Lab 000

Data cleaning and workflow (1/N)

Edward Rubin
10 January 2020

Admin

Admin

Basic **workflow** (best) practices (*i.e., Projects*)

- RStudio and projects
- Naming conventions
- Pipes (`%>%`)
- Data cleaning with `dplyr`

Reminder Readings for next week

- ISL Ch1–Ch2
- Prediction Policy Problems by Kleinberg *et al.* (2015)

Improving your workflow

Improving your workflow

Data cleaning, manipulation, and analysis can be grueling, but optimizing your workflow can speed things along and make them less painful.†

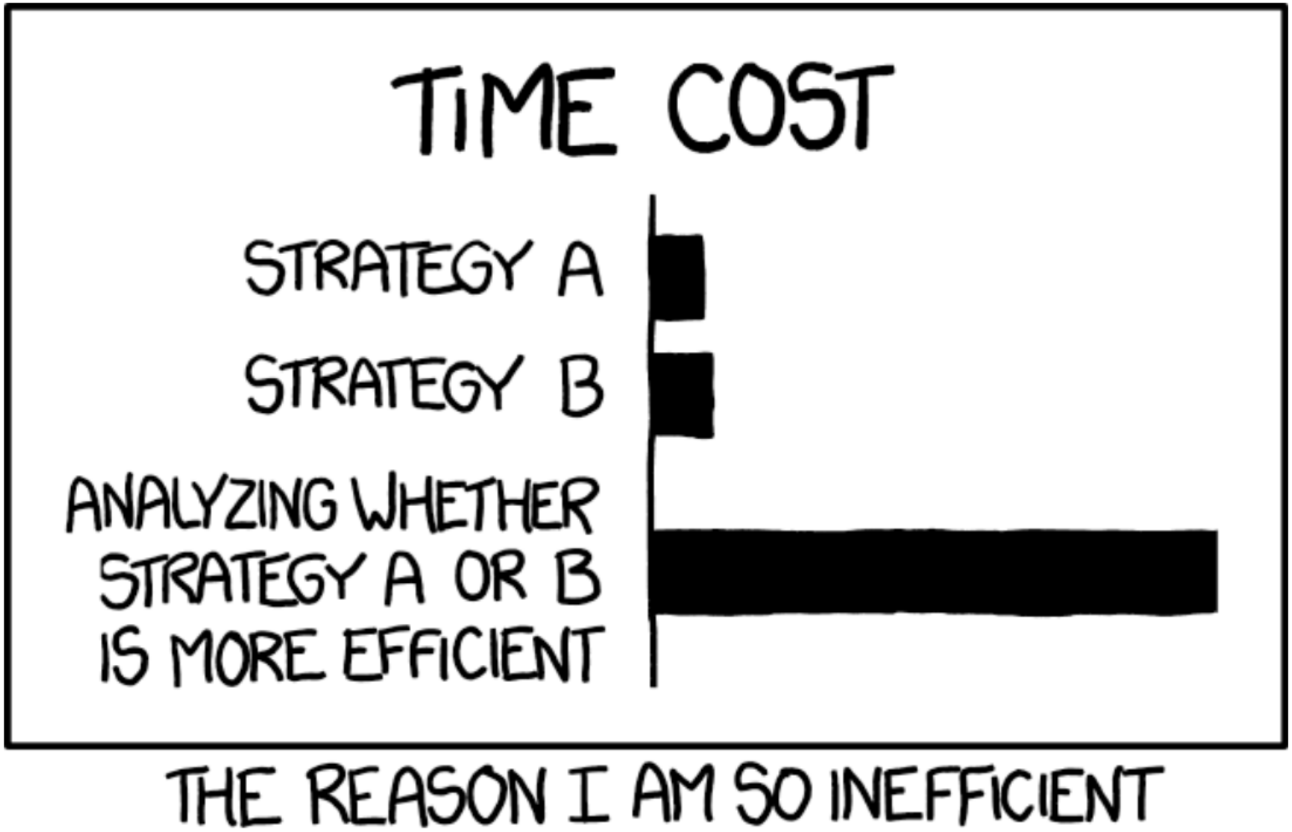
A few dimensions that can help

- Understand how to interact with RStudio
- Use `R` projects
- Follow reasonable naming conventions
- `dplyr` and pipes
- Write your own functions (future lab)
- Use loops and parallelization (future lab)
- Hire an intern/assistant to do your work for you

---

† Notice that I said *less* painful.

