Project Organization

Abhijit Dasgupta

BIOF 339

Objectives

- Project Organization
  - How to maintain long-term sanity

Why organize?

Common Objectives

- Maximize
  - Time to think about a project
  - Reliability/Reproducibility
- Minimize
  - Data errors
  - Programmer/Analyst errors
  - Programming Time
  - Re-orientation time when revisiting

Our inclination

- Once we get a data set
  - Dig in!!
  - Start "playing" with tables and figures
  - Try models on-the-fly
  - Cut-and-paste into reports and presentations

DON'T DO THIS!!

BIOF 339: Practical R

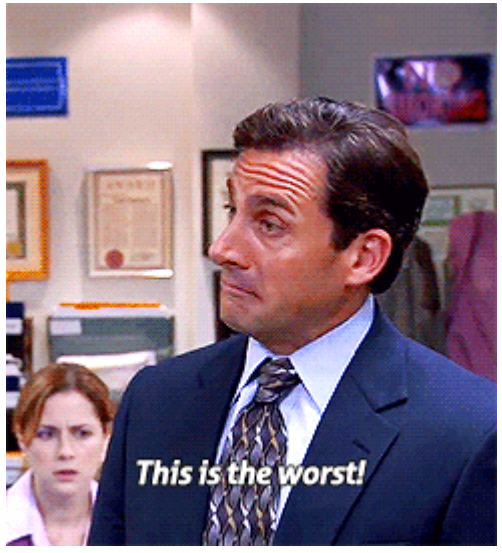
Abhijit's story

Many years ago

- 25 year study of rheumatoid arthritis
- 5600 individuals
- Several cool survival analysis models
- Needed data cleaning, validation and munging, and some custom computations
- Lots of visualizations

Many years ago

- Resulted in a muddle of 710 files (starting from 4 data files)
- Unwanted cyclic dependencies for intermediate data creation
- Lots of ad hoc decisions and function creation with scripts
- Almost impossible to re-factor and clean up
- Had to return to this project for 3 research papers and revision cycles!!!



Who's the next consumer of your work??

Yourself in

3 months

1 year

5 years

Can't send your former self e-mail asking what the frak you did.

