RPC 519 R and Bioconductor

Lori Kern

2025-08-27

Table of contents

Preface

This is a selection of material from **The RBioc Book** created by Sean Davis. The original full content may be viewed here. The contents of this book may have minor modifications or additions.

The material is modified and redistributed in accordance with the original Licensing.

Select modifications were inspired by RPC 520 content originally distributed by Martin Morgan.

To Learn more about creating Quarto books visit https://quarto.org/docs/books.

Part I

Introduction

1 About R

In this chapter, we will discuss the basics of R and RStudio, two essential tools in genomics data analysis. We will cover the advantages of using R and RStudio, how to set up RStudio, and the different panels of the RStudio interface.

1.1 What is R?

R is a programming language and software environment designed for statistical computing and graphics. It is widely used by statisticians, data scientists, and researchers for data analysis and visualization. R is an open-source language, which means it is free to use, modify, and distribute. Over the years, R has become particularly popular in the fields of genomics and bioinformatics, owing to its extensive libraries and powerful data manipulation capabilities.

The R language is a dialect of the S language, which was developed in the 1970s at Bell Laboratories. The first version of R was written by Robert Gentleman and Ross Ihaka and released in 1995 (see this slide deck for Ross Ihaka's take on R's history). Since then, R has been continuously developed by the R Core Team, a group of statisticians and computer scientists. The R Core Team releases a new version of R every year.

1.2 Why use R?

There are several reasons why R is a popular choice for data analysis, particularly in genomics and bioinformatics. These include:

1. **Open-source**: R is free to use and has a large community of developers who contribute to its growth and development. What is "open-source"?
2. **Extensive libraries**: There are thousands of R packages available for a wide range of tasks, including specialized packages for genomics and bioinformatics. These libraries have been extensively tested and ara available for free.
3. **Data manipulation**: R has powerful data manipulation capabilities, making it easy (or at least possible) to clean, process, and analyze large datasets.
4. **Graphics and visualization**: R has excellent tools for creating high-quality graphics and visualizations that can be customized to meet the specific needs of your analysis. In most cases, graphics produced by R are publication-quality.

5. **Reproducible research**: R enables you to create reproducible research by recording your analysis in a script, which can be easily shared and executed by others. In addition, R does not have a meaningful graphical user interface (GUI), which renders analysis in R much more reproducible than tools that rely on GUI interactions.
6. **Cross-platform**: R runs on Windows, Mac, and Linux (as well as more obscure systems).
7. **Interoperability with other languages**: R can interfact with FORTRAN, C, and many other languages.
8. **Scalability**: R is useful for small and large projects.

I can develop code for analysis on my Mac laptop. I can then install the *same* code on our 20k core cluster and run it in parallel on 100 samples, monitor the process, and then update a database (for example) with R when complete. In other words, R is a powerful tool that can be used for a wide range of tasks, from small-scale data analysis to large-scale genomics and omics data science projects.

1.3 Why not use R?

- R cannot do everything.
- R is not always the "best" tool for the job.
- R will *not* hold your hand. Often, it will *slap* your hand instead.
- The documentation can be opaque (but there is documentation).
- R can drive you crazy (on a good day) or age you prematurely (on a bad one).
- Finding the right package to do the job you want to do can be challenging; worse, some contributed packages are unreliable.]{}
- R does not have a meaningfully useful graphical user interface (GUI).
- Additional languages are becoming increasingly popular for bioinformatics and biological data science, such as Python, Julia, and Rust.

1.4 R License and the Open Source Ideal

R is free (yes, totally free!) and distributed under GNU license. In particular, this license allows one to:

- Download the source code
- Modify the source code to your heart's content
- Distribute the modified source code and even charge money for it, but you must distribute the modified source code under the original GNU license.

This license means that R will always be available, will always be open source, and can grow organically without constraint.

1.5 Working with R

R is a programming language, and as such, it requires you to write code to perform tasks. This can be intimidating for beginners, but it is also what makes R so powerful. In R, you can write scripts to automate tasks, create functions to encapsulate complex operations, and use packages to extend the functionality of R.

R can be used interactively or as a scripting language. In interactive mode, you can enter commands directly into the R console and see the results immediately. In scripting mode, you can write a series of commands in a script file and then execute the entire script at once. This allows you to save your work, reuse code, and share your analysis with others.

In the next section, we will discuss how to set up RStudio, an integrated development environment (IDE) for R that makes it easier to write and execute R code. However, you can use R without RStudio if you prefer to work in the R console or another IDE. RStudio is not required to use R, but it does provide a more user-friendly interface and several useful features that can enhance your R programming experience.

2 RStudio

RStudio is an integrated development environment (IDE) for R. It provides a graphical user interface (GUI) for R, making it easier to write and execute R code. RStudio also provides several other useful features, including a built-in console, syntax-highlighting editor, and tools for plotting, history, debugging, workspace management, and workspace viewing. RStudio is available in both free and commercial editions; the commercial edition provides some additional features, including support for multiple sessions and enhanced debugging.

2.1 Getting started with RStudio

To get started with RStudio, you first need to install both R and RStudio on your computer. Follow these steps:

1. Download and install R from the official R website.
2. Download and install RStudio from the official RStudio website.
3. Launch RStudio. You should see the RStudio interface with four panels.

