Energy Data Analysis with R

Reto Marek

2020-11-25

Contents

1 Introduction **7**

1.1 Content 7

1.2 Why R and RStudio? 8

1.3 Other useful sources 8

1.4 Acknowledgements 9

2 Getting started **11**

2.1 Install R and R Studio 11

2.2 Install required packages 13

2.3 Create first R Script 13

I R Basics **15**

3 Introduction to R Basics **17**

4 Loading Data **19**

5 Data Wrangling **21**

5.1 Add Metadata for later filtering 21

5.2 Manipulating Data Frames 24

5.3 The pipe `%>%` 25

5.4 Examples 28

6 Explorative Data Analysis **31**

6.1 Get overview 31

6.2 Quickly Exploring Data 33

II Data Visualizations 35

7 Seasonal Plots **37**

7.1 Overlapping 37

7.2 Mini Plots 41

7.3 Polar 44

7.4 Before/After Optimization 47

8 Decomposition **51**

8.1 Long term 51

8.2 Short term 53

9 Heat Maps **57**

9.1 Median-Weeks 57

9.2 Calendar 58

10 Daily Profiles **61**

10.1 Overview 61

10.2 Overlayed 62

10.3 Mean 66

10.4 Decomposed 68

11 Comfort Plots **71**

11.1 Mollier hx Diagram 71

11.2 SIA 180 Thermal Comfort 73

12 Miscellaneous **79**

12.1 Electricity Household 79

12.2 Room Temperature Reduction 86

12.3 Building Energy Signature 89

A Packages in R **93**

A.1 Installing a Package 93

A.2 Loading a Package 94

A.3 Upgrading Packages 94

Chapter 1

Introduction

Preface

This 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.

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

1.1 Content

The book is structured in the following parts:

- R Basics
- Data Visualizations

The part "R Basics" covers general data analysis tasks like data loading, wrangling and aggregation. Basically only simple and quickly created visualizations are presented in this part, which are used in the context of an explorative data analysis. Mostly this part deals with number juggling and table viewing. An "R-beginner" receives information about the installation of the program environment and gets a quick practical introduction.

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

Examples for such graphical representations are covered in the part "Data Visualizations" which brings the calculated numbers to life. The presented code makes the creation of common and useful plots for energy and comfort data fast and easy.

The recipes in this book will show you how to complete certain specific tasks. Examples are shown so that you can understand the basic principle and reproduce the analysis or visualization with your own data. Simply copy the code into your R-script, replace the sample data files with your own and execute the code.

1.2 Why R and RStudio?

In a study commissioned by the Swiss Federal Office of Energy, 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 create 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. The change to an environment like "R" seems to be difficult for many. In the market there are numerous books which make the change to "R" for other disciplines easier. However, experts from the energy and building services engineering industry lack a corresponding work. The present book is intended to close this gap.

The freely available programming language "R" and its graphical user interface "RStudio" offer many more possibilities for data analysis and data visualization.

1.3 Other useful sources

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 useful books available, including:

- R Graphics Cookbook
- Introduction to Data Science - Data Analysis and Prediction Algorithms with R
- Hands-On Programming with R
- Engineering Data Analysis in R
- Forecasting: Principles and Practice
- AFIT Data Science Lab R Programming Guide

1.4 Acknowledgements

The authors would like to express their sincere thanks 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

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 hslu-ige-laes/redutils package on github, which is frequently used throughout this book. Chapter 2.2 shows you how to install it.

Finally beginners find in 2.3 an easy code example which creates a simple time series plot.

2.1 Install R and R Studio

Before we can start the first analysis, we have to install “R” and “RStudio”.

- “R” is a programming language used for statistical computing while “RStudio” provides a graphical user interface
- “R” may be used without “RStudio”, but “RStudio” may not be used without “R”
- Both, “R” and “RStudio” are free of charge and there are no licencse fees
- When you later make an analysis and visualizations, you only work in the graphical user interface “RStudio”

2.1.1 Download and Install R

2.1.1.1 Windows

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

The wizard will install "R" into your `Program Files` folders and 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.

2.1.1.2 Mac OSX

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

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

2.1.1.3 Linux

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

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

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

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

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. Below are the packages used in this book and it is recommended to install them.

- 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 first R Script

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

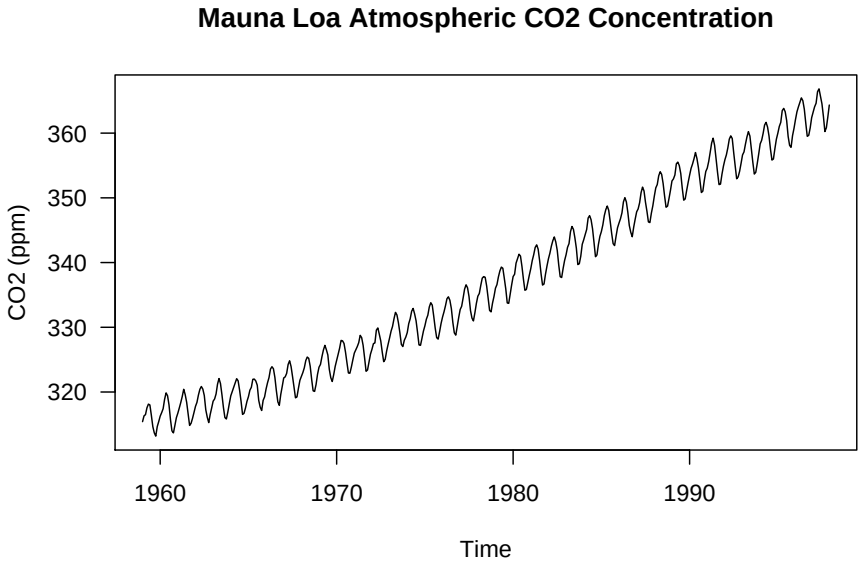
Let's create the first script with a visualization:

- 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

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

Two sources 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"

The part "R-Basics" in this book is additive to the basics referenced above and focuses on practical examples of how data can be loaded, transformed and analyzed.

