

全球衛生的在地參與



INTRODUCTION TO DATA ANALYSIS IN R

Quantitative Workshop III
a SHANTU project

Erik de Jong

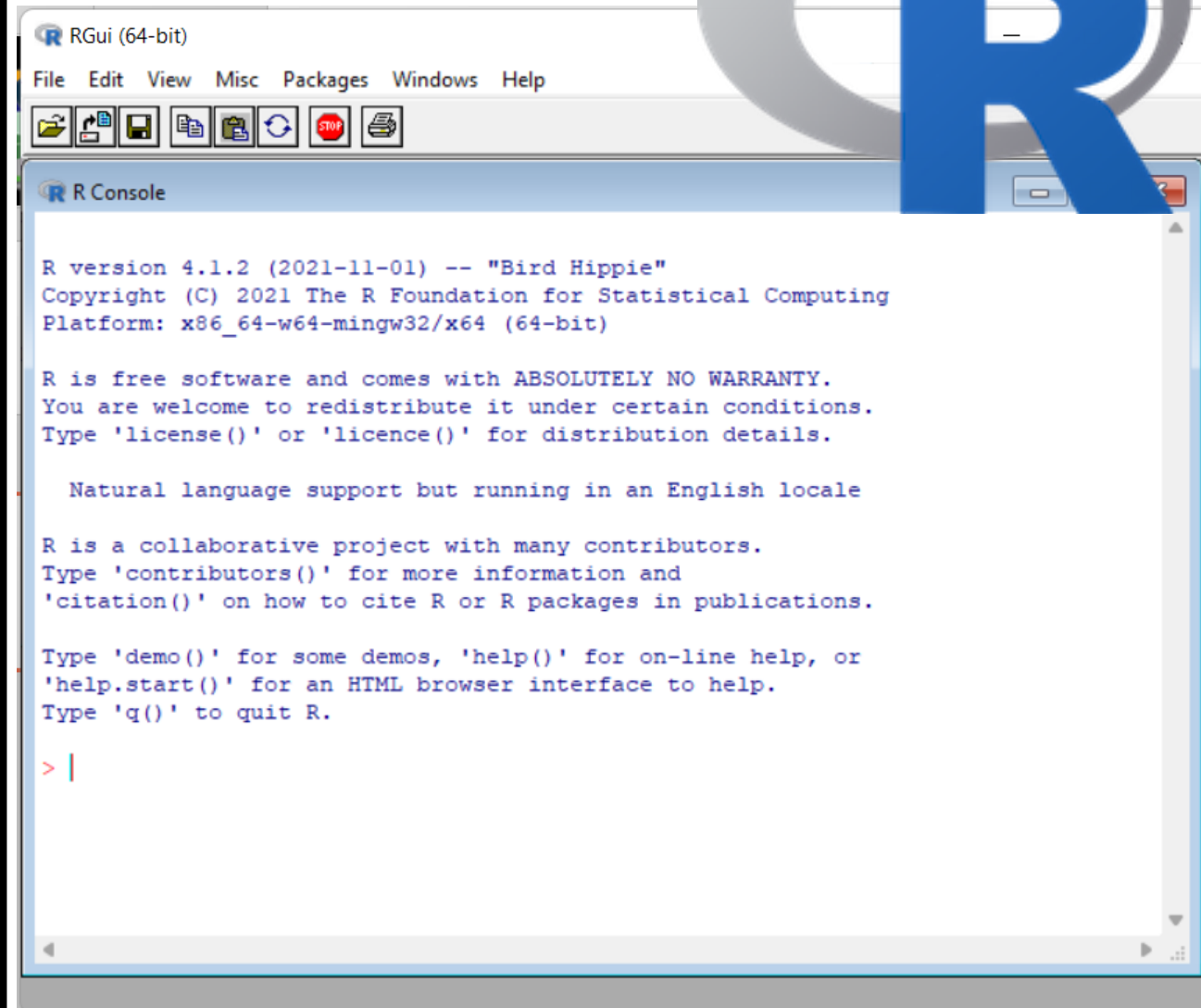


TODAY'S AGENDA

- R and R studio
- Packages
- Functions
- Tips
- Hands-on practice

THE R PROJECT

- R for statistical computing and graphics
- Command line interface
- Download R at <https://cran.r-project.org/> and choose system and “base R”

