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

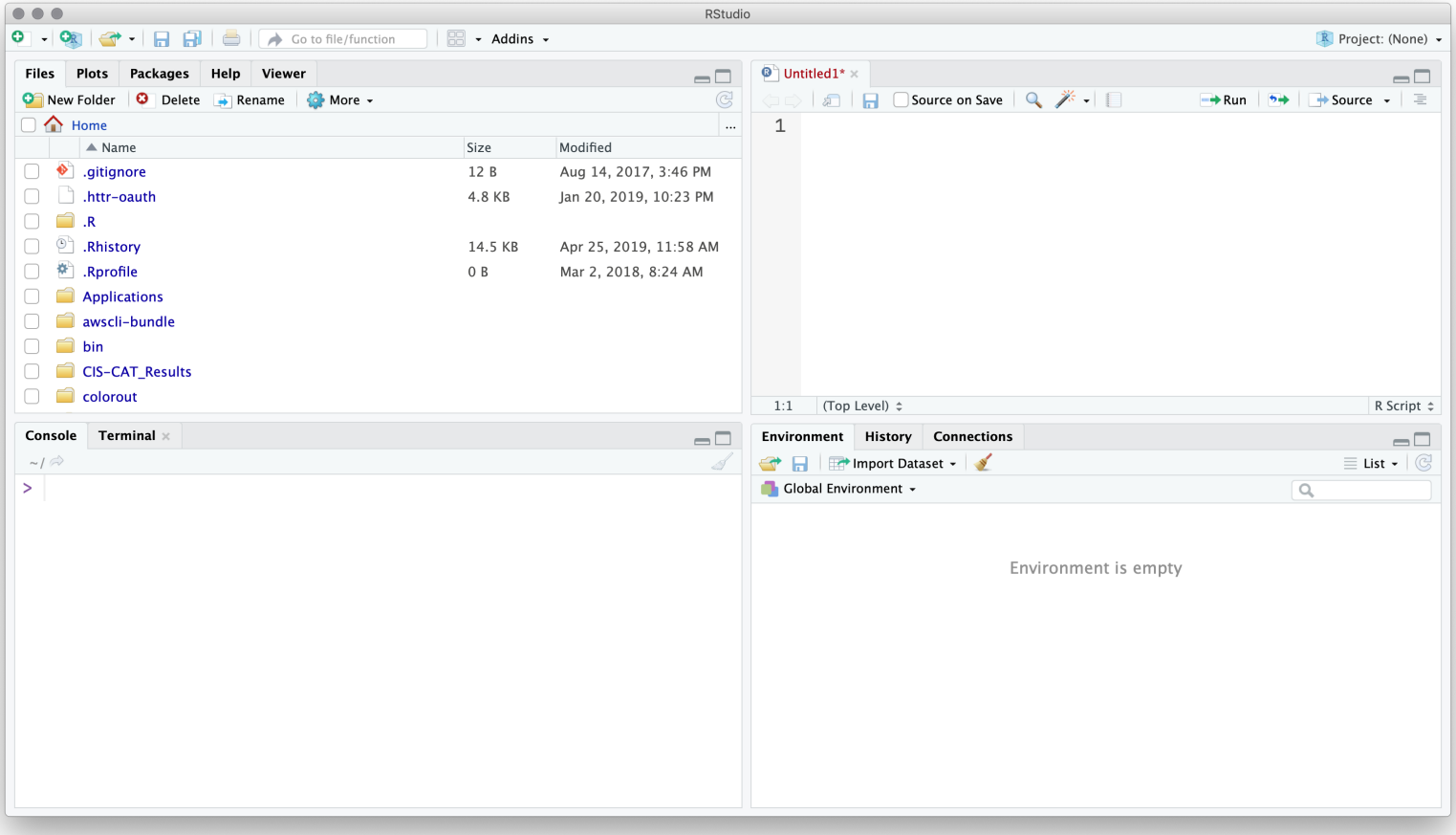
Key points from the last lab(s).

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

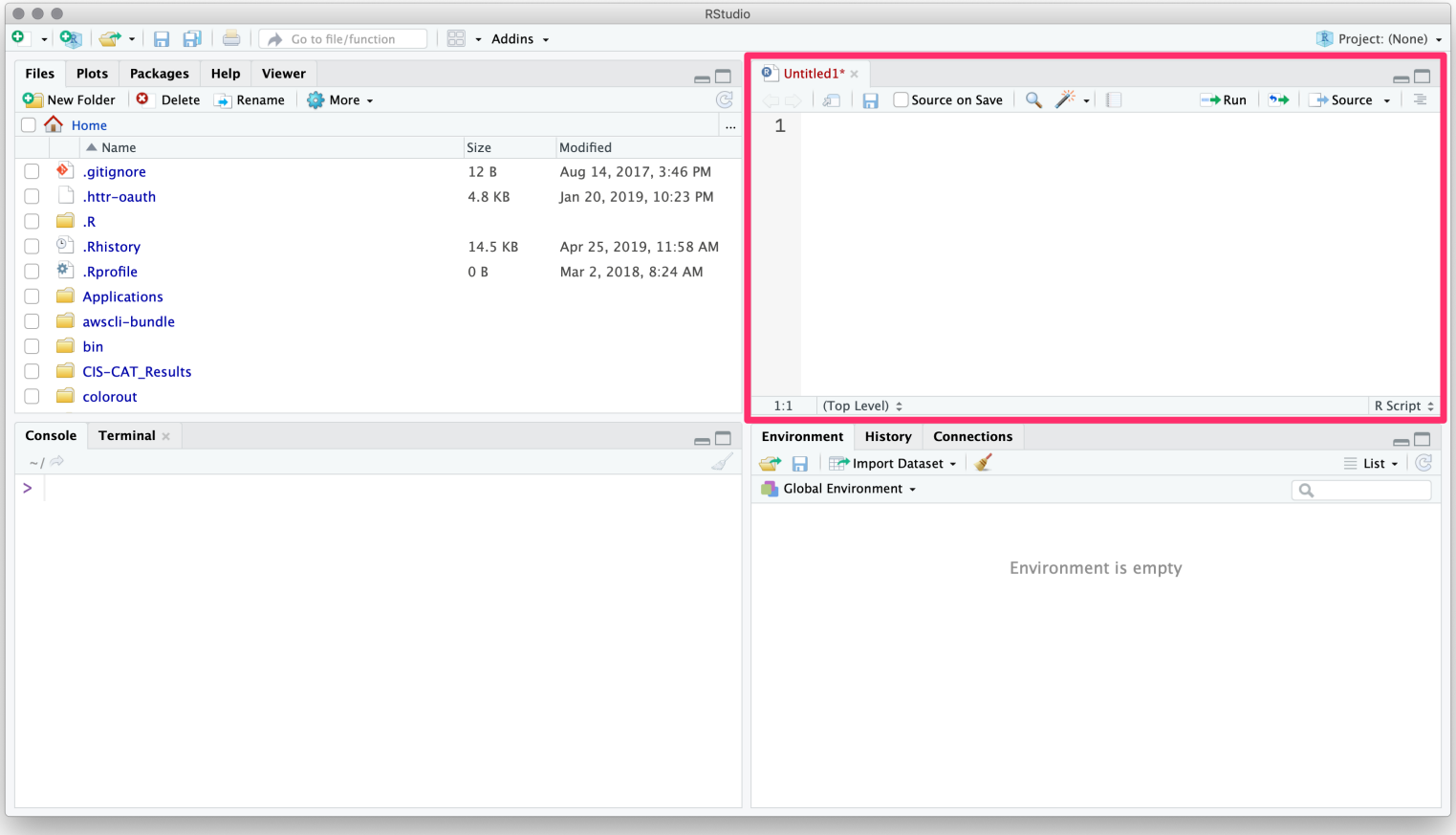
[†] Check out `magrittr` for more pipe options, *e.g.*, `%<>%`.

