Research Data Workshop Series - Introduction to R

Lucas Alcantara, Ph.D.
alcantal@uoguelph.ca

AGRI-FOOD DATA CANADA
AT THE UNIVERSITY of GUELPH

Workshop Outline

- What is R
- Getting Started with R
- R Nuts and Bolts
- Getting Data in and Out of R
- Subsetting R Objects
- Project Time!

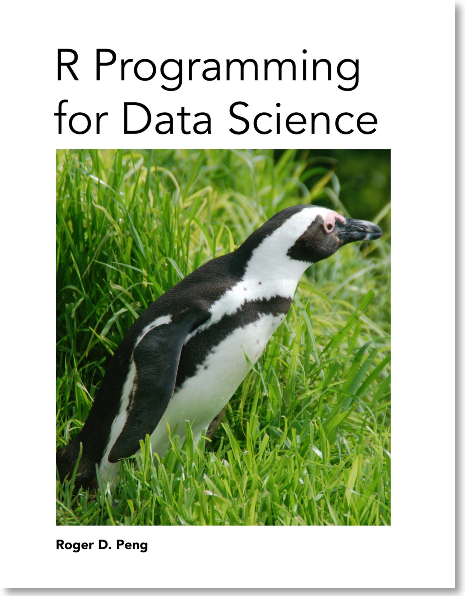
Based heavily on free e-books

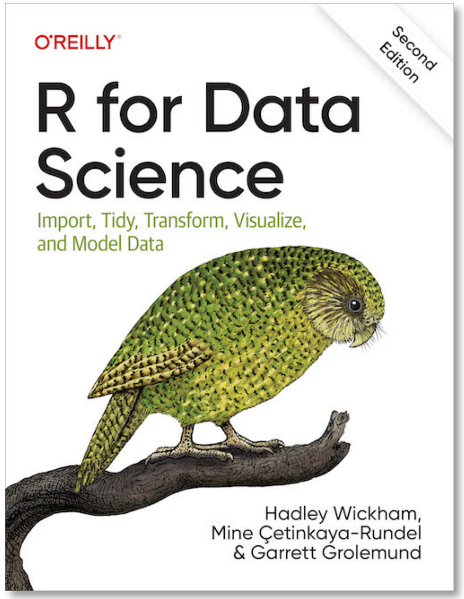
- Peng, R. (2016). *R Programming for Data Science*. Lulu.com.

  E-book available at: https://bookdown.org/rdpeng/rprogdatascience

- Wickham, H., Mine Cetinkaya-Rundel, & Grolemund, G. (2023). *R for data science: Import, tidy, transform, visualize, and model data* (2nd ed.). O’Reilly Media.

  E-book available at: https://r4ds.hadley.nz/





What is R?

What is R?

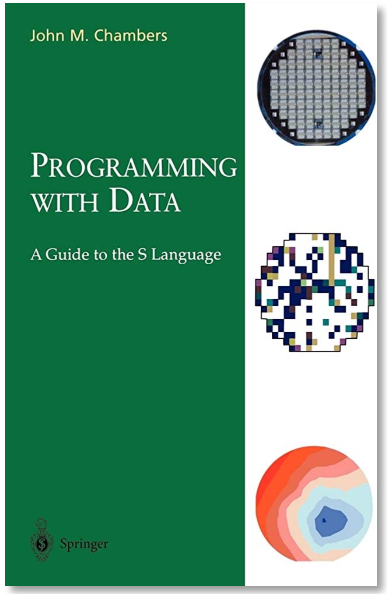
The quick answer is: R is a dialect of S

S is a language that was developed by John Chambers and others at the old Bell Telephone Laboratories (AT&T Corp.)

It was initiated in 1976 as an internal statistical analysis environment - originally implemented as *Fortran* libraries

In 1988, it was rewritten in *C* and began to resemble the system that we have today

1988



1998

The S Philosophy

"[W]e wanted *users* to be able to begin in an interactive environment, where they did not consciously think of themselves as programming. Then as their needs became clearer and their sophistication increased, they should be able to slide gradually into programming, when the language and system aspects would become more important."

Stages in the Evolution of S, John Chambers

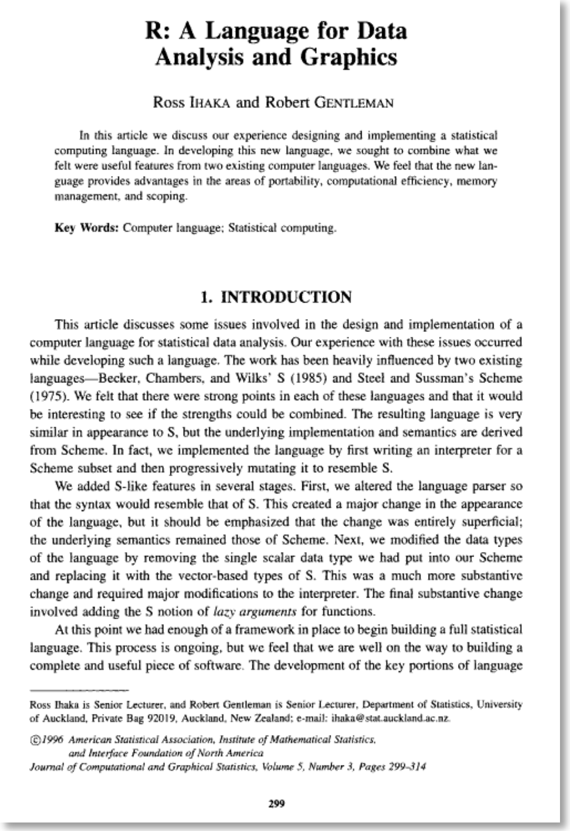
https://web.archive.org/web/20050226075706/http://www.stat.bell-labs.com/S/history.html

Back to R

- 1991: R was created by Ross Ihaka and Robert Gentleman in the Department of Statistics at the University of Auckland.
- 1993: The first announcement of R was made to the public.
- 1996: Paper published with author’s experience developing R

  Ross Ihaka and Robert Gentleman. *R: A language for data analysis and graphics*. Journal of Computational and Graphical Statistics, 5(3):299–314, 1996. https://doi.org/10.2307/1390807

- 1997: R Core Group was formed, and it still controls the source code
- 2000: R version 1.0.0 was released
- 2023: R version 4.3.0 is the latest



R: A Language for Data Analysis and Graphics

Ross Ihaka and Robert Gentleman

In this article we discuss our experience designing and implementing a statistical computing language. In developing this new language, we sought to combine what we felt were useful features from two existing computer languages. We feel that the new language provides advantages in the areas of portability, computational efficiency, memory management, and scoping.

Key Words: Computer language; Statistical computing.

1. INTRODUCTION

This article discusses some issues involved in the design and implementation of a computer language for statistical data analysis. Our experience with these issues occurred while developing such a language. The work has been heavily influenced by two existing languages—Becker, Chambers, and Wilks' S (1985) and Steel and Sussman's Scheme (1975). We felt that there were strong points in each of these languages and that it would be interesting to see if the strengths could be combined. The resulting language is very similar in appearance to S, but the underlying implementation and semantics are derived from Scheme. In fact, we implemented the language by first writing an interpreter for a Scheme subset and then progressively mutating it to resemble S.

We added S-like features in several stages. First, we altered the language parser so that the syntax would resemble that of S. This created a major change in the appearance of the language, but it should be emphasized that the change was entirely superficial; the underlying semantics remained those of Scheme. Next, we modified the data types of the language by removing the single scalar data type we had put into our Scheme and replacing it with the vector-based types of S. This was a much more substantive change and required major modifications to the interpreter. The final substantive change involved adding the S notion of *lazy arguments* for functions.

At this point we had enough of a framework in place to begin building a full statistical language. This process is ongoing, but we feel that we are well on the way to building a complete and useful piece of software. The development of the key portions of language

Ross Ihaka is Senior Lecturer, and Robert Gentleman is Senior Lecturer, Department of Statistics, University of Auckland, Private Bag 92019, Auckland, New Zealand; e-mail: ihaka@stat.auckland.ac.nz.

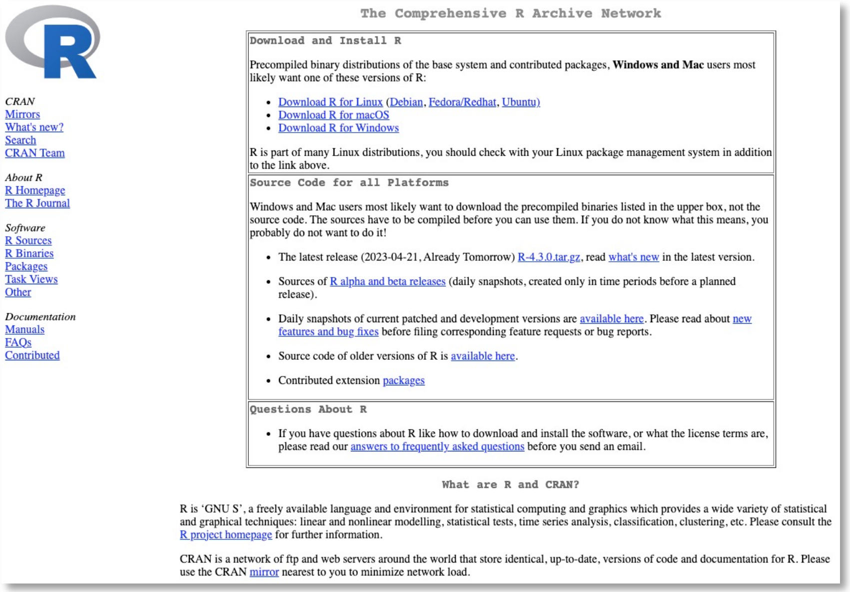
©1996 American Statistical Association, Institute of Mathematical Statistics, and Interface Foundation of North America
Journal of Computational and Graphical Statistics, Volume 5, Number 3, Pages 299–314

299

Design of the R System

The R system is divided into 2 conceptual parts:

- The "base" R system is available from CRAN: Comprehensive R Archive Network (https://cran.r-project.org).
- Everything else: many packages that can be used to extend the functionality of R.



Getting Started with R

Getting Started with R

Install R

https://cran.r-project.org

The Comprehensive R Archive Network

Download and Install R

Precompiled binary distributions of the base system and contributed packages, **Windows and Mac** users most likely want one of these versions of R:

- Download R for Linux (Debian, Fedora/Redhat, Ubuntu)
- Download R for macOS
- Download R for Windows

R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.

Getting Started with R

Install R

https://cran.r-project.org

R for Windows

Subdirectories:

base	Binaries for base distribution. This is what you want to **install R for the first time**.
contrib	Binaries of contributed CRAN packages (for R >= 3.4.x).
old contrib	Binaries of contributed CRAN packages for outdated versions of R (for R < 3.4.x).
Rtools	Tools to build R and R packages. This is what you want to build your own packages on Windows, or to build R itself.

Please do not submit binaries to CRAN. Package developers might want to contact Uwe Ligges directly in case of questions / suggestions related to Windows binaries.

You may also want to read the R FAQ and R for Windows FAQ.