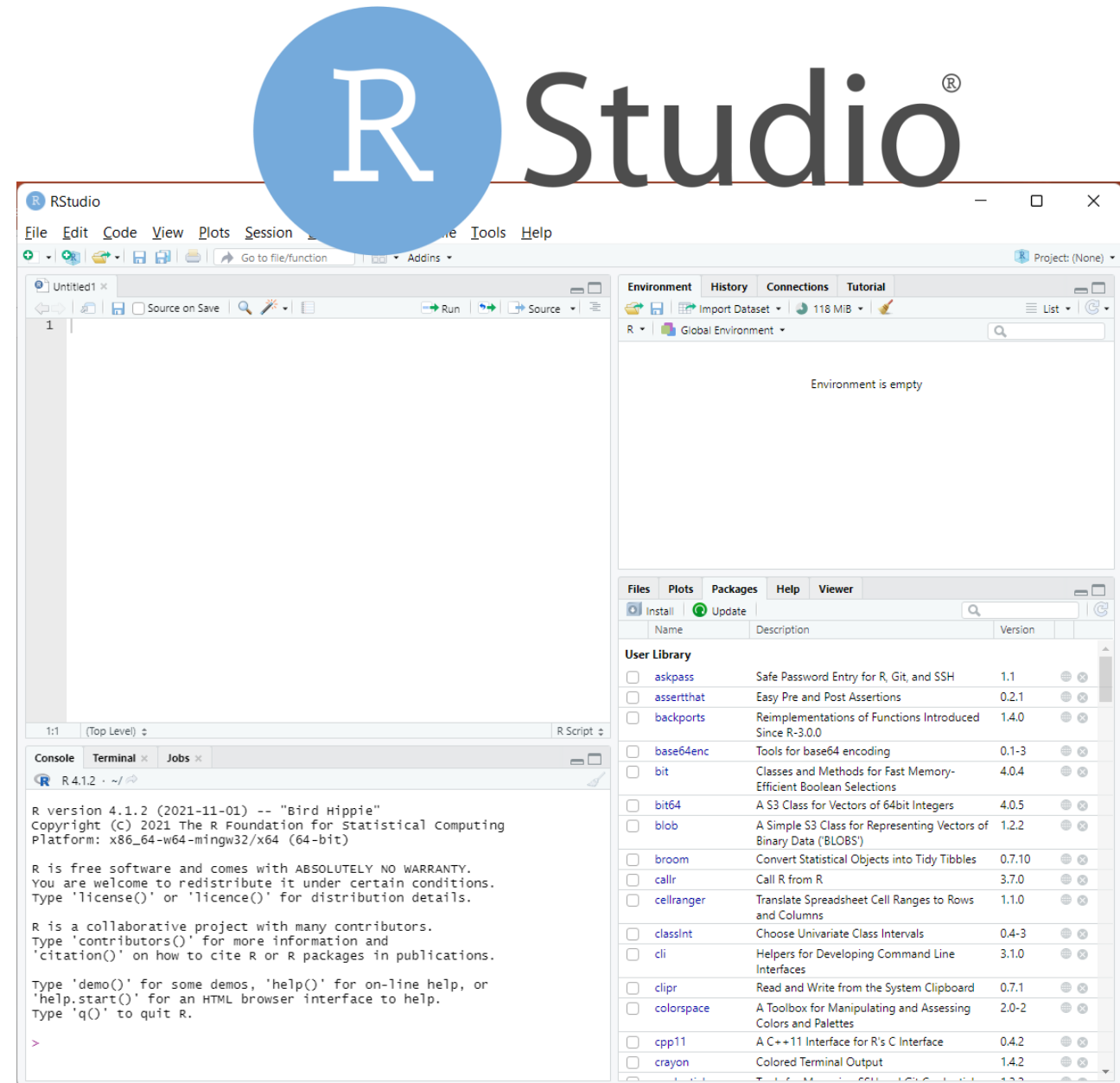


R STUDIO

- Depends on the base version of R
- “all in one” environment
- Runs R in the R studio environment, besides:
 - Code editor & data viewer
 - Environment overview and log
 - Plots, packages, files and help documentation

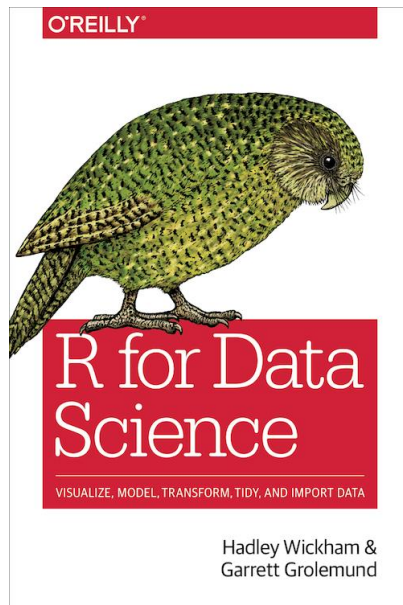
- Download R studio desktop at:

<https://www.rstudio.com/products/rstudio/download/>

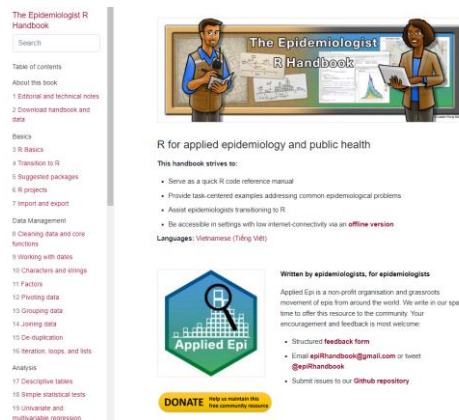


WHY R AND R STUDIO?