> **i** R versions
>
> RStudio works with all versions of R, but it is recommended to use the latest version of R to take advantage of the latest features and improvements. You can check your R version by running `version` (no parentheses)in the R console.
> You can check the latest version of R on the R-project website.

2.2 The RStudio Interface

RStudio's interface consists of four panels (see Figure 2.1):

- **Console** This panel displays the R console, where you can enter and execute R commands directly. The console also shows the output of your code, error messages, and other information.
- **Source** This panel is where you write and edit your R scripts. You can create new scripts, open existing ones, and run your code from this panel.

- **Environment** This panel displays your current workspace, including all variables, data objects, and functions that you have created or loaded in your R session.
- **Plots, Packages, Help, and Viewer** These panels display plots, installed packages, help files, and web content, respectively.

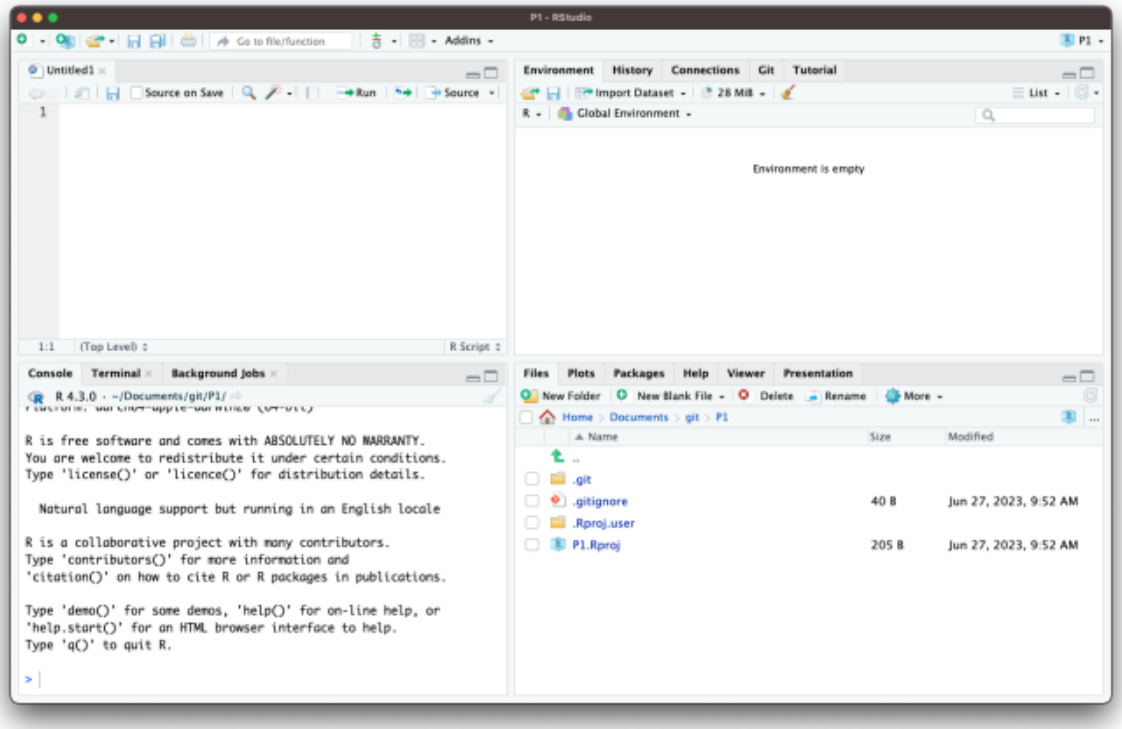


Figure 2.1: The RStudio interface. In this layout, the **source** pane is in the upper left, the **console** is in the lower left, the **environment** panel is in the top right and the **viewer/help/files** panel is in the bottom right.

> **i** Do I need to use RStudio?
>
> No. You can use R without RStudio. However, RStudio makes it easier to write and execute R code, and it provides several useful features that are not available in the basic R console. Note that the only part of RStudio that is actually interacting with R directly is the console. The other panels are simply providing a GUI that enhances the user experience.

Customizing the RStudio Interface

You can customize the layout of RStudio to suit your preferences. To do so, go to **Tools > Global Options > Appearance**. Here, you can change the theme, font size, and panel layout. You can also resize the panels as needed to gain screen real estate (see Figure 2.2).

