Evolutionary Genetics

LV 25600-01 | Lecture with exercises | 4KP

Jean-Claude Walser

jean-claude.walser [at] usys.ethz.ch

HS2024

What lines of code should not be missing in any R **script**?

The **first** and **last** lines of any **R scripts**

Start

```
## clean/reset environment
rm(list = ls())
```

End

```
## Session-Log
> sink("SessionInfo.txt")
>   sessionInfo()
> sink()
```

Loading R packages: What is the difference between the following R commands?

```
(A) install.packages("dplyr")

(B) load("dplyr")

(C) source("dplyr")

(D) require("dplyr")

(E) library("dplyr")
```

```r
# Install R packages from repositories (or local files)

install.packages("dplyr")

install.packages(c("dplyr", "plyr"))

install.packages("lattice", repos="http://cran.r-project.org")

install.packages("local.R.Package", lib.loc="/my/local/R/library")
```

```
## Save&Reload datasets written with the function save

# save object x
save(x, file = "x.RData")

# save all objects
save(list = ls(all = TRUE), file= "all.rda")

# save session
save.image("session.RData")

# load dataset(s)
load("session.RData", envir = parent.frame(), verbose = FALSE)
```

- **save()** writes an external representation of objects to the specified file. The objects can be read back from the file at a later date by using the function **load**.
- **save.image()** is just a short-cut for save my current workspace,
i.e., save(list = ls(all.names = TRUE), file = ".RData", envir = .GlobalEnv).

It is good practice to create separate **R scripts** that you can use to **store sets of related functions**. You can then call those functions using the **source() function**, at the top of your script. R will then load those functions into memory and you can use them!

```
source("my_awesome_functions.R")
```

The **library()** and **require()** can be used to attach and **load add-on packages which are already installed**.

The **library**() by default returns an error if the requested package does not exist.

The **require**() is designed to be used **inside functions** as it gives a warning message and returns a logical value say, FALSE if the requested package is not found and TRUE if the package is loaded.

It is better to use the library() as it gives the error message if the package is not found during the package loading time. This will indeed avoid unnecessary headaches of tracking down the errors caused while attempting to use the library routines which are not installed.

What is the difference between the two R commands?

```
(A) brewer.pal.info["Blues",]
(B) RColorBrewer::brewer.pal.info["Blues",]
```

What is the difference between the two R commands?

```
# Use a particular function from a specific package
RColorBrewer::brewer.pal.info["Blues",]
# The function work without loading the packege
```

```
# The function only works if the package is installed and
loaded
library("RColorBrewer")
brewer.pal.info["Blues",]
```

```
## Packages needed
library(phyloseq)
library(microbiome)
library(randomForest)
library(ranger)
library(RColorBrewer)
library(tidyverse)
library(ggfortify)
library(simex)
library(DT)
library(ggplot2)
library(vegan)
library(dplyr)
library(magrittr)
library(scales)
library(grid)
library(reshape2)
library(knitr)
...
```

R packets should all be loaded (attached) at the beginning of the R script. However, loading a large number of packages can lead to problems (e.g. conflicts, memory)

What could be the problem and how can we work around it?

```
> library(tidyverse)
── Attaching packages ─────────────────────────────────────── tidyverse 1.3.1 ──
✓ tibble  3.1.4     ✓ dplyr   1.0.7
✓ tidyr   1.1.3     ✓ stringr 1.4.0
✓ readr   2.0.1     ✓ forcats 0.5.1
✓ purrr   0.3.4
── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
x microbiome::alpha()     masks ggplot2::alpha()
x dplyr::collapse()       masks IRanges::collapse()
x dplyr::combine()        masks randomForest::combine(), Biobase::combine(), BiocGenerics::combine()
x dplyr::count()          masks matrixStats::count()
x dplyr::desc()           masks IRanges::desc()
x tidyr::expand()         masks S4Vectors::expand(), Matrix::expand()
x tidyr::extract()        masks magrittr::extract()
x dplyr::filter()         masks stats::filter()
x dplyr::first()          masks S4Vectors::first()
x dplyr::lag()            masks stats::lag()
x randomForest::margin()  masks ggplot2::margin()
x tidyr::pack()           masks Matrix::pack()
x ggplot2::Position()     masks BiocGenerics::Position(), base::Position()
x purrr::reduce()         masks GenomicRanges::reduce(), IRanges::reduce()
x dplyr::rename()         masks S4Vectors::rename()
x purrr::set_names()      masks magrittr::set_names()
x dplyr::slice()          masks IRanges::slice()
x tidyr::unpack()         masks Matrix::unpack()
```

```
## Define Function
select     <- dplyr::select
alpha      <- ggplot::alpha()
filter     <- stats::filter()
```

Define the functions at the beginning of the script.

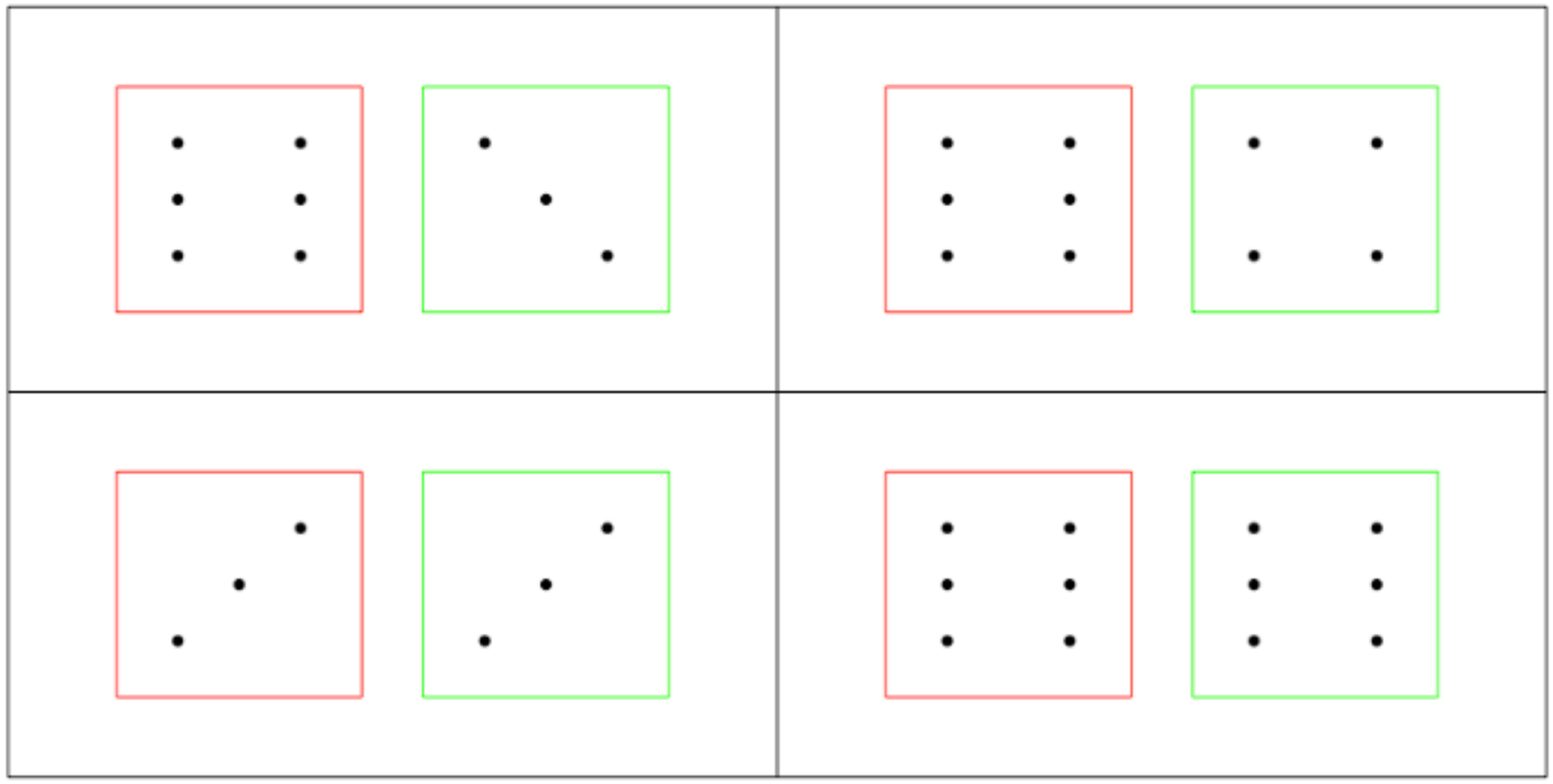
```
dplyr::select()
ggplot::alpha()
stats::filter()
```

