Using RStudio Projects and Writing Good Programs

Ashley I Naimi

Oct 2022

Contents

1 RStudio Projects

When carrying out a scientific project, there are many elements, skills, and practices that must come together to bring the project to a successful end. For example, as a project scientist, you or your team may be responsible for constructing a cohort and collecting data, processes and cleaning data, using statistical methods to analyze the data, and understanding how all of these steps combine to enable you to interpret the results.

However, there is another set of skills that involve how to "build" and carry out a project from the ground up, while avoiding common pitfalls and sources of friction in carrying out and collaborating on a scientific analysis.

In this section, we'll discuss strategies and tools to optimize the scientific workflow and avoid these pitfalls.

2 Setting Up Your Project Workspace

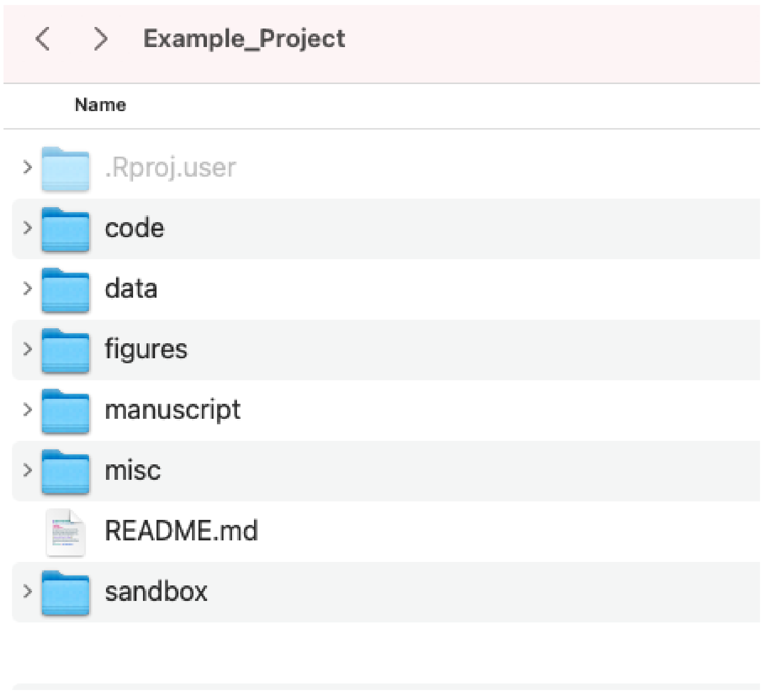
The first step to carry out when starting a new project is to set up a folder structure that works for your team and meets the needs of your project. The approach one should take when setting up a project folder is summed up in the following slogan:

> "everything has a place and everything in its place"

In following this slogan, we need to first establish a project workspace that has a "place for everything". This can include a place for data, code, figures, a manuscript or report to be written, a folder for some miscellaneous items that may be needed to structure or style the analytic results, and a place to try some things that may not be directly related to the analysis.

To provide a practical example, consider that we have a folder on our computer named "Example_Project", which will be the location where we will carry out our analyses.[1] The needs we just listed suggest that a folder structure within the "Example_Project" folder such as the following could be of use:

[1] This full example is available as a part of the course materials.



.Rproj.user
code
data
figures
manuscript
misc
README.md
sandbox

In the above Figure, the **code folder** contains all of the *source code files* needed deploy the analysis.[2] This folder can be considered the main or primary folder, because it contains a record of all the steps needed to produce (or reproduce) the results.

[2] recall the data science pipeline note from the previous section. In this "pipeline" framework, the key *real* elements of an analytic project are the source code files, and not the output that is generated from these files.

The **data folder** contains everything that is needed for the code to run. For example, this folder could include the raw data, the analytic data, and then any other intermediary datasets needed to conduct the analysis of primary interest.

The **figures folder** contains all images and figures relevant to the project (and generated from the source code).

The **manuscript** or **report folder** contains the (e.g. Microsoft Word) document that will be used to report the findings to the wider scientific community, public, or other parties.

