

R for Data Analysis

Trevor French

8/16/2022

Table of contents

I	Introduction	7
	Prerequisites	8
	Structure of the Book	8
	License	9
1	What is R?	10
	1.1 History	10
	1.2 Resources	10
2	What is Data Analysis?	11
	2.1 The Process of Data Analysis	11
	2.2 Resources	12
3	Setup	13
	3.1 Install R	13
	3.2 Install R Studio	15
	3.3 Alternatives	17
	3.3.1 R Studio Cloud	17
	3.3.2 Replit	17
	3.3.3 Kaggle	18
	3.4 Resources	18
II	Part I: Fundamentals	19
4	Getting Familiar with R Studio	21
	4.1 Customization	21
	4.2 Source Pane	23
	4.3 Console	26
	4.4 Environment	27
	4.5 Files	27
	4.6 Resources	28
5	Programming Basics	30
	5.1 Executing Code	30
	5.1.1 Console	30

5.1.2	Script	31
5.2	Comments	33
5.3	Variables	34
5.4	Operators	34
5.4.1	Arithmetic Operators	36
5.4.2	Comparison Operators	37
5.4.3	Logical Operators	38
5.4.4	Assignment Operators	39
5.4.5	Miscellaneous Operators	41
5.5	Functions	41
5.6	Loops	43
5.6.1	While Loops	43
5.6.2	For Loops	44
5.7	Conditionals	45
5.8	Libraries	45
5.9	Resources	46
6	Data Types	48
6.1	Numeric	48
6.1.1	Double	48
6.1.2	Integer	49
6.2	Complex	50
6.3	Character	50
6.4	Logical	50
6.5	Raw	50
6.6	Resources	51
7	Data Structures	52
7.1	Vectors	52
7.2	Lists	52
7.3	Matrices	53
7.4	Factors	53
7.5	Data Frames	54
7.6	Arrays	54
	Exercises	55
	Questions	55
	Answers	55

III Part II: Data Acquisition	56
8 Included Datasets	58
8.1 View Catalog	58
8.2 Working with Included Data	59
8.3 Resources	60
9 Import from Spreadsheets	61
9.1 Import from .csv Files	61
9.2 Import from .xlsx Files	61
9.3 Import and Combine Multiple Files	62
9.4 Resources	62
10 Working with APIs	63
10.1 Install Packages	63
10.2 Require Packages	63
10.3 Make Request	63
10.4 Parse & Explore Data	64
10.5 Adding Parameters to Requests	65
10.6 Adding Headers to Requests	65
10.7 Resources	65
10.7.1 Helpful APIs	65
Exercises	66
Questions	66
Answers	66
IV Part III: Data Preparation	67
11 Data Cleaning	69
11.1 Renaming Variables	69
11.2 Splitting Text	71
11.3 Replace Values	71
11.4 Drop Columns	72
11.5 Drop Rows	74
11.6 Resources	74
12 Handling Missing Data	75
12.1 Replacing Nulls/NAs (constant value?)	75
12.2 Mean Imputation	76
12.3 Multiple Imputation?	76
12.4 Resources	76

13 Outliers	77
13.0.1 Box and Whisker Plot	77
13.1 Resources	77
14 Organizing Data	78
14.1 Sorting	78
14.2 Filtering	79
14.3 Grouping	79
14.4 Resources	79
Exercises	80
Questions	80
Answers	80
 V Part IV: Developing Insights	 81
15 Summary Statistics	83
15.1 Quantitative Data	83
15.1.1 Measures of Central Tendency	83
15.1.2 Scope of Data	83
15.2 Qualitative Data	83
15.2.1 Frequency not necessarily qualitative	83
15.2.2 Word Cloud	83
15.3 Resources	83
16 Regression	84
16.1 Linear Regression	84
16.2 Multiple Regression	85
16.3 Logistic Regression	85
16.4 Resources	85
17 Plotting	86
17.1 Base R	86
17.1.1 Scatter Plot	86
17.1.2 Box Plot	86
17.1.3 Plot Matrix	86
17.1.4 Pie Chart	86
17.1.5 Bar Plot	86
17.1.6 Histogram	87
17.1.7 Density Plot	87
17.1.8 Dot Chart	87
17.2 ggplot2	87
17.2.1 Different types of plots?	87

17.3 Resources	87
Exercises	88
Questions	88
Answers	88
 VI Part V: Reporting	 89
18 Spreadsheets	91
18.1 Export	91
18.1.1 Export .csv Files	91
18.1.2 Export .xlsx Files	94
18.2 Formatting	94
18.3 Formulas	95
18.4 Resources	95
 19 R Markdown	 96
19.1 Format Options (All of the output options)	98
19.2 Example (maybe pdf? and another section for html/powerpoint/etc.)	98
19.3 Including Plots	98
19.4 Resources	98
 20 R Notebook	 99
20.1 Resources	101
 21 R Shiny	 102
21.1 Quickstart	102
21.2 Basic Components of a Shiny Application	105
21.2.1 Server	105
21.2.2 UI	105
21.3 shinyapps.io	105
21.4 Deploying Application	105
21.5 Shinyuieditor?	105
21.6 Resources	105
 Exercises	 106
Questions	106
Answers	106
 References	 107

Part I

Introduction

“There is synthesis when, in combining therein judgments that are made known to us from simpler relations, one deduces judgments from them relative to more complicated relations. There is analysis when from a complicated truth one deduces more simple truths.” -André-Marie Ampère (Hofmann 1996)

Everyone is a data analyst. The purpose of this book is to inspire and enable anyone who reads it to reconsider the methods they currently employ to analyse data. This is not to suggest that the methodologies outlined will be useful or sufficient for everyone who reads it. Some analyses can be performed quickly without the need for additional computation while others will require advanced analytics techniques not outlined in this book; however, the aspiration is that all will be equipped with novel tools and ideas for approaching data analysis.

Prerequisites

No prior knowledge is required to begin this book. The content will start at the very beginning by showing you how to set up your R environment and the basics of programming in R. By the end of the book, you will be able to perform intermediate analytics techniques such as linear regression and automatic report generation.

You will need an environment which you use to run your code. It is recommended that you download R and R Studio locally for this requirement. This book will walk you through how to do that as well as offer alternatives if that is not an option for you.

Structure of the Book

- **Part I (Fundamentals)** will introduce you to the basics of programming in the context of R.
- **Part II (Data Acquisition)** will teach you how to create, import, and access data.
- **Part III (Data Preparation)** will show you how to begin preparing your data for analysis.
- **Part IV (Developing Insights)** goes through the process of searching for and extracting insights from your data.
- **Part V (Reporting)** demonstrates how to wrap your analysis up by developing and automating reports.

Each part will contain several chapters which cover specific ideas related to the overarching topic. At the end of each of these chapters you will find additional resources for you to use to dive deeper into the ideas.

Each part will be concluded with practical exercises for you to test your skills.

License

This website is free to use, and is licensed under the [Creative Commons Attribution-NonCommercial-NoDerivs 4.0](#) License. Physical copies of this book are not currently available; however, you can download a pdf in the top left corner of this site. Feel free to contribute by reporting a typo or leaving a pull request at <https://github.com/TrevorFrench/R-for-Data-Analysis>.

1 What is R?

R was a programming language that was designed specifically for the needs of statistics and data analysis. -Hadley Wickham (Hermans 2021)

R is a statistical programming language used commonly for data analysis across a wide array of disciplines and industries. It's often preferred over similar languages for its robust support of statistical analysis, the ease in which one is able to create beautiful graphics, and its open source nature amongst other reasons.

1.1 History

R was built by Ross Ihaka and Robert Gentleman at the University of Auckland and was first released in 1993.

Robert Gentleman and Ross Ihaka “both had an interest in statistical computing and saw a common need for a better software environment in [their] Macintosh teaching laboratory. [They] saw no suitable commercial environment and [they] began to experiment to see what might be involved in developing one [them]selves.” (Ihaka 1998)

While R was officially first released in 1993, it wasn't until 1995 that Ross Ihaka and Robert Gentleman were convinced by Martin Mächler to release the source code freely (Ihaka 1998).

1.2 Resources

- You can learn more about R at <https://www.r-project.org/>
- Read Ross Ihaka's account of R's origination here <https://www.stat.auckland.ac.nz/~ihaka/downloads/Interface98.pdf>
- “What is R?” by Microsoft <https://mran.microsoft.com/documents/what-is-r>
- R manuals by the R Development Core Team <https://cran.r-project.org/manuals.html>
- R-bloggers <https://www.r-bloggers.com/>
- R User Groups <https://www.meetup.com/pro/r-user-groups/>
- R Studio Community <https://community.rstudio.com/>
- The R Journal <https://journal.r-project.org/>
- Microsoft R Application Network <https://mran.microsoft.com/>

2 What is Data Analysis?

I mean my definition is data science is like data analysis by programming. Which of course begs the question of what data analysis is, and so I think of data analysis as really any activity where the input is data and the output is understanding or knowledge or insights. So I think of that pretty broadly. And then to do data science you're not doing it by pointing and clicking. You're doing it by writing some code in a programming language. -Hadley Wickham (Eremenko 2020)

Data analysis at it's most simple form is the process of searching for meaning in data with the ultimate goal being to draw insight from that meaning.

2.1 The Process of Data Analysis

The process of data analysis can be generally described in six steps:

1. **Gathering Requirements** - Before one embarks on an analysis, it's important to make sure the requirements are understood. Requirements include the questions which your stakeholders are hoping to answer as well as the technical requirements of how you are going to perform your analysis.
2. **Data Acquisition** - As you might imagine, you must acquire your data before conducting an analysis. This may be done through the methods such as manual creation of datasets, importing pre-constructed data, or leveraging APIs.
3. **Data Preparation** - Most data will not be received in the precise format you need to begin your analysis. The process of data preparation is where you will structure and add features to your data.
4. **Developing Insights** - Once your data is prepared, you can now begin to make sense of your data and develop insights about it's meaning.
5. **Reporting** - Finally, it's important to report on your data in such a way that the information is able to be digested by the people who need to see it when they need to see it.

Other sources may include additional steps such as “acting on the analysis”. While this is a critical step for organizations to capture the full value of their data, I would argue that it occurs outside of the *analysis* process.

This book will focus on the technical skills required to conduct an analysis. Because of this, we will be covering steps two through five and omitting step one.

