Introduction to basic R

Abu Bakar Siddique

Bioinformatician, SLUBI

Contents

Basic about R

R is ...

- a programming language
- a programming platform (= environment + interpreter)
- a software project driven by the core team and the community
- a very powerful tool for statistical computing
- a very powerful computational tool in general

Yet ...

- it is very elegant
- it becomes more and more feature-rich

R is not ...

- a tool to replace a statistician
- the very best programming language
- the most elegant programming solution
- the most efficient programming language

R: Engine	RStudio: Dashboard

R

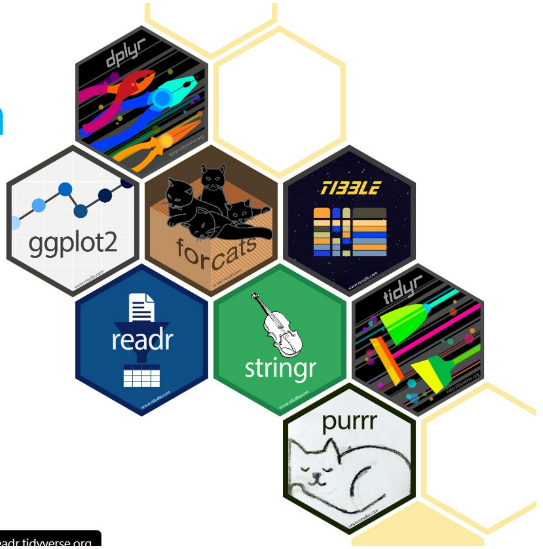
- Programming language
- For data analysis and graphics
- Refers to both the language and the software that interprets it’s scripts
- Free and open source

RStudio

- User interface for working with R
- Wrapper around the R language
- Extends what R can do and facilitates writing R code
- Free and open source

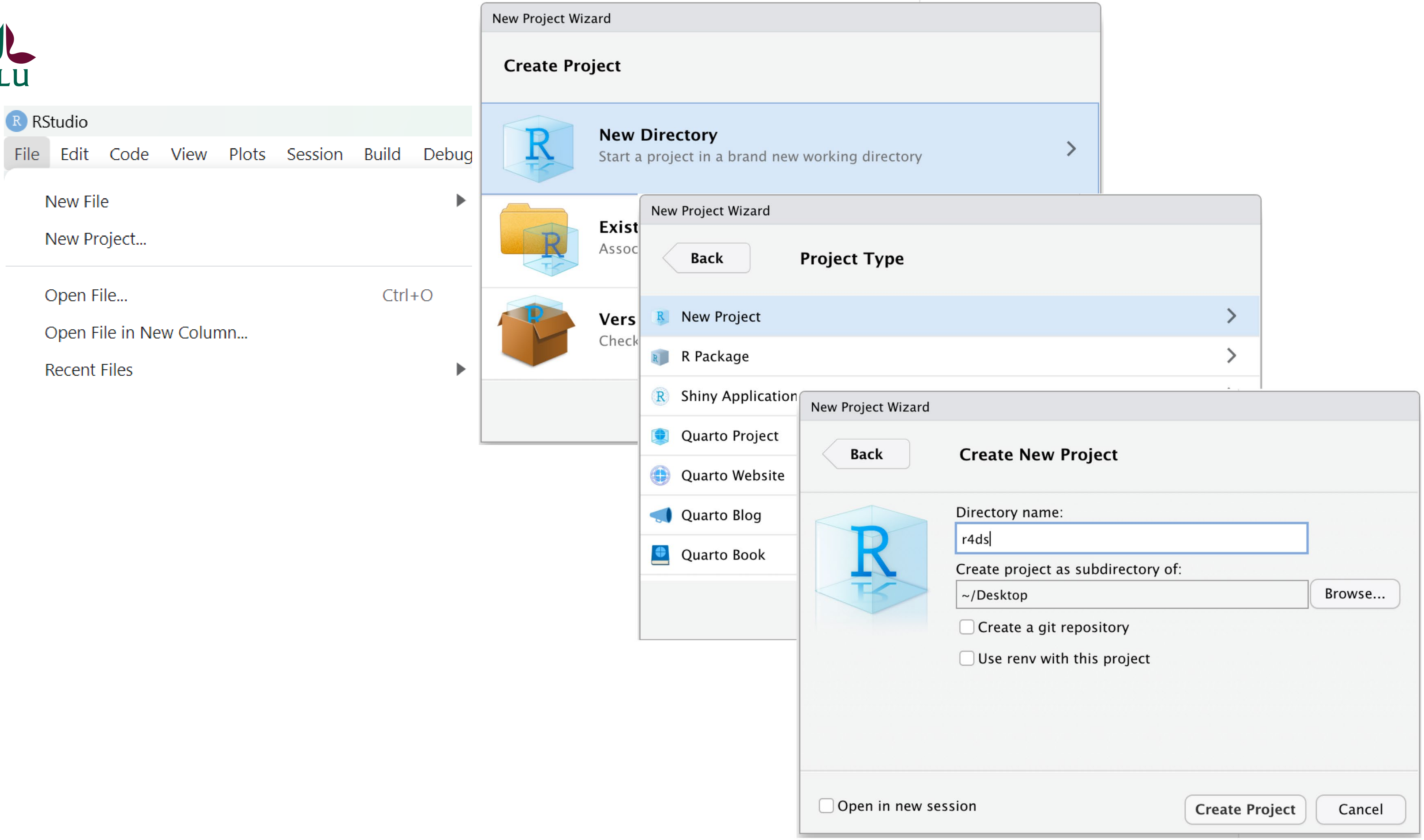
Timeline

- ca. 1992 — conceived by Robert Gentleman and Ross Ihaka (R&R) at the University of Auckland, NZ as a tool for **teaching statistics**
- 1994 — initial version
- 2000 — stable version
- 2011 — RStudio, first release by J.J. Allaire
- ca. 2017 — Tidyverse by Hadley Wickham



s://readr.tidyverse.org

Setting up a project

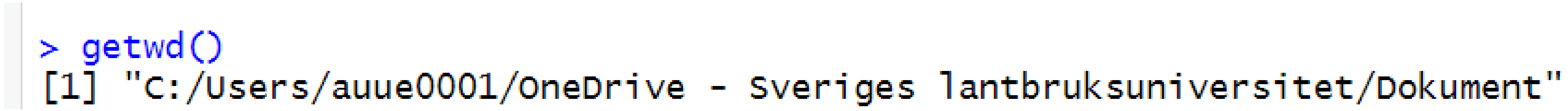


Directory name: r4ds

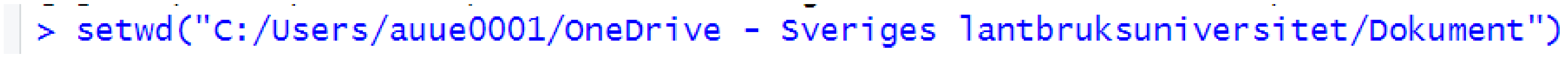
Create project as subdirectory of: ~/Desktop

The working directory

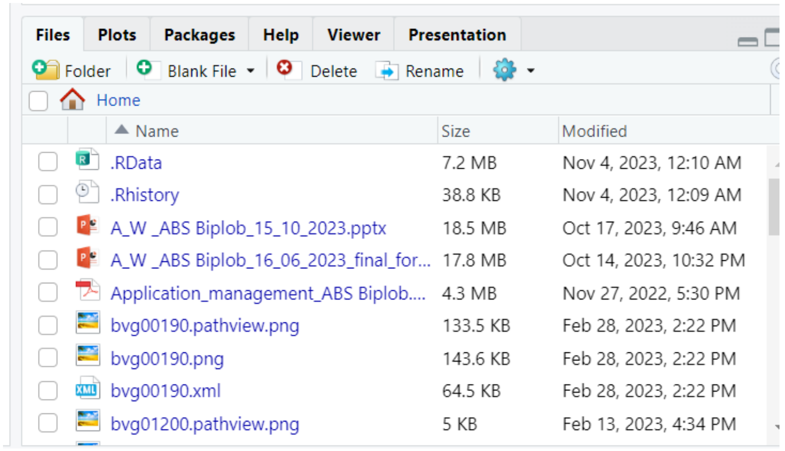
- Where R will look for and save files
- Check working directory with the getwd() , setwd() functions



```
> getwd()
[1] "C:/Users/auue0001/OneDrive - Sveriges lantbruksuniversitet/Dokument"
```



```
> setwd("C:/Users/auue0001/OneDrive - Sveriges lantbruksuniversitet/Dokument")
```