Chapter 4

Loading Data

This chapter introduces two functions that can be used to load data from csv-files 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” csv-Files - Engineering Data Analysis in R - Chapter 3.6 “Piping” csv-Files

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

```
library(dplyr)
library(lubridate)

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

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 <- mutate(df, temp_f = temp_c * 1.8 + 32)

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

5.3 The pipe %>%

With 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 a href=“https://rafalab.github.io/dsbook/tidyverse.html” target=”_blank”>Introduction to Data Science - Chapter 4.5

5.3.1 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
```

5.3.2 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
```

5.3.3 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
```

5.4 Examples

5.4.1 Import csv file from GWF Relay W60 M-Bus Logger

Some data loggers have a cryptic data format which requires some data wrangling ahead before we can use the time series efficiently. Following an example of a M-Bus data loggger.

```
library(tidyr)

# load csv file
df <- read.csv("https://github.com/hslu-ige-laes/edar/raw/master/sampleData/flatHeatAndHotWater_GWF_RelayW6
               stringsAsFactors=FALSE,
               sep =",")
head(df,23)
```

```
##       Datum  Zeit Adr ID.Nr. HST Nr.                  Wert Einheit
## 1  01.01.10 00:00   1      1 GWF   1                   729     kWh
## 2  01.01.10 00:00   1      1 GWF   2                 80.92     m^3
## 3  01.01.10 00:00   1      1 GWF   3                 80.75     m^3
## 4  01.01.10 00:00   1      1 GWF   4                    26      °C
## 5  01.01.10 00:00   1      1 GWF   5                    25      °C
## 6  01.01.10 00:00   1      1 GWF   6                   1.3       K
## 7  01.01.10 00:00   1      1 GWF   7                   375   hours
## 8  01.01.10 00:00   1      1 GWF   8                   375   hours
## 9  01.01.10 00:00   1      1 GWF   9                     0     l/h
## 10 01.01.10 00:00   1      1 GWF  10                     0      kW
## 11 01.01.10 00:00   1      1 GWF  11 31.12.09 23:58 Win  V
## 12 01.01.10 00:00   1      1 GWF  12                    87     HCA
## 13 01.01.10 00:00   1      1 GWF  13                     0     HCA
## 14 01.01.10 00:00   1      1 GWF  14                     0     kWh
## 15 01.01.10 00:00   1      1 GWF  15                     0     m^3
## 16 01.01.10 00:00   1      1 GWF  16                     0     m^3
## 17 01.01.10 00:00   1      1 GWF  17                     0     HCA
## 18 01.01.10 00:00   1      1 GWF  18                     0     HCA
## 19 01.01.10 00:00   1      1 GWF  19              00.00.00
## 20 01.01.10 00:00   1      1 GWF  20                   $00
## 21 01.01.10 00:00   2      4 GWF   1                   751     kWh
## 22 01.01.10 00:00   2      4 GWF   2                 71.97     m^3
## 23 01.01.10 00:00   2      4 GWF   3                  71.9     m^3
##              Beschreibung      Art Modul SP.Nr. Tarif
## 1                  energy instant.     0      0     0
## 2                  volume instant.     0      0     0
## 3                  volume instant.     1      0     0
## 4       flow temperature instant.     0      0     0
## 5     return temperature instant.     0      0     0
## 6  temperature difference instant.     0      0     0
## 7                 on time instant.     0      0     0
## 8          operating time instant.     0      0     0
## 9             volume flow instant.     0      0     0
## 10                 power instant.     0      0     0
## 11            time point instant.     0      0     0
## 12              HCA-Unit instant.     1      0     0
## 13              HCA-Unit instant.     2      0     0
```

```
## 14                energy instant.     0      1     0
## 15                volume instant.     0      1     0
## 16                volume instant.     1      1     0
## 17              HCA-Unit instant.     1      1     0
## 18              HCA-Unit instant.     2      1     0
## 19            time point instant.     0      1     0
## 20           M-Bus state             NA     NA    NA
## 21                energy instant.     0      0     0
## 22                volume instant.     0      0     0
## 23                volume instant.     1      0     0
```

Please note * The Date and Time are in separate columns * The HCA-Unit value has a factor of 10, so the above HCA-value of 87 are in 10 liters Units and result in 87 * 10 = 870 liter * In the example “Adr” is the identification of the flat

We are only interested in Nr. 1 heating energy of the flat and Nr. 12 the impulse count of the additional meter which is the hot water of the flat.

Following the required steps to parse the data and bring them in a format we can later work with:

```
# create timestamp out of column "Date" and "Time"
df <- df %>% mutate(timestamp = paste0(df$Datum, " ", df$Zeit))
df$timestamp <- parse_date_time(df$timestamp,
                                order = "d.m.y H:M",
                                tz = "Europe/Zurich")


# select columns and rearrange
df <- df %>% select(timestamp, Adr, Nr., Wert, Einheit)

# filter out Adr. of interest
df <- df %>% filter(Nr. %in% c(1,12))

# rename columns
df <- df %>% mutate(Nr. = ifelse(Nr. == 1, paste0("Adr",sprintf("%02.0f", Adr), "_energyHeat"), ifelse(Nr. == 12, paste0

# convert value to numeric
df$Wert <- as.numeric(df$Wert)

# multiply HCA-values by factor 10 to get liters and divide by 1000 to get m3
df <- df %>% mutate(Wert = ifelse((Einheit == "HCA"), Wert * 10/1000, Wert))

df <- df %>% select(-Einheit, -Adr)

# convert long table into wide table
df.wide <- as.data.frame(pivot_wider(df,
                                     names_from = "Nr.",
                                     values_from = Wert,
                                     names_sep = "_")
)
```

This is the result:

```
head(df.wide)
```

```
##    timestamp Adr01_energyHeat Adr01_hotWater Adr02_energyHeat Adr02_hotWater
## 1 2010-01-01              729           0.87              751           1.16
## 2 2010-02-01             1850           2.18             2276           7.55
## 3 2010-03-01             2806           5.84             2826          12.94
## 4 2010-04-01             3615           9.92             3354          18.51
## 5 2010-05-01             4150          13.38             3613          24.81
## 6 2010-06-01             4669          17.50             3640          29.72
##   Adr04_energyHeat Adr04_hotWater Adr03_energyHeat Adr03_hotWater
## 1              972           1.56              799           0.07
## 2             2526           8.14             3103           1.32
## 3             3690          14.83             4786           3.73
## 4             4700          20.82             6151           6.05
## 5             5341          27.06             6900           7.74
## 6             5802          31.61             7702           9.49
```

Finally saving the csv file:

```
write.csv2(df.wide,
           file = "flatHeatAndHotWater.csv",
           row.names = FALSE)
```