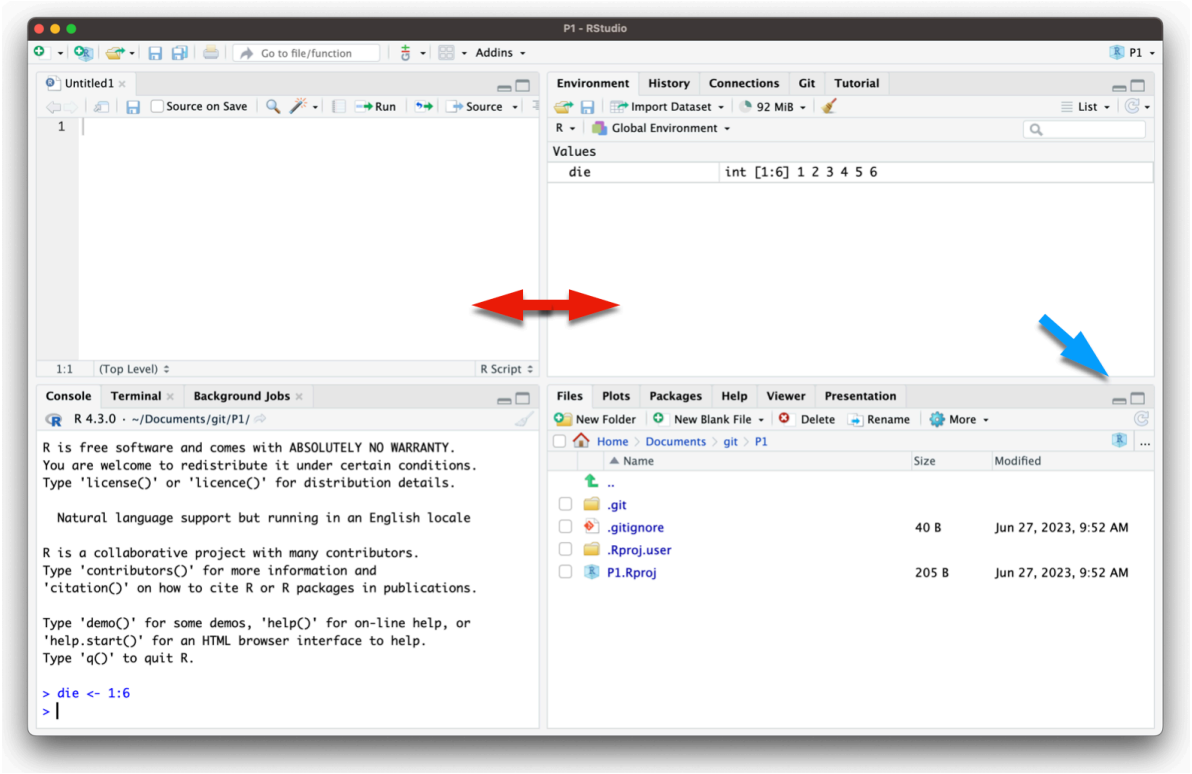


Figure 2.2: Dealing with limited screen real estate can be a challenge, particularly when you want to open another window to, for example, view a web page. You can resize the panes by sliding the center divider (red arrows) or by clicking on the minimize/maximize buttons (see blue arrow).

In summary, R and RStudio are powerful tools for genomics data analysis. By understanding the advantages of using R and RStudio and familiarizing yourself with the RStudio interface, you can efficiently analyze and visualize your data. In the following chapters, we will delve deeper into the functionality of R, Bioconductor, and various statistical methods to help you gain a comprehensive understanding of genomics data analysis.

2.3 Alternatives to RStudio

While RStudio is a popular choice for R development, there are several alternatives you can consider:

1. **Jupyter Notebooks**: Jupyter Notebooks provide an interactive environment for writing and executing R code, along with rich text support for documentation. You can use the IRKernel to run R code in Jupyter.