Name	Size	Modified
.RData	7.2 MB	Nov 4, 2023, 12:10 AM
.Rhistory	38.8 KB	Nov 4, 2023, 12:09 AM
A_W _ABS Biplob_15_10_2023.pptx	18.5 MB	Oct 17, 2023, 9:46 AM
A_W _ABS Biplob_16_06_2023_final_for...	17.8 MB	Oct 14, 2023, 10:32 PM
Application_management_ABS Biplob....	4.3 MB	Nov 27, 2022, 5:30 PM
bvg00190.pathview.png	133.5 KB	Feb 28, 2023, 2:22 PM
bvg00190.png	143.6 KB	Feb 28, 2023, 2:22 PM
bvg00190.xml	64.5 KB	Feb 28, 2023, 2:22 PM
bvg01200.pathview.png	5 KB	Feb 13, 2023, 4:34 PM

Suggested subdirectories

• data/	for raw data and intermediate datasets
• data_output/	modified versions of raw data
• documents/	outlines, drafts, other text
• fig_output/	graphics generated by scripts
• scripts/	R scripts for different analyses or plotting

```
# create subdirectory
dir.create(path = “path_to_working_directory/data_output")
```

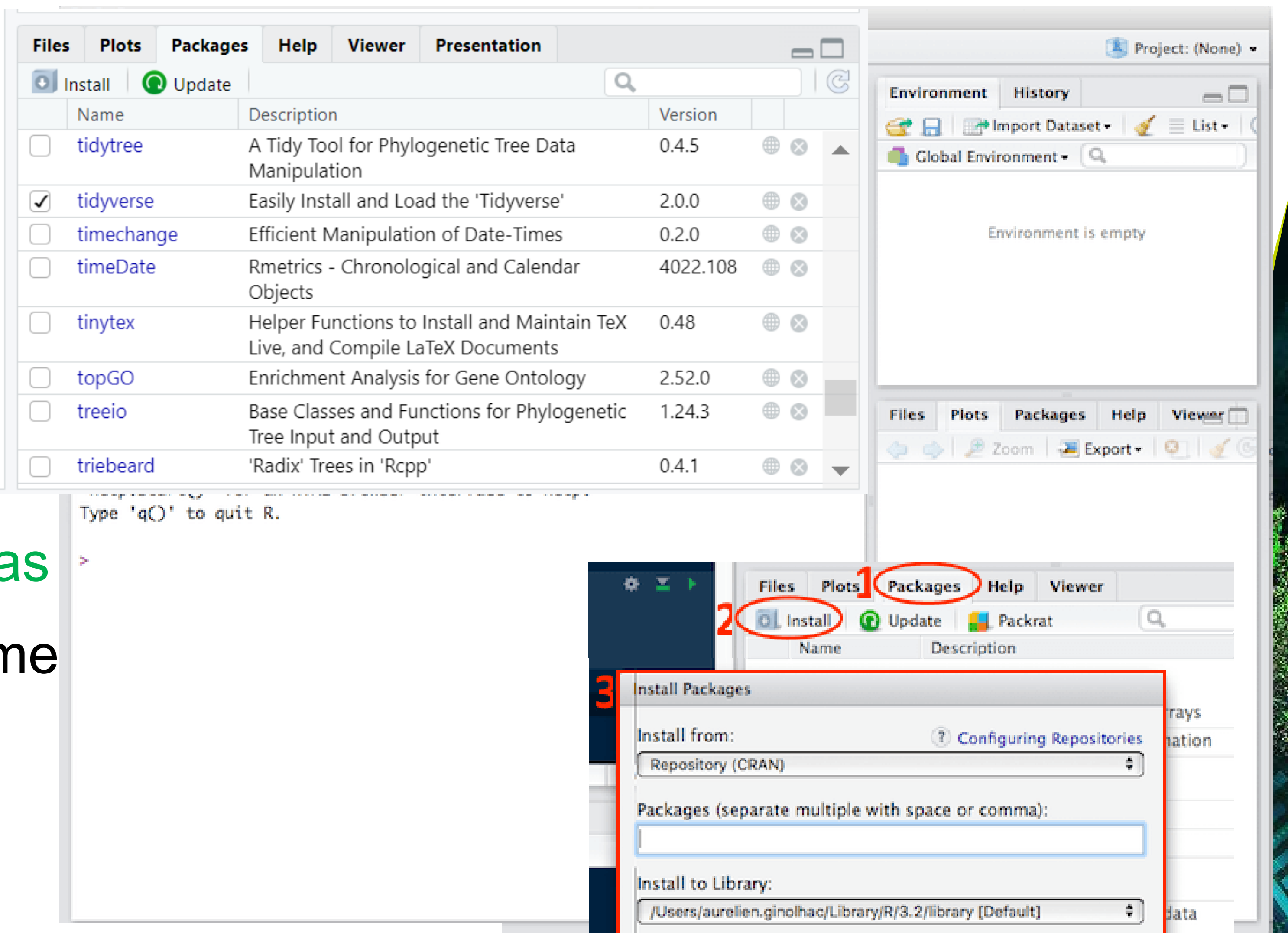
Interacting with R

Packages

- developed by the community
- cover several very diverse areas of science/life
- uniformly structured and documented
- organized in repositories:
  - CRAN

Packages

- developed by the community
- cover several very diverse areas
- uniformly structured and docume
- organized in repositories:
  - CRAN

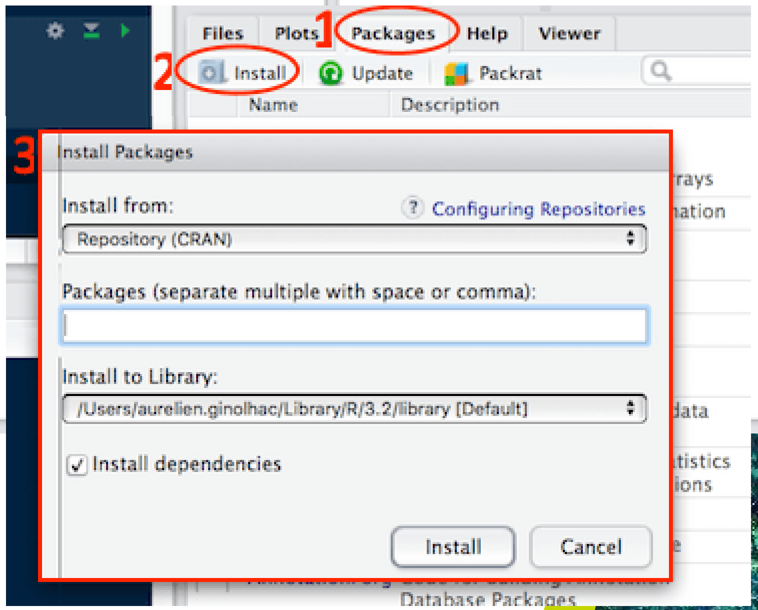


Files | Plots | Packages | Help | Viewer | Presentation

Install | Update

	Name	Description	Version
☐	tidytree	A Tidy Tool for Phylogenetic Tree Data Manipulation	0.4.5
✓	tidyverse	Easily Install and Load the 'Tidyverse'	2.0.0
☐	timechange	Efficient Manipulation of Date-Times	0.2.0
☐	timeDate	Rmetrics - Chronological and Calendar Objects	4022.108
☐	tinytex	Helper Functions to Install and Maintain TeX Live, and Compile LaTeX Documents	0.48
☐	topGO	Enrichment Analysis for Gene Ontology	2.52.0
☐	treeio	Base Classes and Functions for Phylogenetic Tree Input and Output	1.24.3
☐	triebeard	'Radix' Trees in 'Rcpp'	0.4.1

```
Type 'q()' to quit R.

>
```



Install Packages

Install from: Repository (CRAN)

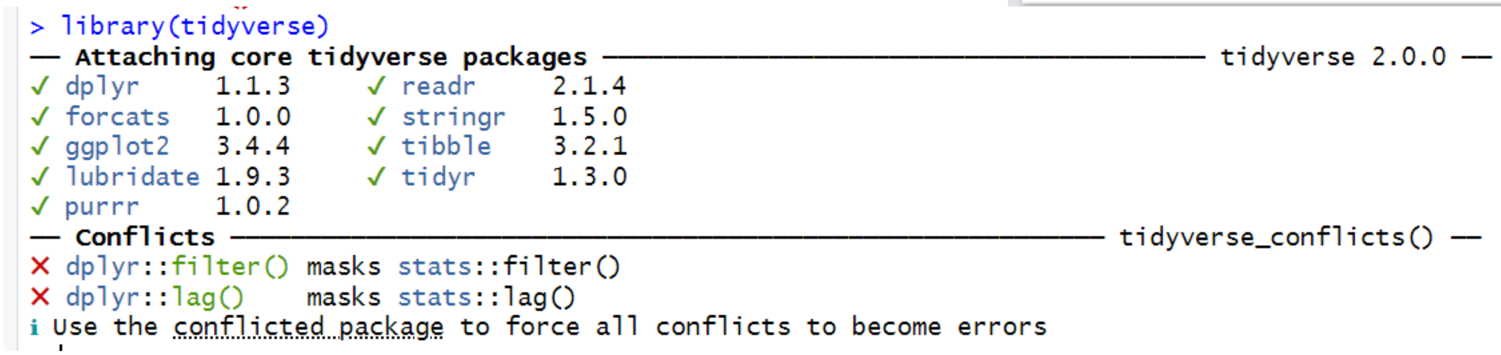
? Configuring Repositories

Packages (separate multiple with space or comma):

