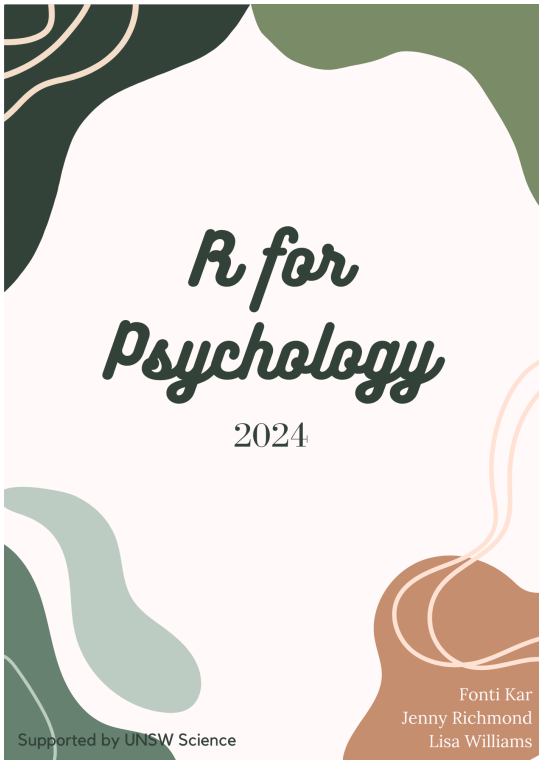
R for Psychology

2024-03-14

Table of contents

Welcome!



This book is written in mind for someone working in Psychology and is venturing into R with little to no experience. Research data in Psychology is unique in that it is collected in formats that are human-readable but not exactly R-readable. Data wrangling conventions will often vary depending on research question and therefore by what type data is collected. In "R for Psychology", we have compiled a series of real-world examples from different sub-disciplines and will walk through the process of wrangling, summary, analyses and visualisations

How to contribute

This book is built using Quarto and hosted via GitHub Pages. It is a living resource, we welcome any contributions from the Psychology community that would improve the quality of this book. There are many ways to contribute:

- Fixing grammar and typos
- Clarification or expanding on existing content
- Contribute real-world data in worked examples
- Authoring an entire chapter

Take a look at our Contributing Guide to see how you can help!

Acknowledgements

This book was created on the unceded territory of the Bedegal people who are the Traditional custodians of the lands where the Kensington campus is located.

The first version of this book was funded by the UNSW Research Infrastructure Scheme.

We would like to thank the following people who have contributed to the book:

@daxkellie

1 Introduction

1.1 What you will learn

How to:

- **Read** your hard-earned data into R
- **Wrangle and clean** the data in a R-friendly format
- **Produce** summary statistics
- **Analyse** your data
- **Visualise** your findings

Pre-analysis stages:

-

Analysis stages:

1.2 How book is organised

This book is organised by data type (e.g. Survey data, Questionnaire data).

Each chapter will walk through the process will work with real-world Psychology data and will walk you through reading in data, to cleaning and eventually analysis and visualising the results.

1.2.1 Conventions

We will refer to packages as `{dplyr}` and functions as `mean()`. Variables and objects (such as file names or data objects) as `age` and `mtcars`. Where it would aid understanding, we will sometimes refer to functions within a particular packages as `dplyr::mutute()`

1.3 Prerequisites

Content drawn from existing resources such as https://r4ds.hadley.nz/intro#prerequisites

1.3.1 R

Download Point to intro to R content (RUWithme, Environmental Computing, Software Carpentry)

1.3.2 RStudio

RStudio projects Point to resource about Rproj (SWC)

Running R code https://r4ds.hadley.nz/intro#running-r-code

1.3.3 Version control with git

1.3.3.1 What is git?

1.3.3.2 Why do I need git?

1.3.4 R packages

Every code section will always begin with calls to R packages. There will be code that is commented out (have **#** preceding the code) for you to install these if you don't have them on your computer

```
# install.packages(dplyr)

library(dplyr)
```

There are few R packages that will be on heavy rotation when it comes to working with Psychology data. #### {tidyverse}

{tidyverse} is a collection of R packages that is essential to a data scientist's toolkit. By installing `{tidyverse}` you are actually installing 8 other packages. The ones we will most often use include:

- {dplyr}
- {ggplot2}
- {tidyr}

The handy thing is, when you load the `{tidyverse}` library into R, it will load the core suite of packages for you instead of you loading each of them independently! Efficiency!! :rocket:

```
library(tidyverse)
```

Other packages that will be helpful for your R workflows include:

- {here}
- {janitor}

At the end of each chapter, we will also include our call to `sessionInfo()` so you can see what version of packages we are using.