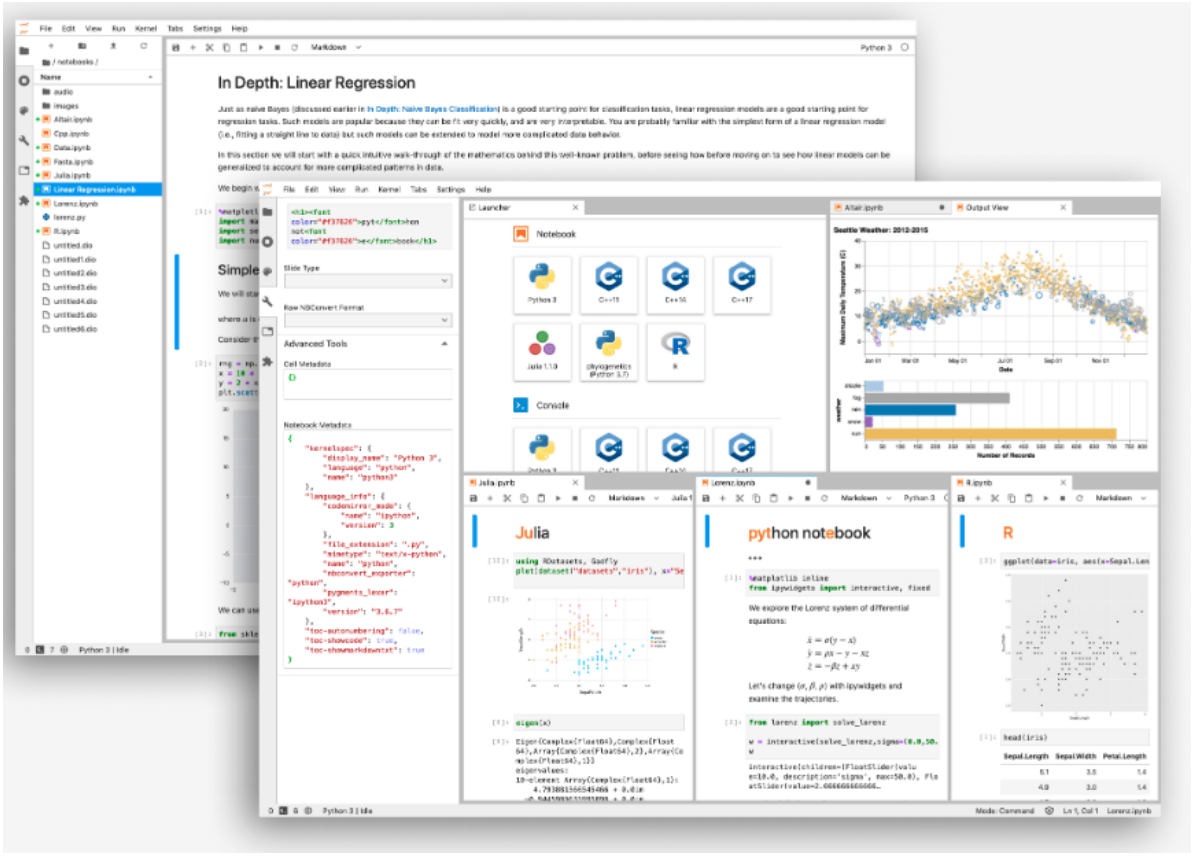


Figure 2.3: Jupyter Notebook interface. This is an interactive environment for writing and executing R code, along with rich text support for documentation.

2. **Visual Studio Code**: With the R extension for Visual Studio Code, you can write and execute R code in a lightweight editor. This setup provides features like syntax highlighting, code completion, and integrated terminal support.

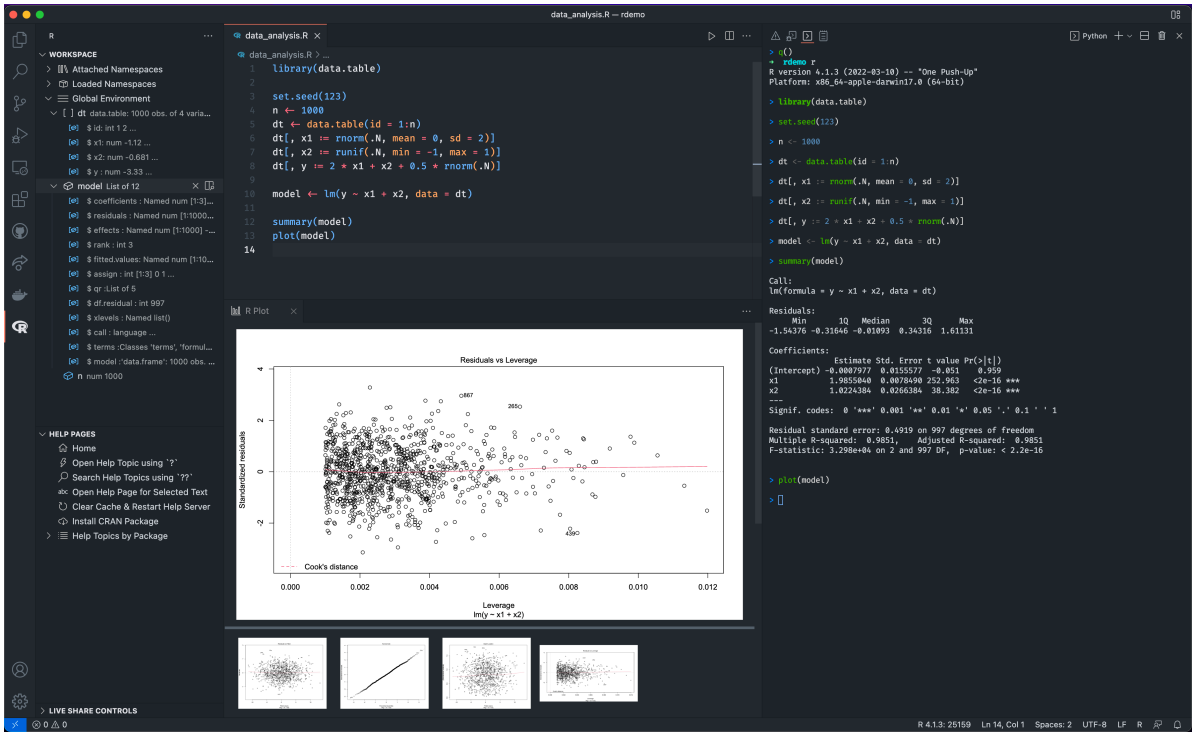


Figure 2.4: Visual Studio Code (VSCode) with the R extension. This is a lightweight alternative to RStudio that provides syntax highlighting, code completion, and integrated terminal support.

3. Positron Workbench: This is a commercial IDE that supports R and Python. It provides a similar interface to RStudio but with additional features for data science workflows, including support for multiple languages and cloud integration.