Chapter 6

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: to learn more about data by visualizing it in different ways. John W. Tukey, considered the founder of exploratory data analysis, once wrote:

> "Exploratory data analysis is graphical detective work." - John W. Tukey

6.1 Get overview

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

6.1.1 Load data

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

6.1.2 Names

show the column headers of a data frame

```
names(df)
```

```
## [1] "time"              "centralOutsideTemp"
```

6.1.3 Structure

show the structure of the data frame

```
str(df)
```

```
## 'data.frame':    21865 obs. of  2 variables:
##  $ time             : chr  "2018-03-21 11:00:00" "2018-03-21 12:00:00" "2018-03-21 13:00:00" "2018-03-2
##  $ centralOutsideTemp: chr  "5.2" "6.7" "6.4" "6" ...
```

6.1.4 Head/Tail

Show the first and last values

```
head(df)
```

```
##                  time centralOutsideTemp
## 1 2018-03-21 11:00:00                5.2
## 2 2018-03-21 12:00:00                6.7
## 3 2018-03-21 13:00:00                6.4
## 4 2018-03-21 14:00:00                  6
## 5 2018-03-21 15:00:00               5.25
## 6 2018-03-21 16:00:00                4.7
```

```
tail(df)
```

```
##                      time centralOutsideTemp
## 21860 2020-09-17 06:00:00               16.6
## 21861 2020-09-17 07:00:00               19.3
## 21862 2020-09-17 08:00:00              21.85
## 21863 2020-09-17 09:00:00               24.6
## 21864 2020-09-17 10:00:00              26.65
## 21865 2020-09-17 11:00:00               28.1
```

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

6.1.5 Five number summary

reveals details like object classes

```
summary(df)
```

```
##      time           centralOutsideTemp
##  Length:21865       Length:21865
##  Class :character   Class :character
##  Mode  :character   Mode  :character
```

6.2 Quickly Exploring Data

There is a great resource where you can find everything you need to know about creating basic plots. Therefore this topic is not covered again in this book. Please refer to the R Graphics Cookbook from Winston Chang - Chapter 2 “Quickly Exploring Data”.

Part II

Data Visualizations

Chapter 7

Seasonal Plots

tbd

7.1 Overlapping

7.1.1 Goal

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

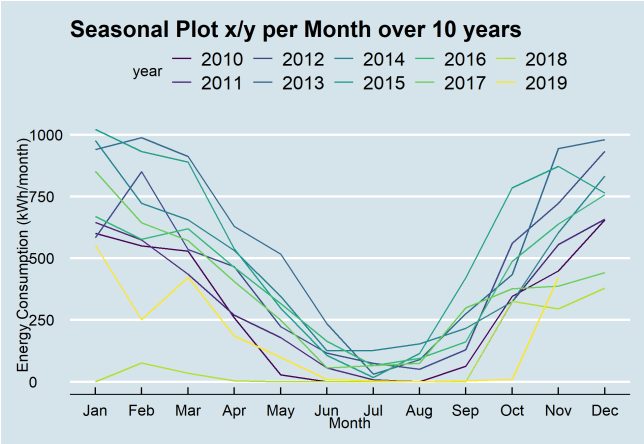


Figure 7.1: Seasonal Plot Overlapping per Month over 10 Years

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.

7.1.2 Data Basis

In general, the values of energy meters, as in our example, increase steadily:

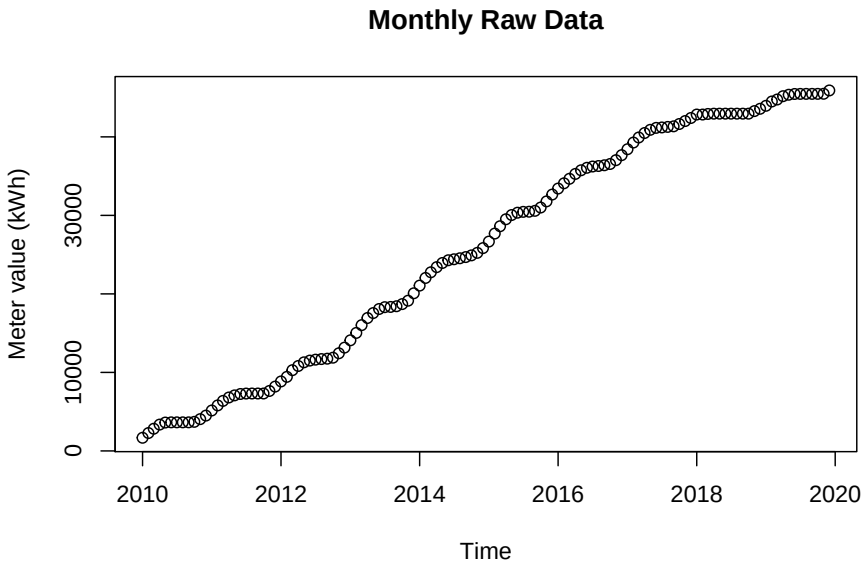


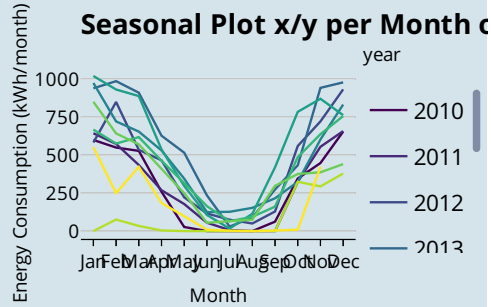
Figure 7.2: Raw Data for Seasonal Plot Overlapping

7.1.3 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 %>% mutate(value = lead(meterValue) - meterValue)