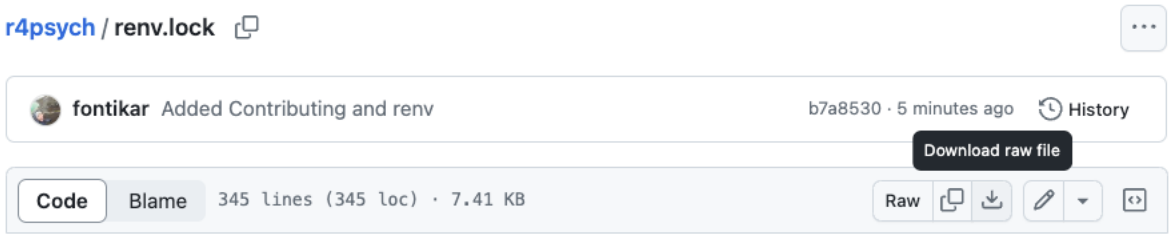
1.4 Virtual environments

Speaking on what package versions as we write this book, we understand the R package space is constantly changing. This means sometimes code will break due to package updates and this is all part of the process! To combat this problem, we've enlisted `renv` to create a reproducible environment for building this book.

1.4.1 Download our virtual environment

The virtual environment used to build this book is stored in a `lockfile`. You can find this file in the GitHub repository where the source code of this book lives.

The lockfile is named `renv.lock`. You can download this file directly but clicking on the file name and clicking on the "Download raw file" button.



Alternatively, you can clone our repository into your computer. Learn more about cloning repositorsies and other GitHub workflows in Happy Git by Jenny Bryan.

Once you have this file downloaded, move it in a relevant project directory and then we can let `{renv}` work its magic.

1.4.2 Install

First things first, lets install `renv` if we don't have it already.

```r
install.packages("renv")

library(renv)
```

1.4.3 Recreate virtual enviroment

Now let's tell `renv` where our downloaded `renv.lock` file is. Specific the path to the file in the function `restore()` and you are good to go!

```r
restore(lockfile = "path_to_renv.lock")
```

2 Organise

There are lots of reasons to use R and R Studio for your data analysis, but reproducibility ranks high on the list. By writing code that reads, wrangles, cleans, visualises, and analyses your data, you are documenting the process that your data has been through from its raw state through to its analysed state. Reproducibility is all about someone else (or you in the future) being able to take your data and the code you have written and use it to produce exactly the same analysis values that you report in your paper or thesis.

With this goal in mind, there are a few best practices we recommend, to ensure that your code is usable in the future, irrespective of what machine it is being run on and by whom.

2.1 R projects

R really cares about where things live on your computer, even if you don't. Humans have gotten out of the habit of thinking very hard about where they put things on their machine - the search capabilities on the modern computer are quite good and you can generally find files quite easily by searching for them.

When you are coding in R, however, you need to explicitly tell R where to find things. You can make this process much easier for yourself by always working within an RStudio Project.

When you work within an RStudio project, you can reference everything in relation to the top level of that project folder. It doesn't matter where that project folder lives on your machine (i.e. Desktop, Documents, OneDrive, Dropbox) the code you write is always relative to that folder. This means that you share that whole project folder with someone else (your collaborators or supervisor), and your code won't break.

> When you start a new analysis project, create a New RStudio Project via the File tab.

Always open RStudio by double clicking on the .RProj file in the folder on your machine. There is an icon in the top right corner of RStudio IDE that shows you which project you are working in and makes it easy to switch between projects.

