BIOSTATS

Akira Terui

Last updated on 2022-09-16

Contents

Introduction

This textbook aims to introduce fundamental statistical techniques and their applications to biological data. A unique aspect of this book is the "flipped-order" introduction. Many statistics courses start with theory; yet, I found it difficult for those unfamiliar with statistics. I will start with a real example of the method, followed by the explanation for an underlying theory/concept. The author is an ecologist, so some methods in this book might not be popular in other fields.

Chapter 1

Project Management

1.1 R Project

For this entire book, I will use `R Project` as a fundamental unit of work-space, in which all the relevant materials (e.g., R scripts `.R` and data files) are assembled together. There are many ways to organize your project, but I usually make a single `R Project` for a collection of scripts and data that will lead to a single publication (see example here). To setup an `R Project` you will need *R Studio* in addition to base *R*. While *R* is a stand-alone software, I strongly recommend to use it with *R Studio*. *R Studio* has many functions that help your data analysis. *R* and *R Studio* can be installed from the following websites:

- R (you can choose any CRAN mirror to download)
- R Studio

Once you open *R Studio*, you will see the following interface (Figure 1.1). There are three major panels in its first appearance – `Console`, `Environment`, and `Files`. `Console` is the place where you write your codes and execute calculation/data manipulation/analysis. `Environment` lists items you saved as an object. `Files` list any files in a designated location in your computer.

Although you can work on your data as it appears, **it is actually a BAD idea**. As you work on your project, numerous materials will be generated. How do you manage files? If you randomly locate those materials within your computer, you will lose necessary items sooner or later. For this reason, I assemble all the relevant materials in a single `R Project`. You can create a new `R Project` with the following procedure.

a. Go to `File > New Project` on the top menu
b. Select `New Directory`
c. Select `New Project`

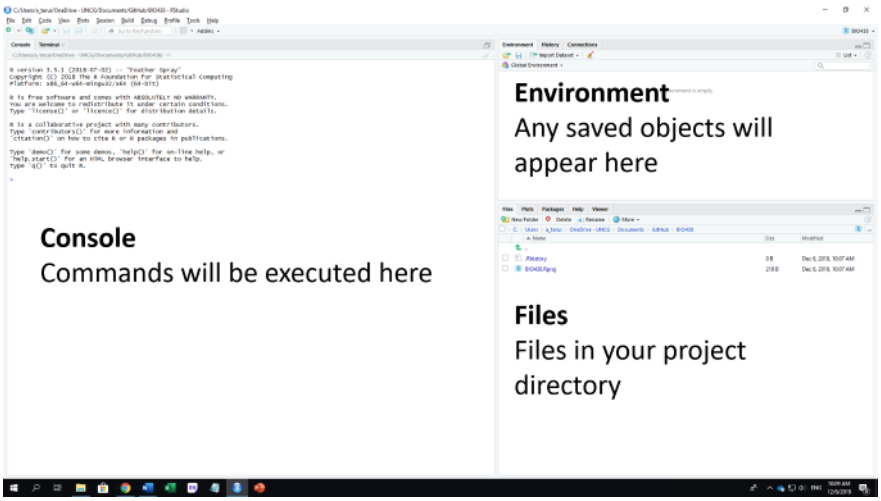

Figure 1.1: R Studio interface.

A new window pops up and prompts you to name a directory with a location in your computer. Click `Browse` to select a location for the directory.

IMPORTANT: When you locate your project directories in your computer, I would strongly recommend to create a designated space. For example, in my computer, I have a folder named `/r_project` in which all the `R Project` directories are located.

1.2 Internal Structure

The internal structure of an `R Project` is extremely important to navigate yourself (others once it's published). `R Project` will be composed of multiple types of files, typically `.R`, `.csv`, `.rds`, `.Rmd` among others. Unless those files are arranged in an organized manner, it is VERY LIKELY to make severe errors in coding. So I take this seriously. Table 1.1 is my suggested subdirectory structure.

Table 1.1: Suggested internal structure of `R Project`

Name	Content
`README.md`	Markdown file explaining contents in the `R Project`. Can be derived from `README.Rmd`.
`/code`	Sub-directory for R scripts (`.R`).
`/data_raw`	Sub-directory for raw data before data manipulation (`.csv` or other formats). Files in this sub-directory MUST NOT be modified unless there are changes to raw data entries.

