

RStudio + Data i/o with R

EC 425/525, Lab 3

Edward Rubin
26 April 2019

Prologue

Schedule

Last time

Working with data in R—especially via `dplyr`.

Today

1. RStudio basics
2. Getting data in and out of R.

Review

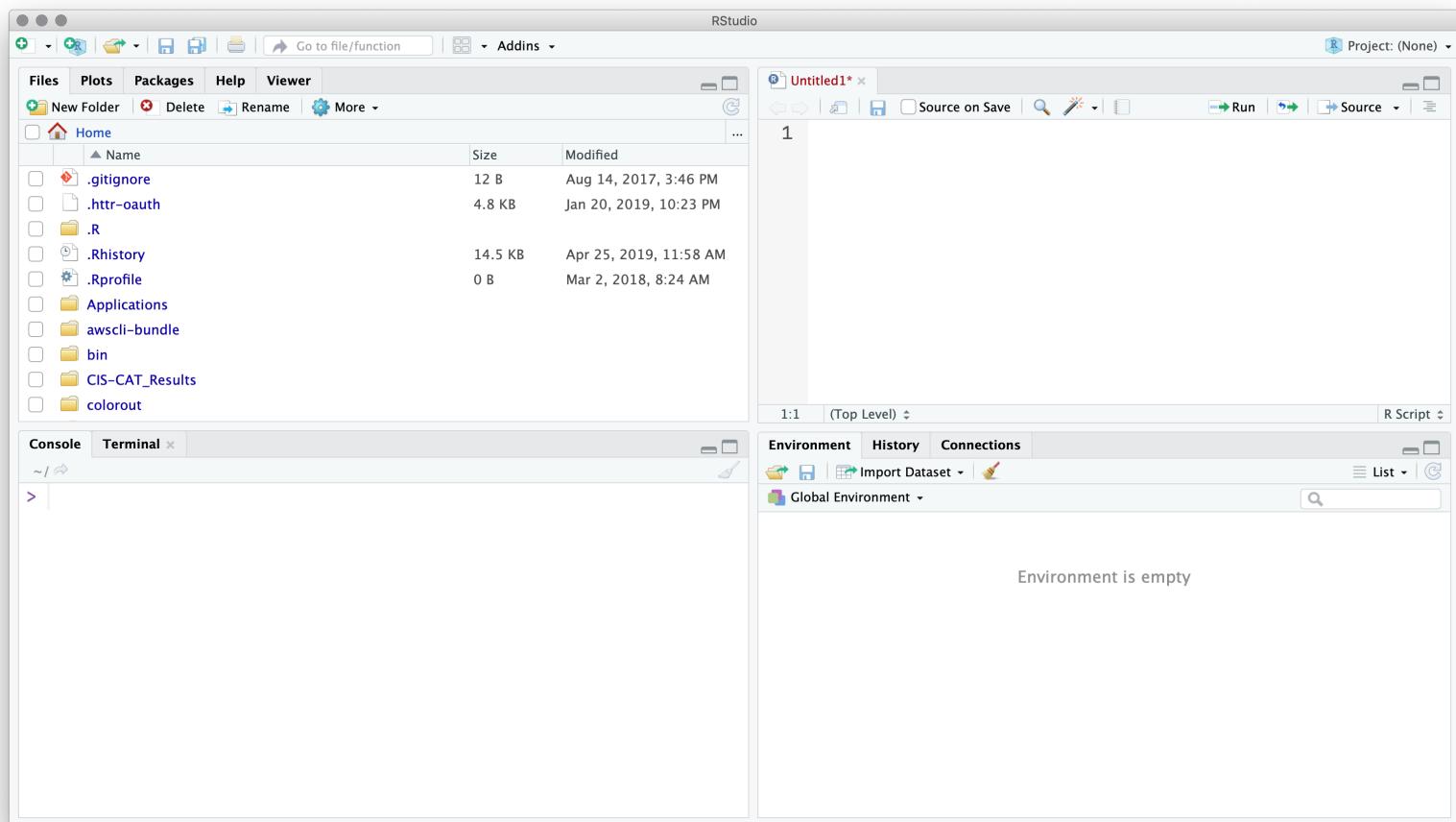
Key points from the last lab(s).

1. `dplyr` is your data-work friend.
2. Pipes (`%>%`) make your life easier.[†]

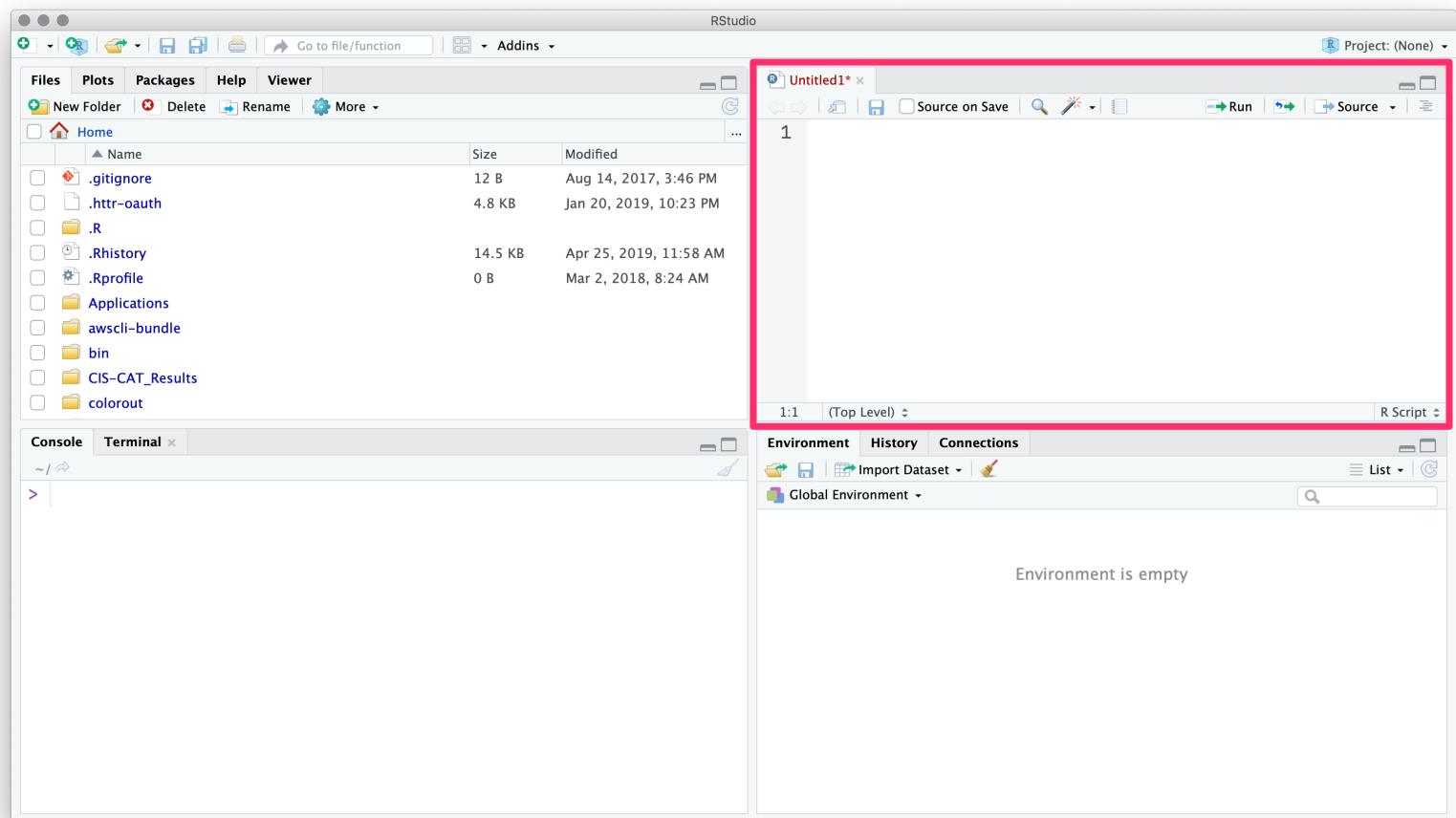
[†] Check out `magrittr` for more pipe options, *e.g.*, `%>>%`.