For data analysis and visualization?

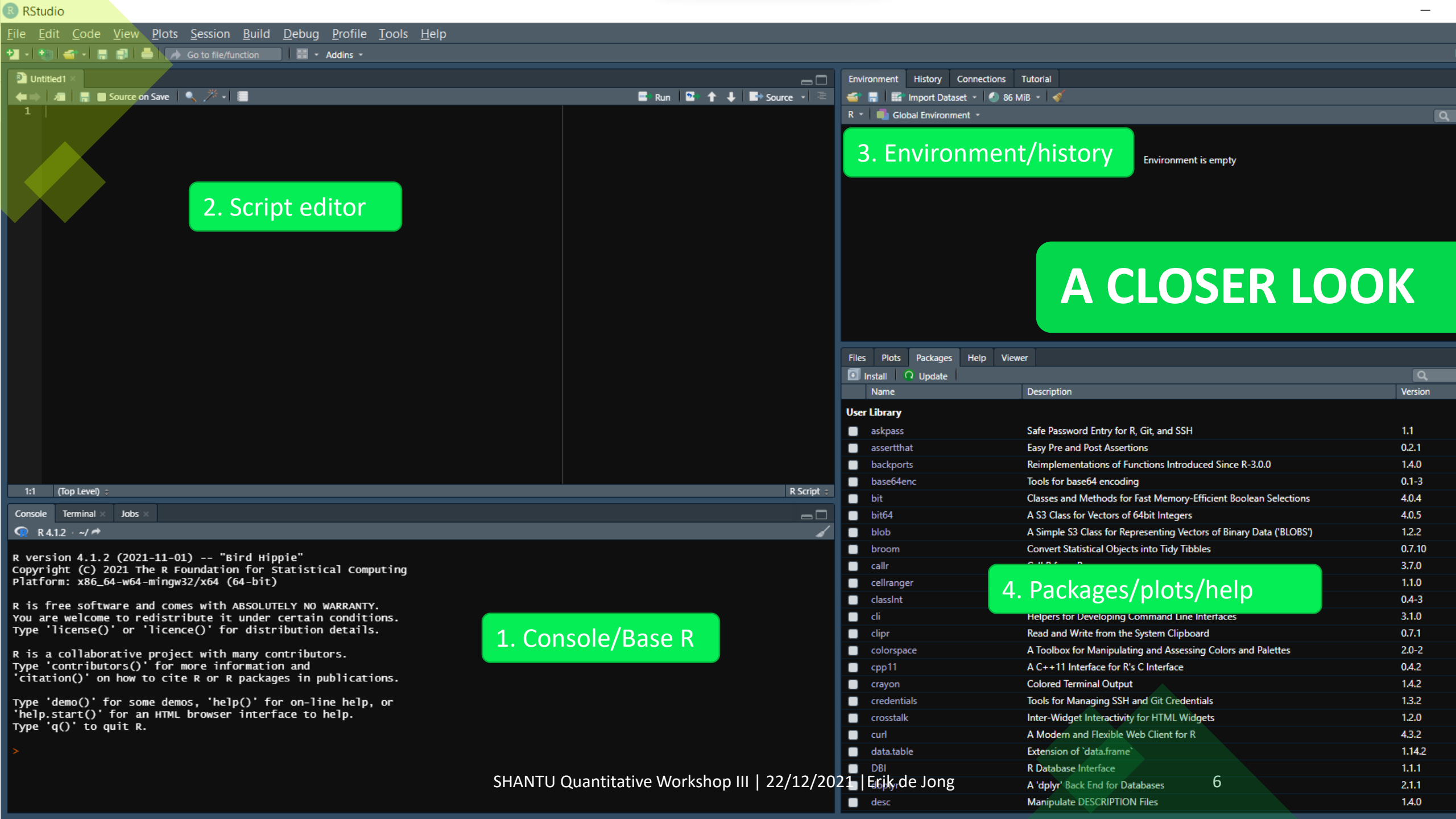


<https://r4ds.had.co.nz/>



<https://www.epirhandbook.com/en/index.html>

- Open source → Free for personal use
- Available for many platforms/distributions (and also older)
- Handles large data
- Every step is documented → analysis can be applied to other data
- Packages designed for specific tasks
- Interactive graphs and plots
- Plotting of special information (GIS)
- Lots of resources
 - Tutorials
 - Free eBooks
 - Stack overflow with specific questions
 - Package documentation
 - Package vignettes



