

# 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

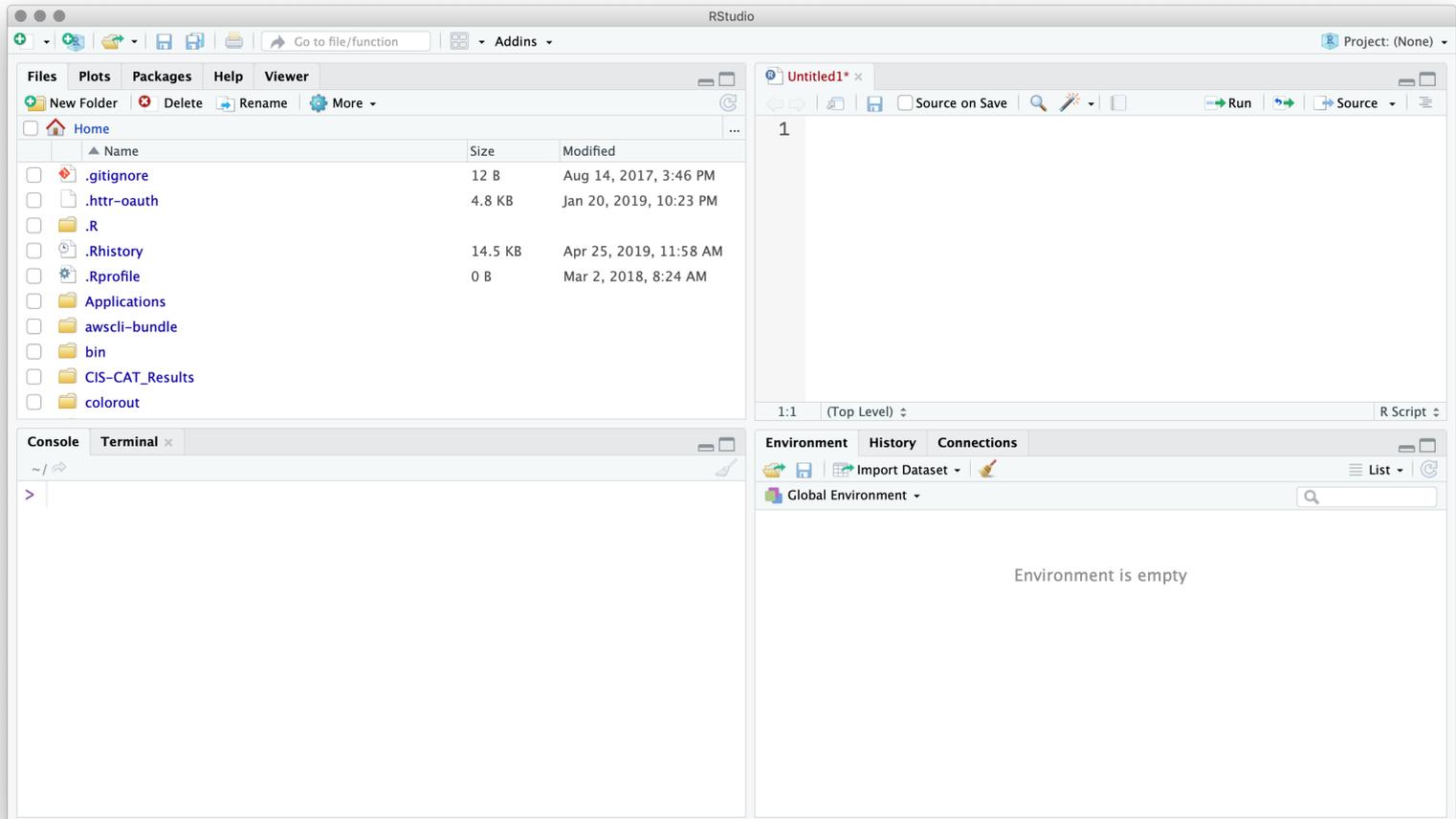
Key points from the last lab(s).

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

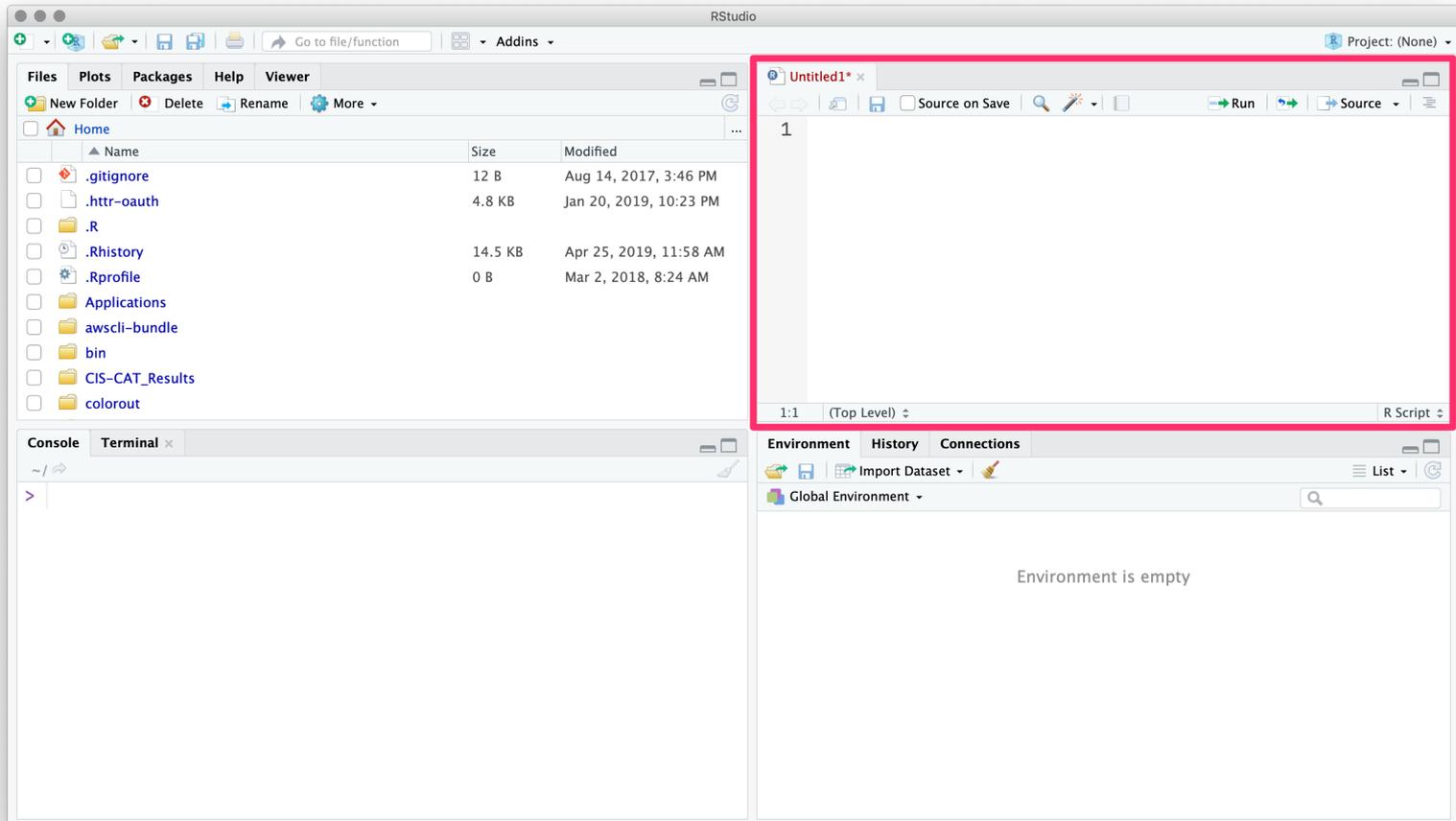
<sup>†</sup> Check out `magrittr` for more pipe options, e.g., `%<>%`.

