Introduction to "R"
Introduction to R and RStudio

Arndt Leininger

@a_leininger
arndt.leininger@fu-berlin.de

28 September 2019

About the course

Program

Saturday, 28 September 2019

10:00h - 11:30h Introduction to R and Rstudio

11:30h - 11:45h break

11:45h - 13:00h Introduction to R and RStudio (cont.)

13:00h - 14:00h Lunch break

14:00h - 15:30h Data Manipulation

15:30h - 15:45h break

15:45h – 16:30h Data Wrangling

Program

Sunday, 29 September 2019

10:00h - 11:15h Recap and Model Estimation

11:15h - 11:30h break

11:30h - 13:00h Data Visualization

13:00h - 14:00h Lunch break

14:00h - 16:30h Data and Model Visualization

Managing Expectations

- R is a (programming) language.
- Just as you cannot learn Latin in 1 1/2 days, you will not be fluent in R after this workshop.

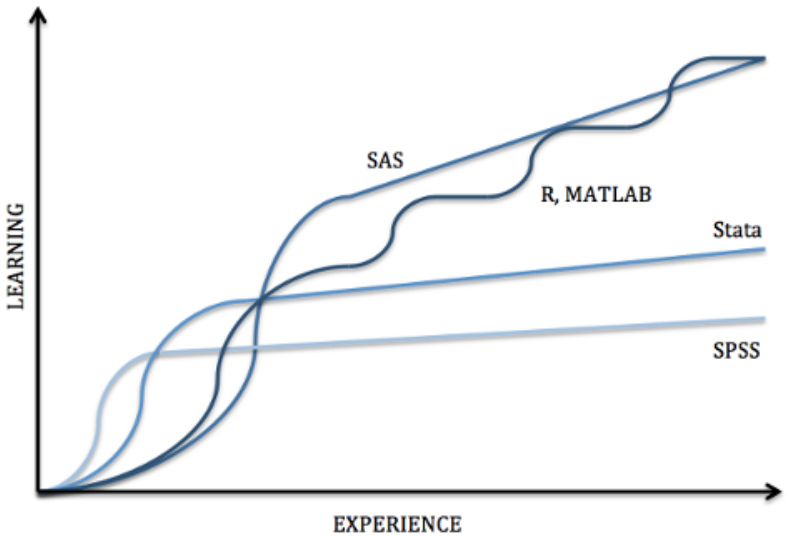


Figure 1: R Learning Curve, https://sites.google.com/a/nyu.edu/statistical-software-guide/summary

