21.0,min(data$tempR)))-1
maxy <- ceiling(max(32.0,max(data$tempR)))+1

# line setpoint heat
df.heatSp <- data.frame(tempOa = c(minx, 19, 23.5, maxx), tempR = c(20.5, 20.5, 22, 22))

# line setpoint cool according to SIA 180:2014 Fig. 4
df.coolSp1 <- data.frame(tempOa = c(minx, 12, 17.5, maxx),tempR = c(24.5, 24.5, 26.5, 26.5))

# line setpoint cool according to SIA 180:2014 Fig. 3
df.coolSp2 <- data.frame(tempOa = c(minx, 10, maxx),tempR = c(25, 25, 0.33 * maxx + 21.8))
```

```
plot <- data %>%
  plot_ly(showlegend = TRUE) %>%
  add_lines(data = df.coolSp2,
            x = ~tempOa,
            y = ~tempR,
            name = "Upper limit SIA 180 passive cooling",
            opacity = 0.7,
            color = "#FDE725FF",
            hoverinfo = "text",
            text = ~ paste("Upper limit SIA 180 passive cooling",
                           "<br />TempR:   ", sprintf("%.1f \u00B0C", tempR),
                           "<br />TempOa: ", sprintf("%.1f \u00B0C", tempOa)
            )
  ) %>%
  add_lines(data = df.coolSp1,
            x = ~tempOa,
            y = ~tempR,
            name = "Upper limit SIA 180 active cooling",
            opacity = 0.7,
            color = "#1E9B8AFF",
            hoverinfo = "text",
            text = ~ paste("Upper limit SIA 180 active cooling",
                           "<br />TempR:   ", sprintf("%.1f \u00B0C", tempR),
                           "<br />TempOa: ", sprintf("%.1f \u00B0C", tempOa)
            )        ) %>%
  add_lines(data = df.heatSp,
            x = ~tempOa,
            y = ~tempR,
            name = "Lower Limit SIA 180",
            opacity = 0.7,
            color = "#440154FF",
            hoverinfo = "text",
            text = ~ paste("Lower Limit SIA 180",
                           "<br />TempR:   ", sprintf("%.1f \u00B0C", tempR),
                           "<br />TempOa: ", sprintf("%.1f \u00B0C", tempOa)
            )        ) %>%
  add_markers(data = data %>% filter(season == "Spring"),
              x = ~tempOa,
              y = ~tempR,
              name = "Spring",
              marker = list(color = "#2db27d", opacity = 0.1),
              hoverinfo = "text",
              text = ~ paste("TempR:   ", sprintf("%.1f \u00B0C", tempR),
                             "<br />TempOa: ", sprintf("%.1f \u00B0C", tempOa),
                             "<br />Date:       ", time,
                             "<br />Season:  ", season
              )
  ) %>%
  add_markers(data = data %>% filter(season == "Summer"),
              x = ~tempOa,
              y = ~tempR,
              name = "Summer",
              marker = list(color = "#febc2b", opacity = 0.1),
              hoverinfo = "text",
              text = ~ paste("TempR:   ", sprintf("%.1f \u00B0C", tempR),
                             "<br />TempOa: ", sprintf("%.1f \u00B0C", tempOa),
```

```
                              "<br />Date:       ", time,
                              "<br />Season:  ", season
                )
  ) %>%
  add_markers(data = data %>% filter(season == "Fall"),
                x = ~tempOa,
                y = ~tempR,
                name = "Fall",
                marker = list(color = "#440154", opacity = 0.1),
                hoverinfo = "text",
                text = ~ paste("TempR:   ", sprintf("%.1f \u00B0C", tempR),
                              "<br />TempOa: ", sprintf("%.1f \u00B0C", tempOa),
                              "<br />Date:       ", time,
                              "<br />Season:  ", season
                )
  ) %>%
  add_markers(data = data %>% filter(season == "Winter"),
                x = ~tempOa,
                y = ~tempR,
                name = "Winter",
                marker = list(color = "#365c8d", opacity = 0.1),
                hoverinfo = "text",
                text = ~ paste("TempR:   ", sprintf("%.1f \u00B0C", tempR),
                              "<br />TempOa: ", sprintf("%.1f \u00B0C", tempOa),
                              "<br />Date:       ", time,
                              "<br />Season:  ", season
                )
  ) %>%
  layout(
    title = "Thermal Comfort according to SIA 180:2014",
    xaxis = list(title = "Moving average outdoor temperature over last 48 hours (\u00B0C)",
                 range = c(minx, maxx),
                 zeroline = FALSE,
                 tick0 = minx,
                 dtick = 2,
                 titlefont = list(size = 14, color = "darkgrey")),
    yaxis = list(title = "Room Temperature (\u00B0C)",
                 range = c(miny, maxy),
                 dtick = 1,
                 titlefont = list(size = 14, color = "darkgrey")),
    hoverlabel = list(align = "left"),
    margin = list(l = 80, t = 50, r = 50, b = 10),
    legend = list(orientation = 'h',
                  x = 0.0,
                  y = -0.3)
  ) %>%

  plotly::config(modeBarButtons = list(list("toImage")),
                 displaylogo = FALSE,
                 toImageButtonOptions = list(
                   format = "png"
                 )
  )

