Energy Data Analysis with R

Reto Marek

2020-10-31

Contents

Preface

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

This book is aimed at R beginners as well as advanced R users and is strongly inspired by the [R Graphics Cookbook] (https://r-graphics.org/). The goal of this book is to additionaly provide specific recipes for energy and comfort related tasks.

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

0.1 Why R and RStudio?

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

Disclaimer The authors decline any liability or responsibility in connection with the published documentation

Chapter 1

Getting started

First you have to install R and RStudio.

Chapter 2

R Basics

2.1 Importing data

2.1.1 csv file

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

```
df <- read.csv("https://github.com/retomarek/r/raw/master/datasets/buildingMonitoringTestDataSet.
```

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

2.1.2 Excel File

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

library(xslx)

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

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

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

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

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

2.2 Data manipulations

2.2.1 data frames

2.2.1.1 change row names of df

```
names(df) <- c("Column1","Column2","Column3")
```

2.2.2 wide to long

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

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

```
## 3      11.90        146.25           2.4                  0                  0
## 4      11.90        157.50           0.8                  0                  0
## 5      11.60        146.25           2.4                  0                  0
## 6      11.75         22.50           0.8                  0                  0
##   FlatHum FlatTemp FlatVolFlowColdwater FlatVolFlowHotwater
## 1      NA       NA                0.006                  0
## 2      NA       NA                0.000                  0
## 3      NA       NA                0.000                  0
## 4      NA       NA                0.000                  0
## 5      NA       NA                0.006                  0
## 6      NA       NA                0.000                  0
```

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

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

```
##                         time          name  value
## 1 2018-09-30T22:00:00.000Z   WthStnPress 1012.3
## 2 2018-09-30T22:00:00.000Z     WthStnHum   87.0
## 3 2018-09-30T22:00:00.000Z    WthStnRain    0.8
## 4 2018-09-30T22:00:00.000Z  WthStnSolRad    0.0
## 5 2018-09-30T22:00:00.000Z    WthStnTemp   12.8
## 6 2018-09-30T22:00:00.000Z WthStnWindDir  157.5
```

2.2.3 long to wide

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

```
##                         time          name  value
## 1 2018-09-30T22:00:00.000Z   WthStnPress 1012.3
## 2 2018-09-30T22:00:00.000Z     WthStnHum   87.0
## 3 2018-09-30T22:00:00.000Z    WthStnRain    0.8
## 4 2018-09-30T22:00:00.000Z  WthStnSolRad    0.0
## 5 2018-09-30T22:00:00.000Z    WthStnTemp   12.8
## 6 2018-09-30T22:00:00.000Z WthStnWindDir  157.5
```

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

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

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

Chapter 3

Data Wrangling

3.1 Add Metadata for later filtering

Firstly we have to load the required libraries and import a dataset into a dataframe:

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

3.1.1 Add 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 <- data

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

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

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

```
head(df,2)
##                  time centralOutsideTemp       year      month        day
## 1 2018-03-21 11:00:00                5.2 2018-01-01 2018-03-01 2018-03-21
## 2 2018-03-21 12:00:00                6.7 2018-01-01 2018-03-01 2018-03-21
##    weekday
## 1 Mittwoch
## 2 Mittwoch
tail(df,2)
##                       time centralOutsideTemp       year      month        day
## 21864 2020-09-17 10:00:00               26.65 2020-01-01 2020-09-01 2020-09-17
## 21865 2020-09-17 11:00:00               28.10 2020-01-01 2020-09-01 2020-09-17
##          weekday
## 21864 Donnerstag
## 21865 Donnerstag
```

3.1.2 Add Season of Year

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

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

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

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

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

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

```
## [1] "Frühling"
```

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

```r
df <- data

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

```r
head(df,2)
##                  time centralOutsideTemp season
## 1 2018-03-21 11:00:00                5.2 Spring
## 2 2018-03-21 12:00:00                6.7 Spring
tail(df,2)
##                       time centralOutsideTemp season
## 21864 2020-09-17 10:00:00               26.65   Fall
## 21865 2020-09-17 11:00:00               28.10   Fall
```

Chapter 4

Explorative Data Analysis

4.1 Get overview

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

4.1.1 Load data

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