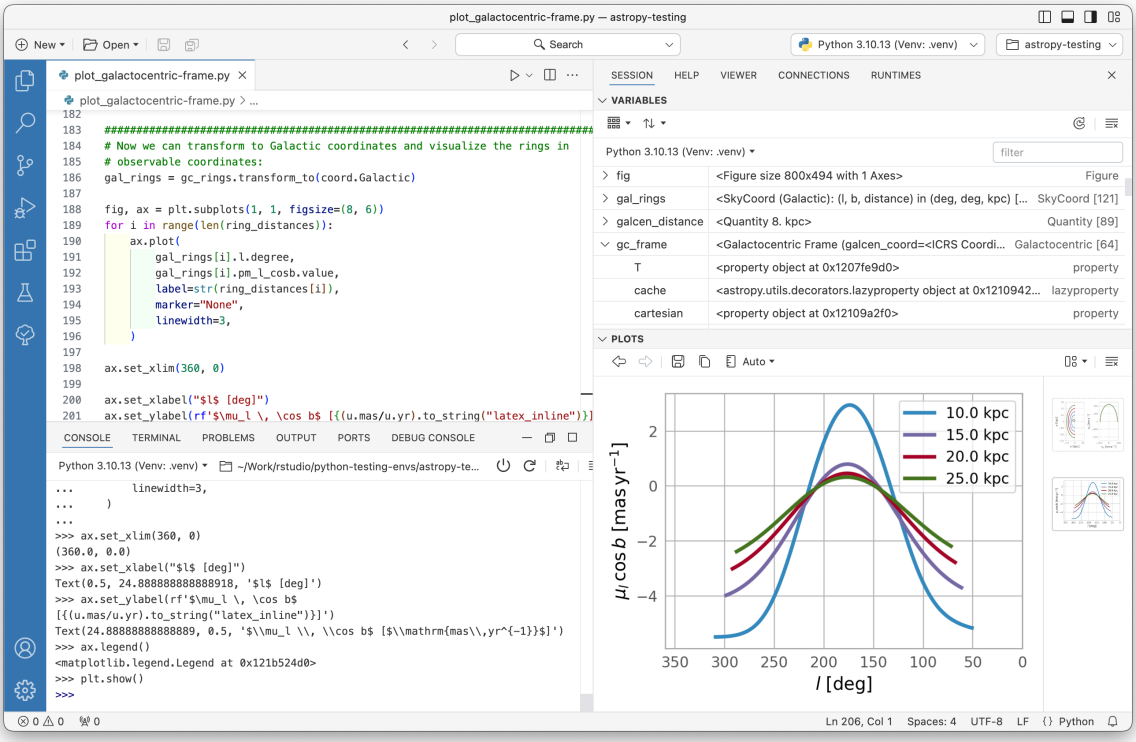


Figure 2.5: Positron Workbench interface. This IDE supports R and Python, providing a similar interface to RStudio with additional features for data science workflows.

4. **Command Line R**: For those who prefer a minimalistic approach, you can use R directly from the command line. This method lacks the GUI features of RStudio but can be efficient for quick tasks, scripting, automation, or when working on remote servers.

Each of these alternatives has its own strengths and weaknesses, so you may want to try a few to see which one best fits your workflow. All are available for free, and you can install them alongside RStudio if you wish to use multiple environments. Each can be installed in Windows, Mac, and Linux.

3 R mechanics

3.1 Starting R

We've installed R and RStudio. Now, let's start R and get going. How to start R depends a bit on the operating system (Mac, Windows, Linux) and interface. In this course, we will largely be using an Integrated Development Environment (IDE) called *RStudio*, but there is nothing to prohibit using R at the command line or in some other interface (and there are a few).

3.2 *RStudio*: A Quick Tour

The RStudio interface has multiple panes. All of these panes are simply for convenience except the "Console" panel, typically in the lower left corner (by default). The console pane contains the running R interface. If you choose to run R outside RStudio, the interaction will be *identical* to working in the console pane. This is useful to keep in mind as some environments, such as a computer cluster, encourage using R without RStudio.

- Panes
- Options
- Help
- Environment, History, and Files

3.3 Interacting with R

The only meaningful way of interacting with R is by typing into the R console. At the most basic level, anything that we type at the command line will fall into one of two categories:

1. Assignments

```
x = 1
y <- 2
```

2. Expressions

```
1 + pi + sin(42)
```

```
[1] 3.225071
```

The assignment type is obvious because either the The `<-` or `=` are used. Note that when we type expressions, R will return a result. In this case, the result of R evaluating `1 + pi + sin(42)` is 3.2250711.

The standard R prompt is a ">" sign. When present, R is waiting for the next expression or assignment. If a line is not a complete R command, R will continue the next line with a "+". For example, typing the following with a "Return" after the second "+" will result in R giving back a "+" on the next line, a prompt to keep typing.

```
1 + pi +
sin(3.7)
```

```
[1] 3.611757
```

R can be used as a glorified calculator by using R expressions. Mathematical operations include:

- Addition: `+`
- Subtraction: `-`
- Multiplication: `*`
- Division: `/`
- Exponentiation: `^`
- Modulo: `%%`

The `^` operator raises the number to its left to the power of the number to its right: for example `3^2` is `9`. The modulo returns the remainder of the division of the number to the left by the number on its right, for example 5 modulo 3 or `5 %% 3` is 2.

3.3.1 Expressions

```
5 + 2
28 %% 3
3^2
5 + 4 * 4 + 4 ^ 4 / 10
```

Note that R follows order-of-operations and groupings based on parentheses.

```
5 + 4 / 9
(5 + 4) / 9
```

3.3.2 Assignment

While using R as a calculator is interesting, to do useful and interesting things, we need to assign *values* to *objects*. To create objects, we need to give it a name followed by the assignment operator `<-` (or, entirely equivalently, `=`) and the value we want to give it:

```
weight_kg <- 55
```

`<-` is the assignment operator. Assigns values on the right to objects on the left, it is like an arrow that points from the value to the object. Using an `=` is equivalent (in nearly all cases). Learn to use `<-` as it is good programming practice.