2. Script editor

1. Console/Base R

3. Environment/history

A CLOSER LOOK

4. Packages/plots/help

```
Untitled1
1
```

```
R 4.1.2 ~/>
R version 4.1.2 (2021-11-01) -- "Bird Hippie"
Copyright (C) 2021 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

Environment is empty

Name	Description	Version
User Library		
<input type="checkbox"/> askpass	Safe Password Entry for R, Git, and SSH	1.1
<input type="checkbox"/> assertthat	Easy Pre and Post Assertions	0.2.1
<input type="checkbox"/> backports	Reimplementations of Functions Introduced Since R-3.0.0	1.4.0
<input type="checkbox"/> base64enc	Tools for base64 encoding	0.1-3
<input type="checkbox"/> bit	Classes and Methods for Fast Memory-Efficient Boolean Selections	4.0.4
<input type="checkbox"/> bit64	A S3 Class for Vectors of 64bit Integers	4.0.5
<input type="checkbox"/> blob	A Simple S3 Class for Representing Vectors of Binary Data ('BLOBS')	1.2.2
<input type="checkbox"/> broom	Convert Statistical Objects into Tidy Tibbles	0.7.10
<input type="checkbox"/> callr	Call R Processes	3.7.0
<input type="checkbox"/> cellranger	Import and Export Data from Spreadsheets	1.1.0
<input type="checkbox"/> classInt	Classes and Methods for Integer Intervals	0.4-3
<input type="checkbox"/> cli	Helpers for Developing Command Line Interfaces	3.1.0
<input type="checkbox"/> clipr	Read and Write from the System Clipboard	0.7.1
<input type="checkbox"/> colorspace	A Toolbox for Manipulating and Assessing Colors and Palettes	2.0-2
<input type="checkbox"/> cpp11	A C++11 Interface for R's C Interface	0.4.2
<input type="checkbox"/> crayon	Colored Terminal Output	1.4.2
<input type="checkbox"/> credentials	Tools for Managing SSH and Git Credentials	1.3.2
<input type="checkbox"/> crosstalk	Inter-Widget Interactivity for HTML Widgets	1.2.0
<input type="checkbox"/> curl	A Modern and Flexible Web Client for R	4.3.2
<input type="checkbox"/> data.table	Extension of 'data.frame'	1.14.2
<input type="checkbox"/> DBI	R Database Interface	1.1.1
<input type="checkbox"/> dplyr	A 'dplyr' Back End for Databases	2.1.1
<input type="checkbox"/> desc	Manipulate DESCRIPTION Files	1.4.0

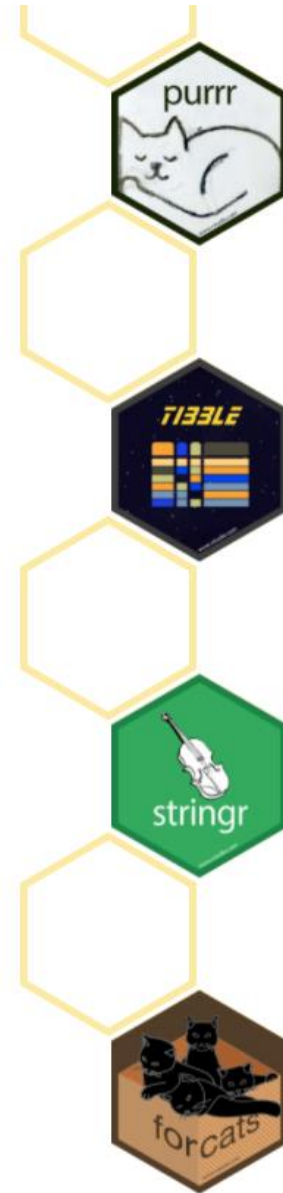
GET STARTED IN R

What do you need to
know beforehand



PACKAGES IN R

- Many people develop (sometimes very specific) packages for R
- Packages are sets of functions that R can carry out specific tasks
- Examples include:
 - Readxl, to Import excel data into R
 - Tidyverse, a combination of 8 essential packages
- Every time you start R you need to load packages, but install is only once



purrr

purrr enhances R's functional programming (FP) toolkit by providing a complete and consistent set of tools for working with functions and vectors. Once you master the basic concepts, purrr allows you to replace many for loops with code that is easier to write and more expressive. Go to docs...

tibble

tibble is a modern re-imagining of the data frame, keeping what time has proven to be effective, and throwing out what it has not. Tibbles are data.frames that are lazy and surly: they do less and complain more forcing you to confront problems earlier, typically leading to cleaner, more expressive code. Go to docs...

stringr

stringr provides a cohesive set of functions designed to make working with strings as easy as possible. It is built on top of stringi, which uses the ICU C library to provide fast, correct implementations of common string manipulations. Go to docs...

forcats

forcats provides a suite of useful tools that solve common problems with factors. R uses factors to handle categorical variables, variables that have a fixed and known set of possible values. Go to docs...

INSTALL PACKAGES IN R

Install packages -----

`install.packages("readxl")` #installs the package read xl

`install.packages("tidyverse")` #installs the tidyverse packages

install is only needed once, loading of packages (library) is needed every time you start R

Library functions -----

`library()` #gives you a list of installed

`library(readxl)` #loads the package read xl for usage

`library(tidyverse)` #loads the tidyverse packages for usage

`search()` #show the packages loaded, available for direct use