# show plot
plot
```


Thermal Comfort according to SIA 180:2014
Room Temperature / °C
−2 0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30
Upper limit SIA 180 passive cooling
Upper limit SIA 180 active cooling
Lower Limit SIA 180
Spring
Summer
Fall
Moving average outdoor temperature last 48 hours

Chapter 16

Miscellaneous

Finally, some visualizations that did not fit into any of the above chapters and are not (yet) worth a separate chapter. But these are not less interesting…

16.1 Electricity Household

16.1.1 Goal

You want to plot an electricity consumption diagram with:

- upper plot with daily energy consumption in kWh/day
- lower plot with standby-losses in Watts

Additionaly we would like to see the consumption of an average Swiss household.

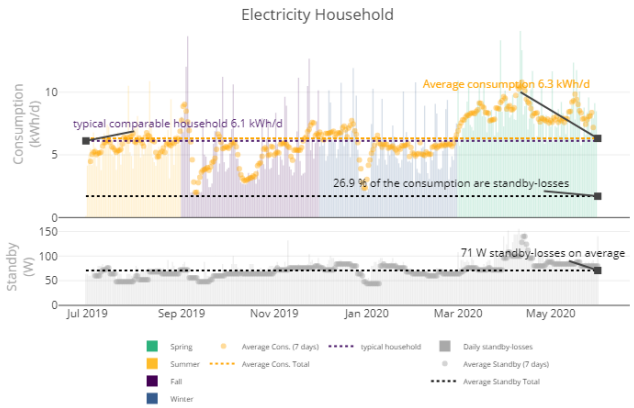


Figure 16.1: Plot Electricity Household

16.1.2 Data Basis

- A csv file with time series of an electric meter in 15 minute interval.

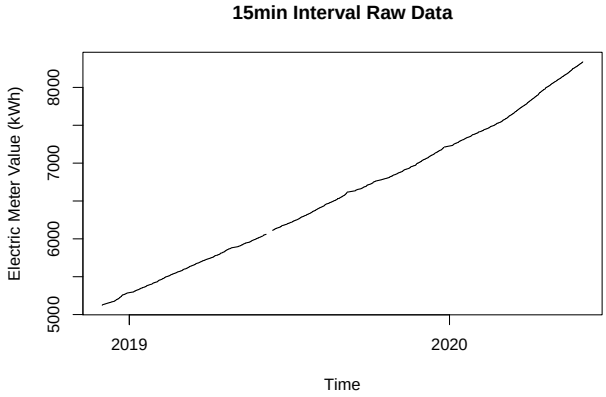


Figure 16.2: Raw Data for Electricity Household Plot

16.1.3 Solution

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

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

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

df$time <- parse_date_time(df$time,
                           order = "YmdHMS",
                           tz = "UTC")

# select room
df <- df %>% select(time, FlatC_Ele)

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

# filter timerange
df <- df %>% filter(timestamp > "2019-07-01")

# Fill missing values with NA
grid.df <- data.frame(timestamp = seq(min(df$timestamp, na.rm = TRUE),
                                      max(df$timestamp, na.rm = TRUE),
                                      by = "15 mins"))
df <- merge(df, grid.df, all = TRUE)

# convert steadily counting energy meter value from kWh to power in kW
```

```
df <- df %>%
  dplyr::mutate(value = (meterValue - lag(meterValue))) %>%
  select(-meterValue)

# remove negative values which occur beause of change summer/winter time
df <- df %>% filter(value >= 0)

# determine date related parameters for later filtering
df$day <- as.Date(df$time, tz = "UTC")
df$week <- lubridate::week(df$time)
df$month <- lubridate::month(df$time)
df$year <- lubridate::year(df$time)

# data cleansing
# tag NA
df <- df %>% dplyr::mutate(deleteNA = ifelse(is.na(value),1,0))

# tag values below 0 and higher than 9.2 kW
df <- df %>% dplyr::mutate(deleteHiLoVal = ifelse(value > 9.2,1, ifelse(value < 0,1,0)))
# Assumption max. fuse 40 ampere (higher fuses for single family houses)
# this results in continuous power 9.2 kW
# this results in an hourly consumption of 9.2kWh
# over 24h = approx. 221 kWh max. consumption per day

# tag whole days which have one or more values to delete, keep only whole valid days
df <- df %>%
  group_by(day) %>%
  dplyr::mutate(delete = sum(deleteNA, na.rm = TRUE) + sum(deleteHiLoVal, na.rm = TRUE))

df <- df %>% ungroup()

# delete full days with invalid data
df <- df %>%
  filter(delete == 0) %>%
  select(-deleteNA, -deleteHiLoVal, -delete)

# determine season for later filtering
df <- df %>% dplyr::mutate(season = redutils::getSeason(timestamp))

# calculate sum and min per day
df <- df %>% dplyr::group_by(day) %>% dplyr::mutate(sum = sum(value))
df <- df %>% dplyr::group_by(day) %>% dplyr::mutate(min = min(value)*1000*4)
df <- df %>% ungroup()

df <- df %>% dplyr::select(day, sum, min, season) %>% unique()

df <- df %>% dplyr::mutate(ravgUsage = zoo::rollmean(x=sum, 7, fill = NA))
df <- df %>% dplyr::mutate(rminStandby = -1 * zoo::rollmaxr(x = -1 * min, 7, fill = NA))

typEleConsVal <- redutils::getTypEleConsHousehold(occupants = 2, rooms = 3.5, bldgType = "multi", laundry =


# Plot
main = "Electricity consumption private household"
minY <- 0
maxYUsage <- max(df %>% select(sum), na.rm=TRUE)
maxYUsage <- max(maxYUsage, typEleConsVal/365)
```