RStudio

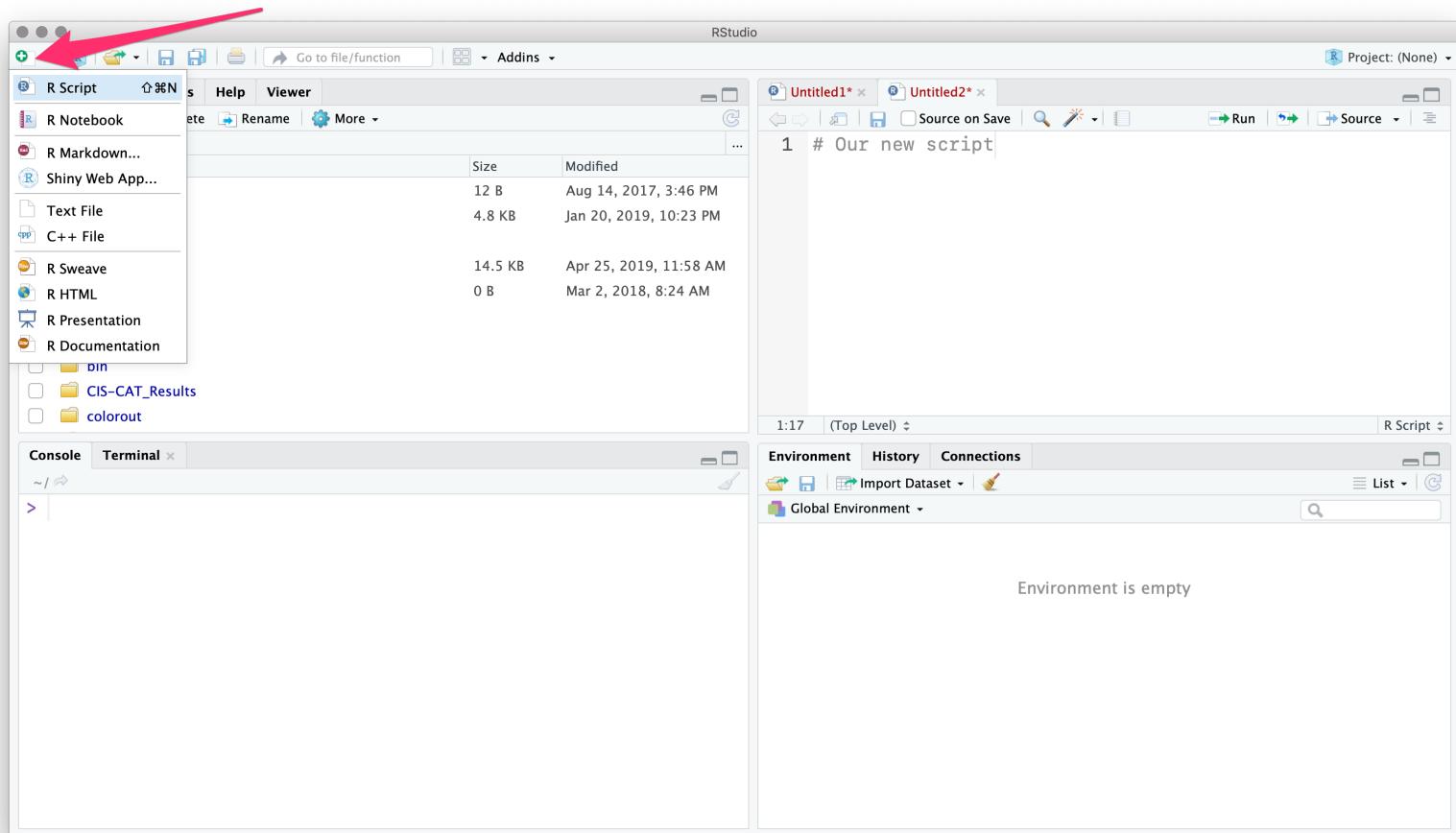
Let's recap some of the major features in RStudio...



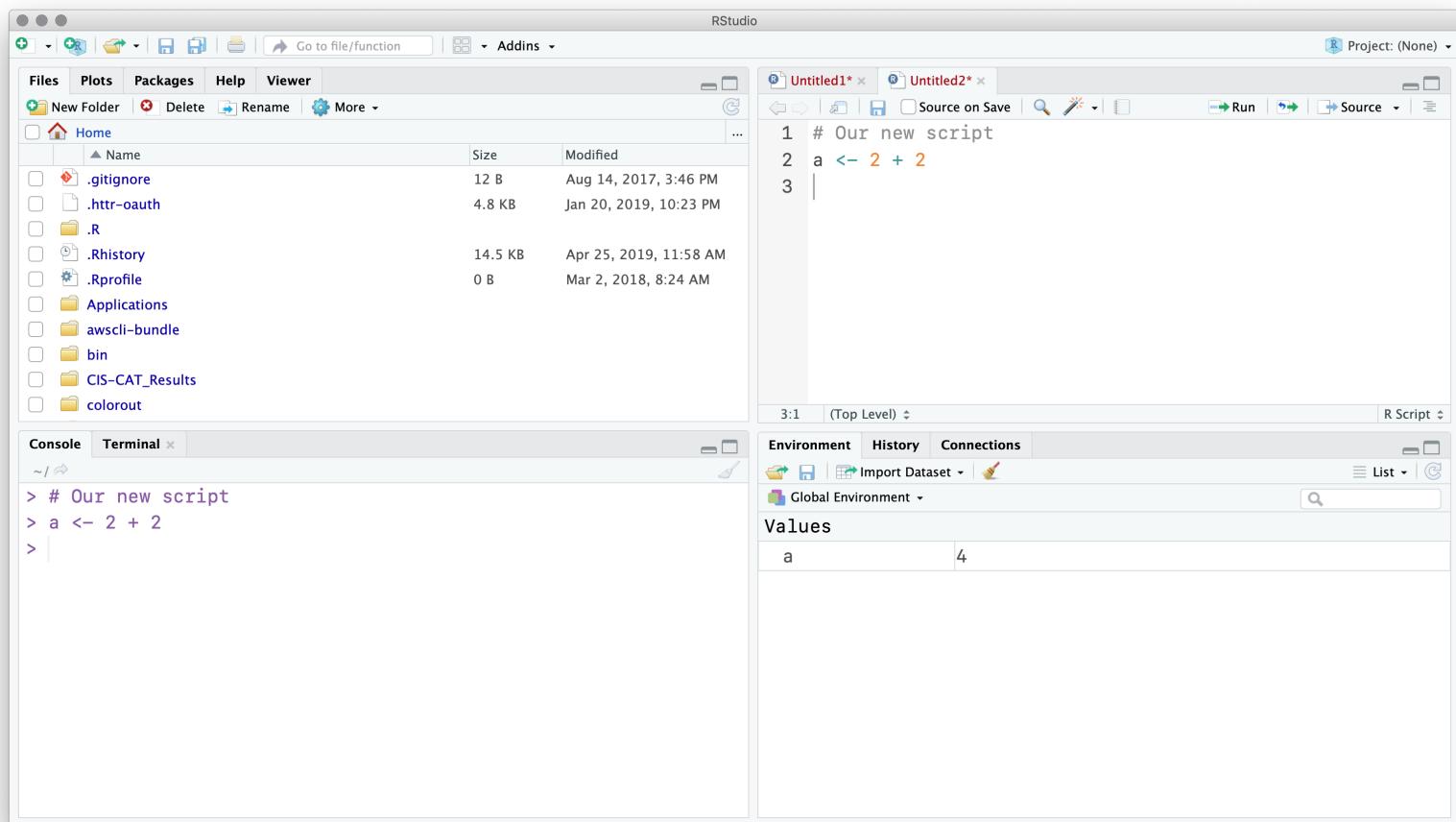
First, you write your R scripts (source code) in the **Source** pane.