- More on Why R is Hard to Learn

Course materials

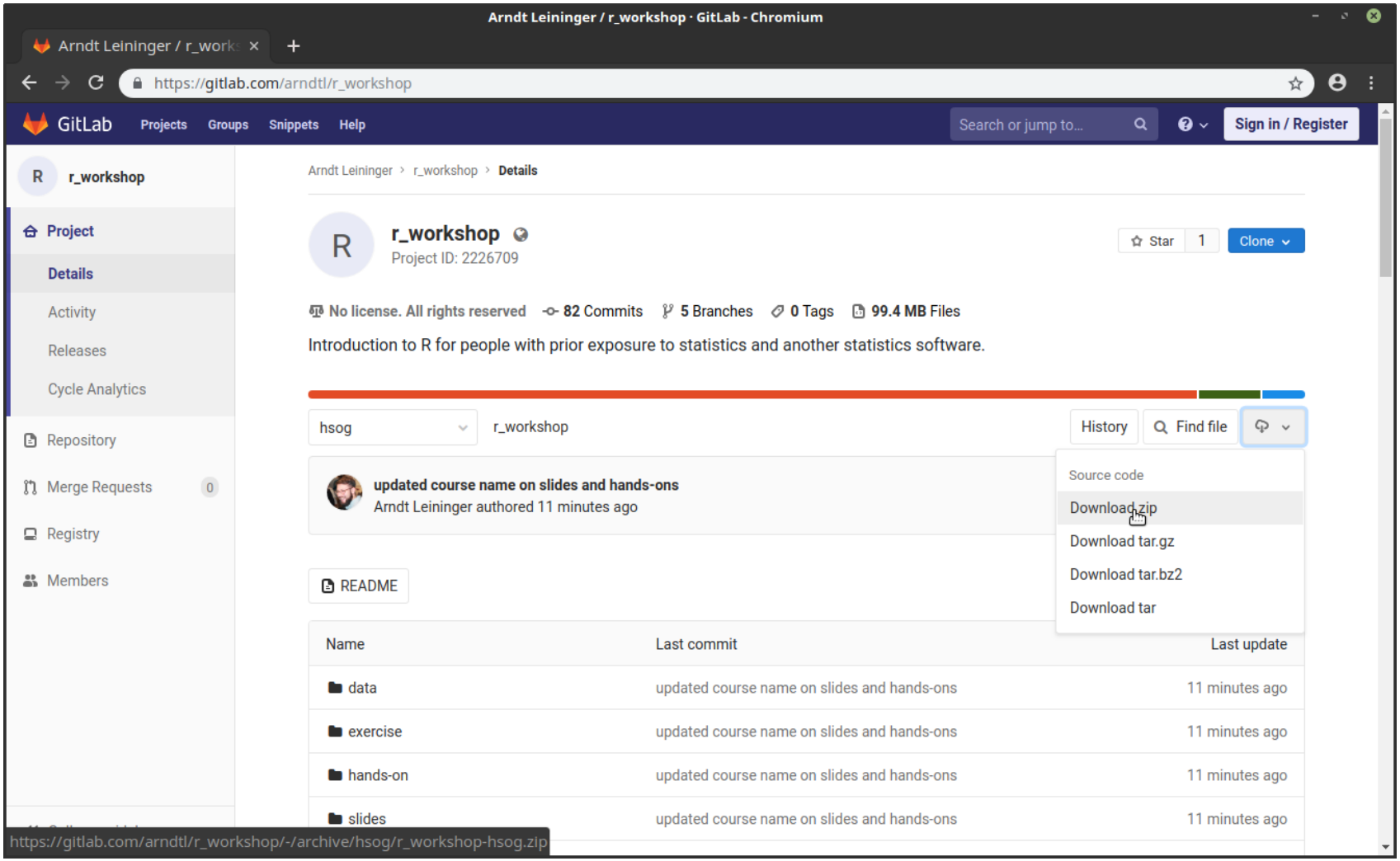


Figure 2: https://gitlab.com/arndtl/r_workshop

Course materials

- Slides, exercises and solutions here
  - https://gitlab.com/arndtl/r_workshop
- Download the zip file
  - or clone the repository if you use Git
- Extract the zip file
- rename the folder to r_workshop if necessary

About R

R

- R is an Open Source environment for statistical computing and graphics available for all common OS: Windows, Mac OS X as well as Linux
- R is being actively developed with two major releases per year and hundreds of releases of add on packages
- R can be extended with 'packages' to add new functionality
- R is an object oriented programming language. Everything in Object-Oriented Programming (OOP) is grouped as self sustainable "objects"

Why R

Because

- Both R and RStudio are open-source and free
- R is a language. Data analysis is done by writing functions and scripts, not by pointing and clicking facilitating reproducible research
- 14932 available packages (as of 20 September 2019)
  - 13314 available packages (as of 1 November 2018)
  - 9293 available packages (as of 5 October 2016)
  - There's a canned solution for almost anything
- A robust and growing community of thousands of contributors and more than two million users around the world
- It works with data from Stata, SPSS, SAS and other statistical software
- Work with multiple datasets at the same time
- Beautiful graphics

Introducing RStudio

Interacting with R

- We'll use R through the RStudio GUI
- The RStudio GUI has 4 panes:
  - *Source*
  - *Console*
  - Two flexible panes, defaults are...
    - Environment, History
    - Files, Plots, Packages, Help, Viewer