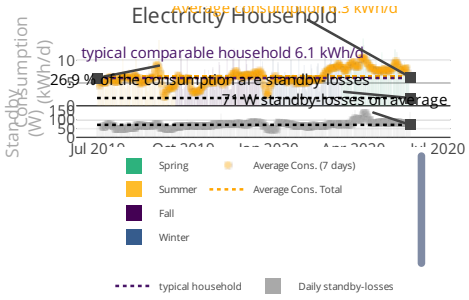
```
maxYStandby <- max(max(df %>% select(min), na.rm=TRUE), 0.25*maxYUsage/24*1000)
minX <- min(df$day)
maxX <- max(df$day)
averageUsage <- mean(df$sum, na.rm=TRUE)
averageStandby <- mean(df$rminStandby, na.rm=TRUE)
shareStandby <- nrow(df %>% select(sum) %>% na.omit()) * averageStandby * 24 / (1000 * sum(df$sum, na.rm=TRUE)) * 100

# legend
l <- list(
  orientation = "h",
  tracegroupgap = "20",
  font = list(size = 8),
  xanchor = "center",
  x = 0.5,
  itemclick = FALSE
)

fig1 <- df %>%
  plot_ly(x = ~day, showlegend = TRUE) %>%
  add_trace(data = df %>% filter(season == "Spring"),
            type = "bar",
            y = ~sum,
            name = "Spring",
            legendgroup = "group1",
            marker = list(color = "#2db27d", opacity = 0.2),
            hoverinfo = "text",
            text = ~ paste("<br />daily usage:              ", sprintf("%.1f kWh/d", sum),
                           "<br />rolling average:        ", sprintf("%.1f kWh/d", ravgUsage),
                           "<br />Average vis. points: ", sprintf("%.1f kWh/d", averageUsage),
                           "<br />Date:                          ", day,
                           "<br />Season:                    ", season
            )
  ) %>%
  add_trace(data = df %>% filter(season == "Summer"),
            type = "bar",
            y = ~sum,
            name = "Summer",
            legendgroup = "group1",
            marker = list(color = "#febc2b", opacity = 0.2),
            hoverinfo = "text",
            text = ~ paste("<br />rolling average:        ", sprintf("%.1f kWh/d", ravgUsage),
                           "<br />Average vis. points: ", sprintf("%.1f kWh/d", averageUsage),
                           "<br />Date:                          ", day,
                           "<br />Season:                    ", season
            )
  ) %>%
  add_trace(data = df %>% filter(season == "Fall"),
            type = "bar",
            y = ~sum,
            name = "Fall",
            legendgroup = "group1",
            marker = list(color = "#440154", opacity = 0.2),
            hoverinfo = "text",
            text = ~ paste("<br />rolling average:        ", sprintf("%.1f kWh/d", ravgUsage),
                           "<br />Average vis. points: ", sprintf("%.1f kWh/d", averageUsage),
                           "<br />Date:                          ", day,
                           "<br />Season:                    ", season
```

```
            )
  ) %>%
  add_trace(data = df %>% filter(season == "Winter"),
            type = "bar",
            y = ~sum,
            name = "Winter",
            legendgroup = "group1",
            marker = list(color = "#365c8d", opacity = 0.2),
            hoverinfo = "text",
            text = ~ paste("<br />rolling average:        ", sprintf("%.1f kWh/d", ravgUsage),
                           "<br />Average vis. points: ", sprintf("%.1f kWh/d", averageUsage),
                           "<br />Date:                        ", day,
                           "<br />Season:                    ", season
            )
  ) %>%
  add_trace(data = df,
            type = "scatter",
            mode = "markers",
            y = ~ravgUsage,
            name = "Average Cons. (7 days)",
            legendgroup = "group2",
            marker = list(color = "orange", opacity = 0.4, symbol = "circle"),
            hoverinfo = "text",
            text = ~ paste("<br />rolling average:        ", sprintf("%.1f kWh/d", ravgUsage),
                           "<br />Average vis. points: ", sprintf("%.1f kWh/d", averageUsage),
                           "<br />Date:                        ", day,
                           "<br />Season:                    ", season
            )
  ) %>%
  add_segments(x = ~minX,
               xend = ~maxX,
               y = ~averageUsage,
               yend = ~averageUsage,
               name = "Average Cons. Total",
               legendgroup = "group2",
               line = list(color = "orange", opacity = 1.0, dash = "dot"),
               hoverinfo = "text",
               text = ~ paste("<br />rolling average:        ", sprintf("%.1f kWh/d", ravgUsage),
                              "<br />Average vis. points: ", sprintf("%.1f kWh/d", averageUsage),
                              "<br />Date:                        ", day,
                              "<br />Season:                    ", season
               )
  ) %>%
  add_segments(x = ~minX,
               xend = ~maxX,
               y = ~averageStandby*24/1000,
               yend = ~averageStandby*24/1000,
               name = "Average Standby Total",
               legendgroup = "group3",
               showlegend = FALSE,
               line = list(color = "black", opacity = 1.0, dash = "dot"),
               hoverinfo = "text",
               text = ~ paste("<br />Average standby power:          ", sprintf("%.0f W", averageStandby),
                              "<br />equals to daily energy:         ", sprintf("%.1f kWh", averageStandby*
                              "<br />Standby percent of total cons.: ", sprintf("%.0f %%", shareStandby)
               )
  ) %>%
```

```
  add_segments(x = ~minX,
               xend = ~maxX,
               y = ~typEleConsVal,
               yend = ~typEleConsVal,
               name = "typical household",
               legendgroup = "group4",
               line = list(color = "#481567FF", opacity = 1.0, dash = "dot"),
               hoverinfo = "text",
               text = ~ paste("<br />typical household:           ", sprintf("%.0f kWh/year", typEleConsVal*365),
                              "<br />equals to daily energy:      ", sprintf("%.1f kWh/day", typEleConsVal),
                              "<br />consumption of current flat: ", sprintf("%.1f kWh/day", averageUsage)
               )
  ) %>%
  add_annotations(
    x = minX,
    y = typEleConsVal,
    text = paste0("typical comparable household ", sprintf("%.1f kWh/d", typEleConsVal)),
    xref = "x",
    yref = "y",
    showarrow = TRUE,
    arrowhead = 7,
    ax = 100,
    ay = -20,
    font = list(color = "#481567FF")
  ) %>%
  add_annotations(
    x = maxX,
    y = averageUsage,
    text = paste0("Average consumption ", sprintf("%.1f kWh/d", averageUsage)),
    xref = "x",
    yref = "y",
    showarrow = TRUE,
    arrowhead = 7,
    ax = -100,
    ay = -60,
    font = list(color = "orange")
  ) %>%
  add_annotations(
    x = maxX,
    y = averageStandby*24/1000,
    text = paste0(sprintf("%.1f %%", shareStandby), " of the consumption are standby-losses"),
    xref = "x",
    yref = "y",
    showarrow = TRUE,
    arrowhead = 7,
    ax = -160,
    ay = -15,
    font = list(color = "black")
  ) %>%
  layout(
    title = main,
    xaxis = list(
      title = ""
    ),
    yaxis = list(title = "Consumption<br>(kWh/d)",
                 range = c(minY, maxYUsage),
                 titlefont = list(size = 14, color = "darkgrey")),
```

```
    hoverlabel = list(align = "left"),
    margin = list(l = 80, t = 50, r = 50, b = 10),
    legend = l
  )

fig2 <- df %>%
  plot_ly(x = ~day, showlegend = TRUE) %>%
  add_trace(data = df,
            type = "bar",
            y = ~min,
            name = "Daily standby-losses",
            legendgroup = "group3",
            marker = list(color = "darkgrey", opacity = 0.2),
            hoverinfo = "text",
            text = ~ paste("<br />daily standby:           ", sprintf("%.0f W", min),
                           "<br />rolling average:        ", sprintf("%.0f W", rminStandby),
                           "<br />Average vis. points: ", sprintf("%.0f W", averageStandby),
                           "<br />Date:                          ", day,
                           "<br />Season:                    ", season
            )
  ) %>%
  add_trace(data = df,
            type = "scatter",
            mode = "markers",
            y = ~rminStandby,
            name = "Average Standby (7 days)",
            legendgroup = "group3",
            marker = list(color = "darkgrey", opacity = 0.5, symbol = "circle"),
            hoverinfo = "text",
            text = ~ paste("<br />daily standby:           ", sprintf("%.0f W", min),
                           "<br />rolling average:        ", sprintf("%.0f W", rminStandby),
                           "<br />Average vis. points: ", sprintf("%.0f W", averageStandby),
                           "<br />Date:                          ", day,
                           "<br />Season:                    ", season
            )
  ) %>%
  add_segments(x = ~minX,
               xend = ~maxX,
               y = ~averageStandby,
               yend = ~averageStandby,
               name = "Average Standby Total",
               legendgroup = "group3",
               line = list(color = "black", opacity = 1.0, dash = "dot"),
               hoverinfo = "text",
               text = ~ paste("<br />Average standby power:          ", sprintf("%.0f W", averageStandby),
                              "<br />equals to daily energy:         ", sprintf("%.1f kWh", averageStandby*
                              "<br />Standby percent of total cons.: ", sprintf("%.0f %%", shareStandby)
               )
  ) %>%
  add_annotations(
    x = maxX,
    y = averageStandby,
    text = paste0(sprintf("%.0f W", averageStandby), " standby-losses on average"),
    xref = "x",
    yref = "y",
    showarrow = TRUE,
    arrowhead = 7,
```

```
    ax = -60,
    ay = -20,
    font = list(color = "black")
  ) %>%
  layout(
    title = "Electricity Household",
    xaxis = list(
      title = ""
    ),
    yaxis = list(title = " Standby<br>(W)",
                 range = c(minY, maxYStandby),
                 titlefont = list(size = 14, color = "darkgrey"),
                 legend = list(orientation = 'h')),
    legend = l
  )

# calculate ratio which is visual representative for comparison
# ratio <- 1/maxYUsage * maxYStandby * 24 / 1000
ratio <- 0.3
fig <- subplot(fig1, fig2, nrows = 2, shareX = TRUE, heights = c(1-ratio, ratio), titleY = TRUE) %>%
  plotly::config(modeBarButtons = list(list("toImage")),
                 displaylogo = FALSE,
                 toImageButtonOptions = list(
                   format = "png"
                 )
  )
fig
```