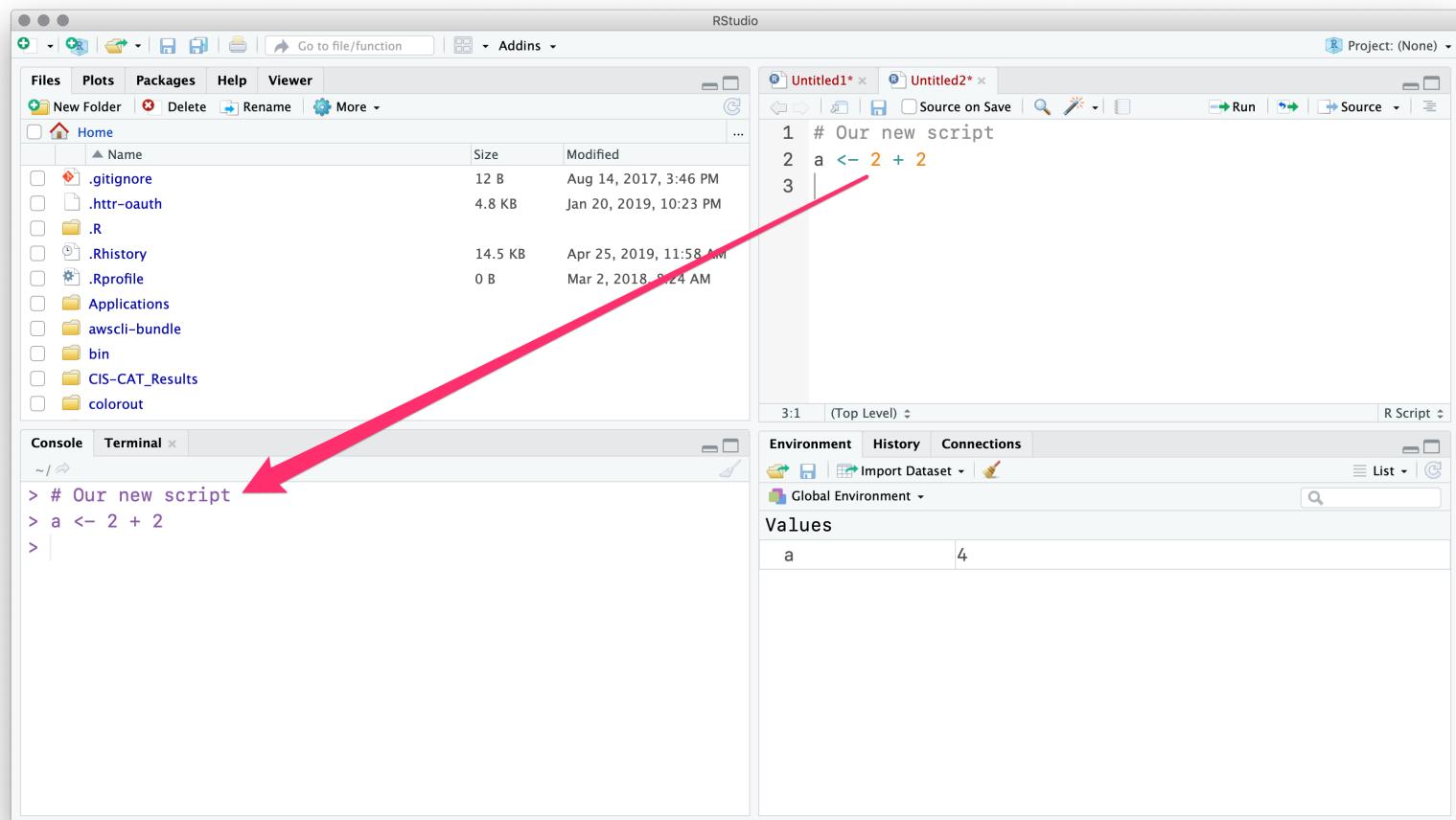
You can use the menubar or ⌘+⌘+N to create new R scripts.



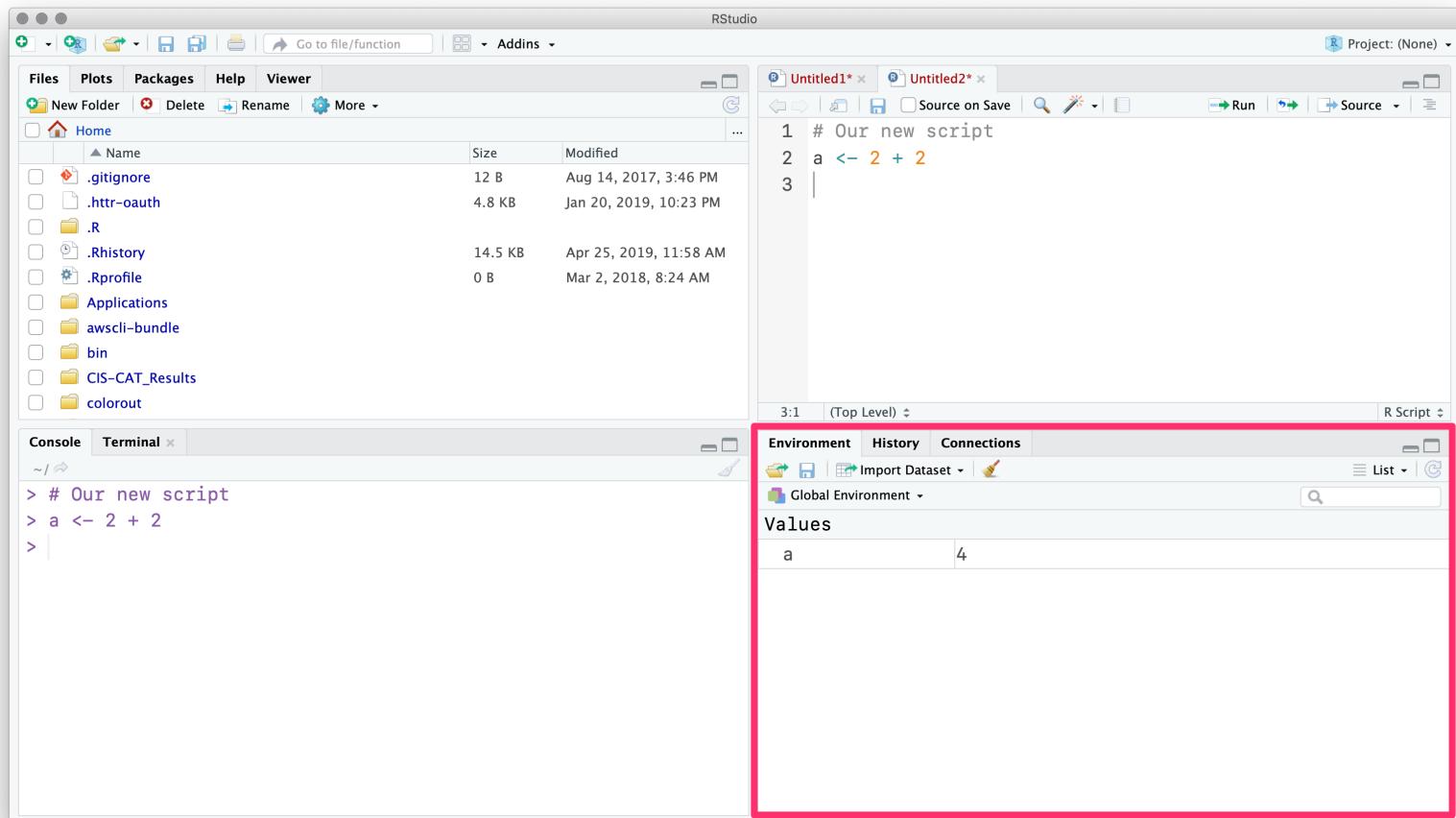
To execute commands from your R script, use ⌘+Enter.



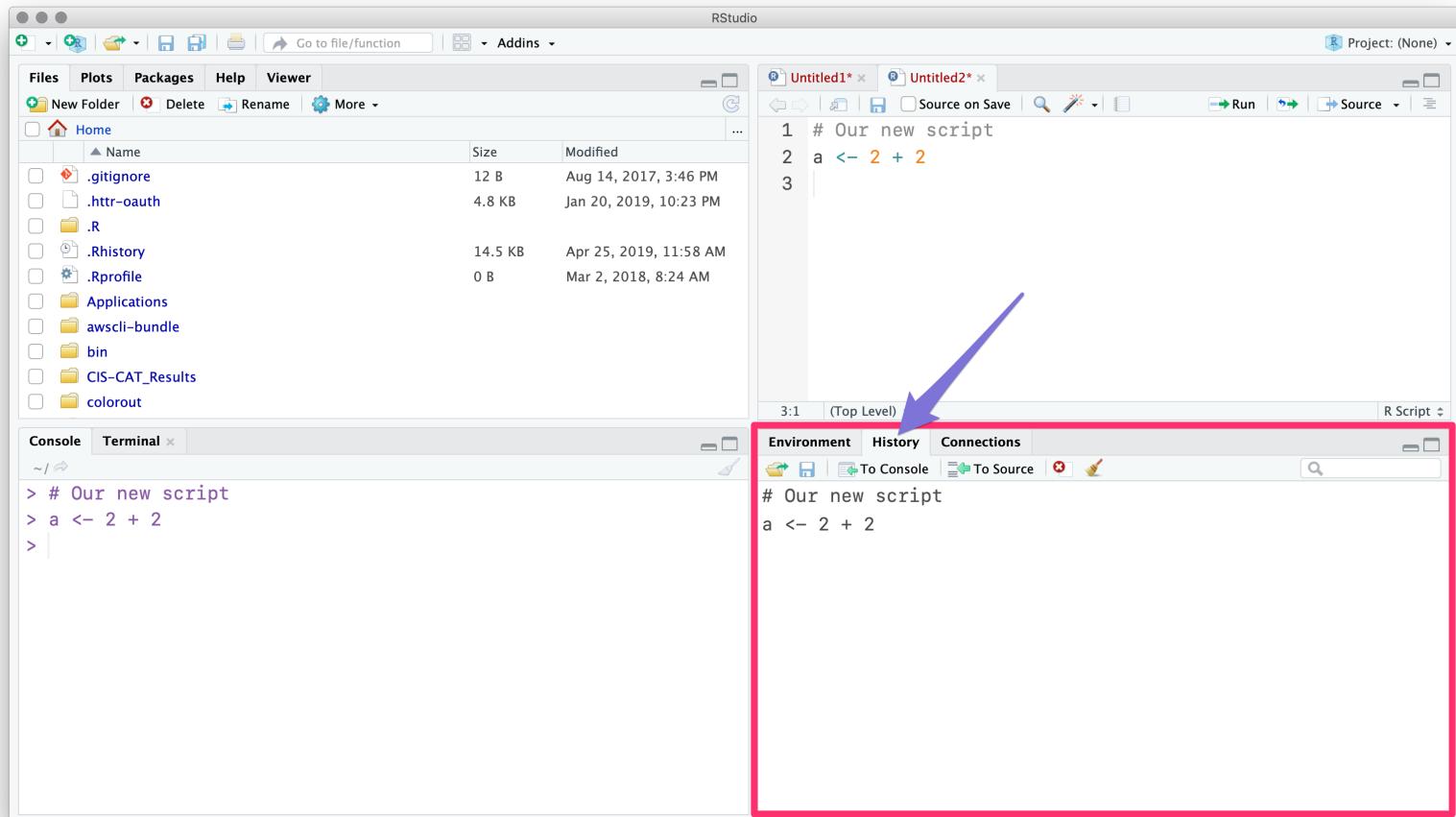
RStudio will execute the command in the terminal.