The **misc folder** is a place that can contain all the other items needed to carry out the analysis that may not belong in any of the other folders. This can include notes from a course or website, email correspondence related to the project, .css files that allow you to style your results in a certain way, conference abstract submission instructions, and other relevant items.

The **sandbox folder** is a place to "play around" in. During the course of a project, you may come up with an idea that is either better than the one you are implementing, that may take the project in a different direction, or that may create a spin-off (i.e., second project) of the current project. The sandbox is a place where you can write code or generate data to explore this idea without affecting the structure of the main project folders.

Finally, the **README.md** file is a guide summarizing anything from the overall mission and goal of the project, down to the specific details of elements, items, code, figures, or whatever. The README file provides instructions for where things might deviate from a more traditional analytic protocol, or other special circumstances that users, collaborators, and/or reviewers of the code or project can become familiar with necessary details.

Once the project folder is created, we are ready to create an RStudio project that allows us to navigate this folder with the RStudio IDE.

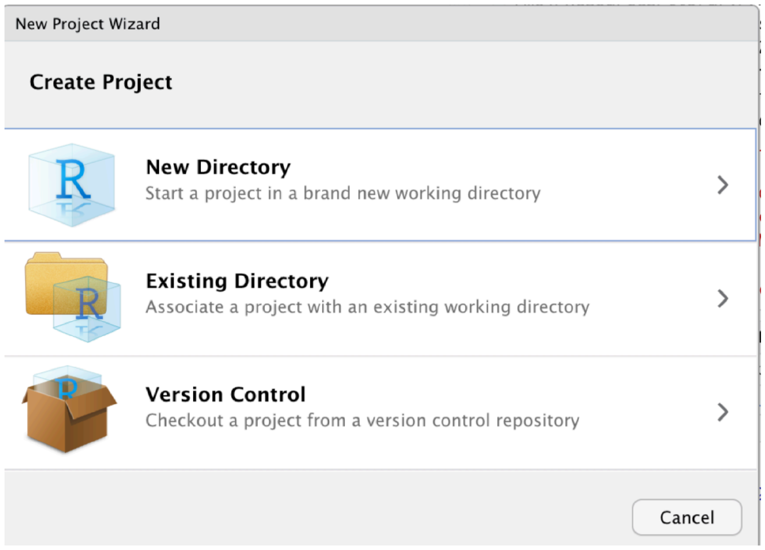
> **The Importance of Folder Structure**:
> Having a consistent and systematic approach to structuring project folders is beneficial for two reasons. First, if the team working on the project understands where things go and why, it makes collaboration more efficient as less time will be spent figuring out how things are organized. Similarly, there will inevitably be a time where you have to pause work on a project, and revisit it several weeks, months, or years later. Not having to spend time re-familiarizing yourself with the project landscape and folder structure can be equally efficient (i.e., collaborating with future self is easier). Second, having an organized project landscape makes it easier to avoid errors and facilitates reproducible scientific results.

3 Projects and the `here` package

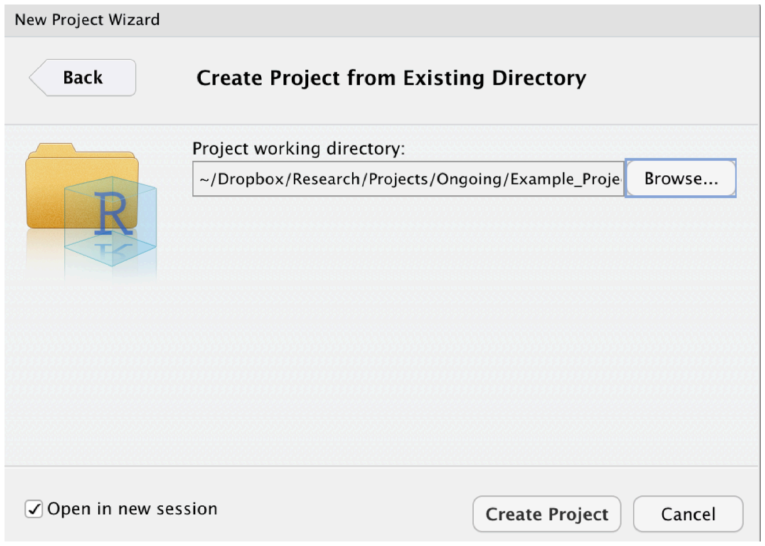
With the project folder structure created, the next step is to link it to an RStudio Project. To do this, we open up an RStudio session, and select