16.1.4 See also

You probably noticed the line with the average consumption value. This gets calculated by the recommended values and formulas of the study Typischer Haushalt-Stromverbrauch - Schlussbericht by Nipkov, J. (2013)

You can find the implementation in redutils function getTypEleConsHousehold() where various parameters can get adapted via function call arguments.

16.2 Room Temperature Reduction

16.2.1 Goal

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

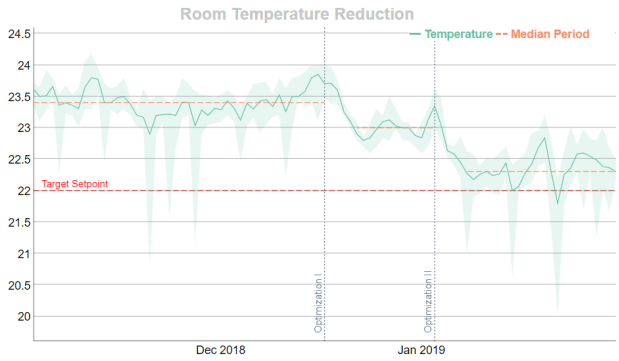


Figure 16.3: Room Temperature Reduction Plot

16.2.2 Data Basis

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

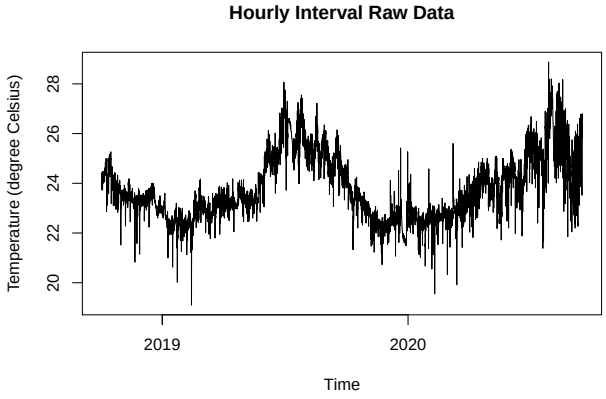


Figure 16.4: Raw Data Temperature for Room Temperature Reduction Plot

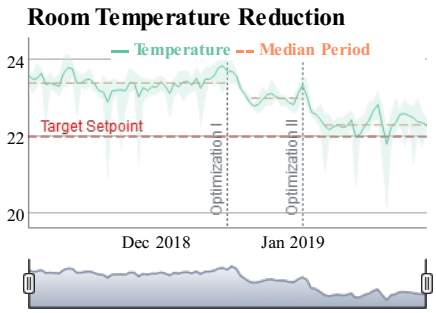
16.2.3 Solution

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

```
library(dplyr)
library(lubridate)
library(dygraphs)
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) %>%
  dplyr::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 %>%
  dplyr::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) %>%
  dplyr::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")
```



16.2.4 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.