```
df <- read.csv("https://github.com/retomarek/r/raw/master/datasets/buildingMonitoringTestDataSet.
```

4.1.2 Names

show the column headers of the data frame

```
names(df)
```

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

4.1.3 Structure

show the structure of the data frame

```
str(df)
```

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

4.1.4 Head/Tail

```
head(df)
```

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

```
tail(df)
```

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

4.1.5 Five number summary

reveals details of a specific series

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

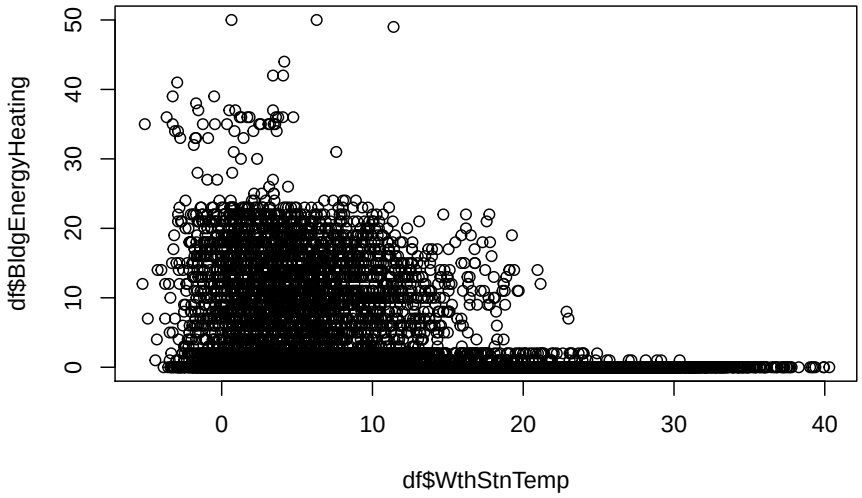
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##   -5.25    5.50   11.25   11.99   17.35   40.30      12
```

4.2 Basic plots

4.2.1 Scatterplot

4.2.1.1 plot()

```
# load data set
df <- read.csv("https://github.com/retomarek/r/raw/master/datasets/buildingMonitoringTe

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



Chapter 5

Data Visualizations

5.1 Building Energy Signature

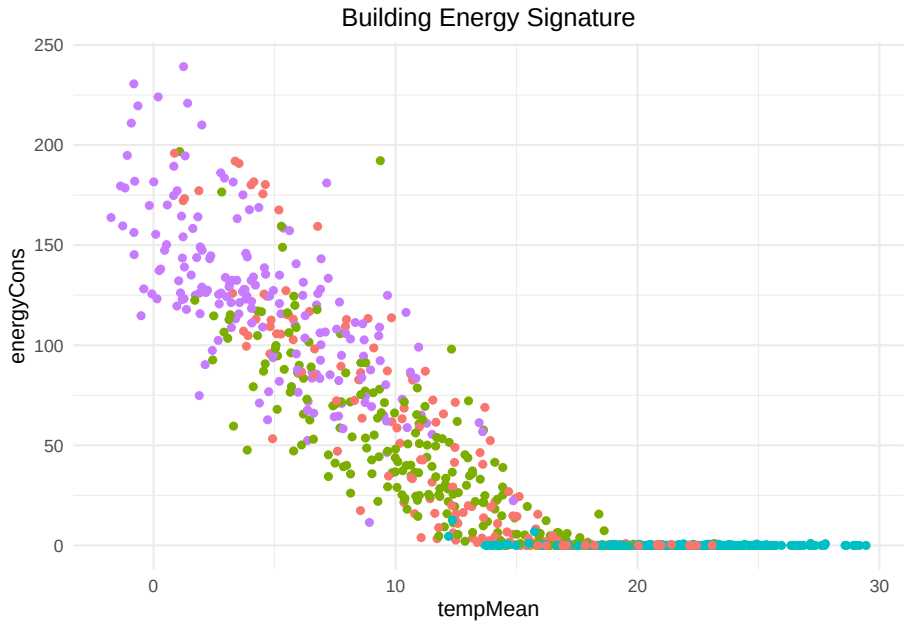
5.1.1 Task

Often the time series data from the outside temperature and the energy data are available in separate formats and time intervals. Therefore the following example imports and aggregates them separately.

5.1.2 Solution

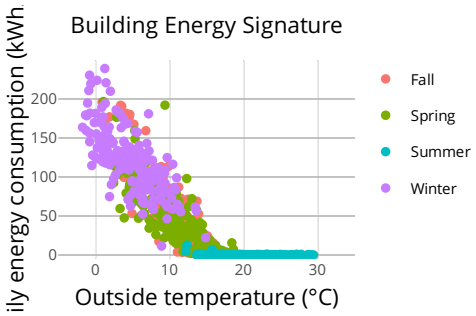
```
library(ggplot2)
library(plotly)
library(dplyr)
library(redutils)
library(lubridate)