File > New Project…

In doing so, the following dialogue box should pop up:

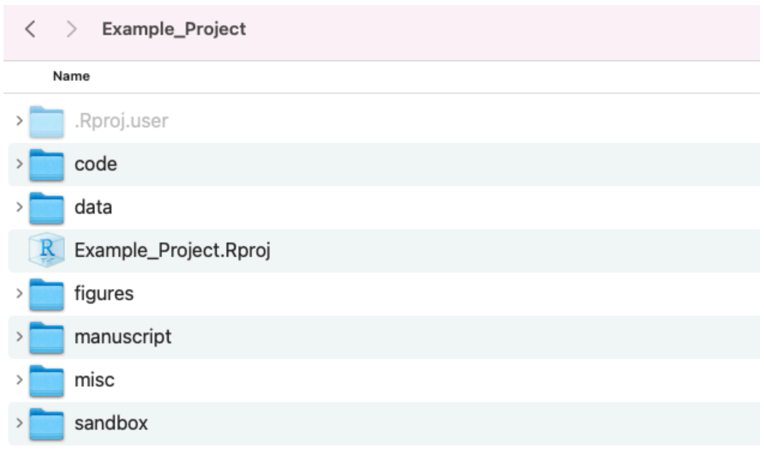

New Project Wizard
Create Project
New Directory
Start a project in a brand new working directory
Existing Directory
Associate a project with an existing working directory
Version Control
Checkout a project from a version control repository
Cancel

After selecting “Existing Directory”, this popup should change to the following:



~/Dropbox/Research/Projects/Ongoing/Example_Proje

After browsing to the location of the “Example_Project” folder, clicking “Create Project” will result in the following file being added to the project folder:

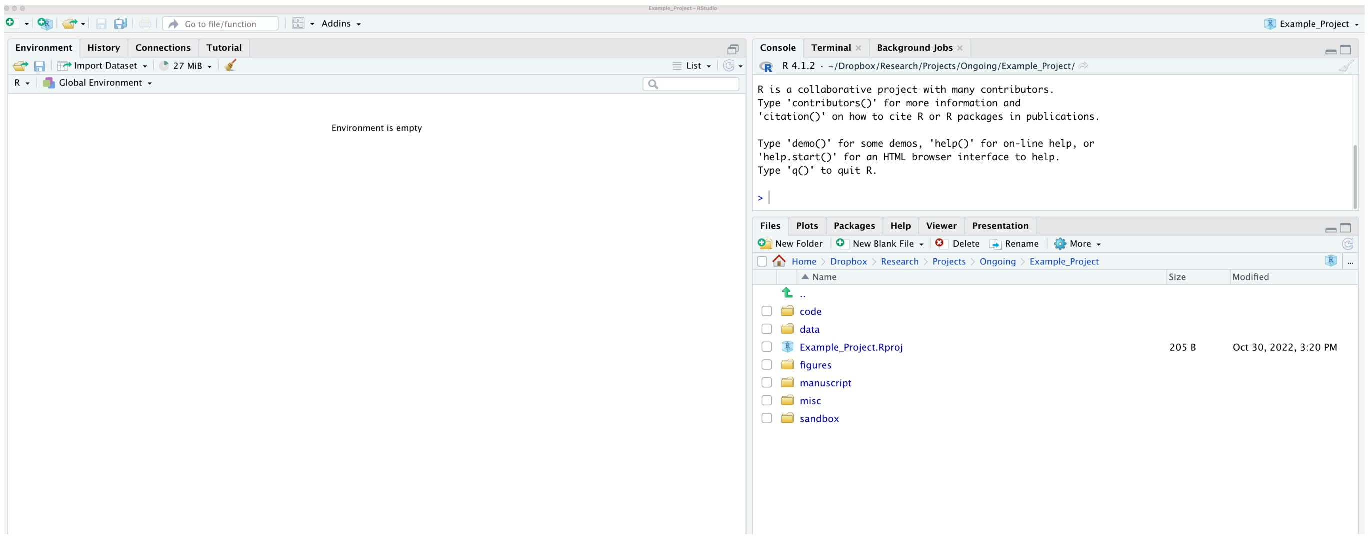


Example_Project

Name

- .Rproj.user
- code
- data
- Example_Project.Rproj
- figures
- manuscript
- misc
- sandbox