# RStudio

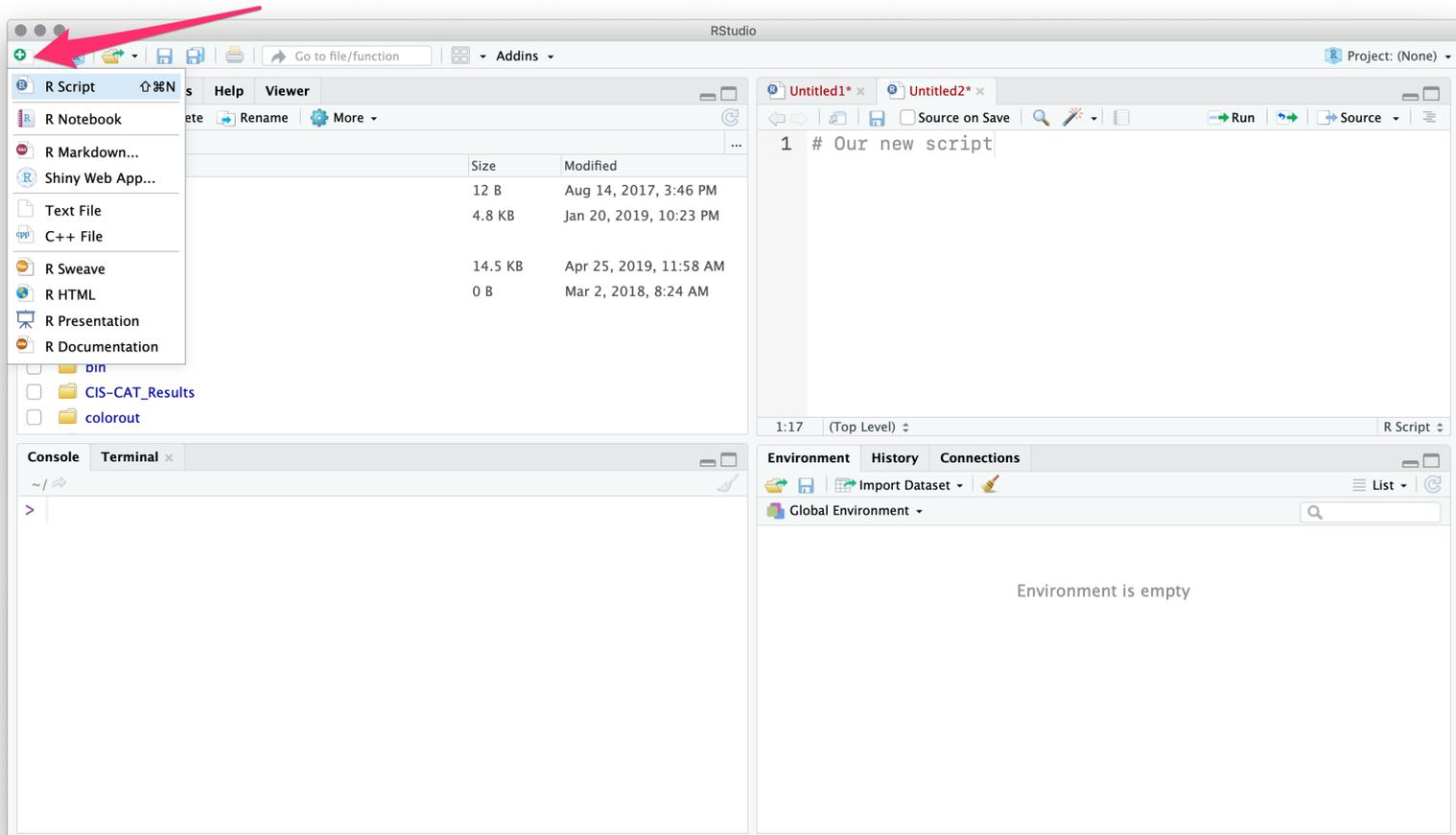
Let's recap some of the major features in RStudio...



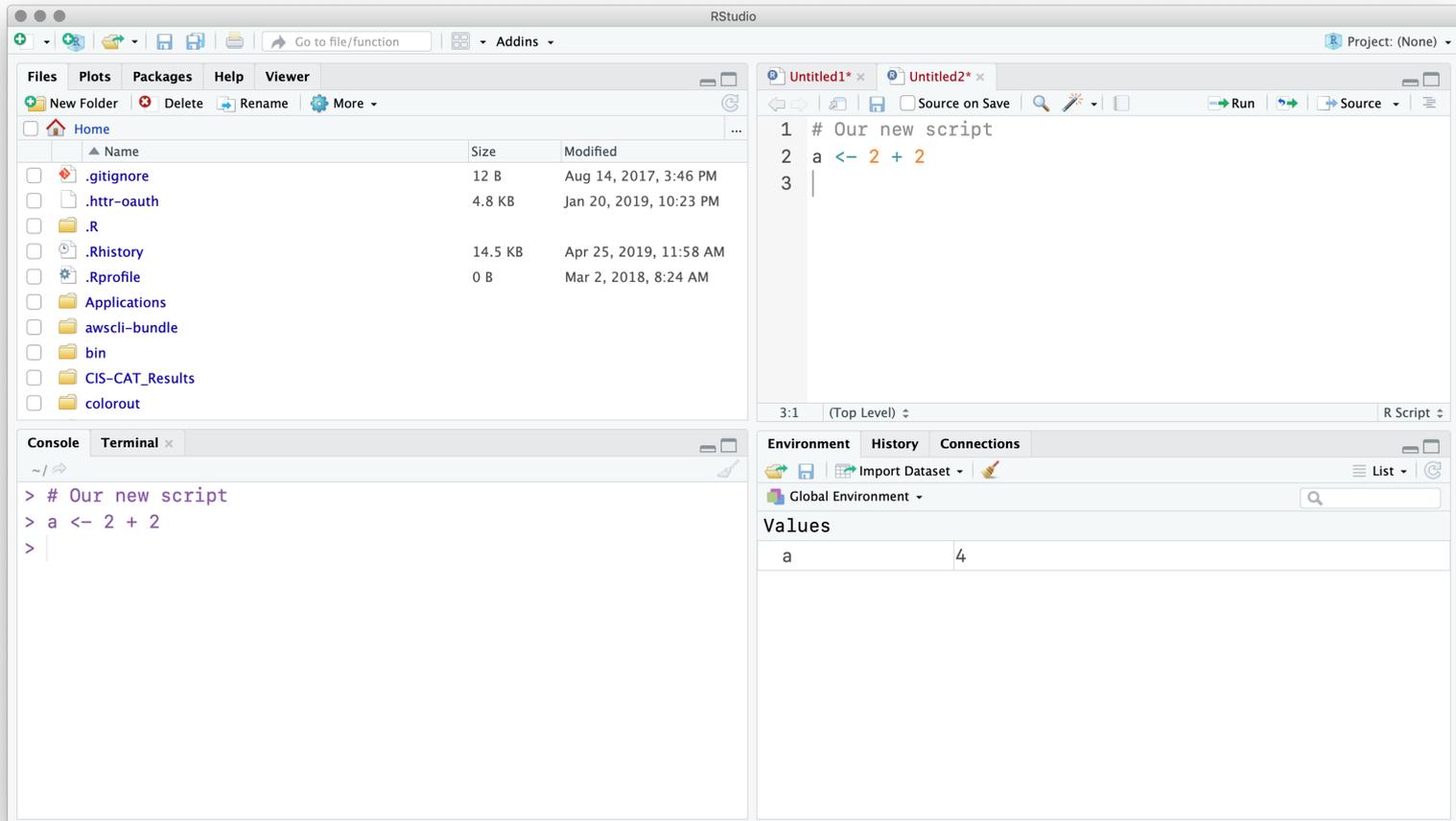
First, you write your R scripts (source code) in the **Source** pane.