Note: CRAN does some checks on these binaries for viruses, but cannot give guarantees. Use the normal precautions with downloaded executables.

Getting Started with R

Install R

R-4.3.0 for Windows

Download R-4.3.0 for Windows (79 megabytes, 64 bit)
README on the Windows binary distribution
New features in this version

This build requires UCRT, which is part of Windows since Windows 10 and Windows Server 2016. On older systems, UCRT has to be installed manually from here.

If you want to double-check that the package you have downloaded matches the package distributed by CRAN, you can compare the md5sum of the .exe to the fingerprint on the master server.

Frequently asked questions

- Does R run under my version of Windows?
- How do I update packages in my previous version of R?

Please see the R FAQ for general information about R and the R Windows FAQ for Windows-specific information.

Other builds

- Patches to this release are incorporated in the r-patched snapshot build.
- A build of the development version (which will eventually become the next major release of R) is available in the r-devel snapshot build.
- Previous releases

Note to webmasters: A stable link which will redirect to the current Windows binary release is
<CRAN MIRROR>/bin/windows/base/release.html.

Last change: 2023-04-21

Getting Started with R

Install R

R for macOS

This directory contains binaries for the base distribution and of R and packages to run on macOS. R and package binaries for R versions older than 4.0.0 are only available from the CRAN archive so users of such versions should adjust the CRAN mirror setting (`https://cran-archive.r-project.org`) accordingly.

Note: Although we take precautions when assembling binaries, please use the normal precautions with downloaded executables.

R 4.3.0 "Already Tomorrow" released on 2023/04/21

Please check the integrity of the downloaded package by checking the signature:

```
pkgutil --check-signature R-4.3.0.pkg
```

in the *Terminal* application. If Apple tools are not avaiable you can check the SHA1 checksum of the downloaded image:

```
openssl sha1 R-4.3.0.pkg
```

Latest release:

For Apple silicon (M1/M2) Macs:
R-4.3.0-arm64.pkg
SHA1-hash: 8ee0276daa9841993f218ebd2a8a7aa86c00d470
(ca. 90MB, notarized and signed)

or

For older Intel Macs:
R-4.3.0-x86_64.pkg
SHA1-hash: d28e528c8e3ee761aa4b871a8d444a1bfbee9bd3
(ca. 92MB, notarized and signed)

R 4.3.0 binary for macOS 11 (**Big Sur**) and higher, signed and notarized packages.

Contains R 4.3.0 framework, R.app GUI 1.79, Tcl/Tk 8.6.12 X11 libraries and Texinfo 6.8. The latter two components are optional and can be ommitted when choosing "custom install", they are only needed if you want to use the `tcltk` R package or build package documentation from sources.

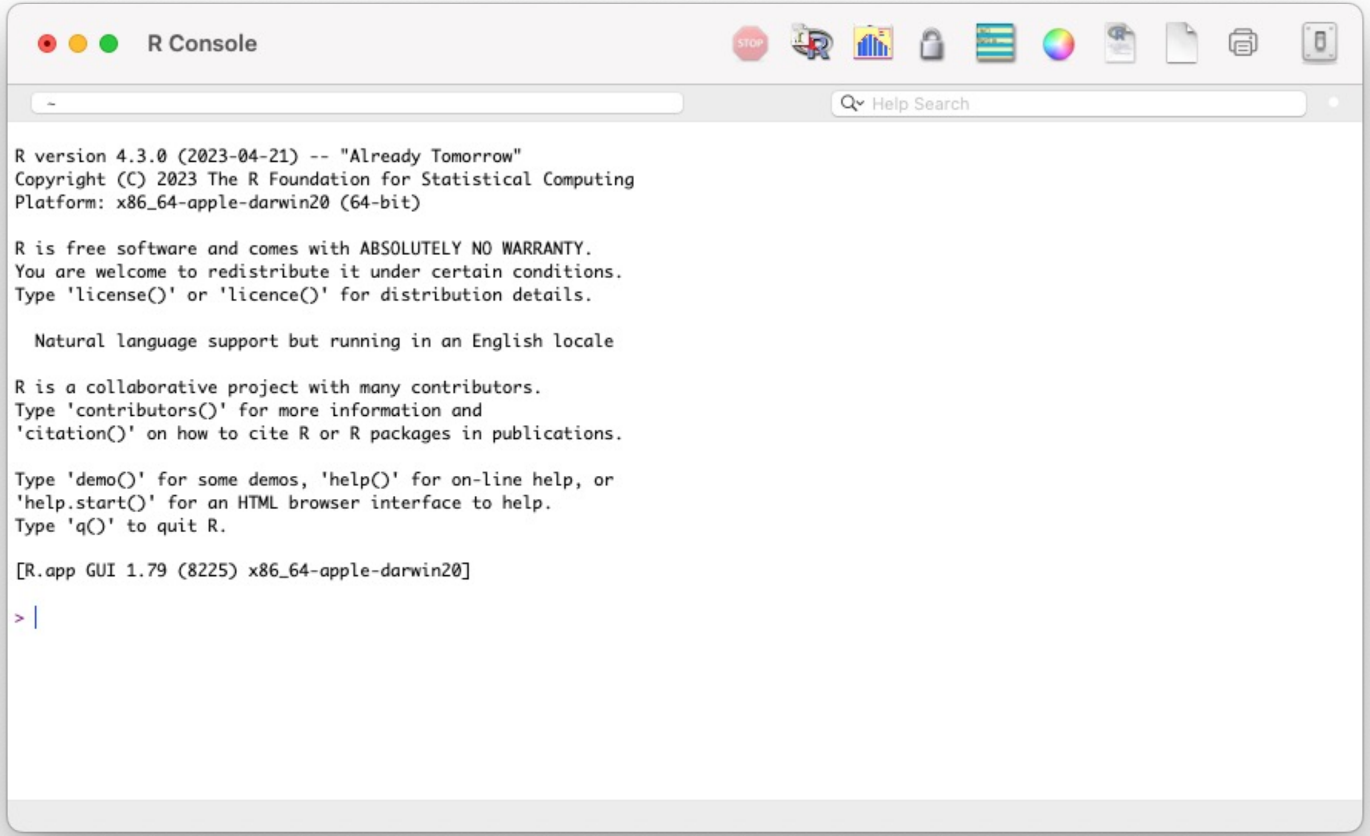
macOS Ventura users: there is a known bug in Ventura preventing installations from some locations without a prompt. If the installation fails, move the downloaded file away from the *Downloads* folder (e.g., to your home or Desktop)

Note: the use of X11 (including `tcltk`) requires XQuartz (version 2.8.5 or later). Always re-install XQuartz when upgrading your macOS to a new major version.

This release uses Xcode 14.2/14.3 and GNU Fortran 12.2. If you wish to compile R packages which contain Fortran code, you may need to download the corresponding GNU Fortran compiler from https://mac.R-project.org/tools. Any external libraries and tools are expected to live in `/opt/R/arm64` (Apple silicon) or `/opt/R/x86_64` (Intel).

Getting Started with R

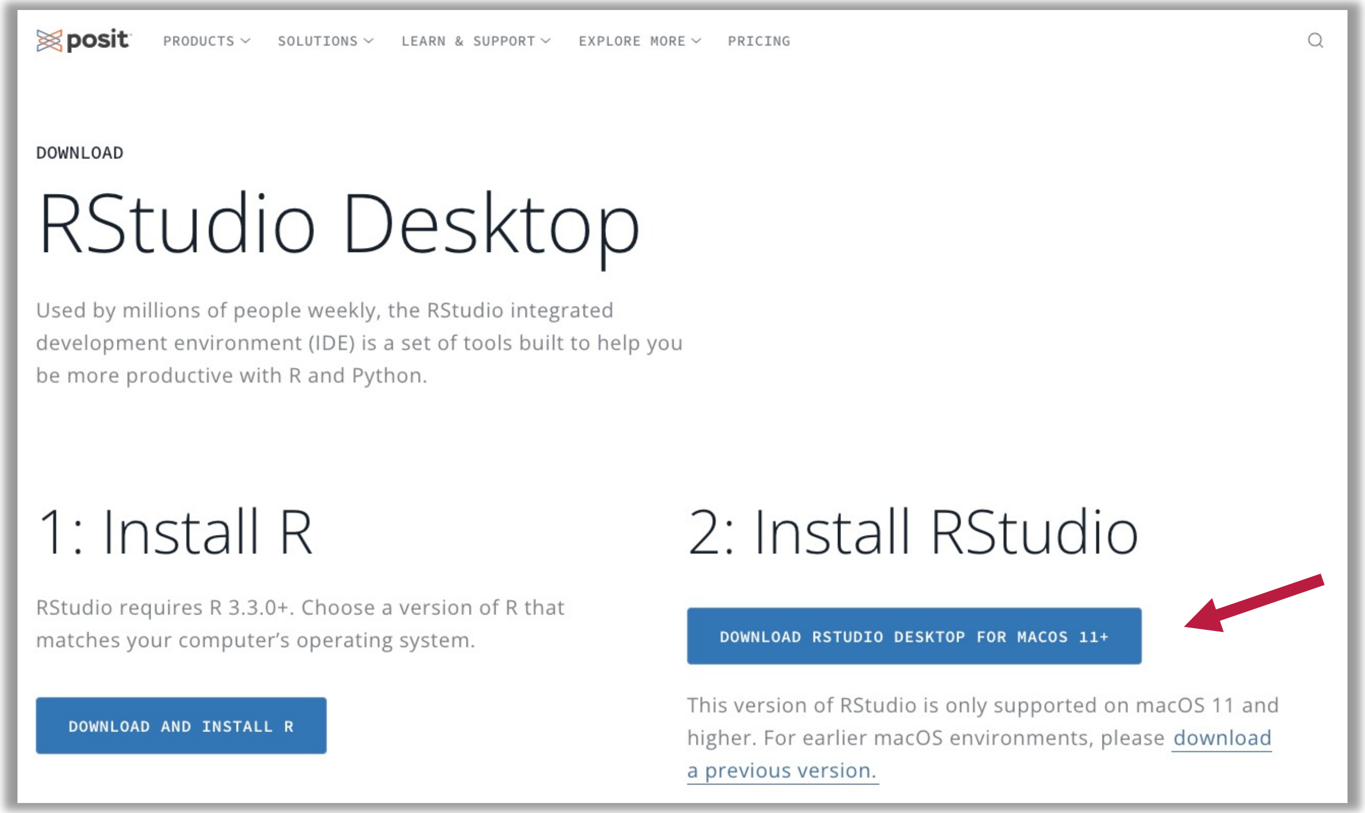
R Console: Not a very user-friendly interface