BIOF 339: Practical R

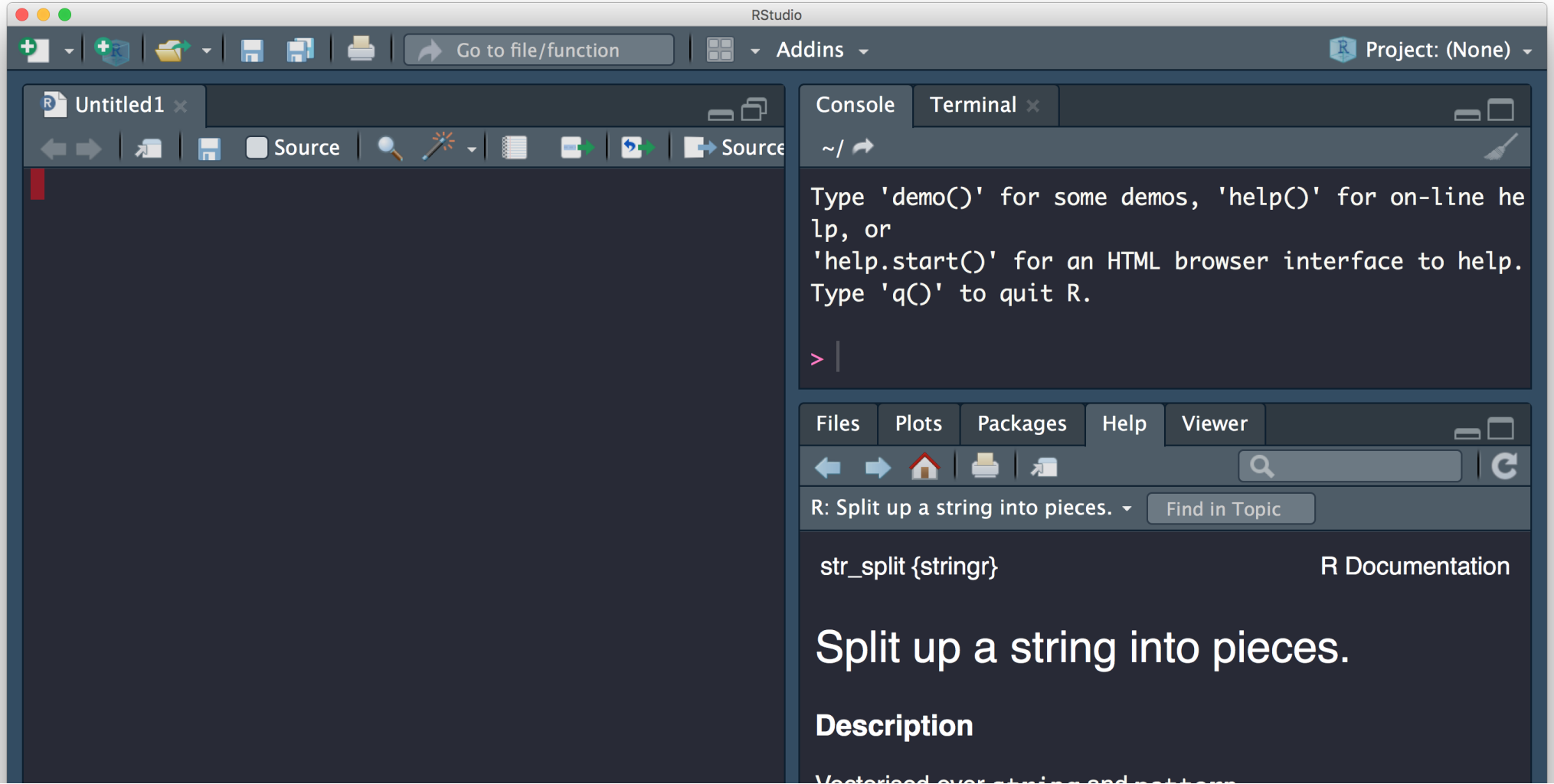
Biggest reason for good practices is

YOUR OWN SANITY

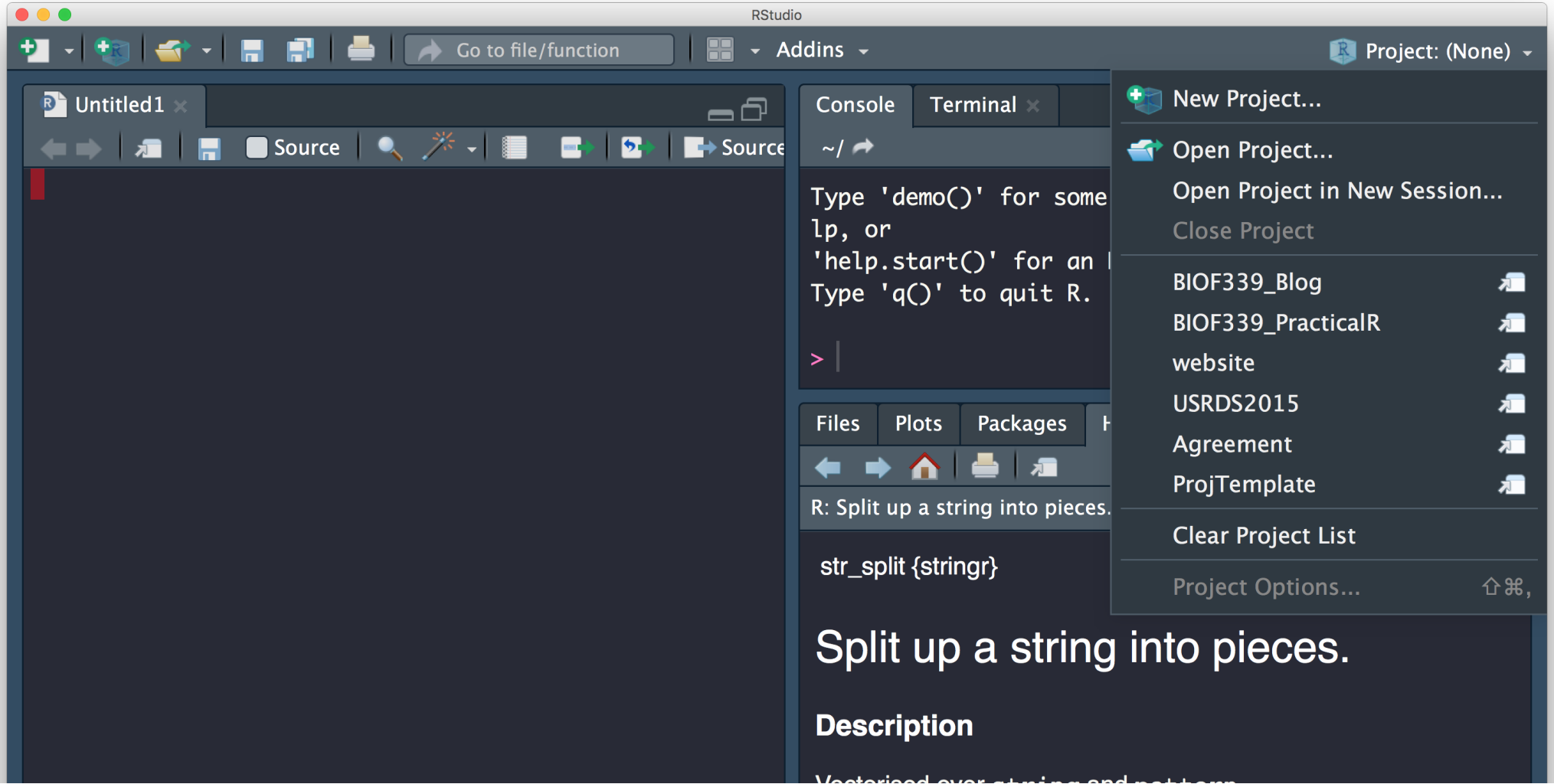
BIOF 339: Practical R

RStudio Projects

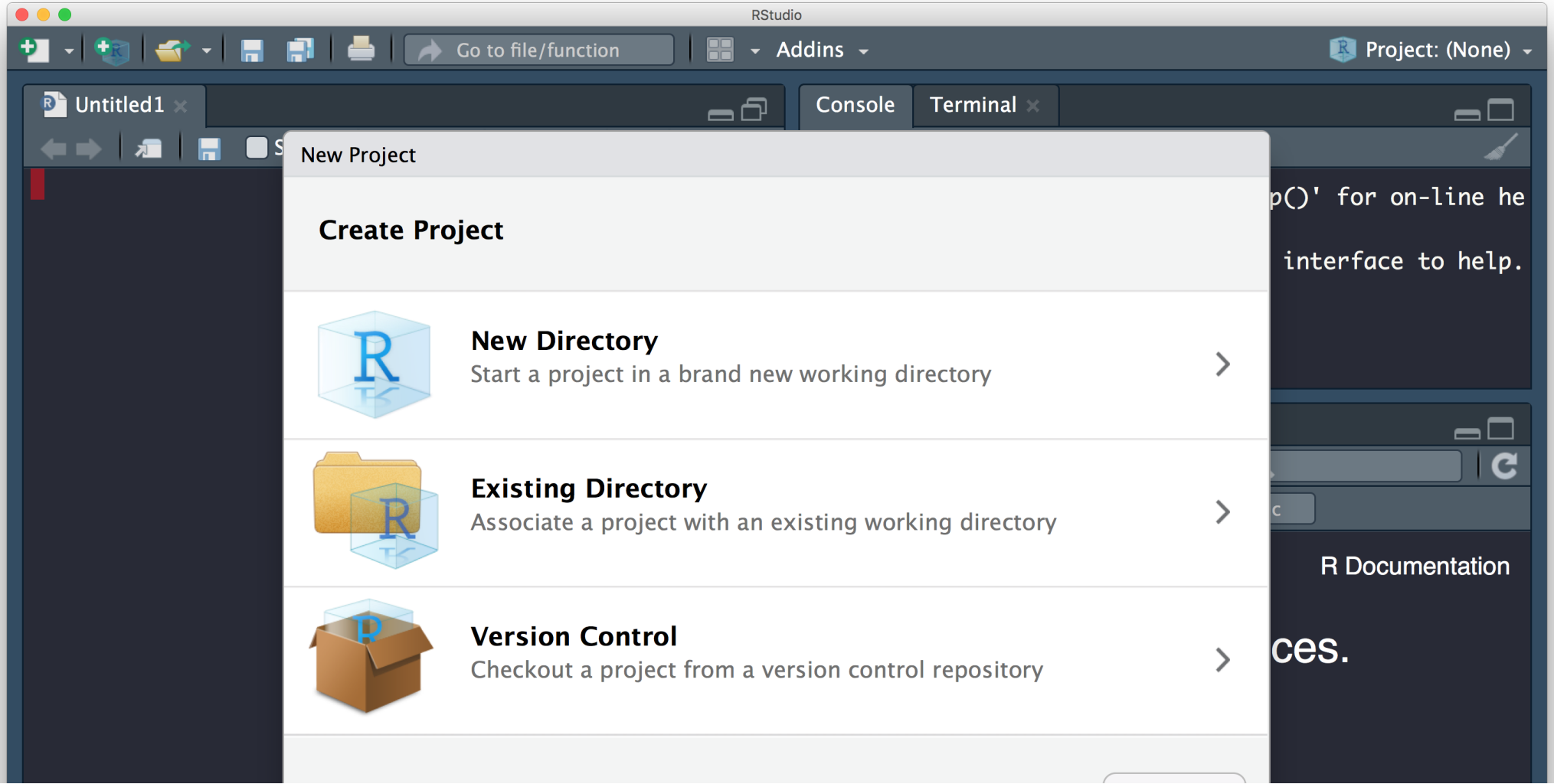
RStudio Projects



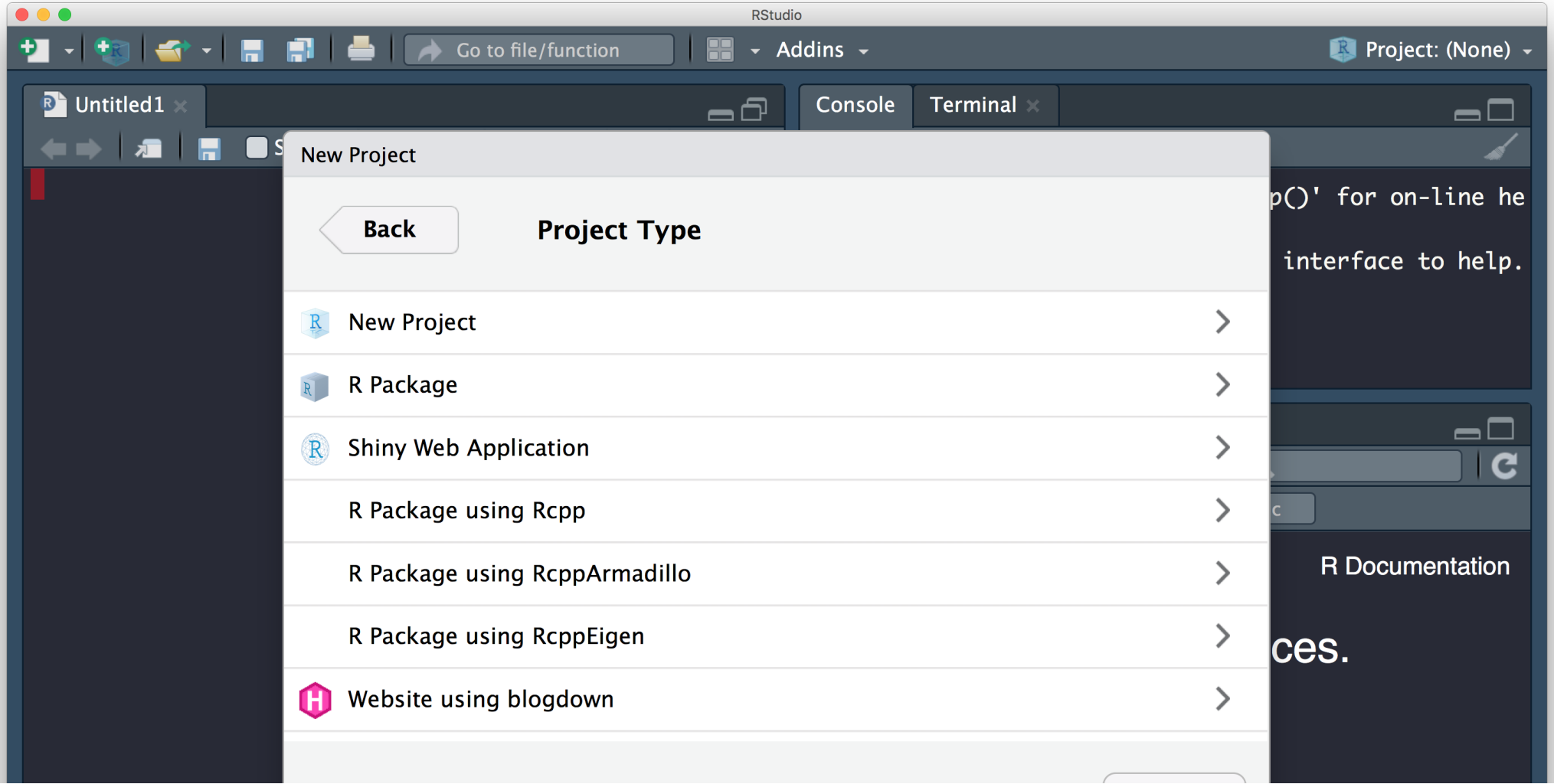
RStudio Projects



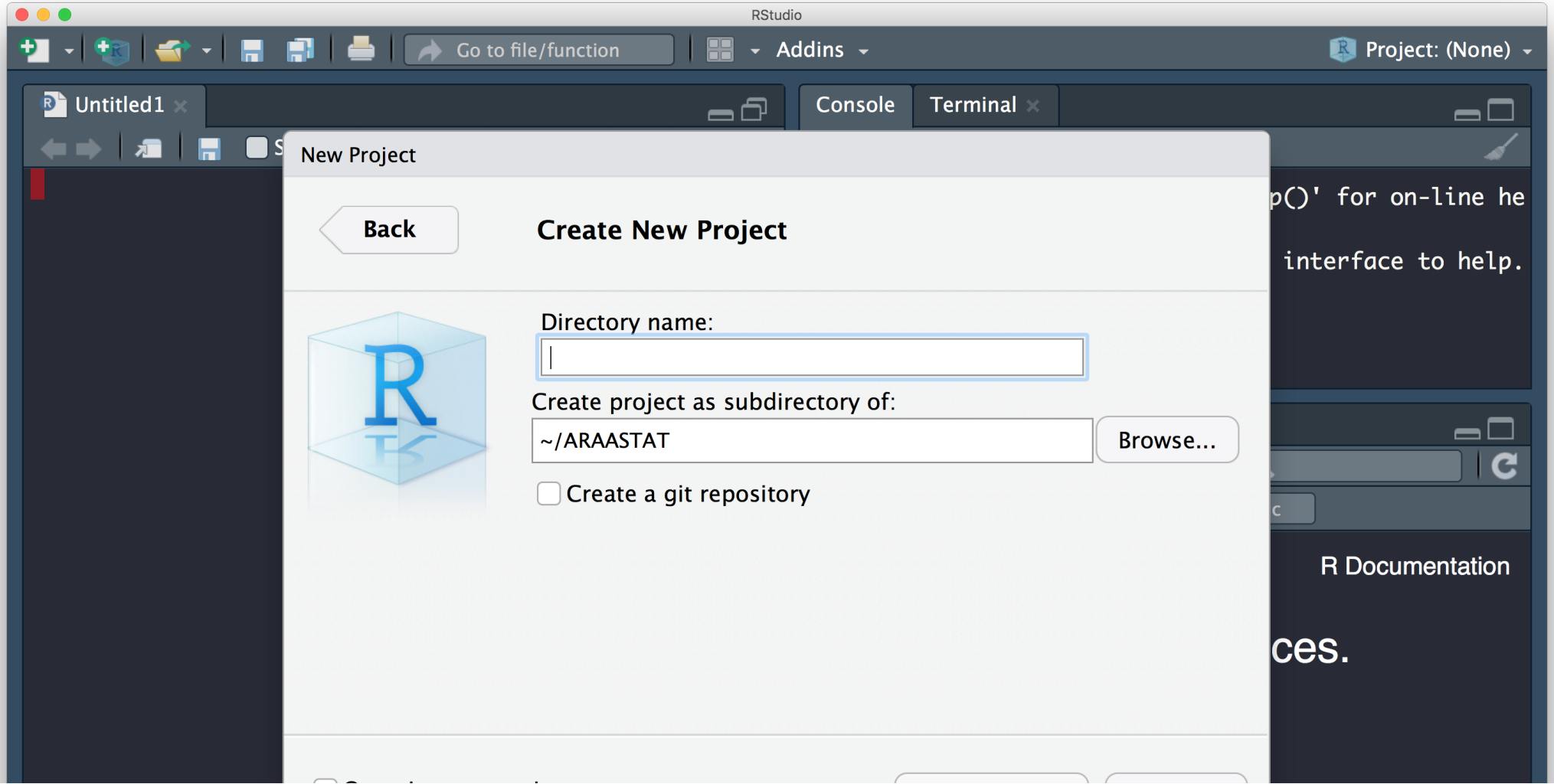
RStudio Projects