RStudio GUI

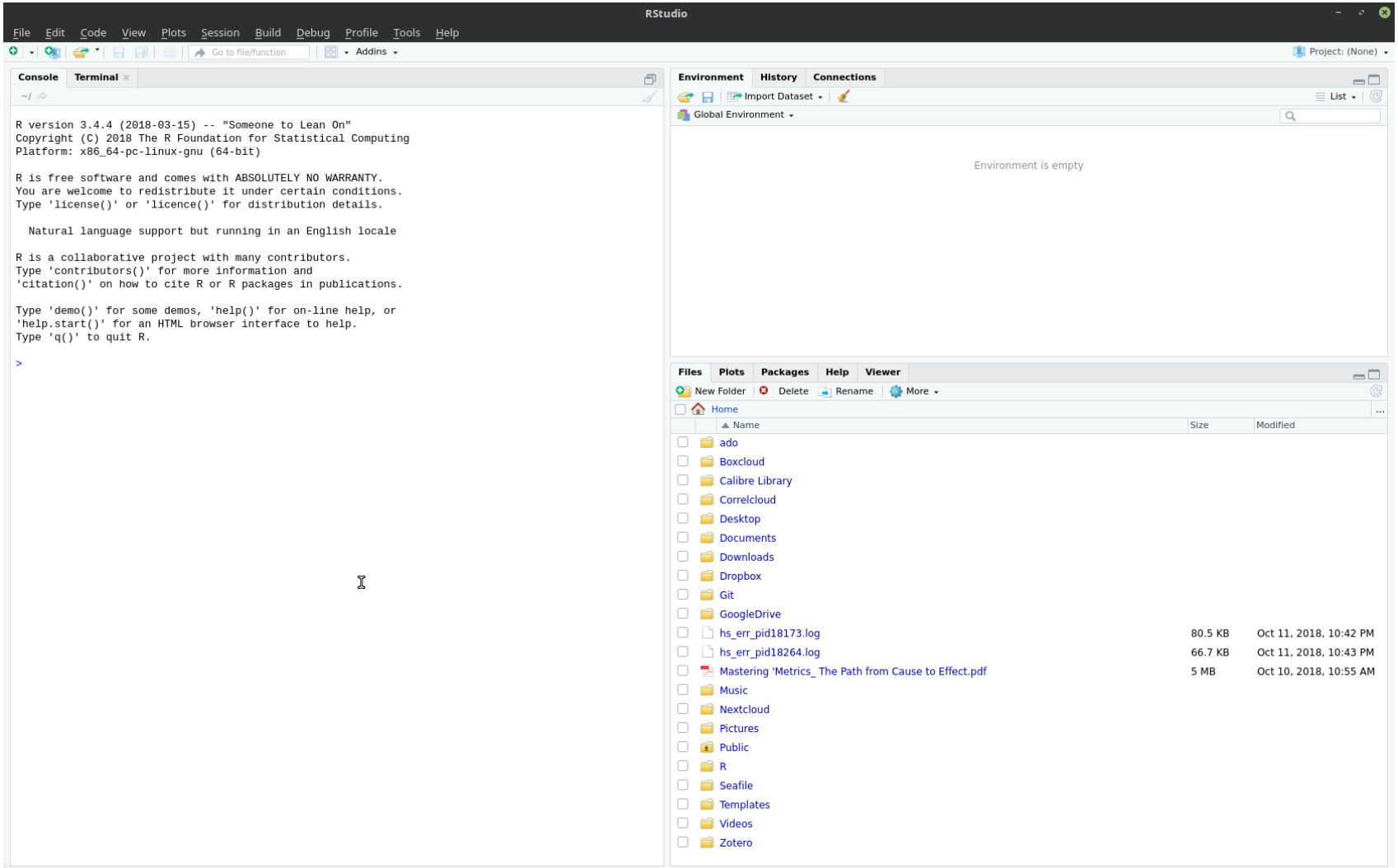


Figure 3: RStudio GUI immediately after starting the software

RStudio GUI

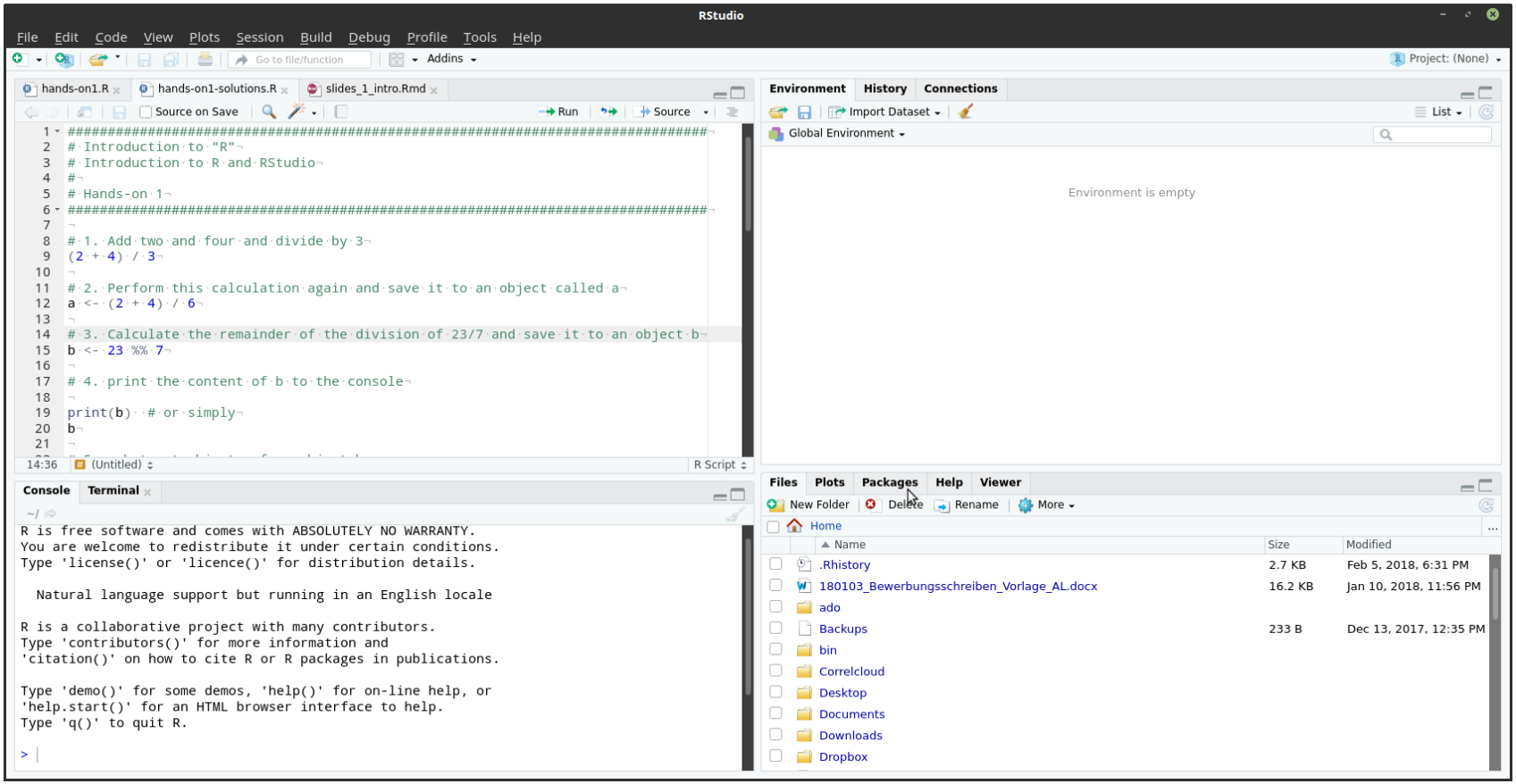


Figure 4: RStudio GUI

RStudio GUI

Options → Pane Layout

Option → Appearance

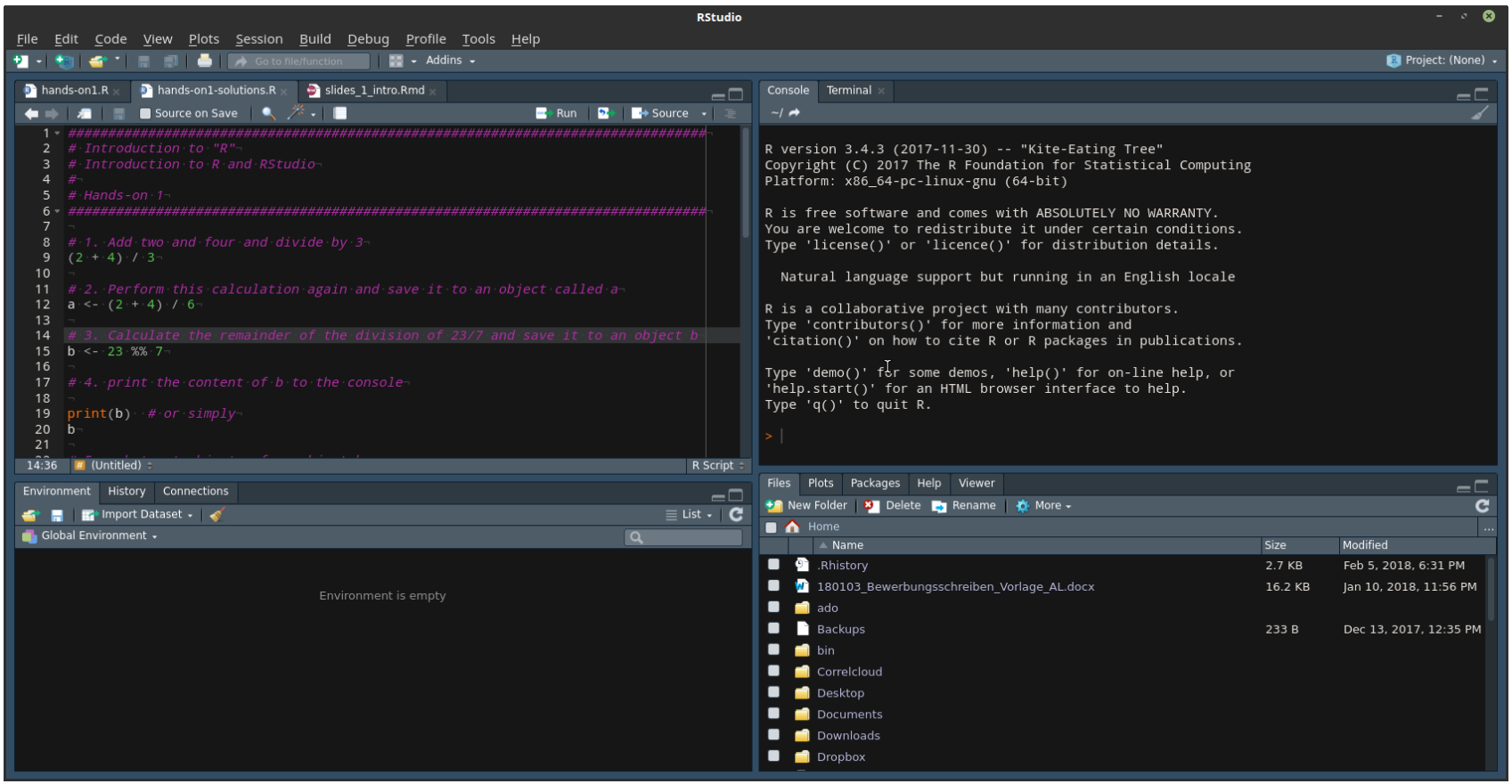


Figure 5: RStudio GUI

First steps in R

Coding in R

Figure 6: R console

- Mathematical expressions will be evaluated

Examples

```
1 + 2
## [1] 3
2^2
## [1] 4
9/3
## [1] 3
# R calculates the result and prints it to console
```

Coding in R

- Character input: R will check for functions and objects in its memory that match the input

```
randomtext
## Error in eval(expr, envir, enclos): object 'randomtext' not f
# returns an error since there is no such object
```

If there is no assignment to an oject output will be printed to the console.

The assignment operator

- Introducing R's assignment operator <-, = also works but is frowned upon
  - Inversely, -> also works
- The shortcut Alt + - produces the assignment operator

```
1 + 3 # calculates the result and prints it to the console
## [1] 4

sum <- 1 + 3  # calculates the result and saves it to
# object 'sum' nothing is printed on the console

sum  # prints 4 on the console
## [1] 4
# when you run an object name it is fed to the print()
# function by default
```

The assignment operator

- There is NO warning that assignment will overwrite prior content
- Objects can take on any type; a numeric object a can easily be turned into a character object a

```
sum
## [1] 4
sum <- '1 + 3'   # "1 + 3" also works
sum
## [1] "1 + 3"
```

Quick hands-on

1. Save the text `"Hello World"` to an object called `obj`.
2. Austria became a member of the European Union in 1995. Calculate how many years it's been since then.