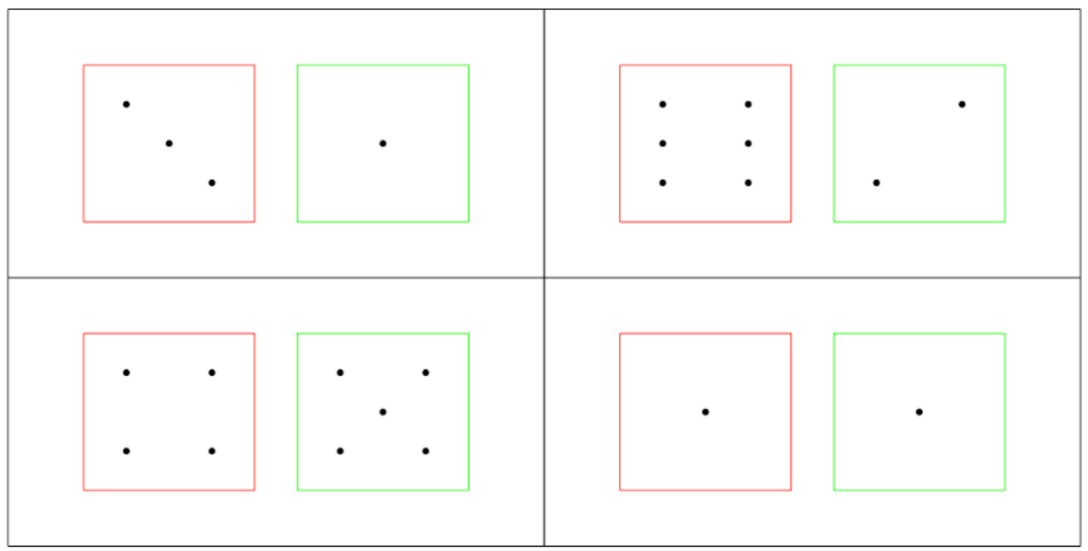
Specify which function you need when you use the function.

Both approaches are valuable. It is vital to understand which function should be utilised by which package, as well as ensuring that the required package is installed. Targeted function loading aids in conserving memory.

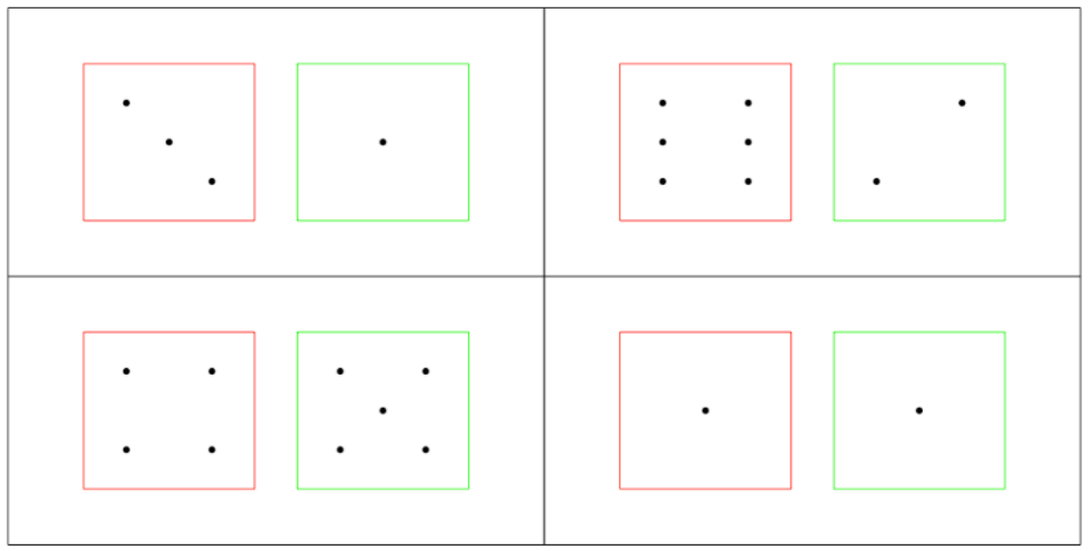
What can I do to make random objects reproducible in R?

```
TeachingDemos:dice(4, 2, plot.it = TRUE)
```





```
set.seed(123)
dice(4, 2, plot.it = TRUE)
```



```
set.seed(123)
dice(4, 2, plot.it = TRUE)
```

What can I do to make random objects reproducible (predictable) in R?

```
# Generate random number(s)
> rnorm(3)
# -0.2141846  0.2189062  0.2942180
> rnorm(3)
#  1.4912150 -1.4957131 -0.5814747
```

```
# Generate random number(s)
set.seed(191029); rnorm(3)
# -0.8421856 -0.4601290  0.9407093

# Next set using the same seed
set.seed(191029); rnorm(3)
# -0.8421856 -0.4601290  0.9407093
```

Nonparametric Resampling

bootstrap(data, mean)

```
Call:
bootstrap(data = CLEC, statistic = mean, seed = 19)
Replications: 10000

Summary Statistics:
     Observed       SE     Mean         Bias
mean 16.50913 3.96866 16.54073 0.03160109
```

bootstrap(data, mean)

```
Call:
bootstrap(data = CLEC, statistic = mean)
Replications: 10000

Summary Statistics:
     Observed       SE     Mean         Bias
mean 16.50913 3.985572 16.47043 -0.03869548
```

Nonparametric Resampling

```
bootstrap(data, mean, seed = 191029)

Call:
bootstrap(data = CLEC, statistic = mean, seed = 191029)
Replications: 10000

Summary Statistics:
     Observed        SE     Mean         Bias
mean 16.50913 3.957816 16.47054 -0.03859261
```


```
bootstrap(data, mean, seed = 191029)

Call:
bootstrap(data = CLEC, statistic = mean, seed = 191029)
Replications: 10000

Summary Statistics:
     Observed        SE     Mean         Bias
mean 16.50913 3.957816 16.47054 -0.03859261
```

Profiling in R

```
x <- matrix( rnorm( 50000 * 900),
        nrow = 50000,
        ncol =   900)

t(x) %*% x    # version A

crossprod(x) # version B
```

```
system.time(t(x) %*% x)
   user  system elapsed
 27.053   0.148  27.277
```

```
system.time(crossprod(x))
   user  system elapsed
 21.166   0.020  21.209
```