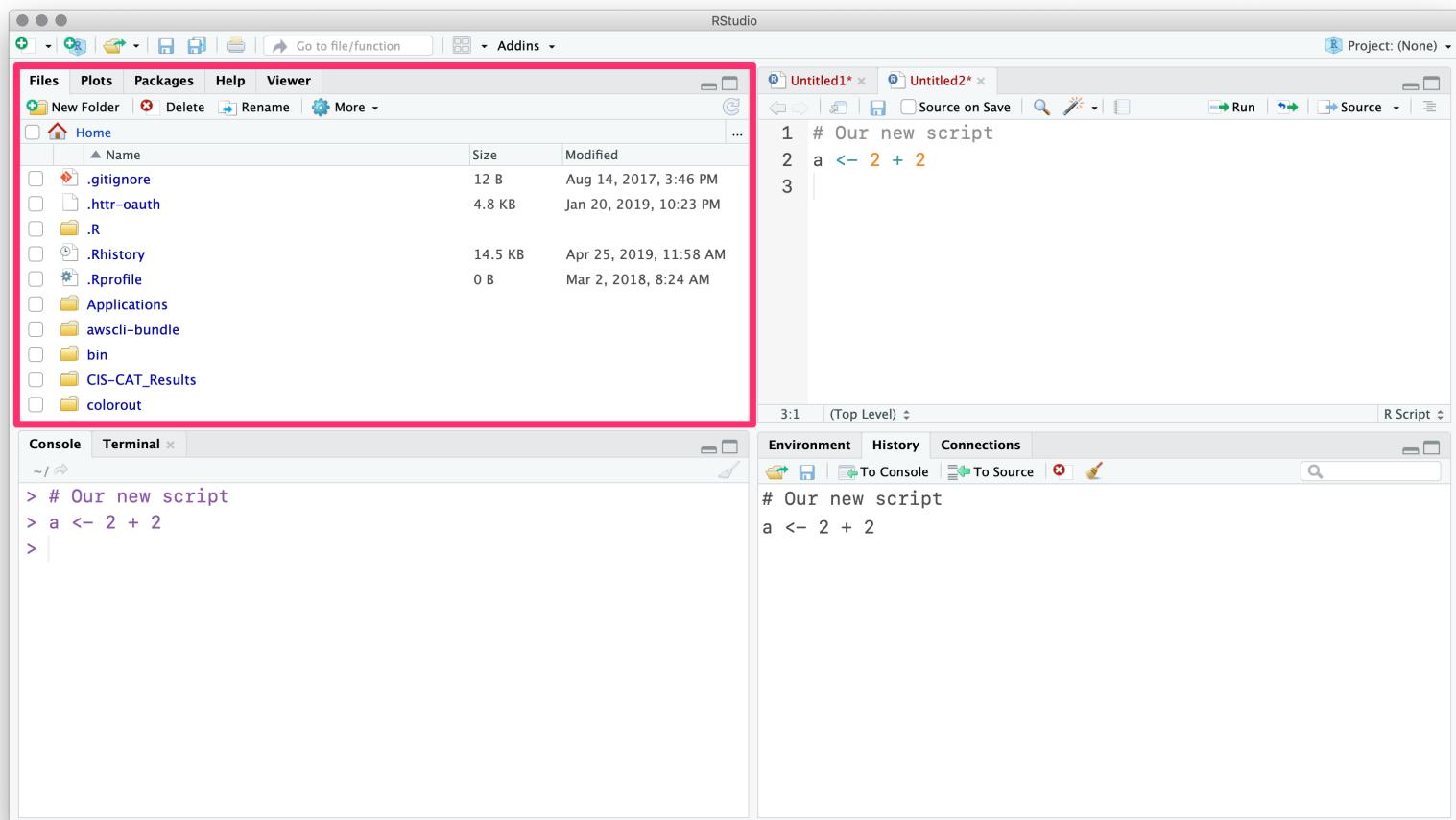
You can see our new object in the **Environment** pane.



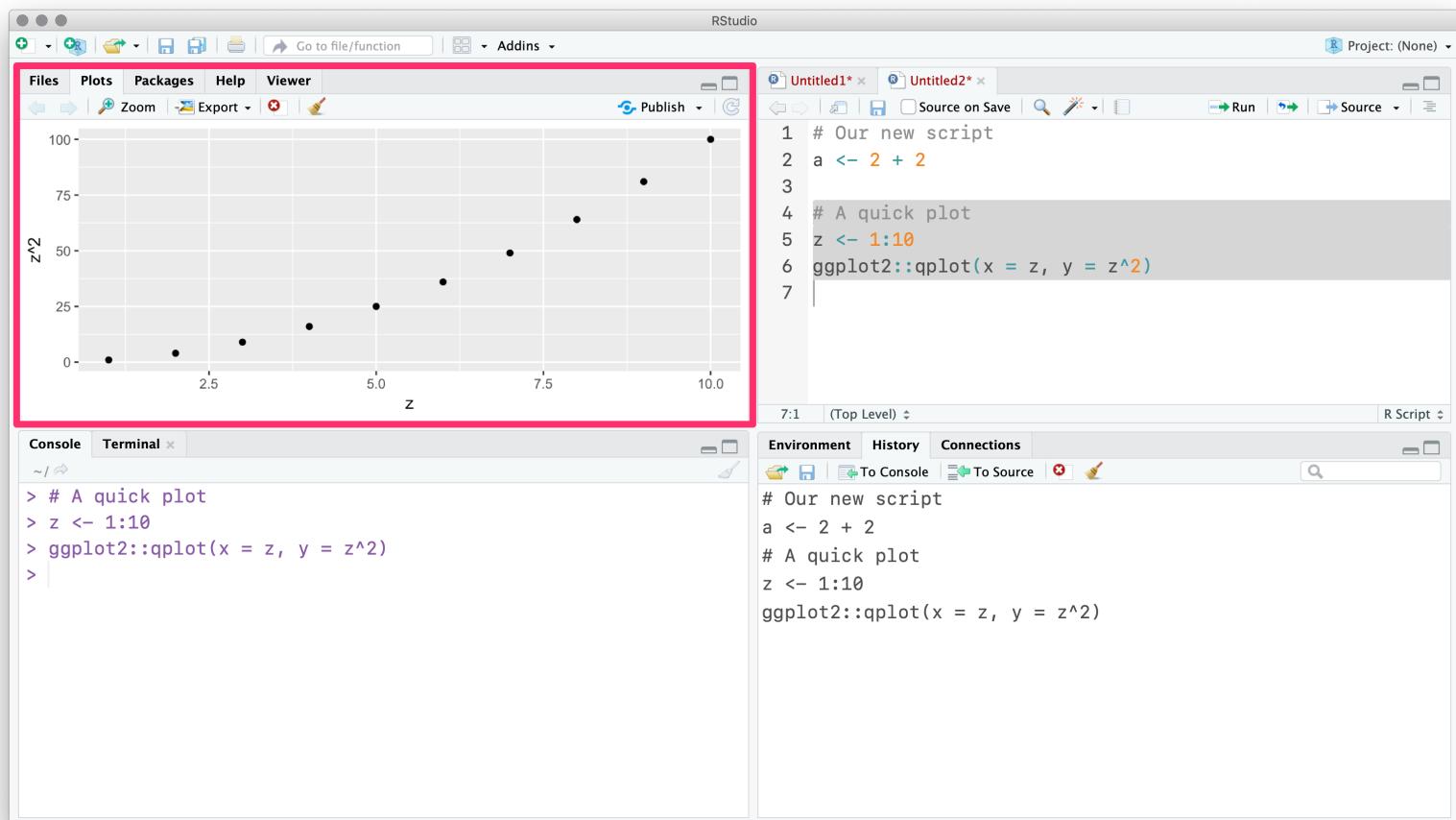
The **History** tab (next to **Environment**) records your old commands.