Double clicking on the `.Rproj` file will open an RStudio session that is isolated to the the `Example_Project` of interest.



Environment is empty

```
R 4.1.2 · ~/Dropbox/Research/Projects/Ongoing/Example_Project/

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

Home › Dropbox › Research › Projects › Ongoing › Example_Project

Name	Size	Modified
..		
code		
data		
Example_Project.Rproj	205 B	Oct 30, 2022, 3:20 PM
figures		
manuscript		
misc		
sandbox		

What's the benefit of using an RStudio Project, instead of just opening R? With RStudio projects, analysts can work on several projects at the same time, while keeping the directories, workspaces, histories, environments, and source documents separate between each project. In other words, each scientific project gets its own unique working environment. With proper coding practices,[3] using RStudio can make collaboration a lot easier.

[3] Including avoiding the use of `setwd()` and relying on the `here` package instead, as we will see below.

4 Writing Good R Programs in the Project Workspace

There are generally a few principles to follow when writing R programs in your project workspace. Let's first create a file we'll call `raw-data.R` that will allow us download the dataset of interest from the Web, and start cleaning it. Within this raw data file, we'll first ensure that all the packages we need are installed and loaded.

As a general principle, it's always a good idea to include, at the top of your file, commands that you will need to execute the program. In our case, we include the following at the top of the `raw-data.R` program:

```r
packages <- c("tidyverse", "here", "VIM")

for (package in packages) {
    if (!require(package, character.only = T,
        quietly = T)) {
        install.packages(package, repos = "http://lib.stat.cmu.edu/R/CRAN")
    }
}
```

```
## VIM is ready to use.

## Suggestions and bug-reports can be submitted at: https://github.com/statistikat/VIM/issues

##
## Attaching package: 'VIM'

## The following object is masked from 'package:datasets':
##
##     sleep
```

```r
for (package in packages) {
    library(package, character.only = T)
}
```

Next, we'll use the tidyverse `read_csv` function to access the NHEFS data from the following website:

https://www.hsph.harvard.edu/miguel-hernan/causal-inference-book/

```
file_loc <- url("https://cdn1.sph.harvard.edu/wp-content/uploads/sites/1268/1268/20/nhefs.csv")
nhefs_data <- read_csv(file_loc)
```

```
## Rows: 1629 Columns: 64
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (64): seqn, qsmk, death, yrdth, modth, dadth, sbp, dbp, sex, age, race, ...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
# let's look at the data
dim(nhefs_data)
```