2.2 Resources

- Data Science & Big Data Analytics: Discovering, Analyzing, Visualizing and Presenting Data <https://onlinelibrary.wiley.com/doi/book/10.1002/9781119183686>
- Managing the Analytics Life Cycle for Decisions at Scale https://www.sas.com/content/dam/SAS/en_us/doc/whitepaper1/manage-analytical-life-cycle-continuous-innovation-106179.pdf

3 Setup

This chapter will walk you through how to download the R programming language as well as R Studio which is a popular tool for interacting with the R ecosystem. Additionally, there are alternatives to R Studio listed at the end of the chapter. However, R Studio is the recommended environment for completing this book.

3.1 Install R

Before you do anything, you'll need to download R. This download will allow your computer to interpret the R code you write later on.

1. Download R From R: [The R Project for Statistical Computing](#)
2. Select “download R”



[\[Home\]](#)

Download

[CRAN](#)

R Project

[About R](#)

[Logo](#)

[Contributors](#)

[What's New?](#)

[Reporting Bugs](#)

[Conferences](#)

[Search](#)

[Get Involved: Mailing Lists](#)

[Get Involved: Contributing](#)

[Developer Pages](#)

[R Blog](#)

The R Project for Statistical Computing

Getting Started

R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS. To [download R](#), please choose your preferred [CRAN mirror](#).

If you have questions about R like how to download and install the software, or what the license terms are, please read our [answers to frequently asked questions](#) before you send an email.

News

- **R version 4.2.0 (Vigorous Calisthenics)** [prerelease versions](#) will appear starting Tuesday 2022-03-22. Final release is scheduled for Friday 2022-04-22.
- **R version 4.1.3 (One Push-Up)** has been released on 2022-03-10.
- **R version 4.0.5 (Shake and Throw)** was released on 2021-03-31.
- Thanks to the organisers of useR! 2020 for a successful online conference. Recorded tutorials and talks from the conference are available on the [R Consortium YouTube channel](#).
- You can support the R Foundation with a renewable subscription as a [supporting member](#)

3. Choose any link but preferably the one closest to your physical location

USA

<https://mirror.las.iastate.edu/CRAN/>
<http://ftp.usg.ju.edu/CRAN/>
<http://rweb.cmrda.ku.edu/cran/>
<https://repo.miserver.it.umich.edu/cran/>
<http://cran.wustl.edu/>
<https://archive.linux.duke.edu/cran/>
<https://cran.case.edu/>
<https://ftp.osuosl.org/pub/cran/>
<http://lib.stat.cmu.edu/R/CRAN/>
<https://cran.mirrors.hoobly.com/>
<https://mirrors.nics.utk.edu/cran/>
<https://cran.microsoft.com/>

Iowa State University, Ames, IA
Indiana University
University of Kansas, Lawrence, KS
MBNI, University of Michigan, Ann Arbor, MI
Washington University, St. Louis, MO
Duke University, Durham, NC
Case Western Reserve University, Cleveland, OH
Oregon State University
Statlib, Carnegie Mellon University, Pittsburgh, PA
Hoobly Classifieds, Pittsburgh, PA
National Institute for Computational Sciences, Oak Ridge, TN
Revolution Analytics, Dallas, TX

4. Choose your operating system



CRAN
Mirrors
What's new?
Task Views
Search

About R
R Homepage
The R Journal

Software
R Sources
R Binaries
Packages
Other

Documentation
Manuals
FAQs
Contributed

The Comprehensive R Archive Network

Download and Install R

Precompiled binary distributions of the base system and contributed packages, **Windows and Mac** users most likely want one of these versions of R:

- [Download R for Linux \(Debian, Fedora/Redhat, Ubuntu\)](#)
- [Download R for macOS](#)
- [Download R for Windows](#)

R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.

Source Code for all Platforms

Windows and Mac users most likely want to download the precompiled binaries listed in the upper box, not the source code. The sources have to be compiled before you can use them. If you do not know what this means, you probably do not want to do it!

- The latest release (2022-03-10, One Push-Up) [R-4.1.3.tar.gz](#), read [what's new](#) in the latest version.
- Sources of [R alpha and beta releases](#) (daily snapshots, created only in time periods before a planned release).
- Daily snapshots of current patched and development versions are [available here](#). Please read about [new features and bug fixes](#) before filing corresponding feature requests or bug reports.
- Source code of older versions of R is [available here](#).
- Contributed extension [packages](#)

Questions About R

- If you have questions about R like how to download and install the software, or what the license terms are, please read our [answers to frequently asked questions](#) before you send an email.

5. Press “Install R for the first time”



CRAN
Mirrors
What's new?
Task Views
Search

About R
R Homepage
The R Journal

Software
R Sources
R Binaries
Packages
Other

Documentation
Manuals
FAQs
Contributed

R for Windows

Subdirectories:

[base](#)
[contrib](#)
[old contrib](#)
[Rtools](#)

Binaries for base distribution. This is what you want to [install R for the first time](#).
Binaries of contributed CRAN packages (for R ≥ 3.4.x).
Binaries of contributed CRAN packages for outdated versions of R (for R < 3.4.x).
Tools to build R and R packages. This is what you want to build your own packages on Windows, or to build R itself.

Please do not submit binaries to CRAN. Package developers might want to contact Uwe Ligges directly in case of questions / suggestions related to Windows binaries.

You may also want to read the [R FAQ](#) and [R for Windows FAQ](#).

Note: CRAN does some checks on these binaries for viruses, but cannot give guarantees. Use the normal precautions with downloaded executables.

6. Press “download”



CRAN
Mirrors
What's new?
Task Views
Search

About R
R Homepage
The R Journal

Software
R Sources
R Binaries
Packages
Task Views
Other

Documentation
Manuals
FAQs
Contributed

R-4.2.1 for Windows

[Download R-4.2.1 for Windows](#) (79 megabytes, 64 bit)
[README on the Windows binary distribution](#)
[New features in this version](#)

This build requires UCRT, which is part of Windows since Windows 10 and Windows Server 2016. On older systems, UCRT has to be installed manually from [here](#).

If you want to double-check that the package you have downloaded matches the package distributed by CRAN, you can compare the [md5sum](#) of the .exe to the [file's md5sum](#) on the master server.

Frequently asked questions

- [Does R run under my version of Windows?](#)
- [How do I update packages in my previous version of R?](#)

Please see the [R FAQ](#) for general information about R and the [R Windows FAQ](#) for Windows-specific information.

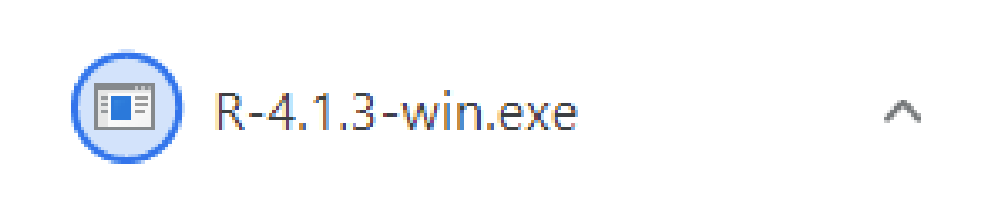
Other builds

- Patches to this release are incorporated in the [r-patched snapshot build](#).
- A build of the development version (which will eventually become the next major release of R) is available in the [r-devel snapshot build](#).
- [Previous releases](#)

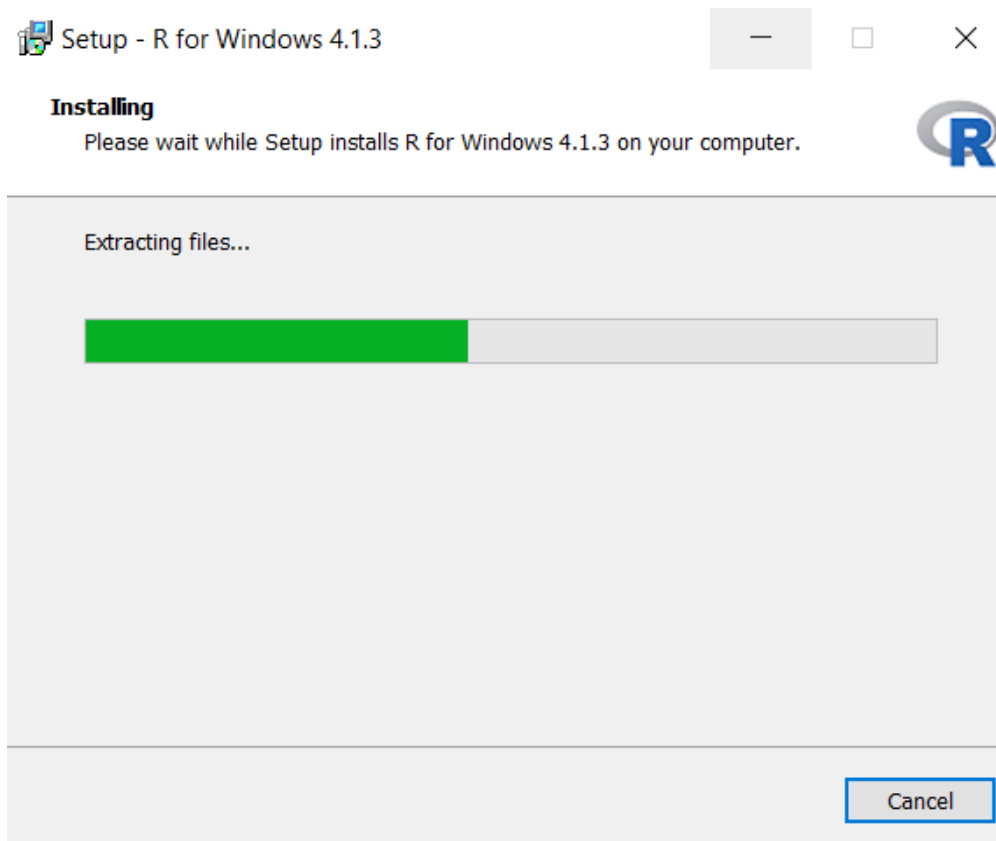
Note to webmasters: A stable link which will redirect to the current Windows binary release is [CRAN_MIRROR::bin/windows/base/release.html](#).