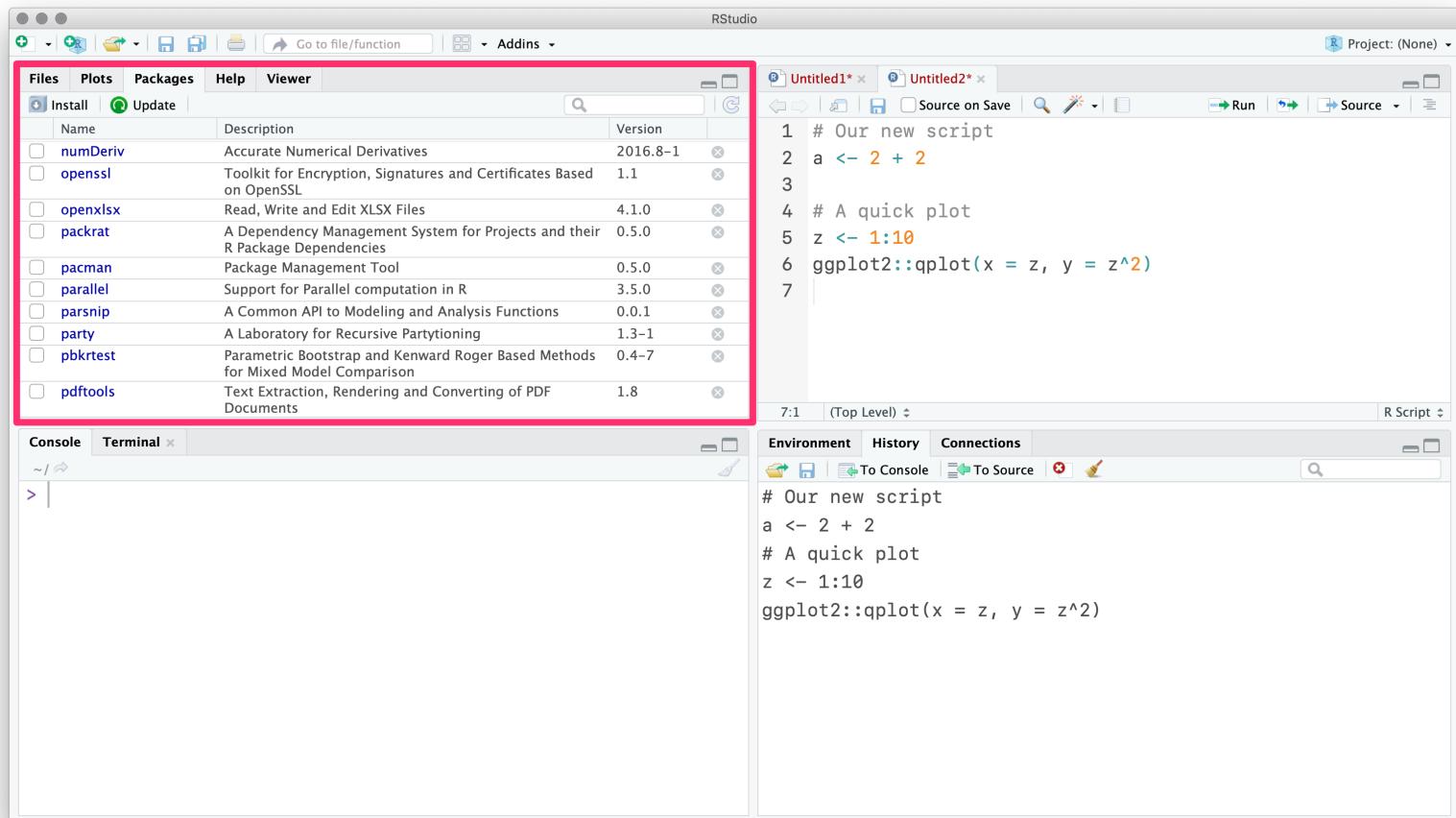
The **Files** pane is file explorer.



The **Plots** pane/tab shows... plots.



Packages shows installed packages



Packages shows installed packages and whether they are **loaded**.

The screenshot shows the RStudio interface. The top menu bar includes Files, Plots, Packages, Help, Viewer, Addins, and a Project dropdown set to (None). The Packages tab is selected, highlighted with a red box. A blue arrow points from the text "Shows installed packages and whether they are loaded." to the checked checkbox next to "pacman" in the list. The list of packages includes:

Name	Description	Version
numDeriv	Accurate Numerical Derivatives	2016.8-1
openssl	Toolkit for Encryption, Signatures and Certificates Based on OpenSSL	1.1
openxlsx	Read, Write and Edit XLSX Files	4.1.0
packrat	A Dependency Management System for Projects and their R Package Dependencies	0.5.0
<input checked="" type="checkbox"/> pacman	Package Management Tool	0.5.0
<input type="checkbox"/> parallel	Support for Parallel computation in R	3.5.0
<input type="checkbox"/> rsnip	A Common API to Modeling and Analysis Functions	0.0.1
<input type="checkbox"/> party	A Laboratory for Recursive Partytioning	1.3-1
<input type="checkbox"/> pbkrtest	Parametric Bootstrap and Kenward Roger Based Methods for Mixed Model Comparison	0.4-7
<input type="checkbox"/> pdftools	Text Extraction, Rendering and Converting of PDF Documents	1.8

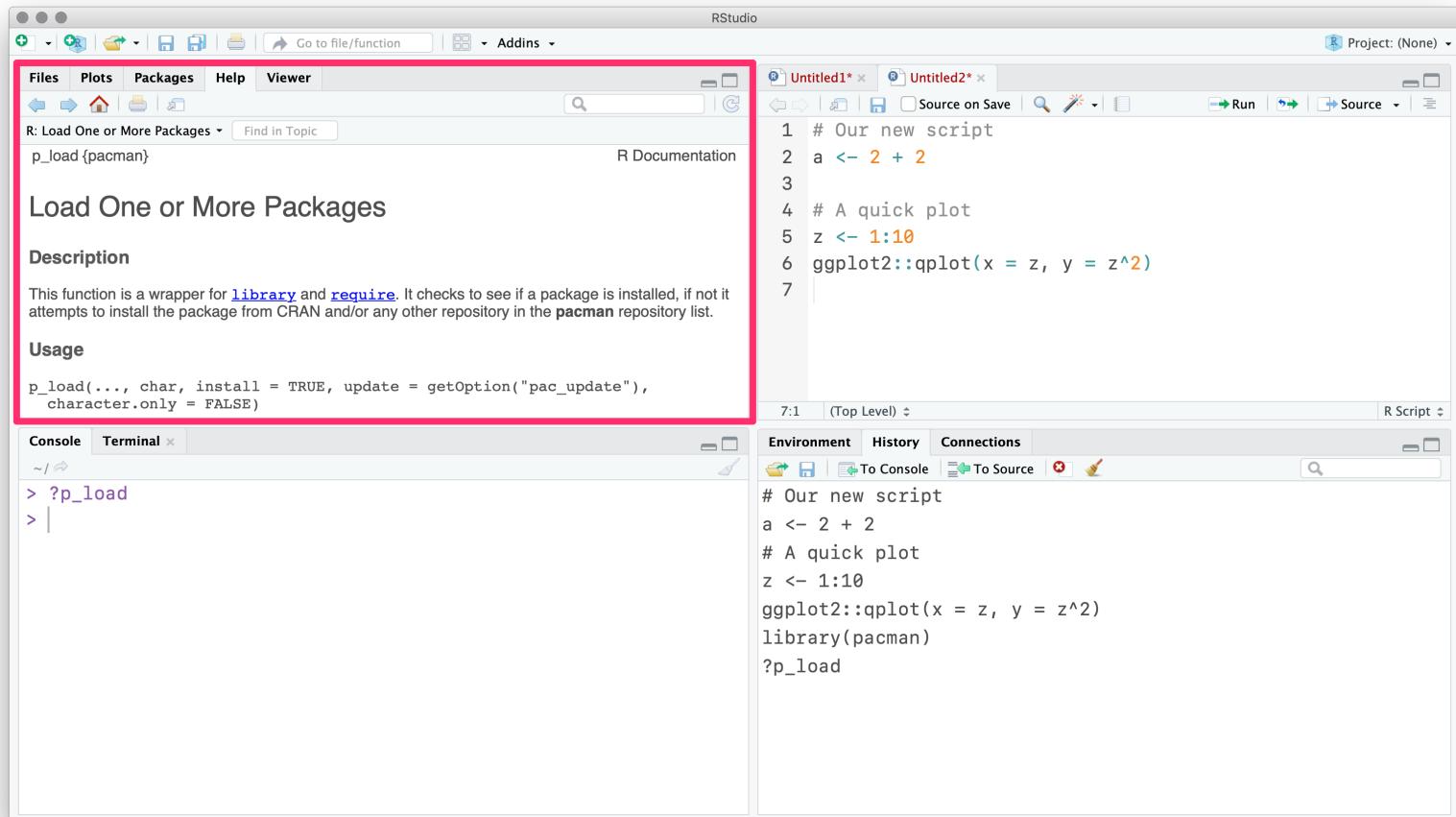
The right panel contains two tabs: Untitled1* and Untitled2*. Untitled1* shows a script with the following code:

```
# Our new script
a <- 2 + 2
# A quick plot
z <- 1:10
ggplot2::qplot(x = z, y = z^2)
```