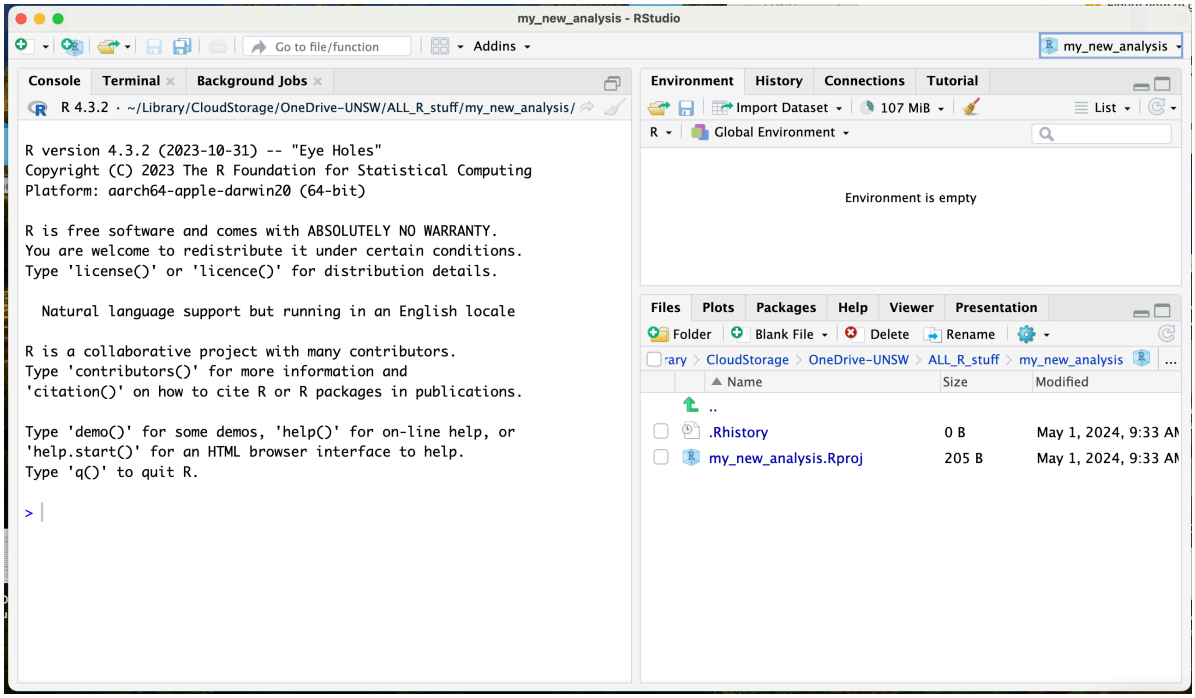


Figure 2.1: Working in an RStudio Project

2.2 where is here?

Once you have set up a project to contain your analysis, you can avoid further file path drama by using the `here()` package. This package makes it super easy to refer to file paths and ensures that your code will be reproducible by someone else.

Once you have installed the `here()` package, use it to tell R where you find your data like this.

```
library(here)
library(readr)

mydata <- read_csv(here("data", "file.csv"))
```

By referring to the location of your data using the `here()` package, here is no need worry about working directories, and you can be sure that your code will work on any machine.

> To read more about why projects and the `here()` package are useful, check out this blog by Jenny Bryan

2.3 folder structure

Once you have your project set up, you might like to think about imposing some structure on it. It is mostly personal preference, but many analysis projects include the following folders.

- data
  - raw-data
  - clean-data
- output
  - figures
  - tables
- manuscript

You always want to keep your raw data untouched and separate from any data that you write back to your machine after your data cleaning process separate, so a raw-data subfolder can be useful.

In addition, you might want to organise your figures and tables into an output folder and put any writing that you are doing in the manuscripts folder.

Usually the scripts (or R Markdown) documents that you write your code in, live in the top level of your project file. In RStudio, your project structure might look something like this.

2.4 naming things

When naming things in your analysis folder, it is a good idea to think about your future self. In all likelihood, when you come back to this analysis in a few months time, you will have no recollection how it actually worked, but you can leave yourself some breadcrumbs by being a bit intentional about naming things.

Your file names should be meaningful and orderly. The name of the file should tell the new user (or future you) what is in the file and where it goes in the process.