Install to Library: /Users/aurelien.ginolhac/Library/R/3.2/library [Default]

✓ Install dependencies

Install | Cancel



```
> library(tidyverse)
── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
✓ dplyr     1.1.3     ✓ readr     2.1.4
✓ forcats   1.0.0     ✓ stringr   1.5.0
✓ ggplot2   3.4.4     ✓ tibble    3.2.1
✓ lubridate 1.9.3     ✓ tidyr     1.3.0
✓ purrr     1.0.2
── Conflicts ─────────────────────────────────────────── tidyverse_conflicts() ──
✗ dplyr::filter() masks stats::filter()
✗ dplyr::lag()    masks stats::lag()
i Use the conflicted package to force all conflicts to become errors
```

Packages

- developed by the community
- cover several very diverse areas of science/life
- uniformly structured and documented
- organized in repositories:
  - CRAN
  - Bioconductor
  - R-Forge
  - GitHub

```
if (!require("BiocManager", quietly = TRUE))
    install.packages("BiocManager")
BiocManager::install(version = "3.18")
```

Assign values to objects

Assign values to objects

```
p <- 3

x <- c(1, 6, 8)

y <- c(“car”, “truck”)
```

Scalars, vectors and matrices

Types of vectors and objects

- **Scalar** (0 dimention):

  p <- 3

- **Vector** (arrays -1 dimention) :

  x <- c(1, 6, 8)
  y <- (“car”, “truck”)

```
> vec1 = c(1,4,6,8,10)
> vec1
[1]  1  4  6  8 10
> vec1[5]
[1] 10
```

- **Matrix** (2 dimension):

```
mat=matrix(data=c(9,2,3,4,5,6),ncol=3)
> mat
     [,1] [,2] [,3]
[1,]    9    3    5
[2,]    2    4    6
```

Types of vectors and objects

- **character**: y <- ("car", "truck")
- **numeric**: x <- c(1.1, 6.2, 8.4)
- **integer**: z <- 2
- **logical**: TRUE, FALSE
- **complex**: 1+4i (complex numbers with real and imaginary parts)

R has many functions to examine features of vectors and other objects:,

•class() - what kind of object is it (high-level)?

•typeof() - what is the object's data type (low-level)?

•length() - how long is it? What about two dimensional objects?

•attributes() - does it have any metadata?

Structures

Data structures

- Matrix
- Data frames
- Factors (r assign a level for each values)
- Arrays
- Lists

```
mat=matrix(data=c(9,2,3,4,5,6),ncol=3)
> mat
     [,1] [,2] [,3]
[1,]    9    3    5
[2,]    2    4    6
```

```
> t = data.frame(x = c(11,12,14),
 y = c(19,20,21), z = c(10,9,7))
> t
   x  y  z
1 11 19 10
2 12 20  9
3 14 21  7
```

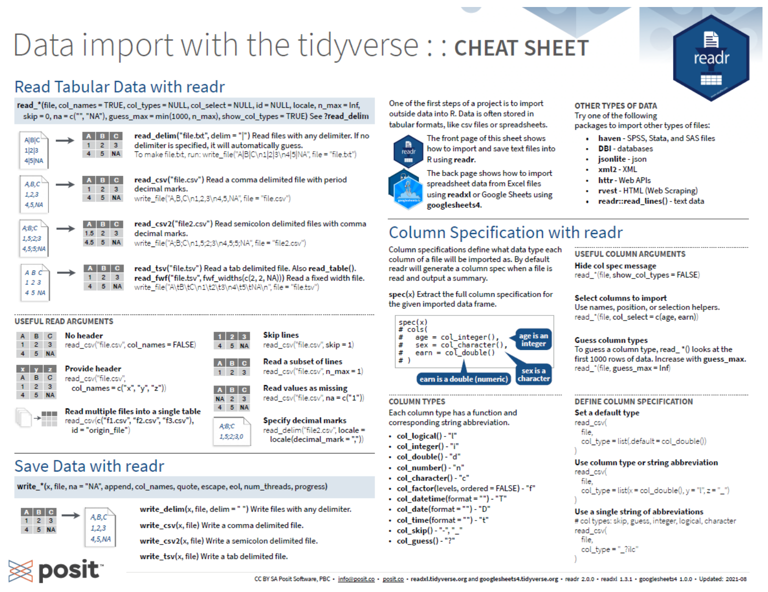
```
> x  <- 1:12

> # Create a 3 x 4 array from the vector
> my_array <- array(values, dim = c(3, 4))

> # Print the array
> print(my_array)
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
> values
 [1]  1  2  3  4  5  6  7  8  9 10 11 12
```

```
> L = list(one=1, two=c(1,2),
 five=seq(0, 1, length=5))
> L
$one
[1] 1
$two
[1] 1 2
```

Getting a dataset in R

Ways to get data in R or in RStudio

- **Manually:**
  - data.frame() function in Base R,
  - or the tibble() function in the tidyverse.
- **Import it from a file:**
  - Fasta: VCF (vcfR package)
  - Text: TXT (readLines() function)
  - Tabular data: CSV, TSV (read.table() or read_delim() fu readr package which contains read_csv())
  - Excel: XLSX (xlsx package)
  - Google sheets: (googlesheets package)
  - Statistics program: SPSS, SAS (haven package)
  - Databases: MySQL (RMySQL package)



Data import with the tidyverse : : CHEAT SHEET

Read Tabular Data with readr

read_*(file, col_names = TRUE, col_types = NULL, col_select = NULL, id = NULL, locale, n_max = Inf, skip = 0, na = c("", "NA"), guess_max = min(1000, n_max), show_col_types = TRUE) See **?read_delim**

read_delim("file.txt", delim = "|") Read files with any delimiter. If no delimiter is specified, it will automatically guess.
To make file.txt, run: write_file("A|B|C\n1|2|3\n4|5|NA", file = "file.txt")

read_csv("file.csv") Read a comma delimited file with period decimal marks.
write_file("A,B,C\n1,2,3\n4,5,NA", file = "file.csv")

read_csv2("file2.csv") Read semicolon delimited files with comma decimal marks.
write_file("A;B;C\n1,5;2;3\n4,5;5;NA", file = "file2.csv")

read_tsv("file.tsv") Read a tab delimited file. Also **read_table()**.
read_fwf("file.tsv", fwf_widths(c(2, 2, NA))) Read a fixed width file.
write_file("A\tB\tC\n1\t2\t3\n4\t5\tNA\n", file = "file.tsv")

USEFUL READ ARGUMENTS

No header
read_csv("file.csv", col_names = FALSE)

Provide header
read_csv("file.csv",
  col_names = c("x", "y", "z"))

Read multiple files into a single table
read_csv(c("f1.csv", "f2.csv", "f3.csv"),
  id = "origin_file")

Skip lines
read_csv("file.csv", skip = 1)

Read a subset of lines
read_csv("file.csv", n_max = 1)

Read values as missing
read_csv("file.csv", na = c("1"))

Specify decimal marks
read_delim("file2.csv", locale =
  locale(decimal_mark = ","))

One of the first steps of a project is to import outside data into R. Data is often stored in tabular formats, like csv files or spreadsheets.

The front page of this sheet shows how to import and save text files into R using **readr**.

The back page shows how to import spreadsheet data from Excel files using **readxl** or Google Sheets using **googlesheets4**.

OTHER TYPES OF DATA
Try one of the following packages to import other types of files:

- **haven** - SPSS, Stata, and SAS files
- **DBI** - databases
- **jsonlite** - json
- **xml2** - XML
- **httr** - Web APIs
- **rvest** - HTML (Web Scraping)
- **readr::read_lines()** - text data

Column Specification with readr

Column specifications define what data type each column of a file will be imported as. By default readr will generate a column spec when a file is read and output a summary.

spec(x) Extract the full column specification for the given imported data frame.

```
spec(x)
# cols(
#   age = col_integer(),
#   sex = col_character(),
#   earn = col_double()
# )
```

COLUMN TYPES
Each column type has a function and corresponding string abbreviation.