> **What about `<-` and `=` for assignment?**
>
> The `<-` and `=` both work fine for assignment. You'll see both used and it is up to you to choose a standard for yourself. However, some programming communities, such as Bioconductor, will strongly suggest using the `<-` as it is clearer that it represents an *assignment* operation.

Objects can be given any name such as `x`, `current_temperature`, or `subject_id` (see below). You want your object names to be explicit and not too long. They cannot start with a number (`2x` is not valid but `x2` is). R is case sensitive (e.g., `weight_kg` is different from `Weight_kg`). There are some names that cannot be used because they represent the names of fundamental functions in R (e.g., `if`, `else`, `for`, see here for a complete list). In general, even if it's allowed, it's best to not use other function names, which we'll get into shortly (e.g., `c`, `T`, `mean`, `data`, `df`, `weights`). When in doubt, check the help to see if the name is already in use. It's also best to avoid dots (`.`) within a variable name as in `my.dataset`. It is also recommended to use nouns for variable names, and verbs for function names.

When assigning a value to an object, R does not print anything. You can force to print the value by typing the name:

```
weight_kg
```

```
[1] 55
```

Now that R has `weight_kg` in memory, which R refers to as the "global environment", we can do arithmetic with it. For instance, we may want to convert this weight in pounds (weight in pounds is 2.2 times the weight in kg).

```
2.2 * weight_kg
```

```
[1] 121
```

We can also change a variable's value by assigning it a new one:

```
weight_kg <- 57.5
2.2 * weight_kg
```

```
[1] 126.5
```

This means that assigning a value to one variable does not change the values of other variables. For example, let's store the animal's weight in pounds in a variable.

```
weight_lb <- 2.2 * weight_kg
```

and then change `weight_kg` to 100.

```
weight_kg <- 100
```

What do you think is the current content of the object `weight_lb`, 126.5 or 220?

You can see what objects (variables) are stored by viewing the Environment tab in Rstudio. You can also use the `ls()` function. You can remove objects (variables) with the `rm()` function. You can do this one at a time or remove several objects at once. You can also use the little broom button in your environment pane to remove everything from your environment.

```
ls()
rm(weight_lb, weight_kg)
ls()
```

What happens when you type the following, now?

```
weight_lb # oops! you should get an error because weight_lb no longer exists!
```

3.4 Rules for Names in R

R allows users to assign names to objects such as variables, functions, and even dimensions of data. However, these names must follow a few rules.

- Names may contain any combination of letters, numbers, underscore, and "."
- Names may not start with numbers, underscore.
- R names are case-sensitive.

Examples of valid R names include:

```
pi
x
camelCaps
my_stuff
MY_Stuff
this.is.the.name.of.the.man
ABC123
abc1234asdf
.hi
```

3.5 About R functions

When you see a name followed by parentheses `()`, you are likely looking a name that represents an R function (or method, but we'll sidestep that distinction for now). Examples of R functions include `print()`, `help()`, and `ls()`. We haven't seen examples yet, but when a name is followed by `[]`, that name represents a variable of some kind and the `[]` are used for "subsetting" the variable. So:

- Name followed by `()` is a function.
- Name with `[]` means a variable that is being subset.

In many cases, when you see a new function used, you may not know what it does. The R `help()` function takes the name of another function and gives back the R help document for that function if there is one. The next section reviews that technique.

3.6 Resources for Getting Help

There is extensive built-in help and documentation within R. A separate page contains a collection of additional resources.

If the name of the function or object on which help is sought is known, the following approaches with the name of the function or object will be helpful. For a concrete example, examine the help for the `print` method.

```
help(print)
help('print')
?print
```

There are also tons of online resources that Google will include in searches if online searching feels more appropriate.

I strongly recommend using `help("newfunction")` for all functions that are new or unfamiliar to you.

There are also many open and free resources and reference guides for R.

- Quick-R: a quick online reference for data input, basic statistics and plots
- R reference card PDF by Tom Short
- Rstudio cheatsheets

3.7 Reflection

- Can you recognize the difference between *assignment* and *expressions* when interacting with R?
- Can you demonstrate an assignment to a variable?
- Do you know the rules for "names" in R?
- Are you able to get help using the R `help()` function?
- Do you know that functions are recognizable as names followed by `()`?

References

A Appendix

A.1 Swirl

The following is from the swirl website.

> The swirl R package makes it fun and easy to learn R programming and data science. If you are new to R, have no fear.

To get started, we need to install a new package into R.

```
install.packages('swirl')
```

Once installed, we want to load it into the R workspace so we can use it.

```
library('swirl')
```

Finally, to get going, start swirl and follow the instructions.

```
swirl()
```

B Git and GitHub

Git is a version control system that allows you to track changes in your code and collaborate with others. GitHub is a web-based platform that hosts Git repositories, making it easy to share and collaborate on projects. Github is NOT the only place to host Git repositories, but it is the most popular and has a large community of users.

You can use git by itself locally for version control. However, if you want to collaborate with others, you will need to use a remote repository, such as GitHub. This allows you to share your code with others, track changes, and collaborate on projects.

> **Note**
>
> It can be confusing to understand the difference between Git and GitHub. In short, Git is the version control system that tracks changes in your code, while GitHub is a platform that hosts your Git repositories and provides additional features for collaboration.