Last change: 2022-06-23

7. Open installer



8. Follow the prompts and leave all options set as their default values



3.2 Install R Studio

After you install R, you'll need an environment to write and run your code in. Most people use a program called "R Studio" for this. To download R Studio follow the steps listed below:

1. Navigate to the R Studio download site: [Download the RStudio IDE](#)

2. Press the “download” button under RStudio Desktop

RStudio Desktop	RStudio Desktop Pro	RStudio Server	RStudio Workbench
Open Source License	Commercial License	Open Source License	Commercial License
Free	\$995 /year	Free	\$4,975 /year (5 Named Users)
Download	Buy	Download	Buy
Learn more	Learn more	Learn more	Evaluation Learn more
Integrated Tools for R	✓	✓	✓
Priority Support	✓		✓
Access to Web Resources		✓	✓

3. Choose the download option for your operating system

RStudio Desktop 2022.02.0+443 - [Release Notes](#)

1. Install R. RStudio requires R 3.3.0+ [↗](#).
2. Download RStudio Desktop. Recommended for your system:



Requires Windows 10/11 (64-bit)



All Installers

Linux users may need to [import RStudio's public code-signing key](#) prior to installation, depending on the operating system's security policy.

RStudio requires a 64-bit operating system. If you are on a 32 bit system, you can use an [older version of RStudio](#).

OS	Download	Size	SHA-256
Windows 10/11	RStudio-2022.02.0-443.exe	176.76 MB	19b870ad
macOS 10.15+	RStudio-2022.02.0-443.dmg	217.18 MB	391d5f18
Ubuntu 18+/Debian 10+	rstudio-2022.02.0-443-amd64.deb	129.00 MB	ad186050

4. Open the installer and accept all defaults



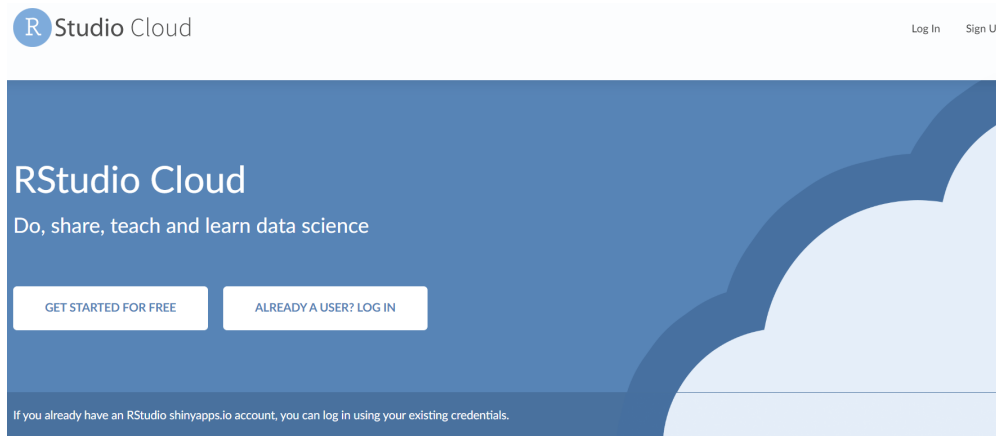
RStudio-2022.02.0...exe



3.3 Alternatives

3.3.1 R Studio Cloud

R Studio Cloud offers users a way to replicate the full R Studio experience without having to download or set anything up on your personal computer. You can sign up for a free account here:



Data science without the hardware hassles

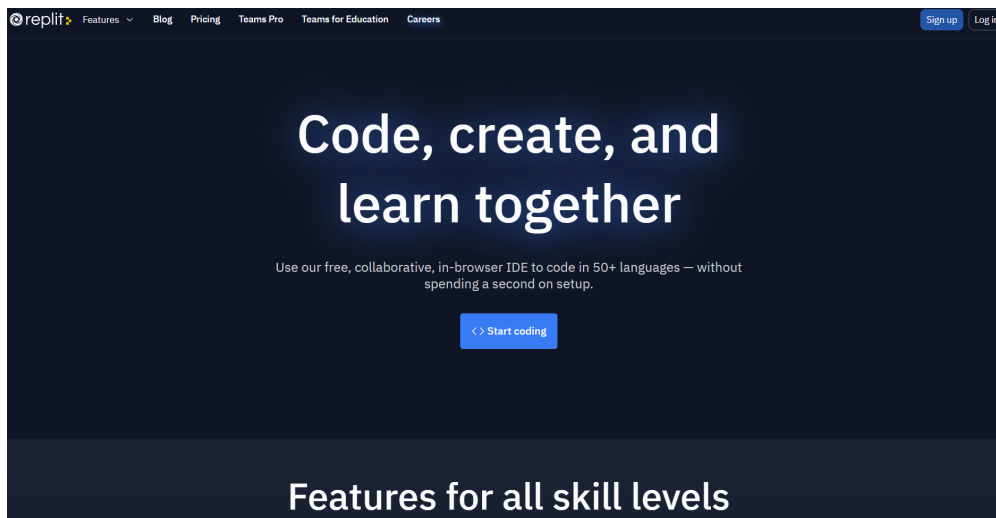
RStudio Cloud is a lightweight, cloud-based solution that allows anyone to do, share, teach

[\\$ AVAILABLE PRICING PLANS](#)

[🔗 RSTUDIO CLOUD GUIDE](#)

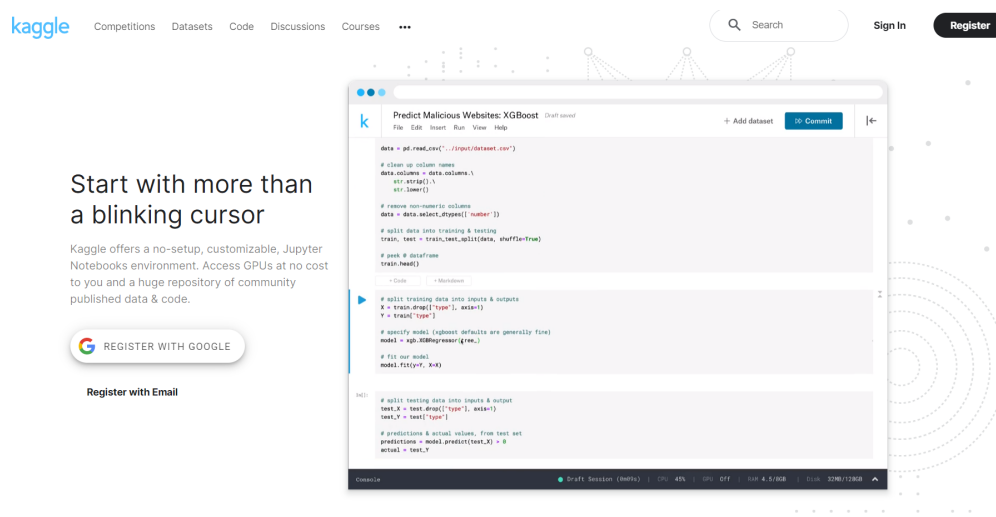
3.3.2 Replit

Replit allows users to code in 50+ languages in the browser. While you won't be able to follow along with the R Studio specific examples, you will be able to run R code. You can sign up for a free account here:



3.3.3 Kaggle

Kaggle is one of the most popular sites for data analysts to compete in data competitions, find data, and discuss data topics. They also have a feature that allows you to write and run R (and Python) code. You can sign up for a free account here:



3.4 Resources

- “R Installation and Administration” by the R Core Team <https://cran.r-project.org/doc/manuals/r-release/R-admin.html>

Part II

Part I: Fundamentals

This section will introduce you to the basics of programming in the context of R. There are four chapters in this book. Each chapter has a brief description listed below. After you have finished reading through each of them, you will have the opportunity to attempt practical exercises to reinforce your newly-gained knowledge.

i Note

For users with a moderate amount of experience in R or another programming language feel free to either skip, skim, or leverage this chapter as a reference guide.

- **Getting Familiar with R Studio-** There are four sections in R Studio. These sections are often referred to as “panes”. This chapter will introduce you to the “source” pane, “console” pane, “environment” pane, and “files” pane. Additionally, you will learn about the different ways you can customize your version of R Studio such as changing the color scheme.
- **Programming Basics-** While the R language certainly has it’s unique advantages, it still leverages principles found in many other programming languages such as functions, comments, and loops. Learn how to apply these and other principles in R.
- **Data Types-** Data is stored differently depending on what it represents when programming. For example, a number is going to be stored as a different data type than a letter is. Learn about the five basic data types in R and how to use them.
- **Data Structures-** In computer science, a data structure refers to the method which one uses to organize their data. There are six basic data structures which are commonly used in R. Learn about each of them in this chapter.

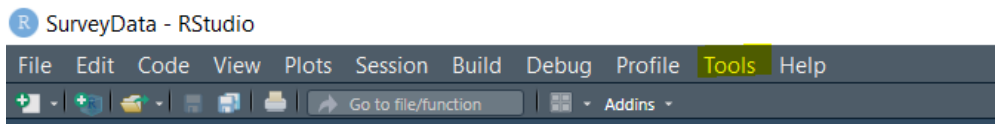
4 Getting Familiar with R Studio

To begin, we are going to walkthrough how to customize your version of R Studio to make it the most comfortable environment for you personally. Following this, we are going to walk through the four panes of R Studio. At a glance, R Studio may seem overwhelming; however, by the end of this chapter you will have learned the essentials needed to embark on your data analysis journey.