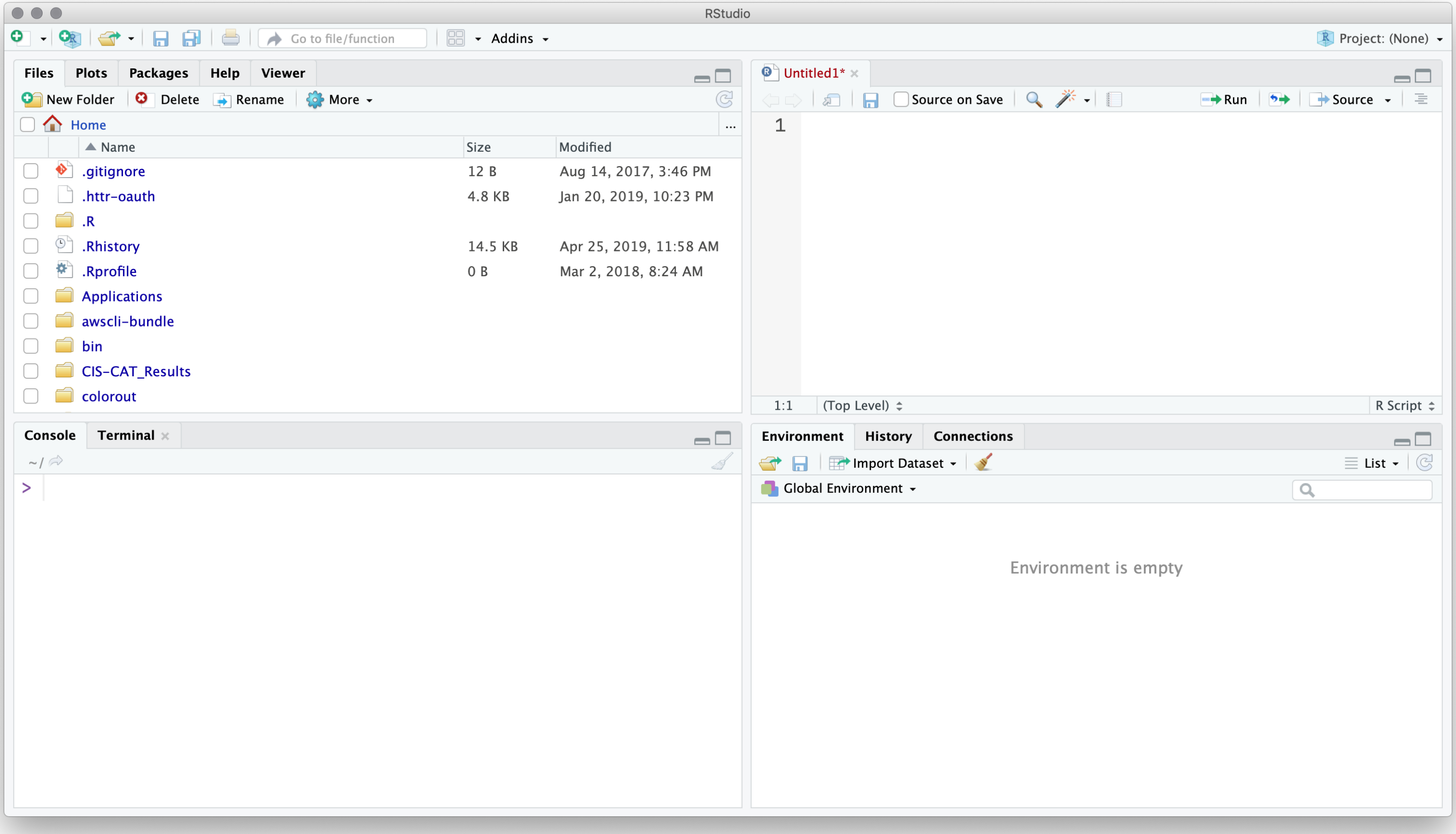
Efficiency



Source: xkcd

RStudio

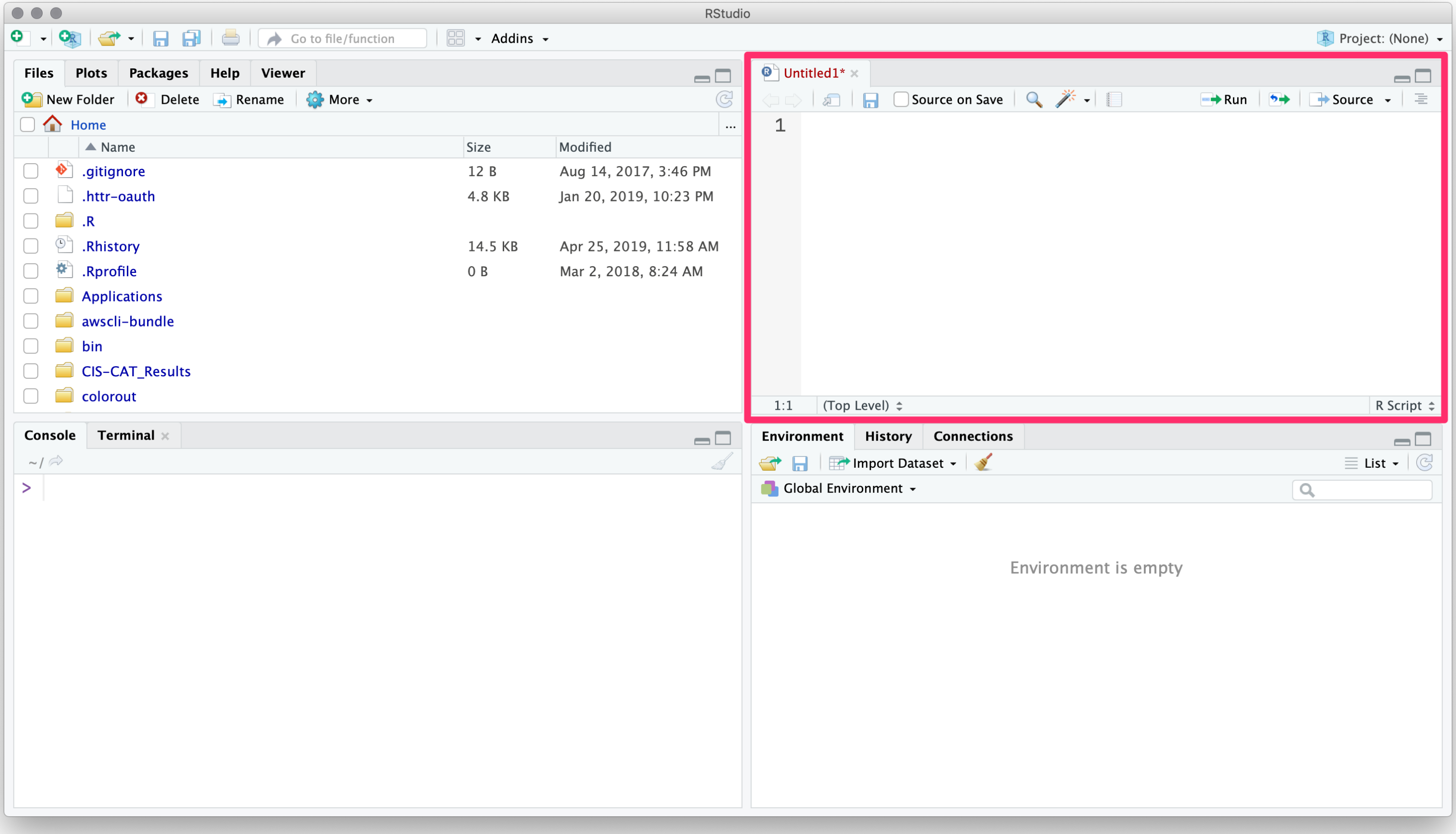
Let's recap some of the major features in **`RStudio`**...



Name	Size	Modified
.gitignore	12 B	Aug 14, 2017, 3:46 PM
.httr-oauth	4.8 KB	Jan 20, 2019, 10:23 PM
.R		
.Rhistory	14.5 KB	Apr 25, 2019, 11:58 AM
.Rprofile	0 B	Mar 2, 2018, 8:24 AM
Applications		
awscli-bundle		
bin		
CIS-CAT_Results		
colorout		

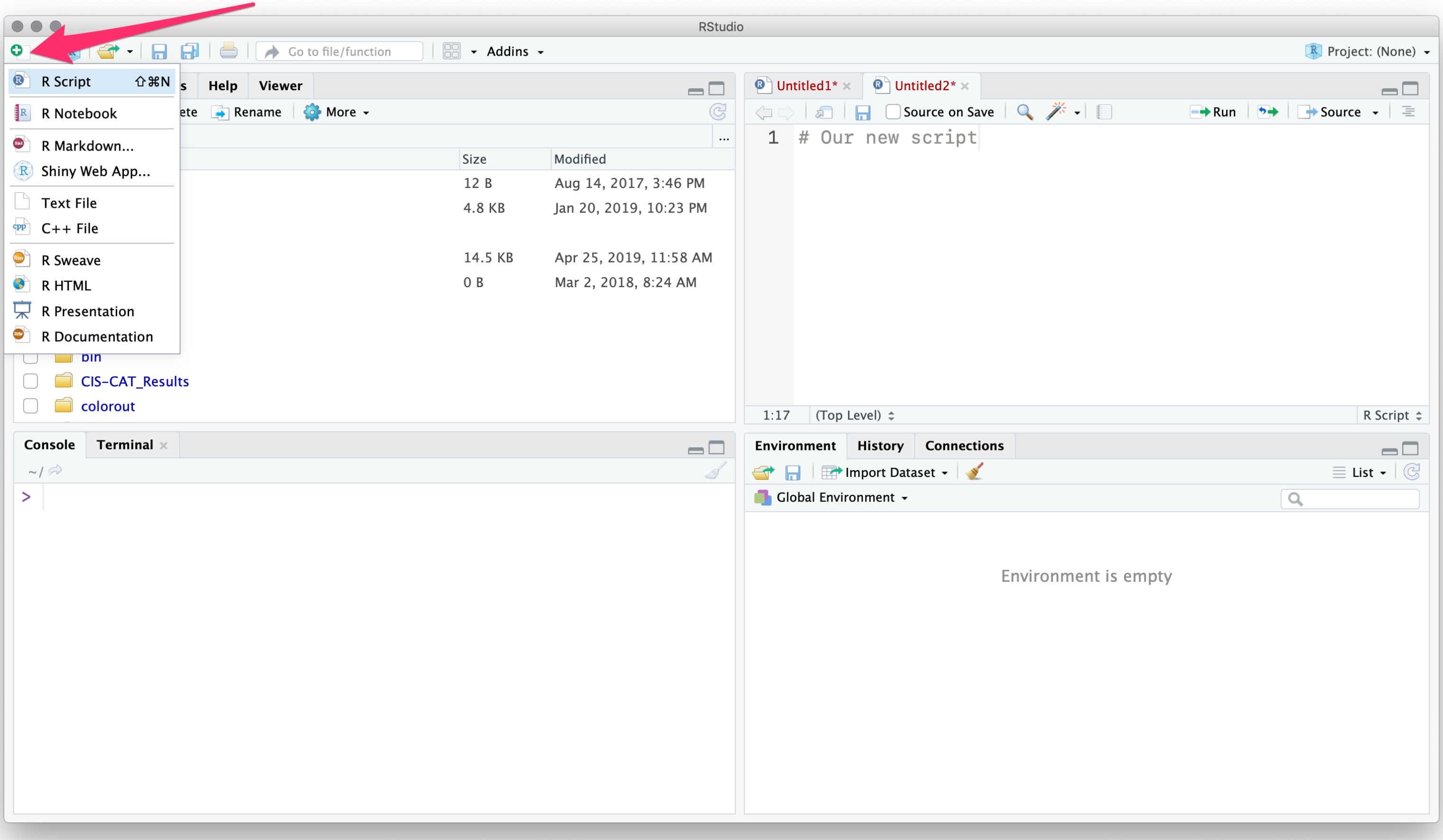
Environment is empty

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



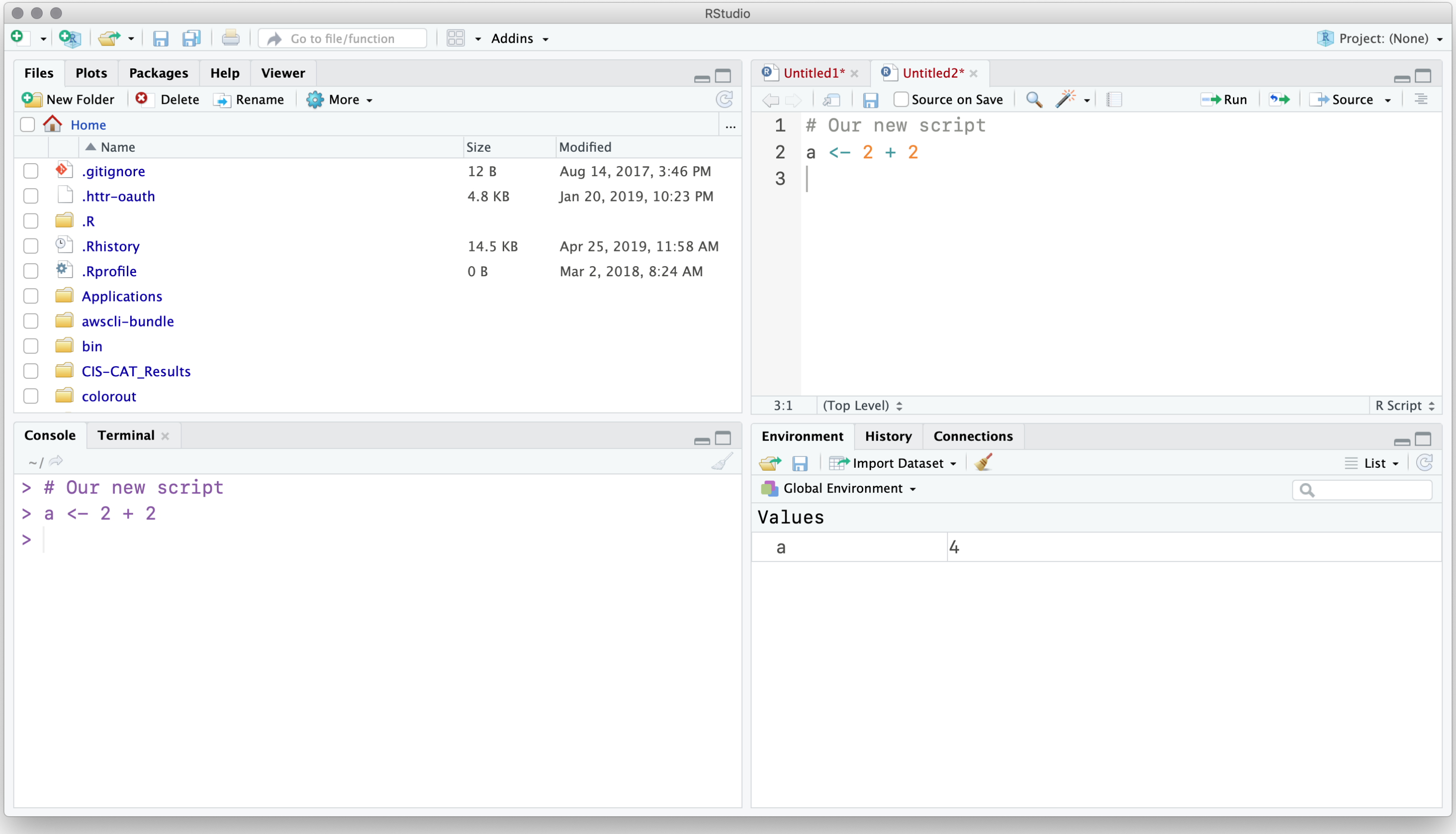
Name	Size	Modified
.gitignore	12 B	Aug 14, 2017, 3:46 PM
.httr-oauth	4.8 KB	Jan 20, 2019, 10:23 PM
.R		
.Rhistory	14.5 KB	Apr 25, 2019, 11:58 AM
.Rprofile	0 B	Mar 2, 2018, 8:24 AM
Applications		
awscli-bundle		
bin		
CIS-CAT_Results		
colorout		