```
# Install packages -----  
install.packages("readxl") #installs the package read xl  
install.packages("tidyverse") #installs the tidyverse packages  
  
# Library functions -----  
library() #gives you a list of installed  
library(readxl) #loads the package read xl for usage  
library(tidyverse) #loads the tidyverse packages for usage  
search() #show the packages loaded, available for direct use
```

FUNCTIONS IN R

And what they do

- Functions are designed to do a specific task, for example:
 - `mean()` gives the average of a string of numbers
 - `plot()` will make a plot (if certain parameters are given)
- Arguments can be passed on to the function between the `()`
- Some arguments have default values

Function name

```
mean (x, trim= , na.rm= )
```

Function arguments

Generic X-Y Plotting

Description

Generic function for plotting of R objects.

For simple scatter plots, `plot.default` will be used. However, there are `plot` methods for many R objects, including `functions`, `data.frames`, `density` objects, etc. Use `methods(plot)` and the documentation for these. Most of these methods are implemented

HELP FUNCTIONS

Build into R

- Many packages and functions, impossible to remember
- R has build in help functions:
 - ? Followed by function name
 - ?? Followed by function name
 - Example() with the function name in brackets

Search Results 

Vignettes:

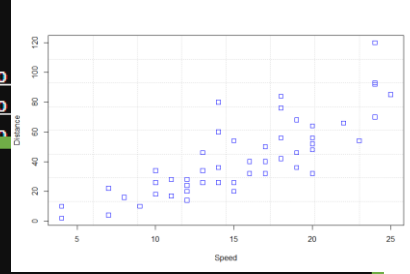
ggplot2::extending-ggplot2	Extending ggplot2	HTML	source	R code
ggplot2::ggplot2-in-packages	Using ggplot2 in packages	HTML	source	R code
ggplot2::ggplot2-specs	Aesthetic specifications	HTML	source	R code
sf::sf5	5. Plotting Simple Features	HTML	source	R code
grid::plotexample	Writing grid Code	PDF	source	R code
survival::splines	Splines, plots, and interactions	PDF	source	R code

Code demonstrations:

sf::ggplot	shows use of geom_sf	(Run demo)
sf::webmap	webmap demo from sp, modified to plot sf objects	(Run demo)
sp::polar	polar plot showing gridlines and gridline labels in epsg 3031	(Run demo)
sp::webmap	OpenStreetMap, ggmap or R, demonstrate	
units::ggforce	Examples of	
graphics::plotmath	Creates a canvas	
tcltk::tkcanvas	dragged with	

```

> example(plot)
plot> Speed <- cars$speed
plot> Distance <- cars$dist
plot> plot(Speed, Distance, panel.first = grid(8, 8),
plot+   pch = 0, cex = 1.2, col = "blue")
Hit <Return> to see next plot:
plot> plot(Speed, Distance,
plot+   panel.first = lines(stats::lowess(Speed, Distance), lty = "dashed"),
plot+   pch = 0, cex = 1.2, col = "blue")
Hit <Return> to see next plot:
  
```



DATATYPES

Commonly used in R

- Numeric (for numbers)
- Integer (natural number)
- Logic (true or false)
- Character (text)

```
> class(4L)
[1] "integer"
> class(a)
[1] "numeric"
> class(s)
Error: object 's' not found
> class("sss")
[1] "character"
> 4
[1] 4
> class(4.4)
[1] "numeric"
> |
```

OPERATORS

R Arithmetic Operators

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
^	Exponent
%%	Modulus
%/%	Integer Division

R Relational Operators

Operator	Description
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	Equal to
!=	Not equal to

OPERATORS

R Logical Operators

Operator	Description
!	Logical NOT
&	Element-wise logical AND
&&	Logical AND
	Element-wise logical OR
	Logical OR

R Assignment Operators

Operator	Description
<-, <<-, =	Leftwards assignment
->, ->>	Rightwards assignment

DATA STRUCTURES

V e c t o r

- Store simple data
- One dimensional
- Can be anything from numeric, integer, logical, or character
- `c()` is a function to create a factor
- `x <- c(45,5,66,5.5,NA)` assigns a numeric values to the vector and names the vector a