3. Save the result of that calculation to `obj`.

<- vs. =

- the origins of the <- symbol come from old APL keyboards that actually had a single <- key on them
- however, there are subtle differences between <- and =

```
x <- y <- 5
x = y = 5
x = y <- 5
x
## [1] 5
y
## [1] 5
x <- y = 5
## Error in x <- y = 5: could not find function "<-<-"

median(a <- 1:10)
## [1] 5.5
median(a = 1:10)
## Error in is.factor(x): argument "x" is missing, with no defau
```

<- vs. < -

```
x<-3
x
x < -3
```

http://stat.ethz.ch/R-manual/R-patched/library/base/html/assignOps.html

Objects vs. text

```
hello
## Error in eval(expr, envir, enclos): object 'hello' not found
'hello'
## [1] "hello"
hello <- 'hello'
hello
## [1] "hello"
hello <- 'Hello'
hello
## [1] "Hello"
```

Understanding functions

- R functions are what in other programs are called commands, i.e. Stata. They take information we give to them, do something with the information, then output something.
- We pass information to functions with arguments.

```
print(x = 'Hello World!')
## [1] "Hello World!"
# x is the name of argument which is supplied with
# the character vector 'Hello Wien' which is then
# printed to the console

print('Hello World!')
## [1] "Hello World!"
# you can omit argumnt names and R will use the order
# of arguments to identify them
```

- The function `print()` is also called when you just enter the name of an object or text ike `"Hello World!"`

Functions vs. Objects

```
print <- 'Hello World!'  # this is an object
print()  # this is a function
## Error in print.default(): argument "x" is missing, with no de
print(print)  # prints the object 'print'
## [1] "Hello World!"
```

Functions alway have brackets even if there're no arguments.

```
Sys.Date()
## [1] "2018-12-02"
```