Getting Started with RStudio

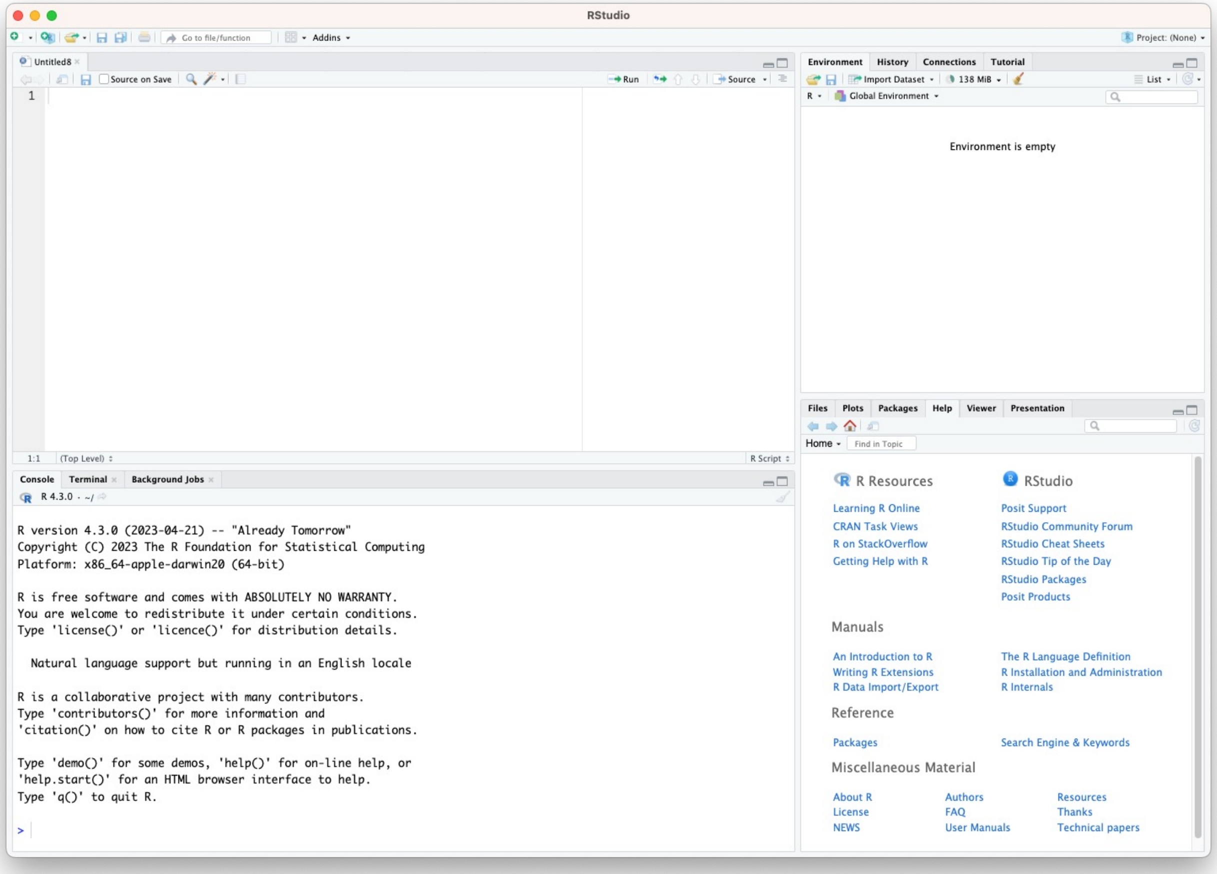
Install RStudio Desktop

https://posit.co/download/rstudio-desktop



Getting Started with RStudio

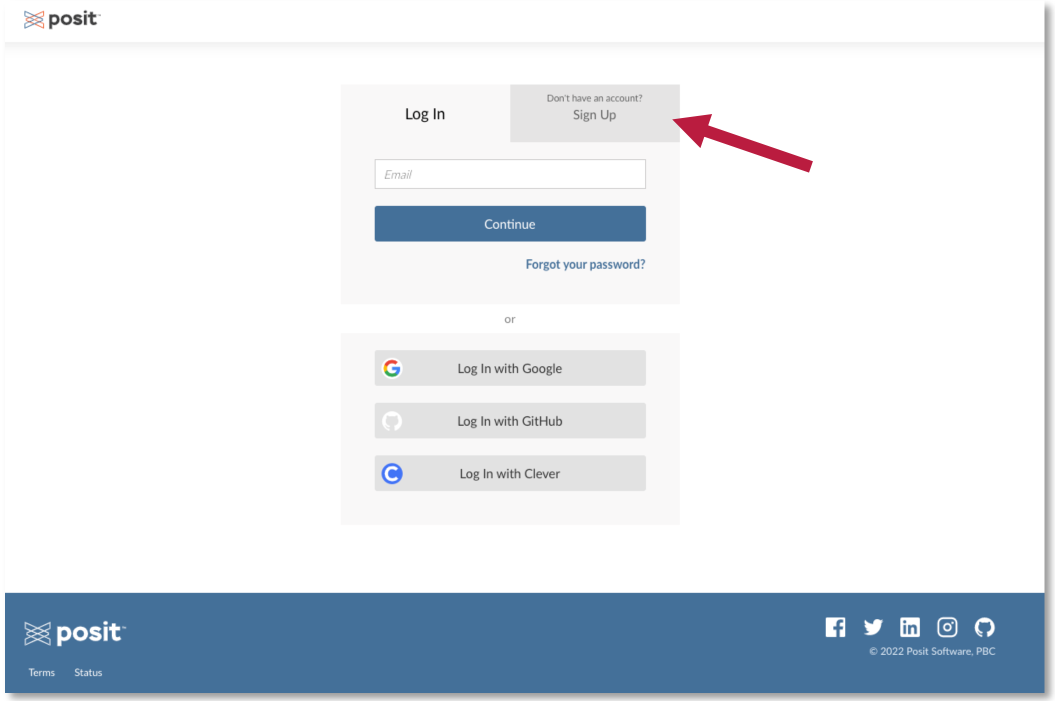
Install RStudio Desktop



Getting Started with RStudio

Another option: Posit Cloud (RStudio on the Cloud)

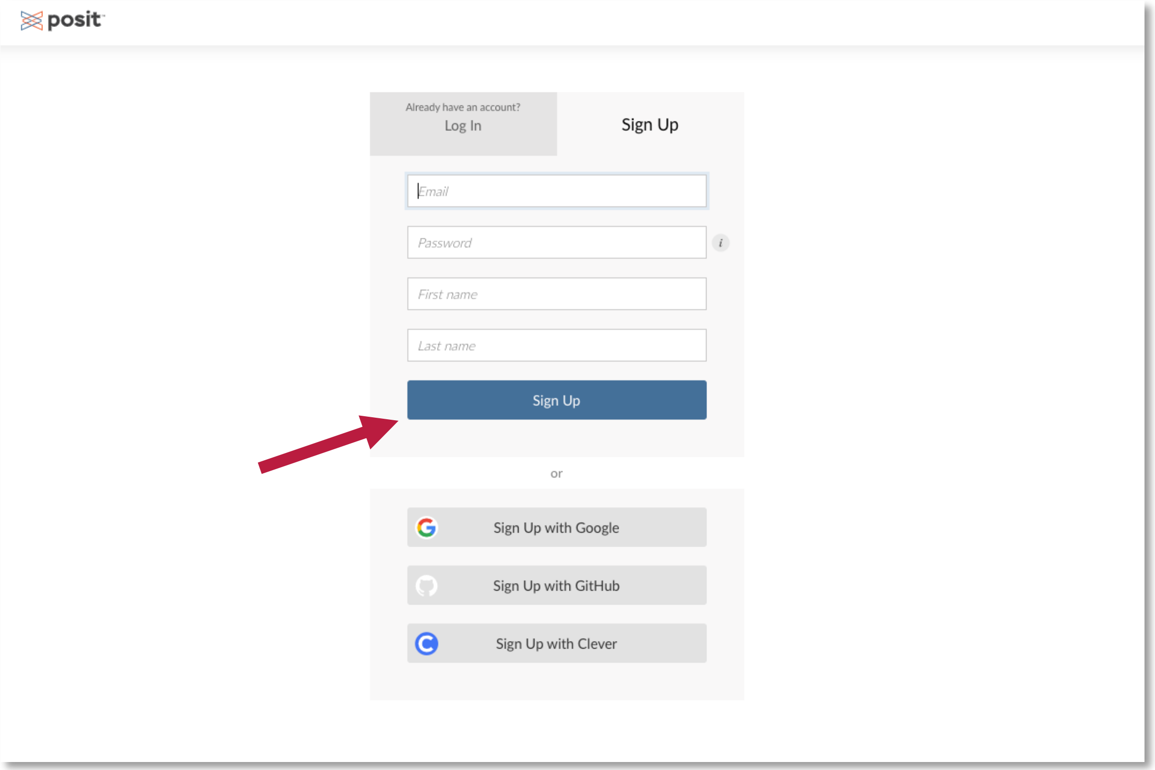
https://posit.cloud



Getting Started with RStudio

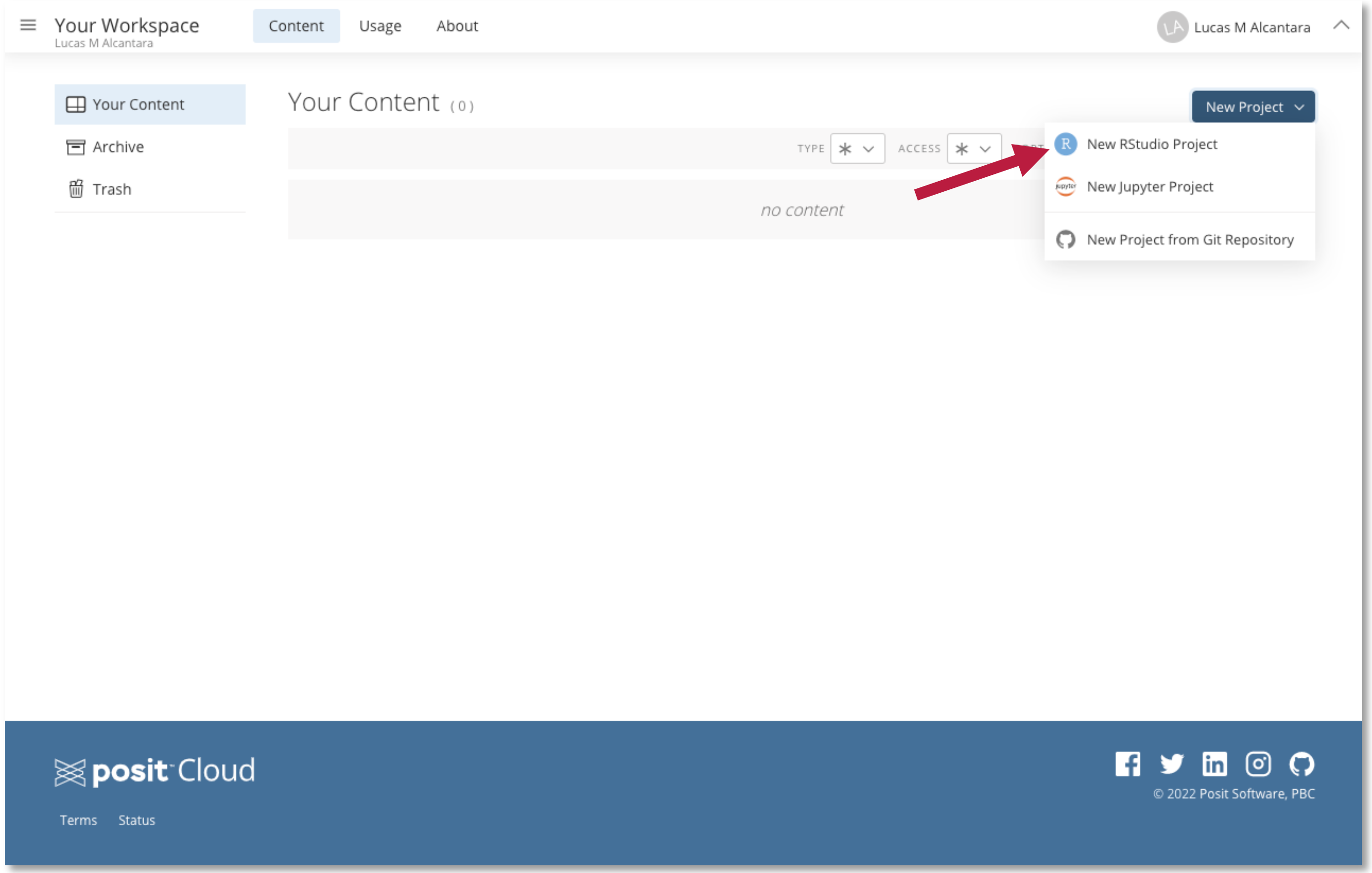
Another option: Posit Cloud (RStudio on the Cloud)