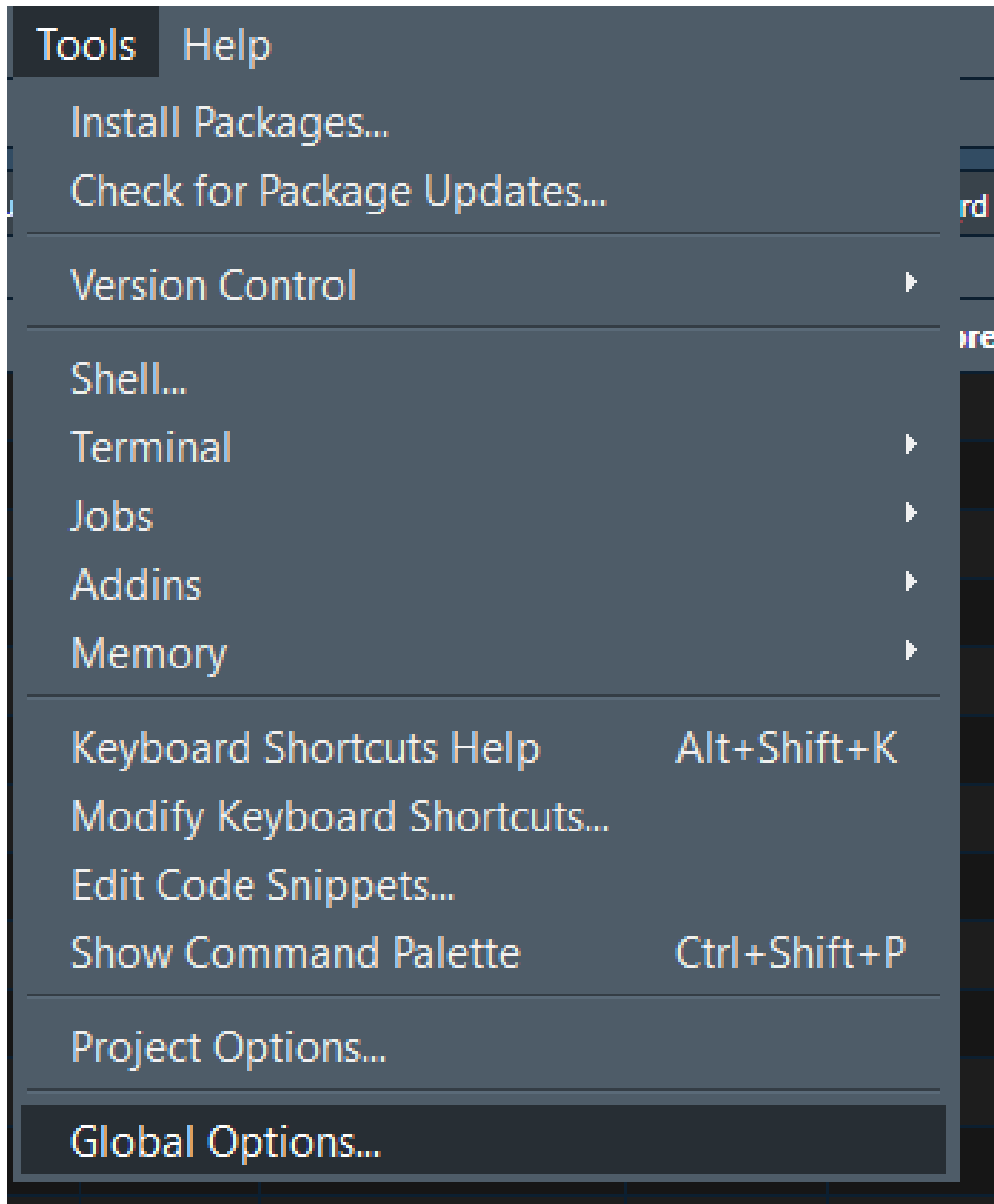
4.1 Customization

You are able to customize how your version of R Studio looks by following these steps:

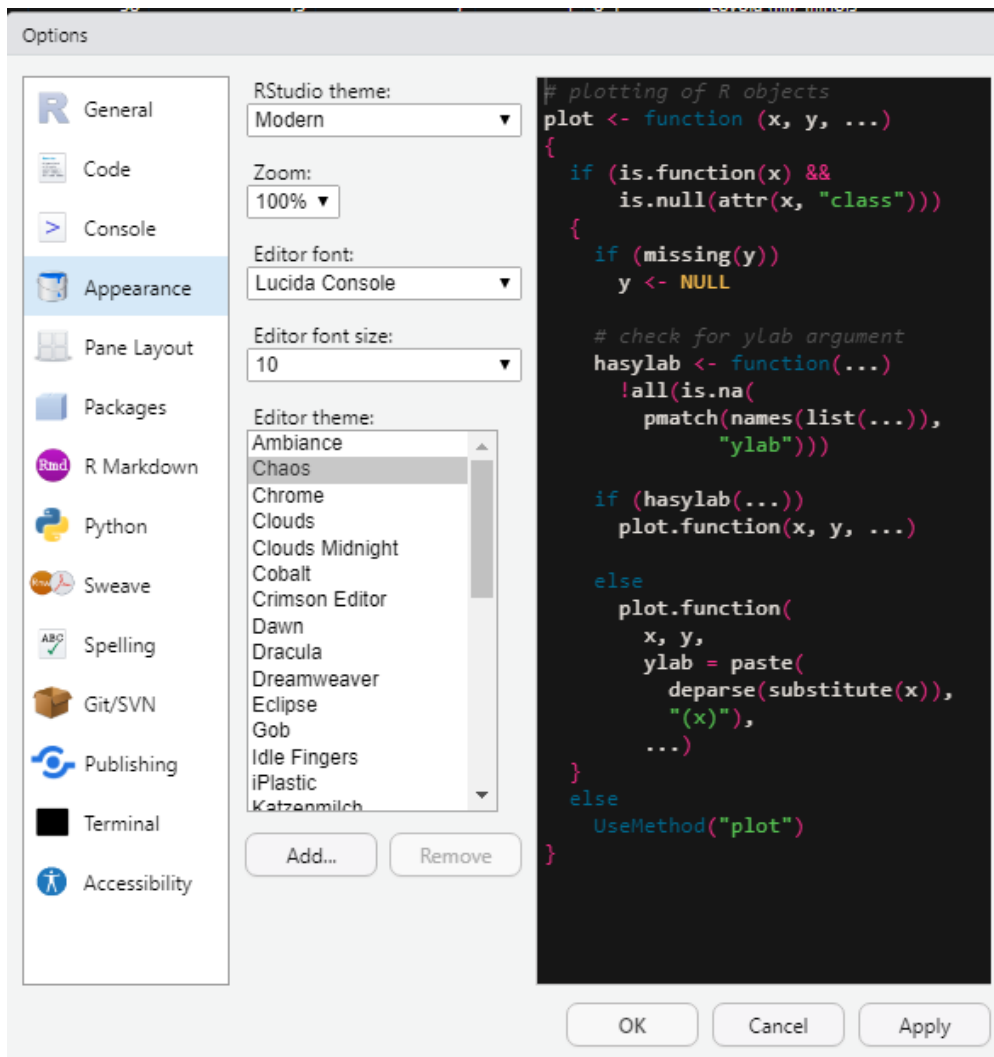
1. Open R Studio and choose ‘tools’ from the toolbar



2. Choose ‘Global Options’



3. Choose 'Appearance' and select your favorite theme from the 'Editor Theme' section

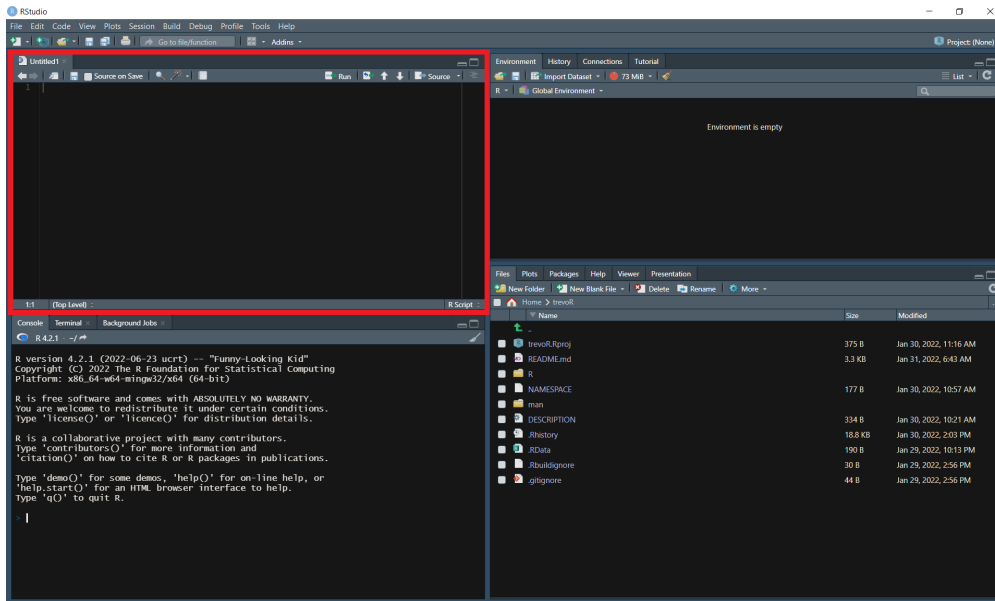


4. Press ‘Apply’

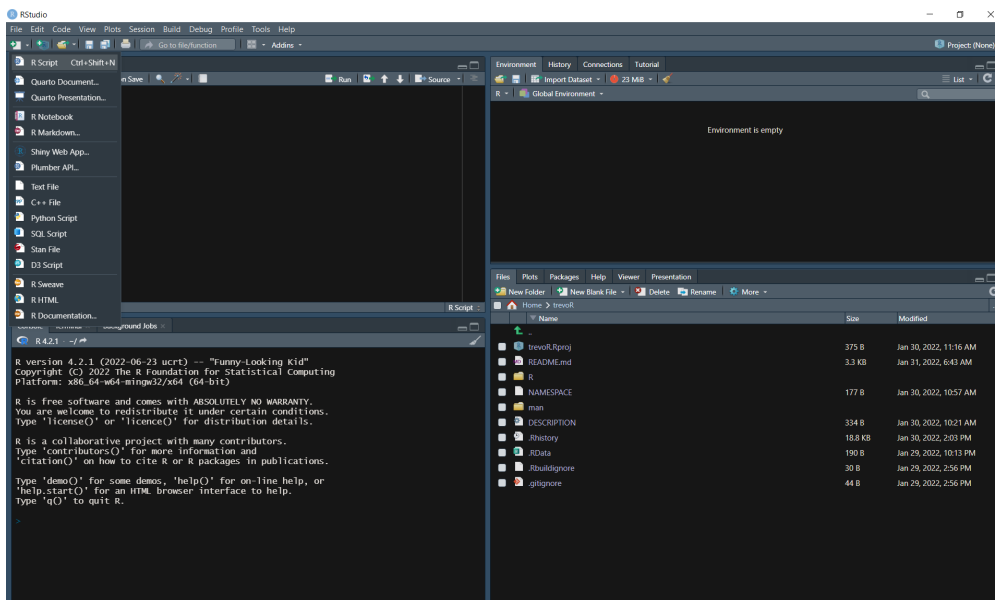
There are other customization options available as well. Feel free to explore the “Global Options” section to make your version of R Studio your own.