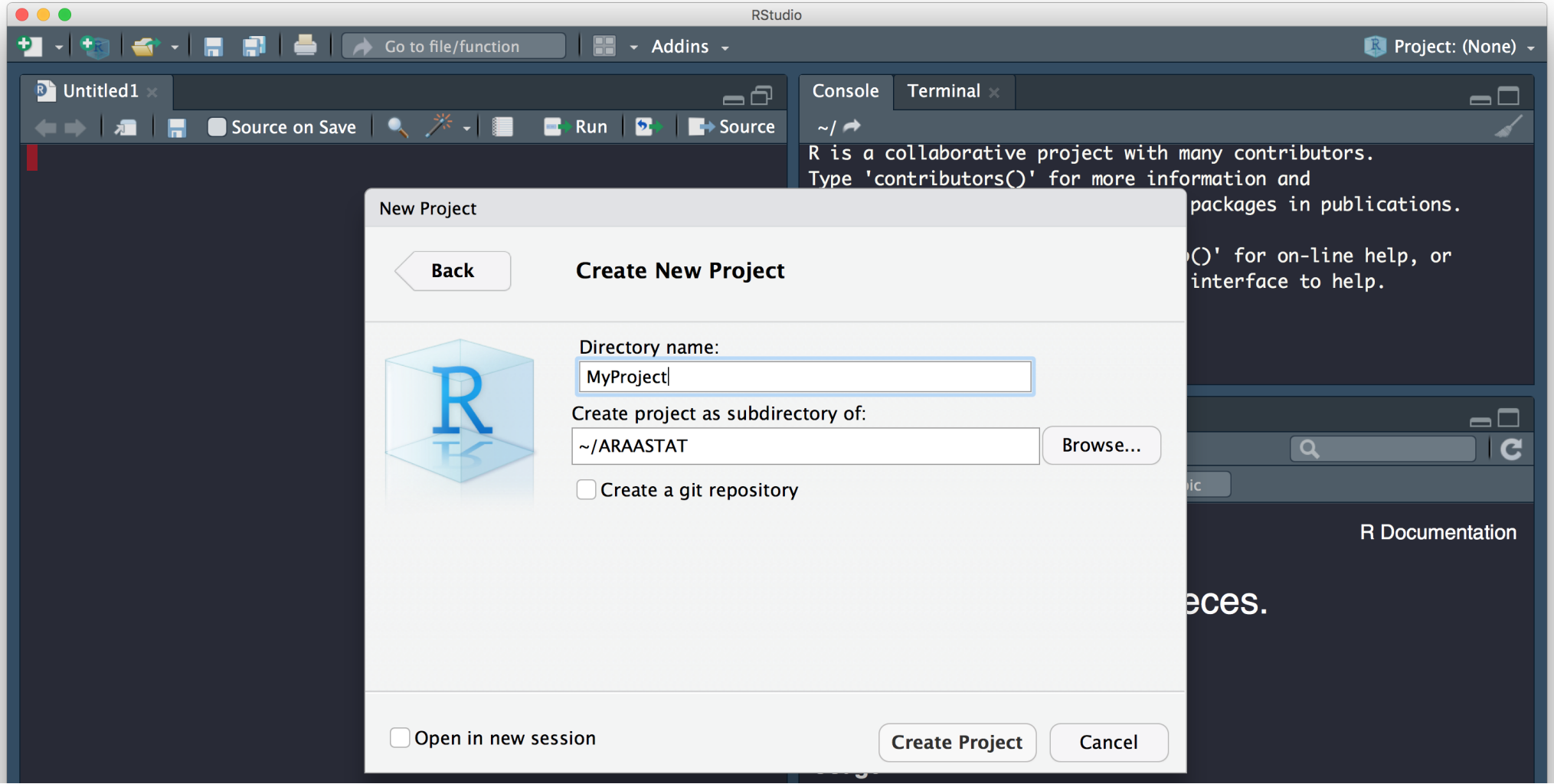
RStudio Projects



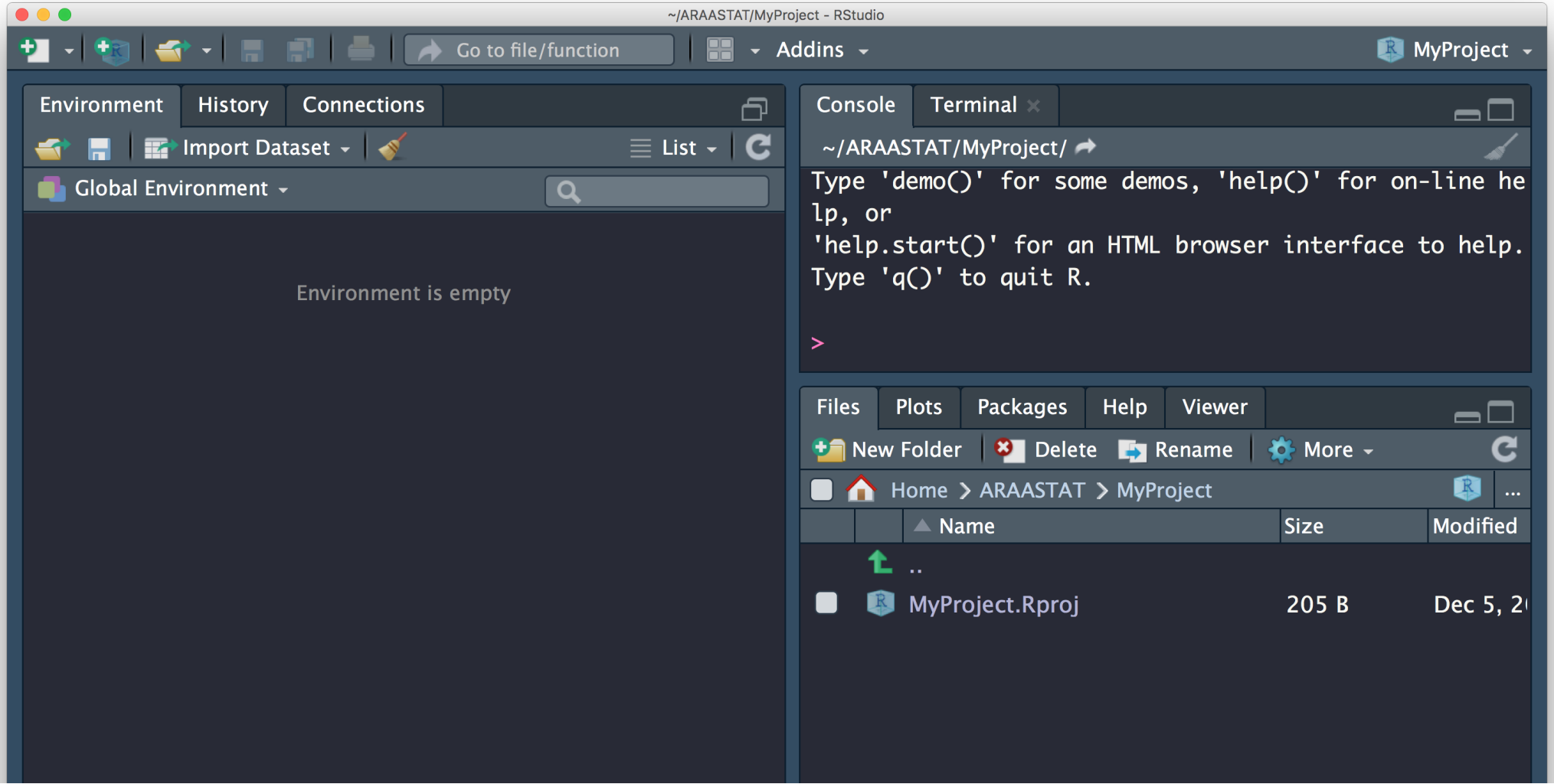
RStudio Projects



RStudio Projects



RStudio Projects



RStudio Projects

When you create a Project, the following obvious things happen:

1. RStudio puts you into the right directory/folder
2. Creates a .Rproj file containing project options
    - You can double-click on the .Rproj file to open the project in RStudio