# remove counter value column
df <- df %>% select(-meterValue)

# value correction (outlier because of commissioning)
df[1,2] <- 600

# 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 = "Energy Consumption (kWh/month)"
                     )

# 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")
```

7.1.4 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:

- in the interactive version you can double-click on year in the legend, then only this year is visible
- click once to activate/deactivate a year

7.2 Mini Plots

7.2.1 Goal

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

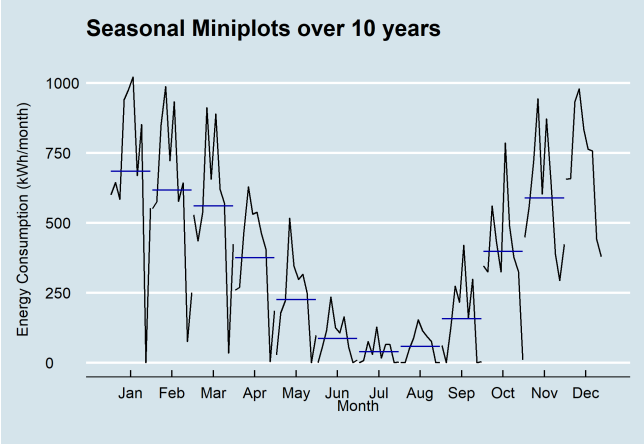


Figure 7.3: Seasonal Plot with mini Time Plots over 10 Years

Here the seasonal patterns for each season are collected together in seprate mini time plots. Be aware that seasons in this context don't correlate with the seasons of the year.

7.2.2 Data Basis

In general, the values of energy meters, as in our example, increase steadily:

7.2.3 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",
```

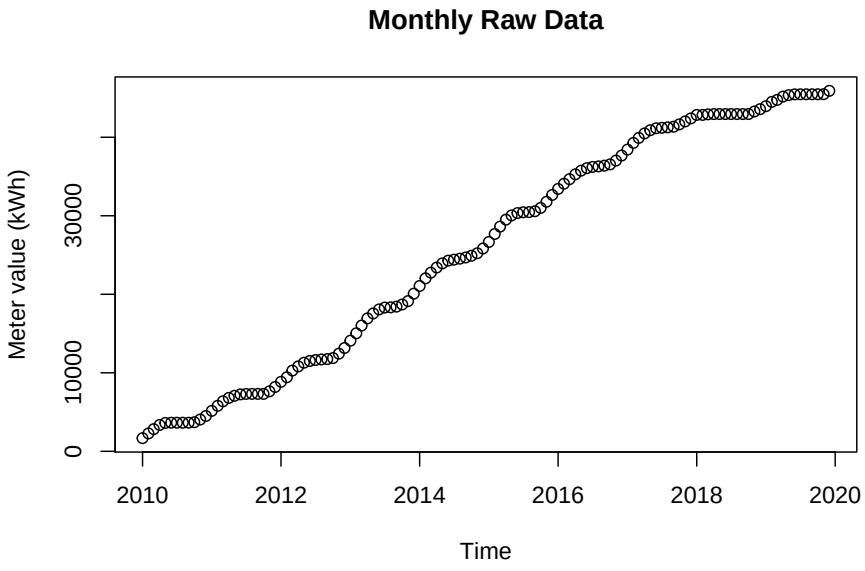
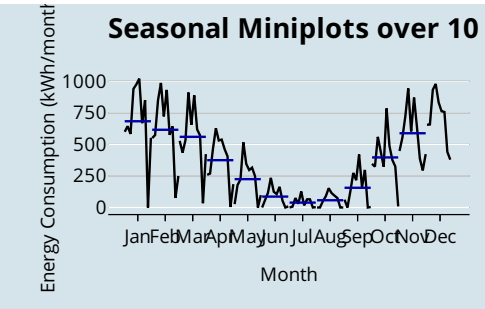


Figure 7.4: Raw Data for Seasonal Miniplots

```
                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 %>% mutate(value = lead(meterValue) - meterValue)

# remove counter value column
df <- df %>% select(-meterValue)

# value correction (outlier because of commissioning)
df[1,2] <- 600

# 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 = "Energy Consumption (kWh/month)\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")
```

7.2.4 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 seasonla pattern with the blue lines

7.3 Polar

7.3.1 Goal

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

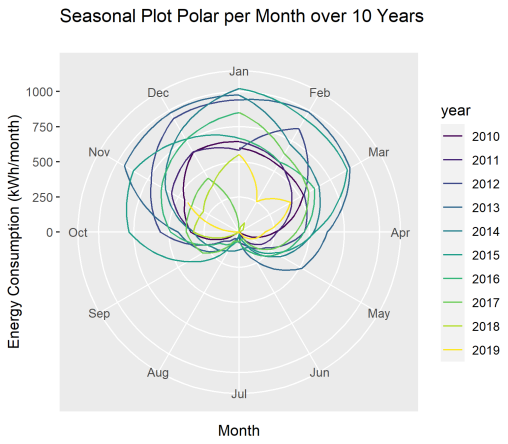


Figure 7.5: Seasonal Plot Polar per Month over 10 Years

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.

7.3.2 Data Basis

In general, the values of energy meters, as in our example, increase steadily:

7.3.3 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)
```

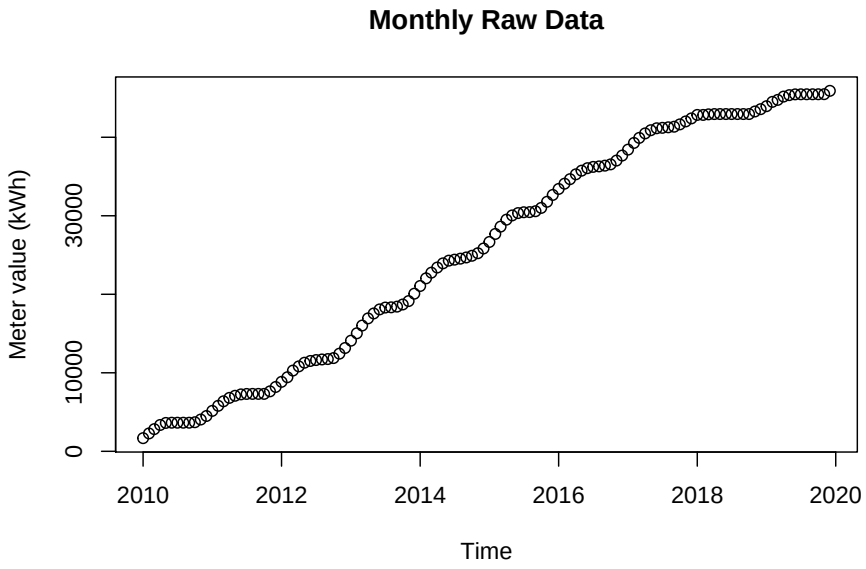
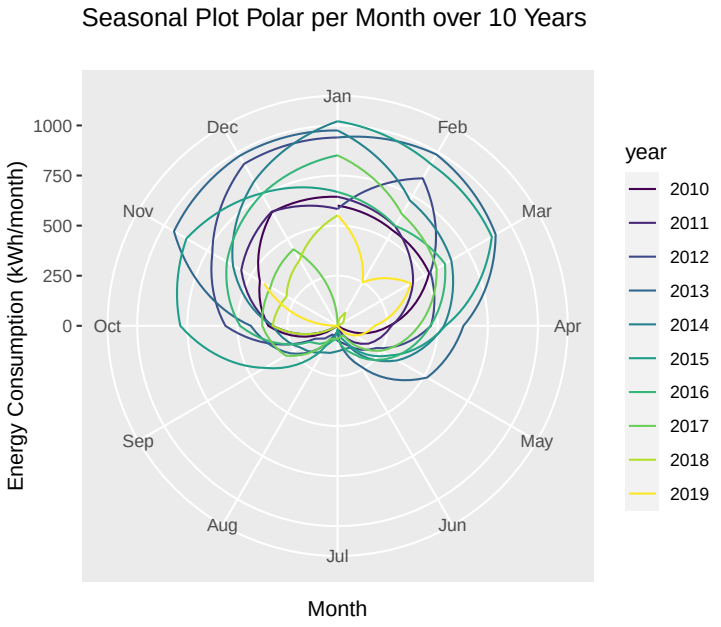


Figure 7.6: Raw Data for Seasonal Plot Polar