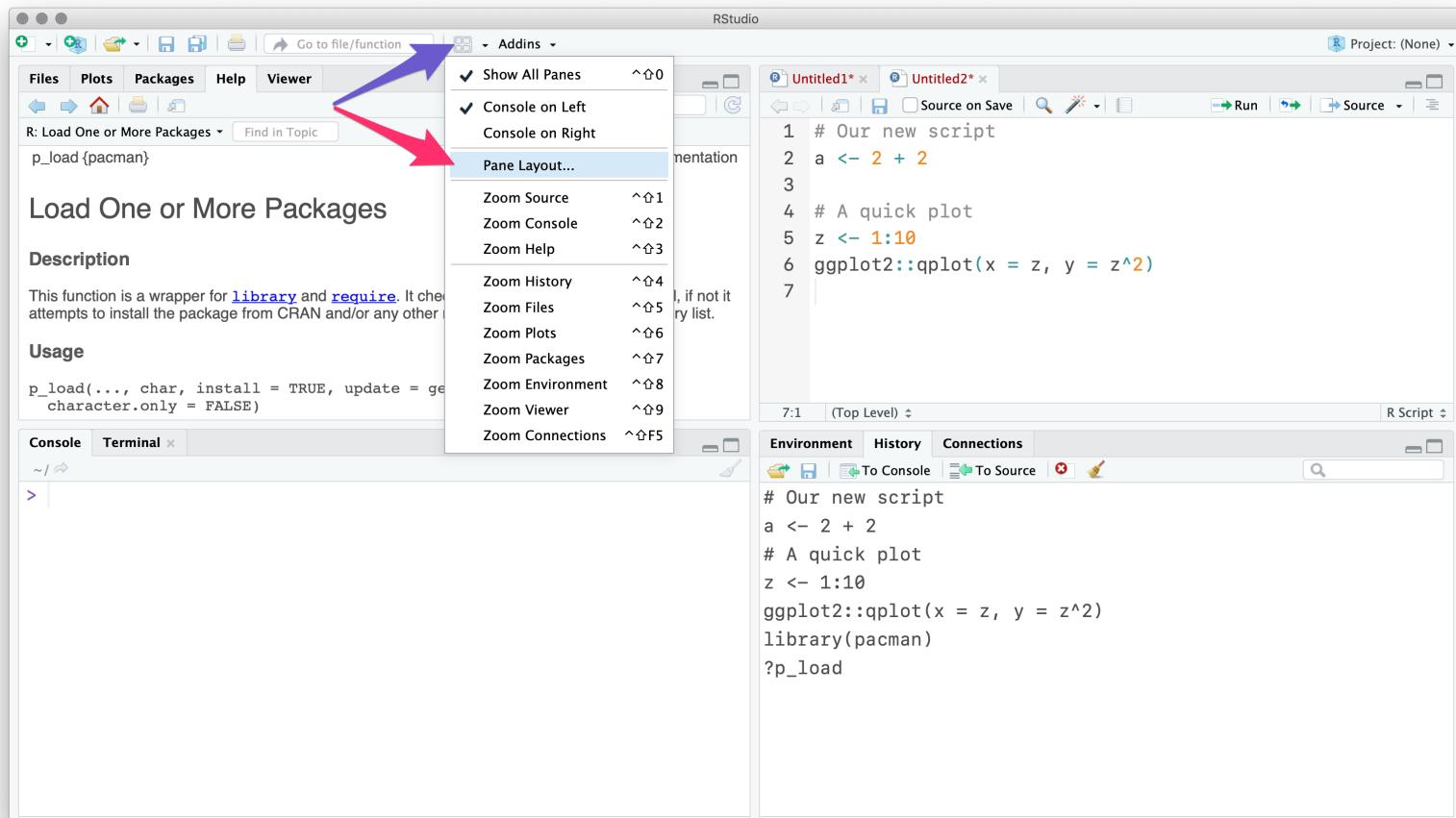
Untitled2* shows the same script with the addition of "library(pacman)" at the bottom.

The bottom left shows the Console and Terminal panes, and the bottom right shows the Environment, History, and Connections panes.

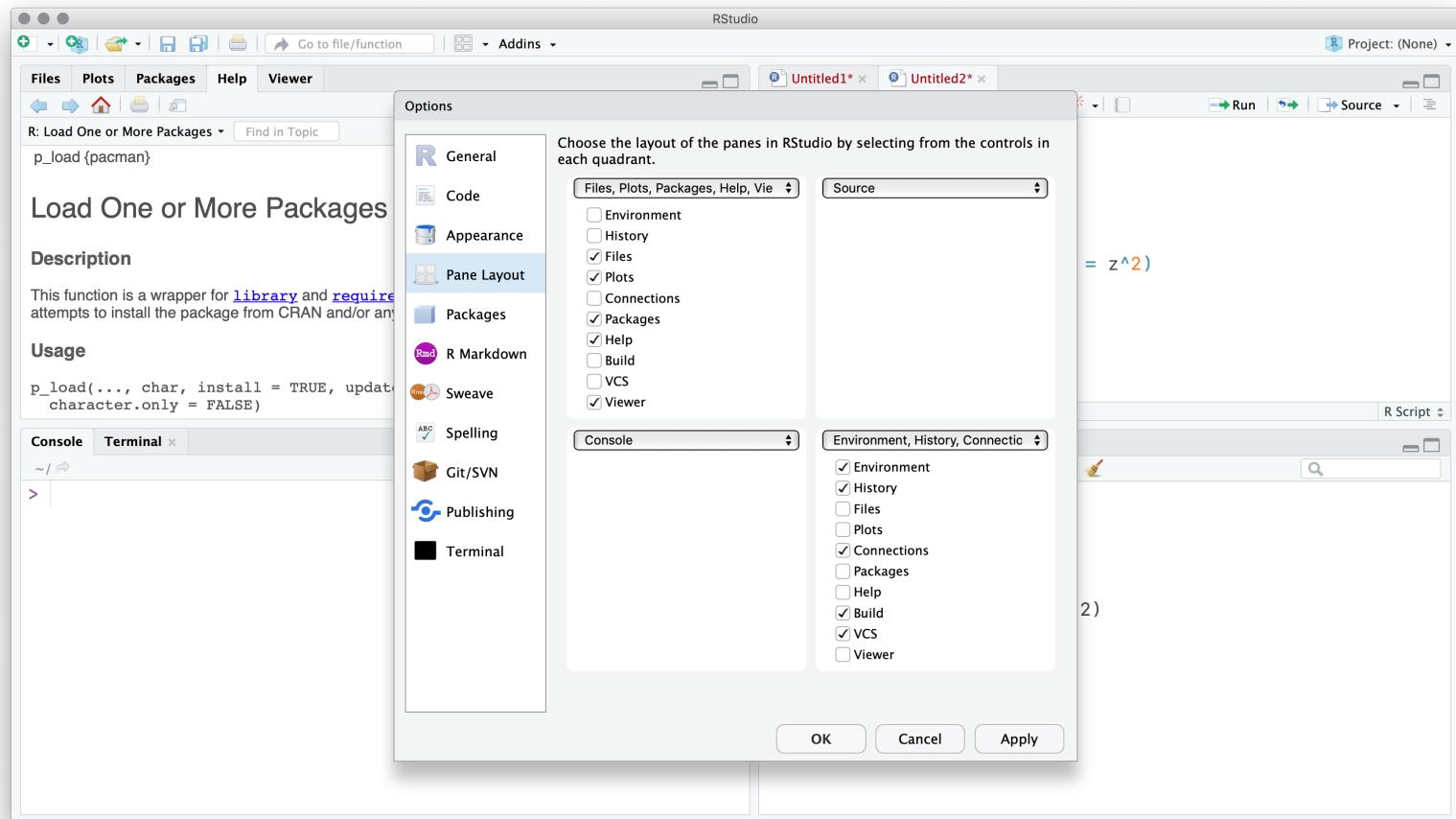
The **Help** tab shows help documentation (also accessible via **?**).



Finally, you can customize the actual layout



Finally, you can customize the actual layout and many other items.



R and RStudio

Best practices

1. Write code in R scripts. Troubleshoot in RStudio. Then run the scripts.
2. Comment your code. (# This is a comment)
3. Name objects and variables with intelligible, standardized names.
 - o **BAD** ALLCARS, Vl123a8, a.fun, cens.12931, cens.12933
 - o **GOOD** unique_cars, health_df, sim_fun, is_female, age
4. Set seeds when generating randomness, e.g., set.seed(123).
5. Parallelize when possible. (Packages: parallel, purrr, foreach, etc.)
6. Use projects in RStudio (next). And organize your projects.