Understanding functions

```
# creates a vector of 100 random draws from a normal
# distribution with mean 1 and standard deviation 2
rnorm(n = 100, mean = 1, sd = 2)
##   [1] -0.707344879  0.727899185  1.918621483  0.258439422  4.
##   [6]  0.397151774 -3.893500155  5.440002461  0.649397114  4.
##  [11]  0.942541332 -0.953931124 -0.169408538 -3.137676621  0.
##  [16] -0.909244099  3.357195076 -3.789542987  0.324946211  2.
##  [21]  3.264130573  2.868485670  4.410816798  1.299895083  1.
##  [26] -0.372693152 -0.252689172  3.664434643 -0.603908002  4.
##  [31] -0.279225108  1.225974652  0.363343214  1.385382122 -0.
##  [36]  0.053095021 -0.851303464  0.049885457  2.737777983 -2.
##  [41]  3.402432201  1.232511620  0.077457832 -0.846792555  0.
##  [46]  1.980387986  1.830261038  3.269633973  3.577006674 -3.
##  [51]  0.760260254  1.243975237  0.763603120  4.066515647  1.
##  [56]  2.242083657 -0.895299377  0.110240180  0.456215260  0.
##  [61] -2.205282847 -0.227519134  1.418071528  2.307632949  5.
##  [66]  3.682111093 -0.178927702  1.822191572 -0.340547732  0.
##  [71]  2.479903359  2.949368148  4.265056353  0.372564362  2.
##  [76]  2.082144201  0.219684553  2.304222931  2.136638959 -0.
##  [81] -1.836921098  0.262967499 -0.007715346  2.263246120  1.
```

Understanding functions

```
rnorm(100, 1, 2)  # does the same
##   [1] -0.98056021 -1.55611758 -2.49909187 -1.92152342  2.0826
##   [6]  1.93576599  1.56232244  5.90416531 -0.90633413  1.7556
##  [11]  1.51793540  0.33429831  0.30081082  3.82647904 -0.7317
##  [16]  0.25688217  1.03657106 -0.01175756  0.03901799 -0.3222
##  [21]  1.32056669  2.32711434  0.24726687  0.19404016 -0.1379
##  [26]  2.29400128  0.51495972  0.35373969 -0.99231190  1.8237
##  [31]  1.05163114  1.03569712 -2.00559252  2.24561696 -0.8348
##  [36]  4.09007248  4.18996684 -1.27606811  0.73655098  3.8459
##  [41]  1.65909812 -0.82735470  6.04408268  0.07984806  3.9382
##  [46] -0.44244249  2.31461512  0.97975365  2.08750001  5.3584
##  [51]  4.31666785  0.73012345  0.72089116 -0.33036295  1.8399
##  [56]  1.04075804  1.13990214  2.99554077  0.95820985 -0.4696
##  [61] -0.43200577  0.33053650  0.83331540 -0.86989307  2.3328
##  [66]  1.67649715  4.92787844 -3.33225991 -0.74181376 -3.5437
##  [71]  0.70283465 -1.13978773 -2.97021267  2.64203838  2.3346
##  [76]  0.01394200  1.40467928 -0.29154057  2.50408381  2.8268
##  [81] -1.09231370  0.81305931  0.65870670  0.66587899  4.2618
##  [86]  1.83137155  1.48196914  3.11230765  0.48762924  0.3157
##  [91]  0.82125204 -1.75860933 -0.45735483 -0.29379886  1.9154
```

Understanding functions

```
rnorm(1, 2, 100)  # does this do the same?
```

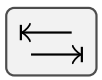
Understanding functions

```
rnorm(1, 2, 100)  # does this do the same?
## [1] -59.52945

d <- rnorm(100, 1, 2)  # the output of the function can of
# course be assigned to an object

d  # prints the object to the console
##   [1] -1.12223316  1.23576691  1.46813447  2.17204766  2.0712
##   [6]  3.89424491  0.25093104 -2.37979916  0.88302276  1.5284
##  [11] -0.81341596  0.42733343  2.28766626  1.73776331 -2.9127
##  [16] -2.60684672 -0.96654004 -1.63936442  0.80217490 -0.2506
##  [21]  0.65499491 -2.49487388 -0.80883236  0.61378627  3.7418
##  [26]  3.70488976  1.38433962  3.46548749  2.05874223  0.5640
##  [31] -0.70548441  1.07446648  2.14457177  0.85810856 -0.1800
##  [36]  1.19453528  0.60815483  1.35143330  5.43913239  3.1887
##  [41]  0.39299010  1.70490800  2.52949118  3.96936553  1.1573
##  [46] -0.72746364  2.69747352  1.31394570  3.00258934  3.1556
##  [51]  3.45155386  4.15318934 -2.18398450 -3.07968681  1.8207
##  [56]  0.68040153 -0.02505842  2.17457597 -0.24416019  2.1803
##  [61] -2.60669945  6.18675894  1.36343843 -1.36540162 -2.8299
```

Getting help

- When writing function calls hit  for autocomplete of function names, argument names and arguments
- Type `help()` and the name of the function within the brackets.
  - For example, `help(lm)` will take you to the `lm()` functions's manual
  - `?lm` is shorter and does the same

Reading a help file

1. *Description* what the command does
2. *Usage* Shows a call to the function with all arguments set to their default.
3. *Details* and *Values* Explanation of further particulars
4. *Examples* Code snippets

- You don't always need to set a lot of arguments but you should pay attention to what the defaults are.
- If you don't know the function you want to use, you will have try and describe your problem to a search engine of your choice.

Writing good code

Good code should be readable by humans as well as computers.