https://posit.cloud



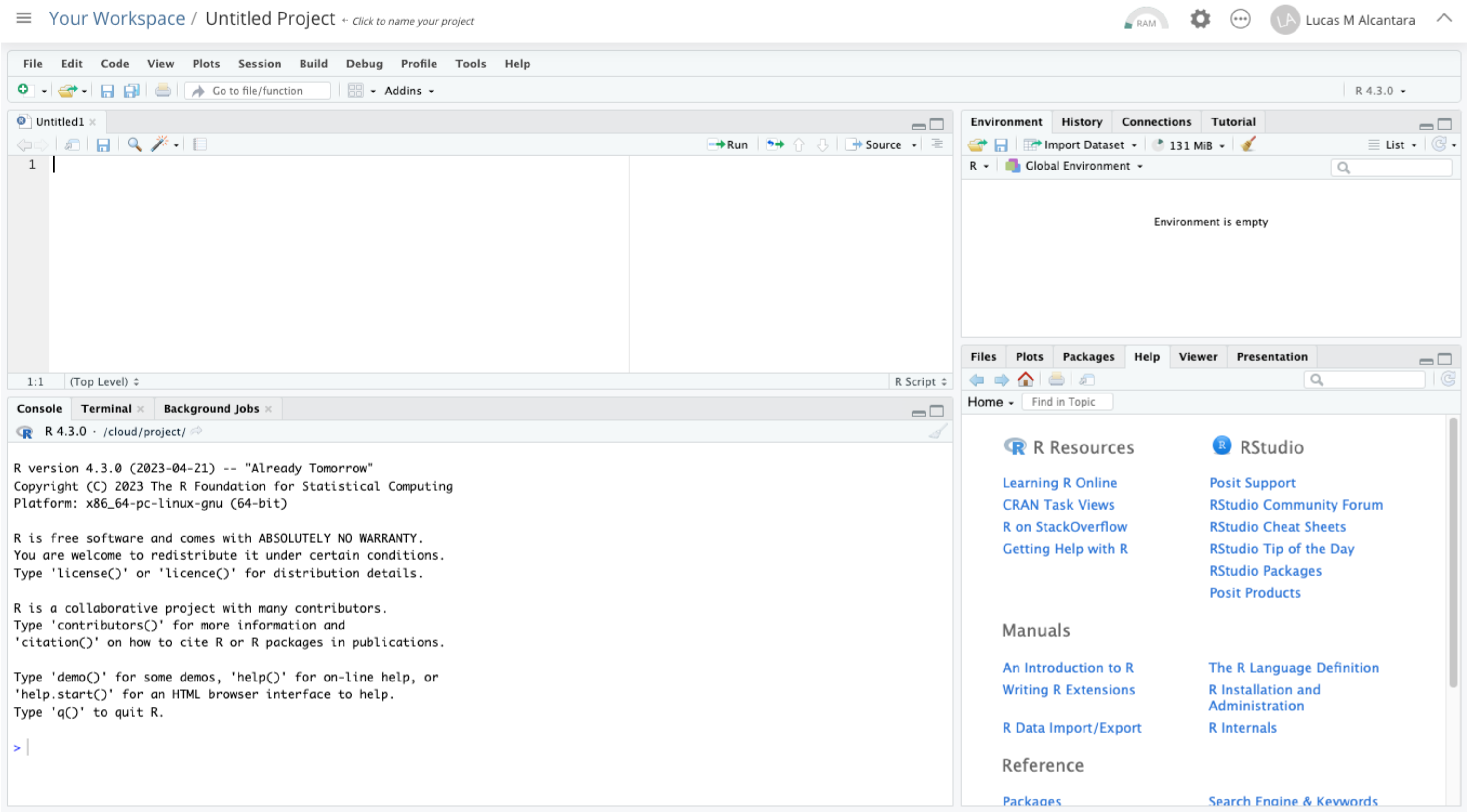
Getting Started with RStudio

Another option: Posit Cloud (RStudio on the Cloud)



Getting Started with RStudio

Another option: Posit Cloud (RStudio on the Cloud)



R Nuts and Bolts

Entering Input

We type expressions on the R console. The '<-' symbol is the assignment operator.

```
> x <- 1
> print(x)
[1] 1
> x
[1] 1
> msg <- "hello"
```

The # character indicates a comment. Anything to the right of the # is ignored (including the # itself).

```
x <-  ## Incomplete expression
```

Evaluation

When a complete expression is entered at the prompt, it is evaluated, and the result of the evaluated expression is returned. The result may be *auto-printed*.

```
> x <- 5   ## nothing printed
> x        ## auto-printing occurs
[1] 5
> print(x)  ## explicit printing
[1] 5
```

The [1] shown on the output above indicates that 'x' is a vector and 5 is its first element.

```
> x <- 11:30
> x
 [1] 11 12 13 14 15 16 17 18 19 20 21 22
[13] 23 24 25 26 27 28 29 30
```

R Objects

R has five basic or “atomic” classes of objects:

- character
- numeric (real numbers)
- integer
- complex
- logical (True/False)

The most basic type of R object is a vector.

-> A *vector* can only contain objects of the same class.

Exception: list

Numbers

Numbers in R are generally treated as numeric objects, i.e., double precision real numbers

- 1 = 1.00
- 2 = 2.00

If you explicitly want an integer, you need to specify the 'L' suffix

- 1 = numeric object
- 1L = integer object

Numbers

There is also a special number 'Inf' which represents infinity

- 1/0 = Inf
- -1/0 = -Inf
- 1/Inf = 0

The value 'NaN' represents an undefined value ("not a number")

- 0 / 0 = NaN

Attributes

Attributes are like metadata for the R object. If any, it can be accessed using the 'attributes()' function

- dimensions (e.g., matrices, arrays)
- class (e.g., integer, numeric)
- length
- other user-defined attributes/metadata
- etc.

Not all R objects contain attributes, in which case the attributes() function returns NULL.

Creating Vectors

The 'c()' function can be used to create vectors of objects by concatenating things together.

```
> x <- c(0.5, 0.6)        ## numeric
> x <- c(TRUE, FALSE)     ## logical
> x <- c(T, F)            ## logical
> x <- c("a", "b", "c")   ## character
> x <- 9:29               ## integer
> x <- c(1+0i, 2+4i)      ## complex
```

Mixing Objects

What will be the class of y on each of the following codes?

```
> y <- c(1.7, "a")
> y <- c(TRUE, 2)
> y <- c("a", TRUE)
```

Mixing Objects

What will be the class of y on each of the following codes?

```
> y <- c(1.7, "a")   ## character
> y <- c(TRUE, 2)    ## numeric
> y <- c("a", TRUE)  ## character
```

When different objects are mixed in a vector, coercion occurs so that every element in the vector is of the same class.

Explicit Coercion

Objects can be explicitly coerced from one class to another using the 'as.*' functions, if available.

```
> x <- 0:6
> class(x)
[1] "integer"
> as.numeric(x)
[1] 0 1 2 3 4 5 6
> as.logical(x)
[1] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
> as.character(x)
[1] "0" "1" "2" "3" "4" "5" "6"
```

Explicit Coercion

When R can't figure out how to coerce an object, it can result in NAs being introduced by coercion. A warning message is usually shown by R:

```
> x <- c("a", "b", "c")
> as.numeric(x)
Warning: NAs introduced by coercion
[1] NA NA NA
> as.logical(x)
[1] NA NA NA
> as.complex(x)
Warning: NAs introduced by coercion
[1] NA NA NA
```

Matrices

Matrices are vectors with a dimension attribute. The dimension attribute is itself an integer vector of length 2 (number of rows, number of columns)

```
> m <- matrix(nrow = 2, ncol = 3)
> m
     [,1] [,2] [,3]
[1,]   NA   NA   NA
[2,]   NA   NA   NA
> dim(m)
[1] 2 3
> attributes(m)
$dim
[1] 2 3
```

Matrices

Matrices are constructed *column-wise*, so entries can be thought of starting in the “upper left” corner (1,1) and running down the columns (1,2; 2,1; 2,2…).

```
> m <- matrix(1:6, nrow = 2, ncol = 3)
> m
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

Matrices

Matrices can also be created directly from vectors by adding a dimension attribute.

```
> m <- 1:10
> m
 [1]  1  2  3  4  5  6  7  8  9 10
> dim(m) <- c(2, 5)
> m
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10
```

Matrices

Matrices can be created by column-binding or row-binding with the 'cbind()' and 'rbind()' functions.

```
> x <- 1:3
> y <- 10:12
> cbind(x, y)
     x  y
[1,] 1 10
[2,] 2 11
[3,] 3 12
> rbind(x, y)
  [,1] [,2] [,3]
x    1    2    3
y   10   11   12
```

Lists

Lists are a special type of vector that can contain elements of different classes.

```
> x <- list(1, "a", TRUE, 1 + 4i)
> x
[[1]]
[1] 1

[[2]]
[1] "a"

[[3]]
[1] TRUE

[[4]]
[1] 1+4i
```

Lists

Lists are a special type of vector that can contain elements of different classes.

```
> x <- list(1, "a", TRUE, 1 + 4i)
> x
[[1]]
[1] 1

[[2]]
[1] "a"

[[3]]
[1] TRUE

[[4]]
[1] 1+4i
```

Based on the functions you learned, how can you tell what the classes of each of these objects are?

Factors

Factors are used to represent categorical data and can be unordered or ordered. "Integer vector labelled in alphabetical order"

```
> x <- factor(c("yes", "yes", "no", "yes", "no"))
> x
[1] yes yes no  yes no
Levels: no yes
> table(x)
x
 no yes
  2   3
> ## See the underlying representation of factor
> unclass(x)
[1] 2 2 1 2 1
attr(,"levels")
[1] "no"  "yes"
```

Factors

The order of the levels of a factor can be set using the ‘levels’ argument to factor().