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



```
1  # Our new script
```

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



Name	Size	Modified
.gitignore	12 B	Aug 14, 2017, 3:46 PM
.httr-oauth	4.8 KB	Jan 20, 2019, 10:23 PM
.R		
.Rhistory	14.5 KB	Apr 25, 2019, 11:58 AM
.Rprofile	0 B	Mar 2, 2018, 8:24 AM
Applications		
awscli-bundle		
bin		
CIS-CAT_Results		
colorout		

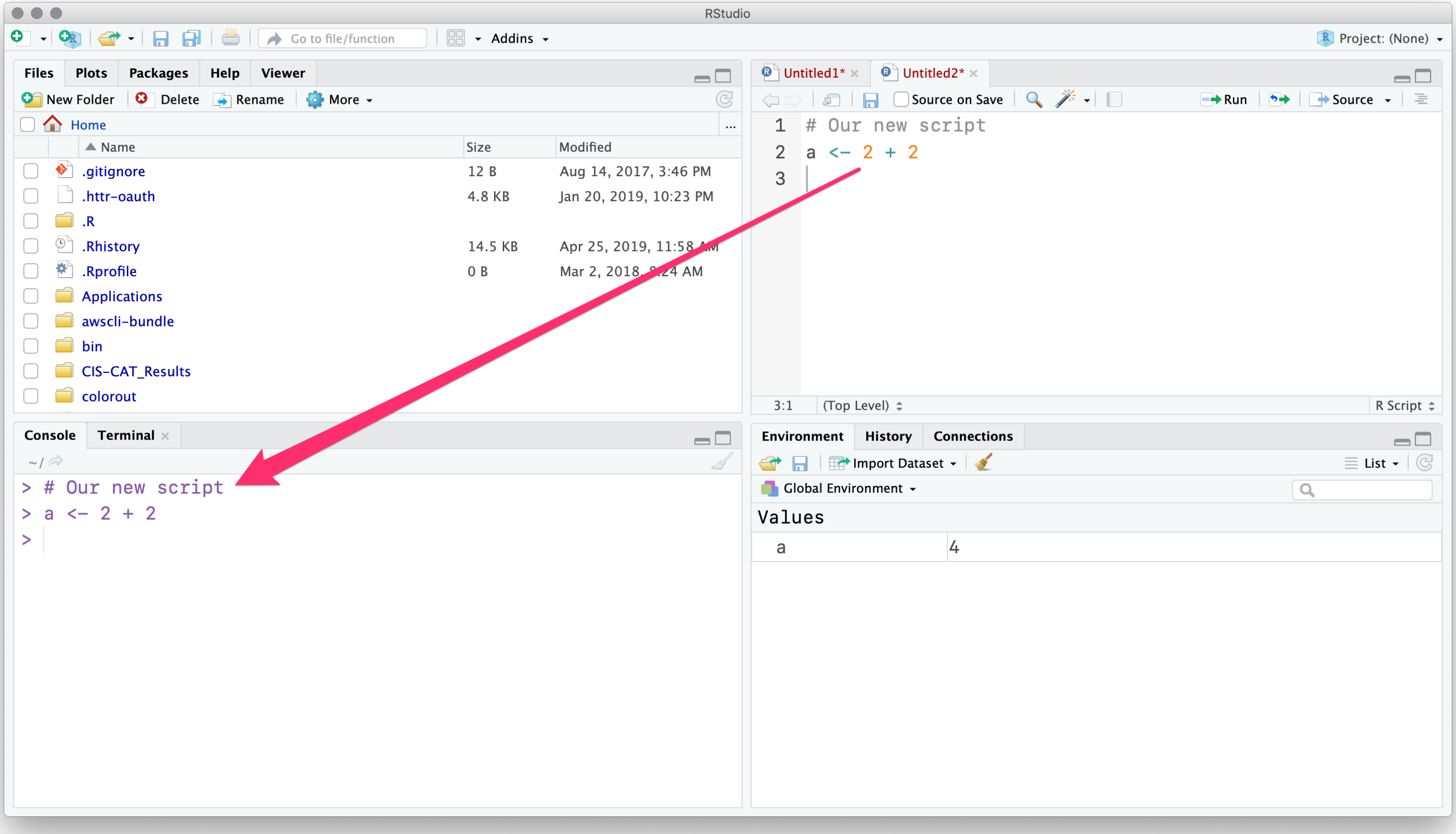
```
# Our new script
a <- 2 + 2

```

```
> # Our new script
> a <- 2 + 2
>
```

Values

a	4

`RStudio` will execute the command in the terminal.



Name	Size	Modified
.gitignore	12 B	Aug 14, 2017, 3:46 PM
.httr-oauth	4.8 KB	Jan 20, 2019, 10:23 PM
.R		
.Rhistory	14.5 KB	Apr 25, 2019, 11:58 AM
.Rprofile	0 B	Mar 2, 2018, [illegible]:24 AM
Applications		
awscli-bundle		
bin		
CIS-CAT_Results		
colorout		

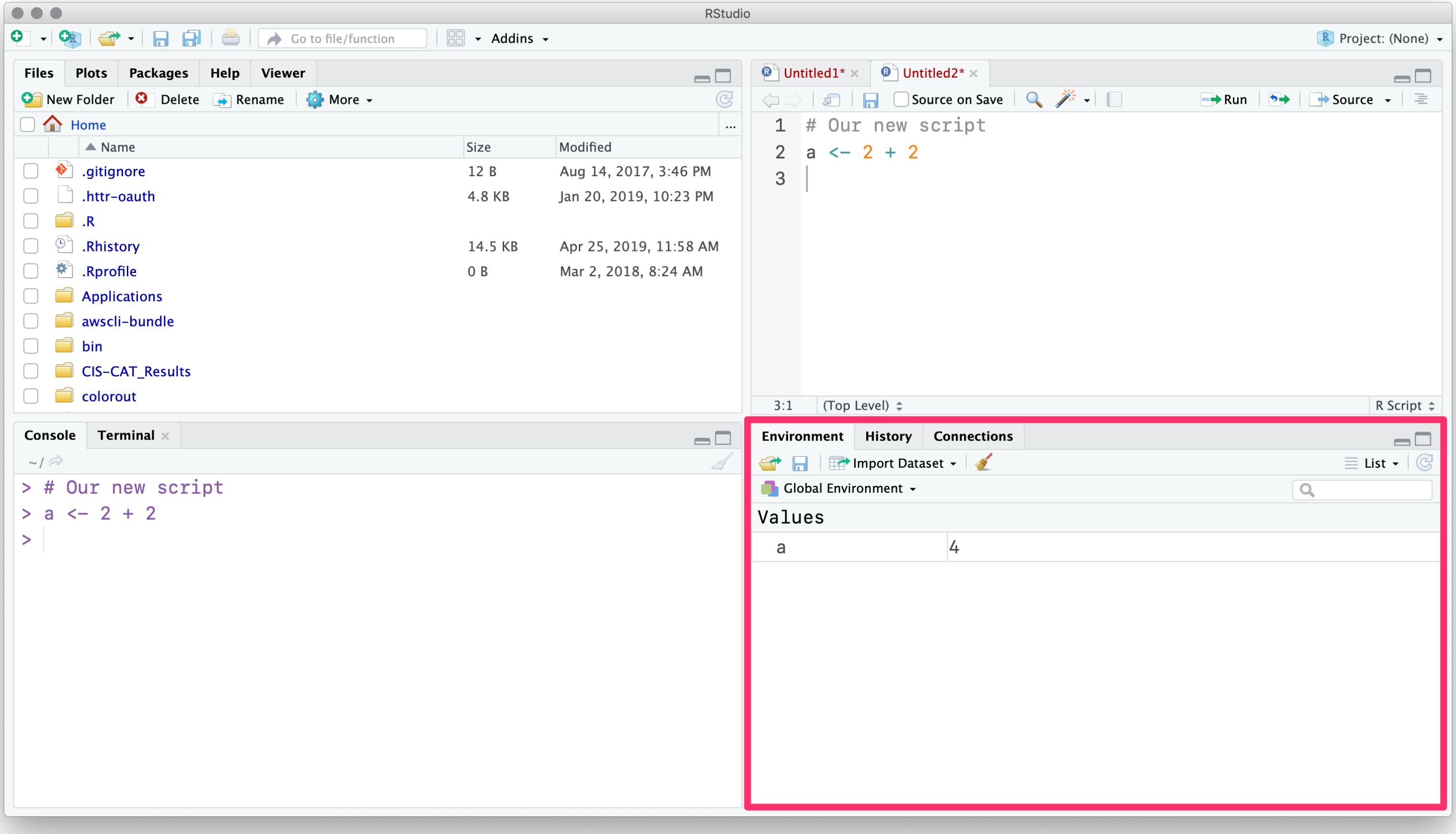
```
# Our new script
a <- 2 + 2
```

```
> # Our new script
> a <- 2 + 2
>
```

Values

a	4

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



Name	Size	Modified
.gitignore	12 B	Aug 14, 2017, 3:46 PM
.httr-oauth	4.8 KB	Jan 20, 2019, 10:23 PM
.R		
.Rhistory	14.5 KB	Apr 25, 2019, 11:58 AM
.Rprofile	0 B	Mar 2, 2018, 8:24 AM
Applications		
awscli-bundle		
bin		
CIS-CAT_Results		
colorout		