RStudio

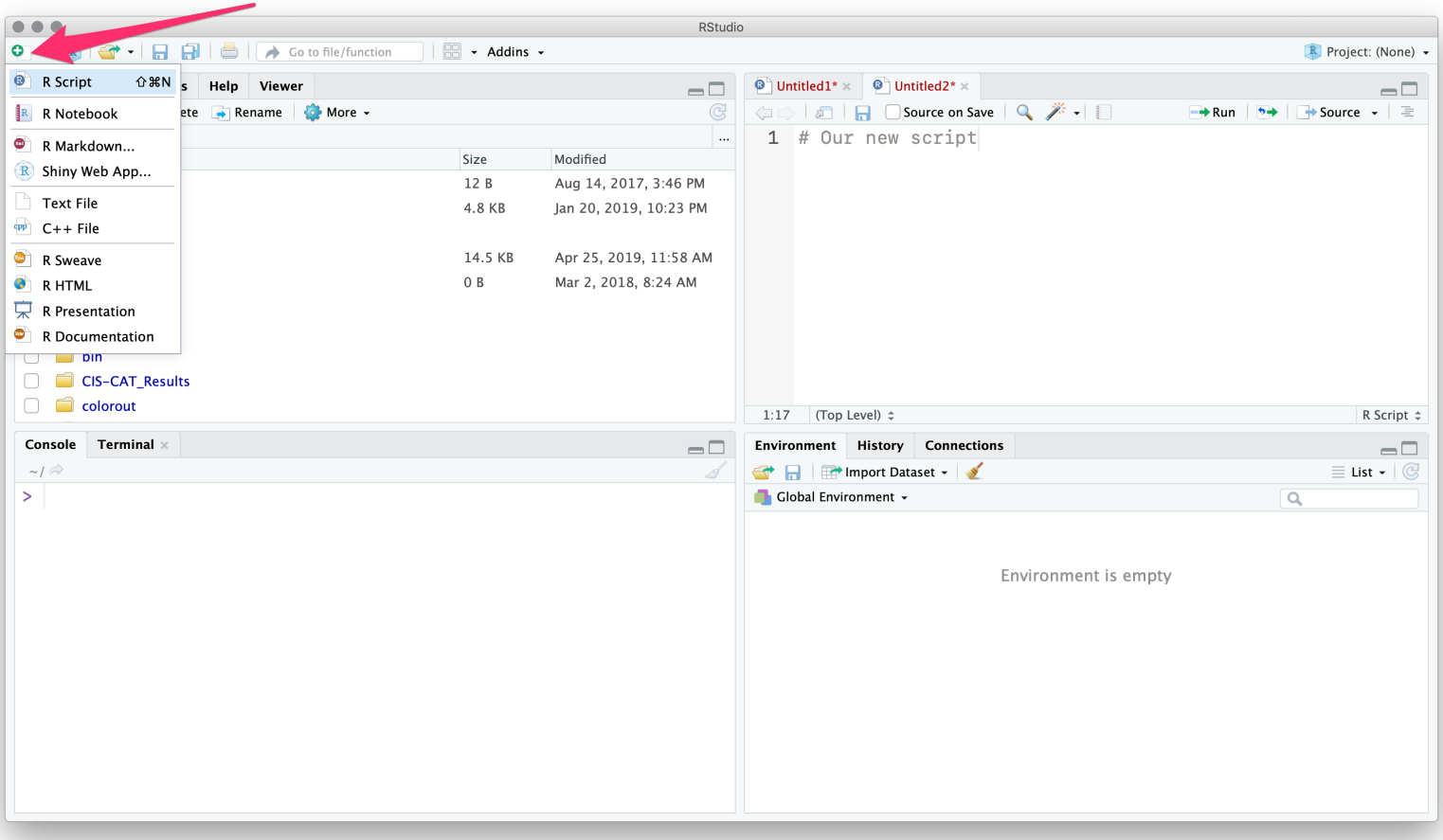
Let's recap some of the major features in `RStudio`...



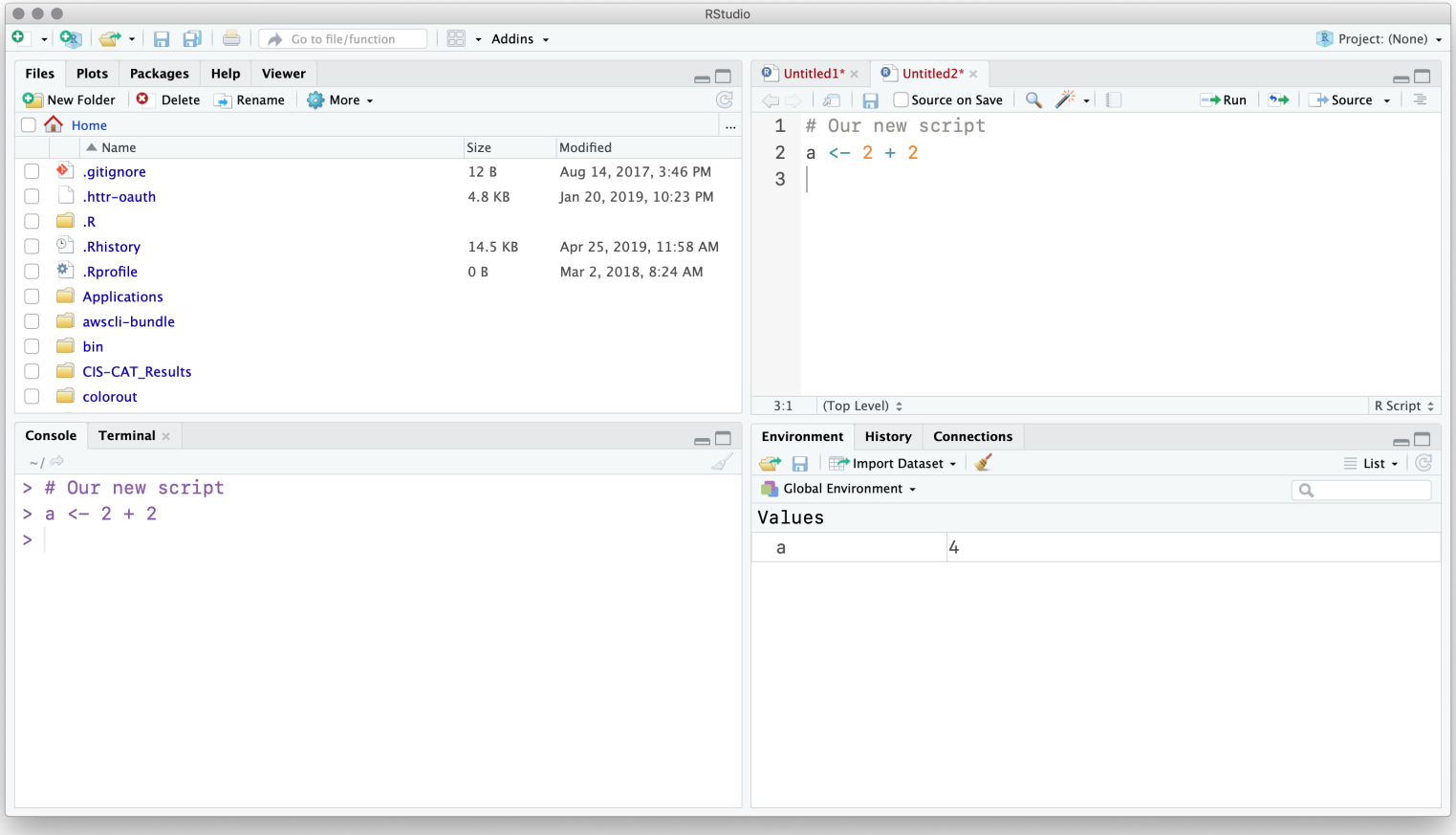
First, you write your R scripts (source code) in the **Source** pane.



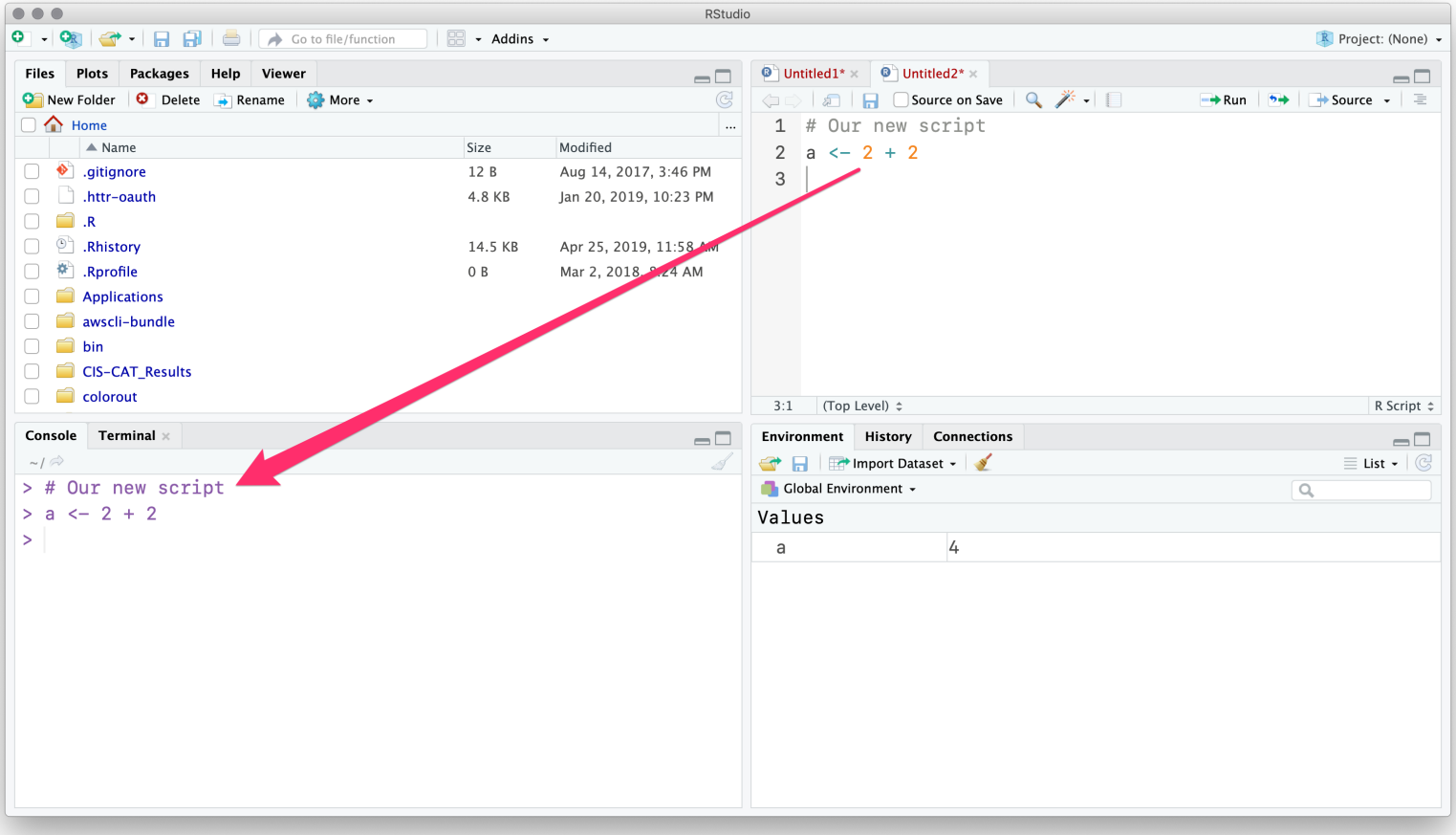
You can use the menubar or ⇧+⌘+N to create new R scripts.