4.2 Source Pane

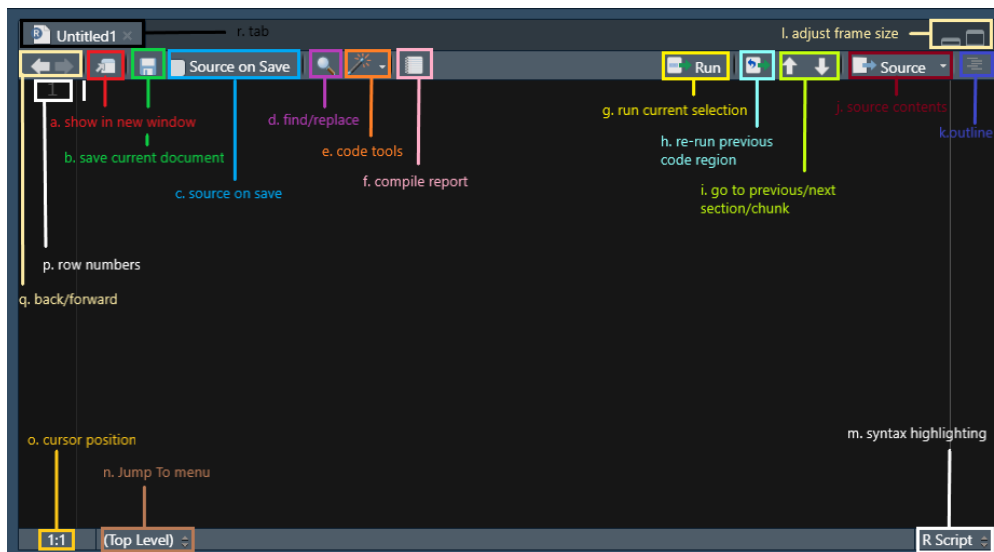
The source pane is the top left pane in R Studio. This is where you will write and edit your code.



If you don't see the source pane, you may need to create a new R Script by pressing “Ctrl + Shift + N” (“Cmd + Shift + N” on Mac) or by selecting “R Script” from the “New File” dropdown in the top left corner.



Each element of the source pane is outlined below.



- a. **Show in New Window**- This allows you to pop the source pane into a new window by itself.
- b. **Save Current Document**- This saves the file contained in the tab you currently have active.
- c. **Source on Save**- Automatically sources your file every time you hit save. “Sourcing” is similar to “Running” in the sense that both will execute your code; however, sourcing will execute your saved file rather than sticking lines of code into the console.
- d. **Find/Replace**- this feature allows you to find and replace specified text, similar to find and replace features in other tools such as Excel.
- e. **Code Tools**- This brings up a menu of options which help you to code more efficiently. Some of these tools include formatting your code and help with function definitions.
- f. **Compile Report**- This allows you to compile a report directly from an R script without needing to use additional frameworks such as R Markdown.
- g. **Run Current Selection**- This allows you to highlight a portion of your code and run only that portion.
- h. **Re-run Previous Code Region**- This option will execute the last section of code that you ran.
- i. **Go to Previous/Next Section/Chunk**- These up and down arrows allow you to navigate through sections of your code without needing to scroll.
- j. **Source Contents**- This option will save your active document if it isn’t already saved and then source the file.
- k. **Outline**- Pressing this option will pop open an outline of your current file.
- l. **Adjust Frame Size**- These two options will adjust the size of the source pane inside of R Studio.
- m. **Syntax Highlighting**- This allows you to adjust the syntax highlighting of your active document to match the highlighting of other file types.
- n. **“Jump To” Menu**- This menu allows you to quickly jump to different sections of your

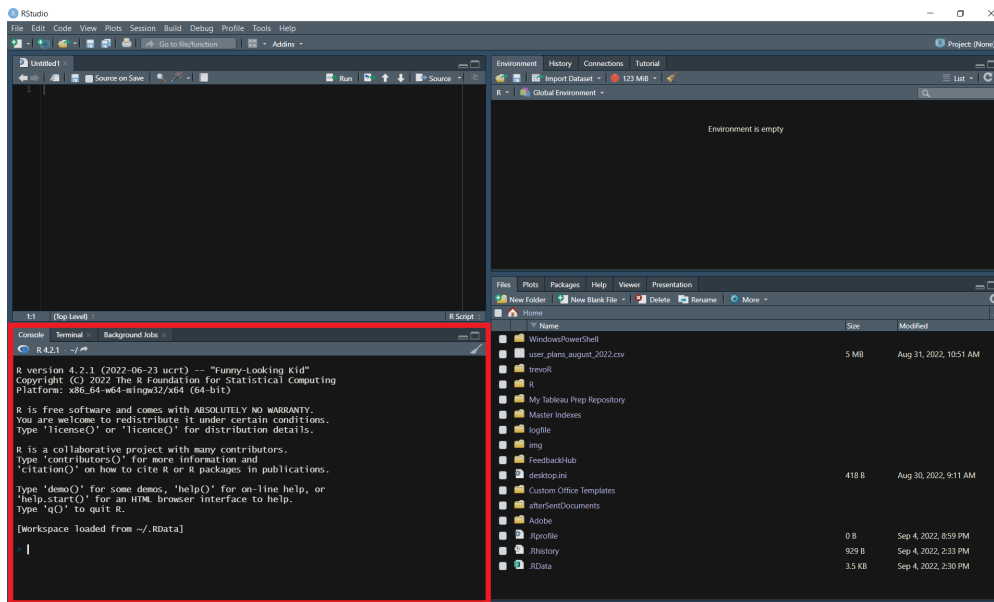
code.

- o. **Cursor Position**- This displays your current cursor position by row and column.
- p. **Row Numbers**- The left-hand side of your document will display the row number for each line of your code.
- q. **Back/Forward**- These arrows are navigation tools that will allow you to redo/undo the following actions: opening a document (or switching tabs), going to a function definition, jumping to a line, and jumping to a function using the function menu (Paulson 2022).
- r. **Tab**- This is a tab in the traditional sense, meaning you are able to have a collection of documents open displayed as tabs. These tabs will have the title of your document and often an icon of some sort to demonstrate the file type.

4.3 Console

The console pane is the bottom left pane in R Studio. This pane has three tabs: “Console”, “Terminal”, and “Background Jobs”.

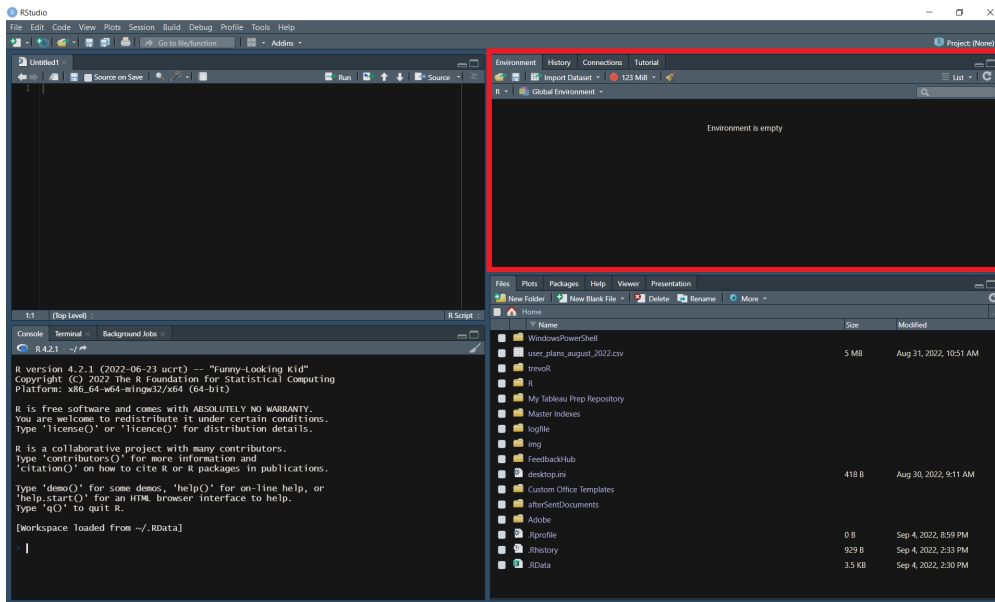
- The “Console” tab is where you will be able to run R code directly without writing a script (this will be covered in the next chapter).
- The “Terminal” tab is the same terminal you have on your computer. This can be adjusted in the global options.
- The “Background Jobs” tab is where you can start and manage processes that need to run behind the scenes.



4.4 Environment

The environment pane is the top right pane in R Studio. This is where you will manage all things related to your development environment. This pane has four tabs: “Environment”, “History”, “Connections”, and “Tutorial”.

- The “Environment” tab will display all information relevant to your current environment. This includes data, variables, and function. This is also the place where you can view and manage your memory usage as well as your workspace.
- The “History” tab allows you to view the history of your executed code. You can search through these commands and even select and re-execute them.
- The “Connections” tab is where you can create and manage connections to databases.
- The “Tutorial” tab delivers tutorials powered by the “learnr” package.

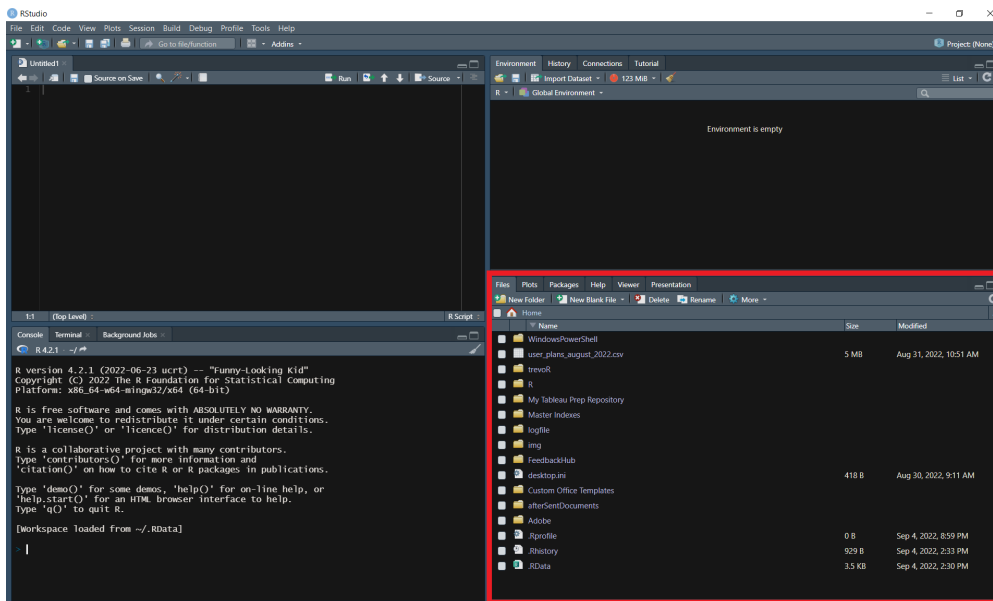


4.5 Files

The files pane is the bottom right pane in R Studio. This pane has six tabs: “Files”, “Plots”, “Packages”, “Help”, “Viewer”, and “Presentation”.