3. Displays the project name in the project toolbar (right top of the window)

RStudio Projects

The following not-so-obvious things happen:

1. A new R session (process) is started
2. The .Rprofile file in the project's main directory (if any) is sourced by R
3. The .RData file in the project's main directory is loaded (this can be controlled by an option).
4. 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).
5. The current working directory is set to the project directory.
6. Previously edited source documents are restored into editor tabs, and
7. Other RStudio settings (e.g. active tabs, splitter positions, etc.) are restored to where they were the last time the project was closed.

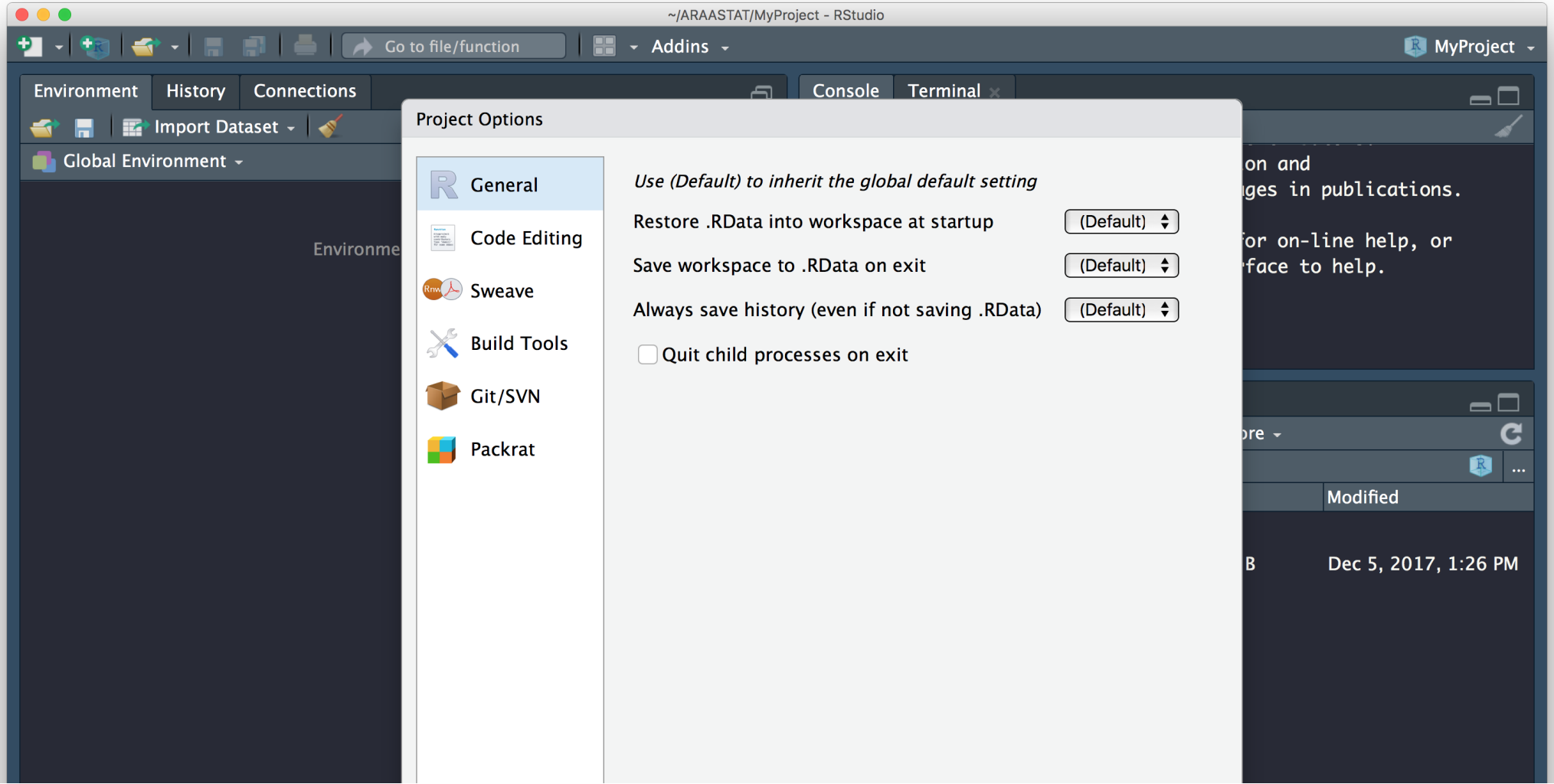
RStudio Projects

I use Projects so that:

1. I'm always in the right directory for the project
2. I don't contaminate one project's analysis with another (different sandboxes)
3. I can access different projects quickly
4. I can version control them (Git) easily (topic for beyond this class)
5. I can customize options per project

A great reference for git from a R user's perspective is Jenny Bryan's excellent happygitwithr.com

RStudio Projects



BIOF 339: Practical R

Project organization

Project structure

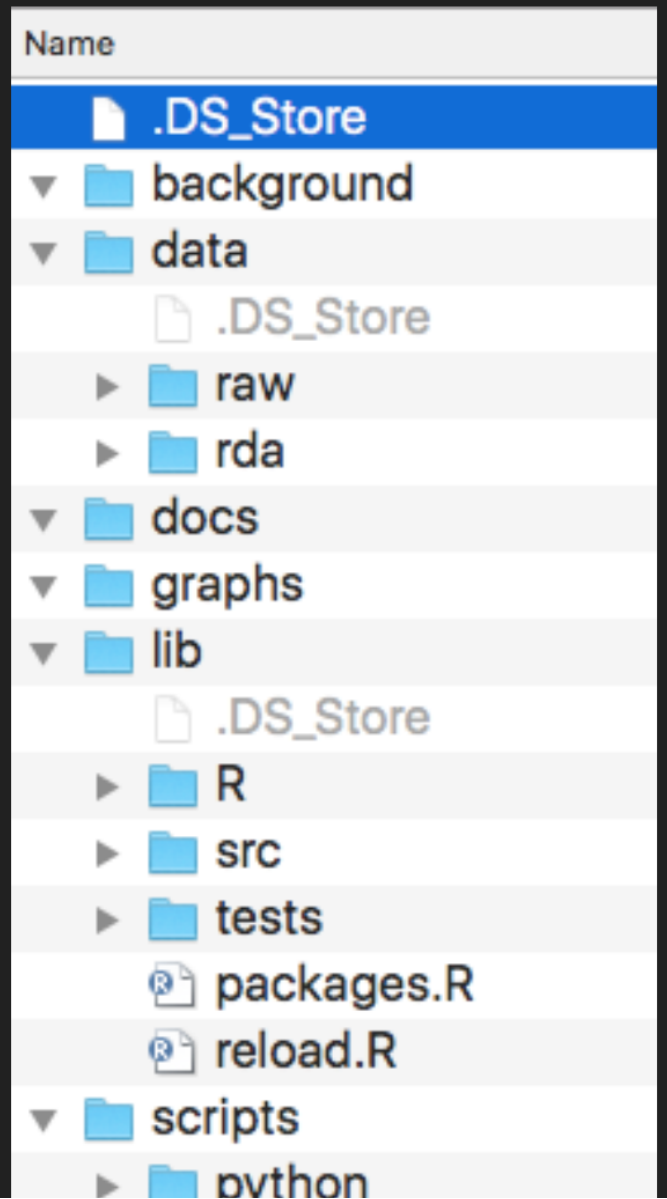
I always work with RStudio Projects to encapsulate my projects.

However, each project needs to maintain a file structure to know where to find things

Use a template to organize each project

- Before you even get data
- Set up a particular folder structure where
  - You know what goes where
  - You already have canned scripts/packages set up
- Make sure it's the same structure every time
- Next time you visit, you don't need to go into desperate search mode

MY STRUCTURE



Background materials

Raw data (storage, not to be touched)

Intermediate and final R data sets

Generated documents (docx, html, pdf)

Graphs (pdf, png, tiff)

Custom R functions (all of them, without exception)

Custom C/C++ functions

Unit tests

List of R packages for the project

Automated loading of functions and packages in a separate environment

Scripts for file management and conversion

File naming

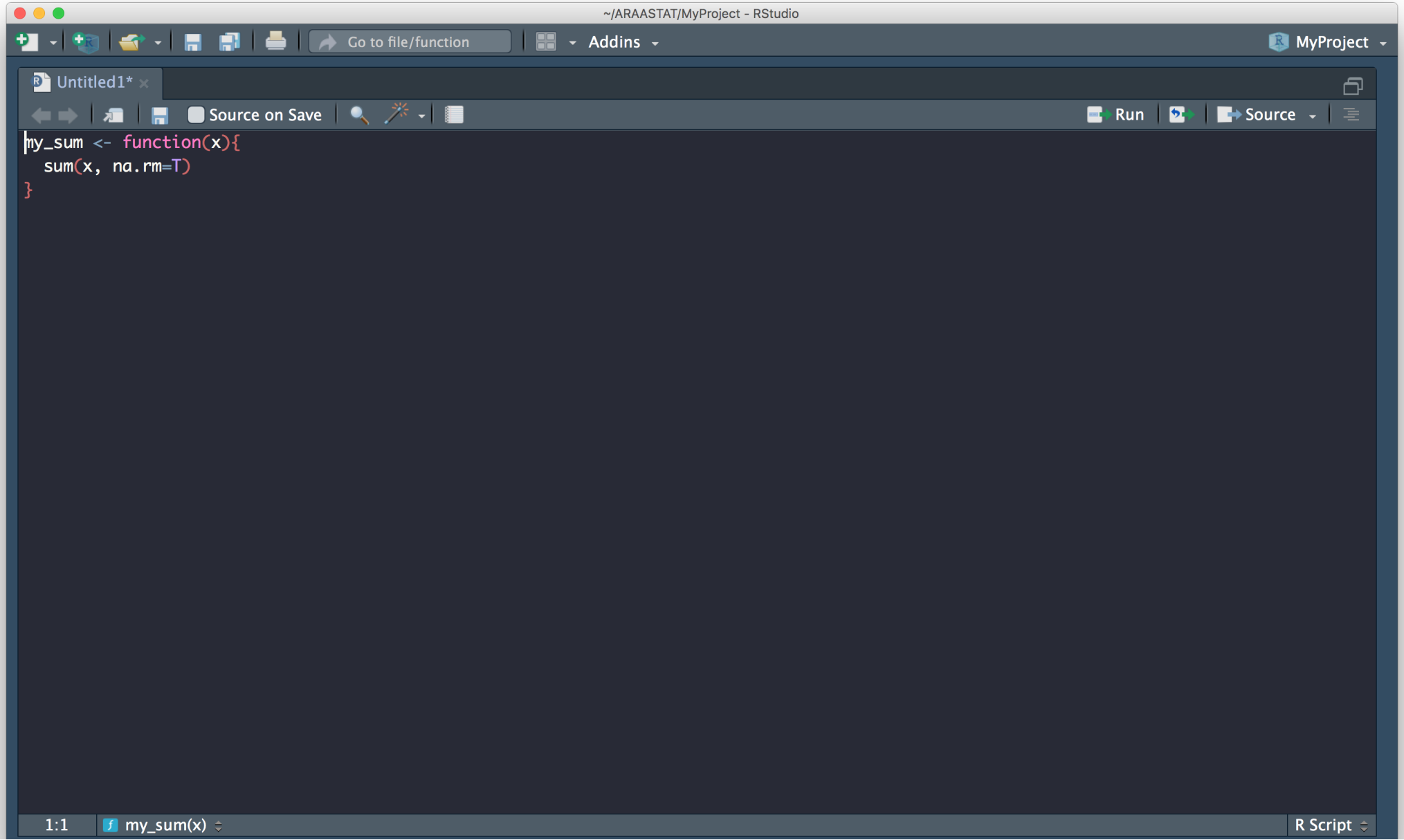
- Use descriptive file names
- Be explicit
  - File1.R, File4.R won't help you
  - DataMunging.R, RegressionModels.R will
- Well-chosen names saves a lot of time and heartache
- Another choice is to let yourself know the order of how things are to be run in a workflow by naming files
  - 01-Ingestion.R
  - 02-Munging.R
  - 03-EDA.R
  - 04-Modeling.R