- **col_logical()** - "l"
- **col_integer()** - "i"
- **col_double()** - "d"
- **col_number()** - "n"
- **col_character()** - "c"
- **col_factor**(levels, ordered = FALSE) - "f"
- **col_datetime**(format = "") - "T"
- **col_date**(format = "") - "D"
- **col_time**(format = "") - "t"
- **col_skip()** - "-", "_"
- **col_guess()** - "?"

USEFUL COLUMN ARGUMENTS

Hide col spec message
read_*(file, show_col_types = FALSE)

Select columns to import
Use names, position, or selection helpers.
read_*(file, col_select = c(age, earn))

Guess column types
To guess a column type, read_ *() looks at the first 1000 rows of data. Increase with **guess_max**.
read_*(file, guess_max = Inf)

DEFINE COLUMN SPECIFICATION

Set a default type
read_csv(
  file,
  col_type = list(.default = col_double())
)

Use column type or string abbreviation
read_csv(
  file,
  col_type = list(x = col_double(), y = "l", z = "_")
)

Use a single string of abbreviations
col types: skip, guess, integer, logical, character
read_csv(
  file,
  col_type = "_?ilc"
)

Save Data with readr

write_*(x, file, na = "NA", append, col_names, quote, escape, eol, num_threads, progress)

write_delim(x, file, delim = " ") Write files with any delimiter.

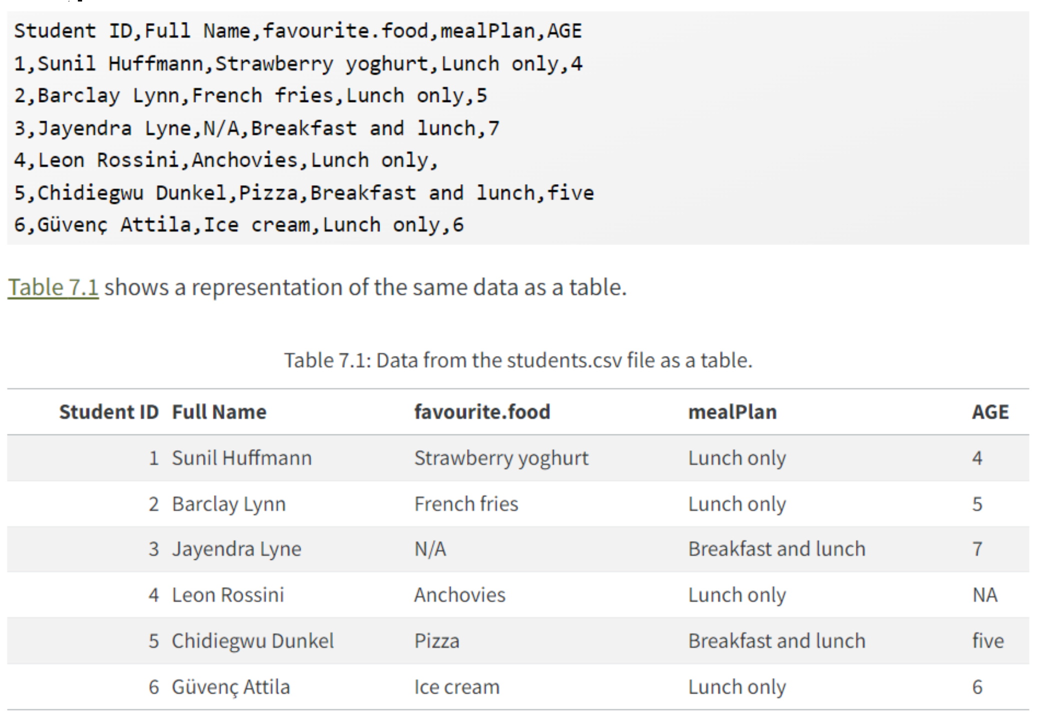
write_csv(x, file) Write a comma delimited file.

write_csv2(x, file) Write a semicolon delimited file.

write_tsv(x, file) Write a tab delimited file.

posit

CC BY SA Posit Software, PBC • info@posit.co • posit.co • readxl.tidyverse.org and googlesheets4.tidyverse.org • readr 2.0.0 • readxl 1.3.1 • googlesheets4 1.0.0 • Updated: 2021-08

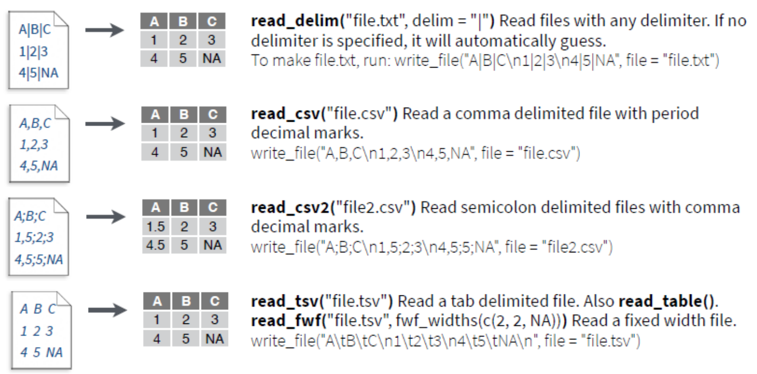


```
Student ID,Full Name,favourite.food,mealPlan,AGE
1,Sunil Huffmann,Strawberry yoghurt,Lunch only,4
2,Barclay Lynn,French fries,Lunch only,5
3,Jayendra Lyne,N/A,Breakfast and lunch,7
4,Leon Rossini,Anchovies,Lunch only,
5,Chidiegwu Dunkel,Pizza,Breakfast and lunch,five
6,Güvenç Attila,Ice cream,Lunch only,6
```

Table 7.1 shows a representation of the same data as a table.

Table 7.1: Data from the students.csv file as a table.

Student ID	Full Name	favourite.food	mealPlan	AGE
1	Sunil Huffmann	Strawberry yoghurt	Lunch only	4
2	Barclay Lynn	French fries	Lunch only	5
3	Jayendra Lyne	N/A	Breakfast and lunch	7
4	Leon Rossini	Anchovies	Lunch only	NA
5	Chidiegwu Dunkel	Pizza	Breakfast and lunch	five
6	Güvenç Attila	Ice cream	Lunch only	6

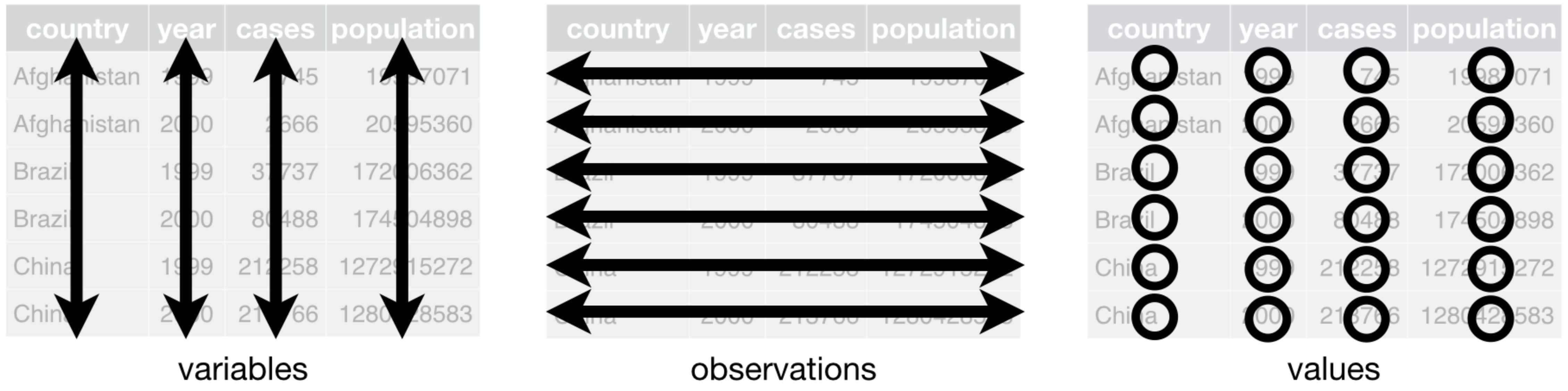


read_delim("file.txt", delim = "|") Read files with any delimiter. If no delimiter is specified, it will automatically guess.
To make file.txt, run: write_file("A|B|C\n1|2|3\n4|5|NA", file = "file.txt")