- The “Files” tab is a file explorer of sorts. You can view the contents of a directory, navigate to new directories, and manage files here.

- The “Plots” tab is where the output of your generated plots will show up. You can also export your plots from this tab.
- The “Packages” tab allows you to view all available packages within your environment. From this tab, you can read more about each package as well as update and require packages.
- The “Help” tab allows you to search for information about functions to include examples, descriptions, and available parameters.
- The “Viewer” tab is where certain types of content such as quarto documents will be displayed when rendered.
- The “Presentation” tab is similar to the “Viewer” tab except the content type will be presentations.



4.6 Resources

- “Editing and Executing Code in the RStudio IDE” from the R Studio Support team <https://support.rstudio.com/hc/en-us/articles/200484448-Editing-and-Executing-Code>
- “Code Folding and Sections in the RStudio IDE” from the R Studio Support team <https://support.rstudio.com/hc/en-us/articles/200484568-Code-Folding-and-Sections-in-the-RStudio-IDE>
- “Keyboard Shortcuts in the RStudio IDE” from the R Studio Support team <https://support.rstudio.com/hc/en-us/articles/200711853-Keybaord-Shortcuts-in-the-RStudio-IDE>

- “Navigating Code in the RStudio IDE” from the R Studio Support team <https://support.rstudio.com/hc/en-us/articles/200710523-Navigating-Code-in-the-RStudio-IDE>

5 Programming Basics

This chapter will walk you through how to execute code and write scripts in R. You will then build upon that knowledge by learning about comments, variables, operators, functions, loops, conditionals, and libraries. While this chapter is titled “Programming Basics”, the knowledge you will have learned by the end of this chapter is enough for you to accomplish a huge variety of tasks.

5.1 Executing Code

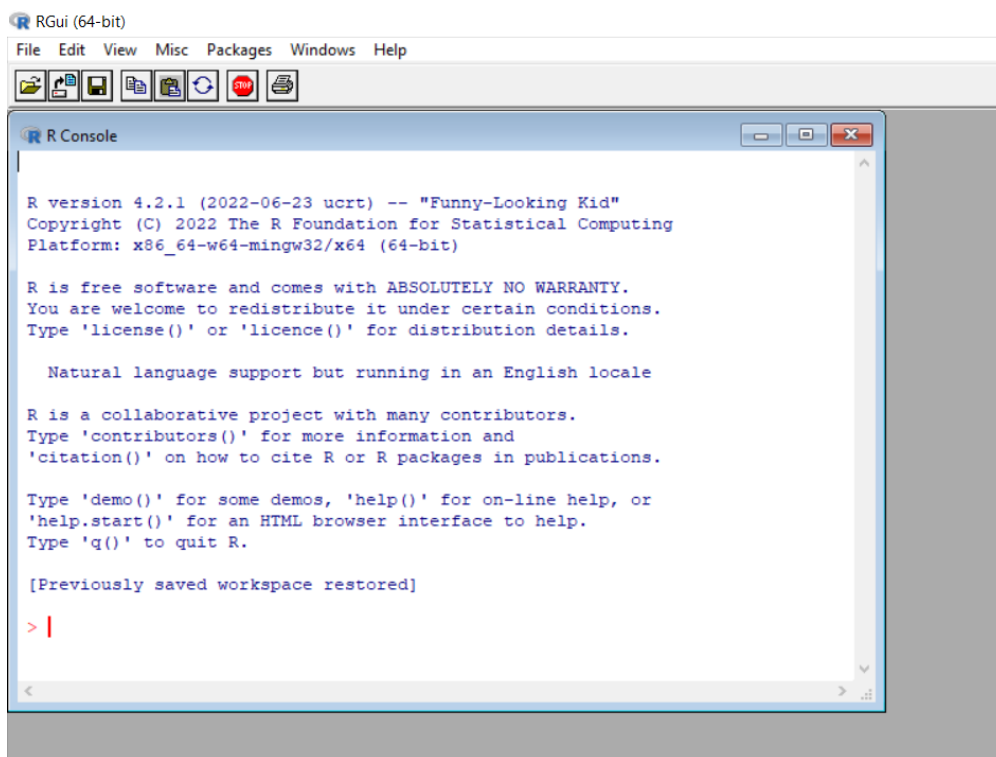
When working in most programming languages, you will generally have the option to execute code one of two ways:

- in the console
- in a script

5.1.1 Console

The first way to run code is directly in the console. If you’re working in R Studio, you will access the console through the “console” pane.

Alternatively, if you downloaded R to your personal computer, you will likely be able to search your machine for an app named “RGui” and access the console this way as well.



In the following example, the text “`print(3+2)`” is typed into the console. The user then presses enter and sees the result: “[1] 5”.

```
print(3+2)
```

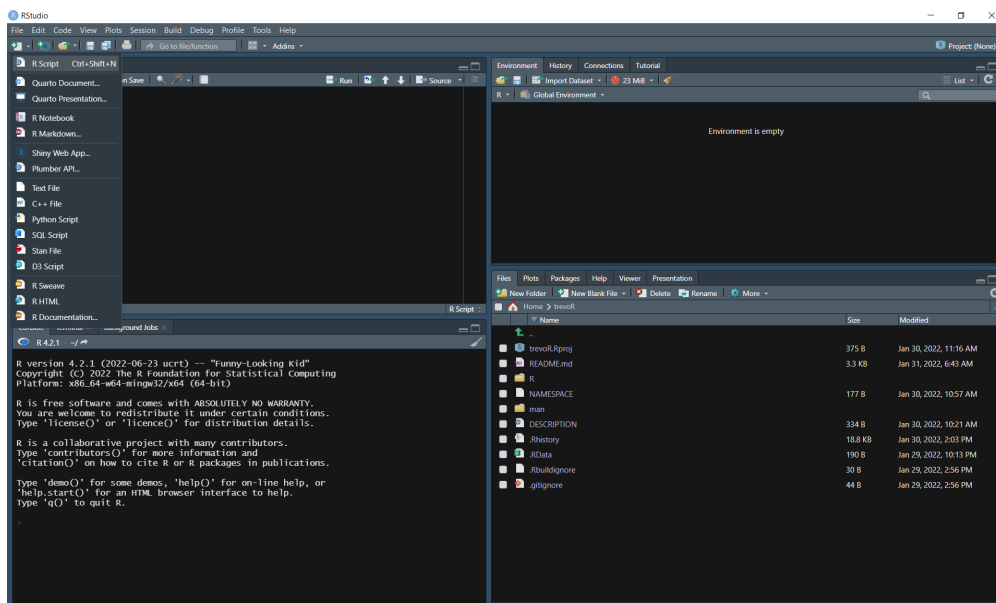
```
[1] 5
```

You may be wondering what “[1]” represents. This is simply a line number in the console and can be ignored for most practical purposes. Additionally, most of the examples in this book will be structured in this way: formatted code immediately followed by the code output.

5.1.2 Script

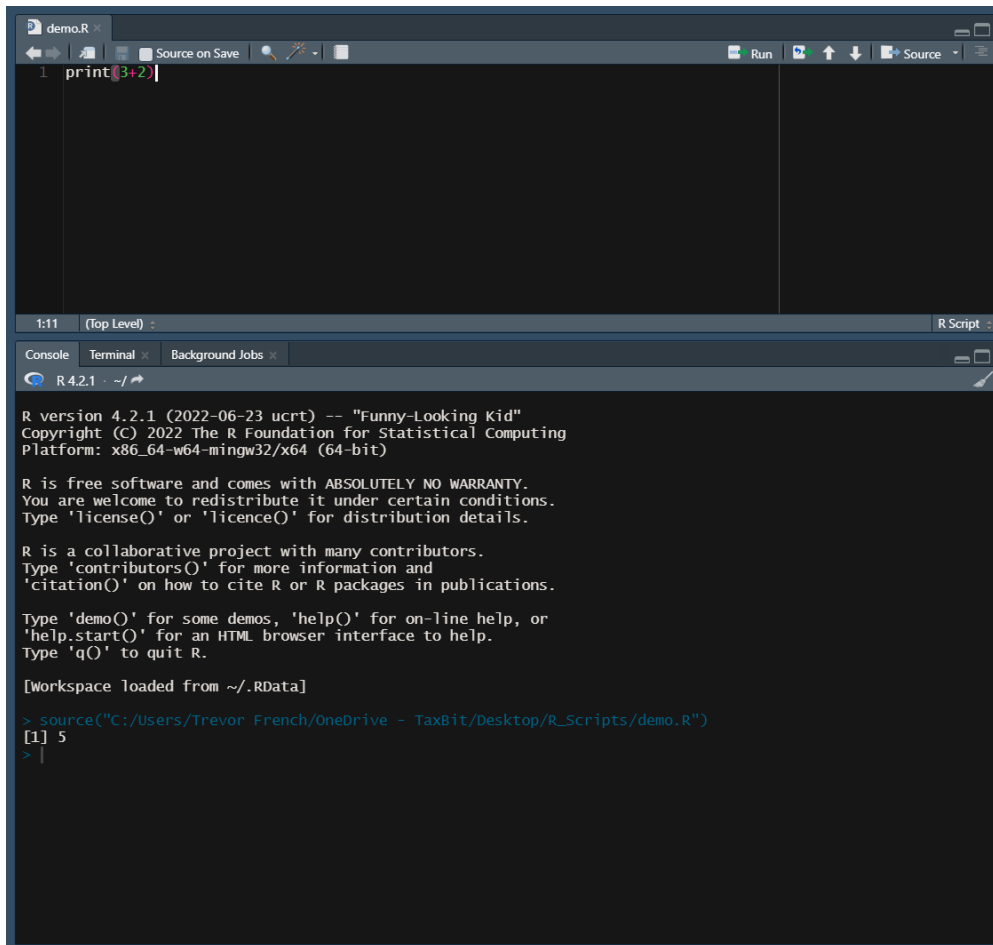
You likely will be using scripts most of the time when working in R. A script is just a file that allows you to type out longer sequences of code and execute them all at once.

For those of you following along in R Studio, you can create a script by pressing “Ctrl + Shift + N” on Windows or by selecting “R Script” from the “New File” dropdown in the top left corner.



From here you can type the same command from before into the source pane. Next, you'll want to save your file by pressing "Ctrl + S" on Windows or by selecting "Save" from the "File" dropdown in the top left corner. Now just give your file a name and your file will automatically be saved as a ".R" file.

Finally, run your newly created R script by pressing the "source" button.