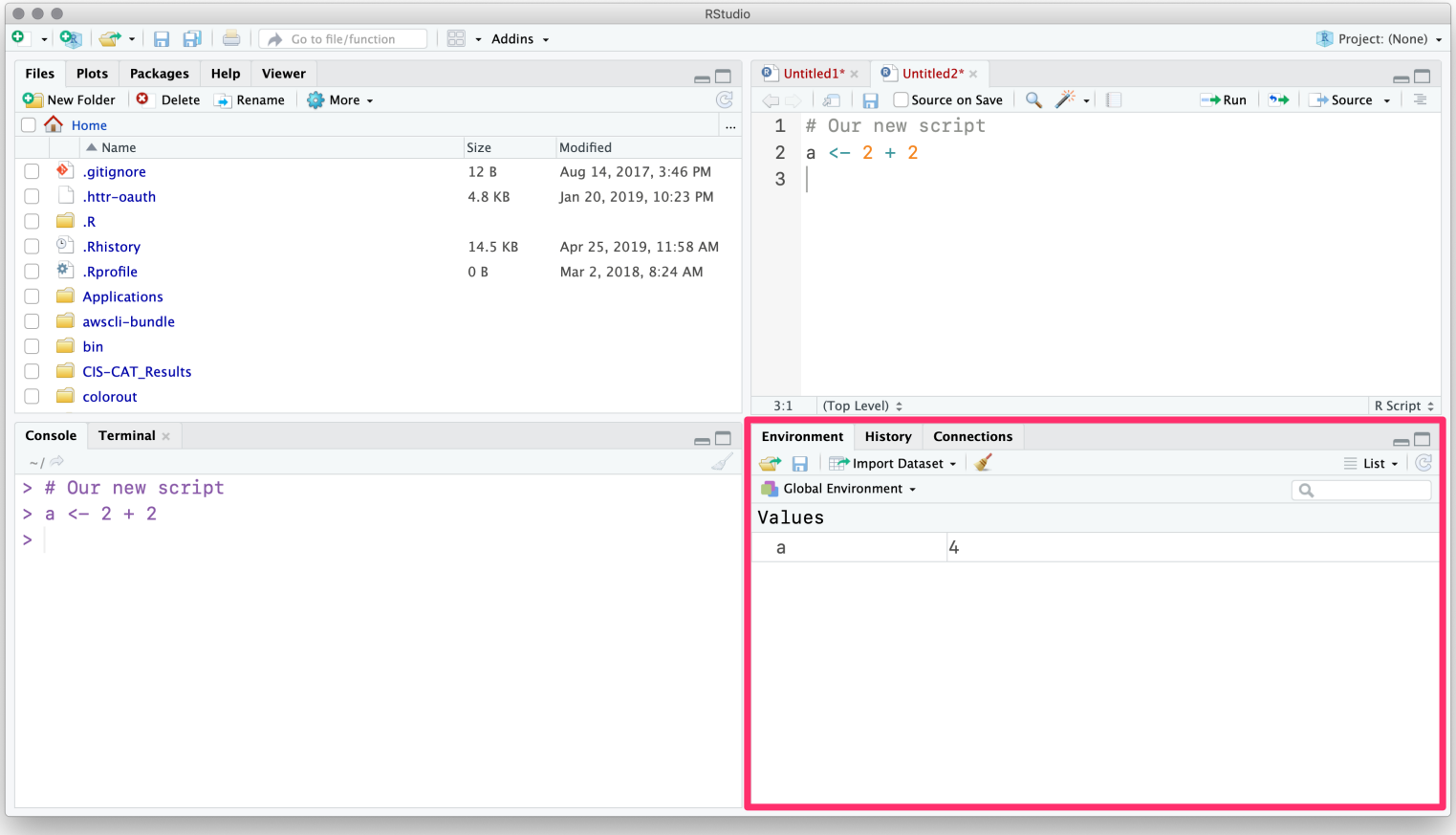
To execute commands from your `R` script, use ⌘+`Enter`.



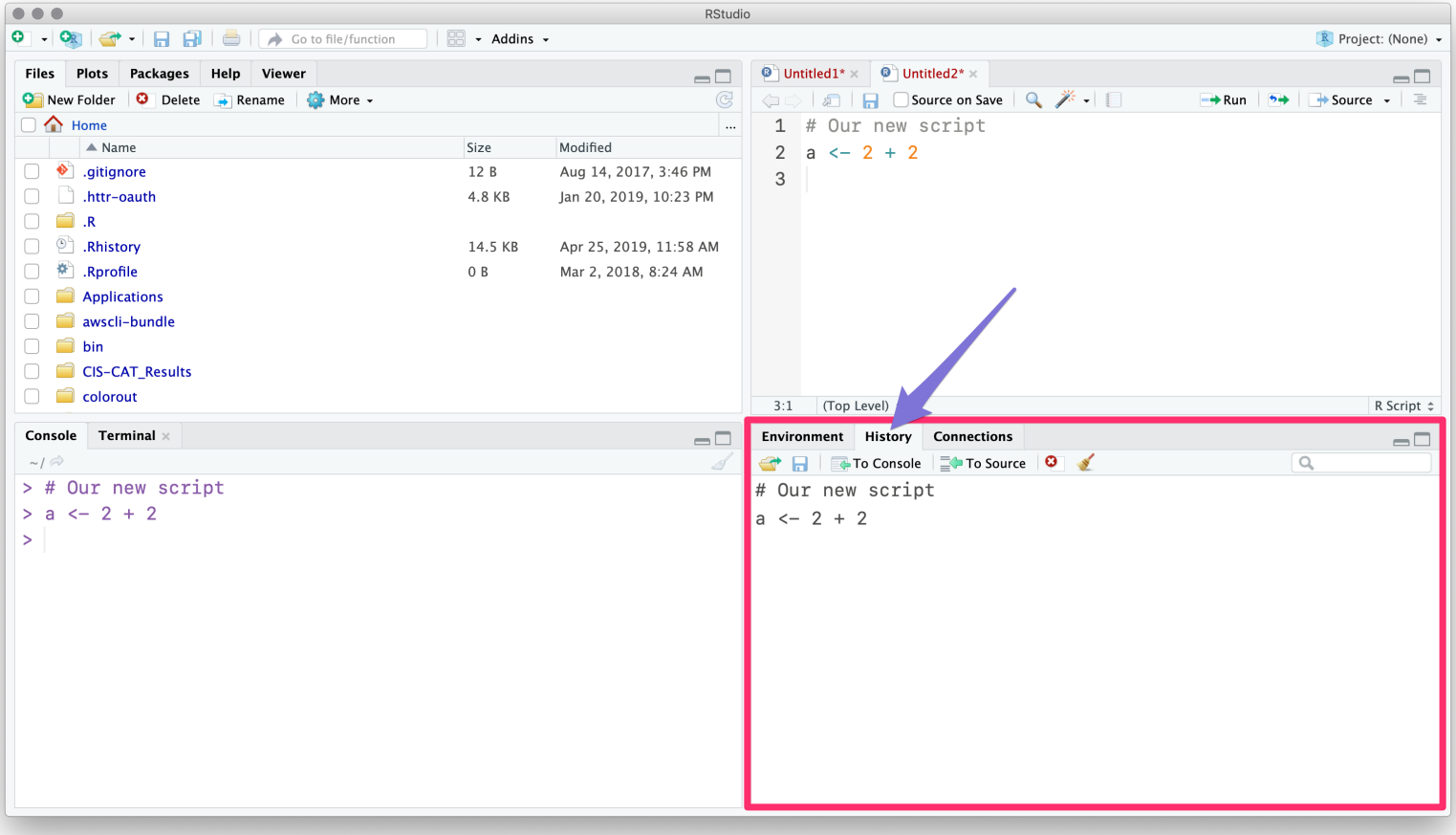
`RStudio` will execute the command in the terminal.