Name	Content
`/data_format`	Sub-directory for formatted data (`.csv`, `.rds`, or other formats).
`/output`	Sub-directory for result outputs (`.csv`, `.rds`, or other formats). This may include statistical estimates from linear regression models etc.
`/rmd`	(Optional) Sub-directory for Rmarkdown files (`.Rmd`). Rmarkdown allows seamless integration of R scripts and text.

1.3 File Name

It is also critical to have **consistent naming rules** for your files. As you make progress on your project, the number of files in each sub-directory will increase, perhaps exponentially. You will find it difficult navigating yourself unless you have clear naming rules for files (and even worse for others). Here are some recommendations:

- **NO SPACE.** Use underscore.
  - Do: `script_week1.R`
  - Don't: `script week1.R`
- **NO UPPERCASE.** Use lowercase for file names.
  - Do: `script_week1.R`
  - Don't: `Script_week1.R`
- **BE CONSISTENT.** Apply consistent naming rules within a project.
  - Do: R scripts for figures always start with a common prefix, e.g., `figure_XXX.R` `figure_YYY.R`(`XXX` and `YYY` specifies further details).
  - Don't: R scripts for figures start with random text, e.g., `XXX_fig.R` , `Figure_Y2.R` , `plotB.R`.

1.4 Robust coding

While this is not mandatory, I strongly recommend to use *R Studio* with *Git* & *GitHub* (see Chapter 2 Appendix for guidance). Coding is innately error-prone[1]; every programmers, even bright ones, make mistakes - no exception. However, the critical difference between beginner and advanced programmers is whether they develop robust coding rules with a self-error-detection system. *Git* is the key to this. I will touch on this occasionally throughout this book.

[1] well WAY BETTER than manual data manipulation!

Chapter 2

Appendix

2.1 *Git & GitHub*

In this section, I will cover how to integrate *Git* and *GitHub* into *R Studio*. *R Studio* is excellent as is, but it becomes even better when combined with *Git* and *GitHub*. *Git* is a free and open source distributed version control system. ***Git* tracks changes in your codes codes while you work on your project so you are aware of any changes you made to your script (and other) files**. Tracking changes is extremely important to avoid unintended errors in your codes. This feature also helps avoid creating redundant files. While *Git* is a local system, it has an online counterpart called *GitHub*.

To make this system work, you'll need to go through some processes. The first step is to install *Git* onto your computer:

- **Windows**: Install *Git* from here. You will be asked about "Adjusting your PATH environment". Choose "*Git* from the command line and also from 3rd-party software" if it is not selected.
- **Mac**: Follow the instruction here.

Then open R Studio and do `Create Project` > `New Directory` > `New Project`. If you see a check box `Create a git repository`, check and create a new project (Figure 2.1). You will see a *Git* pane on the upper right panel.

If you can't find the above, do the following: `Tools` in the menu bar > `Terminal` > `New Terminal`, and type `where git` in the terminal. This will tell you where git executable is located in your computer. Then, go to `Tools` in the menu bar > `Global Options` > `Git/SVN`. You will see *Git executable* in the box, where you can specify the location of git executable.

Next, go to *GitHub* and create your account (free!). But, give some thoughts on your user name. My advice is the following. First, **use lowercase only.**