Δ 6.07s

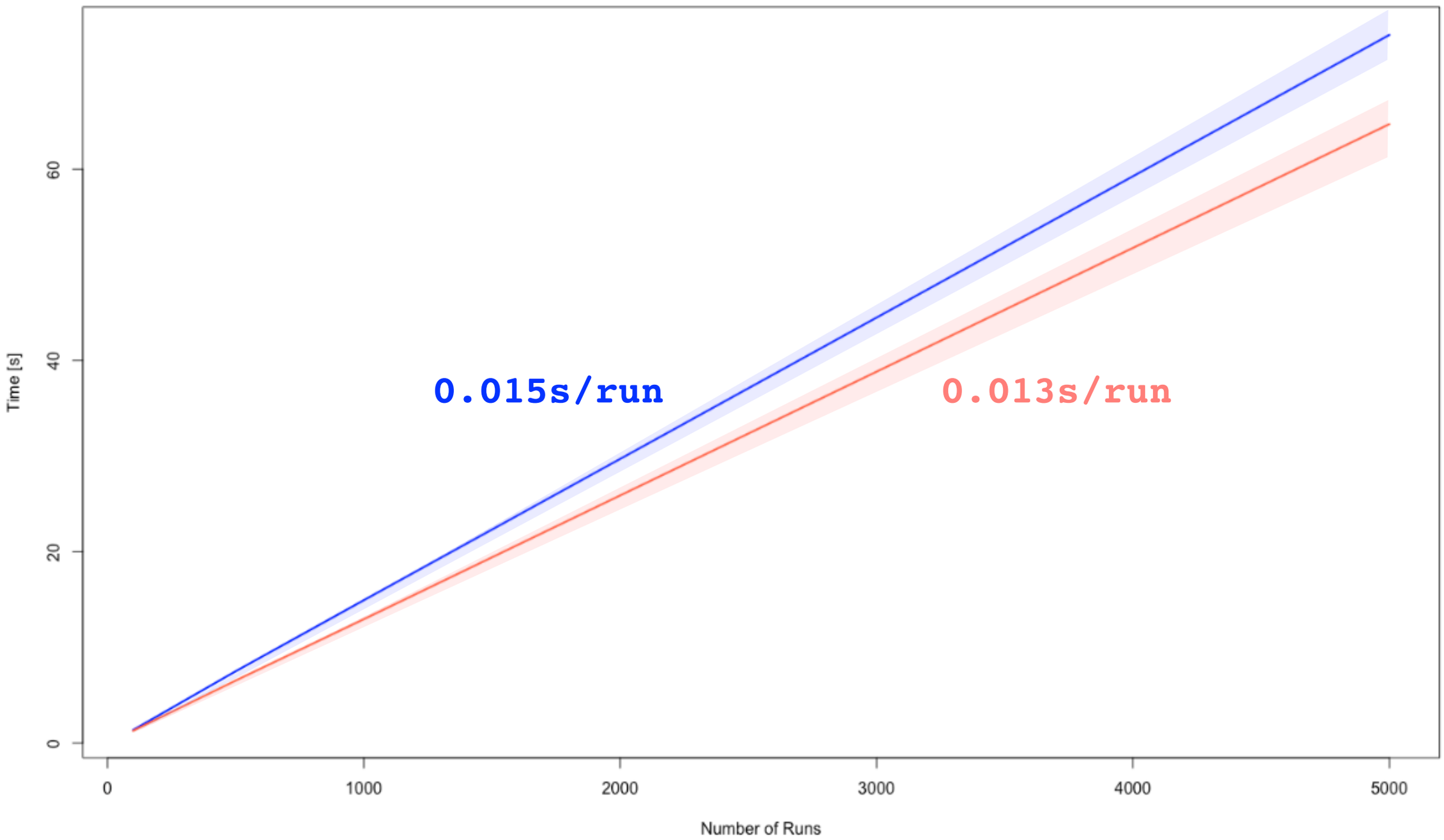
```
ex1 <- function(n) {
  for(i in 1:n) {
  x <- mean(rt(100000, df = 4))
  cat(paste("...", i, sep = ""))
  }
  print("END")
}

ex2 <- function(n) {
  for(i in 1:n) {
    d <- rt(100000, df = 4)
    x <- mean(d)
    cat(paste("...", i, sep = ""))
  }
  print("END")
}

system.time(ex1(500))
system.time(ex2(500))
```

```
n=100: -0.014s
n=200: -0.056s
n=300: -0.073s
n=400: -0.083s
n=500: -0.157s
```

```
ex1 <- function(n) {
  for(i in 1:n) {
  x <- mean(rt(100000, df = 4))
  cat(paste("...", i, sep = ""))
  }
  print("END")
}
```

```
ex3 <- function(n) {
  for(i in 1:n) {
    x <- mean(rt(100000, df = 4))
  }
}

system.time(ex1(500))
system.time(ex3(500))
```

```
n =   100: -0.069s
n =   500: -0.649s
n = 1000: -1.771s
```


Time [s]
0
20
40
60
0.015s/run
0.013s/run
0
1000
2000
3000
4000
5000
Number of Runs

Task: Write a function for the two solutions below and compare the execution time.

```
# Solution A (loop)
n <- 10^8
A <- 0
for (j in c(1:n)) {A <- j + A}
A
```

```
# Solution B
n <- 10^8
sum(1:n)
```

```
ex1 <- function(n) {
  A <- 0
  for (j in c(1:n)) {A <- j + A}
  print(A)
}

ex2 <- function(n) {
  print(sum(1:n))
}

system.time(ex1(10^8)); system.time(ex2(10^8))
```

```
[1] 5e+15
   user  system elapsed
  2.237   0.203   2.447
[1] 5e+15
   user  system elapsed
  0.001   0.000   0.000
```


tidyverse
www.rstudio.com

The **tidyverse** is a language for solving data science challenges with R. Its is a collection of R packages that share a high-level design philosophy and low-level grammar and data structures, so that learning one package makes it easier to learn the next.

There are four basic principles to a tidy API:

- Reuse existing data structures.
- Compose simple functions with the pipe.
- Embrace functional programming.
- Design for humans.

Visualization is a corner stone of both exploratory analysis and science communication.

- Any data set can be transformed into one (or a few) tidy data tables.
- Any data set in a tidy data table can be visualised.

Tidy data format: every column is a variable, every row is an observation, and every cell contains a single value.

Data Import

```
library(readr) # core package
```

```
data <- read_csv("data.csv",
  col_names = c("ID","SampleName", "Loction"),
  skip = 1)
```

```
data <- read_excel("data.xlsx",
  col_names = c("ID","SampleName", "Loction"),
  skip = 1)
```

Combine Data Files

```
map(list.files(pattern = "*.csv"), read_csv)
```

```
map_df(list.files(pattern = "*.csv"), read_csv)
```

The **map_df** function will work with different column names and data types. Data gets matched by column name and mis-match will create new columns. Mixed datatypes will be adjusted in the “best” (safest) format.

Compose functions with the **pipe**

`[shift]+[command]+m`

`%>%`

Instead of using nested or muliple consecutive functions you link them together like pieces of plumbing so that your data passes through them and changes as it goes.