# load time series data and aggregate daily mean values
dfOutsideTemp <- read.csv("https://github.com/hslu-ige-laes/edar/raw/master/sampleData/centralOut
                          stringsAsFactors=FALSE, sep =";")
dfOutsideTemp$time <- parse_date_time(dfOutsideTemp$time, "YmdHMS", tz = "Europe/Zurich")
dfOutsideTemp$day <- as.Date(cut(dfOutsideTemp$time, breaks = "day"))
dfOutsideTemp <- dfOutsideTemp %>% group_by(day) %>% mutate(tempMean = mean(centralOutsideTemp))
dfOutsideTemp <- dfOutsideTemp %>% select(day, tempMean) %>% unique() %>% na.omit()

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

```
dfHeatEnergy <- dfHeatEnergy %>% select(time, energyHeatingMeter) %>% na.omit()
dfHeatEnergy$time <- parse_date_time(dfHeatEnergy$time, "YmdHMS", tz = "Europe/Zurich")
dfHeatEnergy$day <- as.Date(cut(dfHeatEnergy$time, breaks = "day"))
dfHeatEnergy <- dfHeatEnergy %>% group_by(day) %>% mutate(energyMax = max(energyHeating
dfHeatEnergy <- dfHeatEnergy %>% select(day, energyMax) %>% unique() %>% na.omit()
dfHeatEnergy <- dfHeatEnergy %>% mutate(energyCons = energyMax - lag(energyMax)) %>% s

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

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

# static chart with ggplot
p <- ggplot2::ggplot(df) +
  ggplot2::geom_point(aes(x = tempMean,
                          y = energyCons, color=season,
                          text = paste("</br>Date:  ", as.Date(df$day),
                                       "</br>Temp: ", round(df$tempMean, digits = 1),
                                       "</br>Energy: ", round(df$energyCons, digits =
                                       "</br>Season: ", df$season))
                      ) +
  ggtitle("Building Energy Signature") +
        theme_minimal() +
        theme(
          legend.position="none",
          plot.title = element_text(hjust = 0.5)
        )
p
```



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

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



5.1.3 Discussion

blabla

5.1.4 See Also

Appendix A

Installing R and R Studio

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

A.1 Download and Install R

A.1.1 Windows

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

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

A.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.

A.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.

A.2 Download and Install RStudio

R Studio is a development environment for R.

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

A.3 Open RStudio

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

Appendix B

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.

B.1 Install Packages

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

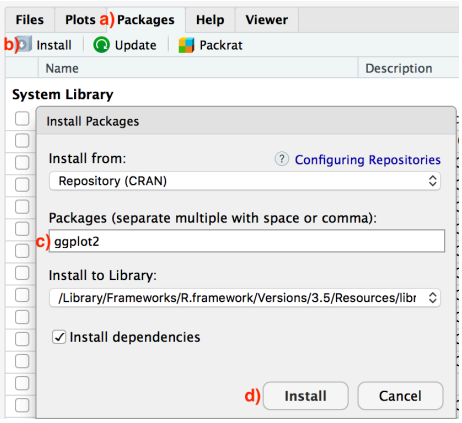


Figure B.1: Install packages via RStudio GUI

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("retomarek/redutils")
```

B.2 Loading Packages

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")
```

B.3 Updating 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)
```