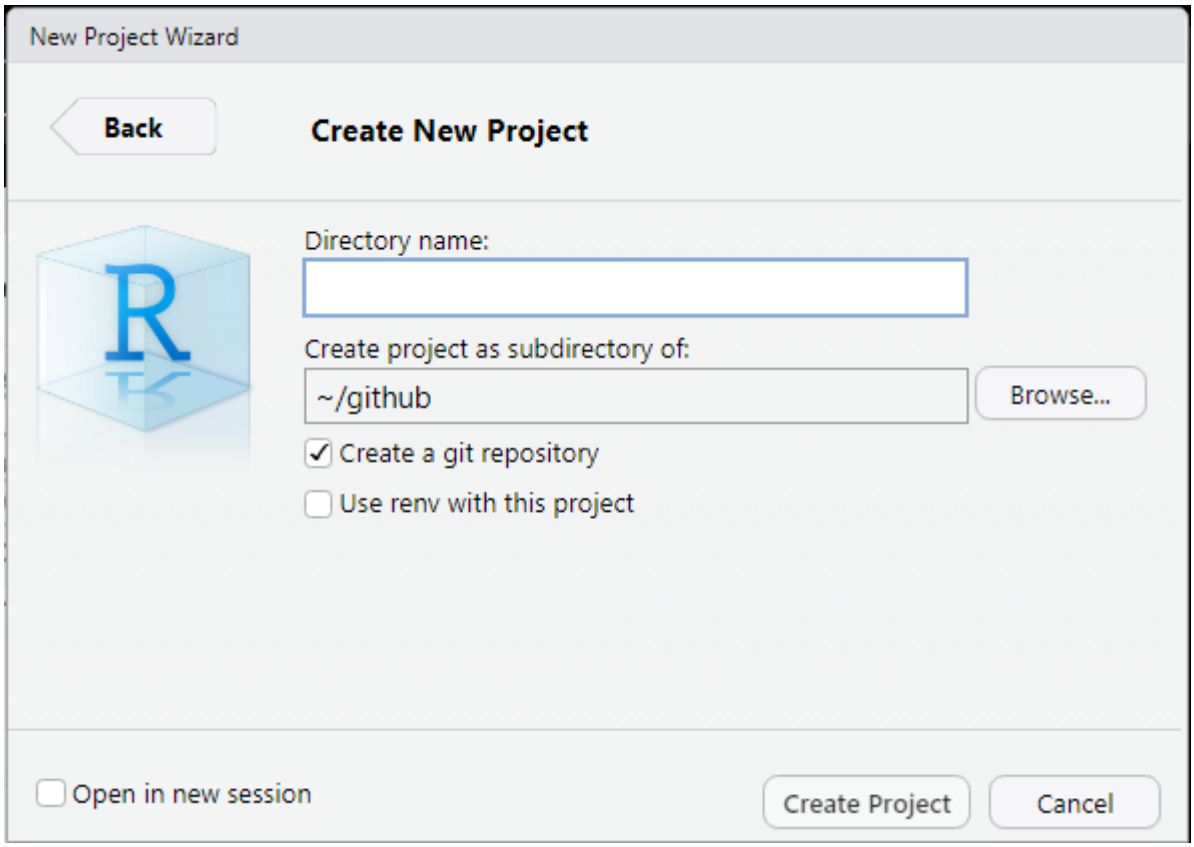

Figure 2.1: After installing Git, you should see 'Create a git repository'.

Second, **include your name** to make it easy to find you on *GitHub*.

R Studio works seamlessly with *Git* or *GitHub*, but it is helpful to use a *Git client* as it provides visual aids. There are choices for a *Git client* (see options here) but I will use *GitHub Desktop* (available here) for our exercise. Install GitHub Desktop onto you computer.

2.2 Commit & Push

2.2.1 Register Your Git repo

Open the `R Project` you've just created as a git repository. Let's make a sample `.R` file (`Ctr + Shift + N`) and save it (name as `sample.R`). For example:

```
## produce 100 random numbers that follows a normal distribution
x <- rnorm(100, mean = 0, sd = 1)

## estimate mean
mean(x)

## estimate SD
sd(x)
```

Open GitHub Desktop App. You will see the following GUI (Figure 2.2):

Hit `current repository` (top left) and `Add` > `Add existing repository` (Figure 2.3):

2.2.2 Commit

GitHub Desktop will prompt you to enter a local path to your *Git* repository. Browse and select your directory of the `R Project` - the local Git repository will show up in the list of repositories in *GitHub Desktop* (left side bar in Figure2.3). Now, you are ready to **Commit** your files to *Git*! Commit is the procedure to record your file change history in *Git*. To make this happen, click your *Git* repository on GitHub Desktop, and you will see the following:

At this stage, your file (`.R` or else) is saved onto your local directory, **but has not been recorded in *Git* as a change history.** To make a Commit, the first thing to do is to choose a file(s). There are checkboxes next to each of the new files. If this box is checked, you are going to commit changes to *Git*. Once you selected the files you want to make a commit, go to the bottom left of the window. There is a small box saying `summary (required)` or `Create sample.R`.This is the place where you can put any title that describes what you did in this commit, and **you cannot Commit unless you enter this information!** For example, I would write `initial commit` for this exercise - from the second commit, you should put a more informative commit message so you can track changes when needed. You can google recommendations for