R and RStudio

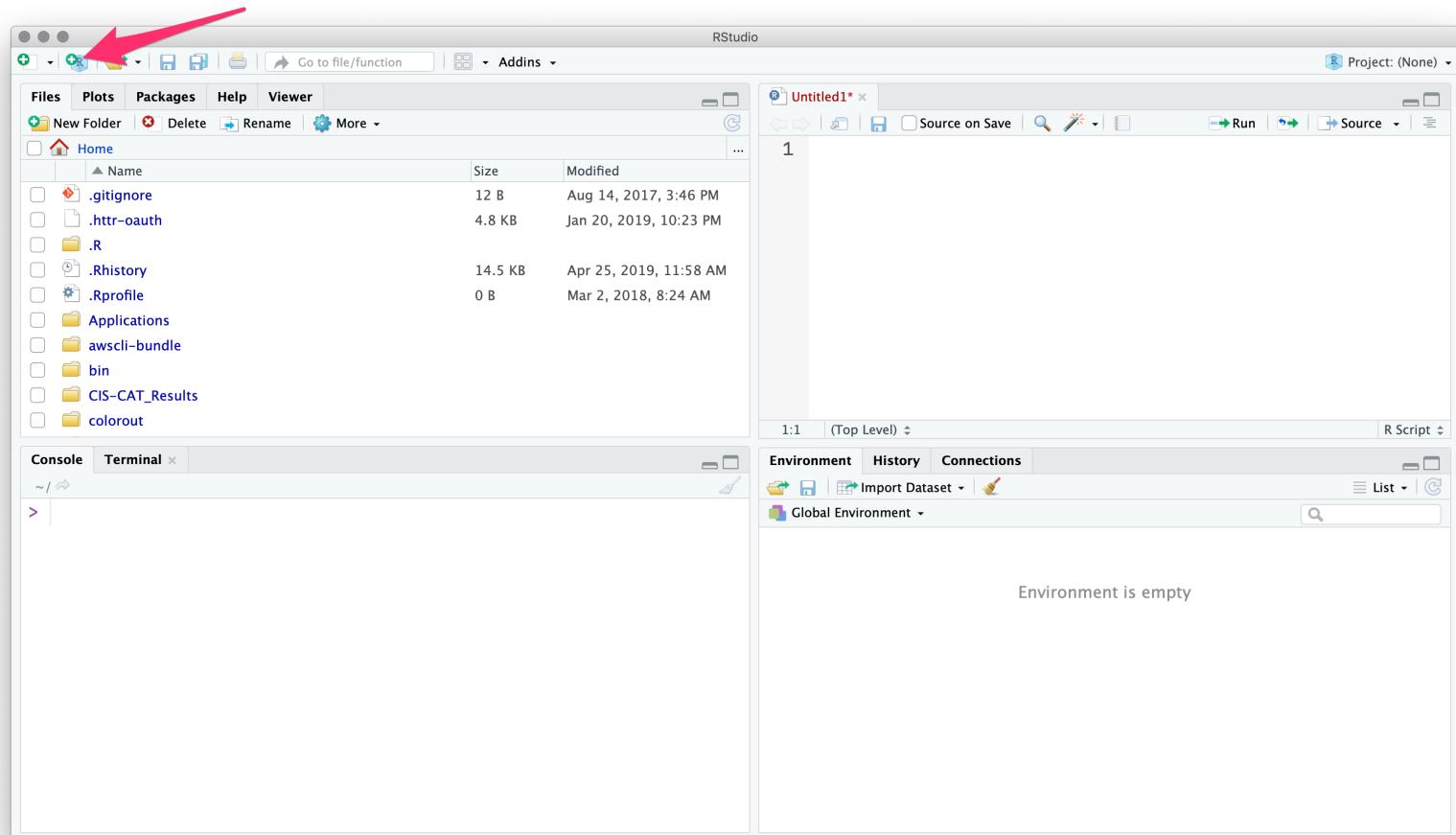
Projects

Projects in R offer several benefits:

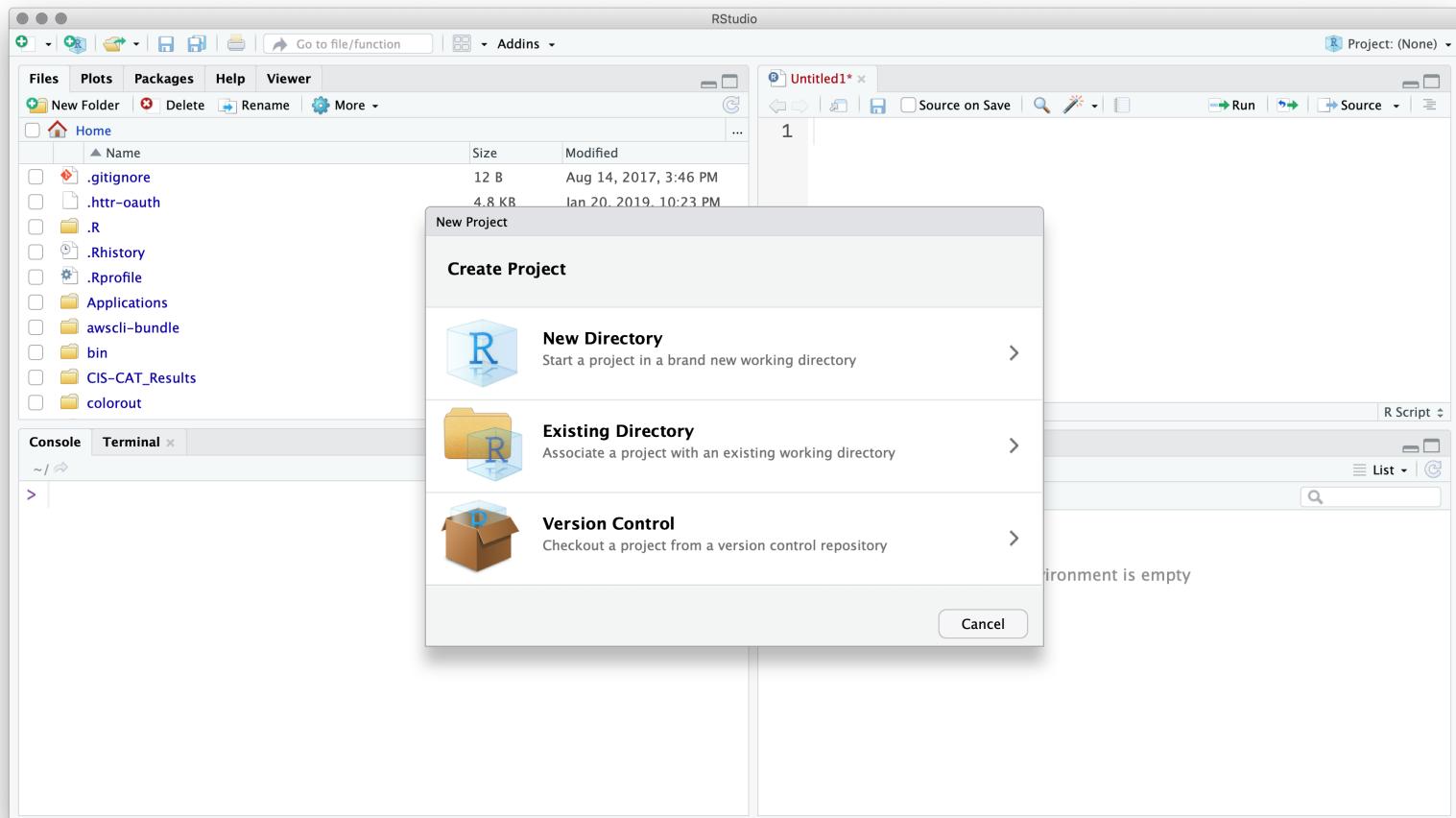
1. Act as an anchor for working with files.
2. Make your work (projects) easily reproducible.[†]
3. Help you quickly jump back into your work.

[†] In this class, we're assuming reproducibility is good/desirable.