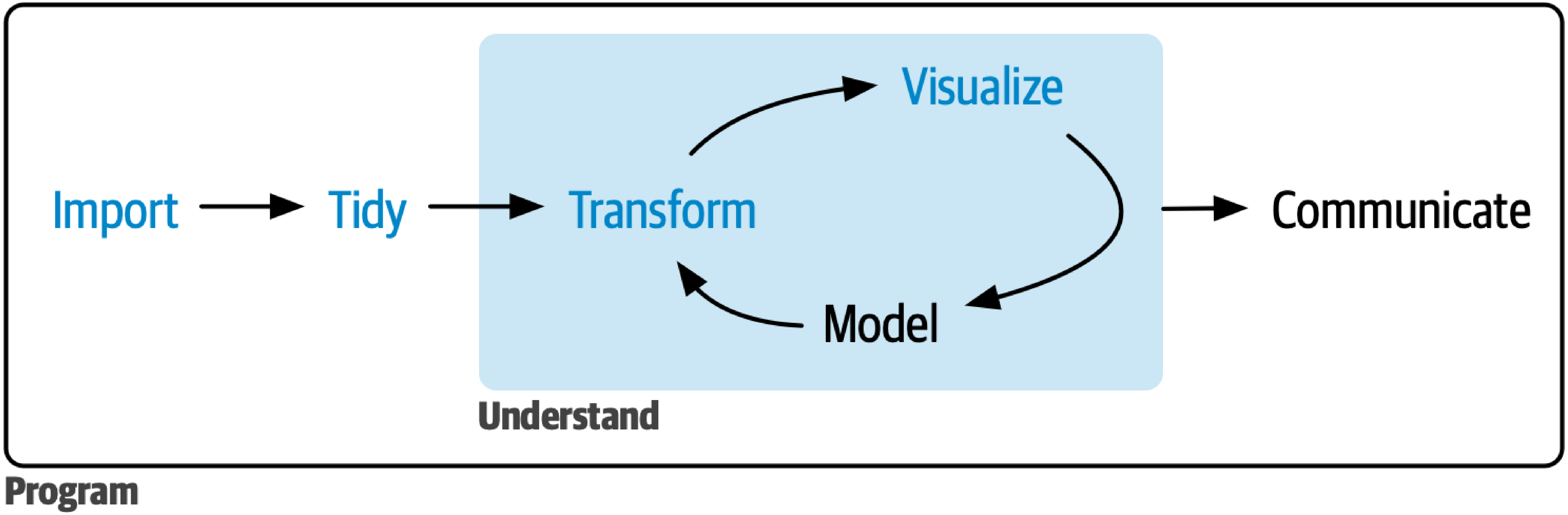
read_csv("file.csv") Read a comma delimited file with period decimal marks.
write_file("A,B,C\n1,2,3\n4,5,NA", file = "file.csv")

read_csv2("file2.csv") Read semicolon delimited files with comma decimal marks.
write_file("A;B;C\n1,5;2;3\n4,5;5;NA", file = "file2.csv")

read_tsv("file.tsv") Read a tab delimited file. Also **read_table()**.
read_fwf("file.tsv", fwf_widths(c(2, 2, NA))) Read a fixed width file.
write_file("A\tB\tC\n1\t2\t3\n4\t5\tNA\n", file = "file.tsv")


country
year
cases
population
variables
observations
values

Processes


Import
Tidy
Transform
Visualize
Model
Communicate
Understand
Program

How to export or save as results or data

```
write.csv(df , file = “path/to/your/saving/folder/df.csv")
```

```
writexl::write_xlsx (test_df, path = "C:/Users/your_username/test_df.xlsx",
col_names = TRUE, format_headers = TRUE)
```

How to export or save as plot

```
dev.print (device=jpeg, file="path/to/your/saving/plots/figure_1a.jpg",
width=par("din")[1]*300, res=300, quality=100)


ggsave("path/to/your/saving/plots/figure_1a.png", plot = p1, bg ="white")
```

!! Practical session (afternoon session): read a test csv file into Rstudio!!

Seeking help

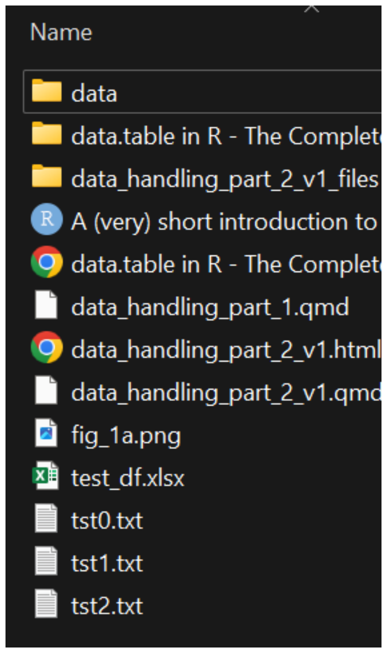
- RStudio help interface
  - ?sum
  - help.search(), with term in “” inside parentheses
- Google or chatgpt “R <task>”
- When asking others
  - Use correct words
  - Reduce to reproducible example
  - Always include output of sessionInfo() function

Handle large datasets

Start by looking at the file names and sizes:

```
fhvhv_csv_files <- list.files("original_csv", recursive=TRUE, full.names = TRUE)
data.frame(file = fhvhv_csv_files, size_Mb = file.size(fhvhv_csv_files) / 1024^2)
```

```
##                                          file   size_Mb
## 1  original_csv/2020/01/fhvhv_tripdata_2020-01.csv 1243.4975
## 2  original_csv/2020/02/fhvhv_tripdata_2020-02.csv 1313.2442
## 3  original_csv/2020/03/fhvhv_tripdata_2020-03.csv  808.5597
## 4  original_csv/2020/04/fhvhv_tripdata_2020-04.csv  259.5806
## 5  original_csv/2020/05/fhvhv_tripdata_2020-05.csv  366.5430
## 6  original_csv/2020/06/fhvhv_tripdata_2020-06.csv  454.5977
## 7  original_csv/2020/07/fhvhv_tripdata_2020-07.csv  599.2560
## 8  original_csv/2020/08/fhvhv_tripdata_2020-08.csv  667.6880
## 9  original_csv/2020/09/fhvhv_tripdata_2020-09.csv  728.5463
## 10 original_csv/2020/10/fhvhv_tripdata_2020-10.csv  798.4743
## 11 original_csv/2020/11/fhvhv_tripdata_2020-11.csv  698.0638
## 12 original_csv/2020/12/fhvhv_tripdata_2020-12.csv  700.6804
```



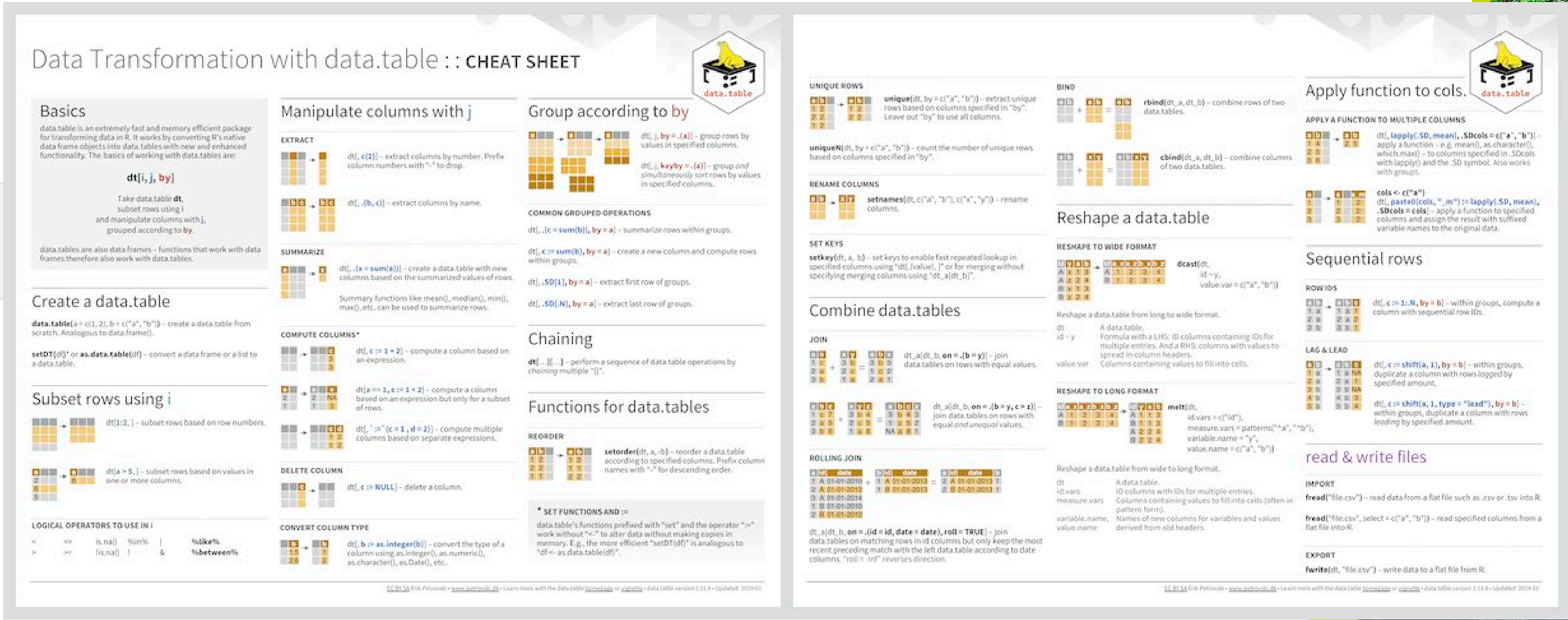
Name
- data
- data.table in R - The Complet
- data_handling_part_2_v1_files
- A (very) short introduction to
- data.table in R - The Complet
- data_handling_part_1.qmd
- data_handling_part_2_v1.html
- data_handling_part_2_v1.qmd
- fig_1a.png
- test_df.xlsx
- tst0.txt
- tst1.txt
- tst2.txt