16.3 Building Energy Signature

16.3.1 Goal

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

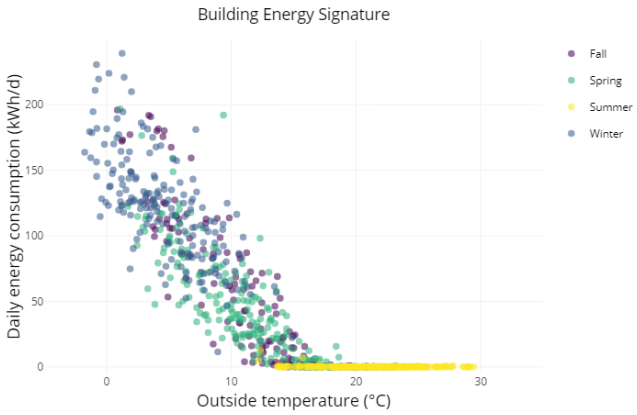


Figure 16.5: Building Energy Signature Plot

16.3.2 Data Basis

Two separate csv files with time series data from the outside temperature and the energy data with unaligned time intervals:

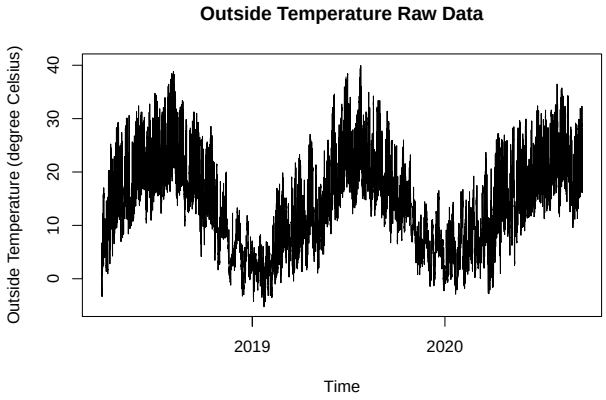


Figure 16.6: Outside Temperature Raw Data for Building Energy Signature Plot

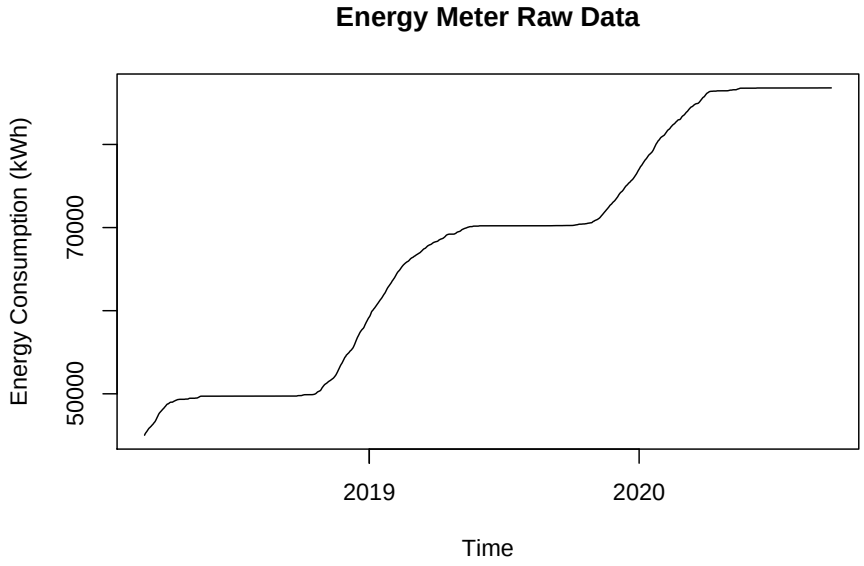


Figure 16.7: Energy Meter Raw Data for Building Energy Signature Plot

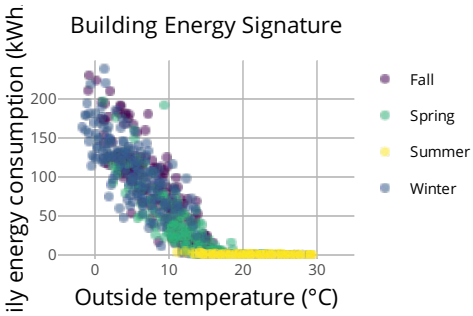
16.3.3 Solution

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) %>%
  dplyr::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,
                                     orders = "YmdHMS",
                                     tz = "Europe/Zurich")

dfHeatEnergy$day <- as.Date(cut(dfHeatEnergy$time, breaks = "day"))
dfHeatEnergy <- dfHeatEnergy %>%
  group_by(day) %>%
  dplyr::mutate(energyMax = max(energyHeatingMeter)) %>%
  ungroup()

dfHeatEnergy <- dfHeatEnergy %>%
  select(day, energyMax) %>%
  unique() %>%
```

```
  na.omit()

dfHeatEnergy <- dfHeatEnergy %>%
  dplyr::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 %>% dplyr::mutate(season = redutils::getSeason(df$day))

