R for Physicians: Learning with Ease & Efficiency

Ernie Pedapati, MD

6/12/23

Table of contents

Preface

Figure: AI-generated art depicting an adventure map.

Welcome to your essential guide to learning R for medical professionals. If you're a physician eager to master R, you may find yourself at a crossroads. On the one hand, you have the vast online universe of random tutorials, StackOverflow threads, and YouTube videos. On the other, you have highly-reviewed online courses that promise a comprehensive understanding of R but often don't cater to your specific needs as a medical professional.

This book is different.

As a physician who ventured into the world of R post-residency (and now uses it daily), I found myself mired in information overload. Online resources, while immensely helpful, were scattered and disjointed. Online courses, on the other hand, offered structure but lacked the specific context that physicians need. This gap is what led me to pen this guide - a streamlined, context-rich resource tailor-made for physicians.

This guide is built on real-life experiences and hurdles that I faced in my journey of learning R. The issues that confused me, the challenges that stumped me, the "aha" moments I had - they're all in here. But more than that, this guide anticipates and addresses the likely issues that you, as a physician learning R, will encounter.

Each chapter is a step further in your journey, with examples and use-cases centered around medical scenarios, making the learning experience highly relevant and practical for you. Whether it's data manipulation, statistical analysis, or data visualization, the examples and use-cases are drawn from medical scenarios to make your learning experience as relevant as possible.

Embrace this guide as an ally in your pursuit of R mastery. It's a journey that's already been tread, with roadblocks cleared and signposts erected for easy navigation. This book isn't about wandering aimlessly through tutorials, it's about embarking on a well-blazed trail, specifically charted for physicians by a physician. So welcome aboard, let's turn the page and commence our shared adventure.

1 Introduction

1.1 What is R?

Figure 1.1 AI-generated art capturing R statistics in the abstract

R is a free, open-source programming language and software environment specifically designed for statistical computing and graphics. Ross Ihaka and Robert Gentleman created R at the

University of Auckland and it is now maintained by the R Development Core Team. R is essentially composed of a core language and a variety of user interfaces. The core language, often referred to as "base R", is where all the computation and processing happens. The user interfaces, such as RStudio, provide an intuitive frontend where users write code, visualize data, and manage their workflows.

The idea for R came about when Drs. Ross Ihaka and Robert Gentleman were teaching an introductory statistics course and were unsatisfied with the statistical software available to them. They wanted to create a software that was free, user-friendly, and provided an effective way to teach their students statistics.

Interestingly, R is named partly after the first names of the two R authors (Robert and Ross) and partly as a play on the name of S, an influential statistical programming language at the time.

Since its creation, R has grown exponentially, with a vibrant community of users and developers from various fields like academia, industry, and data science. It's maintained by a large, global group of volunteers who continually add to its capabilities by creating new packages.

1.2 Why R?

For those of us who've engaged with statistical analysis during our education or careers, the memory of installing a hard-to-get software replete with countless menus, allowing interaction primarily through mouse clicks, is all too familiar. Perhaps you ran various commands on data, or dove into different analyses, all while navigating through these menus and outputs. I recall those times vividly and suspect many of you have similar experiences.

R, to many, might initially appear as an obscure programming language tucked away in academia or tech-based industries.

But let me challenge your perception!

The very traits of R that lend it an air of obscurity are, in fact, its most significant assets.

1.3 Great things about R

1. **You don't have to install R**:

   Unlike traditional software, R doesn't demand space on your computer. It runs smoothly in the cloud, making it accessible from anywhere, on any device. More importantly, the scripts, the 'statistical documents' you write in R, are not one-off commands. They are reusable, editable, and shareable pieces of code that capture your entire analytical process from start to finish.

2. **R is free:**

   I still have bad memories of trying to find a student copy of an expensive statistics software I bought in college. The complete and latest R suite is free to run on the platform of your choice and the 100% cloud-based R-Studio has a generous amount of resources for their free plan.

3. **Active and Friendly Community:**

   R has a large, active, and helpful user community. This means help is often readily available through online forums, blogs, and tutorials.

4. **Narratives, not isolation:**

   Writing scripts in R provides a natural and coherent flow to your work, a linear narrative, if you will. Instead of isolated tables and analyses separated by an output window, you have a comprehensive, logical story.

5. **Awesome outputs:**

   The table and graphics capabilities of R are second to none. Packages such as `flextable` and `ggplot2` provide advanced functionality for creating high-quality, customizable, and publication-ready graphics. You will quickly recognize in print and media that R is everywhere!

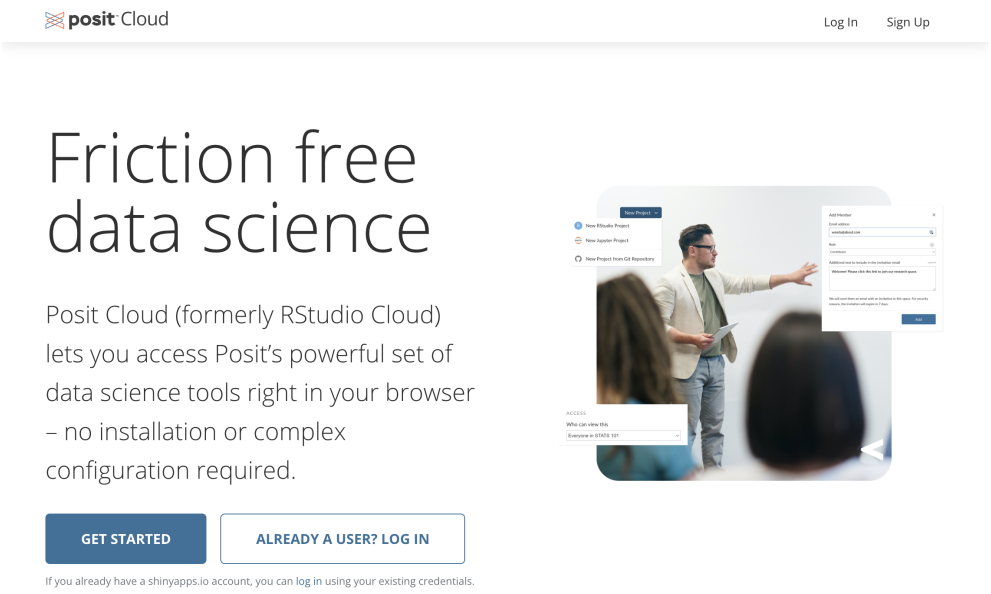
After spending some time with R, becoming familiar with its capabilities, and experiencing its versatility firsthand, you might find it hard to even recognize it compared to the statistical programs you used back in college.

1.4 Using RStudio Cloud

RStudio Cloud is a great tool that simplifies the process of setting up R. It allows you to run R directly from your web browser, eliminating the need to install software locally and handle any setup hassles. In 2023, RStudio was renamed Posit Studio and Posit Cloud but in this book, I will continue to refer to it as RStudio and RCloud.