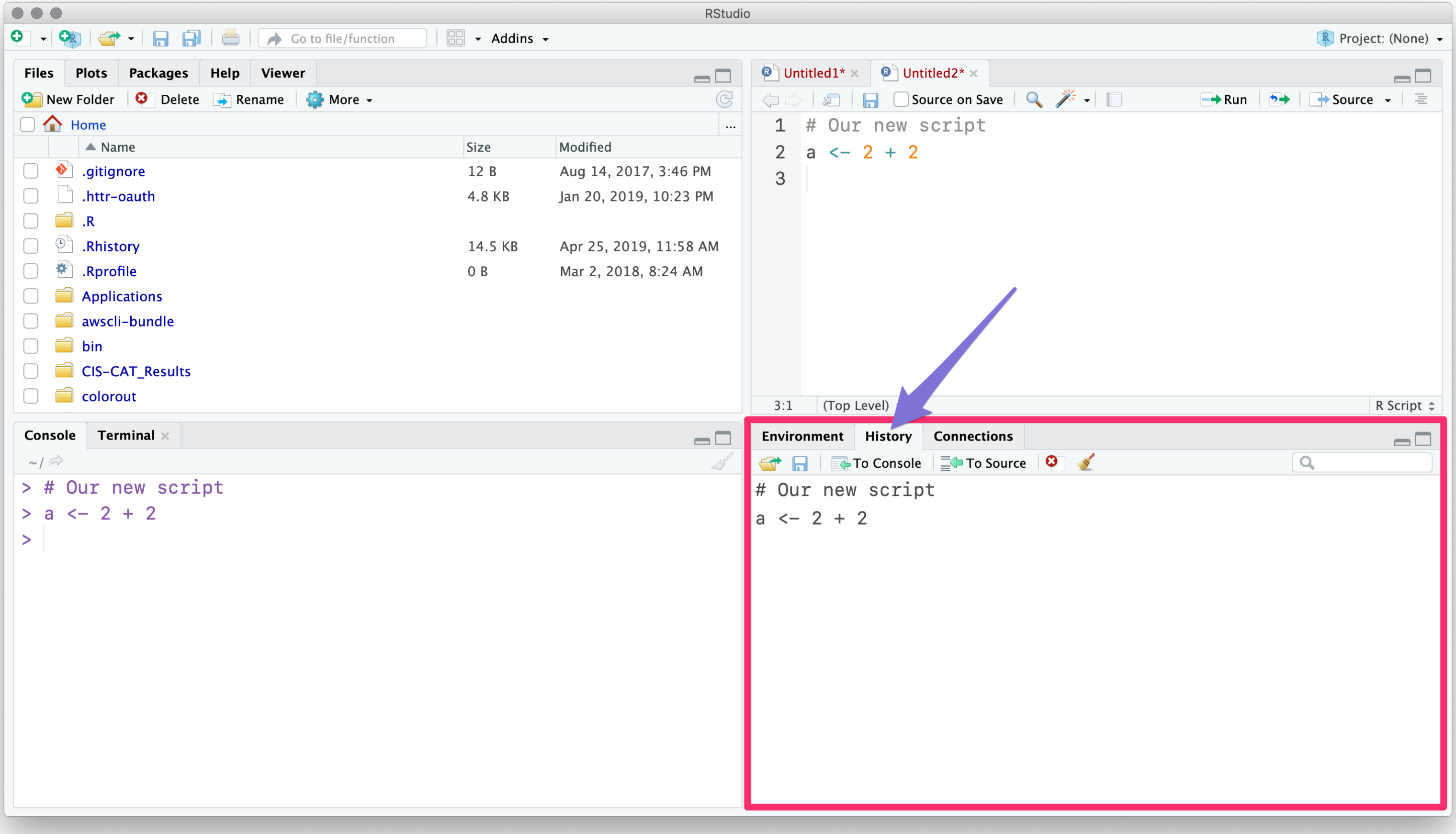
```r
# Our new script
a <- 2 + 2
```

```
> # Our new script
> a <- 2 + 2
>
```

Values

a	4

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



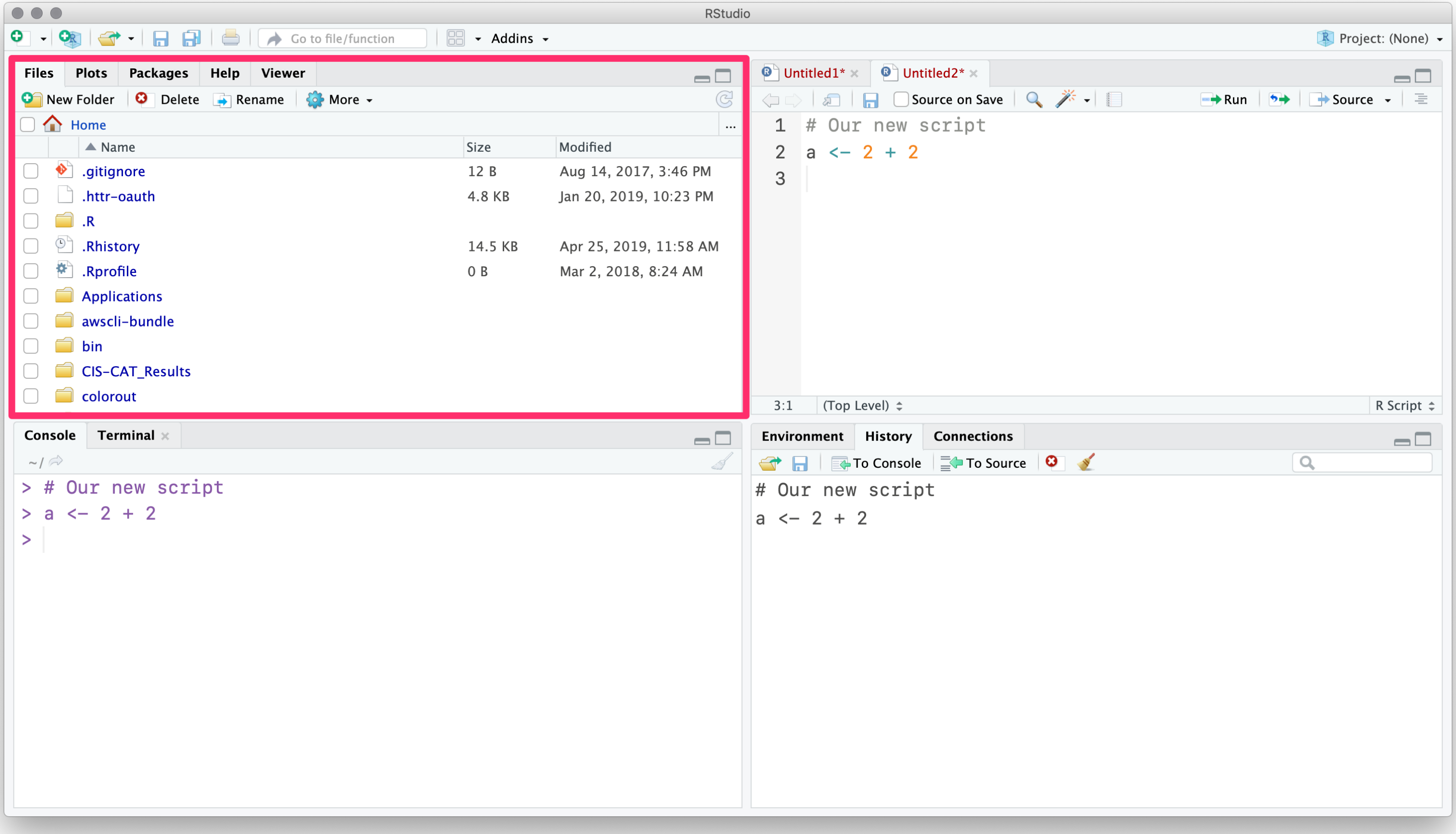
Name	Size	Modified
.gitignore	12 B	Aug 14, 2017, 3:46 PM
.httr-oauth	4.8 KB	Jan 20, 2019, 10:23 PM
.R		
.Rhistory	14.5 KB	Apr 25, 2019, 11:58 AM
.Rprofile	0 B	Mar 2, 2018, 8:24 AM
Applications		
awscli-bundle		
bin		
CIS-CAT_Results		
colorout		

```r
# Our new script
a <- 2 + 2
```

```
> # Our new script
> a <- 2 + 2
>
```

```
# Our new script
a <- 2 + 2
```

The **Files** pane is file explorer.