B.1 install Git and GitHub CLI

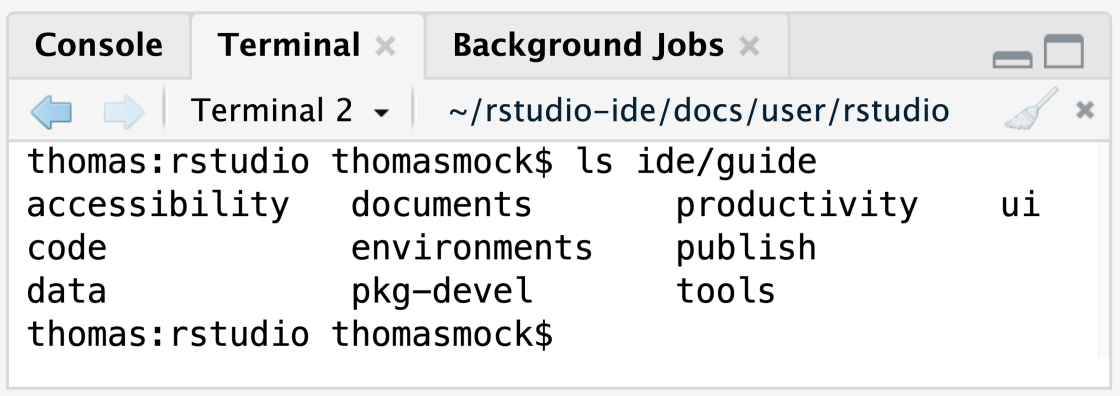
To use Git and GitHub, you need to have Git installed on your computer. You can download it from git-scm.com. After installation, you can check if Git is installed correctly by running the following command in your terminal:

```
git --version
```

We also need the `gh` command line tool to interact with GitHub. You can install it from cli.github.com. To install, go to the releases page and download the appropriate version for your operating system. For the Mac, it is the file named something like "Macos Universal" and the file will have a .pkg extension. You can install it by double-clicking the file after downloading it.

> **Using the RStudio Terminal**
>
> If you are using RStudio, you can use the built-in terminal to run Git commands. To open the terminal, go to the "Terminal" tab in the bottom pane of RStudio. This allows you to run Git commands directly from RStudio without needing to switch to a separate

terminal application.



For more details, see the RStudio terminal documentation.

B.2 Configure Git

After installing Git, you need to configure it with your name and email address. This information will be used to identify you as the author of the commits you make. Run the following commands in your terminal, replacing "Your Name" and "you@example.com" with your actual name and email address:

```
git config --global user.name "Your Name"
git config --global user.email "you@example.com"
```

B.3 Create a GitHub account

If you don't already have a GitHub account, you can create one for free at github.com.

B.4 Login to GitHub CLI

After installing the GitHub CLI, you need to log in to your GitHub account. Run the following command in your terminal:

```
gh auth login
```

B.5 Introduction to Version Control with Git

Welcome to the world of version control! Think of Git as a "save" button for your entire project, but with the ability to go back to previous saves, see exactly what you changed, and even work on different versions of your project at the same time. It's an essential tool for reproducible and collaborative research.

In this tutorial, we'll learn the absolute basics of Git using the command line directly within RStudio.

B.5.1 Key Git Commands We'll Learn Today:

- **`git init`**: Initializes a new Git repository in your project folder. This is the first step to start tracking your files.
- **`git add`**: Tells Git which files you want to track changes for. You can think of this as putting your changes into a "staging area."
- **`git commit`**: Takes a snapshot of your staged changes. This is like creating a permanent save point with a descriptive message.
- **`git restore`**: Discards changes in your working directory. It's a way to undo modifications you haven't committed yet.
- **`git branch`**: Allows you to create separate timelines of your project. This is useful for developing new features without affecting your main work.
- **`git merge`**: Combines the changes from one branch into another.

B.6 The Toy Example: An R Script

First, let's create a simple R script that we can use for our Git exercise. In RStudio, create a new R Script and save it as `data_analysis.R`.

```
# data_analysis.R

# Load necessary libraries
library(ggplot2)
library(dplyr)

# Create some sample data
data <- data.frame(
  x = 1:10,
```

```
  y = (1:10) ^ 2
)

# Initial data summary
summary(data)
```

B.7 Let's Get Started with Git!

Open the **Terminal** in RStudio (you can usually find it as a tab next to the Console). We'll be typing all our Git commands here.