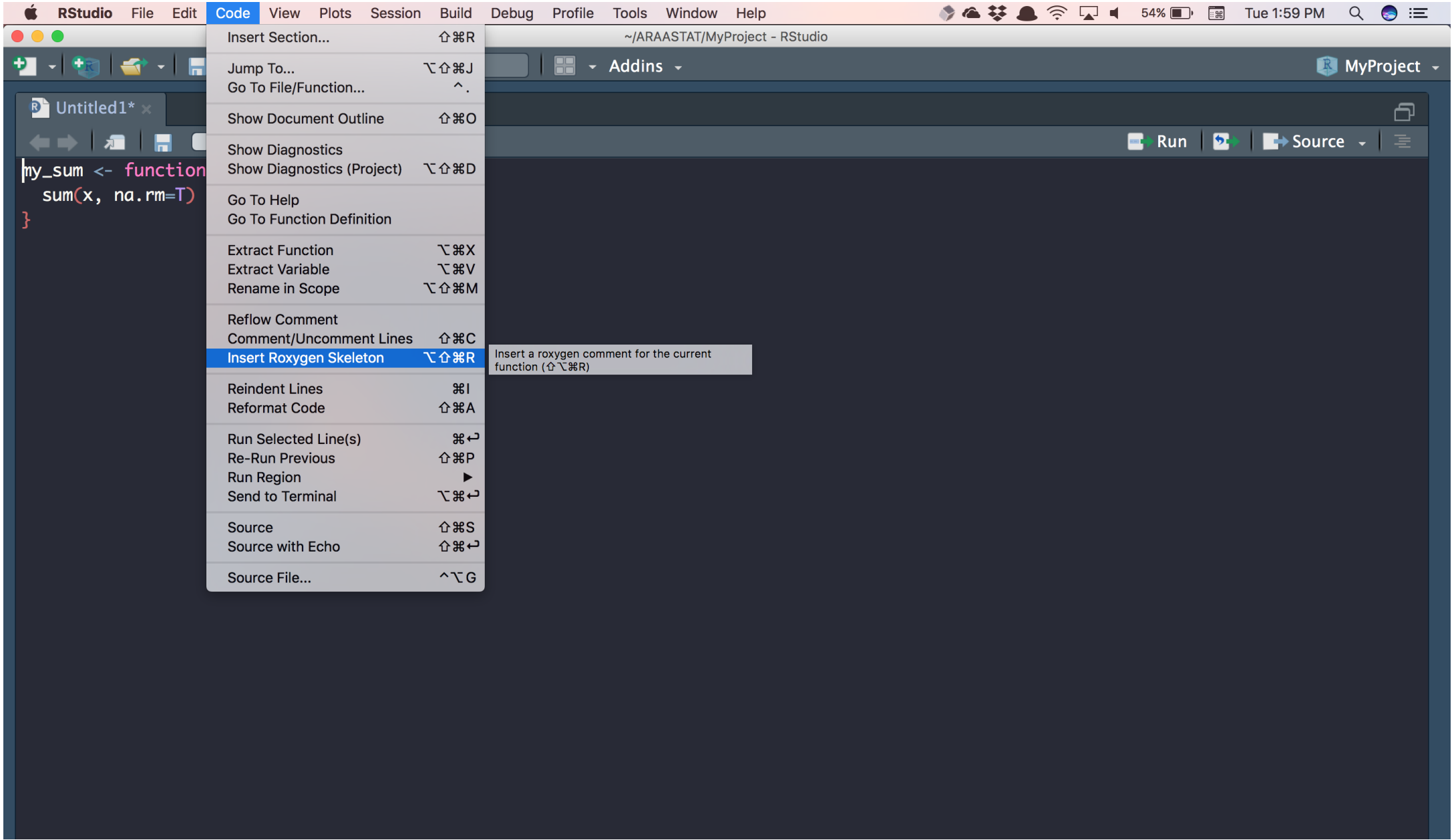
Documentation

- Create at least a README file to describe what the project is about.
- I've started creating a "lab notebook" for data analyses
  - Usually named `Notebook.Rmd`
  - Either a straight R Markdown file or a R Notebook
  - Keep notes on
    - What products (data sets, tables, figures) I've created
    - What new scripts I've written
    - What new functions I've written
    - Notes from discussions with colleagues on decisions regarding data, analyses, final products

Documentation

- Document your code as much as you can
  - Copious comments to state what you're doing and why
- If you write functions
  - Use Roxygen to document the inputs, outputs, what the function does and an example


~/ARAASTAT/MyProject - RStudio
Go to file/function
Addins
MyProject
Untitled1*
Source on Save
Run
Source
my_sum <- function(x){
sum(x, na.rm=T)
}
1:1
my_sum(x)
R Script