# static chart with ggplot
plot <- 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)
        )

# interactive chart
plotly::ggplotly(plot, 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)
```



16.3.4 Discussion

This visualization allows a quick detection of malfunctions and provides valuable information on the energy efficiency of the building.

- A constant indoor temperature is assumed
- It is also assumed that the outside temperature is the parameter with the greatest influence on heating energy consumption
- This method is suitable for buildings with stable internal heat loads and relatively low passive solar heat loads

16.4 Building Energy Signature Proposed

16.4.1 Goal

Create a plot of a building energy signature as proposed in “Development and validation of energy signature method”, Eriksson et al, 2020:

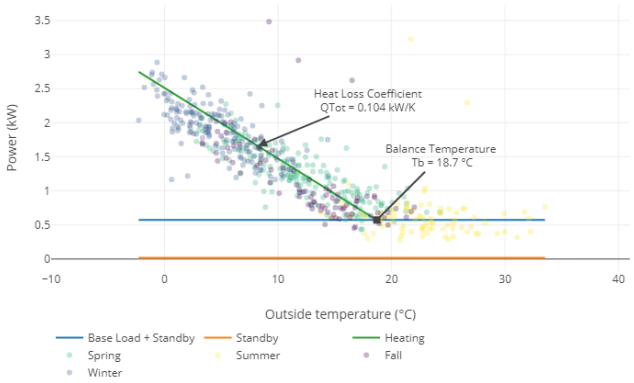
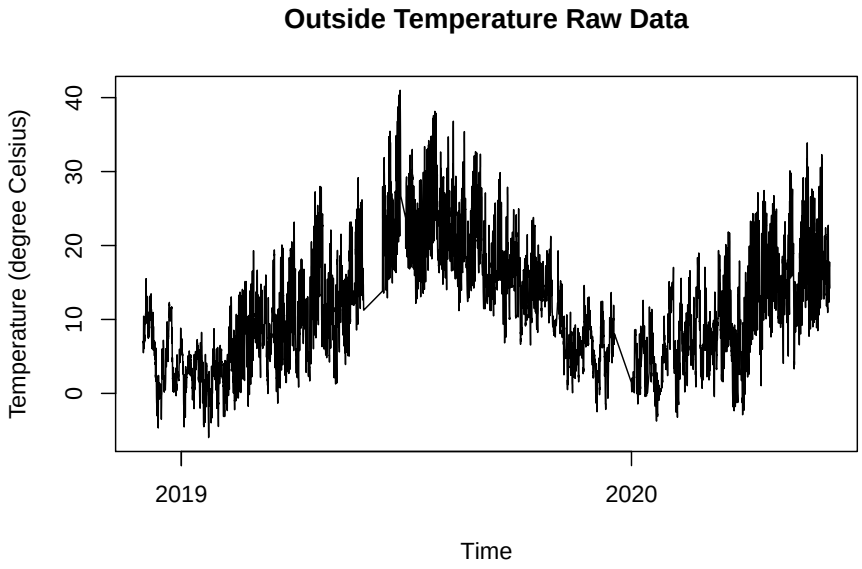
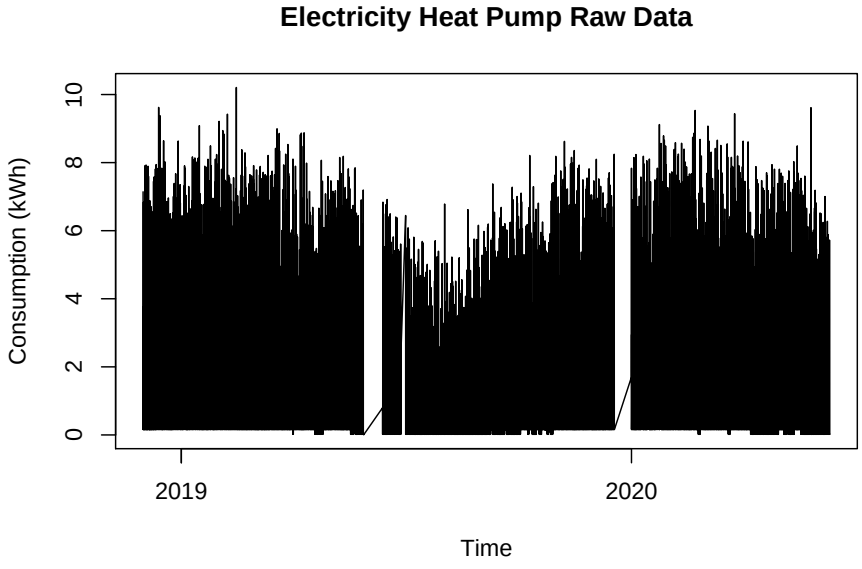
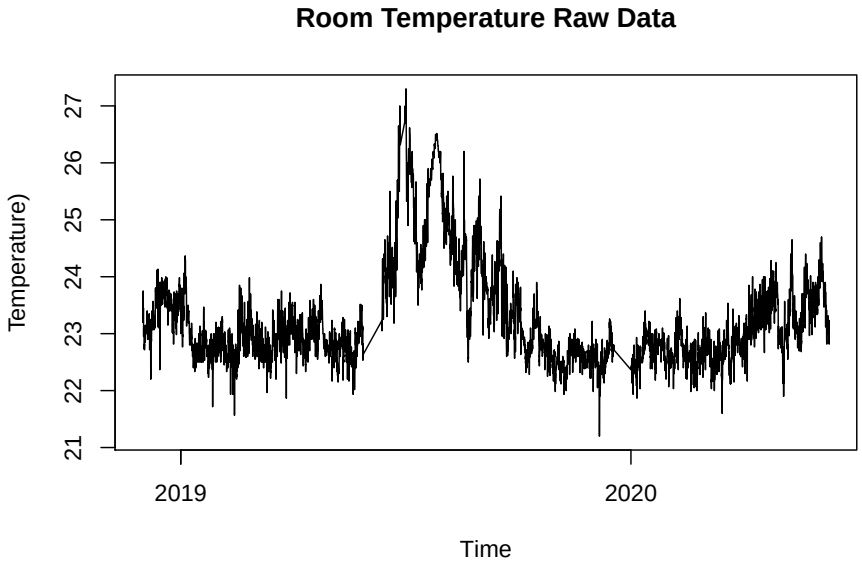


Figure 16.8: Building Energy Signature Plot ”PES”

16.4.2 Data Basis






Room Temperature Raw Data
Temperature)
21
22
23
24
25
26
27
2019
2020
Time

16.4.3 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:

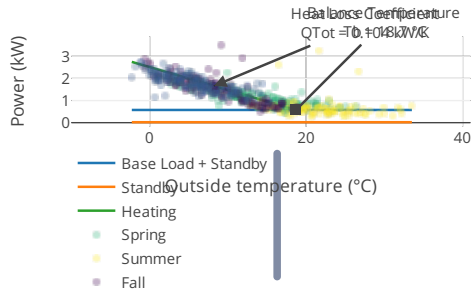
```
library(plotly)
library(plyr)
library(lubridate)
library(redutils)

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

names(df.all)[1] <- "time"
names(df.all)[2] <- "TOa"
names(df.all)[3] <- "Energy"
names(df.all)[4] <- "TRoom"

df.all$time <- as.POSIXct(df.all$time,
                format="%Y-%m-%dT%H:%M:%OSZ", tz="Europe/Zurich")

df.all <- df.all %>%
  dplyr::mutate(season = getSeason(time),
                day = lubridate::date(time),
                month = lubridate::month(time),
                week = lubridate::week(time),
                hour = lubridate::hour(time),
                power = Energy
                ) %>%
  na.omit()

# Initial Balance Temperature Tb
Tb <- 12

# Initial difference
delta <- 1

while(abs(delta) > 0.1){

  # calculate standby power pStby
  # conditions
  # - TOa > Tb
  df <- df.all %>% group_by(week) %>%
    dplyr::mutate(keep = (mean(TOa, na.rm = TRUE) > Tb),
                  pStby = min(power, na.rm = TRUE)) %>%
    filter(keep == TRUE) %>%
    select(-Energy)

  pStby <- mean(df$pStby, na.rm = TRUE)