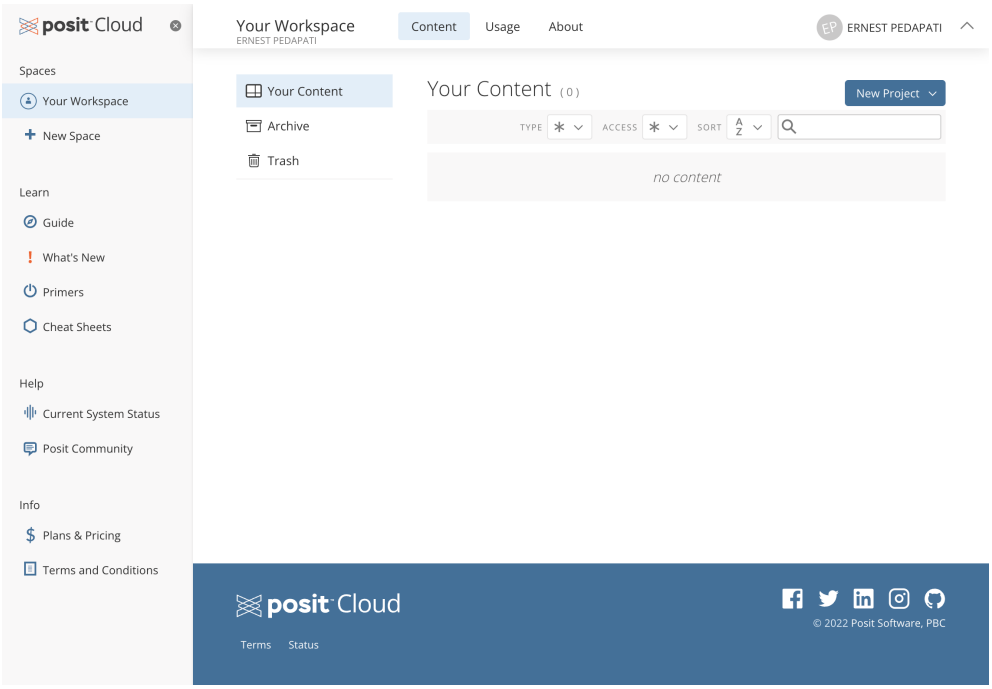
1. **Creating an Account**

   First, navigate to the RStudio Cloud website (**https://rstudio.cloud/**). If you don't have an account yet, click on "Sign Up" to create one. Enter your details, then click "Register". You'll receive an email to confirm your account.



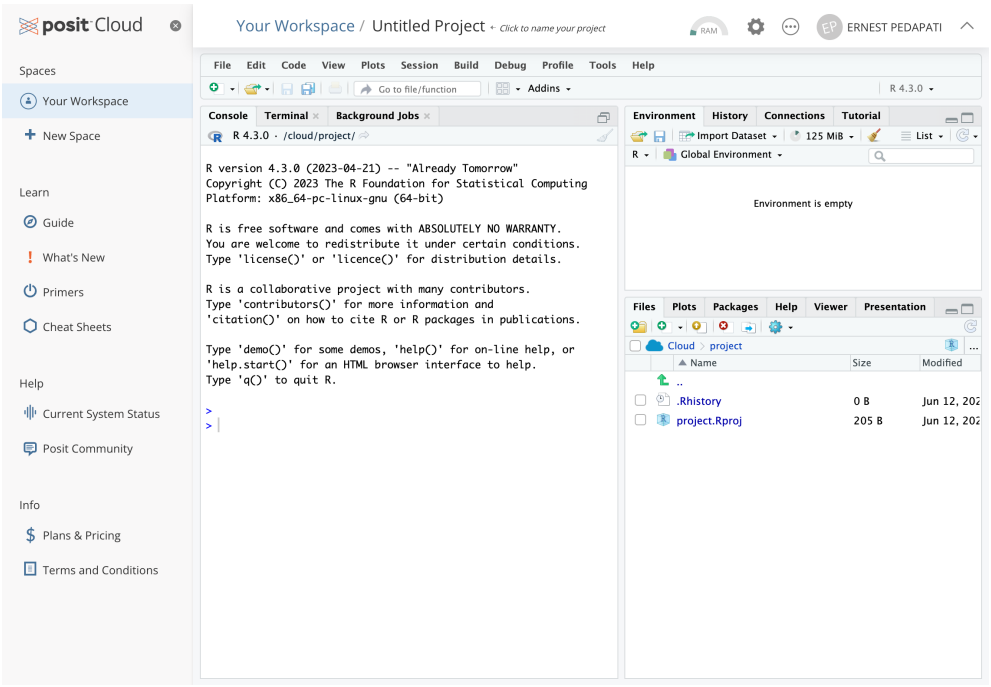
2. **Creating a New Project**

   After you've logged in, you'll see your RStudio Cloud workspace. Click on the "New Project" button to start a new R project. Enter a name for your project and then click on "Create Project".



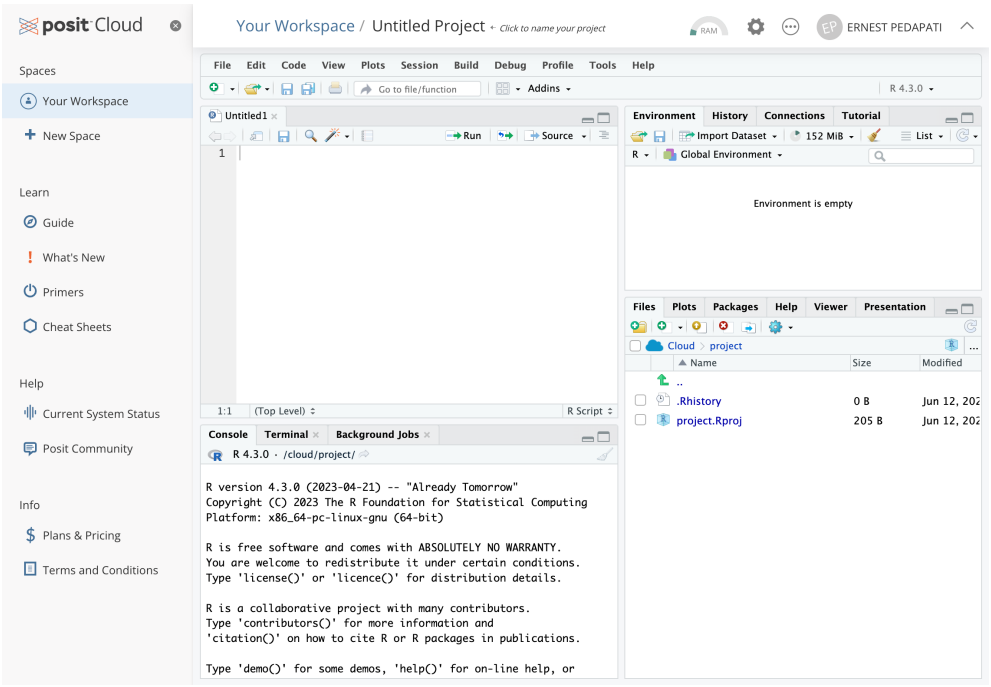
3. **RStudio Cloud Interface**

   Now you're inside the RStudio interface, running within your web browser. On the left, you'll see the R console where you can enter R commands. The right panel contains tabs for plots, packages, help, and files. The top-left panel is for scripts or R Markdown files.