You can use the menubar or  $\text{⌘} + \text{⌘} + \text{N}$  to create new R scripts.



To execute commands from your R script, use  $\text{⌘} + \text{Enter}$ .



The screenshot displays the RStudio environment with the following components:

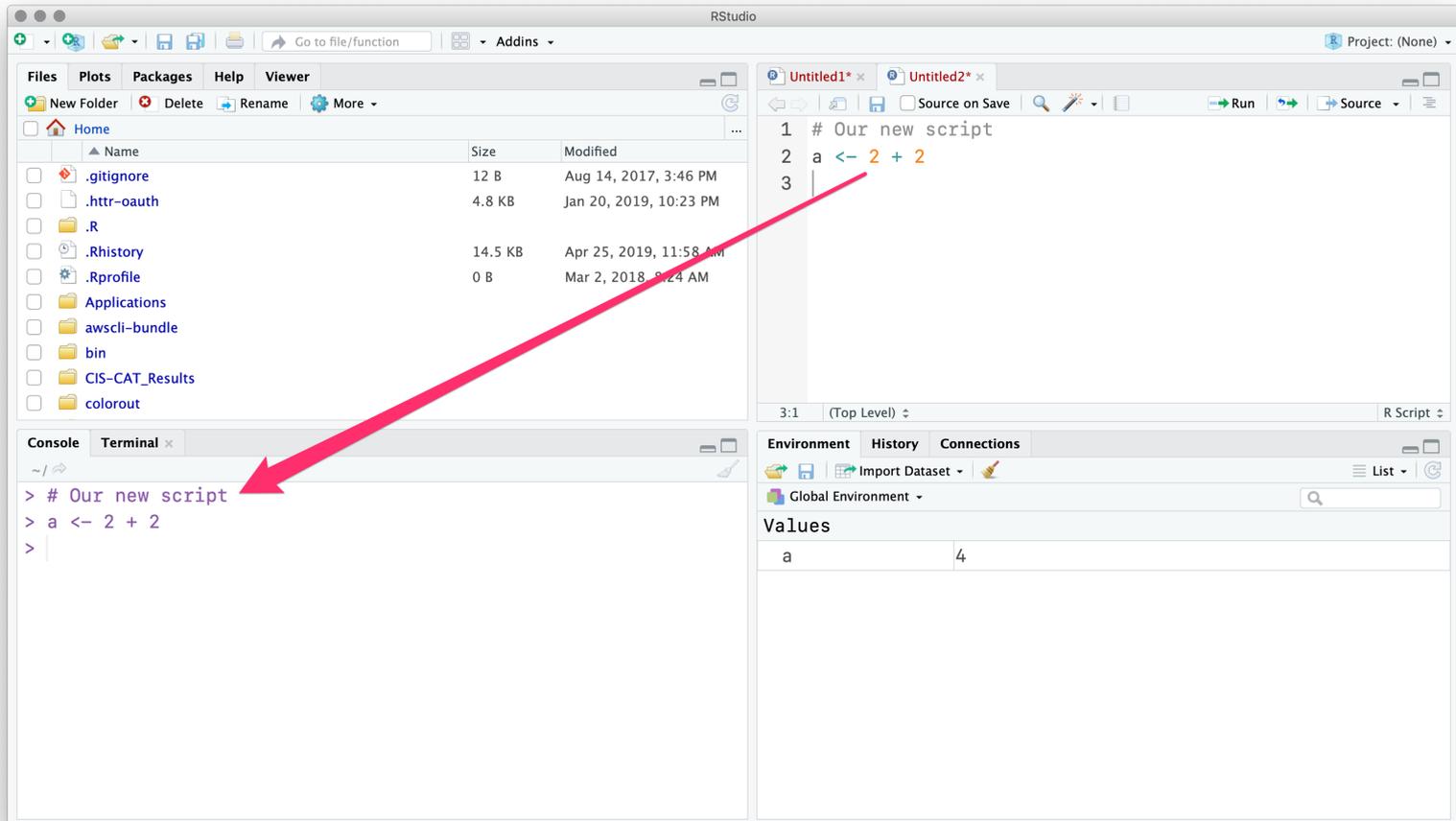
- Files Panel:** Shows a file explorer view of the home directory with files like `.gitignore`, `.httr-oauth`, `.R`, `.Rhistory`, `.Rprofile`, `Applications`, `awscli-bundle`, `bin`, `CIS-CAT_Results`, and `colorout`.
- Source Editor:** Contains a script with the following code:

```
1 # Our new script
2 a <- 2 + 2
3 |
```
- Console:** Shows the execution of the script:

```
> # Our new script
> a <- 2 + 2
> |
```
- Environment Panel:** Shows the `Global Environment` with a table of values:

Values	
a	4

RStudio will execute the command in the terminal.



The screenshot shows the RStudio interface with a script editor on the right and a terminal on the bottom left. A red arrow points from the script editor to the terminal, indicating the execution of the code.