- cleaning-functions.R
- 1_wrangle.Rmd
- 2_visualise.Rmd
- 3_analyse.Rmd

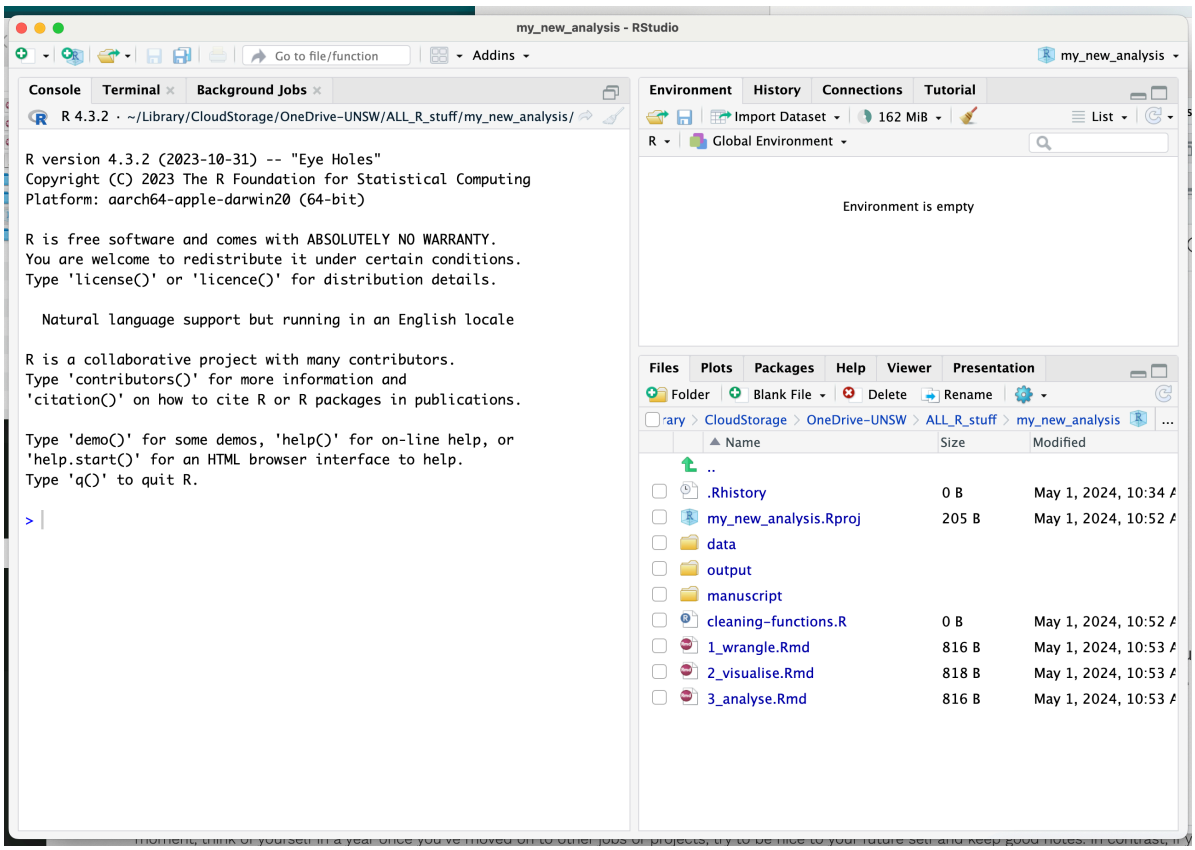


Figure 2.2: A project structure template

In this project, I can tell by glancing at the file names that I have a script (.R) that contains functions and three R Markdown files that contain each stage of the analysis.

Sticking with lower case is a good idea; avoid special characters and use - or _ to fill any gaps.

Find more useful naming tips in the Tidyverse Style guide.

2.5 documenting things

2.5.1 README

In addition to the breadcrumbs that our file names leave, it is also a good idea to leave explicit notes to your future self (or someone else) in a README.md file. This is a simple text file that contains instructions for how the user should engage with your project.

Create a new Markdown file and save it as README.md

Use it to leave yourself instructions that look a bit like this.

2.5.2 R Markdown

In addition to leaving your future self explicit notes about how to engage with the project generally in a README document, it is also best practice to document your code in a way that makes it really clear what the code is doing and why. For this reason, we recommend using R Markdown documents (rather than R scripts) for your analysis.

R Markdown is a handy file format that allows you to intersperse chunks of code with notes. This kind of document makes it easy to writing explanations, interpretations, thoughts for your future self as you code. This kind of documentation makes it much more likely to be able to make sense of what you have done and why, when you come back to your analysis in a few months time.

R Markdown documents can also be "knitted" into html, pdf, or word documents, allowing you to share your analysis (and associated thoughts) with collaborators, even if they don't know R.