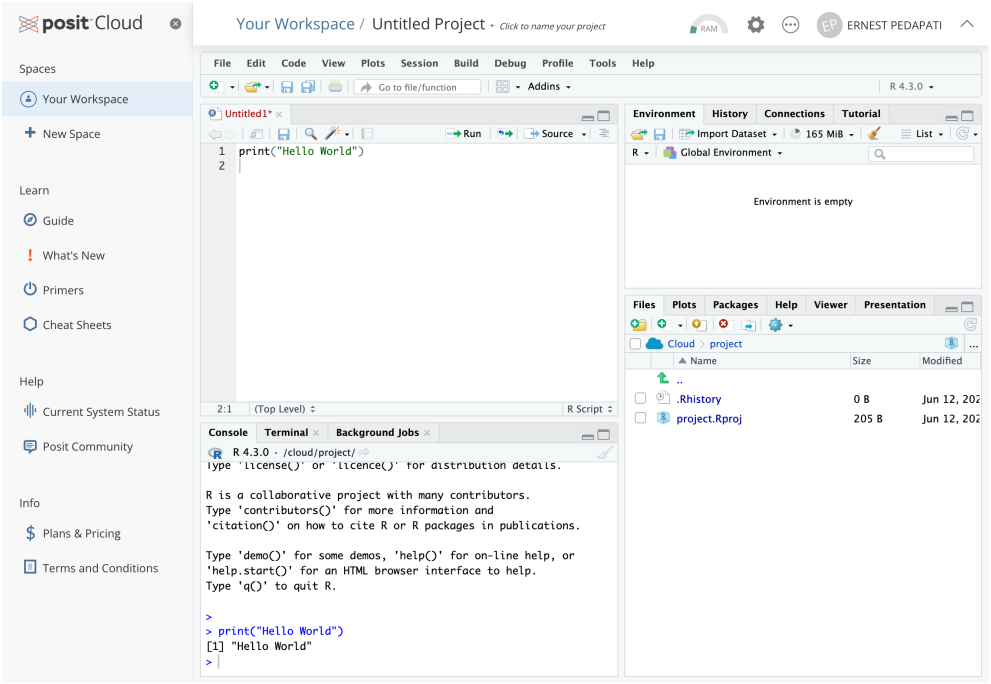
4. **Writing and Running R Code**

   To start coding, click on the "File" menu, then "New File", and then "R Script". An editor will open where you can write your R code. After writing your code, you can run it by clicking on the "Run" button, or by pressing Ctrl+Enter (Cmd+Enter on Mac).



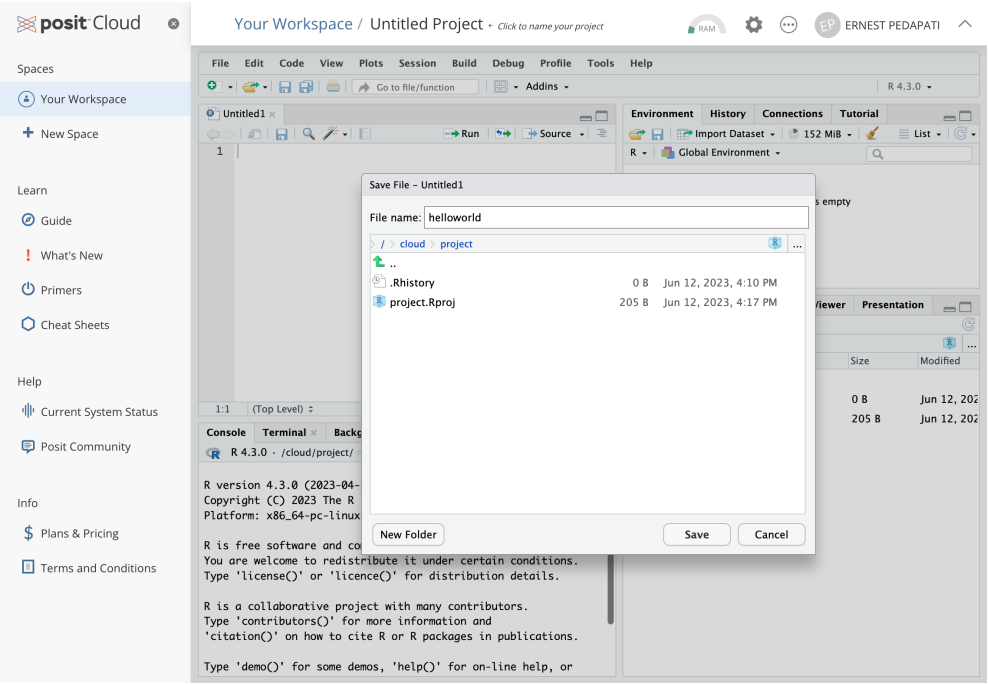
5. **Run some practice code**

Let's test out your setup by printing "Hello World!". In your empty script type `print("Hellow World!")` and click on the "Run" button.



6. **Saving and Sharing Your Work**

   RStudio Cloud autosaves your work as you go, so you don't have to worry about losing your code. If you want to share your project, click on the "Settings" gear icon in the top-right corner of the project, and set "Who can view this project" to "Everyone". You can then share the URL of your project with others.



Congratulations! You're now up and running with RStudio Cloud. You have a versatile, powerful tool at your fingertips, ready to tackle your data analysis needs.