5.2 Comments

Comments are present in most (if not all) programming languages. They allow the user to write text in their code that isn't executed or read by computers. Comments can serve many purposes such as notes, instructions, or formatting.

Comments are created in R by using the “#” symbol. Here's an example:

```
# This is a comment  
print(3+2)
```

```
[1] 5
```

Some programming languages allow you a “bulk-comment” feature which allows you to quickly comment out multiple consecutive lines of text. However, in R, there is no such option. Each line must begin with a “#” symbol, as such:

```
# This is the first line of a comment
# This is the second line of a comment
print(3+2)
```

```
[1] 5
```

Comments don’t have to start at the beginning of a line. You are able to start comments anywhere on a line like in this example:

```
print(3+2) # This comment starts mid-line
```

```
[1] 5
```

5.3 Variables

Variables are used in programming to give values to a symbol. In the following example we have a variable named “rate” which is equal to 15, a variable named “hours” which is equal to 4, and a variable named “total_cost” which is equal to rate * hours.

```
rate <- 15
hours <- 4
total_cost <- rate * hours
print(total_cost)
```

```
[1] 60
```

5.4 Operators

An operator is a symbol that allows you to perform an action or define some sort of logic. The following image demonstrates the operators that are available to you in R.

Operators

Arithmetic	
+	Addition
-	Subtraction
*	Multiplication
/	Division
^	Exponent
%%	Modulus
%/%	Integer Division

Comparison	
==	Equal
!=	Not equal
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

Misc	
:	Creates a series of numbers in a sequence
%in%	Checks if element exists in vector
%*%	Matrix multiplication

Logical	
&	Vectorized AND operator
&&	AND
	Vectorized OR operator
	OR
!	NOT

Assignment	
<-, ->	local
<<-, ->>	global

5.4.1 Arithmetic Operators

Arithmetic operators allow users to perform basic mathematic functions. The examples below demonstrate how these operators might be used. For those not familiar, the modulus operator will return the remainder of a division operation while integer (or Euclidean) division returns the result of a division operation without the fractional component.

```
3 + 3
```

```
[1] 6
```

```
3 - 3
```

```
[1] 0
```

```
3 * 3
```

```
[1] 9
```

```
3 ^ 3
```

```
[1] 27
```

```
10 / 7
```

```
[1] 1.428571
```

```
10 %% 7
```

```
[1] 3
```

```
10 %/% 7
```

```
[1] 1
```

5.4.2 Comparison Operators

Comparison operators allow users to compare values. The examples below demonstrate how these operators might be used.

```
3 == 3
```

```
[1] TRUE
```

```
3 != 3
```

```
[1] FALSE
```

```
3 > 3
```

```
[1] FALSE
```

```
3 < 3
```

```
[1] FALSE
```

```
3 >= 3
```

```
[1] TRUE
```

```
3 <= 3
```

```
[1] TRUE
```

5.4.3 Logical Operators

Logical operators allow users to say “AND”, “OR”, and “NOT”. The following examples demonstrate how these operators might be used in conjunction with comparison operators as well as the difference between standard logical operators and “vectorized” logical operators.

In this example, we will evaluate two vectors of the same length from left to right. Each vector has seven observations (-3, -2, -1, 0, 1, 2, 3). Rather than simply returning a single “TRUE” or “FALSE”, this will return seven “TRUE” or “FALSE” values. In this case, the first element of each vector (“-3” and “-3”) will be evaluated against their respective conditions and return “TRUE” only if both conditions are met. This will then be repeated for each of the remaining elements.

```
# Vectorized "AND" operator
((-3:3) >= 0) & ((-3:3) <= 0)
```

```
[1] FALSE FALSE FALSE  TRUE FALSE FALSE FALSE
```

This example will return a single “TRUE” only if both conditions are met, otherwise “FALSE” will be returned.

```
# Standard "AND" operator
(3 >= 0) && (-3 <= 0)
```

```
[1] TRUE
```

This example is the same as the previous one with the exception that we have negated the second condition with a “NOT” operator.

```
# Standard "AND" operator with "NOT" operator
(3 >= 0) && !(-3 <= 0)
```

```
[1] FALSE
```

The following two examples are essentially the same as the first two except that we are using “OR” operators rather than “AND” operators

```
# Vectorized "OR" operator
((-3:3) >= 0) | ((-3:3) <= 0)
```

```
[1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE
```

```
# Standard "OR" operator  
(3 >= 0) || (-3 <= 0)
```

```
[1] TRUE
```

5.4.4 Assignment Operators

Assignment operators allow users to assign values to something. For most users, only “<-” or “->” will ever be used. These are called local assignment operators. However, there is another type of operator called a global assignment operator which is denoted by “<-” or “<-”.

Understanding the difference between local and global assignment operators in R can be tricky to get your head around. Here’s an example which should clear things up.

First, let’s create two variables named “global_var” and “local_var” and give them the values “global” and “local”, respectively. Notice we are using the standard assignment operator “<-” for both variables.

```
global_var <- 'global'  
local_var <- 'local'
```

```
global_var
```

```
[1] "global"
```

```
local_var
```

```
[1] "local"
```

Next, let’s create a function to test out the global assignment operator (“<-”). Inside this function, we will assign a new value to both of the variables we just created; however, we will use the “<-” operator for the local_var and the “<-” operator for the global_var so that we can observe the difference in behavior.

Note

Functions are covered directly after this section. If the concept of functions is unfamiliar to you, feel free to jump ahead and come back later.

```

my_function <- function() {
  global_var <<- 'na'
  local_var <- 'na'
  print(global_var)
  print(local_var)
}

my_function()

```

```

[1] "na"
[1] "na"

```

This function performs how you would expect it to intuitively, right? The interesting part comes next when we print out the values of these variables again.

```

global_var

```

```

[1] "na"

```

```

local_var

```

```

[1] "local"

```

From this result, we can see the difference in behavior caused by the differing assignment operators. When using the “<-” operator inside the function, it’s scope is limited to just the function that it lives in. On the other hand, the “<<-” operator has the ability to edit the value of the variable outside of the function as well.

You may now be wondering why both the local and the global assignment operators have two separate denotations. The following example demonstrates the difference between the two.

```

x <- 3
3 -> y

x

```

```

[1] 3

```



```
y
```

```
[1] 3
```

There is also a third assignment operator that can be used: “=”. However, it is generally considered best practice to just use the local assignment operator. You can find more information about these three operators in the resources section.

5.4.5 Miscellaneous Operators

The “:” operator allows users to create a series of numbers in a sequence. This was demonstrated in the logical operator section. The `%in%` operator checks if an element exists in a vector. Both of these operators are demonstrated in the following example.

```
3 %in% 1:3
```

```
[1] TRUE
```

Finally, the “`%*%`” operator allows users to perform matrix multiplication as is demonstrated below. First, let’s create a 2x2 matrix and then let’s multiply it by itself.

```
x <- matrix(
  c(1,3,3,7)
  , nrow = 2
  , ncol = 2
  , byrow = TRUE)

x %*% x
```

```
      [,1] [,2]
[1,]    10    24
[2,]    24    58
```

5.5 Functions

Functions allow you to execute a predefined set of commands with just one command. The syntax of functions in R is as follows.

```
# Create a function called function_name
function_name <- function() {
  print("Hello World!")
}

# Call your newly created function
function_name()
```

```
[1] "Hello World!"
```

To go one step further, you can also add “arguments” to a function. Arguments allow you to pass information into the function when it is called. Here’s an example:

```
# Create a function called add_numbers which will add
# two specified numbers together and print the result
add_numbers <- function(x, y) {
  print(x + y)
}

# Call your newly created function twice with different inputs
add_numbers(2, 3)
```

```
[1] 5
```

```
add_numbers(50, 50)
```

```
[1] 100
```

Finally, you can return a value from a function as such:

```
# Create a function called calculate_raise which multiplies
# base_salary and annual_adjustment and returns the result
calculate_raise <- function(base_salary, annual_adjustment) {
  raise <- base_salary * annual_adjustment
  return(raise)
}

# Calculate John's raise
johns_raise <- calculate_raise(90000, .05)
```

```
#Calculate Jane's raise
janes_raise <- calculate_raise(100000, .045)

print("John's Raise:")
```

```
[1] "John's Raise:"
```

```
print(johns_raise)
```

```
[1] 4500
```

```
print("Jane's Raise:")
```

```
[1] "Jane's Raise:"
```

```
print(janes_raise)
```

```
[1] 4500
```

5.6 Loops

There are two types of loops in R: while loops and for loops.

5.6.1 While Loops

While loops are executed as follows:

```
# Set i equal to 1
i <- 1

# While i is less than or equal to three, print i
# The loop will increment the value of i after each print
while (i <= 3) {
  print(i)
  i <- i + 1
}
```

```
}
```

```
[1] 1  
[1] 2  
[1] 3
```

Additionally, you can add 'break' statements to while loops to stop the loop early.

```
i <- 1  
  
while (i <= 10) {  
  print(i)  
  if (i == 5) {  
    print("Stopping halfway")  
    break  
  }  
  i <- i + 1  
}
```

```
[1] 1  
[1] 2  
[1] 3  
[1] 4  
[1] 5  
[1] "Stopping halfway"
```

5.6.2 For Loops

For loops are executed as follows:

```
employees <- list("jane", "john")  
  
for (employee in employees) {  
  print(employee)  
}
```

```
[1] "jane"  
[1] "john"
```

5.7 Conditionals

You are also able to execute a command if a condition is met by using “if” statements.

```
if (2 > 0) {  
  print("true")  
}
```

```
[1] "true"
```

You can add more conditions by adding “else if” statements.

```
if (2 > 3) {  
  print("two is greater than three")  
} else if (2 < 3) {  
  print("two is not greater than three")  
}
```

```
[1] "two is not greater than three"
```

Finally, you can catch anything that doesn’t meet any of your conditions by adding an “else” statement at the end.

```
x <- 20  
if (x < 20) {  
  print("x is less than 20")  
} else if (x > 20) {  
  print("x is greater than 20")  
} else {  
  print("x is equal to 20")  
}
```

```
[1] "x is equal to 20"
```

5.8 Libraries

Libraries allow you to access functions other people have created to perform common tasks.

In this example, we will be installing and loading a common package named “dplyr”.

You first need to install it using the following command.

```
install.packages("dplyr")
```

Next, you will require the package by using this command.

```
library(dplyr)
```

You are now able to access all of the functions available in the dplyr library!

Sometimes users in the R community create their own packages that aren't distributed through the CRAN network. You can still use these libraries but you'll just have to perform an extra step or two. One of the most common places to host packages is [Github](#). The following example will demonstrate how to load a package that I created from Github.