Nested Functions

```
mean(iris$Sepal.Length[iris$Species == "setosa"] *
iris$Sepal.Width[iris$Species == "setosa"])
```

Consecutive Functions

```
is_setosa    <- iris$Species == "setosa"
setosa       <- iris[is_setosa, ]
setosa_area <- setosa$Sepal.Length * setosa$Sepal.Width
mean(setosa_area)
```

dplyr with pipes (%>%)

```
iris %>%
  filter(Species == "setosa") %>%
  mutate(Sepal.Area = Sepal.Length * Sepal.Width) %>%
  summarize(Mean.Sepal.Area = mean(Sepal.Area))
```

```
iris %>%
  group_by(Species) %>%
  mutate(Sepal.Area = Sepal.Length * Sepal.Width) %>%
  summarize(Area.Sepal.Mean = mean(Sepal.Area))
```

```
iris %>%
  group_by(Species) %>%
  mutate(Sepal.Area = Sepal.Length * Sepal.Width,
         Petal.Area = Petal.Length * Petal.Width) %>%
  select(c(Sepal.Length, Sepal.Width, Sepal.Area,
Petal.Length, Petal.Width, Petal.Area, Species))
```

```
head(iris, n = 5)
```

```
library(magrittr)
library(tidyverse)
```

```
iris %>%
  head(n=5) -> iris_top5
```

```
length(toupper(letters))
```

```
letters %>%
  toupper() %>%
  length()
```

```
letters %>%
  toupper %>%
  length
```

```
LETTERS %>%
  length
```

```
sample(letters, size = 10, replace = TRUE)
```

```
letters %>%
  sample(., size = 10, replace = TRUE)
```

```
sample(toupper(letters), size = 10, replace
= TRUE)
```

```
letters %>%
  toupper() %>%
  sample(., size = 10, replace = TRUE)
```

```
sort(toupper(names(mtcars)))
```

```
mtcars %>%
  names() %>%
  toupper() %>%
  sort
```

select

```
iris %>%
    select(Species)
```

```
iris %>%
    select(-Species)
```

```
iris %>%
    select(c(1:3,5))
```

```
iris %>%
  select(-Species, Species) %>%
  head()
```

select > helper functions

```
iris %>%
select(contains("Spe") %>%
head(., n = 3)
```

select_if > content based selection

```
iris %>%
select_if(is.numeric)
```

```
iris %>%
select_if(~is.numeric(.) &
n_distinct(.)>30)
```

~ custom condition: checks if the column is numeric and if the number of unique values in the column is more than 30

as_mapper > repeated selection

```
is_num_b30 <- as_mapper(~is.numeric(.) &
n_distinct(.) > 30)
is_num_b30(1:30)
is_num_b30(1:31)
```

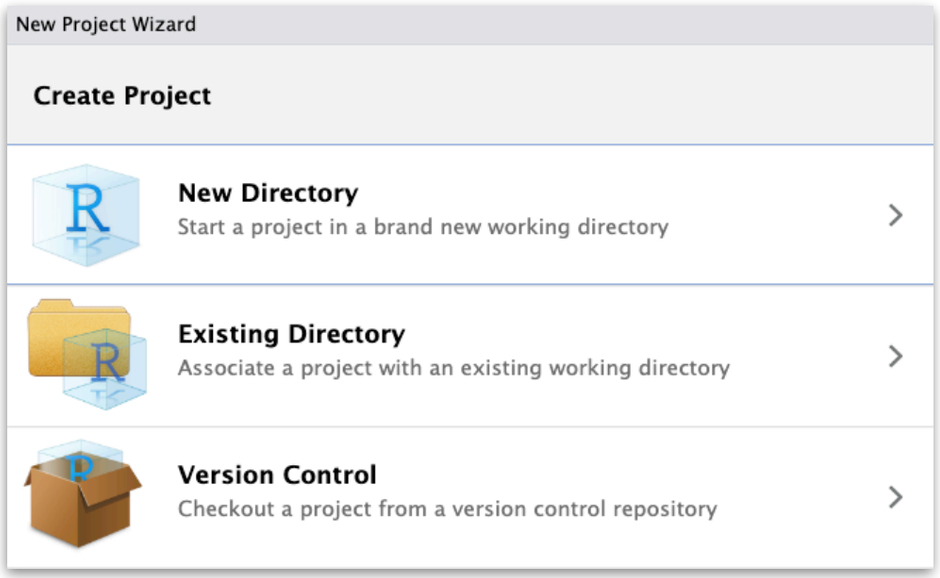
Aggregate functions

n() to return the number of rows

n_distinct() to return the number of unique values in a column

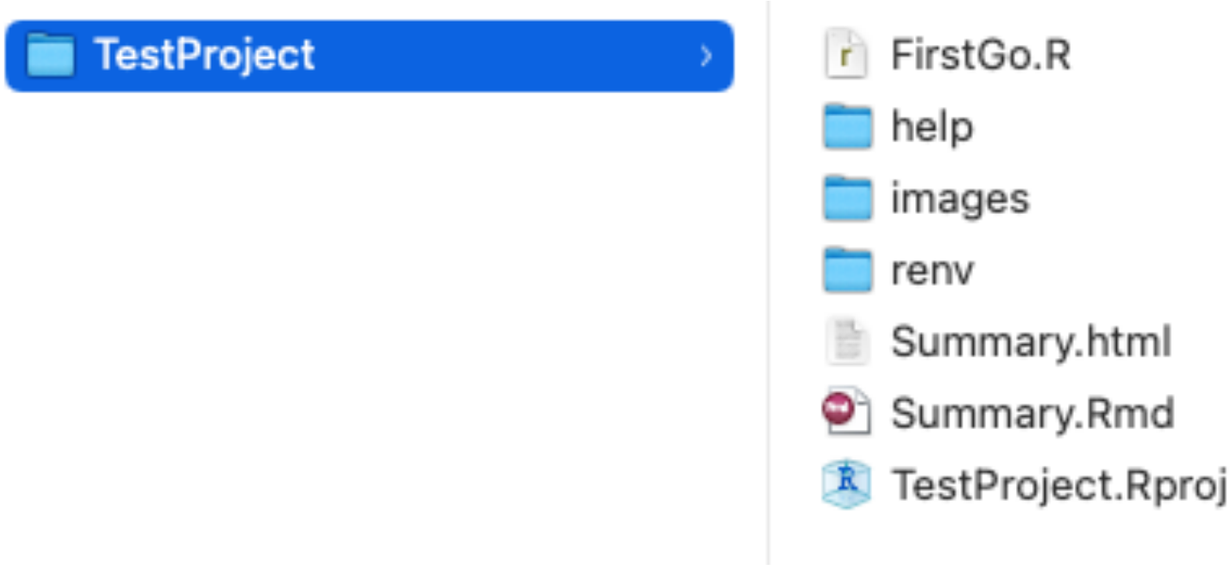
first(), **last**(), and **nth**() for retrieving values in specific positions in a column

RStudio **Projects**



An RStudio Project is **a way to organize your work** in the RStudio integrated development environment (IDE). It is a directory on your computer that contains R scripts, data files, and other resources related to a specific data analysis or programming task.

RStudio Projects



RStudio Project is a way to organize your R-related work, manage dependencies, and maintain a clean and structured workflow, making it easier to collaborate and ensure reproducibility in your data analysis and programming tasks.

RStudio **Snippets**

In RStudio, "snippets" are small, **reusable pieces of code or text t**hat can be quickly i**nserted into your R scripts** or documents using a simple keyword or keyboard shortcut. Snippets are a handy feature that helps you save time and reduce repetitive typing by allowing you to insert predefined code templates, common text blocks, or placeholders for variable names, function arguments, and other elements.In short, snippets are a great way to automate the insertion of commonly used code.