**Script Editor (Untitled1\* x):**

```
1 # Our new script
2 a <- 2 + 2
3
```

**Terminal:**

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

**Environment Panel:**

Values	
a	4

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

The screenshot shows the RStudio interface. The top-left pane displays the file explorer with a list of files and folders. The top-right pane shows the source editor with the following R code:

```
1 # Our new script
2 a <- 2 + 2
3 |
```

The bottom-left pane shows the console with the following output:

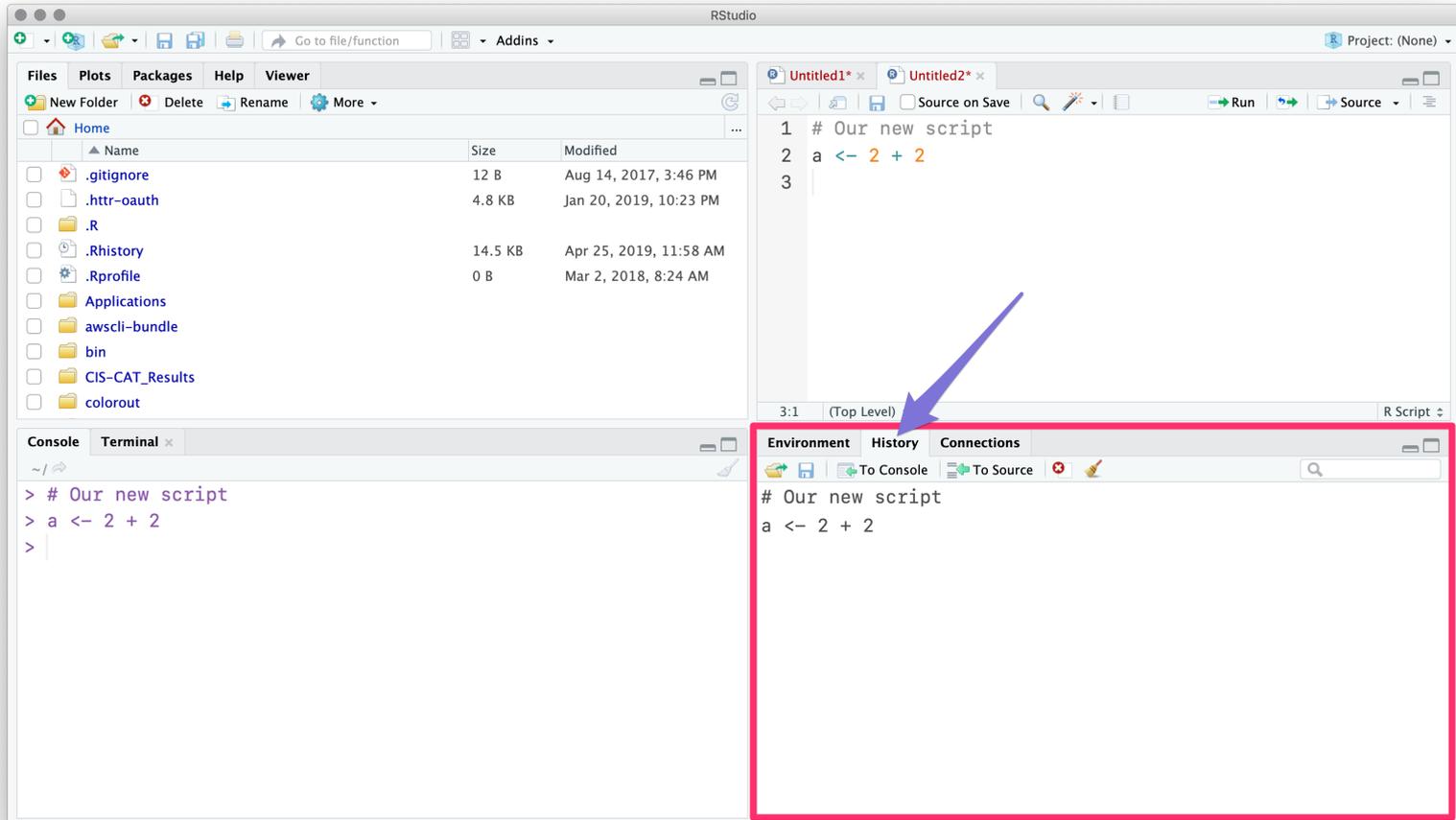
```
> # Our new script
> a <- 2 + 2
> |
```

The bottom-right pane, titled "Environment", shows the current environment. It displays the "Global Environment" and a list of "Values":

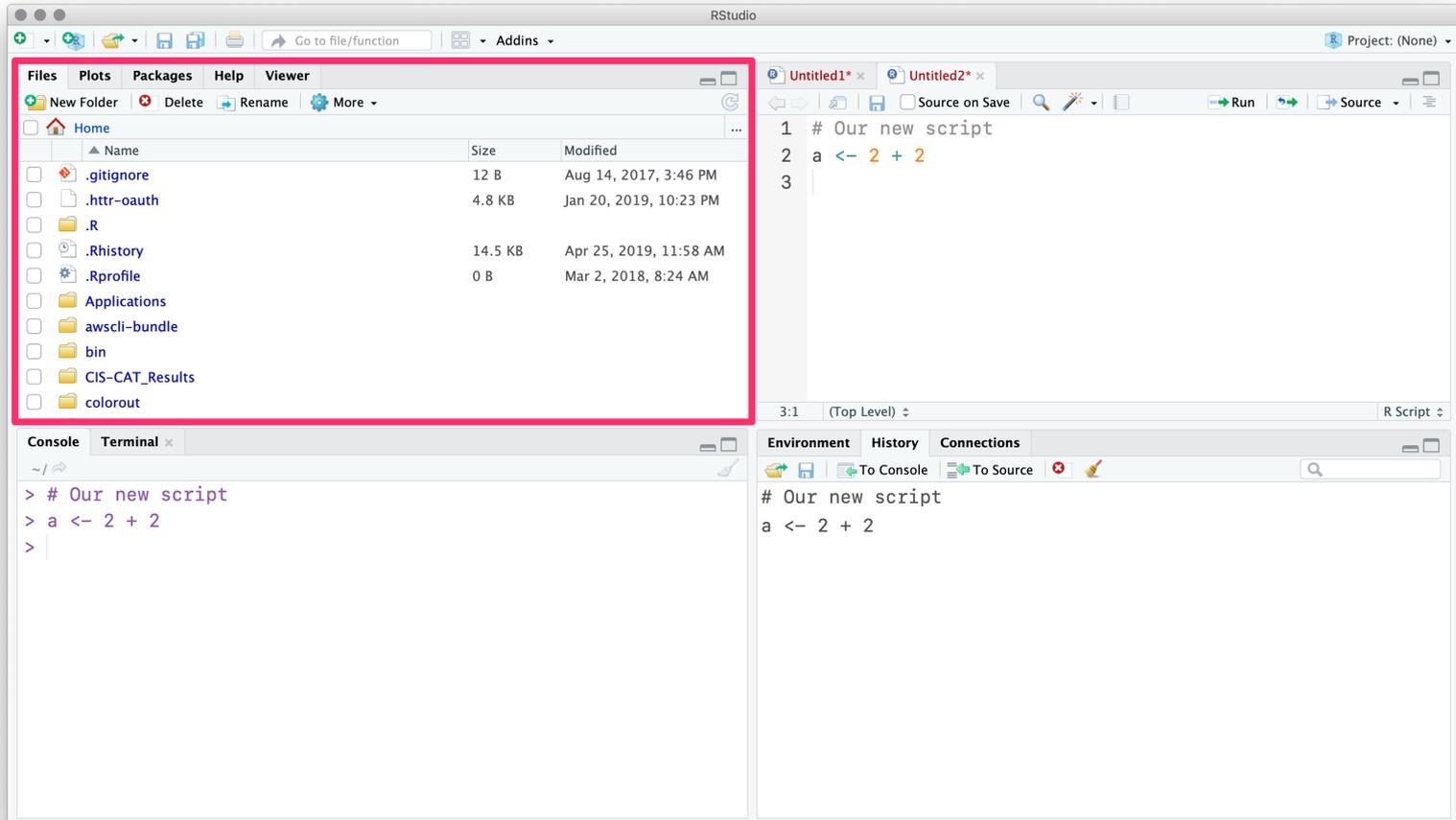
Object Name	Value
a	4

The Environment pane is highlighted with a red border.