```
> x <- factor(c("yes", "yes", "no", "yes", "no"))
> x  ## Levels are put in alphabetical order
[1] yes yes no  yes no
Levels: no yes
> x <- factor(c("yes", "yes", "no", "yes", "no"),
+             levels = c("yes", "no"))
> x
[1] yes yes no  yes no
Levels: yes no
```

Missing Values

Missing values are denoted by NA or NaN for q undefined mathematical operations.

- is.na() is used to test if objects are NA
- is.nan() is used to test if objects are NaN
- NA values have a class (integer NA, character NA, etc.)
- NaN values are NA, but NOT the other way around

Missing Values

```r
> ## Create a vector with NAs in it
> x <- c(1, 2, NA, 10, 3)
> ## Return a logical vector indicating which elements are NA
> is.na(x)
[1] FALSE FALSE  TRUE FALSE FALSE
> ## Return a logical vector indicating which elements are NaN
> is.nan(x)
[1] FALSE FALSE FALSE FALSE FALSE

> ## Now create a vector with both NA and NaN values
> x <- c(1, 2, NaN, NA, 4)
> is.na(x)
[1] FALSE FALSE  TRUE  TRUE FALSE
> is.nan(x)
[1] FALSE FALSE  TRUE FALSE FALSE
```

Data Frames

Data frames are used to store tabular data in R. It’s a special type of list where every element of the list must have the same length.

- Columns = Elements of the list
- Number of rows = Length of each element of the list

Data frames can store different classes of objects in each column.

Data frames have both column and row names

Data Frames

Data frames can be created using the 'data.frame()' function

```
> x <- data.frame(foo = 1:4, bar = c(T, T, F, F))
> x
  foo   bar
1   1  TRUE
2   2  TRUE
3   3 FALSE
4   4 FALSE
> nrow(x)
[1] 4
> ncol(x)
[1] 2
```

Names

R objects can have names, which is very useful for writing readable code and self-describing objects.

```
> x <- 1:3
> names(x)
NULL
> names(x) <- c("New York", "Seattle", "Los Angeles")
> x
   New York     Seattle Los Angeles
          1           2           3
> names(x)
[1] "New York"    "Seattle"     "Los Angeles"
```

Names

R objects can have names, which is very useful for writing readable code and self-describing objects.

```
> x <- list("Los Angeles" = 1, Boston = 2, London = 3)
> x
$`Los Angeles`
[1] 1

$Boston
[1] 2

$London
[1] 3
> names(x)
[1] "Los Angeles" "Boston"      "London"
```

Names

R objects can have names, which is very useful for writing readable code and self-describing objects.

```
> m <- matrix(1:4, nrow = 2, ncol = 2)
> dimnames(m) <- list(c("a", "b"), c("c", "d"))
> m
  c d
a 1 3
b 2 4

> colnames(m) <- c("h", "f")
> rownames(m) <- c("x", "z")
> m
  h f
x 1 3
z 2 4
```

Getting Data in and Out of R

Reading and Writing Data

There are a few ways to read and write data into R.

Target	Read function	Write Function
Tabular data	read.table(), read.csv(), etc.	write.table(), write.csv(), etc.
Lines	readLines()	writeLines()
R code files	source()	-
Workspace files	load()	save()

Reading Data Files with read.table()

The read.table() function is one of the most used functions for reading data.

Let's look at the Help page: ?read.table

Reading Data Files with read.table()

A few tips on what you can do to read your data faster:

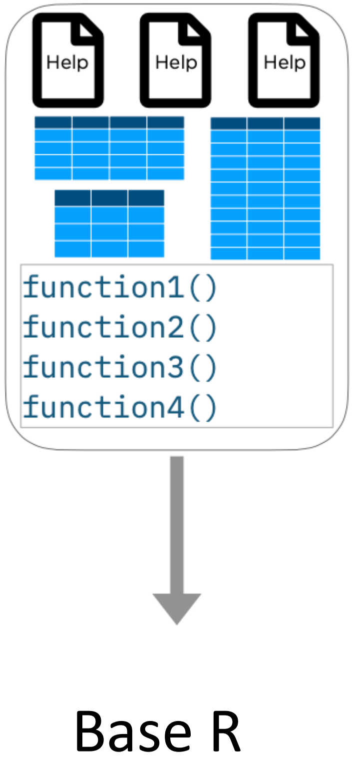
- Set the argument comment.char = ""
- Set the argument stringsAsFactors = FALSE
- Use the argument colClasses
  - Give the specific classes