RStudio **Snippets**

Tools -> Global Options -> Code -> Tab Editing -> Snippets -> "Edit Snippets"

```
snippet header
  ##
  ## Title: ${1}
  ##
  ## Author: Jean-Claude Walser
  ##
  ## Date Created: `r paste(Sys.Date())`
  ##
  ## Copyright (c) J-CW, `r paste(format(Sys.Date(), "%Y"))`
  ## Email: jean-claude.walser--usys.ethz.ch
  ##
  ## ------------------------------------------------------
  ## Notes:
  ##
  ## ------------------------------------------------------

  ## Clean/reset environment
  rm(list = ls())

snippet sende
  ## Session-Log
  sink("SessionInfo.txt")
  sessionInfo()
  sink()
```

The contents of the snippet should be indented below using the Tab key (rather than with spaces).

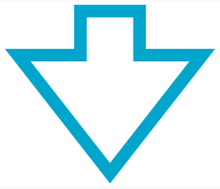
RStudio **Addins**

R addins are a powerful feature of RStudio that allow you to **extend its functionality**, streamline your workflow, and create custom tools and actions to enhance your R programming and data analysis experience. In short, addins are R functions with some special registration metadata.

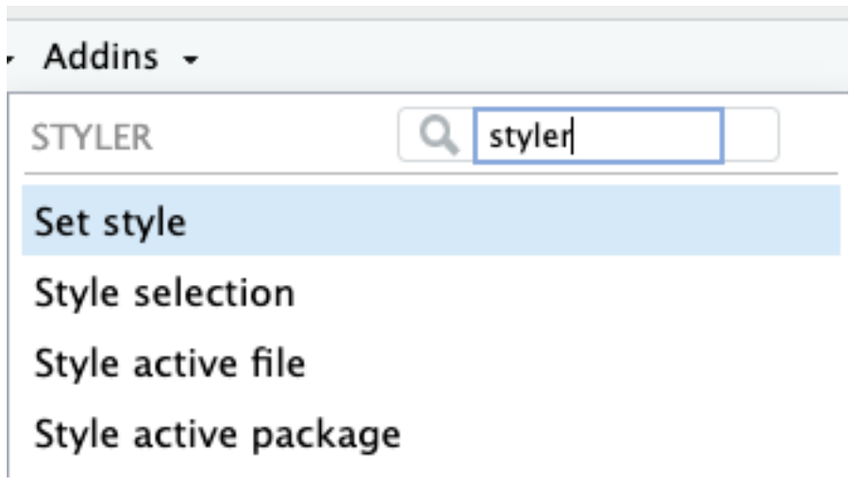
Package: styler (addins)

Styler formats your code according to the Tidyverse style guide (or your own custom style guide) so you can focus on the content of your code. It helps keep coding style consistent across projects and makes collaboration easier. You can access the Styler from

```
a=3; if(a>4){message<-"big"}else{message<-"small"}; paste("a is ", message)
```



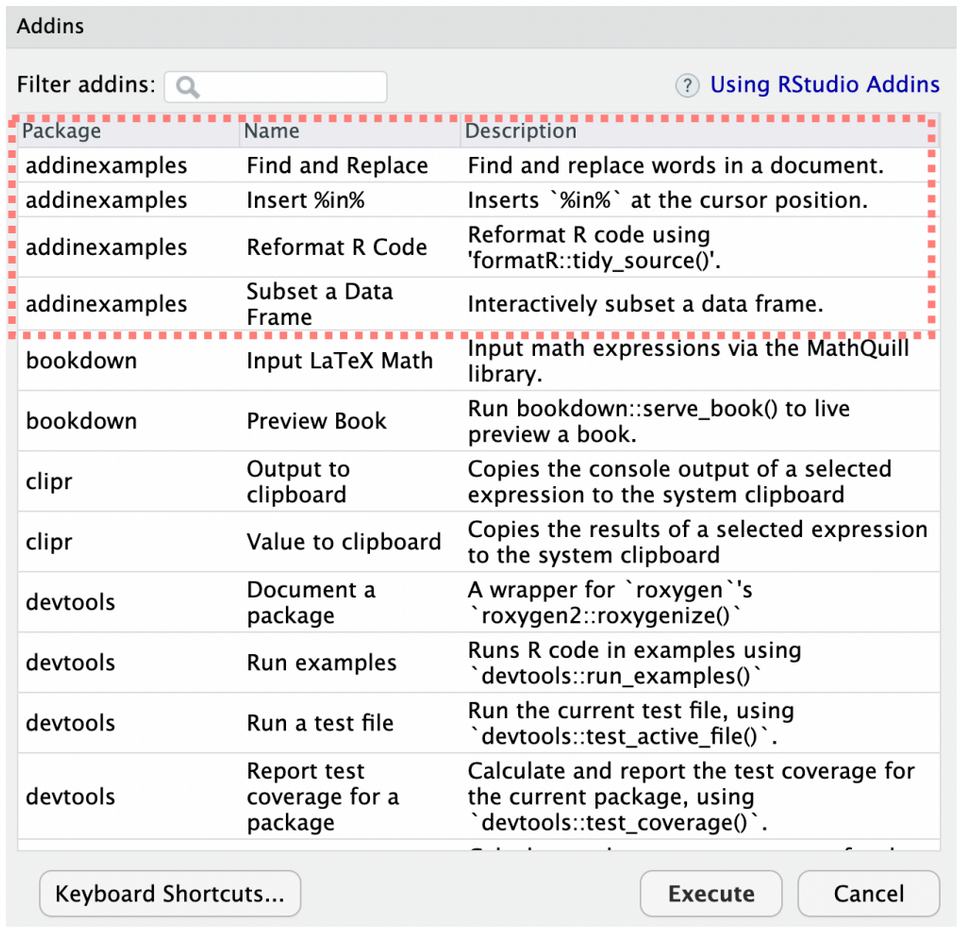
```
a <- 3
if (a > 4) {
  # First option
  message <- "big"
} else {
  # Second
  message <- "small"
}
paste("a is ", message)
```

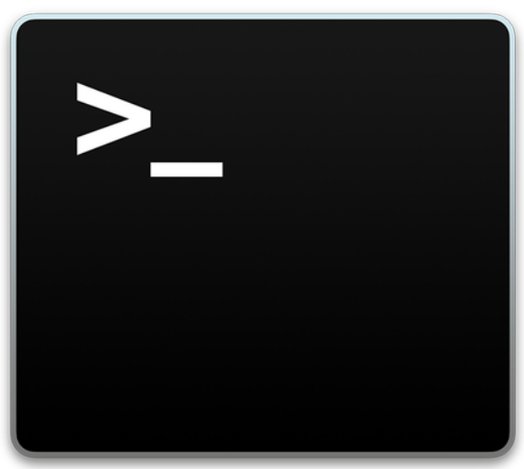


RStudio Addins Example

```
devtools::install_github("rstudio/addinexamples", type = "source")
```

[Tools] > [Addins] > [Browse Addins…]