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



The **Files** pane is file explorer.



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

The screenshot displays the RStudio interface with three main panes:

- Plots Pane (top left, highlighted with a red border):** Shows a scatter plot of  $z^2$  versus  $z$ . The x-axis is labeled 'z' and ranges from 0 to 10.0. The y-axis is labeled ' $z^2$ ' and ranges from 0 to 100. The plot contains 10 data points forming a parabolic curve.
- Source Pane (top right):** Contains the following R code:

```
1 # Our new script
2 a <- 2 + 2
3
4 # A quick plot
5 z <- 1:10
6 ggplot2::qplot(x = z, y = z^2)
7
```
- Console Pane (bottom left):** Shows the execution of the code:

```
> # A quick plot
> z <- 1:10
> ggplot2::qplot(x = z, y = z^2)
>
```
- Environment Pane (bottom right):** Shows the current environment with the following objects:

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

# Packages shows installed packages

The screenshot shows the RStudio interface with the 'Packages' pane highlighted by a red box. The 'Packages' pane displays a list of installed packages with columns for Name, Description, and Version. The packages listed are:

Name	Description	Version
<input type="checkbox"/> numDeriv	Accurate Numerical Derivatives	2016.8-1
<input type="checkbox"/> openssl	Toolkit for Encryption, Signatures and Certificates Based on OpenSSL	1.1
<input type="checkbox"/> openxlsx	Read, Write and Edit XLSX Files	4.1.0
<input type="checkbox"/> packrat	A Dependency Management System for Projects and their R Package Dependencies	0.5.0
<input 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"/> parsnip	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 main editor window shows an R script with the following code:

```
1 # Our new script
2 a <- 2 + 2
3
4 # A quick plot
5 z <- 1:10
6 ggplot2::qplot(x = z, y = z^2)
7
```

The console window shows the output of the script:

```
# 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**.

The screenshot displays the RStudio interface. The 'Packages' pane on the left lists installed packages, with 'pacman' checked. The 'Console' pane shows the command `library(pacman)` being executed. The 'Source' pane shows an R script with the following code:

```
1 # Our new script
2 a <- 2 + 2
3
4 # A quick plot
5 z <- 1:10
6 ggplot2::qplot(x = z, y = z^2)
7
```

The 'Environment' pane on the right shows the output of the script, including the `library(pacman)` command.

Name	Description	Version
<input type="checkbox"/> numDeriv	Accurate Numerical Derivatives	2016.8-1
<input type="checkbox"/> openssl	Toolkit for Encryption, Signatures and Certificates Based on OpenSSL	1.1
<input type="checkbox"/> openxlsx	Read, Write and Edit XLSX Files	4.1.0
<input type="checkbox"/> 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"/> rpart	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 **Help** tab shows help documentation (also accessible via `?`).

The screenshot displays the RStudio interface with the Help tab selected. The Help tab shows the documentation for the `p_load` function from the `pacman` package. The documentation includes a description, usage, and a console window showing the command `?p_load` being executed. The source editor on the right shows the R script being edited, which includes a comment, a simple calculation, and a plot command.

**Files** **Plots** **Packages** **Help** **Viewer**

R: Load One or More Packages

`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)
```

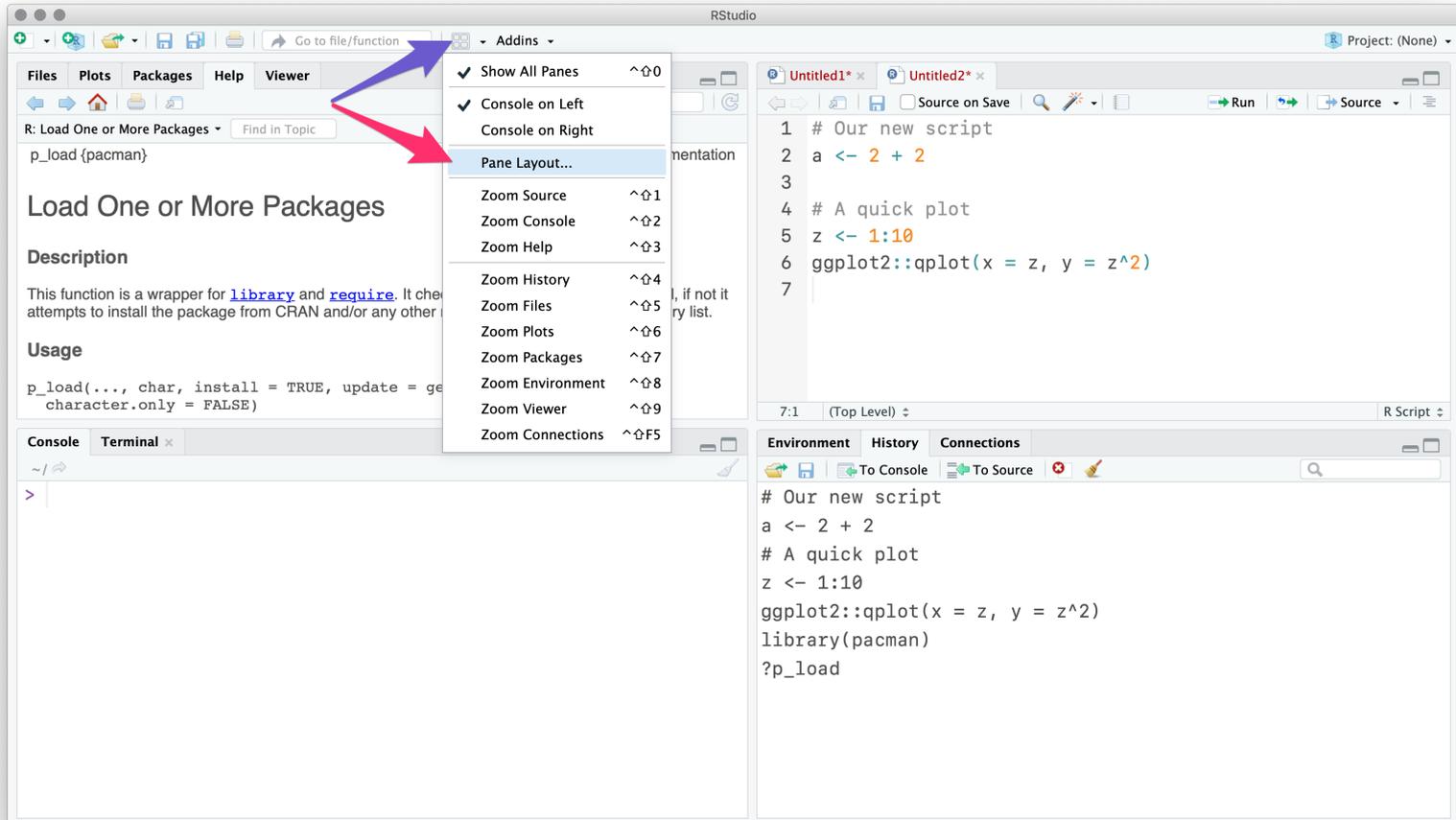
**Console** **Terminal**

```
> ?p_load
> |
```