2 Getting Started with R

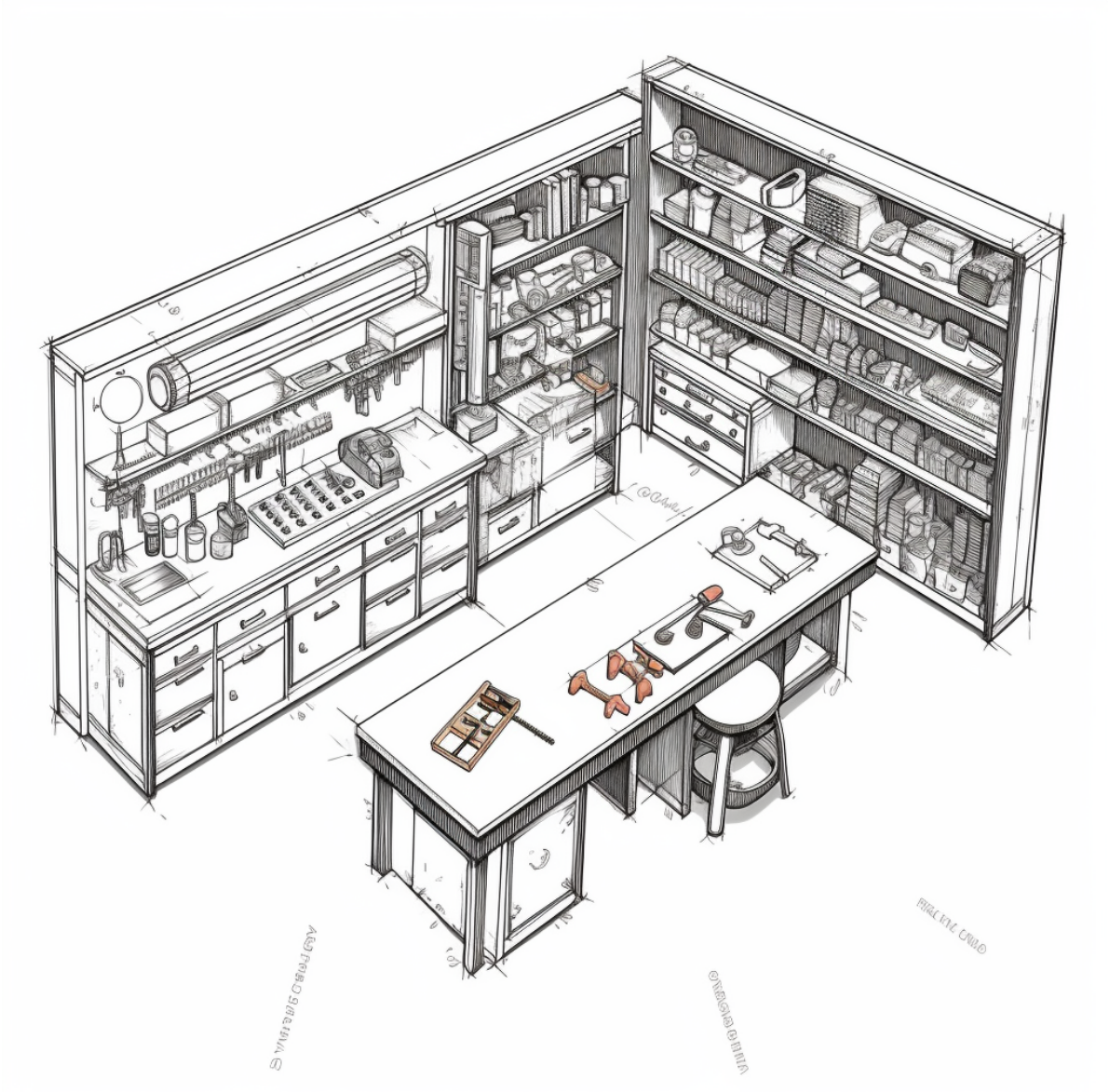


Figure 2.1: AI-generated representation of the R “workshop” analogy

2.1 The Big Picture

In previous chapters, we learned what R is and why we would want to use it. We setup access to R though our web browser by signing up for a free RStudio account. Let's move ahead to learning about R packages. R packages instantly give you access to a universe of tools and datasets.

2.2 Introduction

The `medicaldata` package is a collection of datasets that are relevant to medical research. It offers a robust collection of medical datasets extracted from a wide range of study designs, including randomized controlled trials, retrospective and prospective cohort studies, and case-control studies. These datasets encompass a diverse array of medical conditions and treatment approaches, providing rich opportunities for learning, exploration, and analysis.

The package contains over 19 datasets, covering a wide range of medical topics, including cancer, cardiovascular disease, diabetes, and mental health. One of the datasets, 'strep_tb', for example, is drawn from the groundbreaking 1948 trial of Streptomycin treatment for tuberculosis, the first modern randomized, placebo-controlled clinical trial. The datasets in the `medicaldata` package are all in a standard format, which makes them easy to use with R.

2.3 R Packages

How do we get access to all of these interesting datasets?

The beauty of R lies in its simplicity and ease of access to a wealth of data and tools. Unlike traditional methods where you might have to navigate to a website, download files, and manually place them into specific directories, R simplifies this process immensely. One of the big advantages of R is that it provides the capability to access numerous tools and datasets directly from the command line using a single line of code.

2.4 Packages are Toolboxes

Think of R as large workshop with access to a main tool depot. with access to lots of specialized toolkits. In this workshop you are assigned a personal workbench. The toolboxes are designed and put together by different craftsmen, making them unique in the tools they contain. If you've identified a toolbox that you need for a specific project, you first have to bring that toolbox into your workbench. In fact, it would not be unusual to retrieve several toolkits depending on needs of your project.

However, just lugging the toolboxes to your workbench doesn't necessarily mean you can immediately use the tools they contains. If you want access to all your screwdrivers at once, you will need open a specialized screwdriver toolkit. On the other hand, you might only want a single screwdriver from a special toolkit. this is more than just keeping your workbench tidy, you also don't want to have duplicates of similar tools around which may lead to confusion.

2.5 Breaking down the workshop analogy

Let's connect this analogy with learning R

- The workshop represents R and RStudio
- The personal workbench is your R project
- The toolboxes are R packages
- The tool depot is the Comprehensive R Archive Network (CRAN) package repository (more on this later!)
- Lugging the toolkit to your workbench is analogous to installing the package
- Opening the entire toolkit is adding it to your active R libraries
- Selecting a single tool from a toolkit is the same as using the `::` operator on a package.

2.6 Software repositories in R

2.6.1 CRAN (https://cran.r-project.org/)

The CRAN repository where officially approved and tested packages are stored.These packages are well-documented, reliable, and updated regularly. So, if you need a tool for a common task, you're likely to find a toolbox containing it in the CRAN repository.

2.6.2 Development repositories

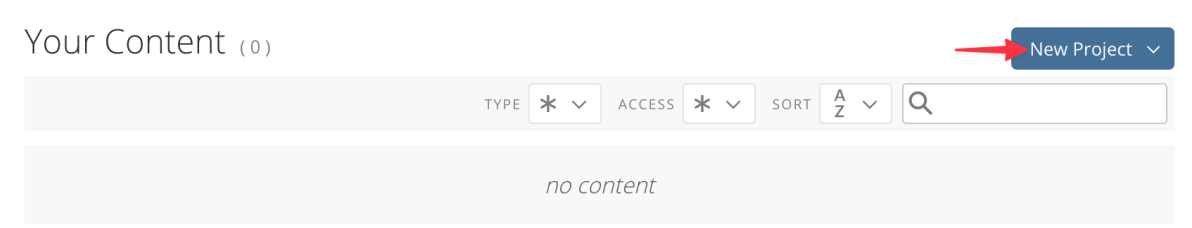
Places like Github and other code repositories offer exciting, cutting-edge tools that may not have made their way to the main CRAN depot yet. While the tools from these workshops can be highly useful, they also come with a word of caution as they may not be as thoroughly tested and documented as those in the CRAN depot. You may also you need to use the latest advancements with well-known packages that have not been updated on CRAN yet.

2.7 Walkthrough

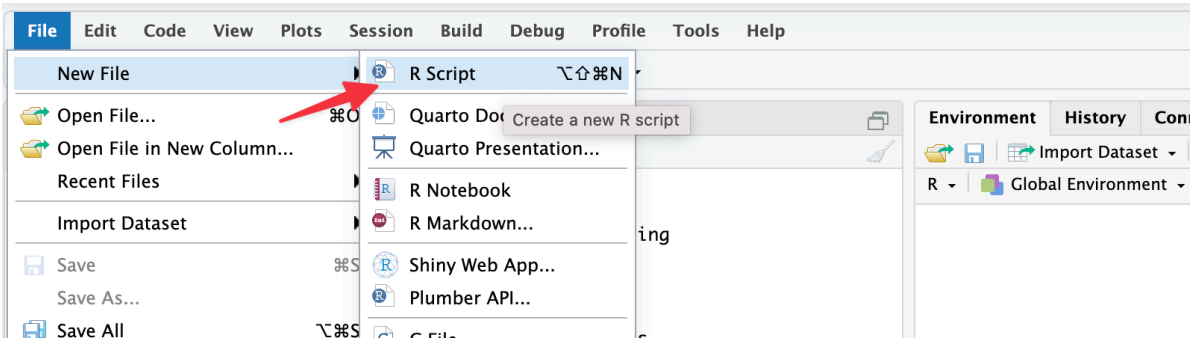
Let's walk through loading the `medicaldata` package which will provide the datasets we will use thoughout the rest of book.