  # calculate hotwater power pHw
```

```
# conditions
# - TOa > Tb
  df <- df.all %>%
  group_by(week) %>%
  dplyr::mutate(keep = (mean(TOa, na.rm = TRUE) > Tb)) %>%
  dplyr::mutate(pHw = mean(power, na.rm = TRUE)) %>%
  filter(keep == TRUE)

pHw <- mean(df$pHw, na.rm = TRUE) - pStby

# Assumption for internal heat gains
pIhg <- 3*90*9/1000/4.5  # 3 W/m^2, 90m^2/flat, 9 flats, COP WP 4.5, 24 hours per day

# calculate internal heat generation pIhg
# conditions
# - Month is Jan, Feb and March
# - TOa < Tb
df <- df.all %>%
  filter(month %in% c(1,2,3)) %>%
  group_by(day) %>%
  dplyr::mutate(keep = (max(TOa, na.rm = TRUE) < Tb)) %>%
  filter(keep == TRUE) %>%
  dplyr::mutate(meanPower = mean(power, na.rm = TRUE)) %>%
  dplyr::mutate(meanTOa = mean(TOa, na.rm = TRUE))  %>%
  dplyr::mutate(meanTRoom = mean(TRoom, na.rm = TRUE)) %>%
  select(day, meanPower, meanTOa,  meanTRoom, week) %>%
  unique()

# calculate Building total heat loss coefficient QTot in kW/K
df <- df %>%
  dplyr::mutate(QTot = (meanPower - pStby - pHw + pIhg)/(meanTRoom - meanTOa))

QTot <- mean(df$QTot, na.rm = TRUE)

result <- NULL

# Iterative determination of balance temperature Tb
for(i in seq(10,30,0.1)){
  df.i <- df.all %>%
    dplyr::mutate(tempDiff = i - TOa) %>%
    filter(tempDiff > 0)

  df.i <- df.i %>%
    dplyr::mutate(powerCalc = (QTot * tempDiff + pStby + pHw))

  percDiff <- 100/sum(df.all$power, na.rm = TRUE) * sum(df.i$powerCalc, na.rm = TRUE)

  result = rbind(result, data.frame(Tb = i,
                                    percDiff = percDiff,
                                    power = sum(df.all$power),
                                    powerCalc = sum(df.i$powerCalc)))
}

Tb.new <- result[which(abs(result$percDiff-100) == min(abs(result$percDiff - 100), na.rm = TRUE)),1]
delta <- Tb - Tb.new
Tb <- Tb.new
```

```
}

# create plot
df <- df.all %>%
  group_by(day) %>%
  dplyr::mutate(TOaMean = mean(TOa, na.rm = TRUE)) %>%
  dplyr::mutate(powerSum = mean(power, na.rm = TRUE)) %>%
  select(day, TOaMean, powerSum, season) %>%
  na.omit() %>%
  unique() %>%
  ungroup()

names(df)[2] <- "TOa"
names(df)[3] <- "Power"

# df <- df %>% dplyr::mutate((EnergyHeat = TOa * -0.04955 + 4.5865) * Energy)
# names(df)[5] <- "EnergyHeat"

df.fall <- df %>% filter(season=="Fall")
df.winter <- df %>% filter(season=="Winter")
df.spring <- df %>% filter(season=="Spring")
df.summer <- df %>% filter(season=="Summer")

# annotations
a_Tb <- list(
  x = Tb,
  y = pStby + pHw,
  text = paste0("Balance Temperature<br> Tb = ",
                round(Tb, digits = 1),
                " \u00B0C"
                ),
  showarrow = TRUE,
  arrowhead = 7,
  ax = 70,
  ay = -70
)

a_QTot <- list(
  x = (Tb-min(df$TOa, na.rm = TRUE))/2+min(df$TOa, na.rm = TRUE),
  y = (Tb-min(df$TOa, na.rm = TRUE)) * QTot/2+pStby + pHw,
  text = paste0("Heat Loss Coefficient<br>QTot = ",
                round(QTot, digits = 3),
                " kW/K"),
  showarrow = TRUE,
  arrowhead = 2,
  ax = 120,
  ay = -50
)

# Create plot
p <- plot_ly()

p <- p %>% add_lines(x = c(min(df$TOa, Tb, na.rm = TRUE),max(df$TOa, na.rm = TRUE)),
                     y = rep((pStby + pHw), 2),
                     name = "Base Load + Standby",
                     hoverinfo = "text",
                     text = ~ paste("Base Load + Standby:  ", sprintf("%.3f kW", pStby + pHw)
```

```
                    ))
p <- p %>% add_lines(x = c(min(df$TOa, Tb, na.rm = TRUE), max(df$TOa, na.rm = TRUE)),
                    y = rep(pStby, 2),
                    name = "Standby",
                    hoverinfo = "text",
                    text = ~ paste("Standby:  ", sprintf("%.3f kW", pStby)
                    ))
p <- p %>% add_lines(x = c(min(df$TOa, na.rm = TRUE), Tb),
                    y = c((Tb-min(df$TOa, na.rm = TRUE)) * QTot + pStby + pHw, pStby + pHw), name = "Heati

p <- p %>% add_markers(data = df.spring,
                       x = ~TOa,
                       y = ~Power,
                       marker = list(color = "#2db27d", opacity = 0.3),
                       name = "Spring",
                       hoverinfo = "text",
                       text = ~ paste("Outside Temp:  ", sprintf("%.1f \u00B0C", TOa),
                                      "<br />Power:  ", sprintf("%.1f kW", Power),
                                      "<br />Date:     ", day,
                                      "<br />Season: ", df.spring$season
                       )
  )

p <- p %>% add_markers(data = df.summer,
                       x = ~TOa,
                       y = ~Power,
                       marker = list(color = "#fde725", opacity = 0.3),
                       name = "Summer",
                       hoverinfo = "text",
                       text = ~ paste("Outside Temp:  ", sprintf("%.1f \u00B0C", TOa),
                                      "<br />Power:  ", sprintf("%.1f kW", Power),
                                      "<br />Date:     ", day,
                                      "<br />Season: ", df.summer$season
                       )
  )


p <- p %>% add_markers(data = df.fall,
                       x = ~TOa,
                       y = ~Power,
                       marker = list(color = "#440154", opacity = 0.3),
                       name = "Fall",
                       hoverinfo = "text",
                       text = ~ paste("Outside Temp:  ", sprintf("%.1f \u00B0C", TOa),
                                      "<br />Power:  ", sprintf("%.1f kW", Power),
                                      "<br />Date:     ", day,
                                      "<br />Season: ", df.fall$season
                       )
  )

p <- p %>% add_markers(data = df.winter,
                       x = ~TOa,
                       y = ~Power,
                       marker = list(color = "#365c8d", opacity = 0.3),
                       name = "Winter",
                       hoverinfo = "text",
                       text = ~ paste("Outside Temp:  ", sprintf("%.1f \u00B0C", TOa),
```