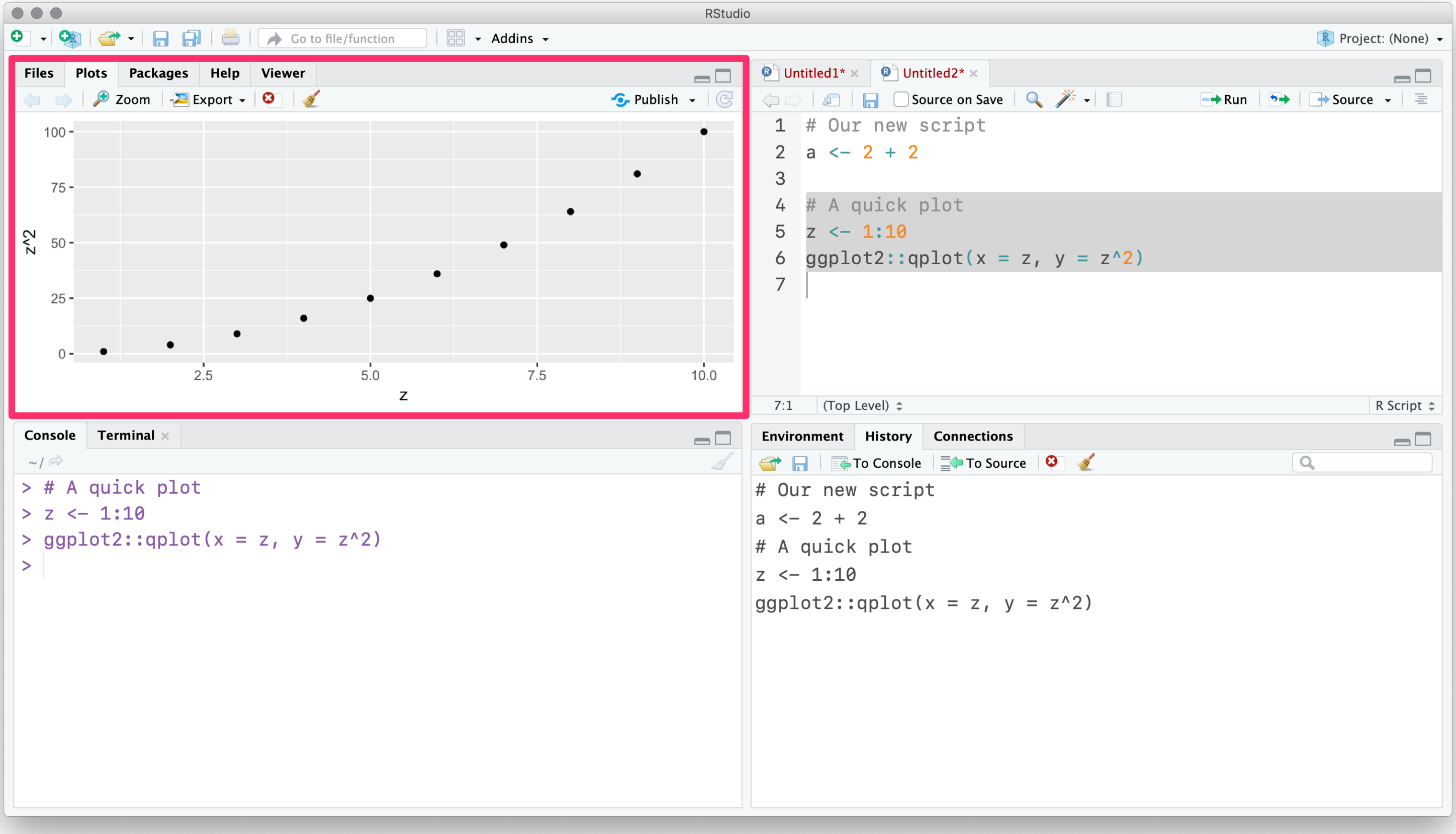
☐	Name	Size	Modified
☐	.gitignore	12 B	Aug 14, 2017, 3:46 PM
☐	.httr-oauth	4.8 KB	Jan 20, 2019, 10:23 PM
☐	.R		
☐	.Rhistory	14.5 KB	Apr 25, 2019, 11:58 AM
☐	.Rprofile	0 B	Mar 2, 2018, 8:24 AM
☐	Applications		
☐	awscli-bundle		
☐	bin		
☐	CIS-CAT_Results		
☐	colorout		

```r
# Our new script
a <- 2 + 2

```

```
> # Our new script
> a <- 2 + 2
>
```

```
# Our new script
a <- 2 + 2
```

The **Plots** pane/tab shows… plots.

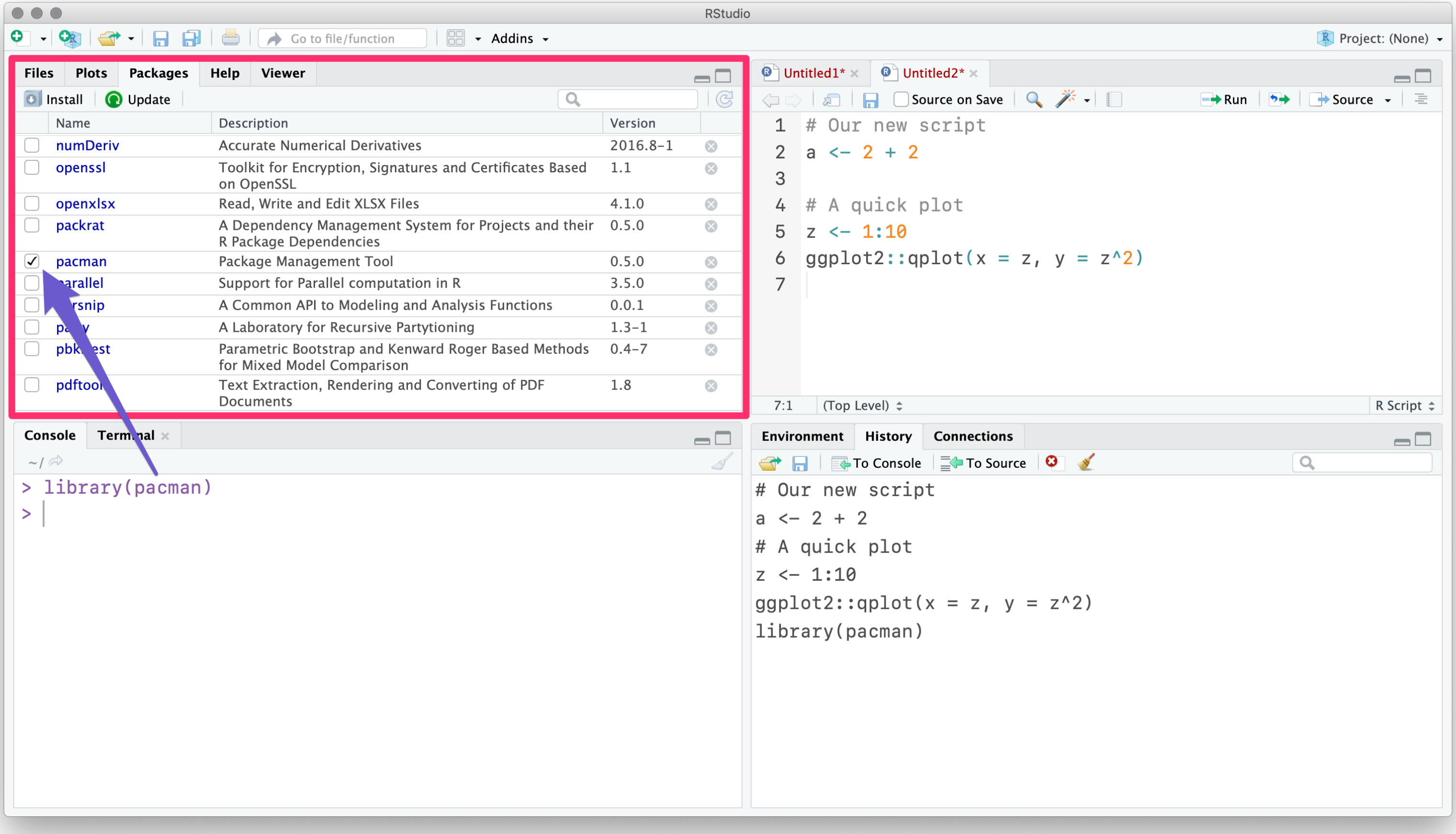


```r
# Our new script
a <- 2 + 2

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

```
> # A quick plot
> z <- 1:10
> ggplot2::qplot(x = z, y = z^2)
>
```

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

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



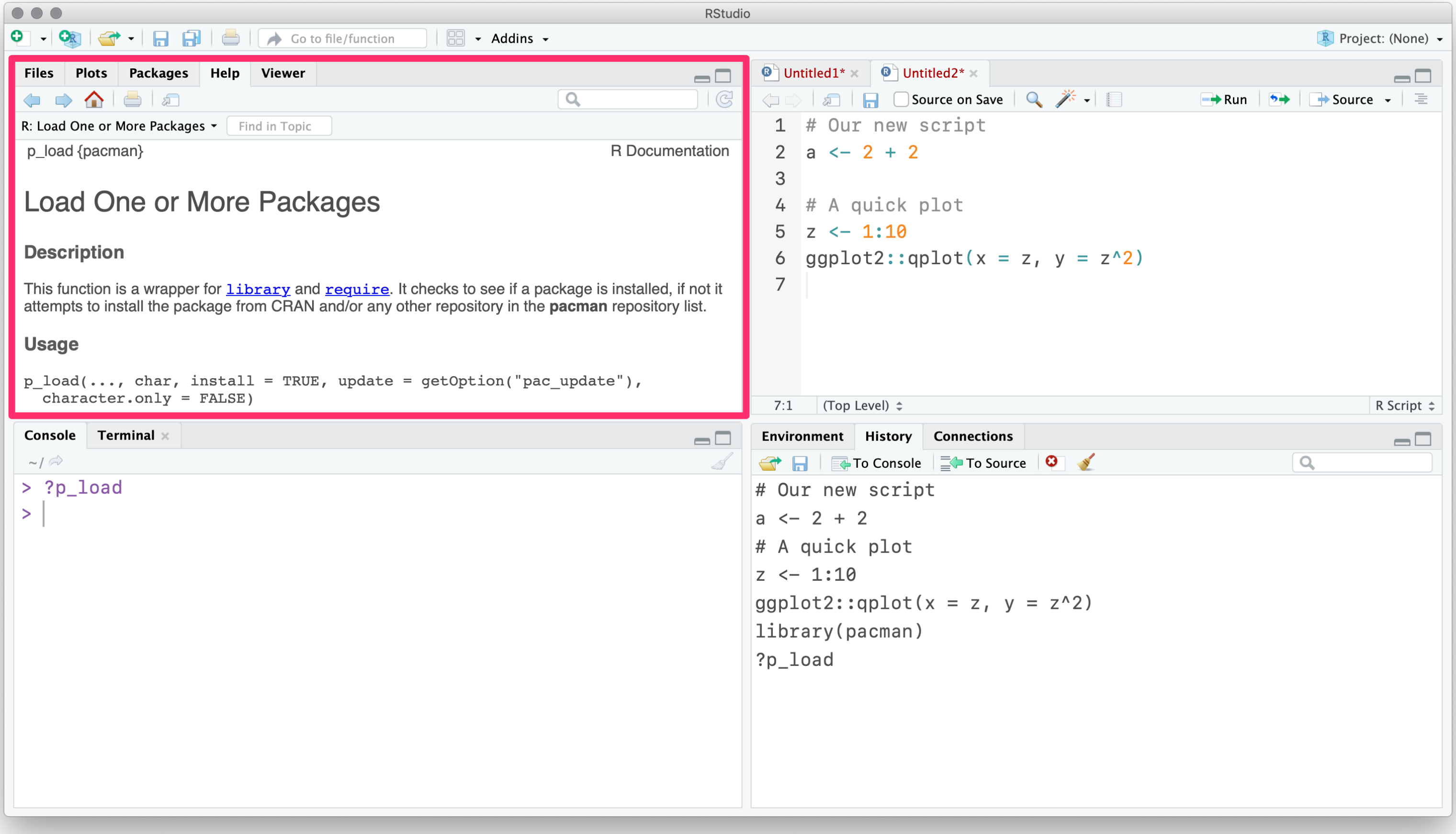
	Name	Description	Version
☐	numDeriv	Accurate Numerical Derivatives	2016.8-1
☐	openssl	Toolkit for Encryption, Signatures and Certificates Based on OpenSSL	1.1
☐	openxlsx	Read, Write and Edit XLSX Files	4.1.0
☐	packrat	A Dependency Management System for Projects and their R Package Dependencies	0.5.0
☑	pacman	Package Management Tool	0.5.0
☐	arallel	Support for Parallel computation in R	3.5.0
☐	rsnip	A Common API to Modeling and Analysis Functions	0.0.1
☐	pa y	A Laboratory for Recursive Partytioning	1.3-1
☐	pbk est	Parametric Bootstrap and Kenward Roger Based Methods for Mixed Model Comparison	0.4-7
☐	pdftool	Text Extraction, Rendering and Converting of PDF Documents	1.8