2.7.1 Setting the stage

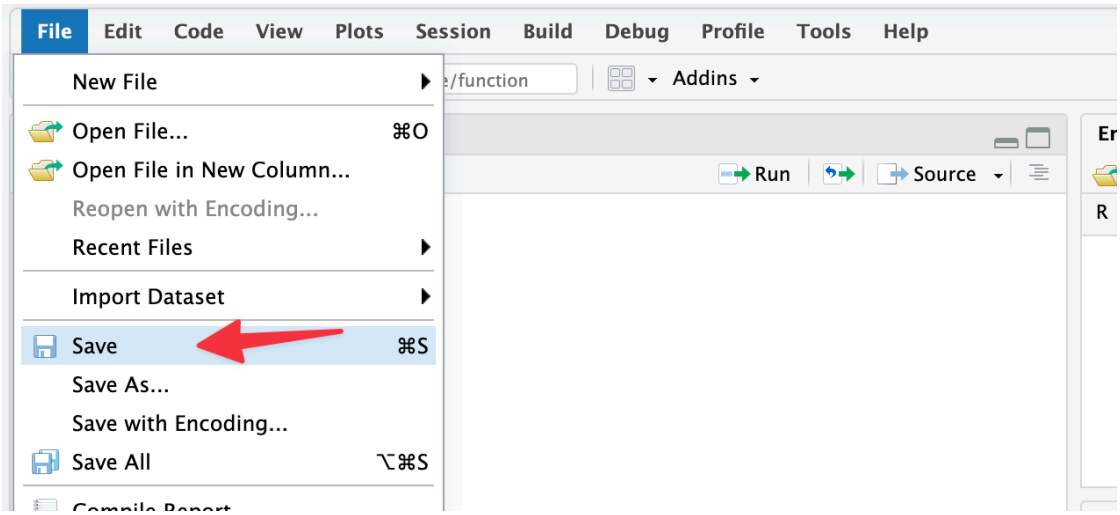
1. First create a new R project in R Cloud



2. Create a new empty R script



3. Save the File as chapter1.R

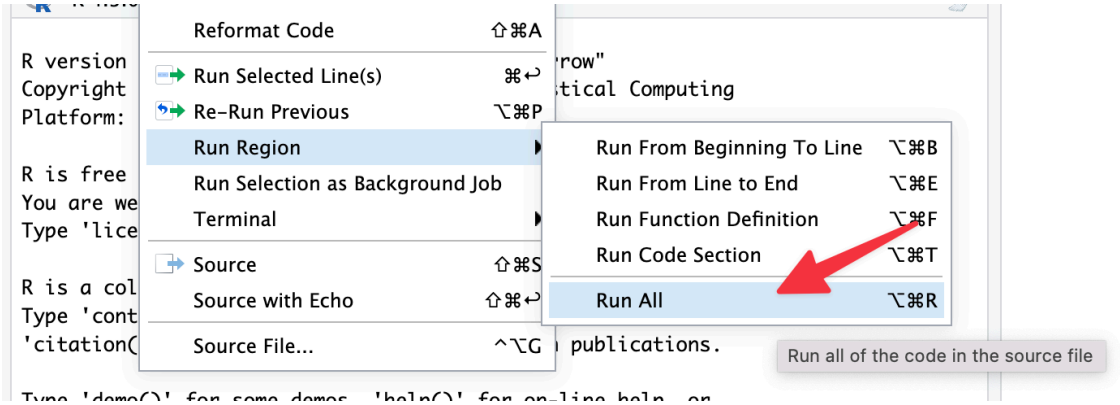


4. Type the following into your blank script:

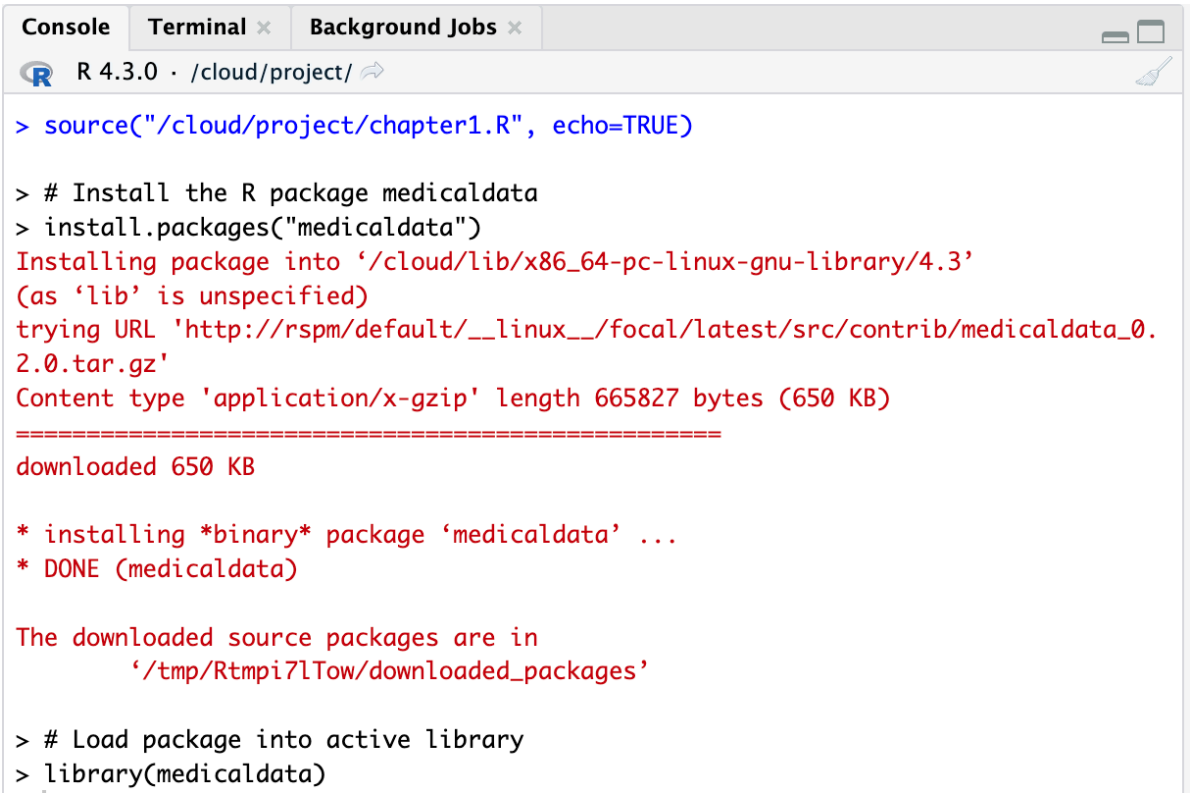
```
# Install the R package medicaldata
install.packages("medicaldata")

# Load package into active library
library(medicaldata)
```