```
# 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 %>% mutate(value = lead(meterValue) - meterValue)

# remove counter value column
df <- df %>% select(-meterValue)

# value correction (outlier because of commissioning)
df[1,2] <- 600

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 = "Energy Consumption (kWh/month)",
                    polar = TRUE
                    )

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




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

7.3.4 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 the next chapter 7.4.

7.4 Before/After Optimization

7.4.1 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.

In the following we will create the following plot:

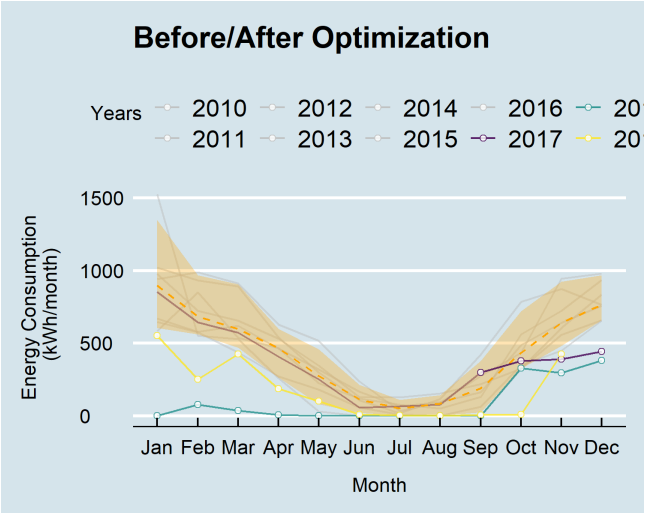


Figure 7.7: Seasonal Plot Overlapping Before/After

7.4.2 Data Basis

In general, the values of energy meters, as in our example, increase steadily:

7.4.3 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
```

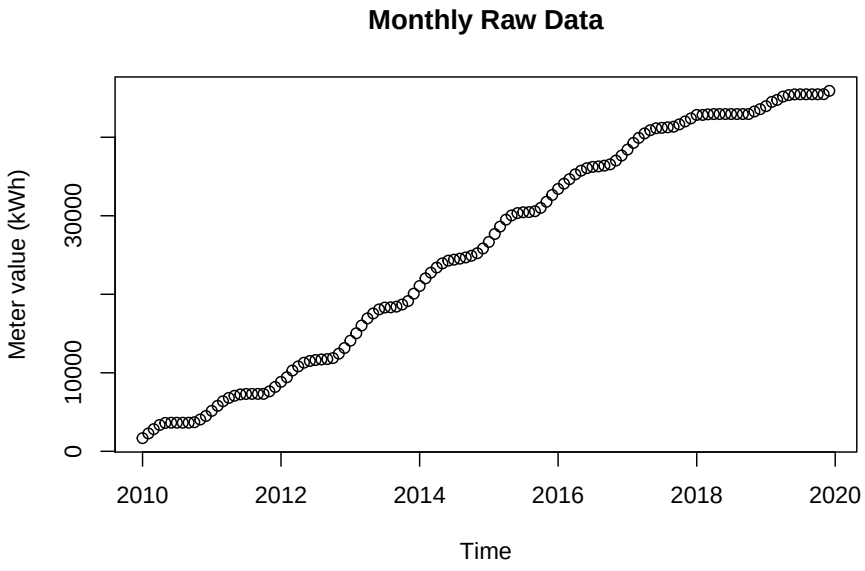
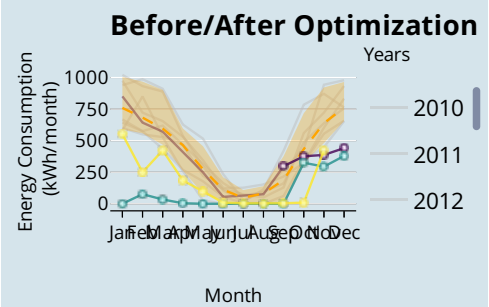


Figure 7.8: Raw Data for Seasonal Plot Overlapping Before/After Optimization

```
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 %>% mutate(value = lead(meterValue) - meterValue)

# remove counter value column
df <- df %>% select(-meterValue)

# value correction (outlier because of commissioning)
df[1,2] <- 600

# create plot
plot <- plotSeasonalXYBeforeAfter(df,
                                  dateOptimization = "2017-09-01",
                                  locTimeZone = "Europe/Zurich",
                                  main = "Before/After Optimization",
                                  ylab = "Energy Consumption \n(kWh/month)"
                                  )
```

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

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

# show plot
plotly
```




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

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

7.4.4 Discussion

- One can clearly see the impact of the optimization
- 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 8

Decomposition

tbd

8.1 Long term

8.1.1 Goal

tbd https://otexts.com/fpp2/tspatterns.html Hyndman, R.J., & Athanasopoulos, G. (2018) Forecasting: principles and practice, 2nd edition, OTexts: Melbourne, Australia. OTexts.com/fpp2. Accessed on