```
> initial <- read.table("datatable.txt", nrows = 100)
> classes <- sapply(initial, class)
> tabAll <- read.table("datatable.txt", colClasses = classes)
```

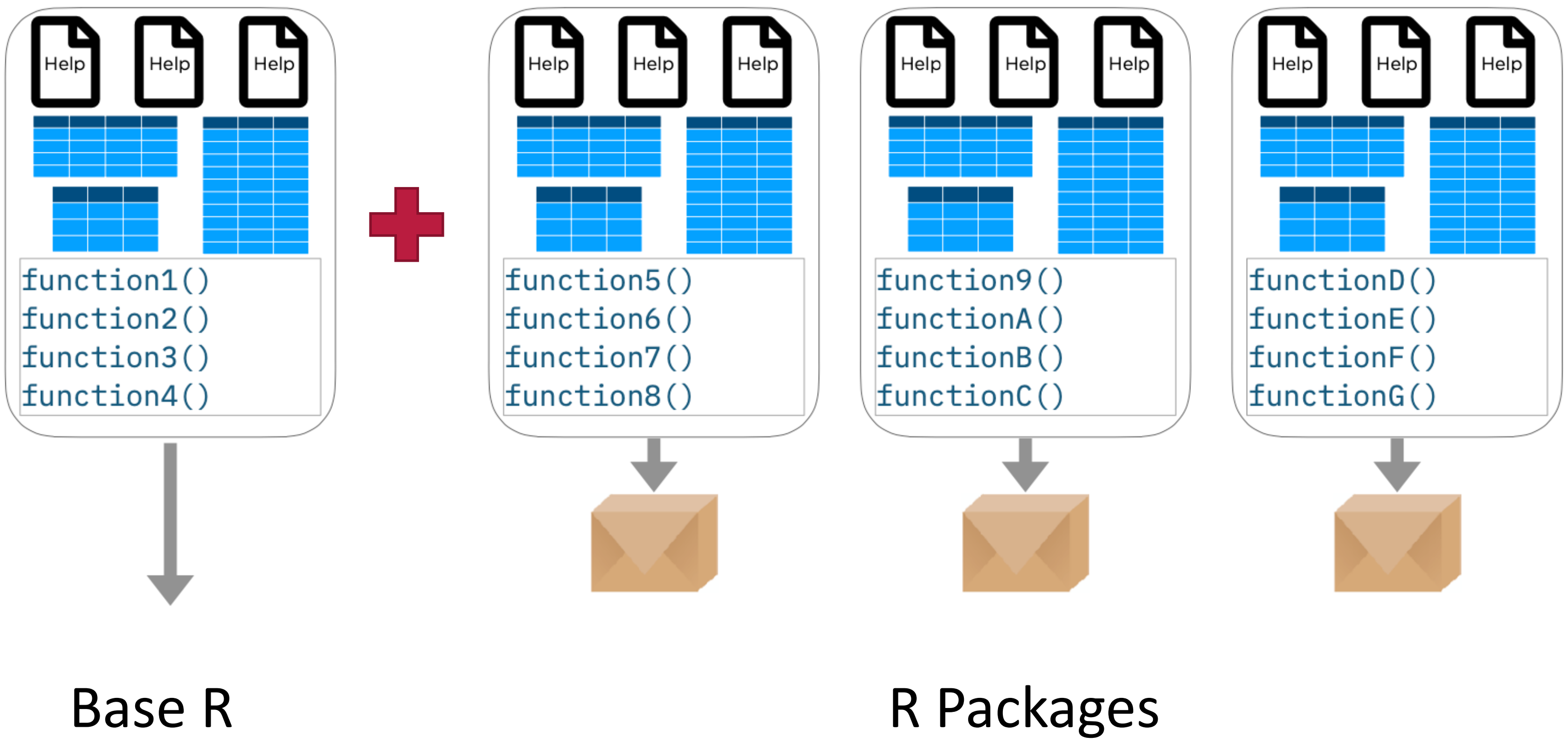
  - Assume all columns have the same class, e.g., character
- Use faster functions from different packages

R Packages

R Packages


Help
Help
Help
function1()
function2()
function3()
function4()
Base R

R Packages


Help
Help
Help
function1()
function2()
function3()
function4()
Help
Help
Help
function5()
function6()
function7()
function8()
Help
Help
Help
function9()
functionA()
functionB()
functionC()
Help
Help
Help
functionD()
functionE()
functionF()
functionG()
Base R
R Packages

R Packages on the Console

Install a package: once per computer/user

- install.packages("readr")

Load a package: once per R session

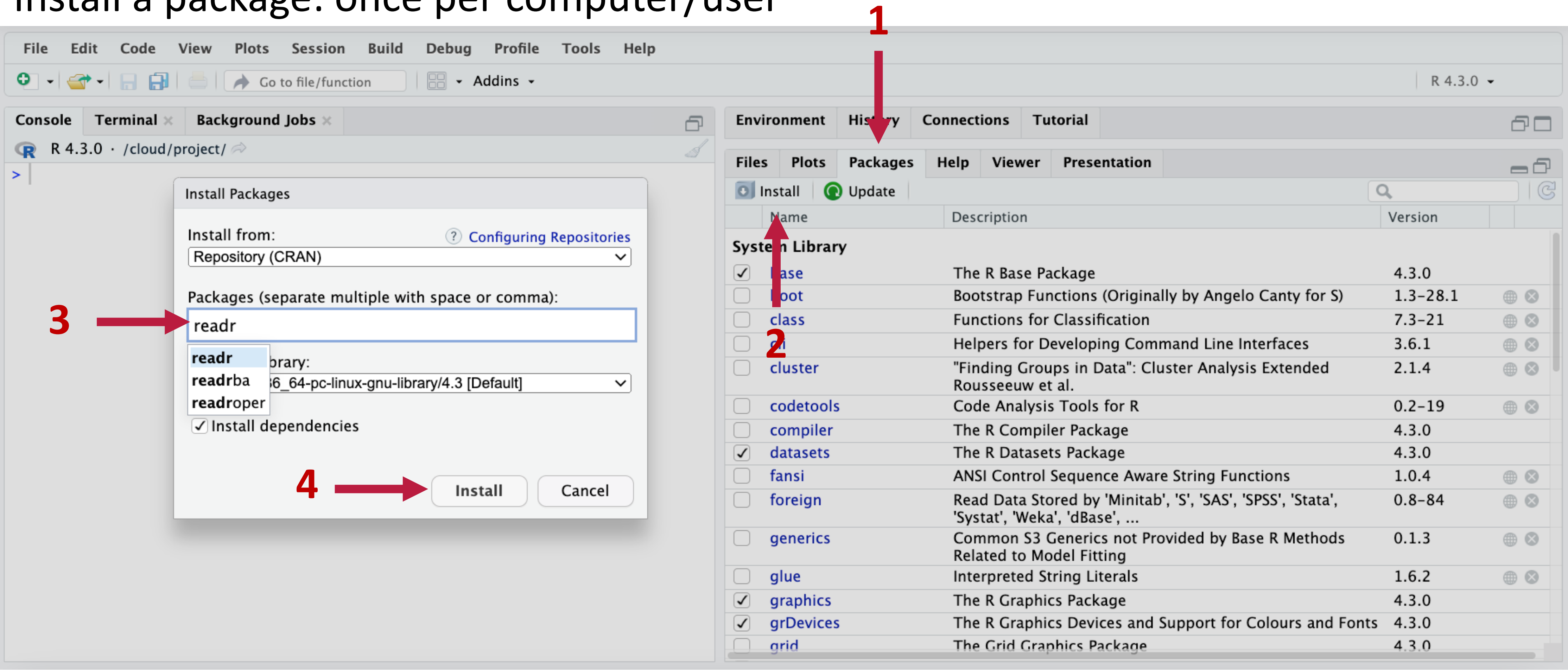
- library(readr)

Remove a package: once per computer/user

- remove.packages("readr")

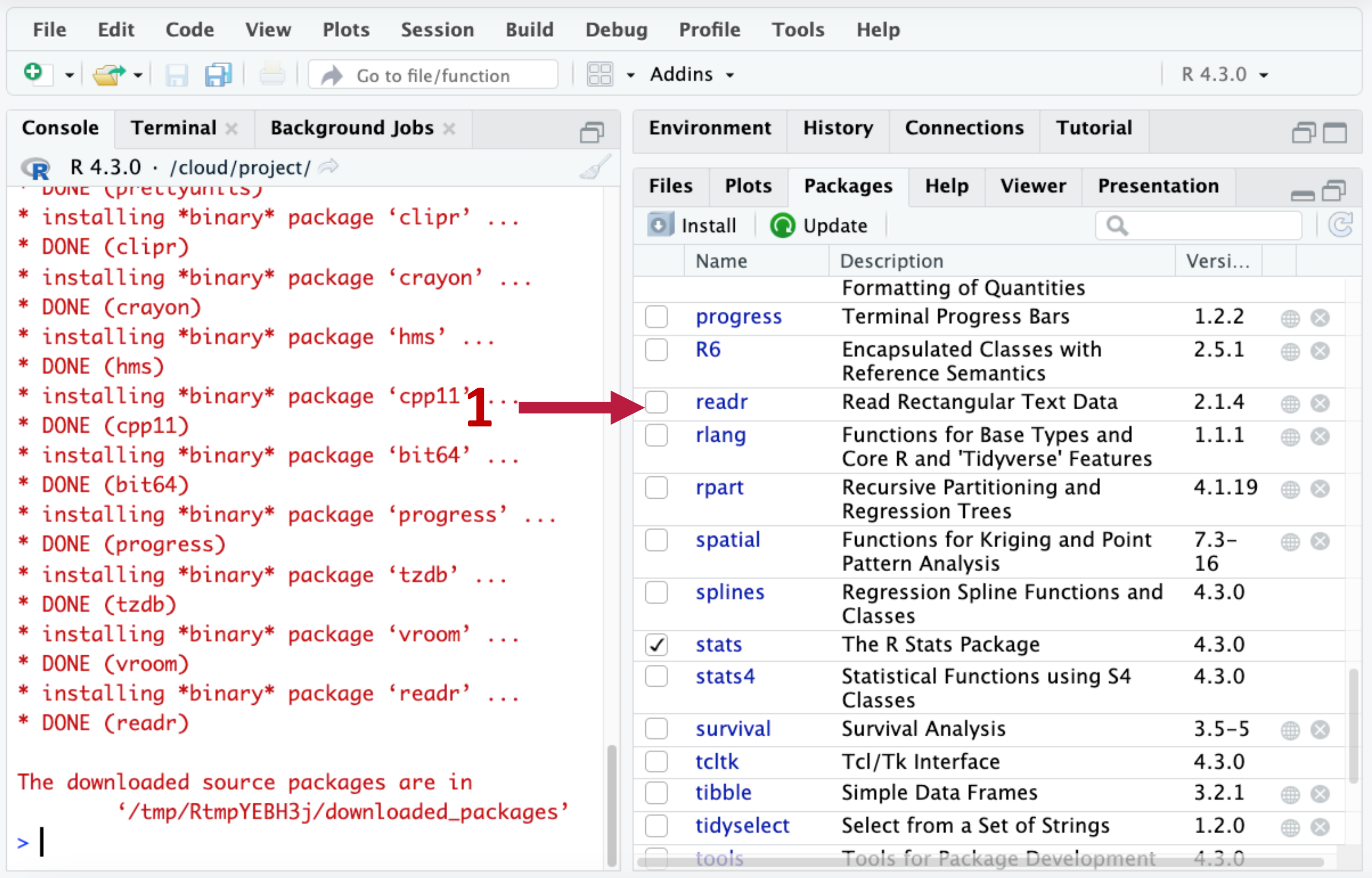
R Packages on RStudio

Install a package: once per computer/user



R Packages on RStudio

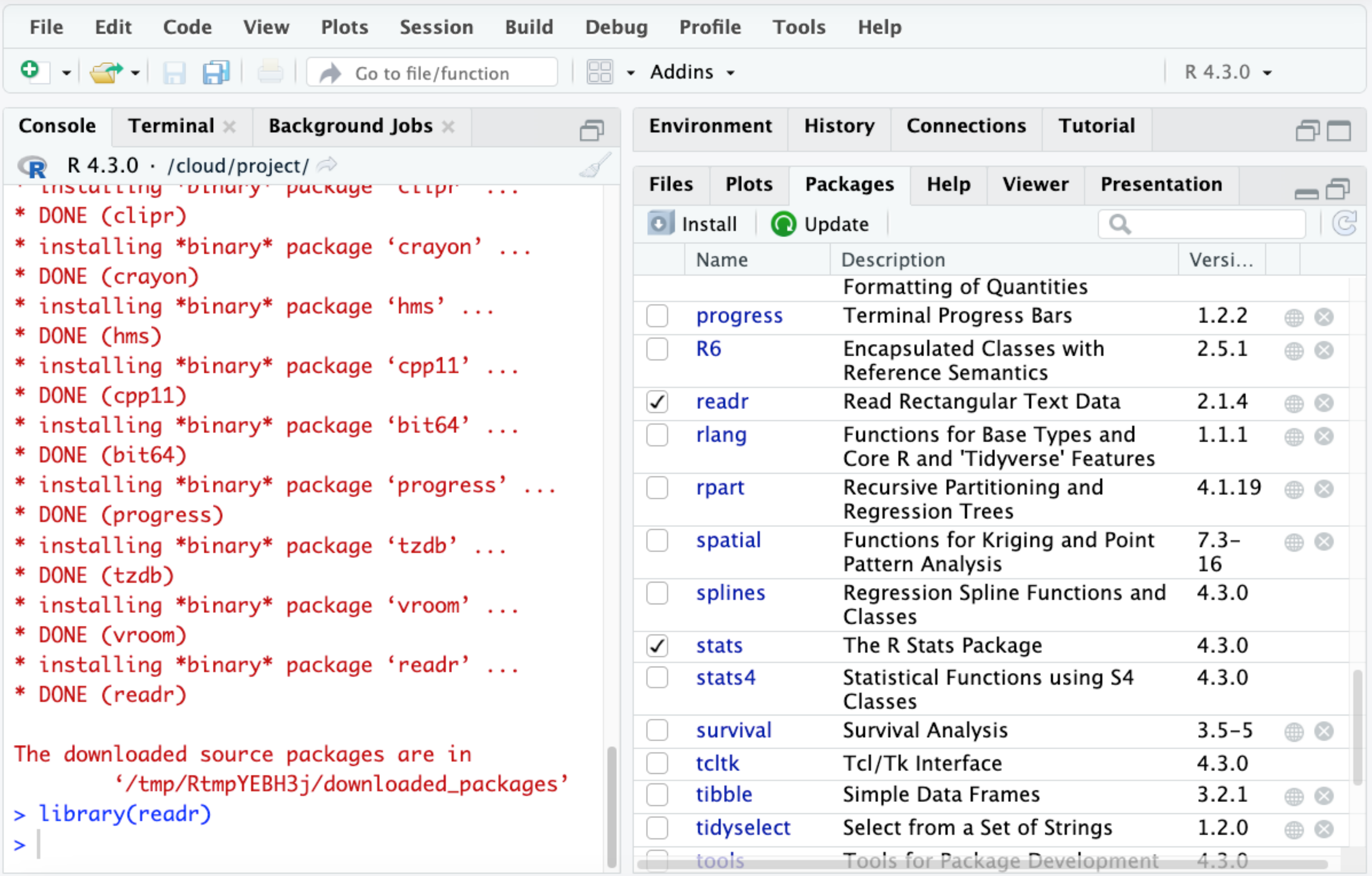
Load a package: once per R session



ADC

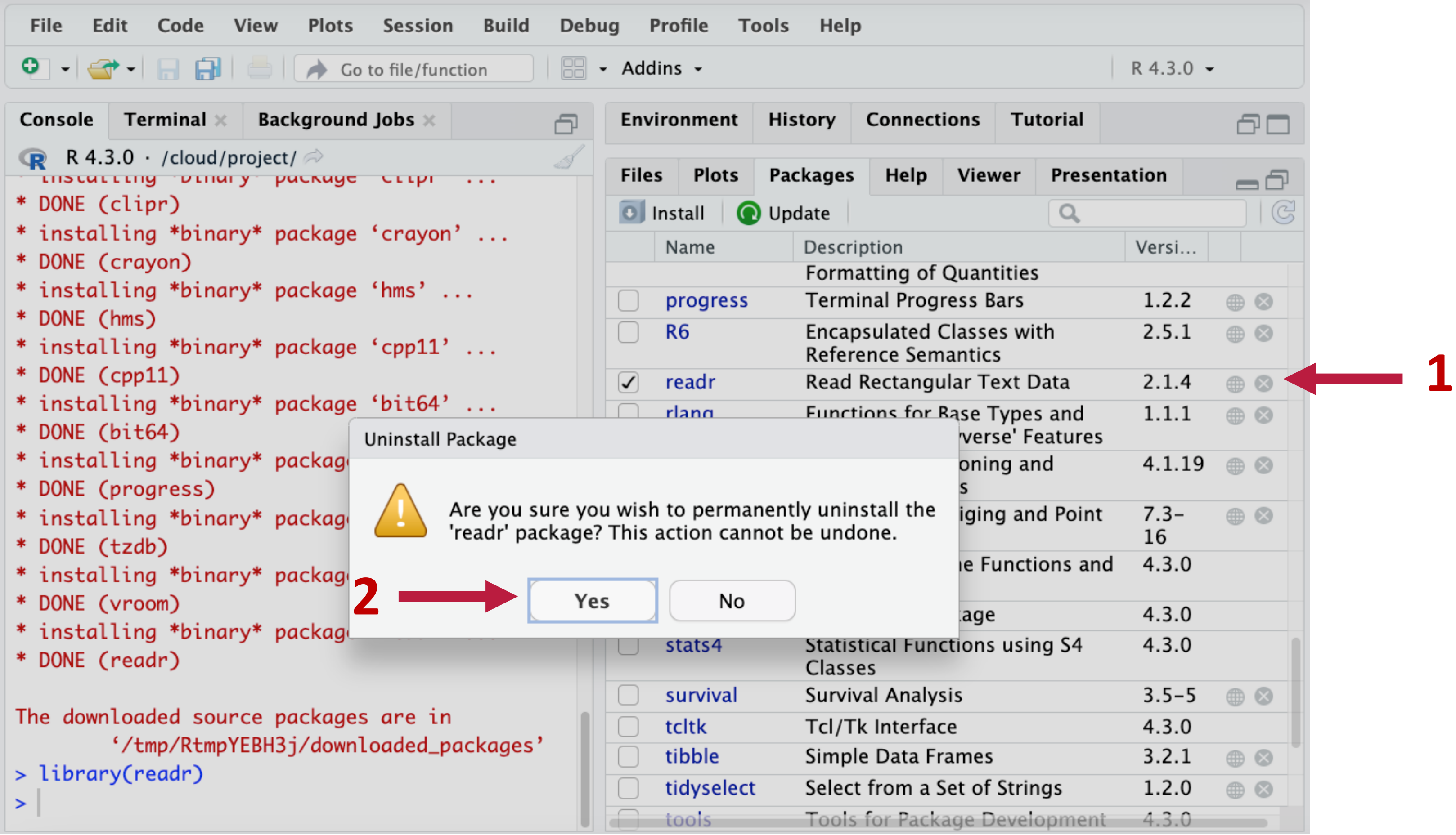
R Packages on RStudio

Load a package: once per R session



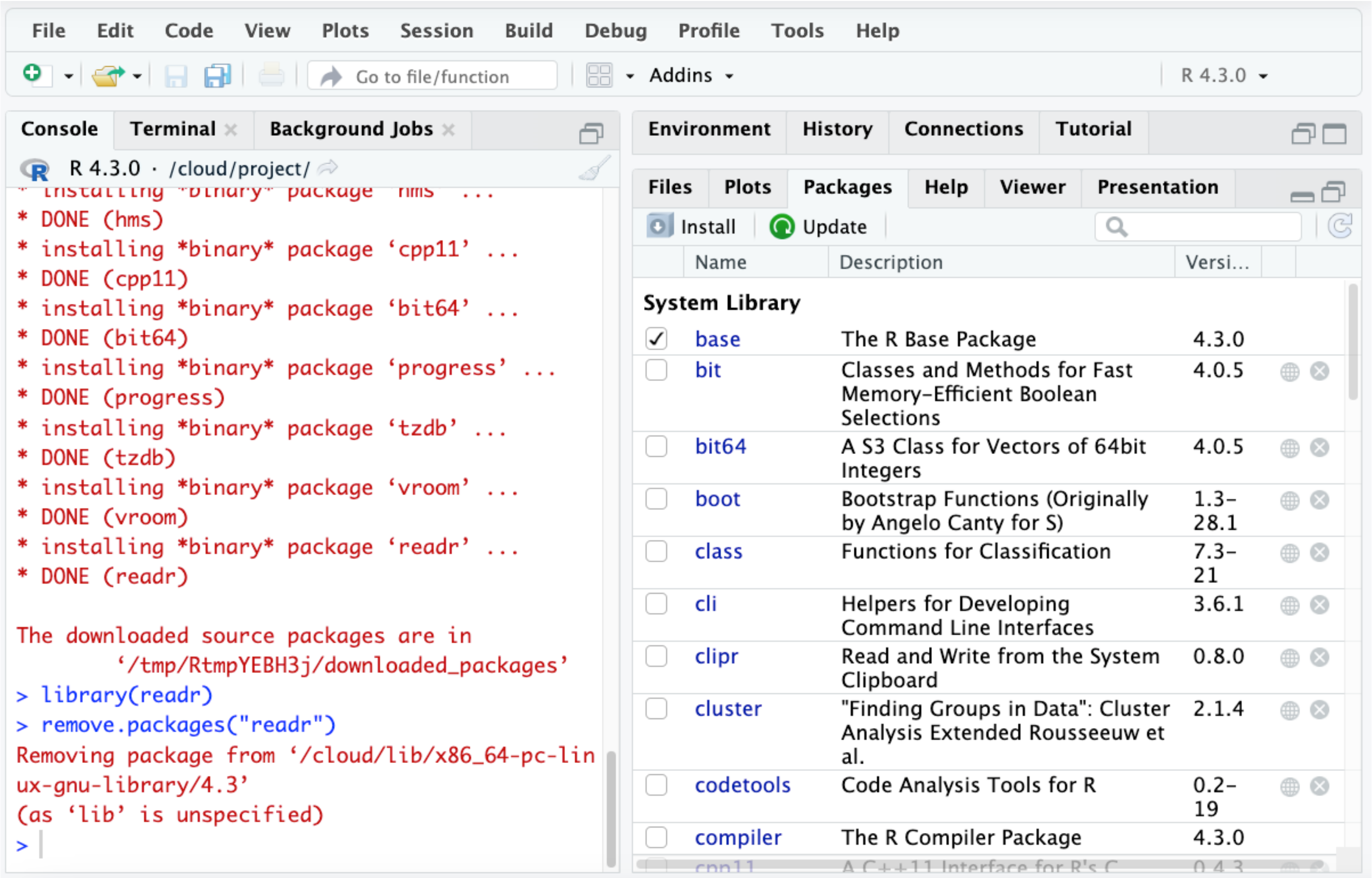
R Packages on RStudio

Remove a package: once per computer/user



R Packages on RStudio

Remove a package: once per computer/user



Conflicting R Packages

Different/Similar functions might have the same name in different packages

- Install and load the ‘dplyr’ package
- Same function name, different packages
- Load order DO matter!
  - Detach both ‘stats’ and ‘dplyr’ packages
  - Load ‘dplyr’ first and ‘stats’ later. Did you notice any difference?
- Try to be specific to which package you want to use to avoid conflicts
  - stats::filter()
  - dplyr::filter()

TIP: Use the ‘conflicted’ package to manage conflicts

R Packages

Loading a package silently:

- suppressPackageStartupMessages(library(dplyr))
- Use it with caution!

Getting Data in and Out of R

Reading Data Files with the readr package

Popular read functions: read_table(), read_csv(), read_fwf

Advantage of using readr functions to read data files

- Easier debugging: warnings indicate which rows/observations triggered them
- Faster reading: automatically guesses column types from the first 1k lines only
- Reads compressed files automatically
- Nice user-oriented features:
  - Progress bar when reading big files
  - Short data description
- R objects from readr are tibbles, not base R's data frames

  *More on this later

Reading Multiple Data Files

Sometimes we have data saved in multiple files (e.g., one file per day)

What's a quick way to read all of them into one R Object?