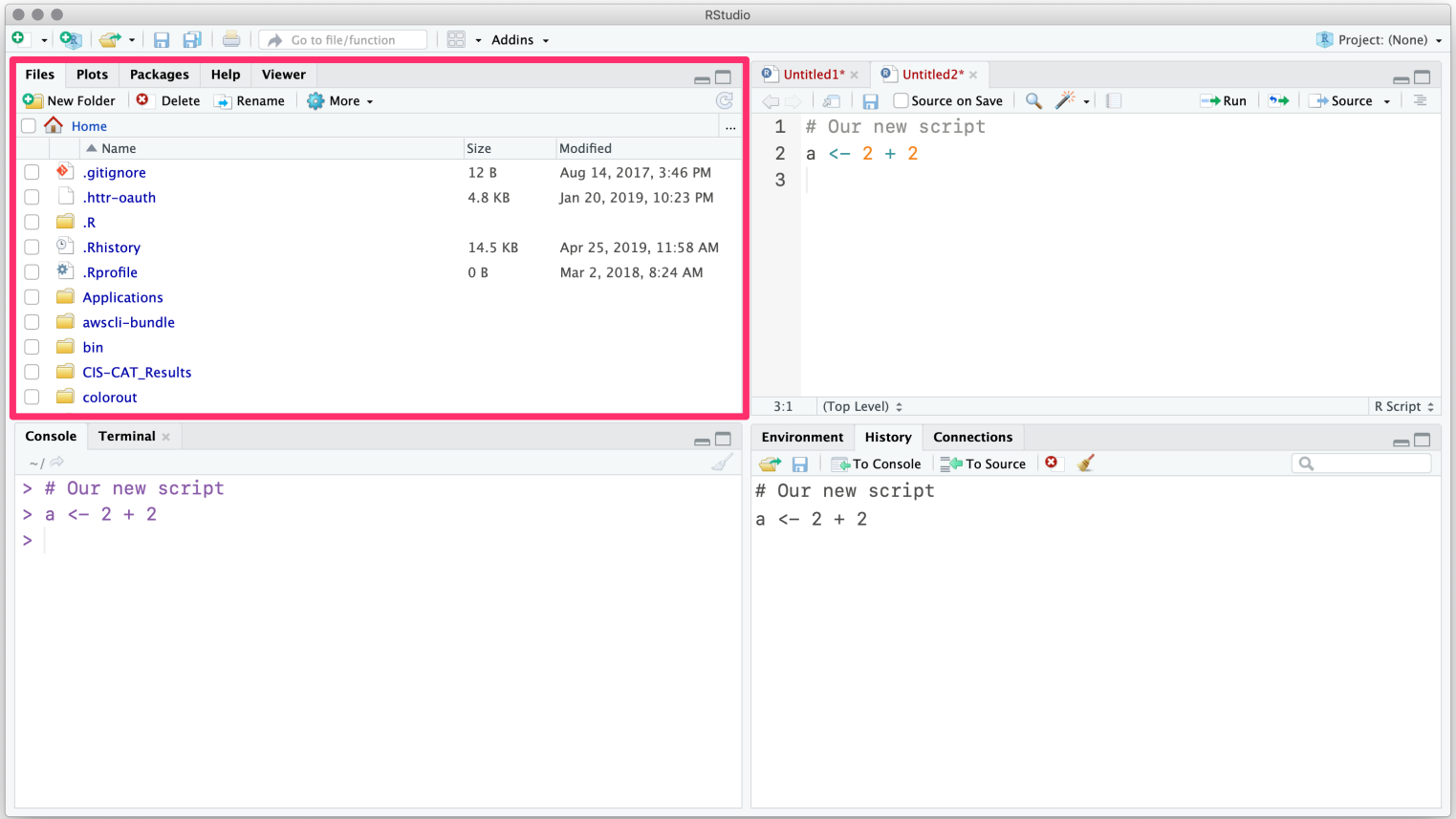
You can see our new object in the **Environment** pane.



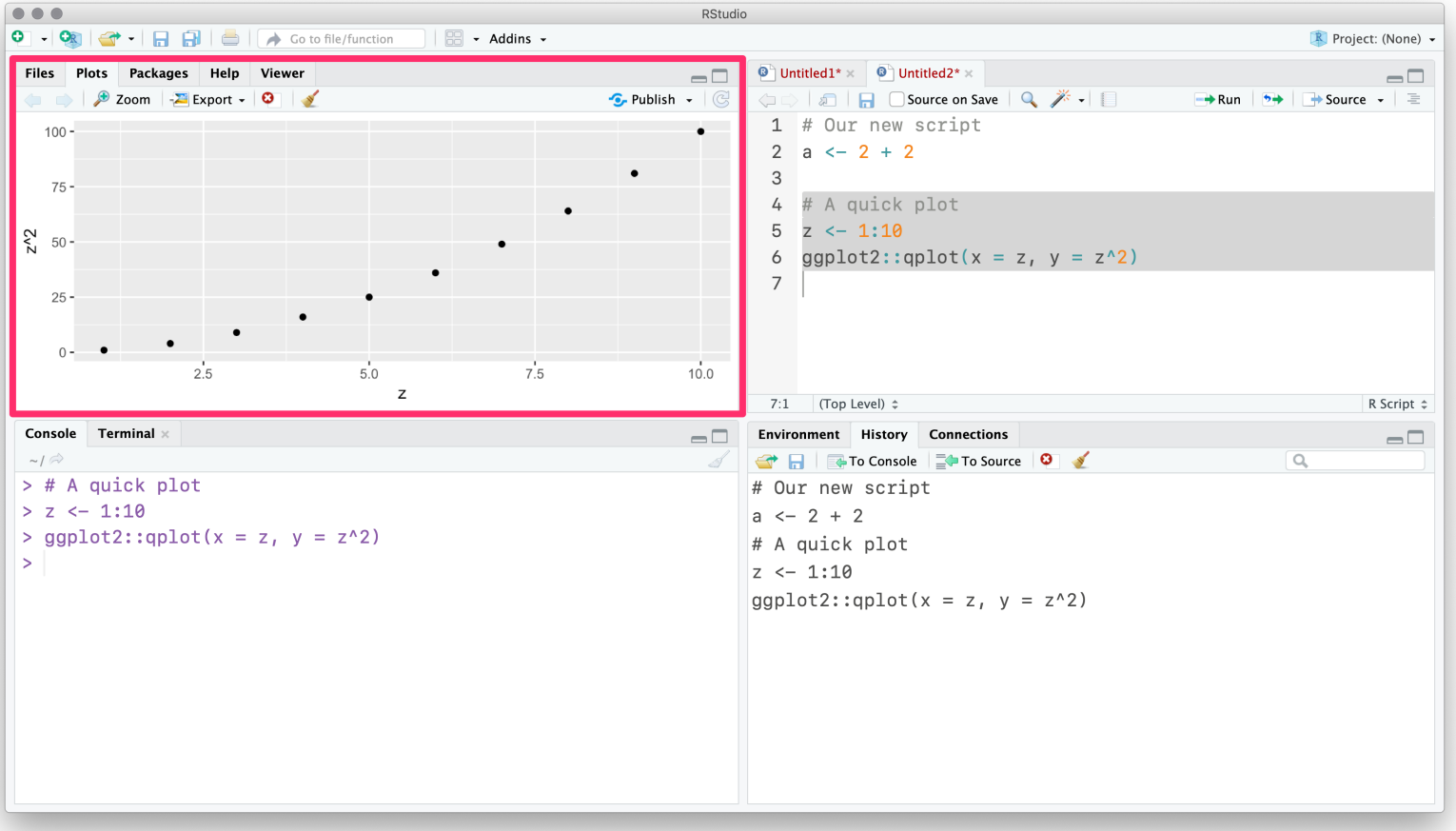
The **History** tab (next to **Environment**) records your old commands.