```
# Our new script
a <- 2 + 2

# A quick plot
z <- 1:10
ggplot2::qplot(x = z, y = z^2)

```

```
> library(pacman)
> 
```

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

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



p_load {pacman} R Documentation

Load One or More Packages

Description

This function is a wrapper for `library` and `require`. It checks to see if a package is installed, if not it attempts to install the package from CRAN and/or any other repository in the **pacman** repository list.

Usage

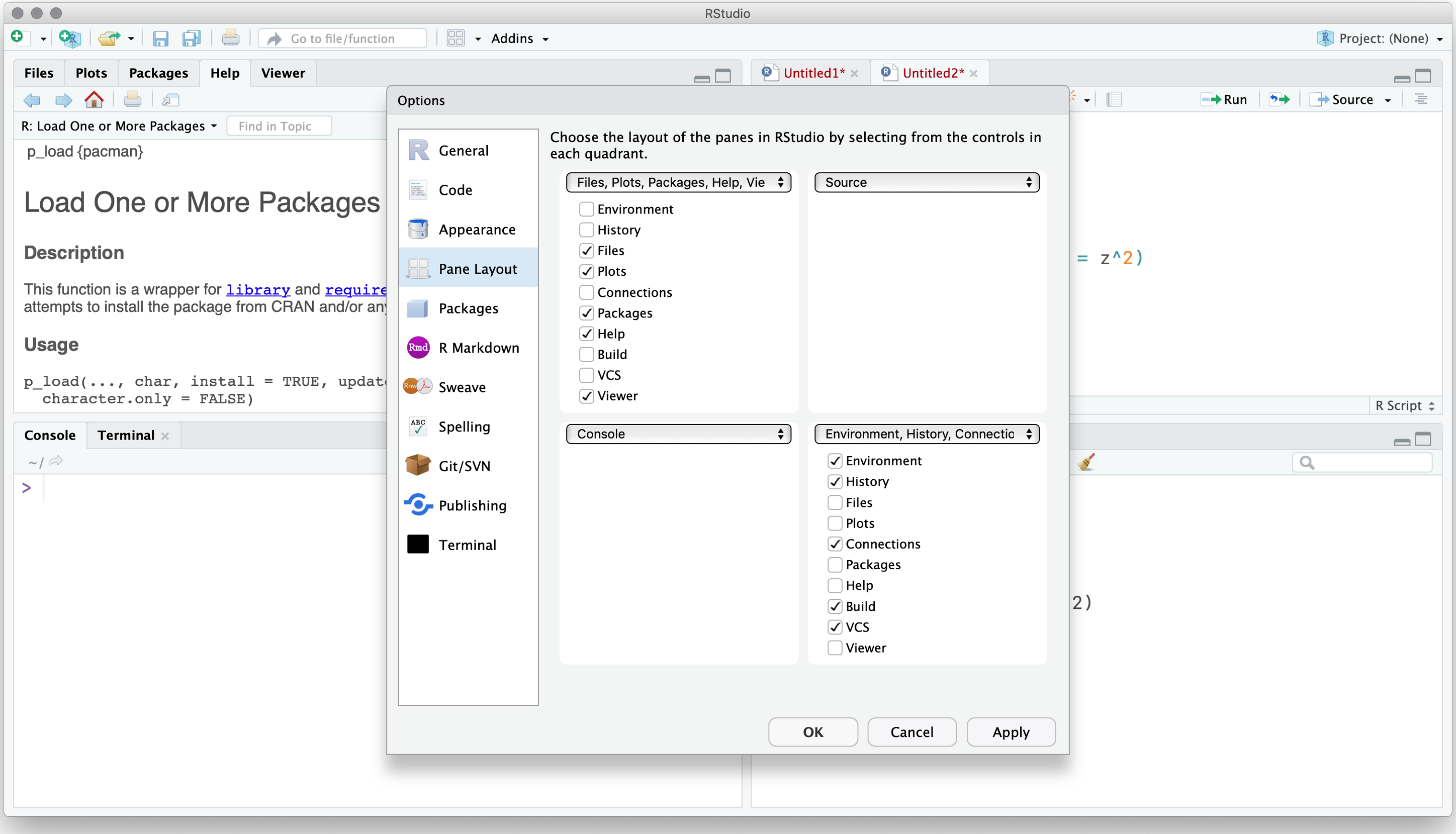
```
p_load(..., char, install = TRUE, update = getOption("pac_update"),
  character.only = FALSE)
```