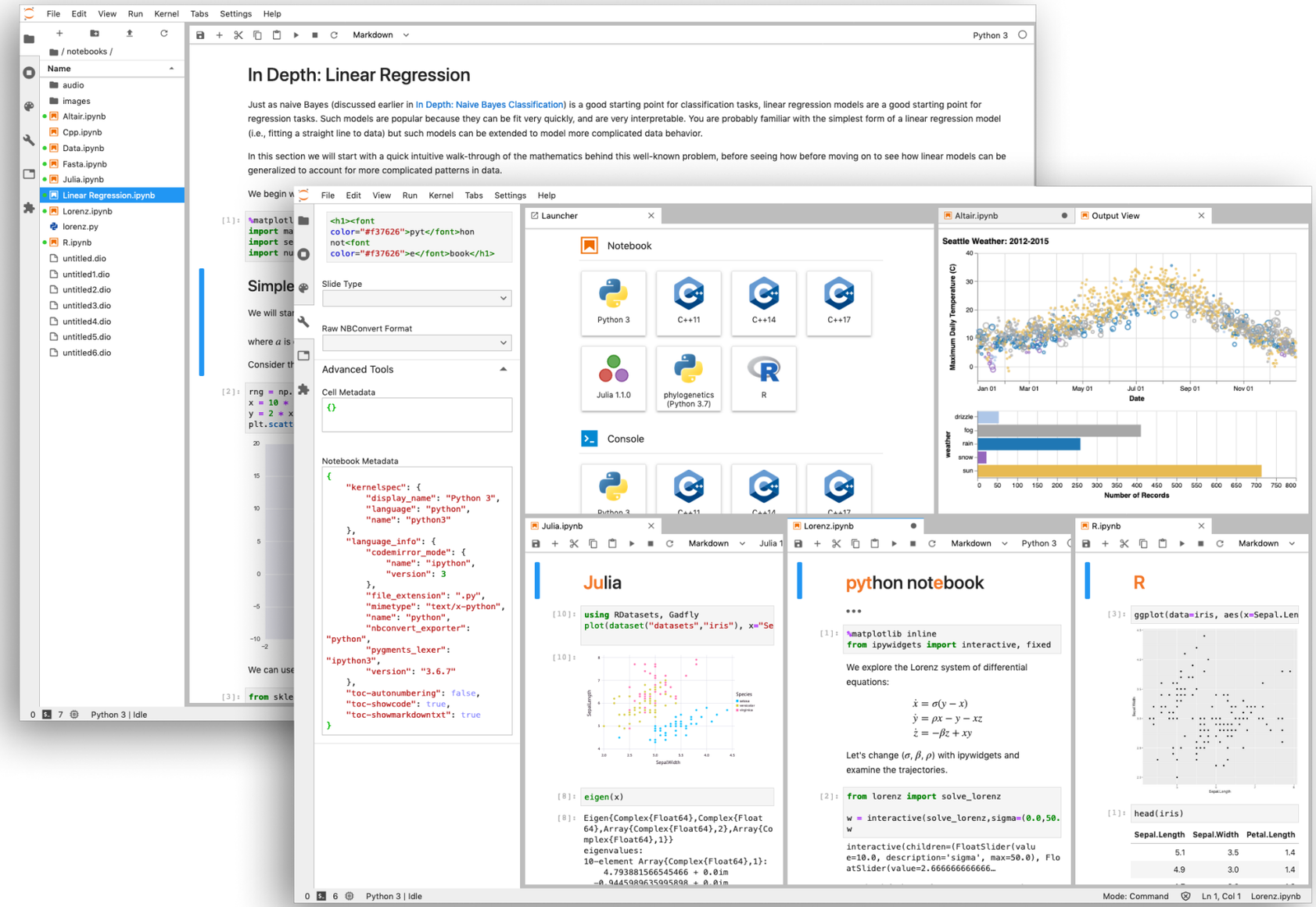
Rscript

```
#!/usr/bin/env Rscript
# usage: Rscript Boxplot.R 'table.txt'

## Define Import
args = commandArgs(trailingOnly = TRUE)

## Load data
d <- read.table(args[1], sep = ";", header = TRUE)

pdf(OUT, paper = "a4")
  boxplot(d$length ~ d$groups, names = d$groups)
dev.off()

## END
print(paste(date(), "Boxplot finished", sep = " "))
```

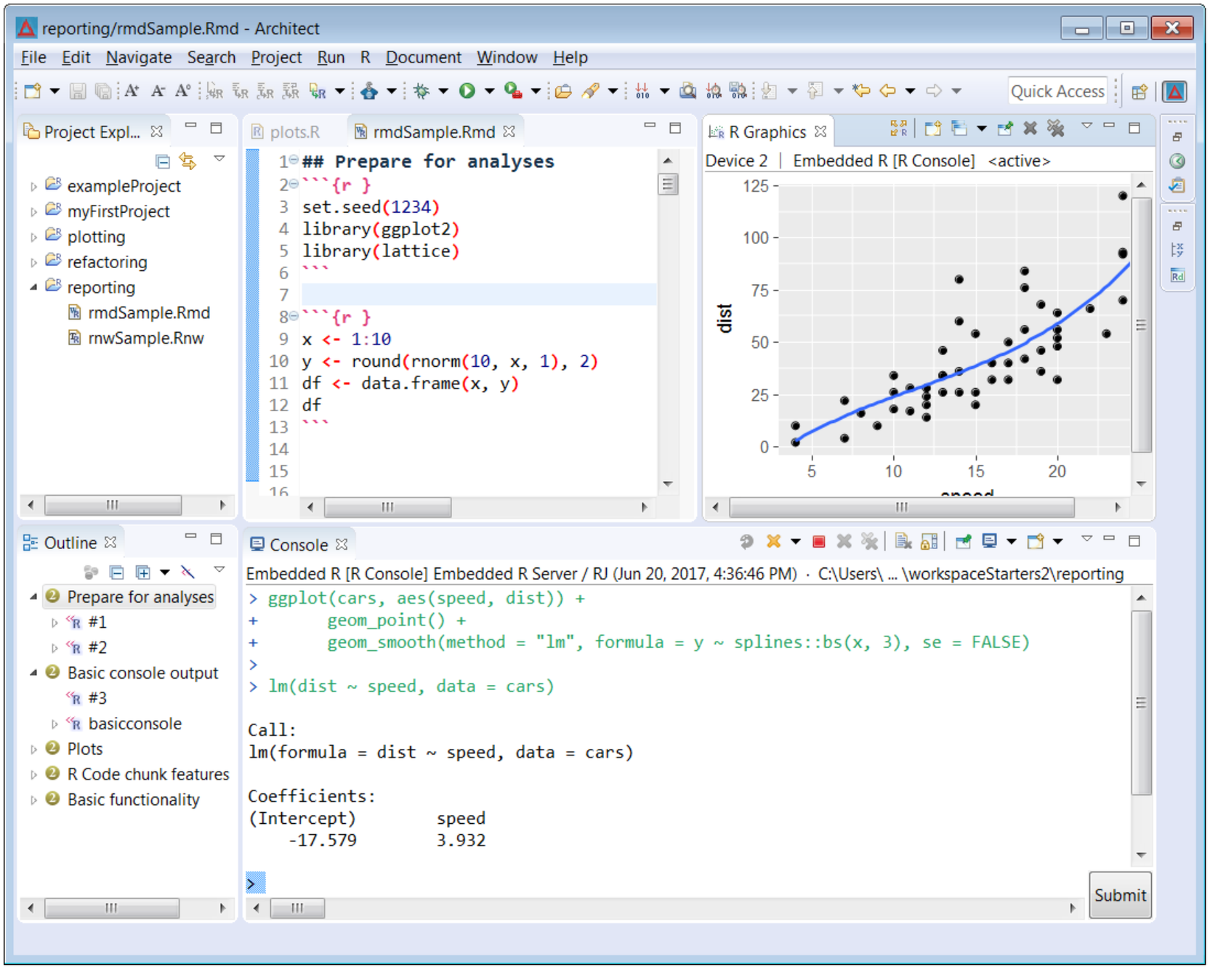
IDE

An integrated development environment (**IDE**) is a software application that provides comprehensive facilities to computer programmers for software development.