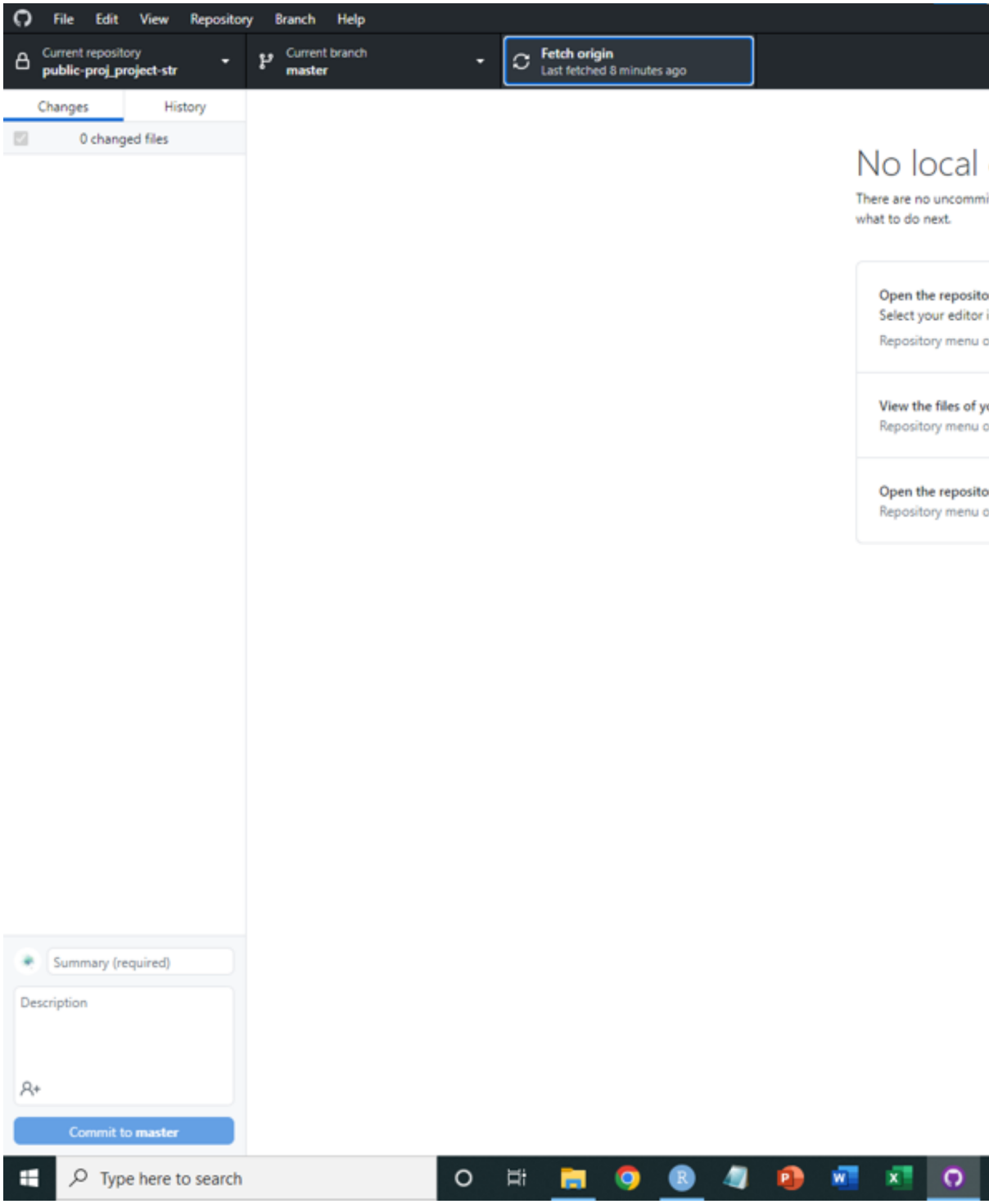

Figure 2.2: GUI for GitHub Desktop

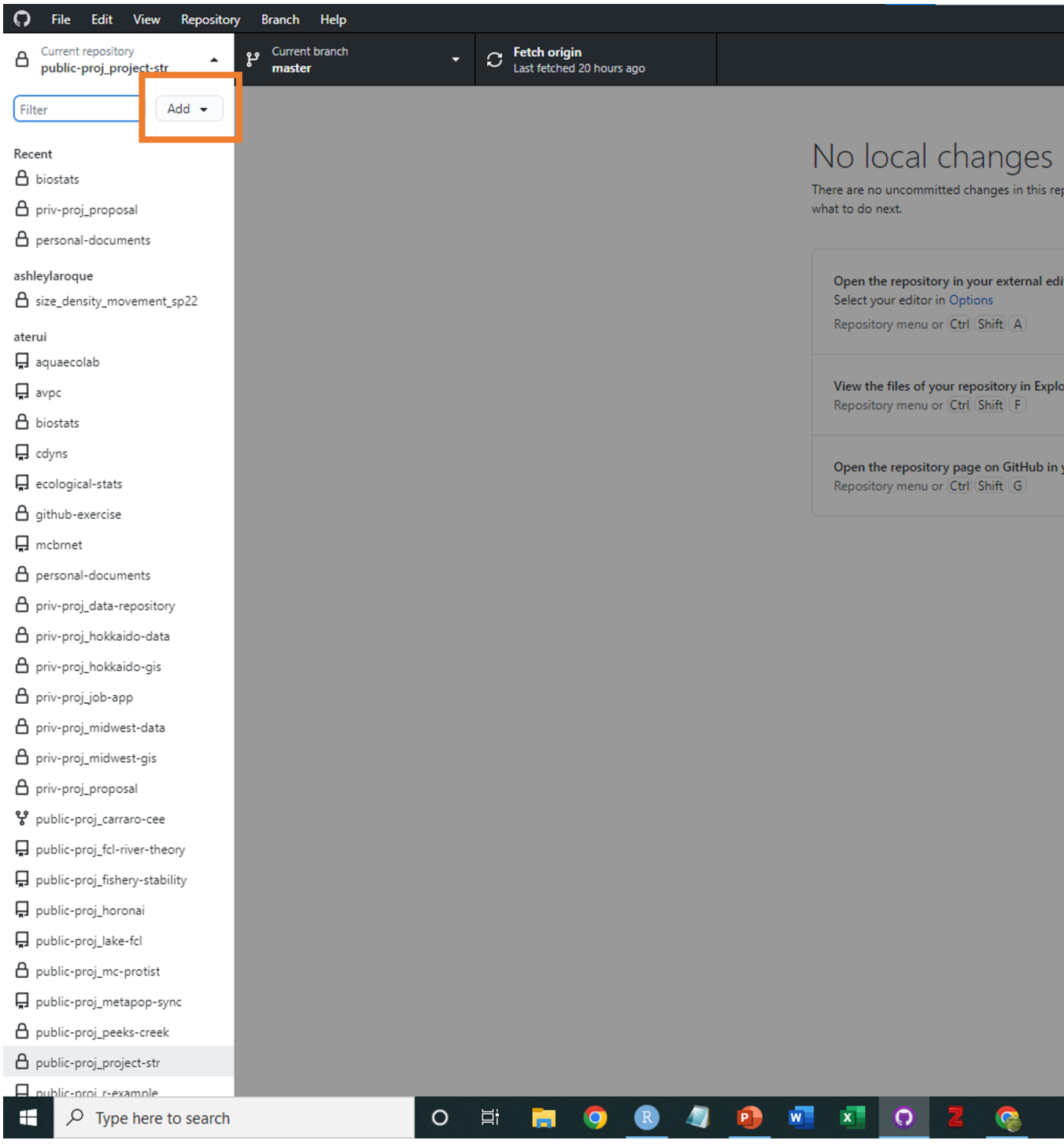

Figure 2.3: Add dropdown on the top left

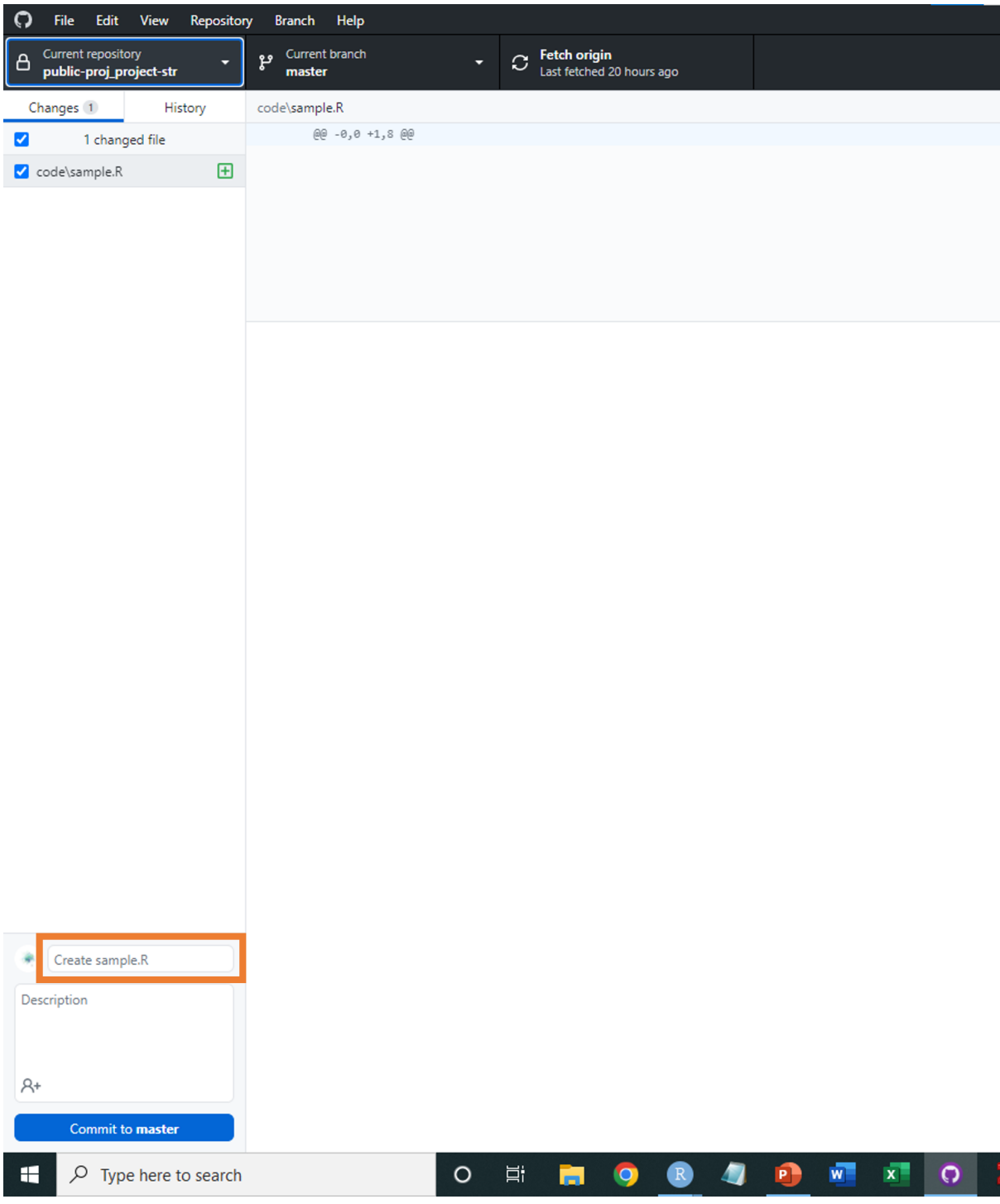

Figure 2.4: You will see ‘Create sample.R‘ or ‘summary (required)‘ on the bottom left

how to write commit titles/descriptions. Then hit `commit to master`. **Now, changes to the selected files have been recorded in *Git*!**

2.2.3 Push

Remember, your changes are recorded in your local computer **but not published in your online repository!** To send local changes to the online *GitHub* repository, you will need to **Push** commits via *GitHub Desktop*. Push is the procedure to send your Commit(s) to *GitHub*. Once you do a Commit, *GitHub Desktop* will ask you whether you want to Push it to an online repository (Figure 2.5). If this is the first push, there is no corresponding repository on *GitHub* tied to your local repository, so *GitHub Desktop* will ask you if you want to publish it on *GitHub* (NOTE: although it says 'publish', your repository will remain private unless you explicitly tell *Git Desktop* to make it public). If you are comfortable with the changes you made, **Push** it!