```
# Our new script
a <- 2 + 2

# A quick plot
z <- 1:10
ggplot2::qplot(x = z, y = z^2)

```

```
> ?p_load
> 
```

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

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



Choose the layout of the panes in RStudio by selecting from the controls in each quadrant.

Files, Plots, Packages, Help, Vie

- ☐ Environment
- ☐ History
- ☑ Files
- ☑ Plots
- ☐ Connections
- ☑ Packages
- ☑ Help
- ☐ Build
- ☐ VCS
- ☑ Viewer

Source

Console

Environment, History, Connectio

- ☑ Environment
- ☑ History
- ☐ Files
- ☐ Plots
- ☑ Connections
- ☐ Packages
- ☐ Help
- ☑ Build
- ☑ VCS
- ☐ Viewer

R and RStudio

Related 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/variables/files with intelligible, standardized names.
    - **BAD** `ALLCARS`, `Vl123a8`, `a.fun`, `cens.12931`, `cens.12933`
    - **GOOD** `unique_cars`, `health_df`, `sim_fun`, `is_female`, `age`
4. Write code that is readable (see comments comment above).
5. Use projects in **RStudio** (next). And organize your projects.

Projects

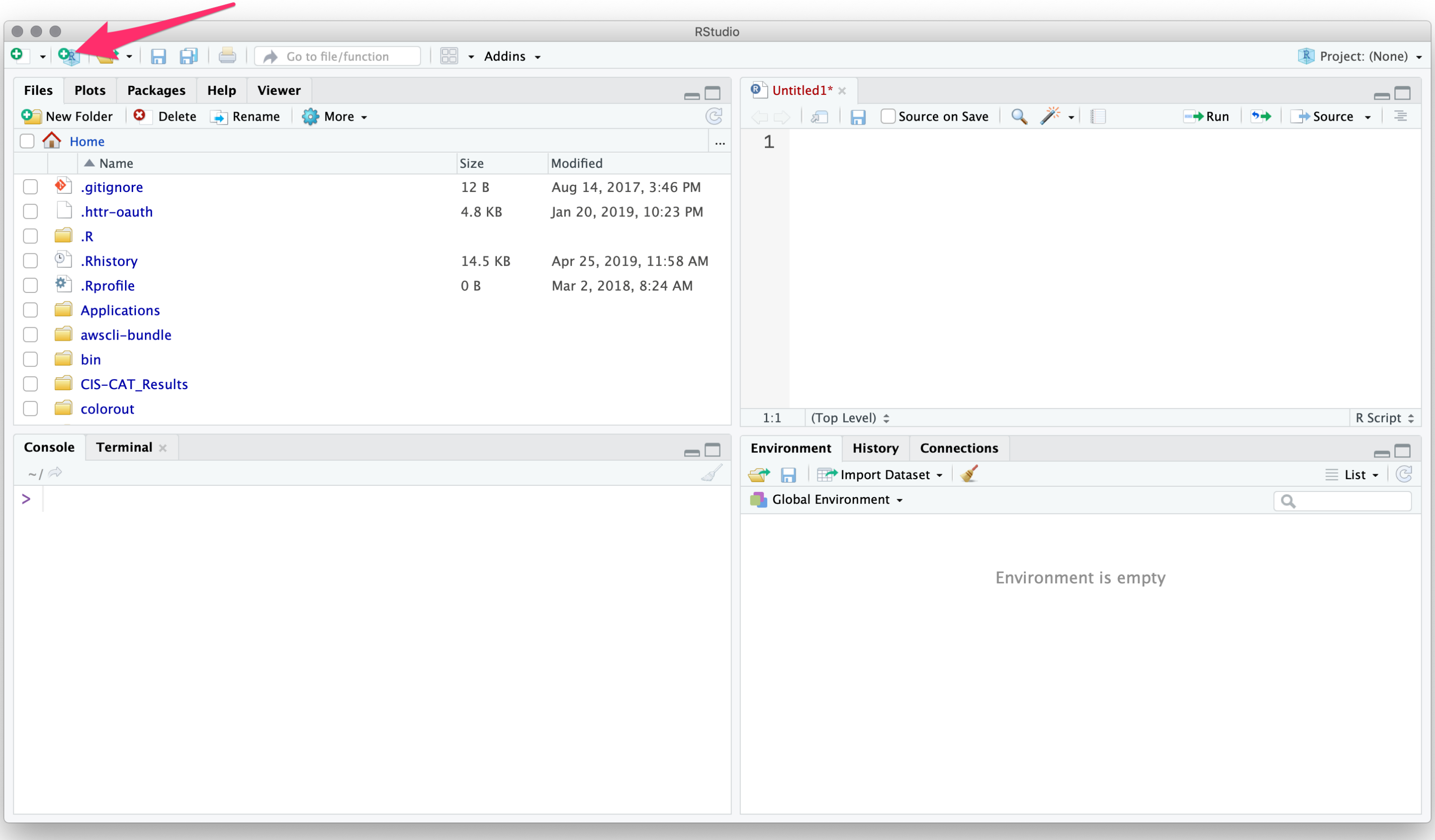
Projects

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.

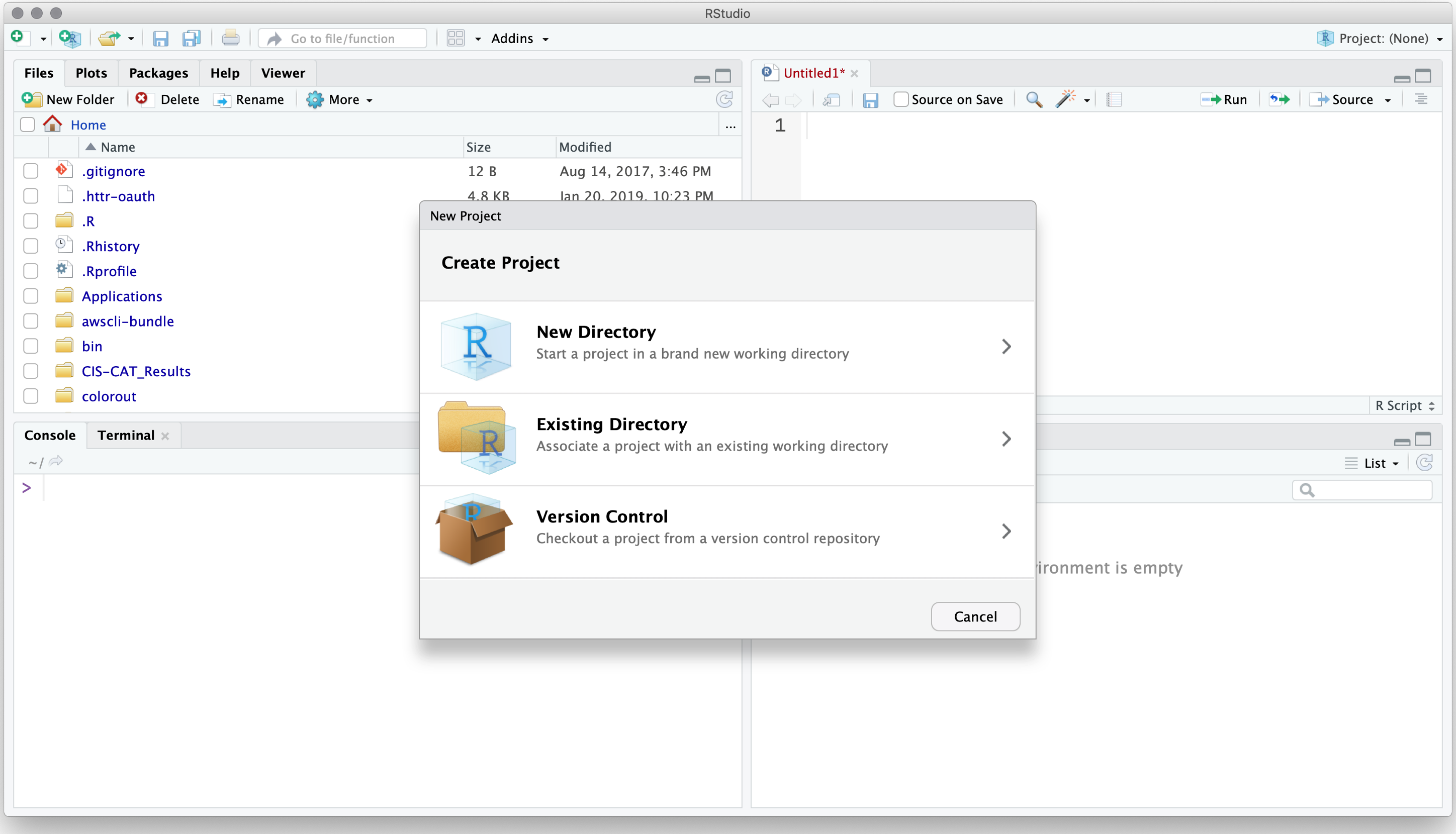
To start a new project, hit the **project icon**.



Name	Size	Modified
.gitignore	12 B	Aug 14, 2017, 3:46 PM
.httr-oauth	4.8 KB	Jan 20, 2019, 10:23 PM
.R		
.Rhistory	14.5 KB	Apr 25, 2019, 11:58 AM
.Rprofile	0 B	Mar 2, 2018, 8:24 AM
Applications		
awscli-bundle		
bin		
CIS-CAT_Results		
colorout		

Environment is empty

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



Name	Size	Modified
.gitignore	12 B	Aug 14, 2017, 3:46 PM
.httr-oauth	4.8 KB	Jan 20, 2019, 10:23 PM
.R		
.Rhistory		
.Rprofile		
Applications		
awscli-bundle		
bin		
CIS-CAT_Results		
colorout		

New Project

Create Project

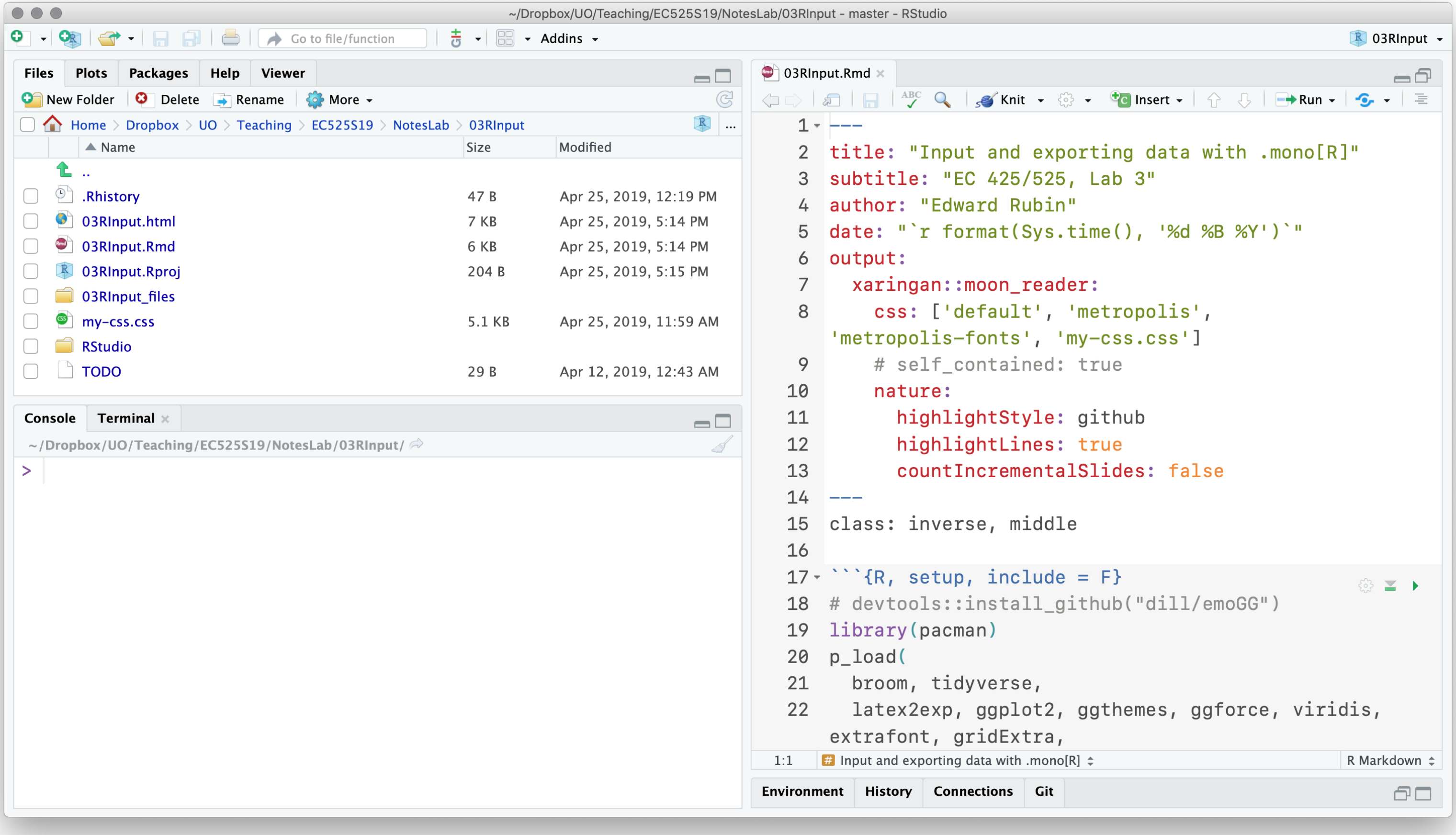
New Directory
Start a project in a brand new working directory

Existing Directory
Associate a project with an existing working directory

Version Control
Checkout a project from a version control repository

Cancel

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



Name	Size	Modified
..		
.Rhistory	47 B	Apr 25, 2019, 12:19 PM
03RInput.html	7 KB	Apr 25, 2019, 5:14 PM
03RInput.Rmd	6 KB	Apr 25, 2019, 5:14 PM
03RInput.Rproj	204 B	Apr 25, 2019, 5:15 PM
03RInput_files		
my-css.css	5.1 KB	Apr 25, 2019, 11:59 AM
RStudio		
TODO	29 B	Apr 12, 2019, 12:43 AM

```
>
```

~~~
---
title: "Input and exporting data with .mono[R]"
subtitle: "EC 425/525, Lab 3"
author: "Edward Rubin"
date: "`r format(Sys.time(), '%d %B %Y')`"
output:
  xaringan::moon_reader:
    css: ['default', 'metropolis', 'metropolis-fonts', 'my-css.css']
    # self_contained: true
    nature:
      highlightStyle: github
      highlightLines: true
      countIncrementalSlides: false
---
class: inverse, middle

```{R, setup, include = F}
# devtools::install_github("dill/emoGG")
library(pacman)
p_load(
  broom, tidyverse,
  latex2exp, ggplot2, ggthemes, ggforce, viridis, extrafont, gridExtra,
~~~

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/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.

Pipes and dplyr

Pipes and dplyr

Introduction

1. Pipes (`%>%`) make your life easier.[†]

2. `dplyr` is your data-work friend.