```
## [1] 1629   64
```

```
nhefs_data
```

```
## # A tibble: 1,629 x 64
##     seqn  qsmk death yrdth modth dadth   sbp   dbp   sex   age  race income
##    <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  <dbl>
##  1   233     0     0    NA    NA    NA   175    96     0    42     1     19
##  2   235     0     0    NA    NA    NA   123    80     0    36     0     18
##  3   244     0     0    NA    NA    NA   115    75     1    56     1     15
##  4   245     0     1    85     2    14   148    78     0    68     1     15
##  5   252     0     0    NA    NA    NA   118    77     0    40     0     18
##  6   257     0     0    NA    NA    NA   141    83     1    43     1     11
##  7   262     0     0    NA    NA    NA   132    69     1    56     0     19
##  8   266     0     0    NA    NA    NA   100    53     1    29     0     22
##  9   419     0     1    84    10    13   163    79     0    51     0     18
## 10   420     0     1    86    10    17   184   106     0    43     0     16
## # ... with 1,619 more rows, and 52 more variables: marital <dbl>, school <dbl>,
## #   education <dbl>, ht <dbl>, wt71 <dbl>, wt82 <dbl>, wt82_71 <dbl>,
## #   birthplace <dbl>, smokeintensity <dbl>, smkintensity82_71 <dbl>,
## #   smokeyrs <dbl>, asthma <dbl>, bronch <dbl>, tb <dbl>, hf <dbl>, hbp <dbl>,
## #   pepticulcer <dbl>, colitis <dbl>, hepatitis <dbl>, chroniccough <dbl>,
## #   hayfever <dbl>, diabetes <dbl>, polio <dbl>, tumor <dbl>,
## #   nervousbreak <dbl>, alcoholpy <dbl>, alcoholfreq <dbl>, ...
```

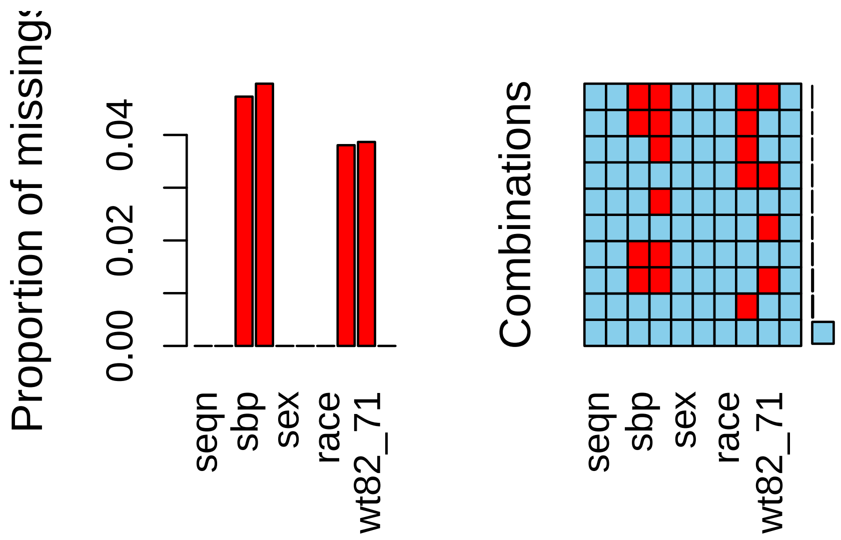
Let's start with selecting the variables we would like to include in our analytic dataset, handle any missing data, and construct any derived variables. We can do this using the piping[4] structure and key functions in the tidyverse:

[4] Note that the shortcut for adding a pipe in code on a Mac is `Command+Shift+M`, and on Windows is `Ctrl+Shirt+M`

```
nhefs_data <- nhefs_data %>%
    select(seqn, qsmk, sbp, dbp, sex, age,
        race, income, wt82_71, death)
```

Let's look at missing data using the `aggr` function in the VIM package:

```
aggr(nhefs_data)
```



We can also use a function we created to evaluate missingness:

```
propMissing <- function(x) {
    mean(is.na(x))
}

apply(nhefs_data, 2, propMissing)
```

```
##        seqn       qsmk        sbp        dbp        sex        age       race
## 0.00000000 0.00000000 0.04726826 0.04972376 0.00000000 0.00000000 0.00000000
##     income     wt82_71      death
## 0.03806016 0.03867403 0.00000000
```

Next we can delete missing data to implement a complete case analysis with our analytic dataset:

```
nhefs_data <- nhefs_data %>%
    na.omit()
```

And we can construct a new variable, mean arterial pressure (MAP), using our measures of systolic and diastolic blood pressure, and remove the blood pressure variables once MAP is created:

```
nhefs_data <- nhefs_data %>%
    mutate(map = dbp + (sbp - dbp)/3) %>%
    select(-sbp, -dbp)
```

Finally, with our variables selected, missing data removed, and new variable created, we can explore our dataset[5]:

[5] In the Example Project code, I also show you how to generate a histogram for age, and save it to the figures folder using the `here` package, which is explained below.