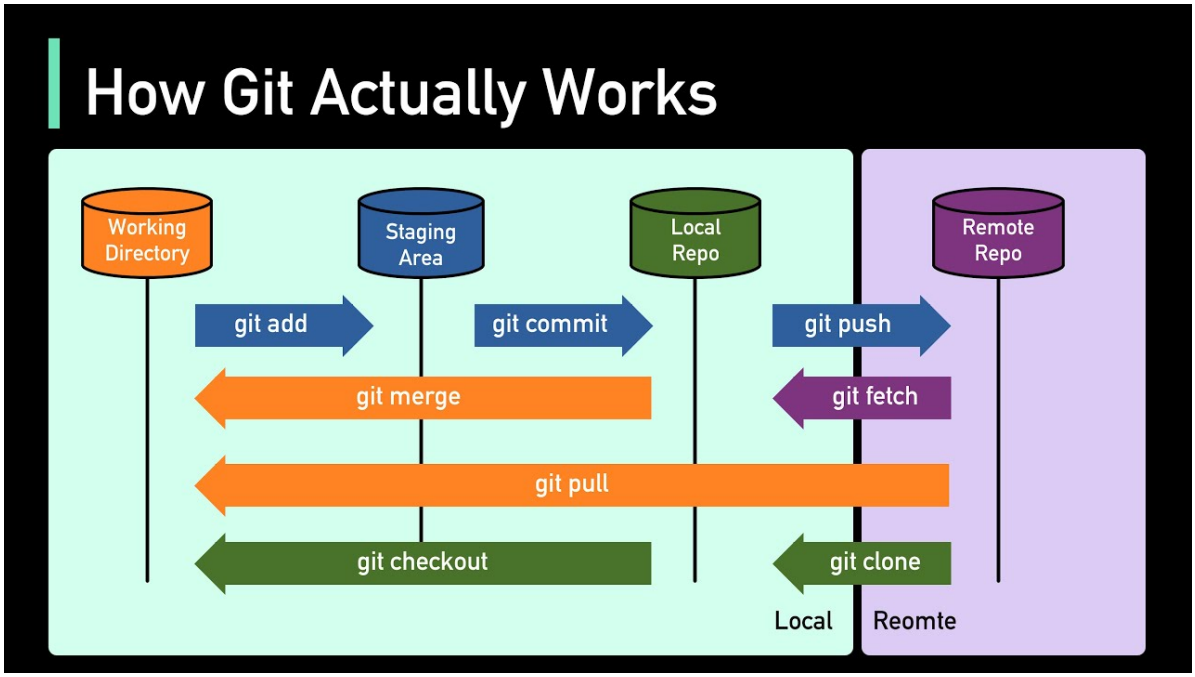


Figure B.1: This is an overview of how git works along with the commands that make it tick. See this video

B.7.1 Step 1: Initialize Your Git Repository

First, we need to tell Git to start tracking our project folder.

```
git init
```

You'll see a message like `Initialized empty Git repository in...`. You might also notice a new `.git` folder in your project directory (it might be hidden). This is where Git stores all its tracking information. Your default branch is automatically named `main`.

B.7.2 Step 2: Your First Commit

Now, let's add our `data_analysis.R` script to Git's tracking and make our first "commit."

1. **Add the file to the staging area:**
   ```
   git add data_analysis.R
   ```
2. **Commit the staged file with a message:**
   ```
   git commit -m "Initial commit: Add basic data script"
   ```
   The `-m` flag lets you write your commit message directly in the command. Good commit messages are short but descriptive!

B.7.3 Step 3: Making and Undoing a Change

Let's modify our R script. Add a plotting section to the end of `data_analysis.R`.

```
# ... (keep the previous code)

# Create a plot
ggplot(data, aes(x = x, y = y)) +
  geom_point() +
  ggtitle("A Simple Scatter Plot")
```

Now, what if we decided we didn't want this change after all? We can use `git restore` to go back to our last committed version.

```
git restore data_analysis.R
```

If you look at your `data_analysis.R` file now, the plotting code will be gone!

B.7.4 Step 4: Branching Out

Branches are a powerful feature. Let's create a new branch to add our plot without messing up our `main` branch.

1. **Create a new branch and switch to it:**

   ```
   git checkout -b add-plot
   ```

   This is a shortcut for `git branch add-plot` and `git checkout add-plot`.

Now, re-add the plotting code to `data_analysis.R`.

```
# ... (keep the previous code)

# Create a plot
ggplot(data, aes(x = x, y = y)) +
  geom_point() +
  ggtitle("A Simple Scatter Plot")
```

Let's commit this change on our new `add-plot` branch.

```
git add data_analysis.R
git commit -m "feat: Add scatter plot"
```

B.7.5 Step 5: Seeing Branches in Action

Now for the magic of branches. Let's switch back to our `main` branch.

```
git checkout main
```

Now, open your `data_analysis.R` script in the RStudio editor. **The plotting code is gone!** That's because the change only exists on the `add-plot` branch. The `main` branch is exactly as we last left it.

Let's switch back to our feature branch.

```
git checkout add-plot
```

Check the `data_analysis.R` script again. **The plotting code is back!** This demonstrates how branches allow you to work on different versions of your project in isolation.

B.7.6 Step 6: Merging Your Work

Our plot is complete and we're happy with it. It's time to merge it back into our `main` branch to incorporate the new feature.

1. **Switch back to the main branch, which is our target for the merge:**

   ```
   git checkout main
   ```

2. **Merge the `add-plot` branch into `main`:**

   ```
   git merge add-plot
   ```

You'll see a message indicating that the merge happened. Now, your `main` branch has the updated `data_analysis.R` script with the plotting code!

C Additional resources

- Base R Cheat Sheet
- Modern Data Visualization with R

C.1 AI

- chatGPT
- Gemini
- Claude
- DeepSeek
- Perplexity

D Data Visualization with ggplot2

Start with this worked example to get a feel for the `ggplot2` package.

- https://rkabacoff.github.io/datavis/IntroGGPLOT.html

Then, for more detail, I refer you to this excellent ggplot2 tutorial.

Finally, for more R graphics inspiration, see the R Graph Gallery.