8.1.2 Basis

- tbd
- monthly energy consumption meter values over 10 years

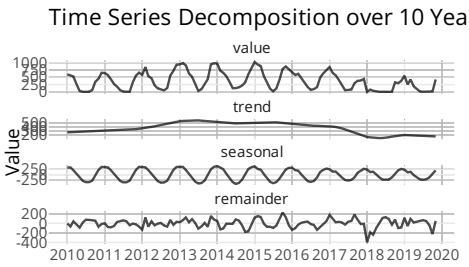
8.1.3 Solution

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)
```

```r
# 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 %>% 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)

p <- 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(p)
```



8.1.4 Discussion

tbd

8.1.5 See Also

tbd

8.2 Short term

8.2.1 Task

tbd https://otexts.com/fpp2/tspatterns.html Hyndman, R.J., & Athanasopoulos, G. (2018) Forecasting: principles and practice, 2nd edition, OTexts: Melbourne, Australia. OTexts.com/fpp2. Accessed on

8.2.2 Basis

- tbd
- 15min energy consumption meter values over five days

8.2.3 Solution

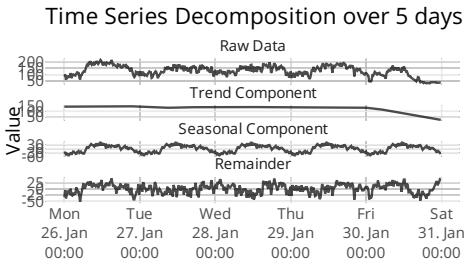
Create a new script, copy/paste the following code and run it:

```
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 %>%
  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 %>%
  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)

p <- 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(p)
```



8.2.4 Discussion

- Trend The trend of a time series refers to the general direction in which the time series is moving. Time series can have a positive or a negative trend, but can also have no trend.
- Seasonal Pattern The seasonal component for time series data refers to its tendency to rise and fall at consistent frequencies.
- Remainder The remainder is what's left of the time series data after removing its trend, cycle, and seasonal components. It is the random fluctuation in the time series data that the above components cannot explain.

8.2.5 See Also

tbd

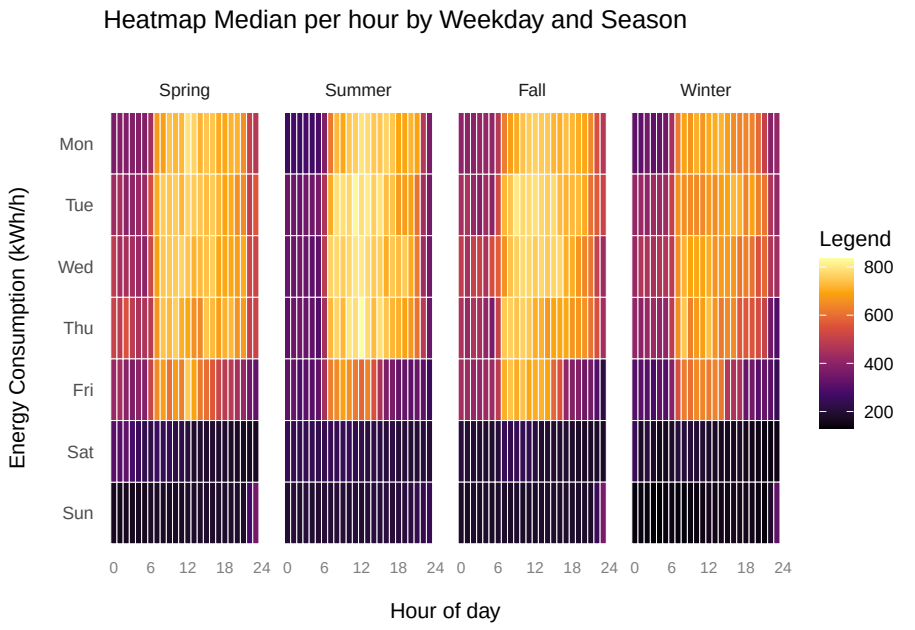
Chapter 9

Heat Maps

tbd

9.1 Median-Weeks

```
library(redutils)
library(plotly)

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

# show the static plot
p
```

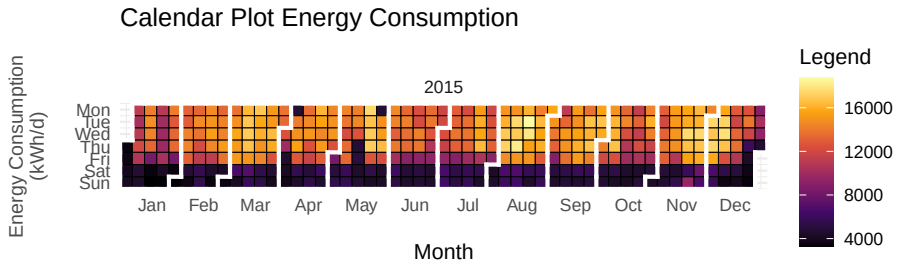



```
# create the interactive plot (optional, uncomment line)
#ggplotly(p)
```

9.2 Calendar

```
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()

p <- 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")

p
```



9.2.1 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

Chapter 10

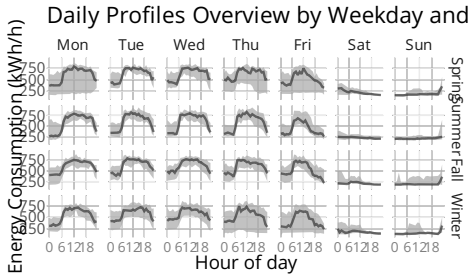
Daily Profiles

tbd

10.1 Overview

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

# 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("timestamp", "meterValue")

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

# 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 %>%
  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)

p <- plotDailyProfilesOverview(df,
                               locTimeZone = "Europe/Zurich",
                               main = "Daily Profiles Overview by Weekday and Season",
                               ylab = "Energy Consumption (kWh/h)",
                               col = "black",
                               confidence = 95.0)

ggplotly(p)
```