We can already guess based on these file sizes that with only 4 Gb of RAM available we're going to have a problem.

Handle large datasets

- Good management strategies for large files
  - if you work with **10 to100 GB** regularly!?
  - *r-datatable.com*
  - library(data.table)
  - R script:
    firstscript.R

```
install.packages("arrow")
```



Data Transformation with data.table : : CHEAT SHEET

Basics

data.table is an extremely fast and memory efficient package for transforming data in R. It works by converting R's native data frame objects into data.tables with new and enhanced functionality. The basics of working with data.tables are:

dt[i, j, by]

Take data.table **dt**,
subset rows using i
and manipulate columns with j,
grouped according to by.

data.tables are also data frames – functions that work with data frames therefore also work with data.tables.

Create a data.table

data.table(a = c(1, 2), b = c("a", "b")) – create a data.table from scratch. Analogous to data.frame().

setDT(df)* or **as.data.table**(df) – convert a data frame or a list to a data.table.

Subset rows using i

dt[1:2,] – subset rows based on row numbers.

dt[a > 5,] – subset rows based on values in one or more columns.

LOGICAL OPERATORS TO USE IN i

<	<=	is.na()	%in%	|	**%like%**
>	>=	!is.na()	!	&	**%between%**

Manipulate columns with j

EXTRACT

dt[, c(2)] – extract columns by number. Prefix column numbers with "-" to drop.

dt[, .(b, c)] – extract columns by name.

SUMMARIZE

dt[, .(x = sum(a))] – create a data.table with new columns based on the summarized values of rows.

Summary functions like mean(), median(), min(), max(), etc. can be used to summarize rows.

COMPUTE COLUMNS*

dt[, c := 1 + 2] – compute a column based on an expression.

dt[a == 1, c := 1 + 2] – compute a column based on an expression but only for a subset of rows.

dt[, `:=`(c = 1 , d = 2)] – compute multiple columns based on separate expressions.

DELETE COLUMN

dt[, c := NULL] – delete a column.

CONVERT COLUMN TYPE

dt[, b := as.integer(b)] – convert the type of a column using as.integer(), as.numeric(), as.character(), as.Date(), etc..

Group according to by

dt[, j, by = .(a)] – group rows by values in specified columns.

dt[, j, keyby = .(a)] – group and simultaneously sort rows by values in specified columns.

COMMON GROUPED OPERATIONS

dt[, .(c = sum(b)), by = a] – summarize rows within groups.

dt[, c := sum(b), by = a] – create a new column and compute rows within groups.

dt[, .SD[1], by = a] – extract first row of groups.

dt[, .SD[.N], by = a] – extract last row of groups.

Chaining

dt[…][…] – perform a sequence of data.table operations by chaining multiple "[]".

Functions for data.tables

REORDER

setorder(dt, a, -b) – reorder a data.table according to specified columns. Prefix column names with "-" for descending order.

* SET FUNCTIONS AND :=

data.table's functions prefixed with "set" and the operator ":=" work without "<-" to alter data without making copies in memory. E.g., the more efficient "setDT(df)" is analogous to "df <- as.data.table(df)".

CC BY SA Erik Petrovski • www.petrovski.dk • Learn more with the data.table homepage or vignette • data.table version 1.11.8 • Updated: 2019-01

UNIQUE ROWS

unique(dt, by = c("a", "b")) – extract unique rows based on columns specified in "by". Leave out "by" to use all columns.

uniqueN(dt, by = c("a", "b")) – count the number of unique rows based on columns specified in "by".

RENAME COLUMNS

setnames(dt, c("a", "b"), c("x", "y")) – rename columns.

SET KEYS

setkey(dt, a, b) – set keys to enable fast repeated lookup in specified columns using "dt[.(value),]" or for merging without specifying merging columns using "dt_a[dt_b]".

Combine data.tables

JOIN

dt_a[dt_b, **on = .(b = y)**] – join data.tables on rows with equal values.

dt_a[dt_b, **on = .(b = y, c > z)**] – join data.tables on rows with equal and unequal values.

ROLLING JOIN

dt_a[dt_b, **on = .(id = id, date = date), roll = TRUE**] – join data.tables on matching rows in id columns but only keep the most recent preceding match with the left data.table according to date columns. "roll = -Inf" reverses direction.

BIND

rbind(dt_a, dt_b) – combine rows of two data.tables.

cbind(dt_a, dt_b) – combine columns of two data.tables.

Reshape a data.table

RESHAPE TO WIDE FORMAT

```
dcast(dt,
      id ~ y,
      value.var = c("a", "b"))
```

Reshape a data.table from long to wide format.

dt	A data.table.
id ~ y	Formula with a LHS: ID columns containing IDs for multiple entries. And a RHS: columns with values to spread in column headers.
value.var	Columns containing values to fill into cells.

RESHAPE TO LONG FORMAT

```
melt(dt,
     id.vars = c("id"),
     measure.vars = patterns("^a", "^b"),
     variable.name = "y",
     value.name = c("a", "b"))
```

Reshape a data.table from wide to long format.

dt	A data.table.
id.vars	ID columns with IDs for multiple entries.
measure.vars	Columns containing values to fill into cells (often in pattern form).
variable.name, value.name	Names of new columns for variables and values derived from old headers.

Apply function to cols.

APPLY A FUNCTION TO MULTIPLE COLUMNS

dt[, lapply(.SD, mean), .SDcols = c("a", "b")] – apply a function – e.g. mean(), as.character(), which.max() – to columns specified in .SDcols with lapply() and the .SD symbol. Also works with groups.

cols <- c("a")
dt[, paste0(cols, "_m") := lapply(.SD, mean), **.SDcols = cols**] – apply a function to specified columns and assign the result with suffixed variable names to the original data.

Sequential rows

ROW IDS

dt[, c := 1:.N, by = b] – within groups, compute a column with sequential row IDs.

LAG & LEAD

dt[, c := shift(a, 1), by = b] – within groups, duplicate a column with rows lagged by specified amount.

dt[, c := shift(a, 1, type = "lead"), by = b] – within groups, duplicate a column with rows leading by specified amount.

read & write files

IMPORT

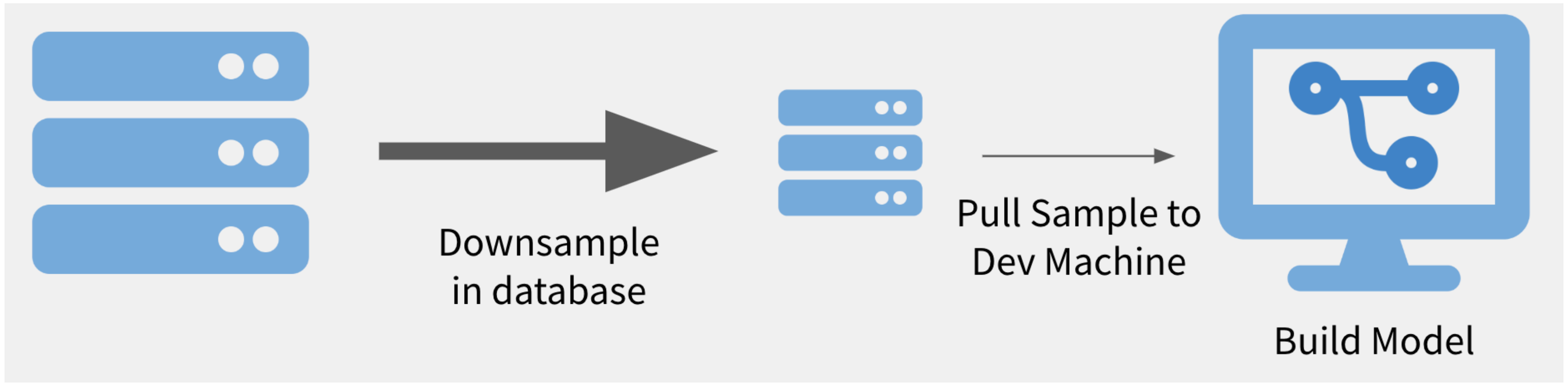
fread("file.csv") – read data from a flat file such as .csv or .tsv into R.