5. Access the `Code` menu and select `Run All`



6. Examine the output in the console window.



2.8 Reviewing the Code

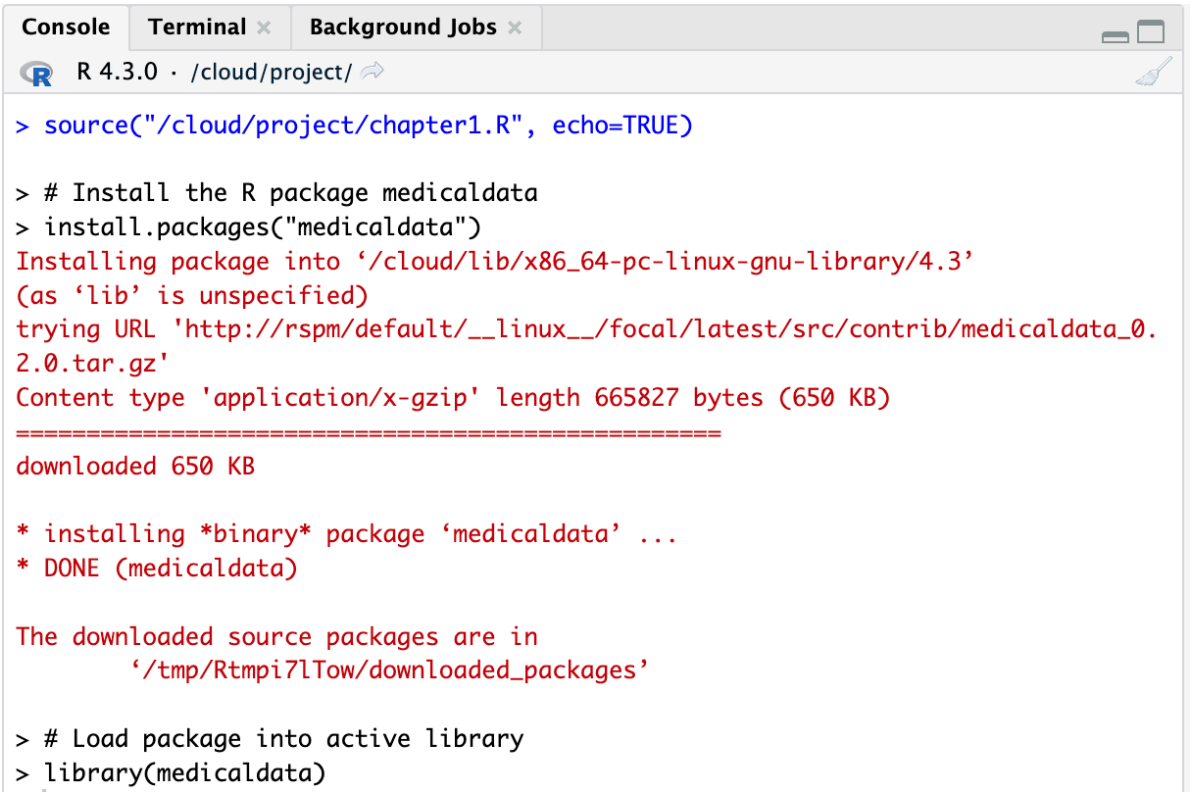
Diving into the code we've written, several key points need to be highlighted.

- Code isn't like regular writing. Forget paragraphs; each command you write stands alone on its own line.
- The pound symbol # leads us to the next key point. Placing this at the start of a line tells R to gloss over this part when executing code. This is what we call a comment and it's a handy way to leave notes for yourself and others.
- The third point concerns functions, such as `install.packages()` and `library()`. Consider functions as time-saving shortcuts for complicated operations. They take inputs and give outputs.
- When you see parentheses associated with a term, think function. Whatever goes inside these parentheses are known as input parameters to the function.
- Look closely at the use of quotes around `medicaldata` in the `install.packages` line, but their absence in the `library` line. In R, quotes aren't just punctuation, they serve a specific function which we'll delve deeper into later.

2.9 Examining the output

Our first interactive coding command produced some interesting output in the R console. Let's take a moment to discuss the R console and understand what it just told us.

The R console is akin to a live conversation with R. When you type a command and hit enter, R listens, processes the request, and then speaks back to you. This "speech" is the output you see on your screen. Let's look at the output from our script.



Now, back to the output of our first command, **`install.packages("medicaldata")`**. This command tells R to install the "medicaldata" package, a collection of ready-made functions and data. R takes this command, connects to a server, and starts to download the package. It provides us with live updates, telling us how large the package is (650 KB), and its download status.

The next few lines, **`* installing *binary* package 'medicaldata' ...`** and **`* DONE (medicaldata)`**, tell us that R has successfully installed the package.

The last line of the output, **`'/tmp/Rtmpi7lTow/downloaded_packages'`**, is R's way of saying "If you need the downloaded files, here's where I've stored them".

Once the installation is complete, we run `library(medicaldata)`. This command tells R to open the toolbox of `medicaldata` and make its tools available for use. There's no output after this command, which usually indicates that the command has run successfully and the package is ready to use.

2.10 Checking the results of our work

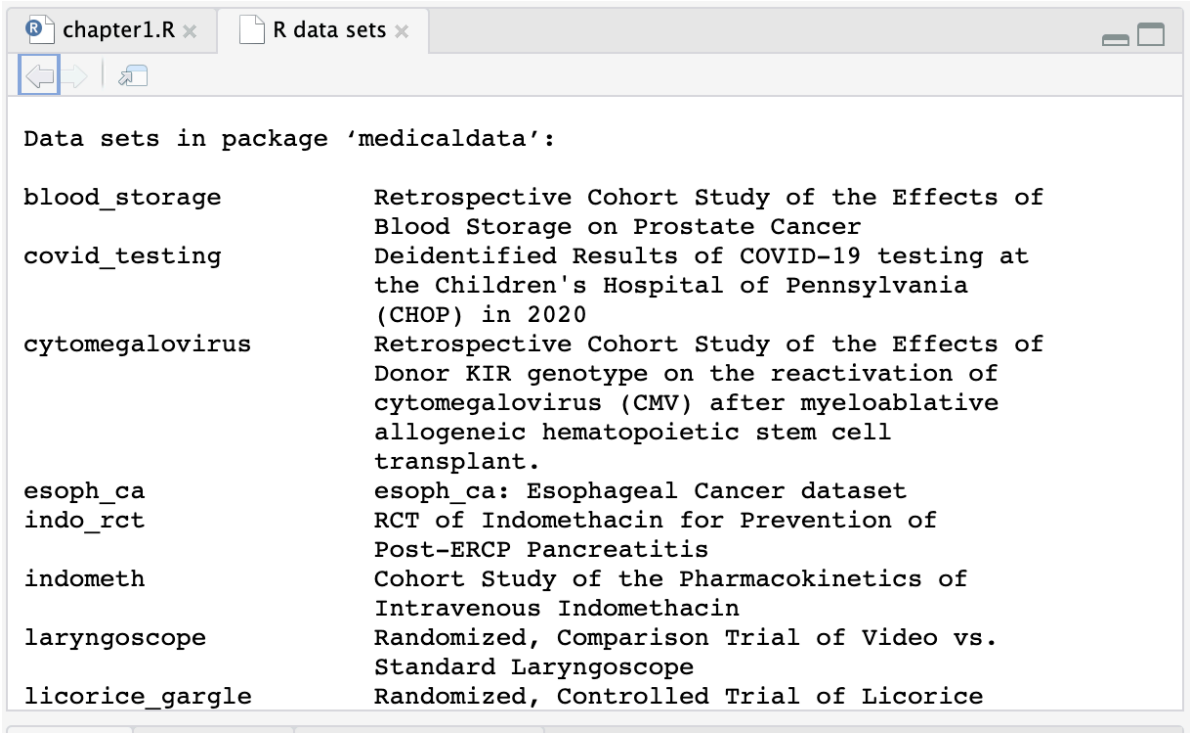
Installing packages is a fundamental aspect of using R. In the next few chapters we will learn more about the language of R to manipulate and process data, but for now let's see the fruit of our labor.

Most R packages have excellent documentation. The medicaldata package is no exception. The guide to the datasets in the package can be found at this link: https://higgi13425.github.io/medicaldata/.