2.2.4 Edit

We went through how we get things uploaded onto *GitHub*, but what happens if we make changes to existing files? To see this, let's make a minor change to your R script. We have created a file `sample.R`:

```
## produce 100 random numbers that follows a normal distribution
x <- rnorm(100, mean = 0, sd = 1)

## estimate mean
mean(x)

## estimate SD
sd(x)
```

Edit this as follows:

```
## produce 100 random numbers that follows a normal distribution
x <- rnorm(100, mean = 0, sd = 1)

## estimate mean
median(x)

## estimate SD
var(x)
```

After making this change, go to *GitHub Desktop* again. *GitHub Desktop* automatically detects the difference between the new and old files and shows which part of the script has been edited! This helps coding quite a bit:

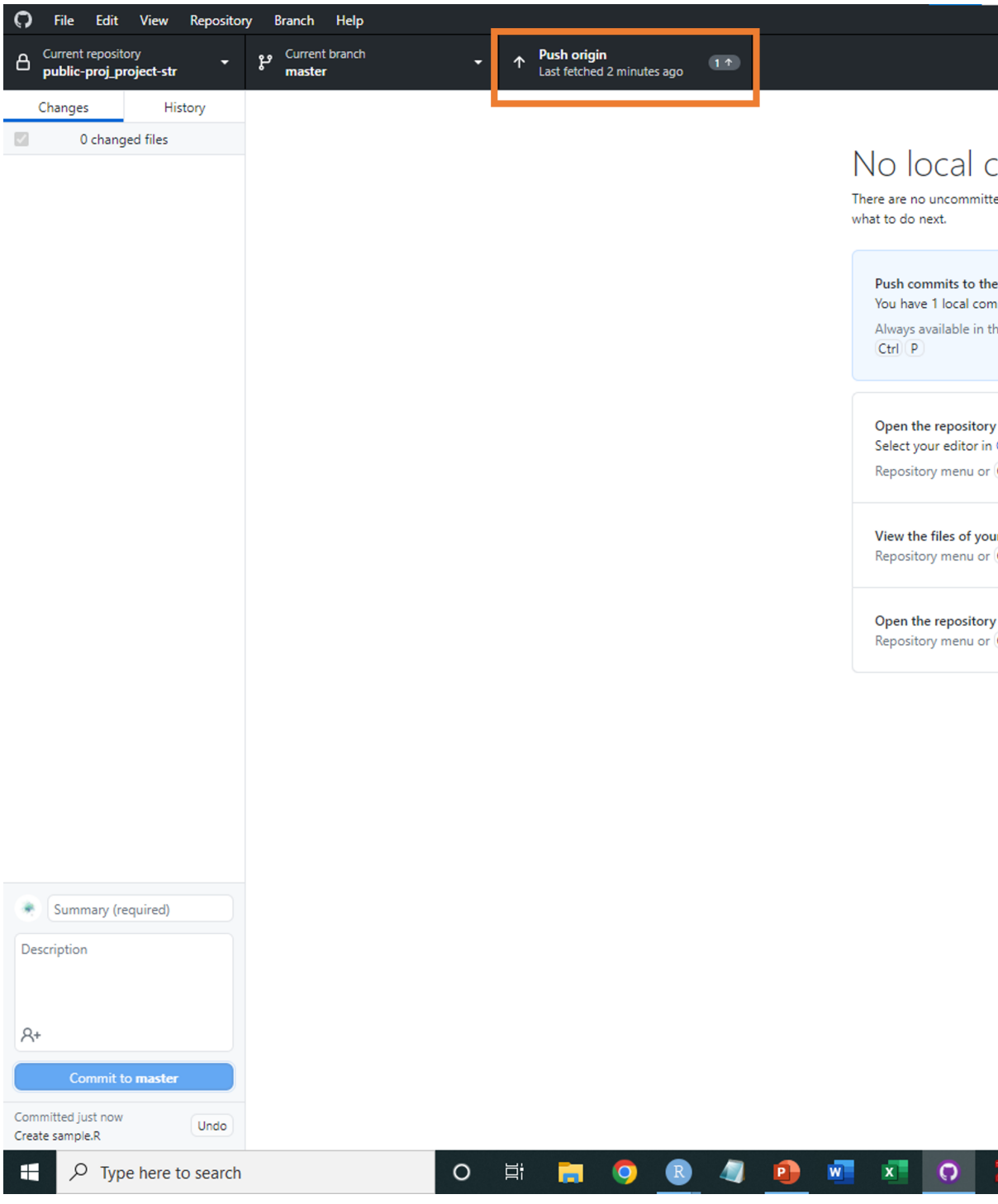

Figure 2.5: To Push your code, hit the highlighted menu button

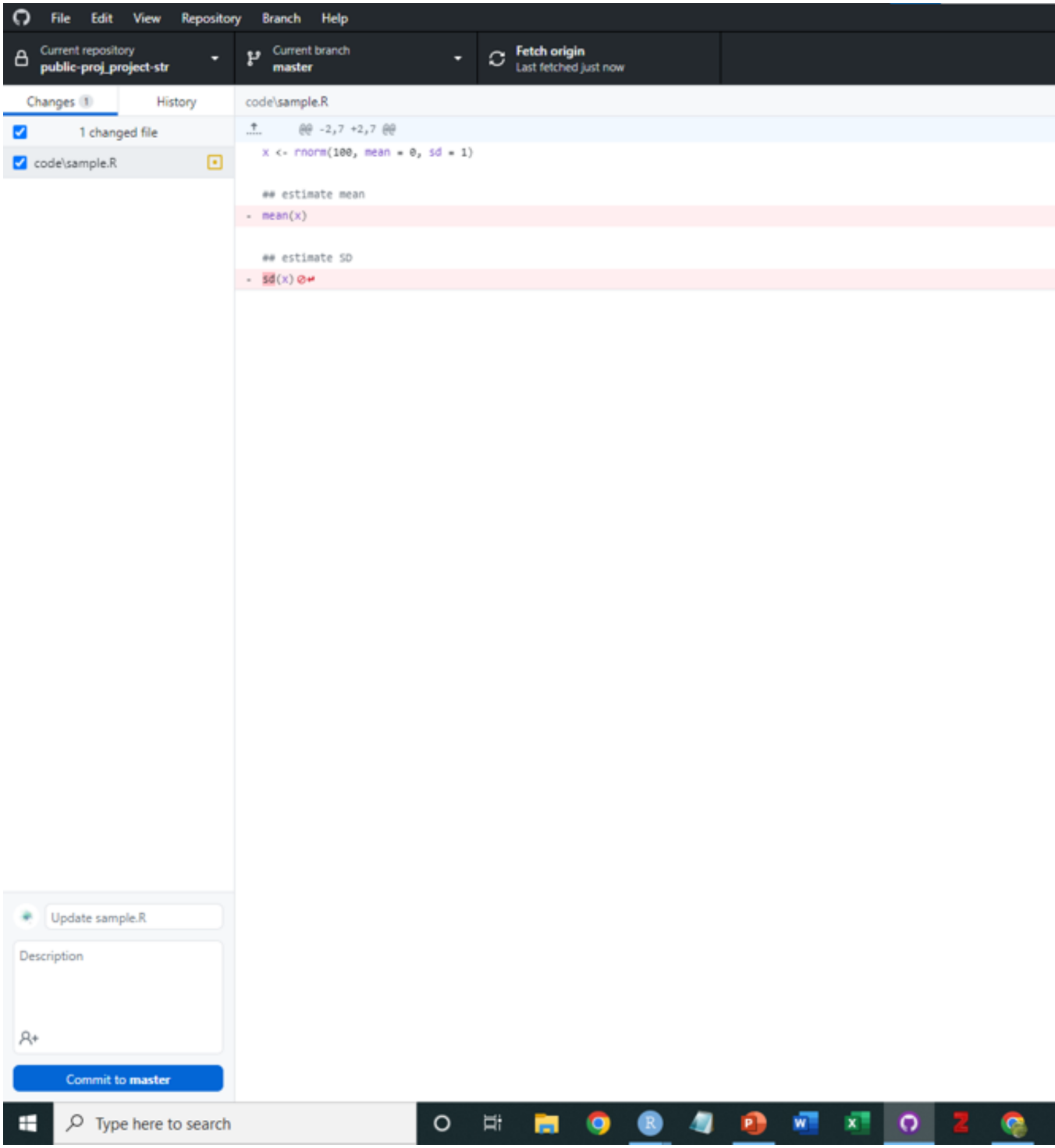

Figure 2.6: Git detects edits to your codes