```
# Load data.table package
library(data.table)

# Create a list of files in a directory
file_list <- list.files(".")

# Apply the fread function to each of those files and store in a list
data_list <- lapply(file_list, fread)

# Bind rows in a list
data <- data.table::rbindlist(data_list)
```

*Use data from the insentec folder

Writing Data Out of R

Like reading, many functions are available to write data out of R. A few examples:

- write.csv | write_csv
- write.table | write_table
- fwrite

Most of them you start specifying the object followed by the file path:

- write.csv(data, “my/path/to/data.csv”)

Common arguments include:

- Column/row names (T/F); Quote (T/F); Encoding (utf8, latin-1, etc.);
- Field separator (comma, space, tab, pipe, etc.); String for NA values

Subsetting R Objects

Common Subsetting Operators

There are three operators that can be used to extract subsets of R objects.

- The [operator
  - Always returns an object of the same class as the original
  - It can be used to select multiple elements of an object
- The [[operator
  - Is used to extract elements of a list or a data frame
  - It can only be used to extract a single element
  - The class of the returned object will not necessarily be a list or data frame
- The $ operator
  - Is used to extract elements of a list or data frame by literal name
  - Its semantics are similar to [[

Subsetting a Vector

The [operator can be used to extract multiple elements of a vector by passing the operator an integer, an integer sequence, or a logical sequence.

```
> x <- c("a", "b", "c", "c", "d", "a")
> x[1]
[1] "a"
> x[2]
[1] "b"

> x[1:4]
[1] "a" "b" "c" "c"

> x[c(1, 3, 4)]
[1] "a" "c" "c"

> x[x > "a"]
[1] "b" "c" "c" "d"
```

Subsetting a Matrix

Matrices can be subsetted in the usual way with (*i,j*) type indices. One of the indices can be missing.

```
> x <- matrix(1:6, 2, 3)
> x
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

```
> x[1, 2]
[1] 3
> x[2, 1]
[1] 2
```

```
> x[1, ]  ## Extract the first row
[1] 1 3 5
> x[, 2]  ## Extract the second column
[1] 3 4
```

```
> x[1, , drop = FALSE]
     [,1] [,2] [,3]
[1,]    1    3    5
```

Subsetting a List

Lists in R can be subsetted using all three operators, each used for a different purpose.

```
> x <- list(foo = 1:4, bar = 0.6)
> x
$foo
[1] 1 2 3 4

$bar
[1] 0.6

> x[[1]]
[1] 1 2 3 4
> x[["bar"]]
[1] 0.6
> x$bar
[1] 0.6
```

*Notice you don't need the quotes when you use the $ operator.

Subsetting a List

The $ operator can only be used with literal names.

```
> x <- list(foo = 1:4, bar = 0.6, baz = "hello")
> name <- "foo"
>
> ## computed index for "foo"
> x[[name]]
[1] 1 2 3 4
>
> ## element "name" doesn’t exist! (but no error here)
> x$name
NULL
>
> ## element "foo" does exist
> x$foo
[1] 1 2 3 4
```

Subsetting Nested Elements of a List

The [[operator can take an integer sequence if you want to extract a nested element.

```
> x <- list(a = list(10, 12, 14), b = c(3.14, 2.81))
>
> ## Get the 3rd element of the 1st element
> x[[c(1, 3)]]
[1] 14
>
> ## Same as above
> x[[1]][[3]]
[1] 14
>
> ## 1st element of the 2nd element
> x[[c(2, 1)]]
[1] 3.14
```

Subsetting Multiple Elements of a List

The [operator can be used to extract multiple elements from a list.

```
> x <- list(foo = 1:4, bar = 0.6, baz = "hello")
> x[c(1, 3)]
$foo
[1] 1 2 3 4

$baz
[1] "hello"
```

*Note that x[c(1, 3)] is NOT the same as x[[c(1, 3)]].