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

Pipes and dplyr

Pipes

We can't go much deeper into the land of `dplyr` without mentioning pipes.

A *pipe* in programming allows you to take the output of one function and plug it into another function as an argument/input.

In `dplyr`, the expression for a pipe is `%>%`.

R's pipe specifically plugs the returned object to the left of the pipe into the first argument of the function on the right fo the pipe, *e.g.*,

```
rnorm(10) %>% mean()
```

```
#> [1] 0.168937
```

Pipes and dplyr

Pipes

Pipes help avoid lots of nested functions, prevent excessive writing to your disc, and increase the readability of our **R** scripts.

Example Three ways to draw 100 N(0,1) observations and calculate the interquartile range (IQR: difference between the 75th and 25th percentiles).

```r
# Save each intermediate step
draw ← rnorm(100)
end_points ← quantile(draw, probs = c(0.25, 0.75))
diff(end_points)
# Lots of nesting
diff(quantile(rnorm(100), probs = c(0.25, 0.75)))
# Piping 💪
rnorm(100) %>% quantile(probs = c(0.25, 0.75)) %>% diff()
```

Pipes and dplyr

Pipes

By default, **R** pipes the output from the LHS of the pipe into the **first** argument of the function on the RHS of the pipe.

E.g., `a %>% fun(3)` is equivalent to `fun(arg1 = a, arg2 = 3)`.

If you want to pipe output into a different argument, you use a period (`.`).

- `b %>% fun(arg1 = 3, .)` is equivalent to `fun(arg1 = 3, arg2 = b)`.
- `b %>% fun(3, .)` is also equivalent to `fun(arg1 = 3, arg2 = b)`.
- `b %>% fun(., .)` is equivalent to `fun(arg1 = b, arg2 = b)`.

The `magrittr` package contains even more piping power.†

† `magrittr` = Magritte (of *this is not a pipe* fame) plus **R**.

dplyr

dplyr

Intro

It's a package. `dplyr` is not installed by default, so you'll need to install it.†

`dplyr` is part of the `tidyverse` (Hadleyverse), and it follows a grammar-based approach to programming/data work.

- `data` compose the subjects of your stories
- `dplyr` provides the *verbs* (action words) :
  `filter()`, `mutate()`, `select()`, `group_by()`, `summarize()`, `arrange()`

Bonus `dplyr` is pretty fast and able to interact with SQL databases.

† or just `p_load(dplyr)` after loading `pacman`.

dplyr

Manipulating variables: `mutate()`

`dplyr` streamlines adding/manipulating variables in your data frame.

Function `mutate(.data, ...)`

- Required argument `.data`, an existing data frame
- Additional arguments Names and values of the new variables
- Output An updated data frame

Example

```
mutate(.data = our_df, new1 = 7, new2 = x * y)
```

dplyr

`mutate()`

Example Take the data frame

```
my_df ← data.frame(x = 1:3, y = 5:7)
```

`mutate()` allows us to create many new variables with one call.

```
mutate(.data = my_df,
  xy = x * y,
  x2 = x^2,
  xy2 = xy^2,
  is_max = x == max(x)
)
```

x	**y**	**xy**	**x2**	**xy2**	**is_max**
1	5	5	1	25	false
2	6	12	4	144	false
3	7	21	9	441	true

Notice `mutate()` returns the original *and* new columns.

dplyr

`mutate()` *vs.* `transmute()`

As their names imply, `mutate()` and `transmute()` are very similar functions.

- `mutate()` returns the original *and* new columns (variables).
- `transmute()` returns only the new columns (variables).

Note Both functions return a new object as *output*—they do not update the object in **R**'s memory. (This is the case for all functions in `dplyr`.)

dplyr

`%>%` and `dplyr`

Each `dplyr` function begins with a `.data` argument so that you can easily pipe in data frames (recall: `mutate(.data, ...)`).

The common workflow in `dplyr` will look something like

```
new_df ← old_df %>% mutate(cool stuff here)
```

which takes `old_df`, does some cool stuff with `mutate()`, and then saves the output of `mutate()` as `new_df`.

dplyr

`filter()`

The `filter()` function does what its name implies: it **filters the rows** of your data frame **based upon logical conditions**.

Example

```r
# Create a dataset
some_df <- data.frame(
  x = 1:10,
  y = 11:20
)
```

```r
# Only keep rows where x is 3
some_df %>% filter(x == 3)
```

x	y
3	13

dplyr

`filter()`

The `filter()` function does what its name implies: it **filters the rows** of your data frame **based upon logical conditions**.

Example

```r
# Create a dataset
some_df <- data.frame(
  x = 1:10,
  y = 11:20
)
```

```r
# Only keep rows where x > 7
some_df %>% filter(x > 7)
```

x	y
8	18
9	19
10	20

dplyr

`filter()`

The `filter()` function does what its name implies: it **filters the rows** of your data frame **based upon logical conditions**.

Example

```r
# Create a dataset
some_df ← data.frame(
  x = 1:10,
  y = 11:20
)
```

```r
# Keep rows where y/x > 3
some_df %>% filter(y/x > 3)
```

x	y
1	11
2	12
3	13
4	14

dplyr

`filter()`

The `filter()` function does what its name implies: it **filters the rows** of your data frame **based upon logical conditions**.

Example

```r
# Create a dataset
some_df <- data.frame(
  x = 1:10,
  y = 11:20
)
```

```r
# Keep rows where x>8 OR y<12
some_df %>%
  filter(x > 8 | y < 12)
```

x	y
1	11
9	19
10	20

dplyr

`filter()`

The `filter()` function does what its name implies: it **filters the rows** of your data frame **based upon logical conditions**.

Example

```r
# Create a dataset
some_df ← data.frame(
  x = 1:10,
  y = 11:20
)
```

```r
# Keep rows where 16 ⩽ y ⩽ 18
some_df %>%
  filter(between(y, 16, 18))
```

x	y
6	16
7	17
8	18

dplyr

`filter()`

The `filter()` function does what its name implies: it **filters the rows** of your data frame **based upon logical conditions**.

Example

```r
# Create a dataset
some_df ← data.frame(
  x = 1:10,
  y = 11:20
)
```

```r
# Keep rows where y > 20
some_df %>% filter(y > 20)
```

x	y
No data available in table	

If you filter your data frame down to nothing, **R** returns a 0-row data frame with the names/number of columns from the original data frame.

dplyr

`select()`

Just as `filter()` grabs row-based subsets of your data frame,
`select()` grabs column-based subsets.

You can select columns using their **names**

```
our_df %>% select(var10, var100)
```

you can select columns using their **numbers**

```
our_df %>% select(10, 100)
```

or you can select columns using **helper fuctions**

```
our_df %>% select(starts_with("var10"))
```

`select()` helps you narrow down a dataset to its necessary features.

dplyr

`summarize()`

Hopefully you're starting to see that functions' names in `dplyr` tell you what the function does.

`summarize()`[†] summarizes variables—you choose the variables and the summaries (*e.g.*, `mean()` or `min()`).

```
the_df %>% summarize(
  mean(x), mean(y), mean(z),
  min(x), max(x),
)
```

would return a 1×5 data frame with the means of `x`, `y`, and `z`; the minimum of `x`; and the maximum of `x`.

[†] or `summarise()` if you ❤️ 🇬🇧

dplyr

`summarize()` and `group_by()`

While sample-wide summarizes are certainly interesting, `dplyr` has one last gem for us: `group_by()`.

`group_by()` groups your observations by the variable(s) that you name.

Specifically, `group_by()` returns a *grouped data frame* that you can then feed to `summarize()`, `mutate()`, or `transmuate` to perform grouped calculations, *e.g.*, each group's mean.

dplyr

Example: Grouped summaries

```r
# Create a new data frame
our_df ← data.frame(
  x = 1:6,
  y = c(0, 1),
  grp = rep(c("A", "B"), each = 3)
)
```

```r
# For dataset 'our_df'...
our_df %>%
  # Group by 'grp'
  group_by(grp) %>%
  # Take means of 'x' and 'y'
  summarize(mean(x), mean(y))
```

x	y	grp
1	0	A
2	1	A
3	0	A
4	1	B
5	0	B
6	1	B

grp	mean(x)	mean(y)
A	2.000	0.333
B	5.000	0.667

Example: Grouped mutation

```r
# Create a new data frame
our_df ← data.frame(
  x = 1:6,
  y = c(0, 1),
  grp = rep(c("A", "B"), each = 3)
)
```

```r
# Add grp means for x and y
our_df %>%
  group_by(grp) %>%
  mutate(
    x_m = mean(x), y_m = mean(y)
  )
```

x	y	grp
1	0	A
2	1	A
3	0	A
4	1	B
5	0	B
6	1	B

x	y	grp	x_m	y_m
1	0	A	2.000	0.333
2	1	A	2.000	0.333
3	0	A	2.000	0.333
4	1	B	5.000	0.667
5	0	B	5.000	0.667
6	1	B	5.000	0.667

dplyr

`arrange()`

`arrange()` will sorts the rows of a data frame using the inputted columns.

`R` defaults to starting with the "lowest" (smallest) at the top of the data frame. Use a `-` in front of the variable's name to reverse sort.

```r
# As is
our_df
```

x	y	grp
1	0	A
2	1	A
3	0	A
4	1	B
5	0	B
6	1	B

```r
# Arrang by y, grp, then -x
our_df %>% arrange(y, grp, -x)
```

x	y	grp
3	0	A
1	0	A
5	0	B
2	1	A
6	1	B
4	1	B

The tidyverse

There's more! `dplyr` and `tidyr` offer even more...[†]

- *Viewing data* `glimpse()`, `top_n()`
- *Sampling* `sample_n()`, `sample_frac()`
- *Summaries* `first()`, `last()`, `nth()`, `n_distinct()`
- *Duplicates* `distinct()`
- *Missingness* `na_if()`, `replace_na()`, `drop_na()`, `fill()`

The folks at RStudio have put together some great cheatsheets, *e.g.*,

- `dplyr`
- data import
- data wrangling

[†] And these are only two of the packages in the `tidyverse`.

Table of contents

Admin

Workflow

`dplyr`