For a quick get up to speed on R Markdown and how to use it, check out RLadiesSydney #RYouWithMe MarkyMark module

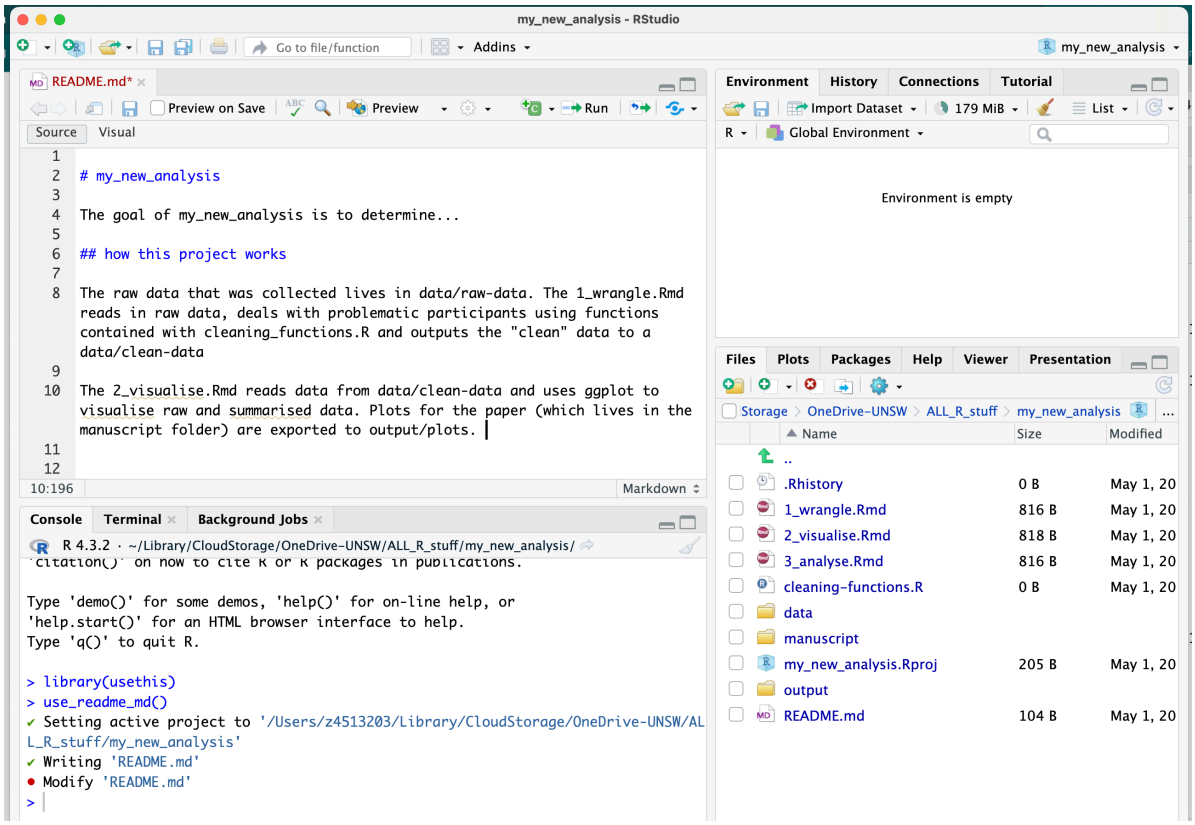


Figure 2.3: An example README file

3 Import

3.1 Reading in Excel spreadsheets

This is

3.1.1 Reading in .csv

3.2 Reading in SPSS

3.3 Reading in Qualtrics data

3.3.1 Reading in .sav

4 Wrangle

4.1 Clean names

4.2 Dealing with labels

4.3 Exclusions

`mutate` `case_when`

`filter`

4.4 Creating scales and indexes

`group_by` `summarise`

`row_wise` `mutate`

4.4.1 Checking reliabilty

5 Describe

6 Plot

7 Making Inferences

Take it away Lisa!

```
1 + 1
```

```
[1] 2
```

8 Contributing

This resource is created in mind so that the community that uses it, can also contribute to it. We hope this mindset will encourage the resource to grow and stay up-to-date for R learners.

Importantly, **all skill levels** are welcome to contribute, even if you think your skills are not up to scratch - this is what this guide is for!

8.0.1 GitHub Workflow

First let's talk about the GitHub part of the workflow.

The content of our book lives publicly in a repository within the UNSW GitHub Organisation.

We will work in branches so as to not overwrite each other's work, and let GitHub do what it does best.

The main branch will be the current approved version of the book. The main branch is what displays at https://nasa-openscapes.github.io/earthdata-cloud-cookbook.

A nice clean workflow with branches is to consider them temporary. You pull the most recent from main, you create a branch locally, you make your edits, you commit regularly, you push regularly to github.com, and then you create a pull request for it to be merged into main, and when it's approved the branch is deleted on github.com and you also delete it locally. That's the workflow we'll walk through here. A great resource on GitHub setup and collaboration is Happy Git with R, which includes fantastic background philosophy as well as bash commands for setup, workflows, and collaboration.