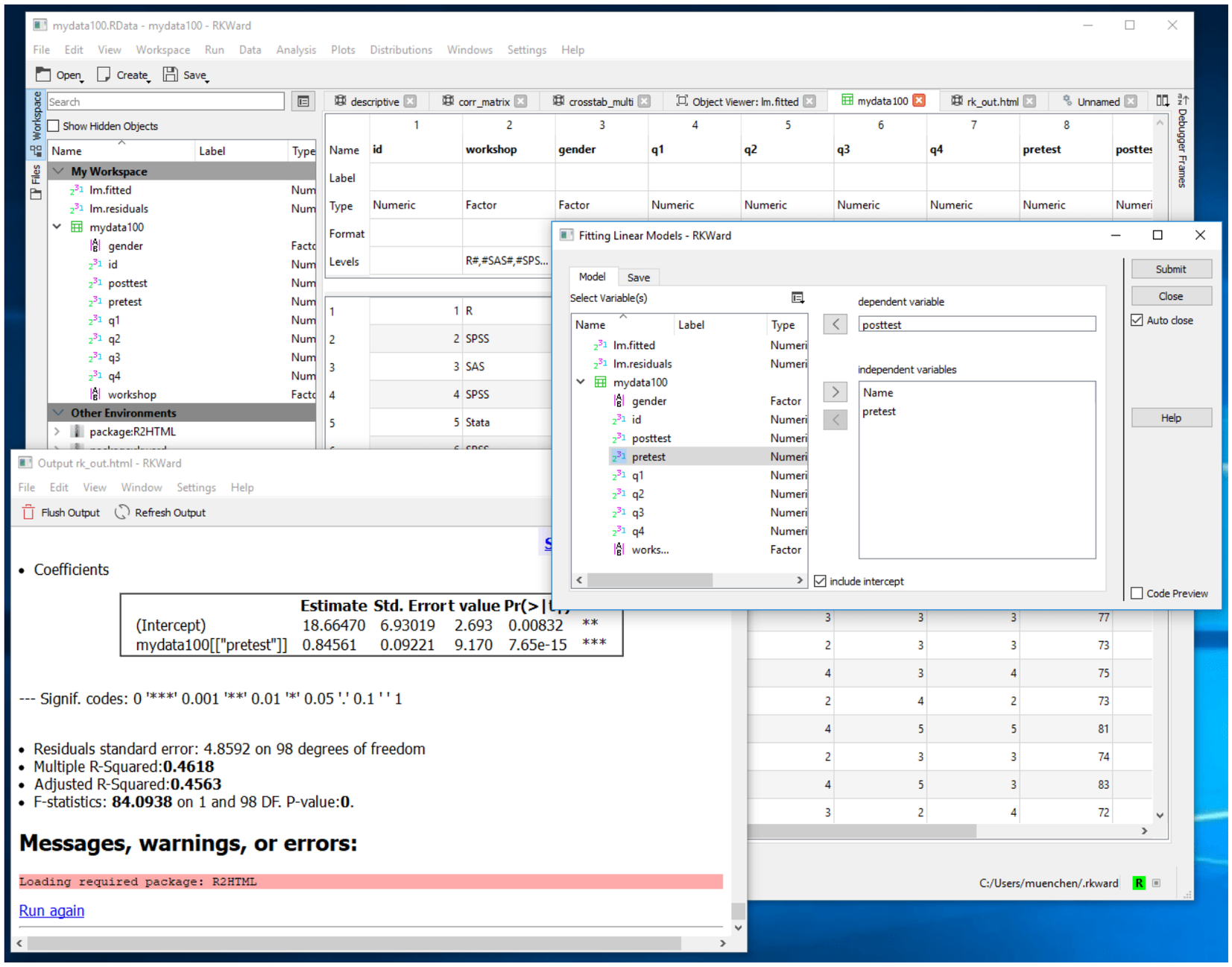
JupyterLab


File Edit View Run Kernel Tabs Settings Help
/ notebooks /
Name
audio
images
Altair.ipynb
Cpp.ipynb
Data.ipynb
Fasta.ipynb
Julia.ipynb
Linear Regression.ipynb
Lorenz.ipynb
lorenz.py
R.ipynb
untitled.dio
untitled1.dio
untitled2.dio
untitled3.dio
untitled4.dio
untitled5.dio
untitled6.dio
Markdown
Python 3
In Depth: Linear Regression
Just as naive Bayes (discussed earlier in In Depth: Naive Bayes Classification) is a good starting point for classification tasks, linear regression models are a good starting point for regression tasks. Such models are popular because they can be fit very quickly, and are very interpretable. You are probably familiar with the simplest form of a linear regression model (i.e., fitting a straight line to data) but such models can be extended to model more complicated data behavior.
In this section we will start with a quick intuitive walk-through of the mathematics behind this well-known problem, before seeing how before moving on to see how linear models can be generalized to account for more complicated patterns in data.
Launcher
Notebook
Python 3
C++11
C++14
C++17
Julia 1.1.0
phylogenetics (Python 3.7)
R
Console
Altair.ipynb
Output View
Seattle Weather: 2012-2015
Maximum Daily Temperature (C)
Date
weather
Number of Records
Slide Type
Raw NBConvert Format
Advanced Tools
Cell Metadata
Notebook Metadata
Julia.ipynb
Julia
Lorenz.ipynb
python notebook
R.ipynb
R
Python 3 | Idle
Mode: Command
Ln 1, Col 1
Lorenz.ipynb