```
# numeric vector c(1,2,3,4.2,-5)
[1] 1.0 2.0 3.0 4.2 -5.0
# vector of consecutive numbers
c(1:5)
[1] 1 2 3 4 5
# character vector
c("apple","grape","lemon","orange")
[1] "apple" "grape" "lemon"
"orange"
# logical
c(TRUE,FALSE,FALSE,TRUE,TRUE)
[1] TRUE FALSE FALSE TRUE TRUE
```

DATA STRUCTURES

Data frames

- Combination of multiple vectors
- Two dimensional
- Fundamental data structure to store datasets
- Variables are as columns
- Observations are as rows
- Each column contains the same data type
- Each row can contain different data types

```
# Create a dataframe
patient_id<-c(1:5)
gender<- c("M","F","F","F","M")
dob<- c("1990-01-04","1987-10-10","1970-11-12",
,"2001-05-05","1977-05-11")
malaria<- c(T,F,F,T,F)
df<- data.frame(patient_id,gender,dob,malaria)
df
```

	patient_id	gender	dob	malaria
1	1	M	1990-01-04	TRUE
2	2	F	1987-10-10	FALSE
3	3	F	1970-11-12	FALSE
4	4	F	2001-05-05	TRUE
5	5	M	1977-05-11	FALSE

DATA STRUCTURES

Data frame value selection

`dataframe[row, column]`

`dataframe[1,]`

`dataframe[,variable_name]`

`dataframe$variable_name`

```
df
  patient_id gender      dob malaria
1          1     M 1990-01-04     TRUE
2          2     F 1987-10-10    FALSE
3          3     F 1970-11-12    FALSE
4          4     F 2001-05-05     TRUE
5          5     M 1977-05-11    FALSE
```

```
df[,4]
```

```
[1] TRUE FALSE FALSE TRUE FALSE
```

```
df["malaria"]
```

```
[1] TRUE FALSE FALSE TRUE FALSE
```

```
df$malaria
```

```
[1] TRUE FALSE FALSE TRUE FALSE
```

DATA STRUCTURES

Other

- Matrix
- List
- Factor

- Some packages work with their own datastructures

SOME TIPS

From my personal experience

- Always use R scrips and save them
- The larger your data gets, data cleaning in R is fast
- If your scripts get more complex → make a connection to git(hub) for version control
- Make your coding readable and searchable:
 - For yourself and others
 - Easier to search later
 - Use # to comment on a line of coding (or use ctrl+shift+c)
 - Use section in your r code (ctrl+shift+r)



Sign up

Where the world builds software

Millions of developers and companies build, ship, and maintain their software on GitHub—the largest and most advanced development platform in the world.

Sign up for GitHub

73+ million

Developers

4+ million

Organizations

84%

Fortune 1000

OTHER RESOURCES

<https://www.rstudio.com/resources/cheatsheets/>

Data transformation with dplyr : : CHEAT SHEET

dplyr functions work with pipes and expect tidy data. In tidy data:



Summarise Cases

Apply summary functions to columns to create a new table of summary statistics. Summary functions take vectors as input and return one value (see back).

```
summary_function
summarise(data, ...)
  Compute table of summaries.
  summarise(mtcars, avg = mean(mpg))

count(data, ..., wt = NULL, sort = FALSE, name = NULL)
  Count number of rows in each group defined by the variables in ... Also tally().
  count(mtcars, cyl)
```

Manipulate Cases

Row functions return a subset of rows as a new table.

```
filter(data, ..., preserve = FALSE)
  Extract rows that meet logical criteria.
  filter(mtcars, mpg > 20)

distinct(data, ..., keep_all = FALSE)
  Remove rows with duplicate values.
  distinct(mtcars, gear)

slice(data, ..., preserve = FALSE)
  Select rows by position.
  slice(mtcars, 10:15)
```

Manipulate Variables

Column functions return a set of columns as a new vector or table.

```
pull(data, var = 1, name = NULL, ...)
  Extract column values as a vector, by name or index.
  pull(mtcars, wt)

select(data, ...)
  Extract columns as a table.
  select(mtcars, mpg, wt)

relocate(data, ..., before = NULL, after = NULL)
  Move columns to new position.
  relocate(mtcars, mpg, cyl, after = last_col())
```

Data visualization with ggplot2 : : CHEAT SHEET

Basics

ggplot2 is based on the grammar of graphics, the idea that you can build every graph from the same components: a data set, a coordinate system, and geoms—visual marks that represent data points.



To display values, map variables in the data to visual properties of the geom (aesthetics) like size, color, and x and y locations.



Complete the template below to build a graph.