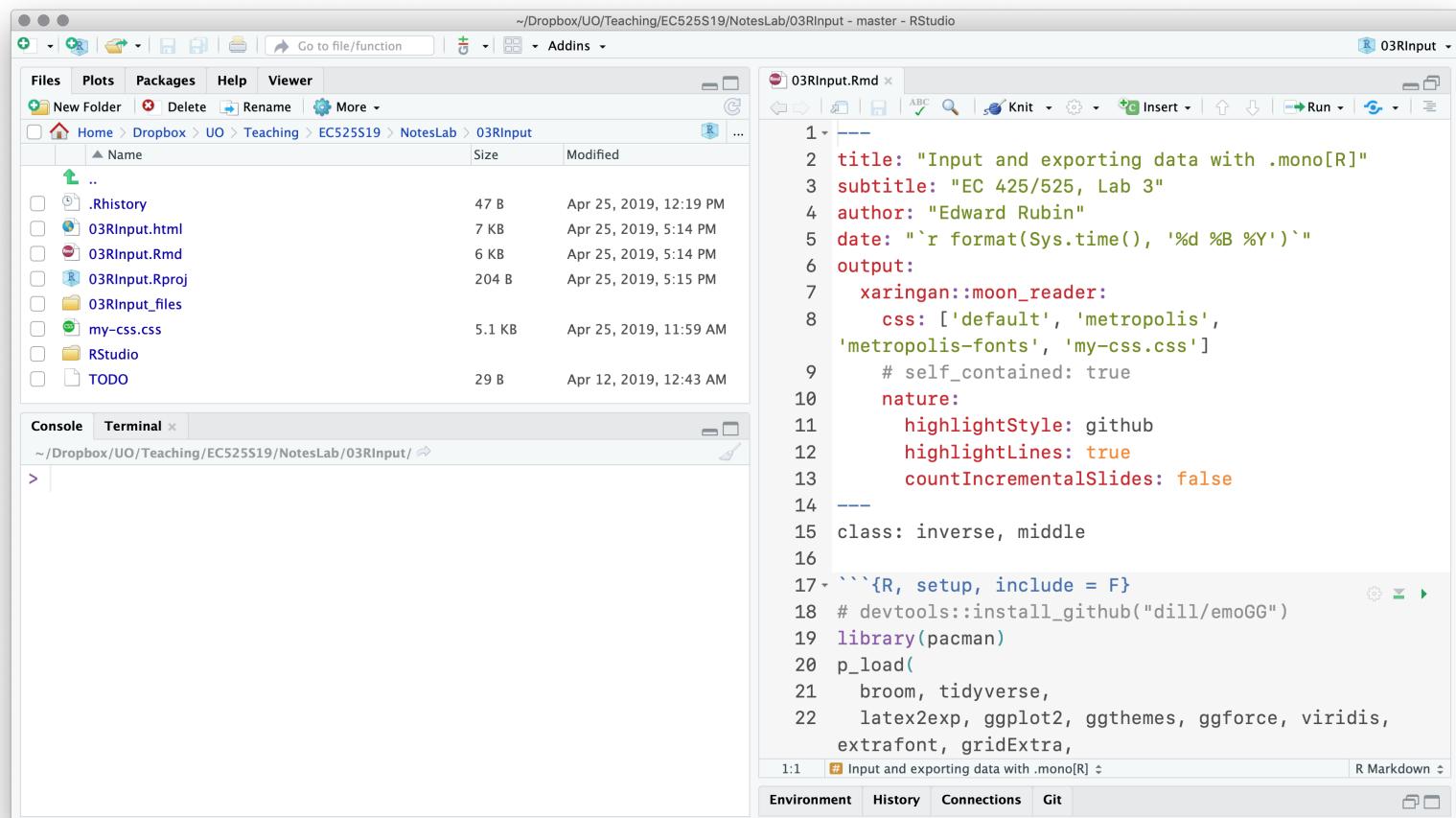
To start a new project, hit the **project icon**.



You'll then choose the folder/directory where your project lives.



If you open (double click) a project, RStudio opens R in that location.



RStudio will 'load' your previous setup (pane setup, scripts, etc.).

The screenshot shows the RStudio interface with the following details:

- File Explorer:** Shows a directory tree under "Dropbox/UO/Teaching/EC525S19/NotesLab/03RInput". The files listed are: .Rhistory, 03RInput.html, 03RInput.Rmd, 03RInput.Rproj, 03RInput_files, my-css.css, RStudio, and TODO.
- Code Editor:** The file "03RInput.Rmd" is open. The code content is as follows:

```
1 ----
2 title: "Input and exporting data with .mono[R]"
3 subtitle: "EC 425/525, Lab 3"
4 author: "Edward Rubin"
5 date: "`r format(Sys.time(), '%d %B %Y')`"
6 output:
7   xaringan::moon_reader:
8     css: ['default', 'metropolis',
9       'metropolis-fonts', 'my-css.css']
9     # self_contained: true
10    nature:
11      highlightStyle: github
12      highlightLines: true
13      countIncrementalSlides: false
14 ----
15 class: inverse, middle
16
17 ````{R, setup, include = F}
18 # devtools::install_github("dill/emoGG")
19 library(pacman)
20 p_load(
21   broom, tidyverse,
22   latex2exp, ggplot2, ggthemes, ggforce, viridis,
23   extrafont, gridExtra,
```

- Terminal:** The terminal pane shows the path: ~/Dropbox/UO/Teaching/EC525S19/NotesLab/03RInput/
- Status Bar:** Shows the file name "03RInput.Rmd", the line number "1:1", the status "# Input and exporting data with .mono[R] #", and the mode "R Markdown".
- Bottom Navigation:** Environment, History, Connections, Git tabs.

R and RStudio

Projects

Without a project, you will need to define long file paths that you'll need to keep updating as folder names/locations change.

```
dir_class <- "/Users/edwardarubin/Dropbox/U0/Teaching/EC525S19/"  
dir_labs <- paste0(dir_class, "NotesLab/")  
dir_lab03 <- paste0(dir_labs, "03RInput/")  
sample_df <- read.csv(paste0(dir_lab03, "sample.csv"))
```

With a project, R automatically references the project's folder.

```
sample_df <- read.csv("sample.csv")
```

Double-plus bonus The `here` package extends projects' reproducibility.

Data i/o

Data i/o

Reading files

Projects solve the hardest part of data input/output in R, *i.e.*, navigating your computer's file structure.

Steps to read in a file

1. Figure out your **file's location** relative to your project's location.
2. **Find the function** that loads your files' file type.
3. **Load the file** with the function (using its location).

Data i/o

Reading CSVs

We can check the files in the current (or any) directory with the `dir()`.

```
dir()
```

```
#> [1] "03RInput_cache"           "03RInput_files"
#> [3] "03RInput_NoPause_cache"  "03RInput_NoPause.Rmd"
#> [5] "03RInput.html"          "03RInput.Rmd"
#> [7] "03RInput.Rproj"         "my-css.css"
#> [9] "RStudio"                "sample.csv"
#> [11] "TODO"
```

Our current directory has the CSV `sample.csv` that I want to load.

Data i/o

Reading CSVs

R's base function for reading CSVs is `read.csv(file)`.

You feed `read.csv()` the directory and name of the CSV.[†]

```
read.csv("sample.csv") %>% head(4)
```

```
#>   pid age first_name is_orange
#> 1   1   68   Jessica   FALSE
#> 2   2   80   Andrew    FALSE
#> 3   3   71   Donald    TRUE
#> 4   4   81   Jacob    FALSE
```

`read.csv()` returns a `data.frame` with the CSV's contents.

[†] There are many other optional arguments, e.g., whether variables are named, variable types, etc.

Data i/o

Reading CSVs

The Hadleyverse (technically, the `tidyverse` package) contains a package called `readr`, which contains the `read_csv()` function.

`read_csv()` is pretty fast, guesses variable well, and returns a `tibble`.[†]

```
p_load(tidyverse)
read_csv("sample.csv") %>% head(3)
```

```
#> # A tibble: 3 x 4
#>   pid      age first_name is_orange
#>   <chr> <dbl> <chr>     <lgl>
#> 1 001      68 Jessica    FALSE
#> 2 002      80 Andrew     FALSE
#> 3 003      71 Donald     TRUE
```

[†] More speed: `fread()` from `data.table`. Notice `read.csv()` to `read_csv()` give `pid` differing classes.

Data i/o

Reading other file types

If you've got a file, chances are R can read it.

- Stata files: `read_dta` in `haven`
- SAS files: `read_sas` in `haven`
- Fixed-width files: `read_fwf()` in `readr` (also: `iotools`)
- Excel files: `read_excel()` in `readxl`
- Raster files: `raster()` in `raster`
- Shapefiles: `st_read()` in `sf`

Data i/o

Writing

If R can read it, then R can write it.

Generally, there is a `write` or `save` function for each `read` function.

```
# Read 'sample.csv'  
sample_df ← read_csv("sample.csv")  
# Write sample_df to 'sample_copy.csv'  
write_csv(  
  x = sample_df,  
  file = "sample_copy.csv"  
)
```

Data i/o

RDS files

While CSVs can be nice—they are readable without loading into a statistical program—when they get big, they can be slow and inefficient.

Enter RDS files, R's compressed, faster answer.

The base functions `readRDS()` and `saveRDS()` read and save RDS files.

`readr` offers `read_rds()` and `write_rds()` for more standard naming.

```
# Write sample_df to 'sample.rds'  
write_rds(x = sample_df, path = "sample.rds")  
# Read 'sample.rds'  
sample_df ← read_rds("sample.rds")
```

Additional resources

More resources related to today's materials.

1. RStudio's [cheatsheet for RStudio](#)
2. [Many other cheatsheets](#) from RStudio

Table of contents

Data, R, and RStudio

1. Schedule
2. Review
3. RStudio features
4. Best practices
5. Projects
6. Data i/o
 - o Reading files
 - o `dir()`
 - o `read.csv()`
 - o `read_csv()`
 - o Other file types
 - o Writing (output)
 - o RDS files
7. More resources