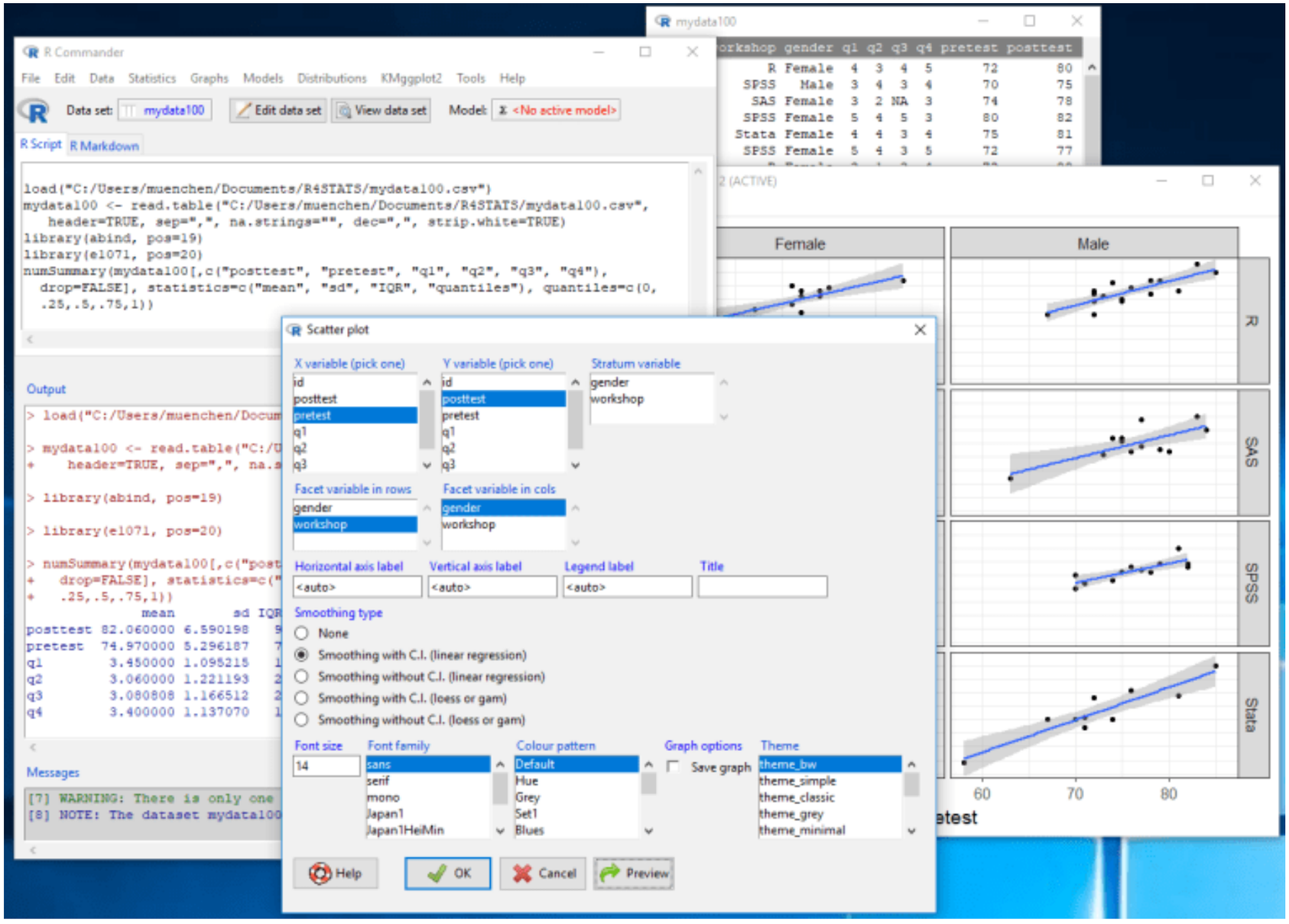
ARCHITECT


reporting/rmdSample.Rmd - Architect
File Edit Navigate Search Project Run R Document Window Help
Quick Access
Project Expl...
exampleProject
myFirstProject
plotting
refactoring
reporting
rmdSample.Rmd
rnwSample.Rnw
plots.R
rmdSample.Rmd
1 ## Prepare for analyses
2 ```{r }
3 set.seed(1234)
4 library(ggplot2)
5 library(lattice)
6 ```
7
8 ```{r }
9 x <- 1:10
10 y <- round(rnorm(10, x, 1), 2)
11 df <- data.frame(x, y)
12 df
13 ```
14
15
16
R Graphics
Device 2 | Embedded R [R Console] <active>
dist
125
100
75
50
25
0
5
10
15
20
Outline
Prepare for analyses
#1
#2
Basic console output
#3
basicconsole
Plots
R Code chunk features
Basic functionality
Console
Embedded R [R Console] Embedded R Server / RJ (Jun 20, 2017, 4:36:46 PM) · C:\Users\ ... \workspaceStarters2\reporting
> ggplot(cars, aes(speed, dist)) +
+ geom_point() +
+ geom_smooth(method = "lm", formula = y ~ splines::bs(x, 3), se = FALSE)
>
> lm(dist ~ speed, data = cars)
Call:
lm(formula = dist ~ speed, data = cars)
Coefficients:
(Intercept) speed
-17.579 3.932
>
Submit

RKWard


mydata100.RData - mydata100 - RKWard
File Edit View Workspace Run Data Analysis Plots Distributions Windows Settings Help
Open Create Save
Search
Show Hidden Objects
Name Label Type
My Workspace
lm.fitted
lm.residuals
mydata100
gender
id
posttest
pretest
q1
q2
q3
q4
workshop
Other Environments
package:R2HTML
descriptive
corr_matrix
crosstab_multi
Object Viewer: lm.fitted
mydata100
rk_out.html
Unnamed
Debugger Frames
Fitting Linear Models - RKWard
Model
Save
Select Variable(s)
dependent variable
posttest
independent variables
Name
pretest
include intercept
Submit
Close
Auto close
Help
Code Preview
Output rk_out.html - RKWard
File Edit View Window Settings Help
Flush Output
Refresh Output
Coefficients
Estimate Std. Error t value Pr(>|t|)
(Intercept) 18.66470 6.93019 2.693 0.00832 **
mydata100[["pretest"]] 0.84561 0.09221 9.170 7.65e-15 ***
--- Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Residuals standard error: 4.8592 on 98 degrees of freedom
Multiple R-Squared:0.4618
Adjusted R-Squared:0.4563
F-statistics: 84.0938 on 1 and 98 DF. P-value:0.
Messages, warnings, or errors:
Loading required package: R2HTML
Run again
C:/Users/muenchen/.rkward

R Commander


R Commander
File Edit Data Statistics Graphs Models Distributions KMggplot2 Tools Help
Data set: mydata100
Edit data set
View data set
Model: <No active model>
R Script
R Markdown
load("C:/Users/muenchen/Documents/R4STATS/mydata100.csv")
mydata100 <- read.table("C:/Users/muenchen/Documents/R4STATS/mydata100.csv",
header=TRUE, sep=",", na.strings="", dec=".", strip.white=TRUE)
library(abind, pos=19)
library(e1071, pos=20)
numSummary(mydata100[,c("posttest", "pretest", "q1", "q2", "q3", "q4"),
drop=FALSE], statistics=c("mean", "sd", "IQR", "quantiles"), quantiles=c(0,
.25,.5,.75,1))
Output
Messages
mydata100
orkshop gender q1 q2 q3 q4 pretest posttest
R Female 4 3 4 5 72 80
SPSS Male 3 4 3 4 70 75
SAS Female 3 2 NA 3 74 78
SPSS Female 5 4 5 3 80 82
Stata Female 4 4 3 4 75 81
SPSS Female 5 4 3 5 72 77
2 (ACTIVE)
Female
Male
SAS
SPSS
Stata
60
70
80
Scatter plot
X variable (pick one)
Y variable (pick one)
Stratum variable
id
posttest
pretest
q1
q2
q3
gender
workshop
Facet variable in rows
Facet variable in cols
Horizontal axis label
Vertical axis label
Legend label
Title
<auto>
Smoothing type
None
Smoothing with C.I. (linear regression)
Smoothing without C.I. (linear regression)
Smoothing with C.I. (loess or gam)
Smoothing without C.I. (loess or gam)
Font size
Font family
Colour pattern
Graph options
Save graph
Theme
14
sans
serif
mono
Japan1
Japan1HeiMin
Default
Hue
Grey
Set1
Blues
theme_bw
theme_simple
theme_classic
theme_grey
theme_minimal
Help
OK
Cancel
Preview

Rbox: an integrated R package for ATOM Editor

Saeid Amiri[1]

Department of Natural and Applied Sciences, University of Wisconsin-Green Bay, Green Bay, WI, USA

Abstract

R is a programming language and environment that is a central tool in the applied sciences for writing program. Its impact on the development of modern statistics is inevitable. Current research, especially for big data may not be done solely using R and will likely use different programming languages; hence, having a modern integrated development environment (IDE) is very important. Atom editor is modern IDE that is developed by GitHub, it is described as "A hackable text editor for the 21st Century". This report is intended to present a package deployed entitled *Rbox* that allows Atom Editor to write and run codes professionally in R.

Keywords: programming; IDE; Jupyter; JavaScript; Web- based interface.

AnalyticFlow