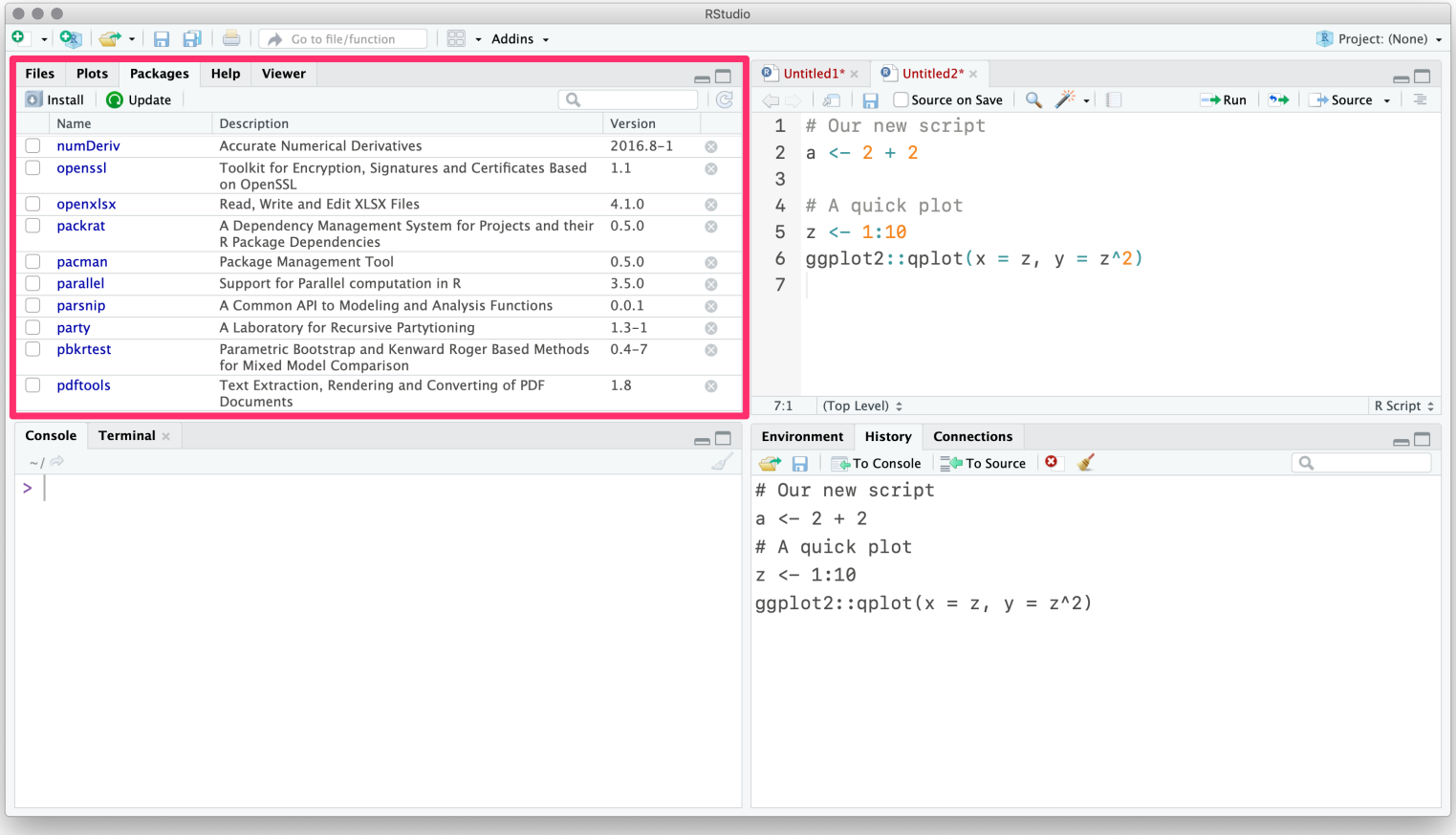
The **Files** pane is file explorer.



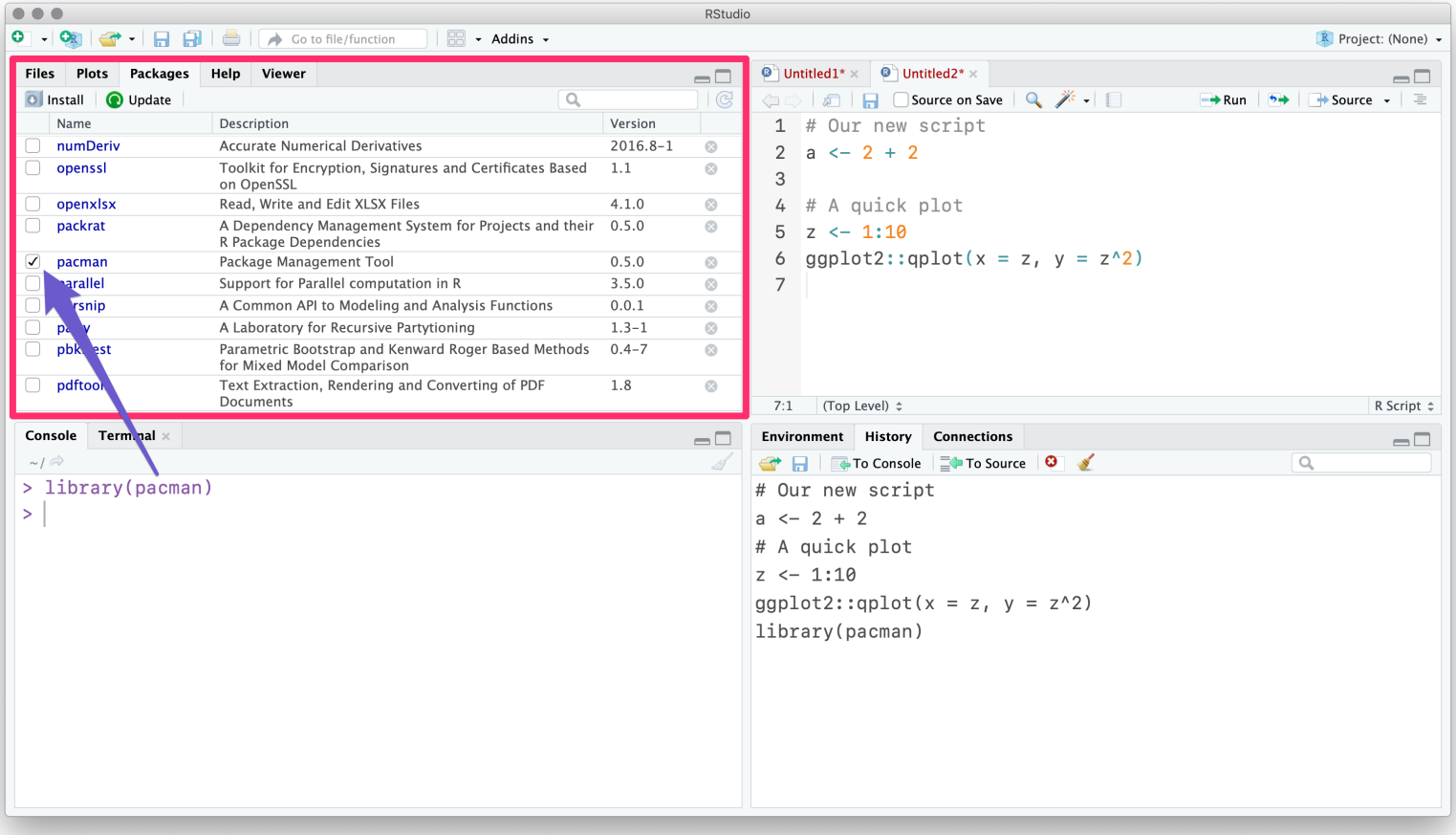
The **Plots** pane/tab shows... plots.