Removing NA Values

A common task in data analysis is removing missing values (NAs).

```
> x <- c(1, 2, NA, 4, NA, 5)
> bad <- is.na(x)
> print(bad)
[1] FALSE FALSE  TRUE FALSE  TRUE FALSE
> x[!bad]
[1] 1 2 4 5
```

Removing NA Values

What if there are multiple R objects and you want to take the subset with no missing values in any of those objects?

```
> x <- c(1, 2, NA, 4, NA, 5)
> y <- c("a", "b", NA, "d", NA, "f")
> good <- complete.cases(x, y)
> good
[1]  TRUE  TRUE FALSE  TRUE FALSE  TRUE
> x[good]
[1] 1 2 4 5
> y[good]
[1] "a" "b" "d" "f"
```

*Note that both vectors must have the same lengths

Removing NA Values

We can also use complete.cases() with data frames

```
> head(airquality)
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
2    36     118  8.0   72     5   2
3    12     149 12.6   74     5   3
4    18     313 11.5   62     5   4
5    NA      NA 14.3   56     5   5
6    28      NA 14.9   66     5   6
> good <- complete.cases(airquality)
> head(airquality[good, ])
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
2    36     118  8.0   72     5   2
3    12     149 12.6   74     5   3
4    18     313 11.5   62     5   4
7    23     299  8.6   65     5   7
8    19      99 13.8   59     5   8
```

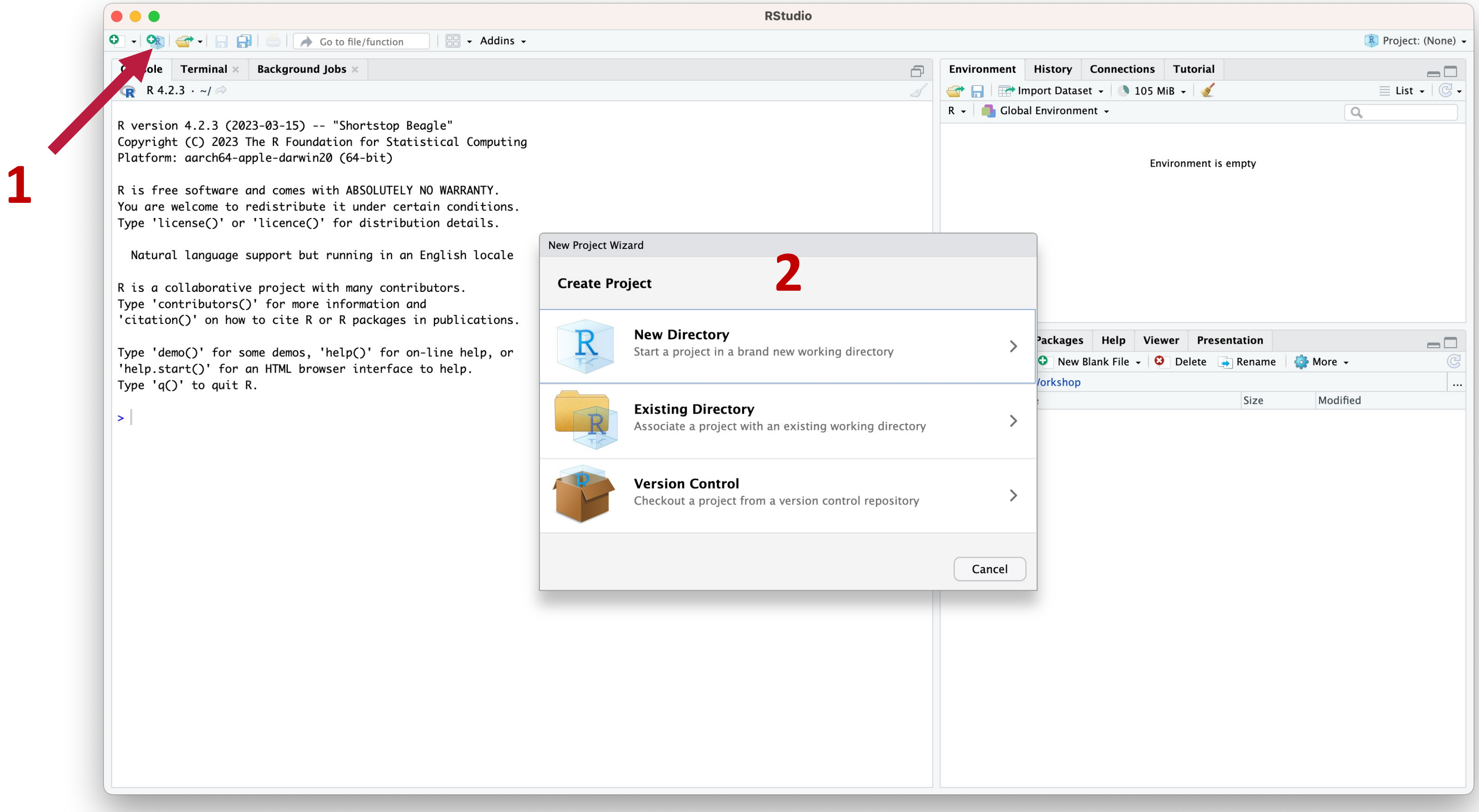
*Or just use drop_na() from the tidyr package:

```
> head(airquality)
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
2    36     118  8.0   72     5   2
3    12     149 12.6   74     5   3
4    18     313 11.5   62     5   4
5    NA      NA 14.3   56     5   5
6    28      NA 14.9   66     5   6
> tidyr::drop_na(head(airquality))
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
2    36     118  8.0   72     5   2
3    12     149 12.6   74     5   3
4    18     313 11.5   62     5   4
```

Project Time!

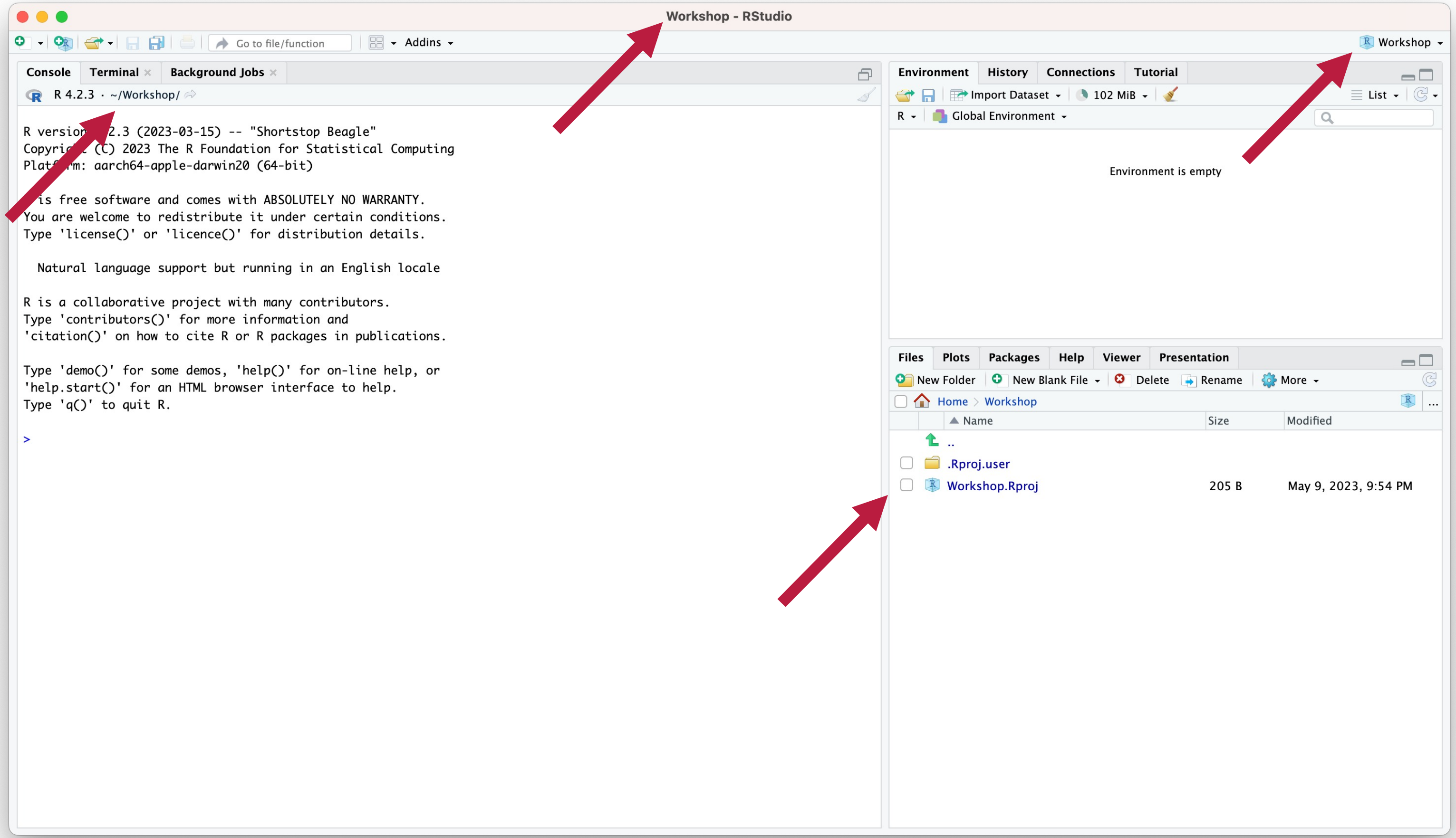
RStudio Projects

RStudio projects make it straightforward to divide your work into multiple contexts, each with their own working directory, workspace, history, and source documents.



RStudio Projects

RStudio projects make it straightforward to divide your work into multiple contexts, each with their own working directory, workspace, history, and source documents.



RStudio Projects

When a project is opened within RStudio the following actions are taken:

- A new R session (process) is started
- The .Rprofile file in the project's main directory (if any) is sourced by R
- The .RData file in the project's main directory is loaded (if project options indicate that it should be loaded).
- The .Rhistory file in the project's main directory is loaded into the RStudio History pane (and used for Console Up/Down arrow command history).
- The current working directory is set to the project directory.
- Previously edited source documents are restored into editor tabs
- Other RStudio settings (e.g., active tabs, splitter positions, etc.) are restored to where they were the last time the project was closed.

https://support.posit.co/hc/en-us/articles/200526207-Using-RStudio-Projects

Reproducible Environments with the renv Package

Isolated

- Installing a new or updated package for one project won’t break your other projects, and vice versa. That’s because renv gives each project its own private package library.

Portable

- Easily transport your projects from one computer to another, even across different platforms. renv makes it easy to install the packages your project depends on.

Reproducible

- renv records the exact R and package versions you depend on, and ensures those exact versions are the ones that get installed wherever you go.

https://docs.posit.co/ide/user/ide/guide/environments/r/renv.html

renv General Workfow

1. Call renv::init() to initialize a new project-local environment with a private R library,
2. Work in the project as normal, installing and removing new R packages as they are needed in the project,
3. Call renv::snapshot() to save the state of the project library to the lockfile (called renv.lock),
4. Continue working on your project, installing and updating R packages as needed.
5. Call renv::snapshot() again to save the state of your project library if your attempts to update R packages were successful
6. Call renv::restore() to revert to the previous state as encoded in the lockfile if your attempts to update packages introduced some new problems.

https://docs.posit.co/ide/user/ide/guide/environments/r/renv.html

renv Cache Location

Platform	Location
Linux	~/.cache/R/renv
macOS	~/Library/Caches/org.R-project.R/R/renv
Windows	%LOCALAPPDATA%/R/cache/R/renv

THANK YOU!

Questions?

Feel free to reach me later at
alcantal@uoguelph.ca

AGRI-FOOD DATA CANADA
AT THE UNIVERSITY of GUELPH