```
ggplot(data = <DATA>) +
  <GEOM_FUNCTION>[mapping = aes(<MAPPINGS>)]
  stat = <STAT>, position = <POSITION> +
  <COORDINATE_FUNCTION> +
  <FACET_FUNCTION> +
  <SCALE_FUNCTION> +
  <THEME_FUNCTION>
```

ggplot(data = mpg, aes(x = cyl, y = hwy))
 Begins a plot that you finish by adding layers to. Add one geom function per layer.

last_plot()
 Returns the last plot.

ggsave("plot.png", width = 5, height = 5)
 Saves last plot as 5 x 5 file named "plot.png" in working directory. Matches file type to file extension.

Aes

Common aesthetic values: color and fill - string ("red", "#RRGGBB")
linetype - integer or string (0 = "blank", 1 = "solid", 2 = "dashed", 3 = "dotted", 4 = "dotteddash", 5 = "longdash", 6 = "twodash")
lineend - string ("round", "butt", or "square")
linejoin - string ("round", "mitre", or "bevel")
size - integer (line width in mm)
shape - integer/shape name or a single character ("a")

Geoms

Use a geom function to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer.

```
GRAPHICAL PRIMITIVES
a <- ggplot(economics, aes(date, unemploy))
b <- ggplot(seals, aes(x = long, y = lat))

a + geom_blank() and a + expand_limits()
  Ensure limits include values across all plots.

b + geom_curve(aes(yend = lat + 1, xend = long + 1, curvature = 1), x, yend, alpha, angle, color, curvature, linetype, size)

a + geom_path(linewidth = "butt", linetype = "round", linemitre = 1)
  x, y, alpha, color, group, linetype, size

a + geom_polygon(aes(alpha = 50), x, y, alpha, color, fill, group, linetype, size)

b + geom_rect(aes(xmin = long, ymin = lat, xmax = long + 1, ymax = lat + 1), x, y, alpha, color, fill, group, linetype, size)

a + geom_ribbon(aes(ymin = unemploy - 900, ymax = unemploy + 900), x, y, alpha, color, fill, group, linetype, size)
```

```
LINE SEGMENTS
common aesthetics: x, y, alpha, color, linetype, size
b + geom_abline(aes(intercept = 0, slope = 1))
b + geom_hline(aes(yintercept = lat))
b + geom_vline(aes(xintercept = long))

b + geom_segment(aes(yend = lat + 1, xend = long + 1))
b + geom_smooth(aes(angle = 1:155, radius = 1))
```

```
ONE VARIABLE CONTINUOUS
c <- ggplot(mpg, aes(hwy)); c2 <- ggplot(mpg)

c + geom_area(stat = "bin")
  x, y, alpha, color, fill, linetype, size

c + geom_density(kernel = "gaussian")
  x, y, alpha, color, fill, group, linetype, size, weight

c + geom_dotplot()
  x, y, alpha, color, fill

c + geom_freqpoly()
  x, y, alpha, color, group, linetype, size

c + geom_histogram(binwidth = 5)
  x, y, alpha, color, fill, linetype, size, weight

c2 + geom_qq(aes(sample = hwy))
  x, y, alpha, color, fill, linetype, size, weight
```

```
both discrete
g <- ggplot(diamonds, aes(cut, color))

g + geom_count()
  x, y, alpha, color, fill, shape, size, stroke

g + geom_jitter(height = 2, width = 2)
  x, y, alpha, color, fill, shape, size
```

```
discrete
d <- ggplot(mpg, aes(lt))

d + geom_bar()
  x, y, alpha, color, fill, linetype, size, weight
```

TWO VARIABLES BOTH CONTINUOUS

```
continuous bivariate distribution
h <- ggplot(diamonds, aes(carat, price))

h + geom_bin2d(binwidth = (0.25, 500))
  x, y, alpha, color, fill, linetype, size, weight

h + geom_density_2d()
  x, y, alpha, color, group, linetype, size

h + geom_hex()
  x, y, alpha, color, fill, size

continuous function
i <- ggplot(economics, aes(date, unemploy))

i + geom_area()
  x, y, alpha, color, fill, linetype, size

i + geom_line()
  x, y, alpha, color, group, linetype, size

i + geom_step(direction = "hv")
  x, y, alpha, color, group, linetype, size
```

```
one discrete, one continuous
f <- ggplot(mpg, aes(class, hwy))

f + geom_col()
  x, y, alpha, color, fill, group, linetype, size

f + geom_boxplot()
  x, y, lower, middle, upper, ymax, ymin, alpha, color, group, linetype, shape, size, weight

f + geom_dotplot(binaxis = "y", stackdir = "center")
  x, y, alpha, color, fill, group

f + geom_violin(scale = "area")
  x, y, alpha, color, fill, group, linetype, size, weight
```