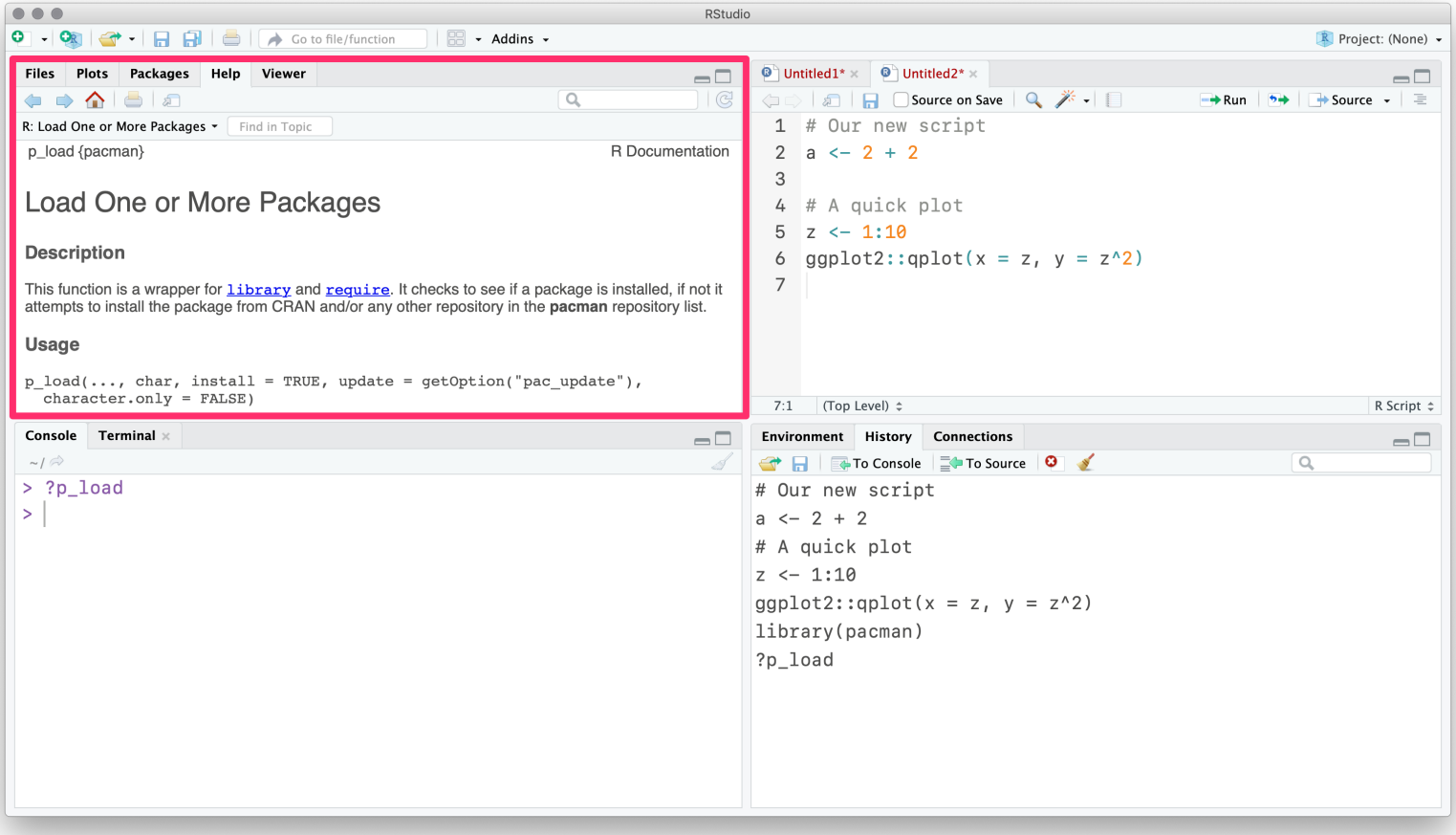
Packages shows installed packages



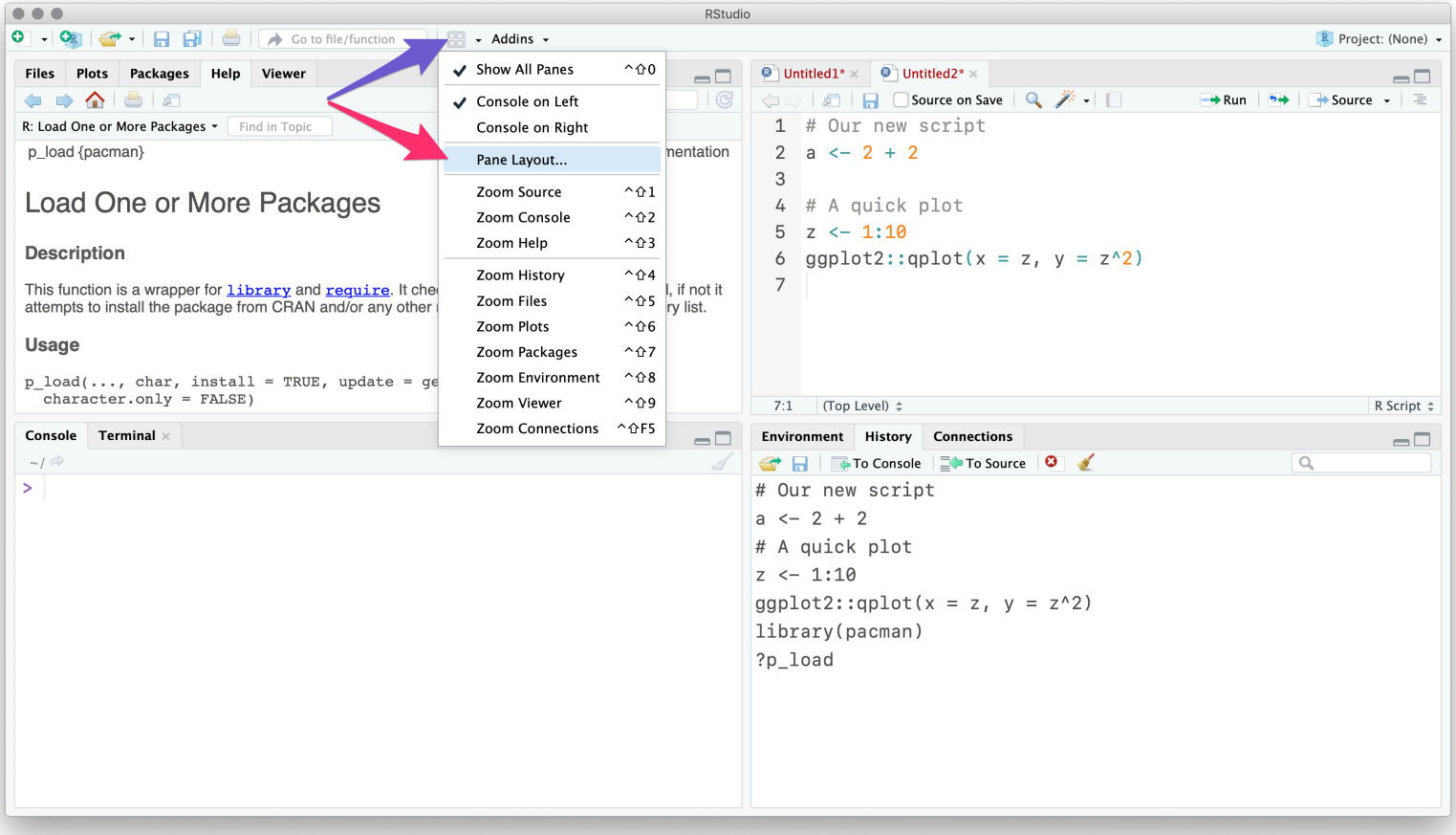
Packages shows installed packages and whether they are **loaded**.



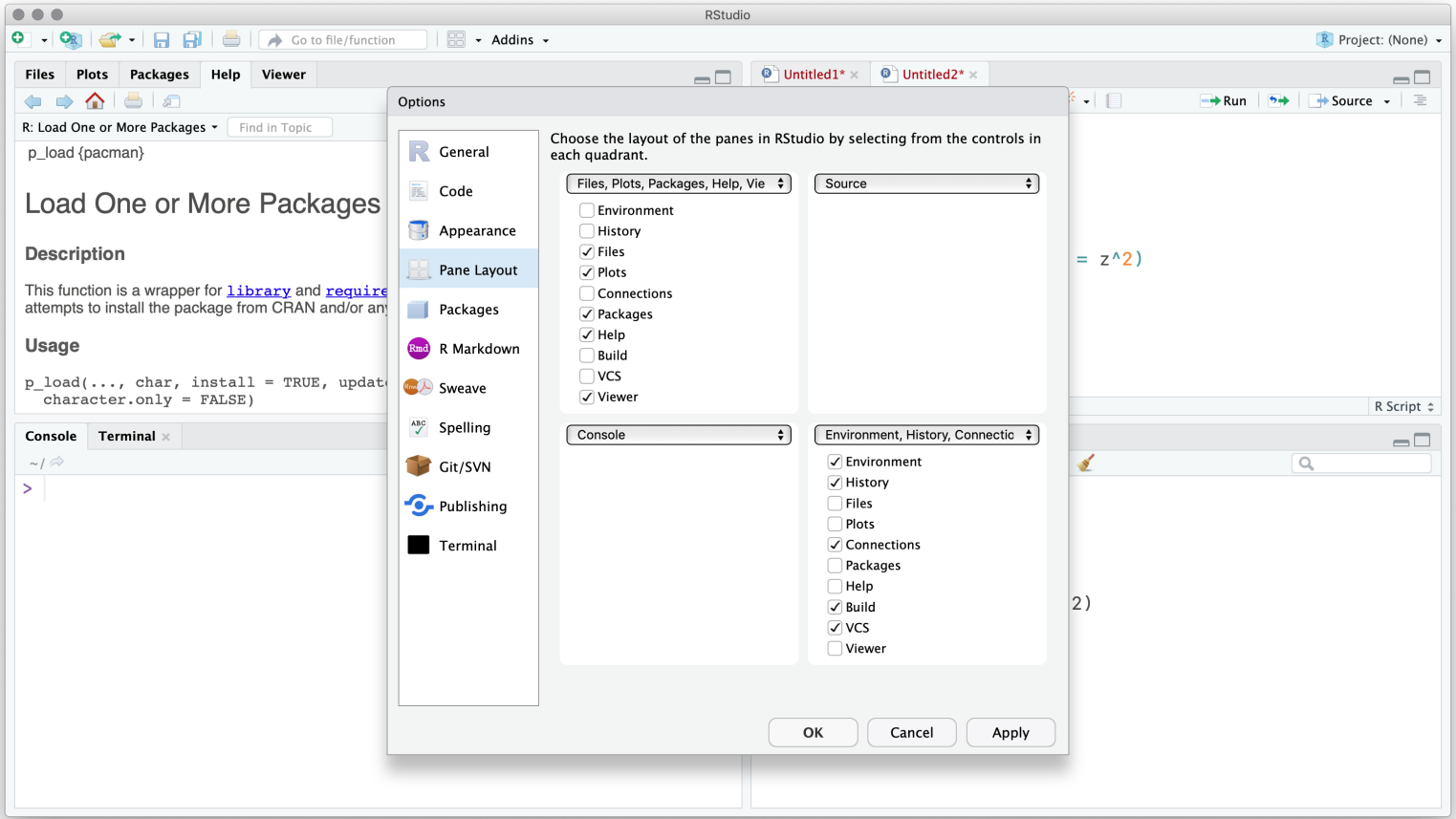
The **Help** tab shows help documentation (also accessible via `?`).