- Use comments to explain what each line is doing (#)
  - This can be annoying in the work process but it's even more annoying to try to understand uncommented code
- Empty lines and spaces make your code easier to read
- Use meaningful names when you create them, and write them consistently (`variablename`, `variableName` or `variable_name`, ...)
- Follow indentation conventions e.g.

```
c('Concatenate two rather length texts',
  'so that they form a vector of lenght 2')
## [1] "Concatenate two rather length texts"
## [2] "so that they form a vector of lenght 2"
```

Comments

- Commenting code is important. It enables others to understand your code (that includes your future self!)
- There is only one comment sign in R: #
- Everything on a line following the # is a comment
- For a multiline comment every line needs to start with a #
- The keyboard shortcut in RStudio for commenting out one or multiple lines is Ctrl + ⇧ + C
  - Use the same shortcut on a comment to 'uncomment'

Comments

- An example from `hands-on/01_intro/hands-on1.R`

```
# Introduction to "R"
# Introduction to R and RStudio
#
# Hands-on 1

# 1. Add two and four and divide by 3
(2 + 4) / 3 # adds 2 and 4 together and then divides by 3
## [1] 2
```

More on RStudio

Running code

. . . from the source editor

- Ctrl + ↵ runs the code on a single line
  - Set "Ctrl + Enter" to "Current line" in Options → Code: If a function is not finished on a line R will wait for further input
  - You can run subparts of a line or multiple lines by highlighting the code you want to execute and then pressing Ctrl + ↵

. . . from the console

- Hit ↵ to run a line of code

Use Ctrl + 1 and Ctrl + 2 to swith between editor and console.

Useful things about the Source Editor

- Line numbering
- Syntax highlighting
- You can set a highlighting scheme and other options for how code is displayed
- Autocomplete for object names and functions
  - Type the first few characters of an object name, hit ⇤⇥, RStudio will suggest what to write
  - Hit ⇤⇥ within a function and you'll get a dropdown of all arguments of the function incl. a brief explanation
  - Hit ⇤⇥ after the equal sign following an argument and you'll get suggestions for inputs to the argument
- Any code is copied from the script to the console and then executed

Useful things about the console

- Almost all of these features can be used in the console
- You can toggle through prior executed functions by
  - [↑] and [↓]
  - [Ctrl]+[↑] and then [↑] and [↓] to toggle
- Clear screen: [Ctrl]+[L]

Keyboard shortcuts thus far

- Assignment operator: Alt + -
- Autocomplete: ⇤⇥
- Comment and uncomment: Ctrl + ⇧ + C
- Run code in script: Ctrl + ↵
- Run code in console: ↵
- Toogle through executed code: ↑ and ↓
- Clear screen: Ctrl + L
- Switch to source: Ctrl + 1
- Switch to console: Ctrl + 2
- Keyboard Shortcut Help: ⇧ + Alt + K

Some mathematical expressions in R

```r
2 + 2  # Addition

3 - 2  # Substraction

2 * 2 # Multiplication

10 / 3  # Division

a %*% b # Matrix multiplication
```

https://stat.ethz.ch/R-manual/R-devel/library/grDevices/html/plotmath.html

Hands-on 1

Hands-on 1

`hands-on/01_intro/hands-on1.R`

Working with data

Opening a file

- Reading a file is done through a function, here we'll use `read.csv()`
- This function is for reading comma-separated vector files, short CSV
- Here's an example

```
V1, V2, V3, V4
1, 0, 2, male
2, 2, 1, female
```

- Using `?read.csv` we learn that the function takes a file path as input
  - All other arguments have defaults and do not need to be specified

File paths

- File paths tell the computer where to read and write information.
- Separators between folder names differ between Windows (\) and Mac/Linux (/).
- However, in R, independent of the OS used, forward slashes (/) are used exclusively.

Absolute and relative file paths on Windows

```
C:
+- Users
   +- Bill
   |  +- Dropbox
   |     +- coolproject
   |        + code
   |        + data
   |          +- master.csv
   +- Melinda
```

- Absolute path to project folder:
  `C:/Users/Bill/Dropbox/coolproject`
- Relativ path to file `master.csv`: `data/master.csv`

Absolute and relative file paths on Mac OS

```
/
+- Users
   +- Steve
   |  +- Dropbox
   |     +- coolproject
   |        + code
   |        + data
   |          +- master.csv
   +- Laurene
```

- Absolute path to folder `coolproject`: `/Users/Steve/project_w_bill_and_linus`
- Relative path to file `master.csv`: `data/master.csv`

Absolute and relative file paths on Linux

```
/
+- home
   +- linus
   |  +- Dropbox
   |     +- coolproject
   |        + code
   |        + data
   |          +- master.csv
   +- tove
```

- Absolute path to project folder: `/home/linus/coolproject`
- Relative path to file script.R: `data/master.csv`

Absolute vs. relative paths

- Ideally use an absolute path only for setting the working directory
- Everything else should be relative paths
- Absolute paths cause errors when you give a project folder to a coauthor/collaborator

```
setwd('C:/Users/Bill/project_w_steve_and_linus') # code for Bill
setwd('/Users/Steve/project_w_bill_and_linus') # code for Steve
setwd('/home/linus/project_w_bill_and_steve') # code for Linus

df <- read.csv('data/master.csv') # same relative path for every
```

Directory structure

- *Absolute paths* start from the top of the tree and specify each subdirectory until the file
  e.g. `C:\Users\Bill\document.docx` (or `/home/linus/document.tex`, or even `https://gitlab.com/arndtl/r_workshop`)
- However in R the paths are `C:/Users/Bill/document.docx` and `/home/linus/document.tex`
- *Relative paths* start from the current working directory; use `../` to move to the directory above the current one
- R starts with your home directory as working directory.
- Say you have your Stats II stuff in `C:\User\JDoe\Dropbox\stats2` then to set this folder as working directory you use
  `setwd('C:/User/JDoe/Dropbox/Hertie/stats2')`