Let's use the instructions from the package author to view the different datasets we now have installed by typing into the console:

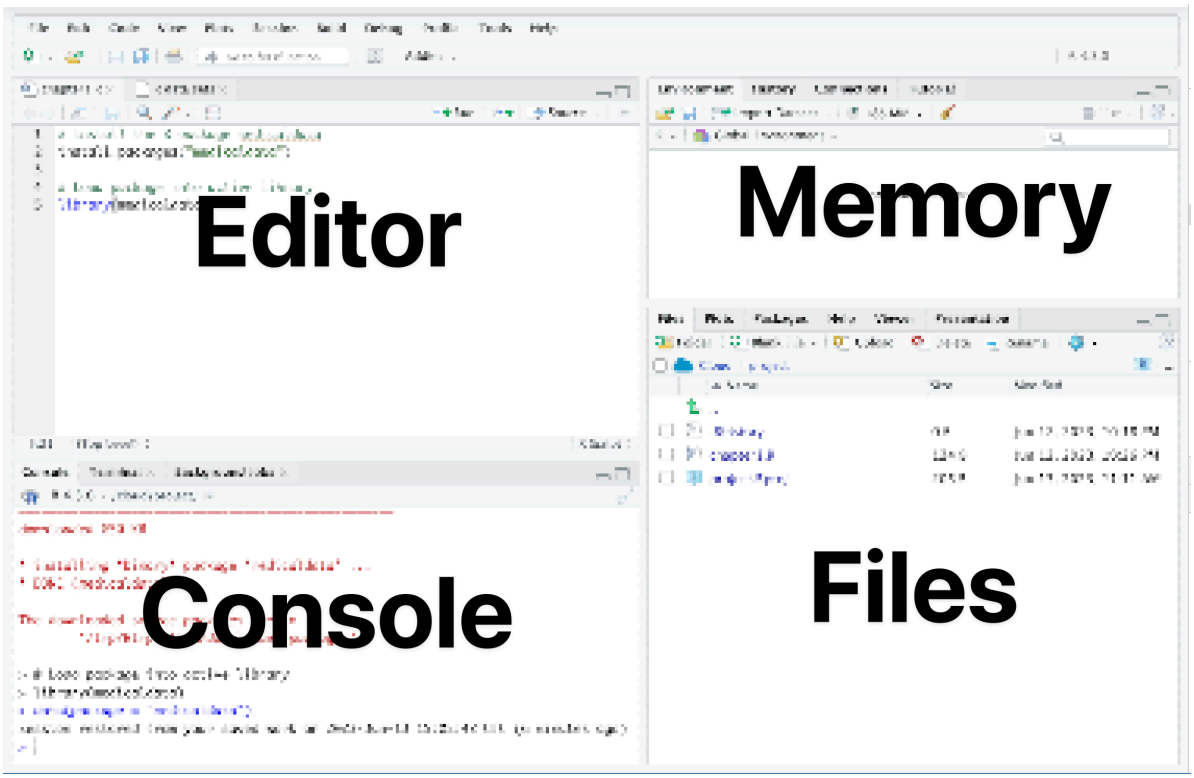
```
data(package = "medicaldata")
```

Once you execute this command, R will display an interactive window showing you all the datasets available in the "medicaldata" package. Each dataset is listed with a brief description of the kind of data it contains, which can be very useful when deciding which dataset to use for a particular analysis.



2.11 Entering data within a script versus the console

When you're working with R, you have two main places where you can enter your data or commands: the script editor and the console.



The script editor is your workspace for crafting R scripts. This is where you write your lines of code, organize your thoughts, define functions, and generally create your R programs. Anything you write in the script editor is saved and can be run as many times as you want, making it ideal for larger, more complex analyses.

On the other hand, the console is the live interaction space where R executes commands and displays results. Anything you type directly into the console is run immediately, but it's not saved once you close your R session. It's a great place for quick calculations, testing small bits of code, or inspecting data.

In essence, the script editor is your drafting table where you design and plan, and the console is more like a chat where you can immediately execute and see your plans come to life. As you continue to work with R, you'll become more comfortable determining when to use each for different tasks.

Indeed, certain commands are best suited for direct execution in the console, especially those that serve to inspect data or check a package's contents. The command to view the datasets within the `medicaldata` package is a good example of this. It's a 'single-use' operation that doesn't necessarily form part of the core workflow in your script, but rather provides you with valuable contextual information.

2.12 Chapter Summary

In this chapter, we introduced the basics of using R, with a specific focus on accessing and utilizing packages from CRAN. We introduced the concept of functions, explained the role of comments, and highlighted the differences between writing code in a script versus executing commands in the console. Through the installation and exploration of the 'medicaldata' package, we demonstrated the ease and power of working with packages in R. This foundational knowledge will serve as a solid base as we delve deeper into R programming in subsequent chapters.

3 Chapter 2: Data Frames in R

Figure 2.1: AI-generated art of an abstract perspective of a data frame.

3.1 The big picture

Now that you've set up your rstudio.cloud account and familiarized yourself with the key elements of the interface, such as the code editor and command console, it's time to delve into the world of data frames. In the realm of medical practice, data frames serve as indispensable tools for organizing and analyzing clinical data.

3.2 Introduction

Data frames are perhaps one of the most important and widely used data structures in R. If you're familiar with spreadsheets from software like Excel, then you can think of a data frame like a spreadsheet in R. It is a two-dimensional table where each column contains values of one variable, and each row contains one set of values from each column. A key thing to note about data frames is that different columns can contain different types of data (numeric, character, etc.), but each column must contain the same number of data items (rows).

3.3 The many names of data frames in R

Data frames in R come with an assortment of names, which can make the initial stages of learning R a bit confusing. Recognizing and understanding these terminologies is crucial as data frames are foundational to data handling in R.

Traditionally, a data frame in R is referred to as a 'data.frame'. However, with the advent of packages like 'tibble' in the tidyverse ecosystem, a data frame can also be called a 'tibble'. Meanwhile, in the data.table package, it's called a 'data.table'.

Each of these types has its unique characteristics and benefits, but they all share the same primary purpose - to help you manage data in a structured, tabular format. Don't let the various names confuse you; they all revolve around the same fundamental concept.

3.4 Exercise 2.1 Hands-on with a data frame

3.4.1 The lunch lecture

Let's take a hands-on tour of investigating a dataset through a practical vignette. There are no shortage of ways in R to interact with data frames.

Vignette 1: You are a 3rd year infectious disease fellow on a research block. Your supervisor has asked you to present a captivating lunch lecture to several visiting faculty. Knowing that many history buffs will be present, you decide to present a reanalysis of an infectious disease landmark - the discovery of streptomycin as a treatment for tuberculosis (TB).

After all, why not? You have heard that the original dataset is only a few lines away in R!