Finally, you can customize the actual layout



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



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.

    - **BAD** `ALLCARS`, `Vl123a8`, `a.fun`, `cens.12931`, `cens.12933`
    - **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.

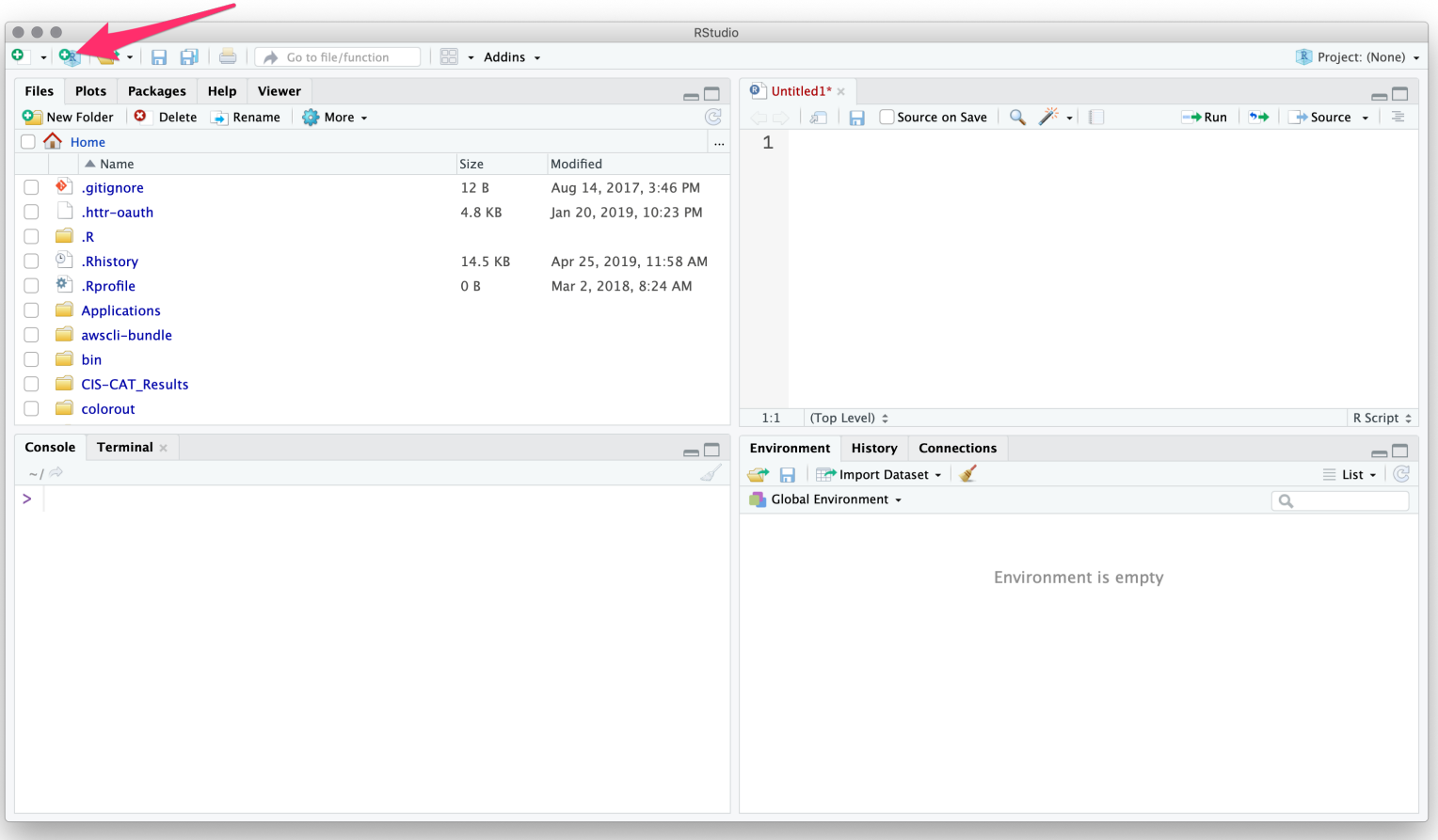
Projects

Projects in R offer several benefits:

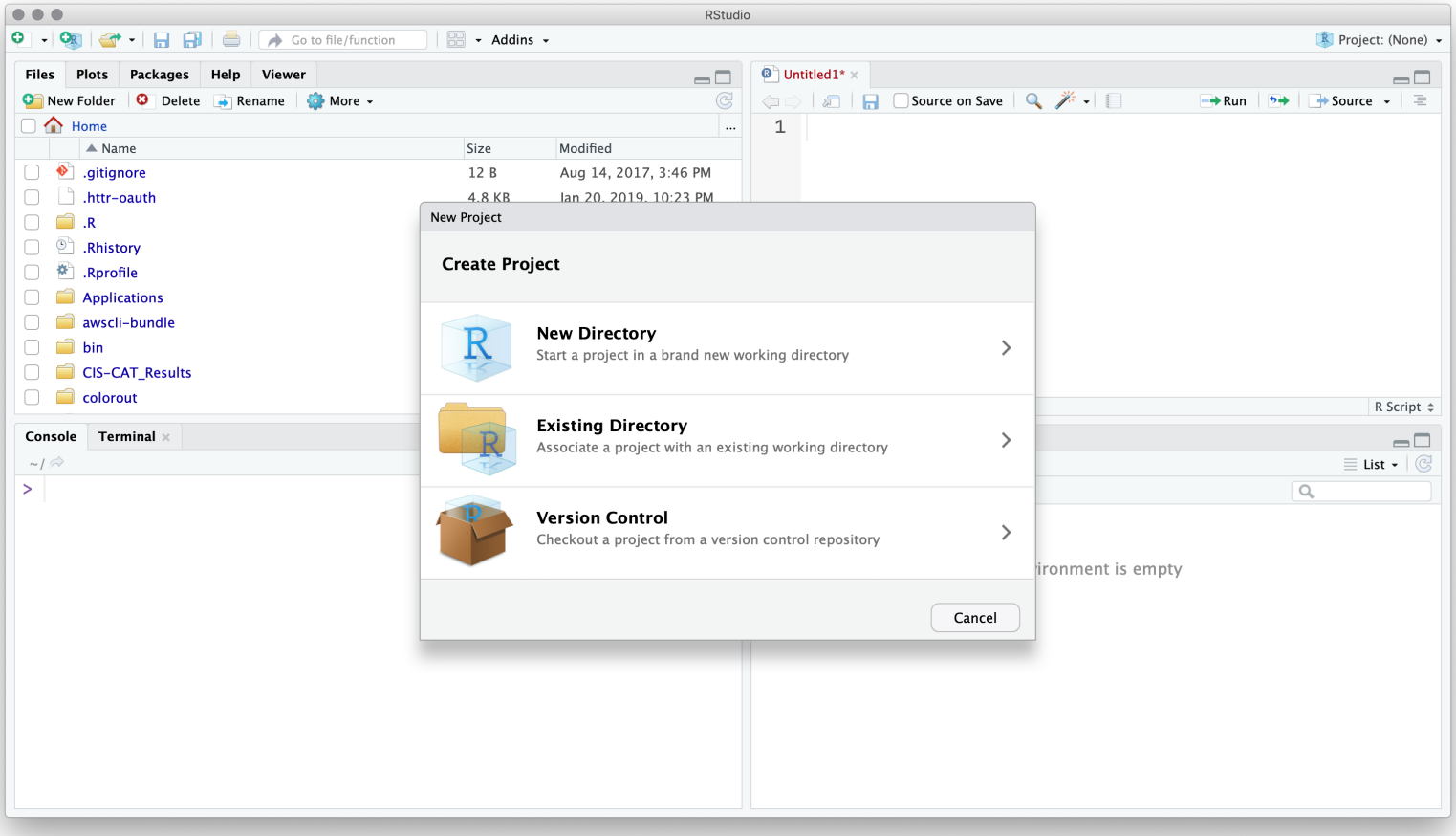
1. Act as an anchor for working with files.
2. Make your work (projects) easily reproducible.†
3. Help you quickly jump back into your work.