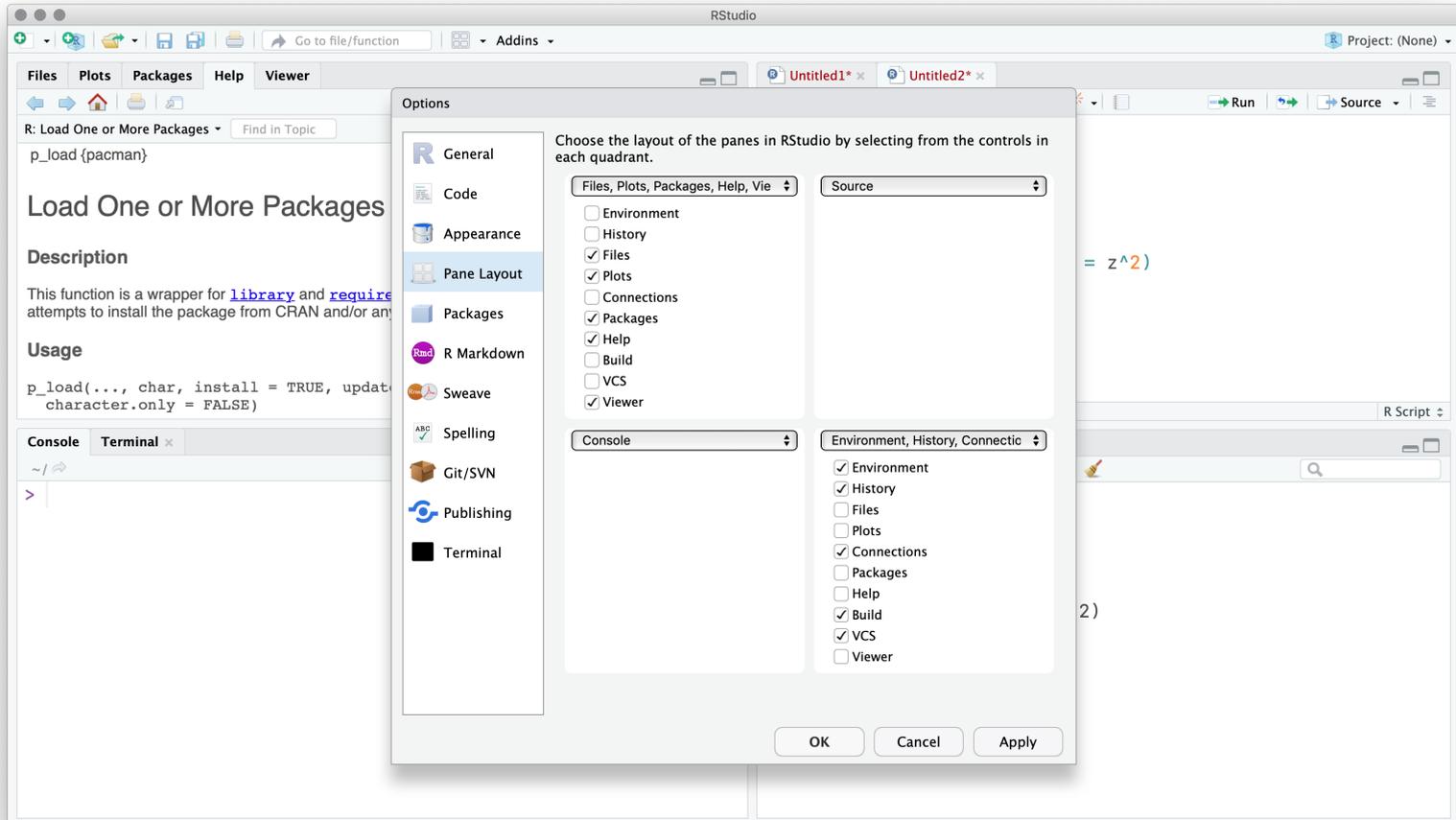
**Environment** **History** **Connections**

```
# 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



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.

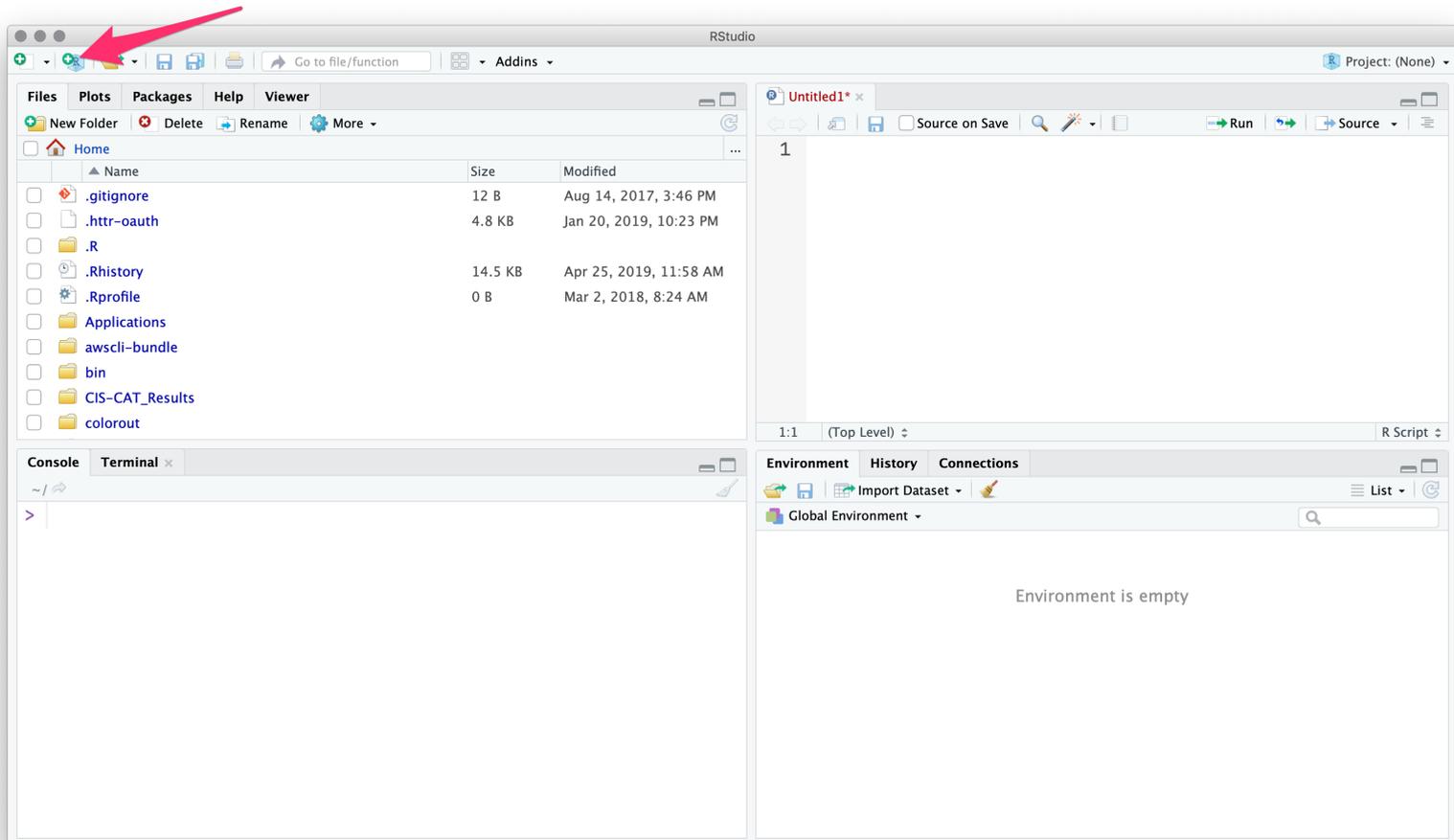
## Projects

Projects in R offer several benefits:

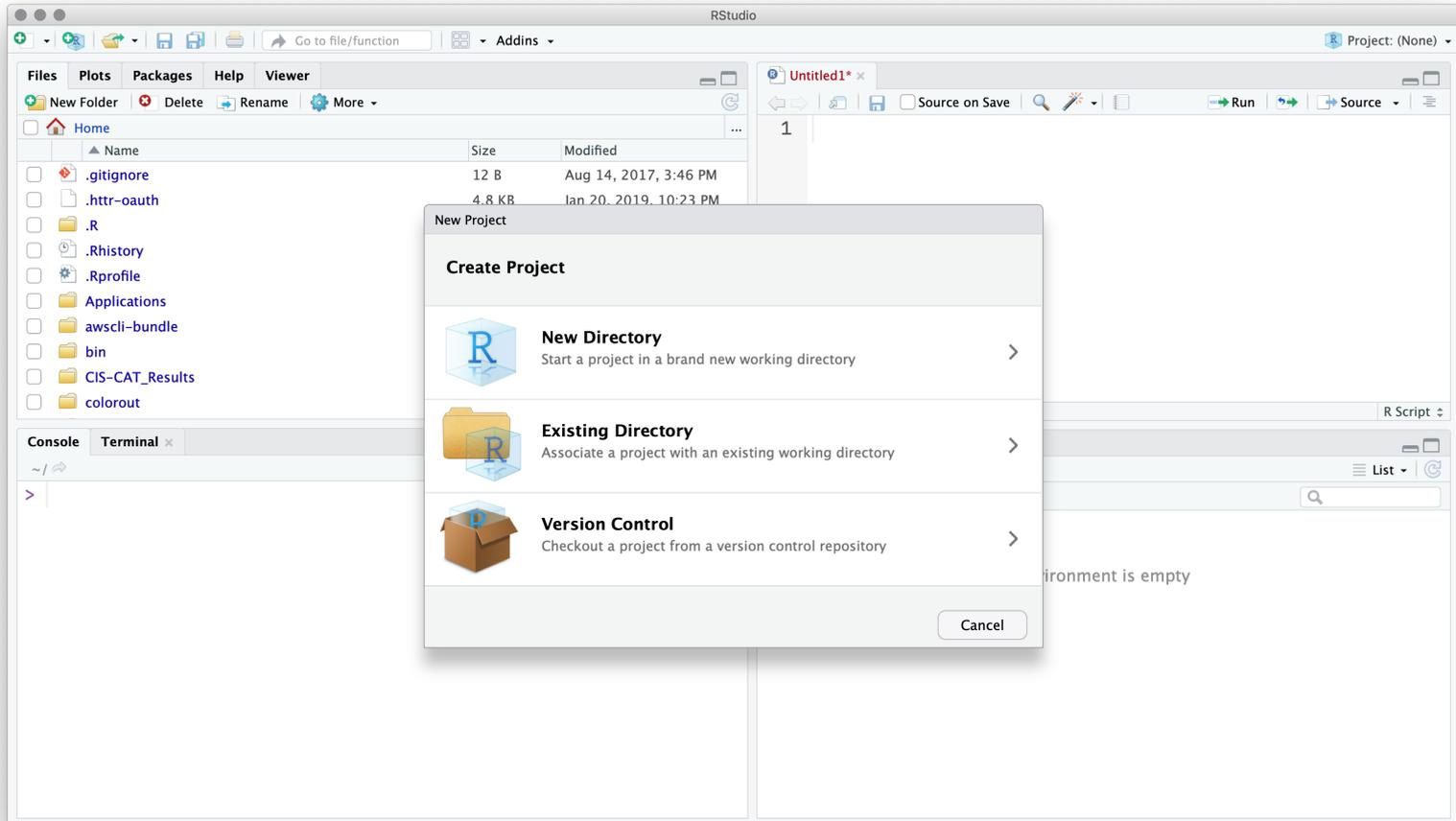
1. Act as an anchor for working with files.
2. Make your work (projects) easily reproducible.<sup>†</sup>
3. Help you quickly jump back into your work.

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