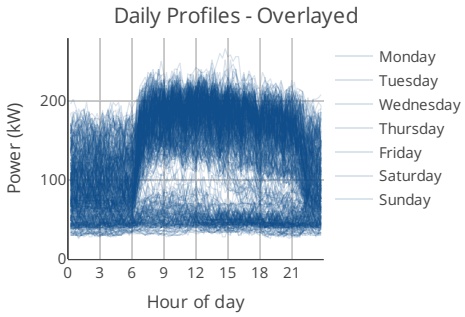
10.2 Overlayed

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

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

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

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

# 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 %>%
  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)

# 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 %>% 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

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",
```

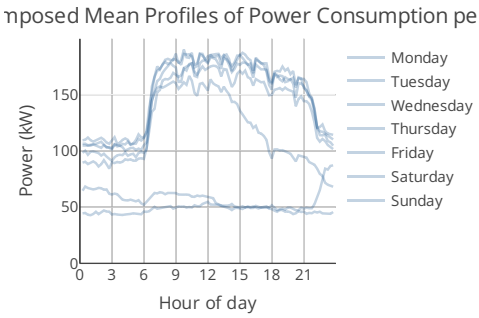
```
        text = ~day,
        hovertemplate = paste("Time: ", format(df$timestamp, "%H:%M"),
                              "<br>Date: ", format(df$timestamp, "%Y-%m-%d"),
                              "<br>Value: %{y:.0f}")) %>%
# workaround with add_trace to have fixed y axis when selecting a dedicated 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)
```



Next we want to create an overview with the mean values for each 15 minute slot per day.

Append the following code at the end of your script:

```
# Calculate Mean value for all 15 minutes for each weekday
df2 <- df %>% group_by(weekday, x) %>% mutate(dayTimeMean = mean(value)) %>% ungroup()

# shrink data frame
df2 <- df2 %>%
  select(x, weekday, timestamp, dayTimeMean) %>%
  unique() %>%
  na.omit() %>%
  arrange(weekday, x)

# plot graph with mean values
maxValMean <- max(df2$dayTimeMean, na.rm = TRUE)*1.05

df2 %>%
  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(df2$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 = "Superimposed Mean Profiles of Power Consumption per 15 min",
         showlegend = TRUE,
         xaxis = list(
           title = "Hour of day",
           tickvals = list(0, 3, 6, 9, 12, 15, 18, 21)
         ),
         yaxis = list(
             title = "Power (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)
```



10.3 Mean

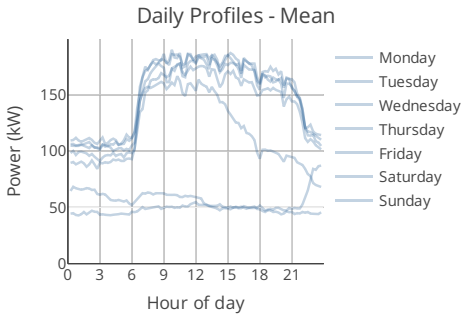
```
# change language to English, otherwise weekdays are in local language
Sys.setlocale("LC_TIME", "English")
```

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

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

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

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

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

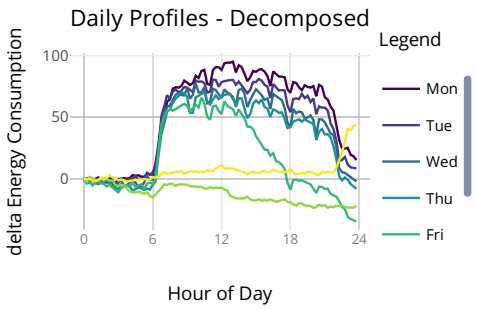
```
# uncomment to filter time range if necessary
#df <- df %>% filter(timestamp > "2015-03-01 00:00:00", timestamp < "2015-04-01 00:00:00")

# 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 %>%
  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)

# 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 %>% 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) %>% 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

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)
```



10.4 Decomposed

```
library(plotly)
library(redutils)

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

p <- plotDailyProfilesDecomposed(data, locTimeZone = "Europe/Zurich")

ggplotly(p)
```


Daily Profiles - Decomposed
delta Energy Consumption
100
50
0
0
6
12
18
24
Hour of Day
Legend
Mon
Tue
Wed
Thu
Fri

Chapter 11

Comfort Plots

tbd

11.1 Mollier hx Diagram

11.1.1 Task

You want to plot a mollier h-x diagram with

- scatter plot of temperature- and humidity sensor data (mean values per day)
- points colored according to season
- comfort zone

11.1.2 Basis

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

11.1.3 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) %>%
  mutate(tempMean = mean(as.numeric(FlatA_Temp)),
         humMean = mean(as.numeric(FlatA_Hum))
         ) %>%
  ungroup()

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

# plot mollier hx diagram
plotMollierHx(data)
```

11.2 SIA 180 Thermal Comfort

11.2.1 Task

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

- scatter plot of indoor- and outdoor temperature sensor data (indoor mean of day, outdoor mean of last 48 hours)
- points colored according to season
- different comfort lines

11.2.2 Basis

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

11.2.3 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, aggregated per day and merged.

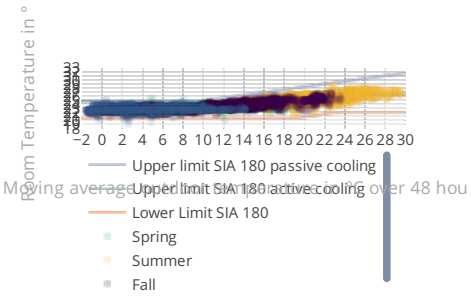
Create a new script, copy/paste the following code and run it:

```
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) %>%
  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)

dfTempOa <- dfTempOa %>%
  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) %>%
  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))

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(
  xaxis = list(title = "Moving average outdoor temperature in \u00B0C over 48 hours",
               range = c(minx, maxx),
               zeroline = FALSE,
               tick0 = minx,
               dtick = 2,
               titlefont = list(size = 14, color = "darkgrey")),
  yaxis = list(title = "Room Temperature in \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 = "svg"
               )
)
```



11.2.4 Discussion

tbd

11.2.5 See Also

tbd

Chapter 12

Miscellaneous

tbd

12.1 Electricity Household

12.1.1 Task

You want to plot an electricity consumption diagram which shows

- 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.