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

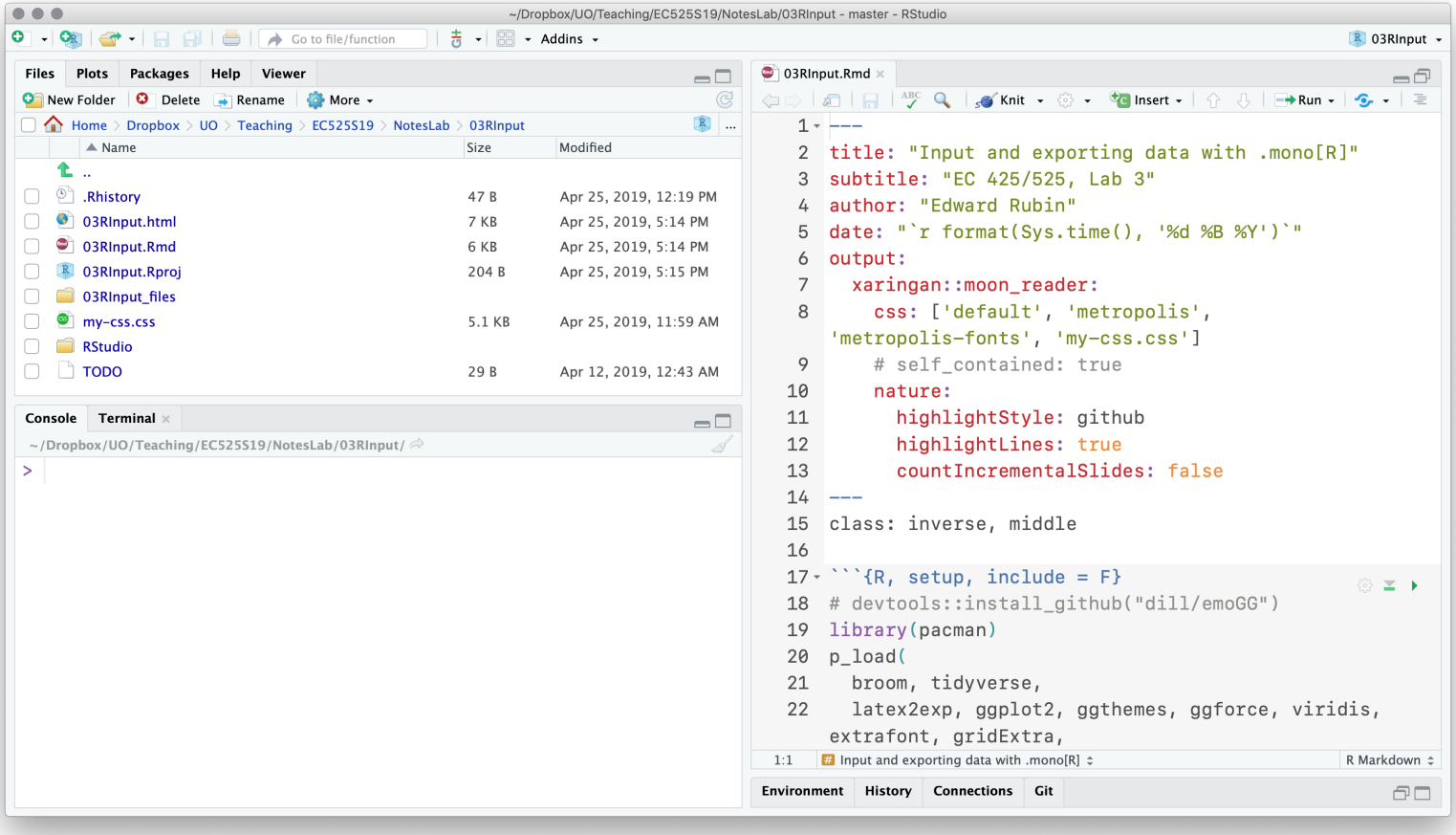
To start a new project, hit the **project icon**.



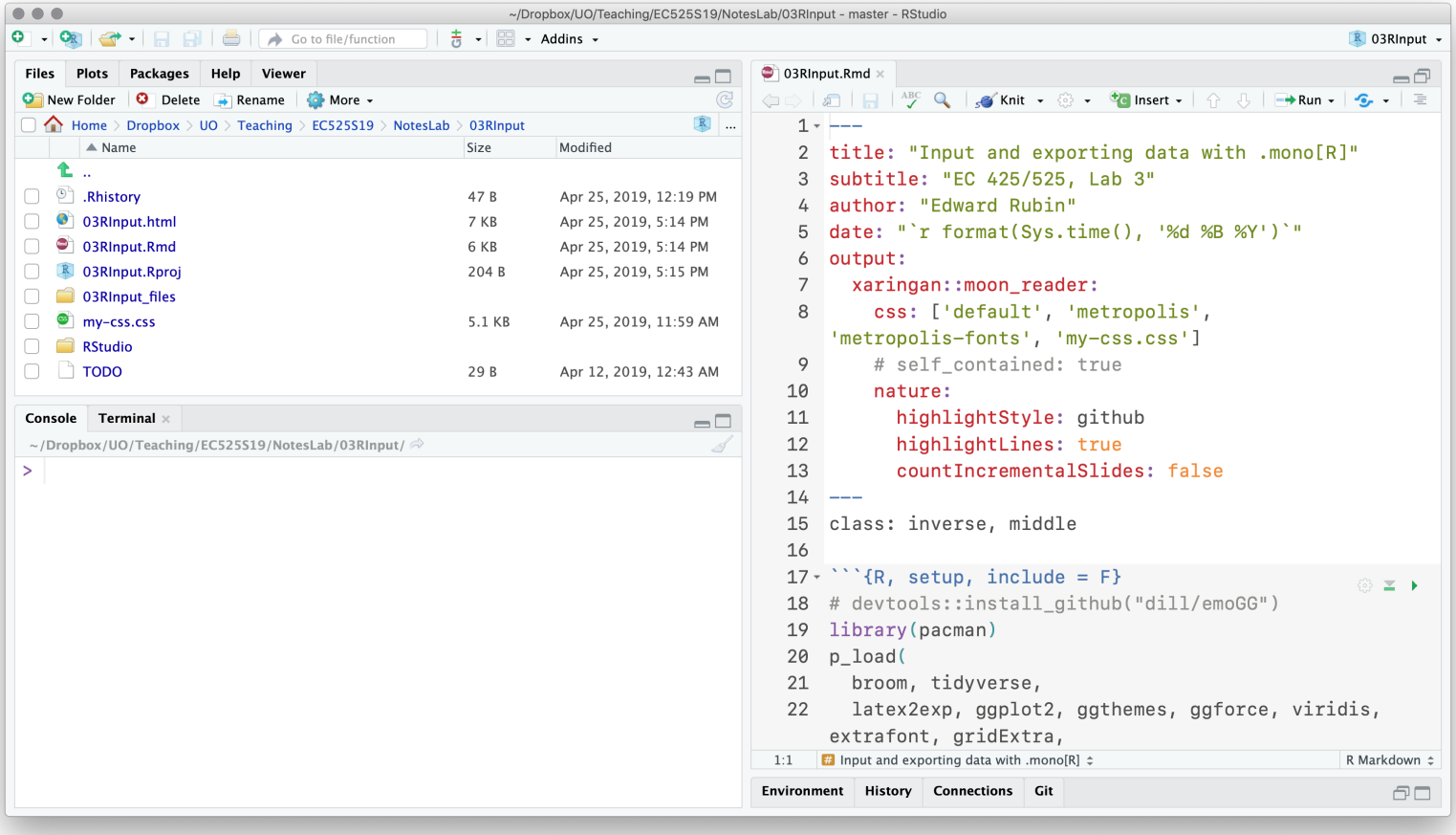
You'll then choose the folder/directory where your project lives.



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



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



Projects

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

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/UO/Teaching/EC525S19/"
dir_labs ← paste0(dir_class, "NotesLab/")
dir_lab03 ← paste0(dir_labs, "03RInput/")
sample_df ← read.csv(paste0(dir_lab03, "sample.csv"))
```

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/UO/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")
```

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/UO/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

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()`.

Reading CSVs

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

```
dir()
```

```
#> [1] "03RInput_cache" "03RInput_files" "03RInput.html"  "03RInput.Rmd"
#> [5] "03RInput.Rproj" "my-css.css"     "RStudio"        "sample.csv"
#> [9] "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.*

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.

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"
)
```

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`