The following assumes you've completed the initial setup from the previous chapter.

8.0.2 Hello Quarto!

This book is built by Quarto which is an open source, cross-language publishing system that allows users to build beautiful things from blogs, to websites and books!

You can learn more about the capabilities of Quarto in this talk by Mine Çetinkaya-Rundel & Julia Stewart Lowndes at posit::conf(2023)

8.0.2.1 Install Quarto

Let's first make sure we the latest version of Quarto installed.

8.0.3 Quarto Workflow

8.1 Book Structure

8.2 Book Practices and Conventions

8.3 Book Contributions

8.3.1 Edits

8.3.2 Additions

8.4 Submit a pull request

9 Data

10 Appendix

11 how to install packages

You only need to install a package once on your machine. Once the package is installed, you will need to use `library()` to load the functions in it every time you want to use it, but the installation is a one time job. So you can either do it in the console, or using the packages tab.

11.1 option 1

Install a package by typing the following command with the name of the package you would like to install in the console.

```
install.packages("packagename")
```

11.2 option 2

Alternatively, search for the package you would like to install in the packages tab.

> Remember once you have installed a package, you will need to use the `library()` function to load it before it will work.

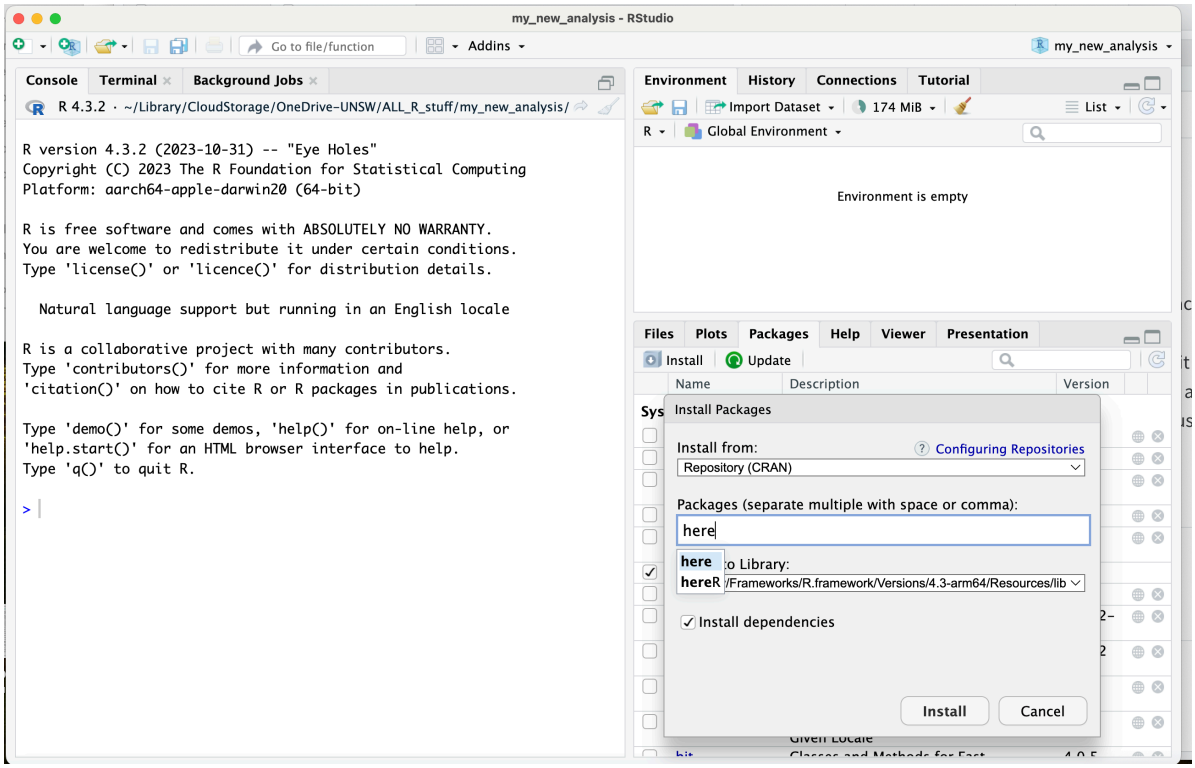


Figure 11.1: You can search for packages and install them from CRAN via the packages tab

References