RStudio File Edit Code View Plots Session Build Debug Profile Tools Window Help
54% Tue 1:59 PM
~/ARAASTAT/MyProject - RStudio
Addins
MyProject
Untitled1*
Run
Source
my_sum <- function
sum(x, na.rm=T)
}
Insert Section... ⇧⌘R
Jump To... ⌥⇧⌘J
Go To File/Function... ^.
Show Document Outline ⇧⌘O
Show Diagnostics
Show Diagnostics (Project) ⌥⇧⌘D
Go To Help
Go To Function Definition
Extract Function ⌥⌘X
Extract Variable ⌥⌘V
Rename in Scope ⌥⇧⌘M
Reflow Comment
Comment/Uncomment Lines ⇧⌘C
Insert Roxygen Skeleton ⌥⇧⌘R
Insert a roxygen comment for the current function (⇧⌥⌘R)
Reindent Lines ⌘I
Reformat Code ⇧⌘A
Run Selected Line(s) ⌘↩
Re-Run Previous ⇧⌘P
Run Region ▶
Send to Terminal ⌥⌘↩
Source ⇧⌘S
Source with Echo ⇧⌘↩
Source File... ^⌥G

Function sanity

The computer follows direction really well

- Use scripts/functions to derive quantities you need for other functions
- Don't hard-code numbers

```
runif(n = nrow(dat), min = min(dat$age), max = max(dat$age))
```

rather than

```
runif(n = 135, min = 18, max = 80)
```

- This reduces potential errors in data transcription
  - These are really hard to catch

Create functions rather than copy-paste code

- If you're doing the same thing more than twice, write a function (*DRY principle*)
- Put the function in its own file, stored in a particular place
  - I store them in `lib/R`.
  - Don't hide them in general script files where other stuff is happening
  - Name the file so you know what's in it
  - One function or a few related functions per file
- Write the basic documentation NOW!

An example from my own work is here

Loading your functions

```
funcfiles <- dir('lib/R', pattern = '.R')
purrr::map(funcfiles, source)
```

Package sanity

Suppose you need to load a bunch of packages and aren't sure whether they are installed on your system or not. You can certainly look in `installed.packages`, but if you have 1000s of packages, this can be slow.

You can use `require`:

```
x <- require(ggiraph)
```

```
Loading required package: ggiraph
```

```
x
```

```
[1] TRUE
```

A more elegant solution is using the pacman package

```
if (!require("pacman")) install.packages("pacman") # make sure pacman is installed
pacman::p_load(ggiraph, stargazer, kableExtra)
```

This will install the package if it's not installed, and then load it up.

Manipulate data with care

- Keep a pristine copy of the data
- Use scripts to manipulate data for reproducibility
  - Can catch analyst mistakes and fix
- Systematically verify and clean
  - Create your own Standard Operating Plan
- Document what you find
  - Lab notebook (example)

Manipulate data with care

- The laws of unintended consequences are vicious and unforgiving, and appear all too frequenty at the data munging stage
- For example, data types can change (factor to integer)
- Test your data at each stage to make sure you still have what you think you have

Track data provenance through the pipeline

- Typically:

  > Raw data >> Intermediate data >> Final data >> data for sub-analyses >> data for final tables and figures

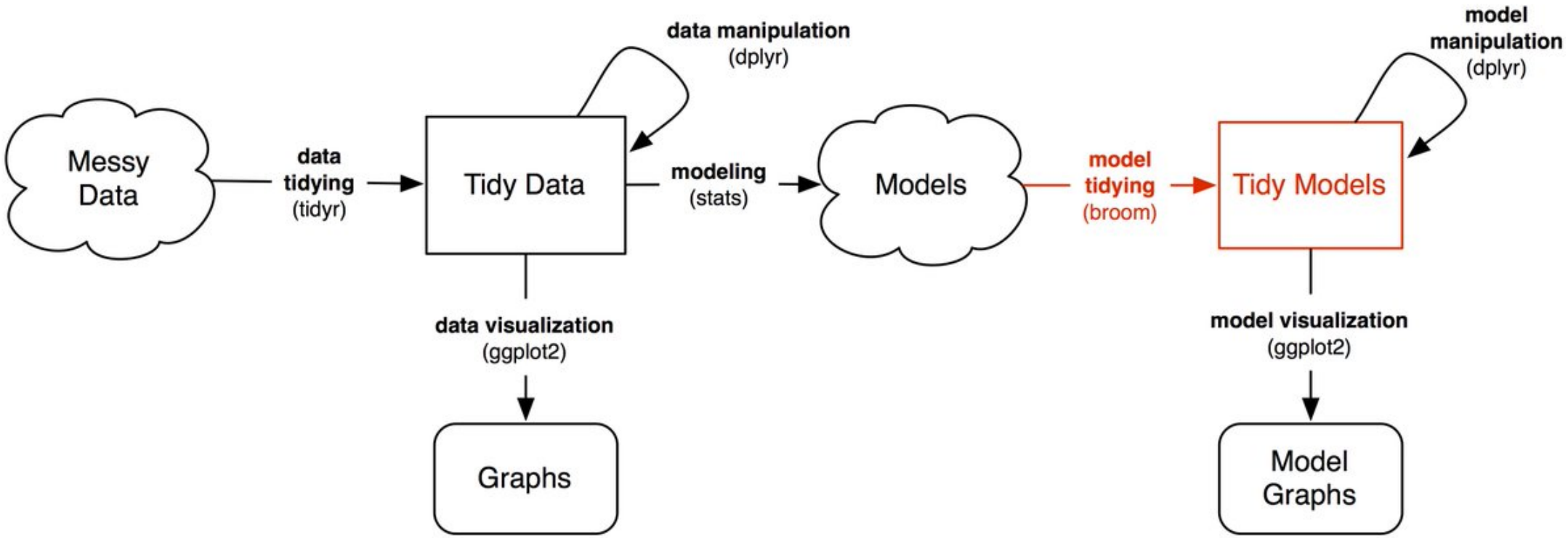
- Catalog and track where you create data, and where you ingest it
- Make sure there are no loops!!

Share preliminary analysis for a sniff

- Share initial explorations with colleagues so they pass a "sniff" test
  - Are data types what you expect
  - Are data ranges what you expect
  - Are distributions what you expect
  - Are relationships what you expect

- This stuff is important and requires deliberate brain power
- May require feedback loop and more thinking about the problem

A general pipeline



David Robinson, 2016

Know where final tables and figures come from

- I create separate files for creating figures and tables for a paper
  - They're called `FinalTables.R` and `FinalFigures.R`. Duh!
- This provides final check that right data are used, and can be updated easily during revision cycle
- It's a long road to this point, so make sure things are good.