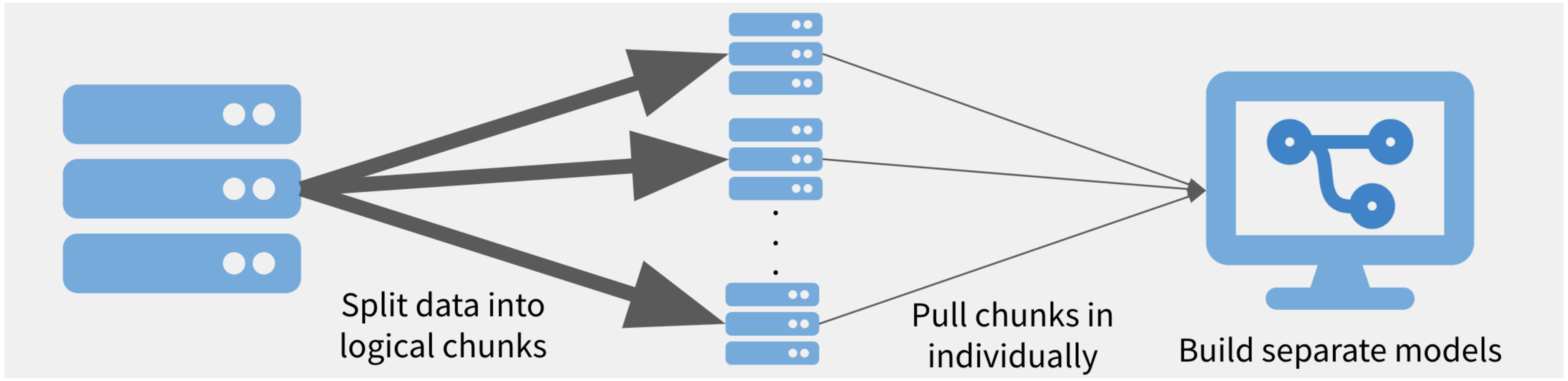
fread("file.csv", select = c("a", "b")) – read specified columns from a flat file into R.

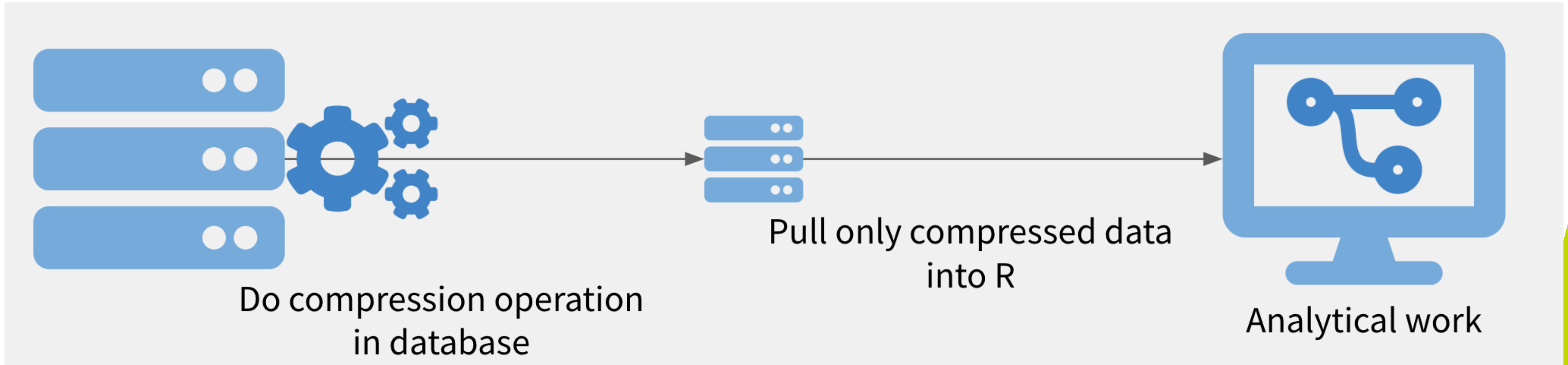
EXPORT

fwrite(dt, "file.csv") – write data to a flat file from R.

CC BY SA Erik Petrovski • www.petrovski.dk • Learn more with the data.table homepage or vignette • data.table version 1.11.8 • Updated: 2019-01


Downsample in database
Pull Sample to Dev Machine
Build Model

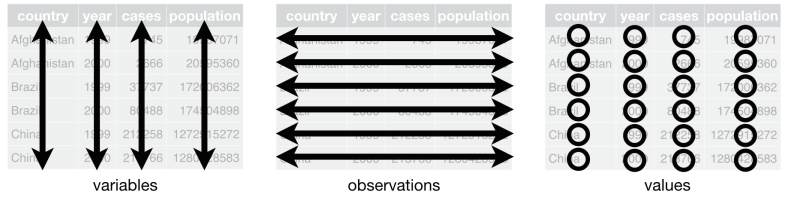

Split data into logical chunks
Pull chunks in individually
Build separate models


Do compression operation in database
Pull only compressed data into R
Analytical work

Good housekeeping strategies for scripts

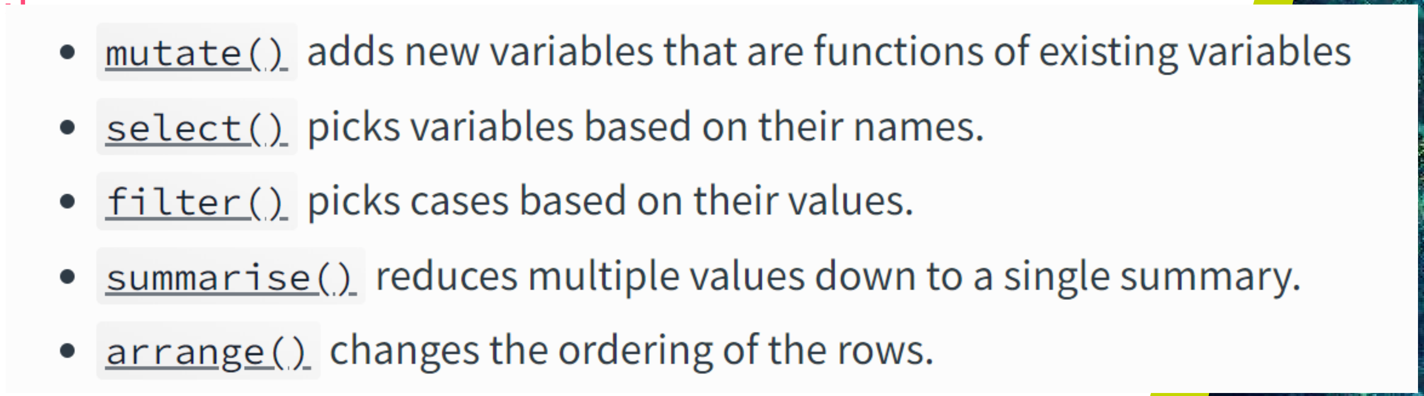
- Comments, Structure:
  - Use #
  - Outline
- Consistent Naming Conventions:
  - Use consistent naming conventions for variables, functions, and objects
  - Avoid duplicating code. create a function or use a loop
- Version Control:
  - Git. Platforms like GitHub or GitLab repositories.
- Imports and Dependencies:
  - List all package imports at the top of your script
  - call rhistory: sessionInfo()
- File Organization:
  - Separate your R scripts, data, documentation, and output files into logical folders.
- Reproducibility:
  - Use Quarto, R Markdown Documents:
  - Documentation Files:
  - Create README files

Filtering of genotype data



- Subset data, metadata
  - Based on "rows (observations)" , "variable" or "values"
  - Filter
- Missing data ("NA", stat, summary)

```
height_sub_12 <- subset(data, height<12)

data_filter_na <- data[!is.na(data$height),]
```



- `mutate()` adds new variables that are functions of existing variables
- `select()` picks variables based on their names.
- `filter()` picks cases based on their values.
- `summarise()` reduces multiple values down to a single summary.
- `arrange()` changes the ordering of the rows.