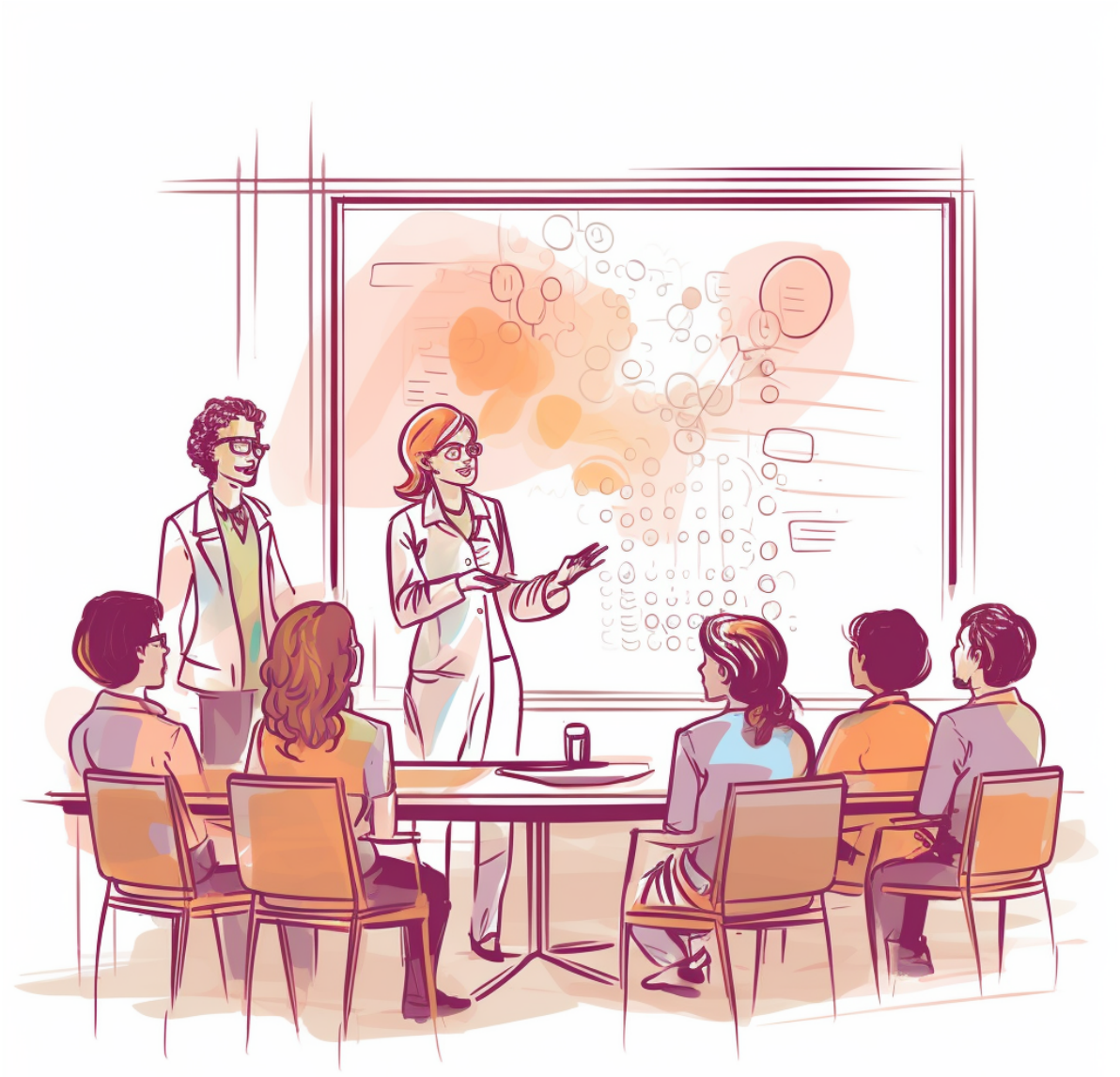


Figure 2.2 AI generated imagine of the captivating lunch talk

For instance, let's consider a simple medical example:

```
patient_data <- data.frame(
  name = c("John Doe", "Jane Doe", "Mary Johnson", "James Smith"),
  age = c(25, 30, 35, 40),
  disease = c("Hypertension", "Diabetes", "Asthma", "None"),
  checkup_date = as.Date(c("2023-01-01", "2023-01-15", "2023-02-01", "2023-02-15"))
)
```

Here, `patient_data` is a data frame that contains four columns: `name`, `age`, `disease`, and `checkup_date`. Each column is a different type of data: character, numeric, character, and date, respectively.

You can access the individual columns of a data frame in the following way:

```
patient_data$name
```

This will return the `name` column of the `patient_data` data frame.

The ability to contain different types of data in each column makes data frames particularly suitable for managing and analyzing real-world data. The majority of the data you'll likely be working with in R will be in the form of data frames.

In the following chapters, we will explore various operations we can perform on data frames such as filtering rows, selecting columns, arranging (or sorting) the data, and summarizing it.

For now, try to create a data frame of your own using the `data.frame()` function. Perhaps you could create a data frame for lab test results, where each row is a patient's lab test and the columns contain details about the test, such as patient's name, test name, test result, and the date of the test.

Let's talk about the three major ecosystems for data manipulation in R: **base R**, **tidyverse**, and **data.table**. These three ecosystems each have different ways of creating and interacting with data frames, and it's important to be aware of these differences.

1. **Base R**: In base R, we use the `data.frame()` function to create data frames. The operations in base R are straightforward but can sometimes become complex for intricate data manipulation tasks.
2. **Tidyverse**: This is a collection of R packages that share an underlying philosophy and common APIs. `tibble`, part of the tidyverse, is an enhanced version of data frames which are easier to work with. Tidyverse functions are highly readable and are often favored for data manipulation and exploration.

3. **Data.table**: This package offers data frames that are faster and more memory-efficient, along with syntax that is more concise. However, it may have a steeper learning curve.

Exercise

Let's create a data frame in each ecosystem using patient information.

1. Base R:

```
patient_data_base <- data.frame(
  name = c("John Doe", "Jane Doe", "Mary Johnson", "James Smith"),
  age = c(25, 30, 35, 40),
  disease = c("Hypertension", "Diabetes", "Asthma", "None")
)
```

2. Tidyverse:

```
library(tibble)
patient_data_tidy <- tibble(
  name = c("John Doe", "Jane Doe", "Mary Johnson", "James Smith"),
  age = c(25, 30, 35, 40),
  disease = c("Hypertension", "Diabetes", "Asthma", "None")
)
```

3. Data.table:

```
library(data.table)
patient_data_dt <- data.table(
  name = c("John Doe", "Jane Doe", "Mary Johnson", "James Smith"),
  age = c(25, 30, 35, 40),
  disease = c("Hypertension", "Diabetes", "Asthma", "None")
)
```

Notice how the syntax changes slightly between the three ecosystems. Throughout this book, we will be focusing mainly on base R and tidyverse, but it's good to be aware of data.table, especially for large datasets. Practice creating these data frames and examine how they are displayed differently in your R console.

4 Summary

This book is designed as a guide, a companion on your journey to mastering R, but it's not just another instruction manual. As a fellow physician who discovered the power of R after my residency, I've encountered and navigated the same challenges you're likely to face. This book synthesizes those experiences into a streamlined learning path that addresses our unique needs and use-cases in the medical field. It is a comprehensive roadmap, crafted to transform you from an R novice to a confident user, capable of leveraging this powerful tool to improve your research, data analysis, and overall work.

Our goal is not to become software engineers or statisticians, but rather to harness the potential of R to make our tasks more efficient and our decisions more data-driven. To this end, this book covers the essentials of R programming, data manipulation, statistical analysis, and data visualization. Moreover, the lessons are grounded in real-world, relatable examples, including using medical datasets, making the learning experience engaging and intuitive. By the end of this book, you will have gained a robust understanding of R, its application in healthcare, and most importantly, the confidence to explore further and ask the right questions of your data.

References