```
                                    "<br />Power:  ", sprintf("%.1f kW", Power),
                                    "<br />Date:     ", day,
                                    "<br />Season: ", df.winter$season
                        )
  )

p <- p %>%
  layout(
    xaxis = list(title = "Outside temperature (\u00B0C)", range = c(min(-10,min(df.all$TOa)), max(35,max(df.all$TOa))),
    yaxis = list(title = "Power (kW)", range = c(min(df$Power, max(df$Power)))),
    showlegend = TRUE,
    legend = list(orientation = "h",
                  y = -0.2,
                  x = 0),
    annotations = list(a_Tb,a_QTot)
  ) %>%
  plotly::config(modeBarButtons = list(list("toImage")), displaylogo = FALSE)

p
```



16.4.4 Discussion

This is a new method, which calculates the balance Temperture, a heating loss coefficient in kW/K, a basic consumption (blue line) and a standby value (orange line) from 15min consumption data of an electric or heat meter in an iterative procedure using the outside and room temperature.

16.5 Plotly Multiple Y Axis

16.5.1 Goal

You want to create a plot with more than one y axis:

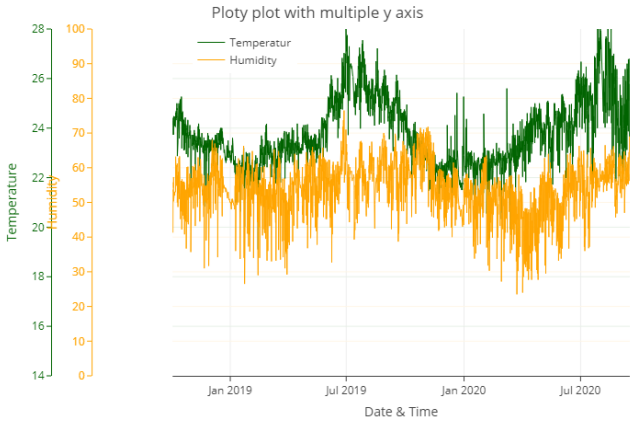


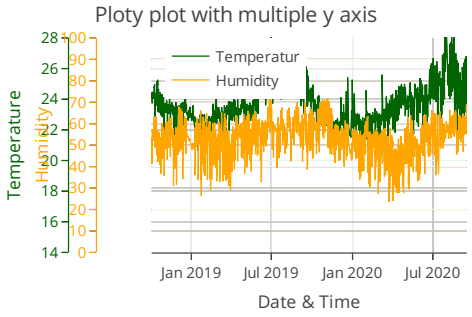
Figure 16.9: Plotly Polot with Multiple Y Axis

16.5.2 Solution

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

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

# 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()
colnames(data) <- c("time", "tempRaw", "humidityRaw")

# create plot
# axis properties
minyaxis1 <- floor(min(14.0,min(data$temperature)))
maxyaxis1 <- ceiling(max(28.0,max(data$temperature)))
minyaxis2 <- 0.0
maxyaxis2 <- 100.0

dateRange <- c(min(data$time), max(data$time))
plotColors <- c("darkgreen", "orange")
pAxisSpacer <- 0.07

plot <- plot_ly(data, x = ~time) %>%
  add_lines(y = ~tempRaw,
            type="scatter",
            mode="lines",
            name='Temperatur',
            line = list(color = plotColors[1], width = 1)
            ) %>%
  add_lines(y = ~humidityRaw,
            type="scatter",
            mode="lines",
            name='Humidity',
            yaxis='y2',
            line = list(color = plotColors[2], width = 1)
            ) %>%
  layout(
    xaxis = list(title = "Date & Time",
                 domain = c(pAxisSpacer*3, 1),
                 type = "date",
                 range = dateRange,
                 ticks='outside',
                 zeroline=TRUE,
                 showline = T),
    yaxis = list(title = 'Temperature',
                 side = "left",
                 color = plotColors[1],
                 range = c(minyaxis1,maxyaxis1),
                 ticks='outside',
```

```
                dtick = 2,
                tick0 = minyaxis1,
                tickmode = "linear",
                position = 0,
                anchor = 'free',
                zeroline = F,
                showline = T),
    yaxis2 = list(title = 'Humidity',
                  side = "left",
                  color = plotColors[2],
                  range = c(minyaxis2,maxyaxis2),
                  ticks='outside',
                  dtick = 10,
                  tick0 = minyaxis2,
                  tickmode = "linear",
                  position = pAxisSpacer,
                  overlaying = "y",
                  anchor = 'free',
                  zeroline=F,
                  showline = T),
    legend = list(x=pAxisSpacer*3.5, y= 1),
    showlegend = T,
    title = list(text = "Ploty plot with multiple y axis")
  )
plot
```



Appendix A

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.

A.1 Installing a Package

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

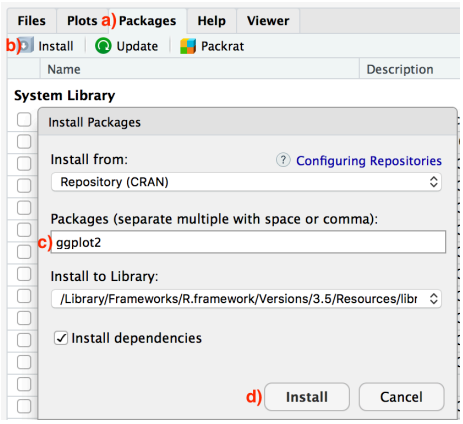


Figure A.1: Install packages via RStudio GUI

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")
```

A.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")
```

A.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)
```