```
maps
data <- data.frame(murder = USArrests$Murder, state = tolower(row.names(USArrests)))
map <- map_data("state")
k <- ggplot(data, aes(fill = murder))

k + geom_map(aes(map_id = murder), map = map)
  + expand_limits(x = map$long, y = map$lat)
  map_id, alpha, color, fill, linetype, size, weight
```

```
THREE VARIABLES
seatsS2 <- with(seats, sqrt(delta_long^2 + delta_lat^2)); l <- ggplot(seats, aes(long, lat))

l + geom_contour(aes(z = 2))
  x, y, z, alpha, color, group, linetype, size, weight

l + geom_contour_filled(aes(fill = z))
  x, y, alpha, color, fill, group, linetype, size, subgroup
```

Visualizing Error

```
df <- data.frame(grp = c("A", "B"), fit = 4.5, se = 1.2)
j <- ggplot(df, aes(grp, fit, ymin = fit - se, ymax = fit + se))

j + geom_crossbar(aes(linetype = "d", fill = "white", color = "black"))
  x, y, ymin, ymax, alpha, color, fill, group, linetype, size

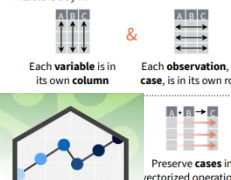
j + geom_errorbar()
  x, y, ymin, ymax, alpha, color, fill, group, linetype, size

j + geom_lineangle()
  x, y, ymin, ymax, alpha, color, fill, group, linetype, size

j + geom_pointrange()
  x, y, ymin, ymax, alpha, color, fill, group, linetype, size
```

Data tidying with tidyr : : CHEAT SHEET

Tidy data is a way to organize tabular data in a consistent data structure across packages. A table is tidy if:



Reshape Data

Pivot data to reorganize values into a new layout.

```
table4a
country year cases
A 1999 0.7K
B 2000 1.9K
C 2001 2.1K

pivot_longer(table4a, cols = 2:3, names_to = "year", values_to = "cases")

table4b
country year cases pop
A 1999 0.7K 194
B 2000 1.9K 194
C 2001 2.1K 194

pivot_wider(table4b, names_from = "year", values_from = "cases")
```

Expand Tables

Create new combinations of variables or identify implicit missing values (combinations of variables not present in the data).

```
expand(data, ...)
  Create a new tibble with all possible combinations of the variables listed in ... Drop other variables.
  expand(mtcars, cyl, gear, carb)
```

Data import with the tidyverse : : CHEAT SHEET

Read Tabular Data with readr

```
read_csv("file.csv", col_types = TRUE, col_select = NULL, col_delim = NULL, locale = "en", max_col = 1, skip = 0, na = c("NA", "NaN"), guess_max = min(1000, n_max), show_col_types = TRUE)
  See Read.Delim

read_delim("file.txt", delim = "\t")
  Read files with any delimiter. If no delimiter is specified, it will automatically guess.
  To make file.txt, run write_file("ABC|1|2|3|4|5|NA", file = "file.txt")

read_csv("file.csv")
  Read a comma delimited file with period decimal marks.
  write_file("A,B,C|1,2,3|4,5,NA", file = "file.csv")

read_csv2("file2.csv")
  Read semicolon delimited files with comma decimal marks.
  write_file("A,B,C|1,2,3|4,5,5NA", file = "file2.csv")

read_csv("file.csv", fwf = widths(c(2, 2, NA)))
  Read a tab delimited file. Also read table().
  write_file("A|B|C|1|2|3|4|5|NA", file = "file.csv")
```

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.

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

```
spec(x)
# cols
# age = col_integer(),
# sex = col_character(),
# earn = col_double()
#
# earn is a double (numeric)
# sex is a character
```

```
USEFUL READ ARGUMENTS
A B C No header
1 2 3 read_csv("file.csv", col_names = FALSE)
4 5 NA

A B C Provide header
1 2 3 read_csv("file.csv", col_names = c("x", "y", "z"))
4 5 NA

A B C Read multiple files into a single table
read_csv(c("1.csv", "2.csv", "3.csv"), id = "origin_file")
A B C
1,2,3
4,5,NA
```

Save Data with readr

```
write("x", file, na = "NA", append, col_names, quote, escape, col_num_threads, progress)

write_delim(x, file, delim = "\t")
  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.
```

HANDS-ON PRACTICE

In R