12.1.2 Basis

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

12.1.3 Solution

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

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

```r
# 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 %>%
  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 %>% mutate(deleteNA = ifelse(is.na(value),1,0))

# tag values below 0 and higher than 9.2 kW
df <- df %>% 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) %>%
  mutate(delete = sum(deleteNA, na.rm = TRUE) + sum(deleteHiLoVal, na.rm = TRUE))

df <- df %>% ungroup()
```

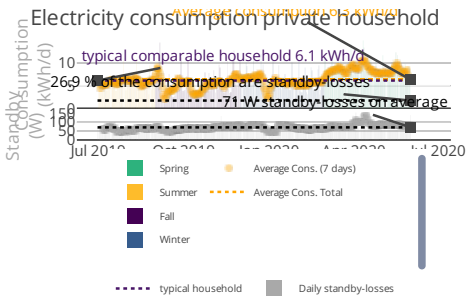
```r
# delete full days with invalid data
df <- df %>%
  filter(delete == 0) %>%
  select(-deleteNA, -deleteHiLoVal, -delete)

# determine season for later filtering
df <- df %>% 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 = "hotWaterSup


# 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*24/1000),
                             "<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*24/1000),
                              "<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(
    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 = "svg"
                 )
  )
fig
```



12.1.4 Discussion

tbd

12.1.5 See Also

tbd

12.2 Room Temperature Reduction

12.2.1 Task

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

You want to create a time series plot with

- the daily median, min and max value

- the overall median of each period
- the desired setpoint

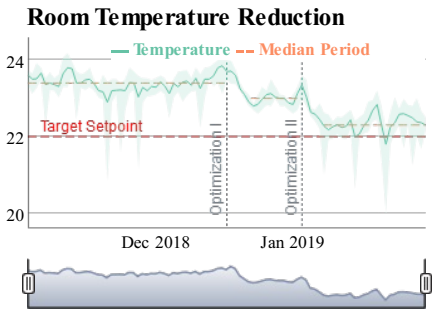
12.2.2 Basis

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

12.2.3 Solution

```
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) %>%
  mutate(minDay = min(as.numeric(FlatA_Temp)),
```

```
        medianDay = median(as.numeric(FlatA_Temp)),
        maxDay = max(as.numeric(FlatA_Temp))
        ) %>%
  ungroup()

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

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

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

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

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



12.2.4 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.

12.3 Building Energy Signature

12.3.1 Task

You want to create a scatter plot with

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

12.3.2 Basis

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

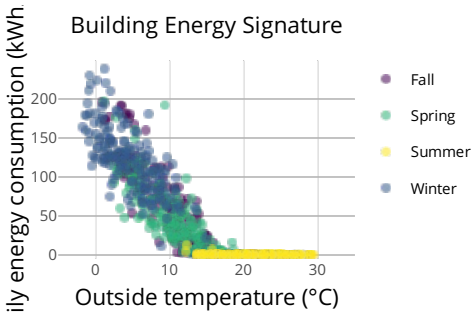
12.3.3 Solution

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.c
                          stringsAsFactors=FALSE,
                          sep =";")
dfOutsideTemp$time <- parse_date_time(dfOutsideTemp$time,
                                      order = "YmdHMS",
                                      tz = "Europe/Zurich")
dfOutsideTemp$day <- as.Date(cut(dfOutsideTemp$time, breaks = "day"))
dfOutsideTemp <- dfOutsideTemp %>%
  group_by(day) %>%
  mutate(tempMean = mean(centralOutsideTemp)) %>%
  ungroup()
dfOutsideTemp <- dfOutsideTemp %>%
  select(day, tempMean) %>%
  unique() %>%
  na.omit()
dfHeatEnergy <- read.csv("https://github.com/hslu-ige-laes/edar/raw/master/sampleData/centralHeating.csv",
                         stringsAsFactors=FALSE,
                         sep =";")
dfHeatEnergy <- dfHeatEnergy %>%
  select(time, energyHeatingMeter) %>%
  na.omit()
dfHeatEnergy$time <- parse_date_time(dfHeatEnergy$time,
                                     orders = "YmdHMS",
                                     tz = "Europe/Zurich")
dfHeatEnergy$day <- as.Date(cut(dfHeatEnergy$time, breaks = "day"))
dfHeatEnergy <- dfHeatEnergy %>%
  group_by(day) %>%
  mutate(energyMax = max(energyHeatingMeter)) %>%
  ungroup()
dfHeatEnergy <- dfHeatEnergy %>%
```

```
  select(day, energyMax) %>%
  unique() %>%
  na.omit()
dfHeatEnergy <- dfHeatEnergy %>%
  mutate(energyCons = energyMax - lag(energyMax)) %>%
  select(-energyMax) %>%
  na.omit()
# merge the data in a tidy format
df <- merge(dfOutsideTemp, dfHeatEnergy, by = "day")
# calculate season
df <- df %>% mutate(season = redutils::getSeason(df$day))
# static chart with ggplot
p <- ggplot2::ggplot(df) +
  ggplot2::geom_point(aes(x = tempMean,
                          y = energyCons,
                          color = season,
                          alpha = 0.1,
                          text = paste("</br>Date:  ", as.Date(df$day),
                                       "</br>Temp: ", round(df$tempMean, digits = 1), "\u00B0C",
                                       "</br>Energy: ", round(df$energyCons, digits = 0), "kWh/d",
                                       "</br>Season: ", df$season))
                      ) +
  scale_color_manual(values=c("#440154", "#2db27d", "#fde725", "#365c8d")) +
  ggtitle("Building Energy Signature") +
        theme_minimal() +
        theme(
          legend.position="none",
          plot.title = element_text(hjust = 0.5)
        )
# interactive chart
plotly::ggplotly(p, tooltip = c("text")) %>%
  layout(xaxis = list(title = "Outside temperature (\u00B0C)",
                      range = c(min(-5,min(df$tempMean)), max(35,max(df$tempMean))), zeroline = F),
         yaxis = list(title = "Daily energy consumption (kWh/d)",
                      range = c(-5, max(df$energyCons) + 10)),
         showlegend = TRUE
         ) %>%
  plotly::config(displayModeBar = FALSE, displaylogo = FALSE)
```


Building Energy Signature
ily energy consumption (kWh
200
150
100
50
0
0
10
20
30
Outside temperature (°C)
Fall
Spring
Summer
Winter

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":

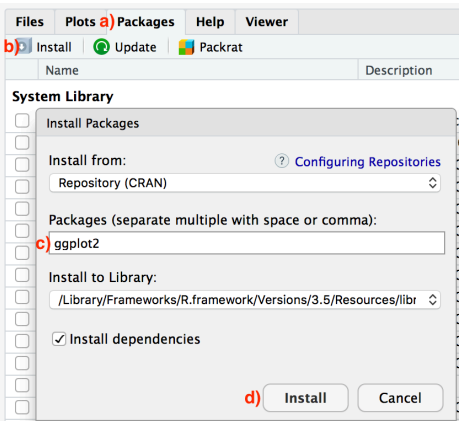


Figure A.1: Install packages via RStudio GUI

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)
```