Conversion of different genotype file formats

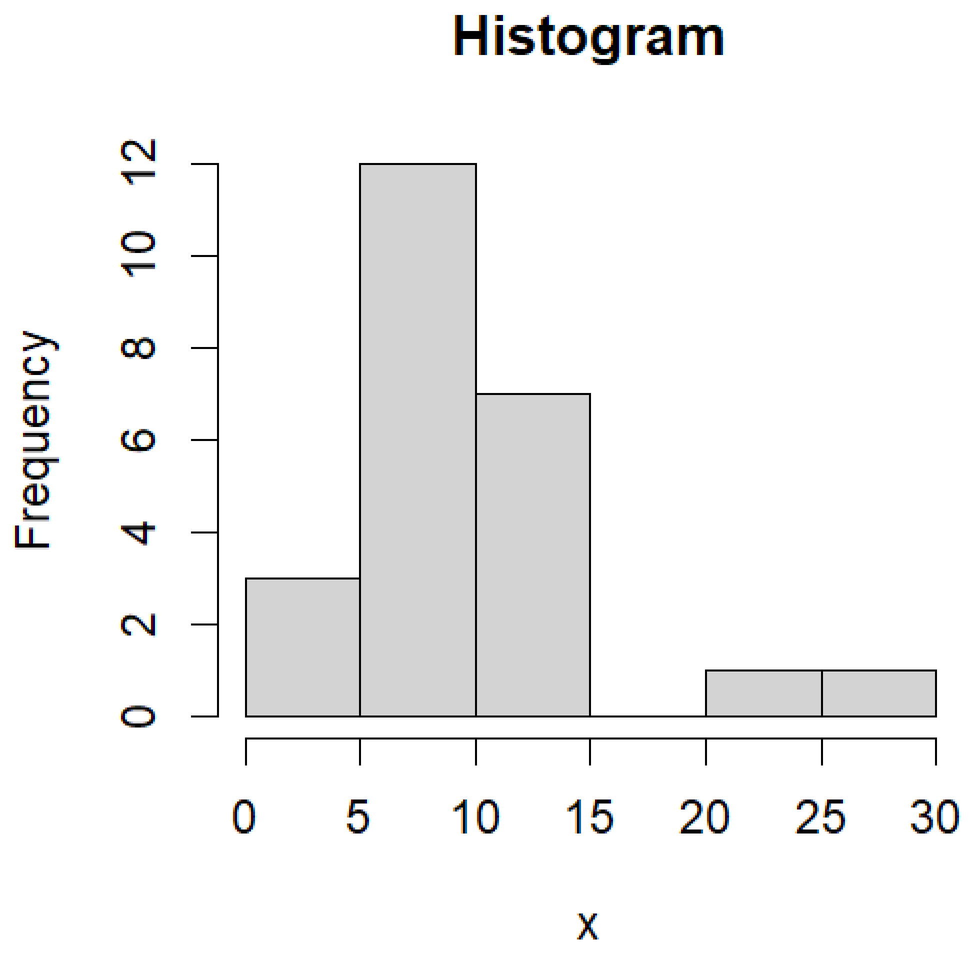
Common programs used to handle the file formats:

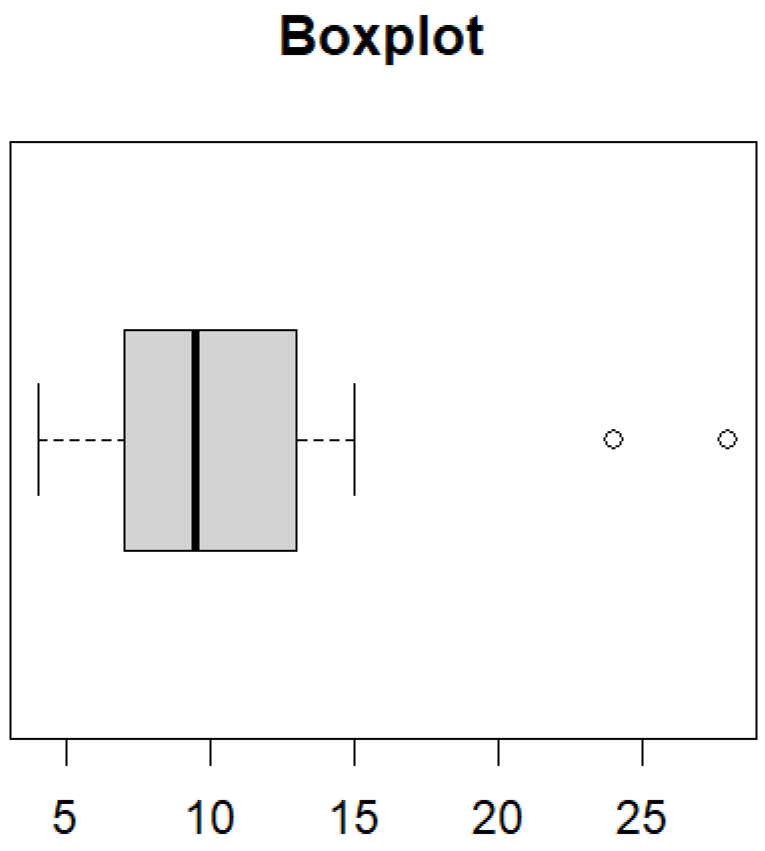
- Hapmap: library(plink), library(VariantAnnotation)
- Numeric: library(readxl, writexl), library(jsonlite)
- Haploid format – one letter code (readLines function)

Outliers in genotype and phenotype datasets

- Histogram, scatterplot, and boxplot, Q-Q plot, chi square test

x = c(10,4,6,8,9,8,7,6,12,14,11,9,8,4,5,10,14,12,15,7,10,14,24,28)



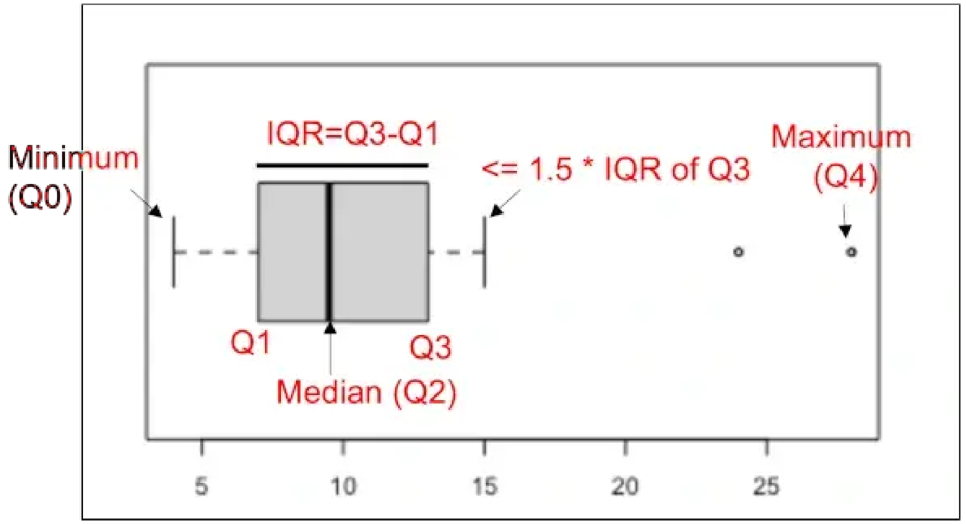


Outliers in genotype and phenotype datasets

- Histogram, scatterplot, and boxplot, Q-Q plot, chi square test

- IQR

```
x = c(10,4,6,8,9,8,7,6,12,14,11,9,8,4,5,10,14,12,15,7,10,14,24,28)

# get values of Q1, Q3, and IQR
summary(x)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   4.00    7.00    9.50   10.62   12.50   28.00

# get IQR
IQR(x)
[1] 5.5

# get threshold values for outliers
Tmin = 7-(1.5*5.5)
Tmax = 12.50+(1.5*5.5)

# find outlier
x[which(x < Tmin | x > Tmax)]
[1] 24 28

# remove outlier
x[which(x > Tmin & x < Tmax)]
[1] 10  4  6  8  9  8  7  6 12 14 11  9  8  4  5 10 14 12 15  7 10 14
```



Questions?!

Acknowledgements

- NBIS