

Statistical Computing

Michael Mayer

February 2023



Statistical Computing: What will we do?

Chapters

1. R in Action
2. Statistical Inference
3. Linear Models
4. Model Selection and Validation
5. Trees
6. Neural Nets

Remarks

- ▶ Chapters 3 to 6:
Statistical ML in Action
- ▶ Two weeks per chapter
- ▶ Exercises at end of chapter notes

R in Action

Outline

Refresh R skills



Data Analysis

- ▶ Base R
- ▶ The pipe
- ▶ dplyr
- ▶ ggplot2
- ▶ R Markdown

Writing Functions

- ▶ Why functions?
- ▶ Code style
- ▶ Organization
- ▶ dplyr and ggplot2
- ▶ plot, print, summary

Data Analysis

- ▶ Throughout the lecture, we will work with R
- ▶ Today, focus is on **data preparation** and **descriptive analysis**
- ▶ Essential part of every analysis
- ▶ Base R has hundreds of functions to help you here

Example



Contributed extension packages help as well. Let's look at some of them ...

The Pipe Operator avoids Function Chains



- ▶ In “magrittr” package
- ▶ On CRAN since 2014 (Stefan Milton Bache)
- ▶ Turns $f(x, y)$ into `x %>% f(y)`
- ▶ Since R 4.1, `"|>"` in base R
- ▶ Use short-cut "Ctrl-shift-m"

Example



Figure: https://en.wikipedia.org/wiki/The_Treachery_of_Images

“dplyr”: Grammar of Data Manipulation



Core “verbs”

- ▶ `select()`
- ▶ `filter()`
- ▶ `arrange()`
- ▶ `mutate()`
- ▶ `summarize()`

Dream team with pipe

- ▶ Verbs take as first argument a dataframe and return modified dataframe
- ▶ On CRAN since 2014 (Hadley Wickham)
- ▶ Like “magrittr”, part of tidyverse
- ▶ Translate between base R and dplyr?

Example

“ggplot2”: Grammar of Graphics



Modify plot layer per layer

- ▶ Simple “+” instead of %>%
- ▶ On CRAN since 2007 (Hadley Wickham)
- ▶ Also in tidyverse

“plotly” package

- ▶ Interactive figures
- ▶ Wraps JavaScript library
- ▶ Can translate ggplot to Plotly

Example

Reports with R Markdown



Markdown text + R code

→ HTML, Word, PDF

On CRAN since 2014 (Yihui Xie)

Example

- ▶ Markdown syntax
- ▶ Simple R Markdown file



Workflow

1. Create .Rmd with YAML header
2. Write analysis and adapt YAML
3. Hit “Knit”/`rmarkdown::render()`

When you hit “Knit”

1. `knitr::knit()` searches code chunks, runs them and “knits” the results with your Markdown text to a temporary .md file
2. The .md file is converted to desired output with Pandoc, using YAML header to specify its call

Writing Functions

Already used many functions

- ▶ `mean()`
- ▶ `ggplot2::ggplot()`
- ▶ `+`

Write **own** functions to

- ▶ avoid code duplication
- ▶ produce readable code

Example

Code style

- ▶ Line length, curly braces, spaces?
- ▶ Comments: Why, not what
- ▶ Defensive programming: `if + stop()`

Example

Organizing functions

`source(functions.R)` or **own package**

Example

dplyr and ggplot2 in functions

Unquoted variable names look nice

- ▶ `diamonds %>% select(price, color)`
- ▶ `ggplot(diamonds, aes(x = color))`
- ▶ `facet_grid(~ color)`

But what if “price” and/or “color” should be passed as function arguments?

Some solutions

- ▶ `select(all_of(c("price", "color")))` → `select(price, color)`
- ▶ `aes(x = .data[["color"]])` → `aes(x = color)`
- ▶ `reformulate("color")` → `~ color`

*used in lin. models
etc
a R formula*

Example

plot, print, summary

Generic functions

- ▶ plot, print, summary, predict, ...
- ▶ Depend on **class** of object

Example

S3 System

- ▶ Simple object oriented system
- ▶ Generic function calls UseMethod() to find class method, e.g.
plot() → UseMethod() → plot.factor()
- ▶ Write new class method of existing generic function
- ▶ Or write new generic

Example