Converting \ to /

```
gsub("\\\\",  "/",  readClipboard())
```

Opening a dataset

```
/
+- home
   +- arndt
      +- Git
         +- r_workshop
            +- data
               +- BundestagForecastReplicationData.csv
```

```r
# 1. Set your working directory
setwd('/home/arndt/Git/r_workshop/')
```

```r
# 2. open the file
# what happens if you run this line of code?
read.csv('data/BundestagForecastReplicationData.csv')
```

Data from Kayser, Mark A. and Arndt Leininger (2016) "A Predictive Test of Voters' Economic Benchmarking: The 2013 German Bundestag Election", *German Politics*, 25(1), pp. 106-130

Opening a dataset

```
# 2. open the file
# this happens
read.csv('data/BundestagForecastReplicationData.csv',
                 stringsAsFactors = F)
```

```
##      X wp year       date              outgovcoa     chdate
## 1    1  1 1949 14.08.1949                        15.09.1949
## 2    2  2 1953 06.09.1953 CDU/CSU, FDP and DP 15.09.1949
## 3    3  3 1957 15.09.1957         CDU/CSU, DP 15.09.1949
## 4    4  4 1961 17.09.1961             CDU/CSU 15.09.1949
## 5    5  5 1965 19.09.1965     CDU/CSU and FDP 16.10.1963
## 6    6  6 1969 28.09.1969     CDU/CSU and SPD 01.12.1966
## 7    7  7 1972 19.11.1972         SPD and FDP 21.10.1969
## 8    8  8 1976 03.10.1976         SPD and FDP 16.05.1974
## 9    9  9 1980 05.10.1980         SPD and FDP 16.05.1974
## 10 10 10 1983 06.03.1983     CDU/CSU and FDP 01.10.1982
## 11 11 11 1987 25.01.1987     CDU/CSU and FDP 01.10.1982
## 12 12 12 1990 02.12.1990     CDU/CSU and FDP 01.10.1982
## 13 13 13 1994 16.10.1994     CDU/CSU and FDP 01.10.1982
## 14 14 14 1998 27.09.1998     CDU/CSU and FDP 01.10.1982
```

Opening a dataset

```
# 2. open the file and assign its content to an object
df <- read.csv('data/BundestagForecastReplicationData.csv',
               stringsAsFactors = F)
```

- The argument `stringsAsFactors = F` will be explained in a bit

getwd()

If you're unsure what your working directory is set to:

```
getwd()
```

```
## [1] "/home/arndt/Git/r_workshop"
```

Quick hands-on

1. Set the working directory to the r_workshop folder using (`setwd()`).
2. Read the file `BundestagForecastReplicationData.csv` contained in the subfolder `data`. Assign the output of `read.csv()` to an object called `df`.

Data in R

- Data sets in R are most often saved as objects of type `data.frame`.
- A `data.frame` ist just another object and so you can have multiple objects of type `data.frame` in your memory at the same time.

```
df2 <- df
```

- An understanding that a data.frame is a matrix is an important basis for competent usage of R.
- As usual: columns are variables and rows are observations.

A first glance at the data

```
dim(df)  # returns row and column count
## [1] 18 14
nrow(df)  # returns the number of rows
## [1] 18
ncol(df) # returns the number of cols
## [1] 14
summary(df)
##        X                wp              year          date
##  Min.   : 1.00   Min.   : 1.00   Min.   :1949   Length:18
##  1st Qu.: 5.25   1st Qu.: 5.25   1st Qu.:1966   Class :charac
##  Median : 9.50   Median : 9.50   Median :1982   Mode  :charac
##  Mean   : 9.50   Mean   : 9.50   Mean   :1981
##  3rd Qu.:13.75   3rd Qu.:13.75   3rd Qu.:1997
##  Max.   :18.00   Max.   :18.00   Max.   :2013
##
##   outgovcoa            chdate            chancellor
##  Length:18          Length:18          Length:18
##  Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character
##
```

Variables

- Variables in a data.frame can be accessed via a simple method in R

```
# Our data.frame contains many variables
# To get a list of the variable names and the variable names onl
names(df)
##  [1] "X"           "wp"          "year"        "date"
##  [6] "chdate"      "chancellor"  "opcandidate" "outgovshare"
## [11] "logterms"    "gergrow"     "benchgrow"   "pid"

# Say we're interested in the variable 'outgovshare'
df$outgovshare
##  [1]   NA 58.0 53.6 45.4 57.1 46.1 54.2 50.5 53.5 55.8 53.4 5
## [15] 47.1 42.3 33.8   NA
```

Finding out about the type of a variable

- `class()` returns the type of a variable or in fact any object in R

```
class(df$wp)
## [1] "integer"
class(df$outgovcoa)
## [1] "character"
class(df)
## [1] "data.frame"
```

Quick quiz

Which of the code snippets are complete statements that R will run without error? What's wrong with the others?

1. `print("Hello World!')`
2. `B <- 2/3`
3. setwd('C:\Users\Arndt\Documents')
4. `getwd`
5. `rnorm(n = 10, mean = 0, sd = 2 # sd is short for standard deviation)`

Interlude: the concatenate function c()

- c() is short for concatenate
- It assembles multiple individual values into a vector
- E.g.

```
c(1, 2, 4, 5)
## [1] 1 2 4 5

c(1, 2, TRUE, "Hallo")
## [1] "1"     "2"     "TRUE"  "Hallo"
```

- If different types of values are pased on to c() the function will force them to be of type character

Variables

There are basically four types of variables in R.

1. Numeric
2. Character
3. Factor
4. Logical

(Yes, there's also a `datetime` type.)