```
dim(nhefs_data)
```

```
## [1] 1476    9
```

```
nhefs_data %>%
    print(n = 5)
```

```
## # A tibble: 1,476 x 9
##    seqn  qsmk   sex   age  race income wt82_71 death   map
##   <dbl> <dbl> <dbl> <dbl> <dbl>  <dbl>   <dbl> <dbl> <dbl>
## 1   233     0     0    42     1     19  -10.1      0 122. 
## 2   235     0     0    36     0     18    2.60     0  94.3
## 3   244     0     1    56     1     15    9.41     0  88.3
## 4   245     0     0    68     1     15    4.99     1 101. 
## 5   252     0     0    40     0     18    4.99     0  90.7
## # ... with 1,471 more rows
```

Now that we've constructed our analytic dataset, it's time we save it to our data folder. The "traditional" way to do this is to start by setting the working directory.

For example, in this pdf note package, which was generated in a project folder called "useR" on my computer, we could set the working directory as follows:

```
getwd()
```

```
## [1] "/Users/ain/Dropbox/Teaching/useR/3. Module3"
```

From this, we can use the information to set the working directory to the "useR" folder. What this does is tell R that the root folder in which all project related directory changes should occur start in the "useR" folder. This way, we can simply save the analytic dataset we created with the `write_csv` function as in the following example:

```
setwd("/Users/ain/Dropbox/Teaching/useR")

getwd()
```

```
## [1] "/Users/ain/Dropbox/Teaching/useR"
```

```
write_csv(nhefs_data, file = "./data/analytic_data.csv")
```

The **major problem** with the above approach is that it makes the code completely dependent upon the computer on which it is being run. If I shared this code with a colleague, that person would have to scour my code files to change each instance of `setwd()` to the directory structure that's specific to their computer. This can create serious problems for collaboration and reproducibility.

Fortunately, there's another solution. In fact, the `here` package was developed precisely to address this issue. When we load the `here` package, we obtain the following information:

```
here()
```

```
## [1] "/Users/ain/Dropbox/Teaching/useR"
```

This package automatically sets the top level working directory to be the location of the RStudio project. With this package, we can then save our analytic dataset easily as:

```
write_csv(nhefs_data, file = here("data",
    "analytic_data.csv"))
```

5 General Overview of the Process

Let's take a moment to consider the general approach we just took to construct an analytic dataset using the `raw-data.R` file. It may be useful to refer to the figure on the data science pipeline:

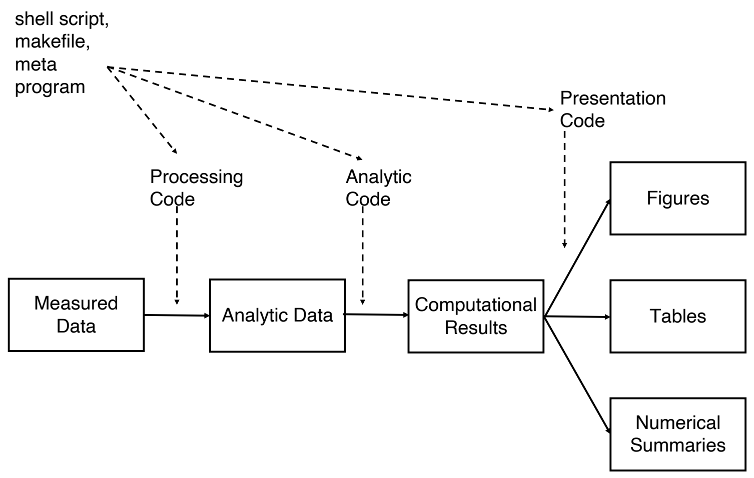


Figure 1: The Data Science Pipeline

Our `raw-data.R` file is basically an element of the "processing code" depicted in the Figure. What we did with this file is import some "raw" data, explore and manipulate, and save relevant items (analytic dataset, figures such as histograms) to their appropriate locations in teh folder structure.

This brings us to some general principles to follow when constructing a program file. First, our `raw-data.R` can be divided into three parts: 1) setup, where we install and load relevant libraries to do the work we need; 2) the code needed to carry out the work; and 3) the output section, where we save what we need to carry out the work.