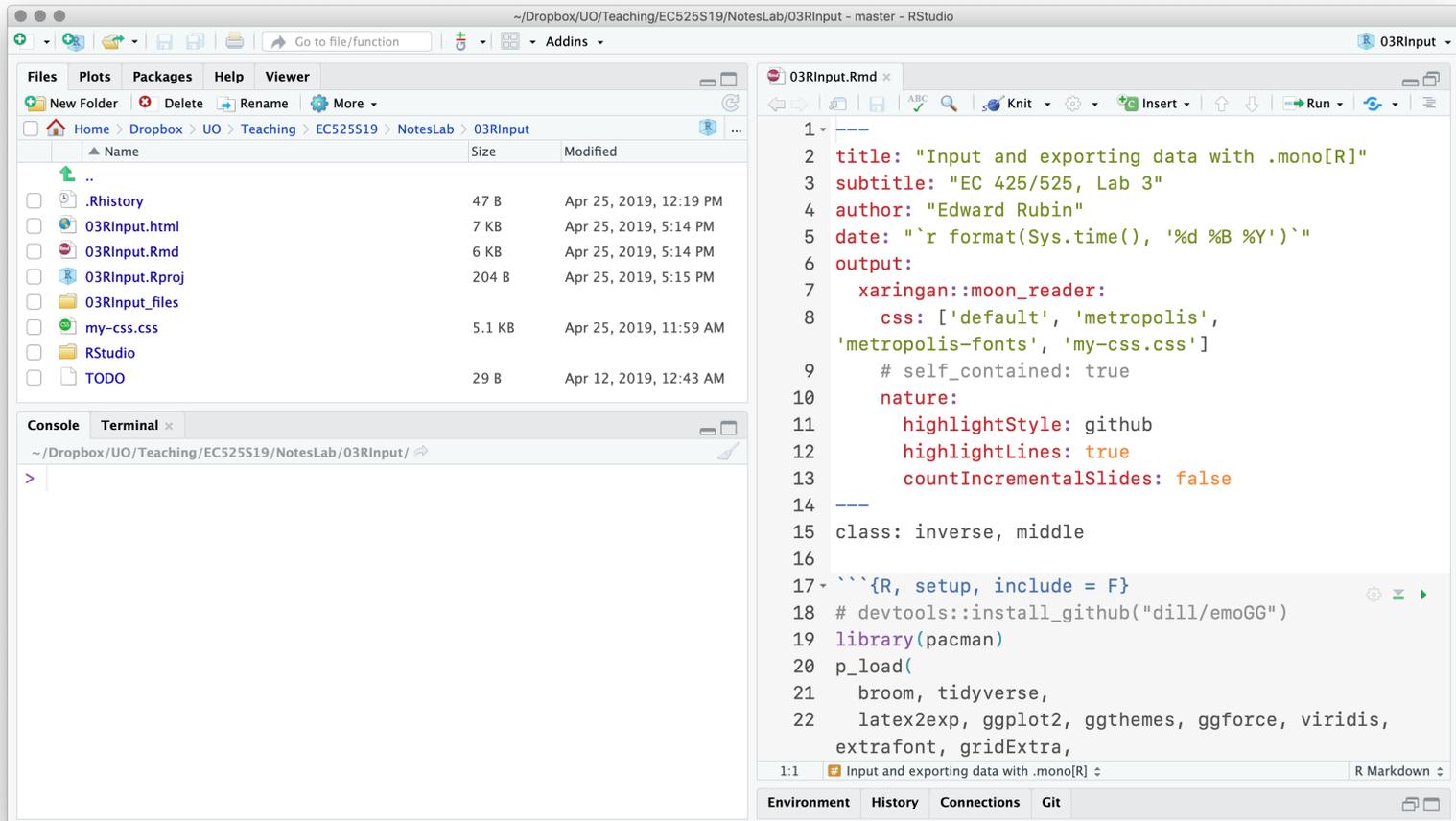
To start a new project, hit the **project icon**.



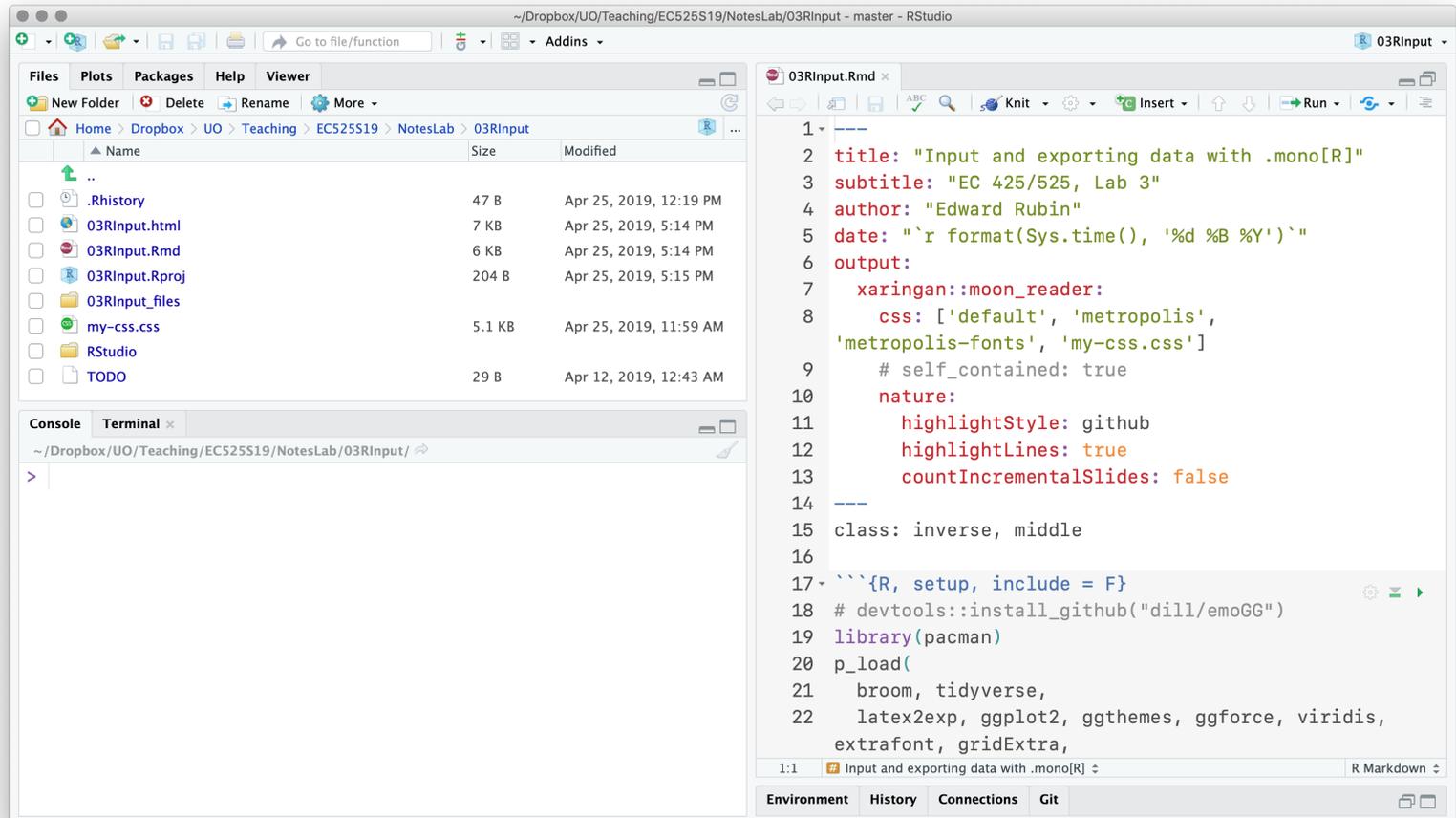
You'll then choose the folder/directory where your project lives.



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/U0/Teaching/EC525S19/"  
dir_labs ← paste0(dir_class, "NotesLab/")  
dir_lab03 ← paste0(dir_labs, "03RInput/")  
sample_df ← read.csv(paste0(dir_lab03, "sample.csv"))
```

# 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")
```

# 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()`.

# 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" "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.<sup>†</sup>

```
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.

<sup>†</sup> 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`.<sup>†</sup>

```
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
```

<sup>†</sup> 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
  - Reading files
    - `dir()`
    - `read.csv()`
    - `read_csv()`
    - Other file types
  - Writing (output)
  - RDS files
7. More resources