Numeric vector

- A numeric vector contains numbers
- With `as.numeric()` you can turn numbers saved as characters into numbers

```
'3' / 3
## Error in "3"/3: non-numeric argument to binary operator

as.numeric('3') / 3
## [1] 1
```

Character vector

- Character vectors contain text just as string variables in Stata
- With `as.character()` you can turn objects into character objects
- Both " and ' can be used

```
# Example
four <- '4'

four
## [1] "4"
four * 4
## Error in four * 4: non-numeric argument to binary operator
as.numeric(four) * 4
## [1] 16
```

Factor variables

- Factors are used to save categorical (nominal or ordinal) variables
- With `as.factor()` you can turn objects into factor objects
- By default, when reading datasets with `read.csv()`, R will turn character variables into factors unless you set `stringsAsFactors = FALSE`

```
fac <- factor(c("Democrat", "Republican", "Independent"))
fac
## [1] Democrat    Republican  Independent
## Levels: Democrat Independent Republican
```

Factor variables

- Ordered factors can be created by setting the argument `ordered=T`

```
data_analysis_software <- c('Excel', 'SPSS', 'Stata', 'R')
data_analysis_software <-
  factor(data_analysis_software,
         levels = data_analysis_software, ordered = T)
data_analysis_software
## [1] Excel SPSS  Stata R
## Levels: Excel < SPSS < Stata < R
# ;-)
```

Logical

- Logical is a boolean factor that takes on the values `TRUE` (also abbreviated as `T`) and `FALSE` (`F`)
- Logical vectors can be used in mathematical operations: `TRUE` is treated as 1 and `FALSE` as 0

```
truefalse <- c(TRUE, FALSE, T, F)
truefalse
## [1]  TRUE FALSE  TRUE FALSE

example <- 2 > 1
example
## [1] TRUE
```

Quick hands-on

1. Create a vector called `boolean` which contains two elements, the values `TRUE` and `FALSE`, by using the concatenate function `c()`
2. Create a vector called `number` containing the values `2` and `3`
3. Multiply the vectors `boolean` and `number`
4. Save the result in a vector called `result`
5. Print the vector `result` to the console
6. Check the type of the vector `result` using `class()`
7. Turn vector `result` into a vector of type `character`

Creating a new variable

- Creating a new variable is done by assigning a value to a previously undefined variable

```
df$newvar <- df$outgovshare / 100
# express vote share in fration instead of percentages
df$newvar
##  [1]    NA 0.580 0.536 0.454 0.571 0.461 0.542 0.505 0.535 0.
## [12] 0.548 0.484 0.413 0.471 0.423 0.338    NA
```

Replacing the content of a variable

```
df$outgovshare <- df$newvar
# the content of the variable is replaced without any warning!
```

Deleting a variable

```r
ncol(df)
## [1] 15
df$newvar <- NULL
ncol(df)
## [1] 14
```

Deleting an object

```
# but
a  <- NULL
a
## NULL
rm(a)  # deletes object a from R's memory
```

- NULL is the logical representation of a statement that is neither TRUE nor FALSE
- https://www.r-bloggers.com/r-na-vs-null/

Calculating the mean

```
mean(df$outgovshare)
## [1] NA
```

Missing values

```
df$outgovshare
##  [1]   NA 58.0 53.6 45.4 57.1 46.1 54.2 50.5 53.5 55.8 53.4 5
## [15] 47.1 42.3 33.8   NA
```

- Missing values in R are denoted by `NA`
- `NA` can appear in numeric, factor and character variables
- `NA`s are not automatically disregarded by all functions

Calculating the mean

```
mean(df$outgovshare, na.rm = T)
## [1] 49.70625
```

Demeaning a variable

```
df$demeaned <- df$outgovshare - mean(df$outgovshare,
                                     na.rm = T)
```

Saving data

- `write.csv()` writes a `data.frame` object to a CSV-file
- It takes an object and file path as input
- Hint: set `row.names = F` otherwise R will write a first column of row names (most of the times simply a running count of the lines) into the file. The first row of this first column will be empty which can cause problems with other programs when trying to open the file

```
getwd()
```

```
## [1] "/home/arndt/Git/r_workshop/"
```

```
write.csv(df, 'data/newdata.csv', row.names = F)
```

row.names = T

```
write.csv(df, 'data/newdata.csv', row.names = F)
write.csv(df, 'data/newdata.csv')
```

Resulting file:

```
"X","wp","year","date","outgovcoa", ...
"1", 1,1,1949,"14.08.1949","","15.09.1949","", ...
"2", 2,2,1953,"06.09.1953","CDU/CSU, FDP and DP", ...
"3", 4,4,1961,"17.09.1961","CDU/CSU","15.09.1949", ...
...
```

row.names = F

```
write.csv(df, 'data/newdata.csv', row.names = F)
```

Resulting file:

```
"X","wp","year","date","outgovcoa", ...
1,1,1949,"14.08.1949","","15.09.1949","", ...
2,2,1953,"06.09.1953","CDU/CSU, FDP and DP", ...
4,4,1961,"17.09.1961","CDU/CSU","15.09.1949", ...
...
```

Hands-on 2

Hands-on 2

`hands-on/01_intro/hands-on2.R`

Appendix

Some commonly used functions in R

http://www.statmethods.net/management/functions.html

Keyboard shortcuts

- ⇤⇥ for autocomplete
- Alt + - produces the assignment operator <-
- Ctrl + ⇧ + C for comments, toggle on/off
- Ctrl + ↵ runs from the script
- ↑ and ↓
- In console Ctrl + ↑ as well as ↑ and ↓ to toggle input
- Switch to source: Ctrl + 1
- Switch to console: Ctrl + 2
- Ctrl + L produces a clear screen

Cheat sheets

Base R

RStudio IDE

Data Visualization with ggplot2

Data Transformation with dplyr

Colors in R

R Reference Card

Built-in Functions

Model formulas in R

Some cheat sheets can also be found through RStudio: Help > Cheatsheets.

Find more at rstudio.com/resources/cheatsheets/