First you'll need to install the "remotes" package. As the name might suggest, this library allows you to access other libraries from *remote* locations.

```
install.packages("remotes")
```

Next you'll need to install the remote package of your choosing. In our case, we'll execute the following code.

```
remotes::install_github("TrevorFrench/trevoR")
```

In the previous example, we used the "install_github" function from the "remotes" package and then specified the Github path of the remote repository by typing "TrevorFrench/trevoR". This code is functionally the same as the "install.packages" function. You may have noticed a new piece of syntax though. The "::" in between "remotes" and "install_github" tells R to use the "install_github" function from the "remotes" library without the need to require the library via the "library" function. This syntax can be used with any other function from any other library.

Now that the remote package is installed, we can require it in the same way we would any other package.

```
library(trevoR)
```

5.9 Resources

- W3 Schools R Tutorial <https://www.w3schools.com/r/>

- Assignment Operators: <https://stat.ethz.ch/R-manual/R-devel/library/base/html/assignOps.html>

6 Data Types

Data is stored differently depending on what it represents when programming. For example, a number is going to be stored as a different data type than a letter is.

There are five basic data types in R:

- **Numeric** - This is the default treatment for numbers. This data type includes integers and doubles.
 - *Double* - A double allows you to store numbers as decimals. This is the default treatment for numbers.
 - *Integer* - An integer is a subset of the numeric data type. This type will only allow whole numbers and is denoted by the letter “L”.
- **Complex** - This type is created by using the imaginary variable “i”.
- **Character** - This type is used for storing non-numeric text data.
- **Logical** - Sometimes referred to as “boolean”, this data type will store either “TRUE” or “FALSE”.
- **Raw** - Used less often, this data type will store data as raw bytes.

6.1 Numeric

6.1.1 Double

Let’s explore the “double” data type by assigning a number to a variable and then check it’s type by using the “typeof” function. Alternatively, we can use the “is.double” function to check whether or not the variable is a double.

```
x <- 6.2
typeof(x)
```

```
[1] "double"
```

```
is.double(x)
```



```
[1] TRUE
```

Next, let's check whether or not the variable is numeric by using the "is.numeric" function.

```
is.numeric(x)
```

```
[1] TRUE
```

This function should return "TRUE" as well, which demonstrates the fact that a double is a subset of the numeric data type.

6.1.2 Integer

Let's explore the "integer" data type by assigning a whole number followed by the capital letter "L" to a variable and then check its type by using the "typeof" function. Alternatively, we can use the "is.integer" function to check whether or not the variable is an integer.

```
x <- 6L
# By using the "typeof" function, we can check the data type of x
typeof(x)
```

```
[1] "integer"
```

```
is.integer(x)
```

```
[1] TRUE
```

Next, let's check whether or not the variable is numeric by using the "is.numeric" function.

```
is.numeric(x)
```

```
[1] TRUE
```

This function should return "TRUE" as well, which demonstrates the fact that an integer is also a subset of the numeric data type.

6.2 Complex

Complex data types make use of the mathematical concept of an imaginary number through the use of the lowercase letter “i”. The following example sets “x” equal to six times i and then displays the type of x.

```
x <- 6i
typeof(x)
```

```
[1] "complex"
```

6.3 Character

```
x <- "Hello!"
typeof(x)
```

```
[1] "character"
```

6.4 Logical

```
x <- TRUE
typeof(x)
```

```
[1] "logical"
```

6.5 Raw

```
x <- charToRaw("Hello!")
print(x)
```

```
[1] 48 65 6c 6c 6f 21
```

```
typeof(x)
```

```
[1] "raw"
```

```
x <- intToBits(6L)
print(x)
```

```
[1] 00 01 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[26] 00 00 00 00 00 00 00 00
```

```
typeof(x)
```

```
[1] "raw"
```

6.6 Resources

- W3 Schools https://www.w3schools.com/r/r_data_types.asp

7 Data Structures

In computer science, a data structure refers to the method which one uses to organize their data. There are six basic data structures which are commonly used in R:

- **Vectors** - Vectors contain of data which is ordered and of the same type.
- **Lists** - Lists are a collection of objects.
- **Matrices** - A matrix is a two-dimensional array where the data is all of the same type.
- **Factors** - Factors are used to designate levels within categorical data.
- **Data Frames** - A data frame contains two-dimensional data where the data can have different types.
- **Arrays** - Arrays are objects which have more than two dimensions (n-dimensional).

7.1 Vectors

We can create a vector by using the “c” function to combine multiple values into a single vector. In the following example, we will combine four separate numbers into a single vector and the output the resulting vector to see what it looks like.

```
x <- c(1, 3, 3, 7)

print(x)
```

```
[1] 1 3 3 7
```

7.2 Lists

```
first_name <- "John"
last_name <- "Smith"
favorite_numbers <- c(1, 3, 3, 7)

person <- list(first_name, last_name, favorite_numbers)
```

```
print(person)
```

```
[[1]]  
[1] "John"  
  
[[2]]  
[1] "Smith"  
  
[[3]]  
[1] 1 3 3 7
```

7.3 Matrices

```
x <- matrix(  
  c(1,3,3,7,1,3,3,7,1,3,3,7)  
  , nrow = 3  
  , ncol = 4  
  , byrow = TRUE)  
  
print(x)
```

```
      [,1] [,2] [,3] [,4]  
[1,]    1    3    3    7  
[2,]    1    3    3    7  
[3,]    1    3    3    7
```

7.4 Factors

```
x <- c("Red", "Blue", "Red", "Yellow", "Yellow")  
  
colors <- factor(x)  
  
print(colors)
```

```
[1] Red    Blue    Red    Yellow Yellow  
Levels: Blue Red Yellow
```

7.5 Data Frames

```
people <- c("John", "Jane")
id <- c(1, 2)
df <- data.frame(id = id, person = people)

print(df)
```

```
  id person
1  1   John
2  2   Jane
```

7.6 Arrays

```
x <- array(
  c(1,3,3,7,1,3,3,7,1,3,3,7)
  , dim = c(1,4,3))

print(x)
```

```
, , 1
```

```
      [,1] [,2] [,3] [,4]
[1,]     1     3     3     7
```

```
, , 2
```

```
      [,1] [,2] [,3] [,4]
[1,]     1     3     3     7
```

```
, , 3
```

```
      [,1] [,2] [,3] [,4]
[1,]     1     3     3     7
```

Exercises

Questions

Write a function called “multiply” that will accept two numbers as arguments and will output the product of those two numbers when called as is demonstrated below.

```
multiply(3, 3)  
  
# [1] 9
```

Answers

One way you could accomplish this task is demonstrated in the following solution.

```
multiply <- function(x, y) {  
  return (x * y)  
}
```

Part III

Part II: Data Acquisition

As you might imagine, you must acquire your data before conducting an analysis. This may be done through the methods such as manual creation of datasets, importing pre-constructed data, or leveraging APIs.

- **Included Datasets-** R comes with a variety of datasets already built in. This chapter will teach you how to view the catalog of included datasets, preview individual datasets, and begin working with the data.
- **Import from Spreadsheets-** Most R users will have to work with spreadsheets at some point in their careers. This chapter will teach you how to import data from spreadsheets whether it's in a .csv or .xlsx file and get the imported data into a format that's easy to work with.
- **Working with APIs-** API stands for Application Programming Interface. These sorts of tools are commonly used to programmatically pull data from a third party resource. This chapter demonstrates how you can begin to leverage these tools in your own workflows.

8 Included Datasets

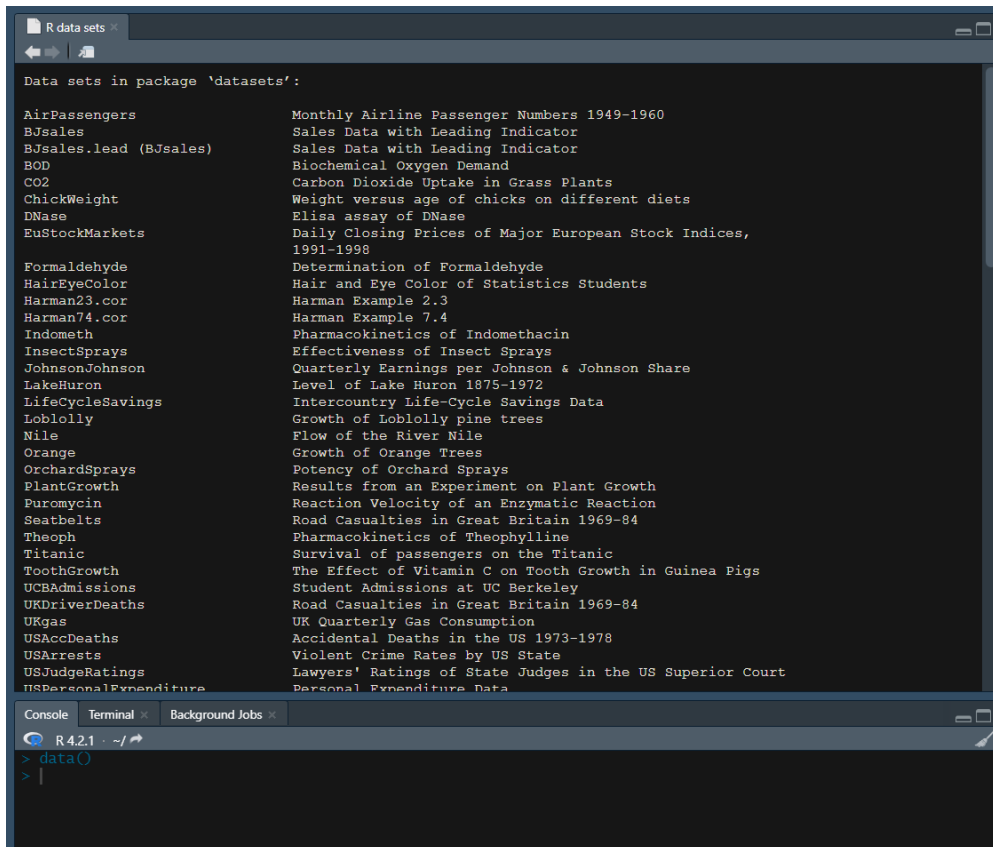
R comes with a variety of datasets already built in. This chapter will teach you how to view the catalog of included datasets, preview individual datasets, and begin working with the data.

8.1 View Catalog

You can view a complete list of datasets available along with a brief description for each one by typing “data()” into your console.

```
data()
```

This will open a new tab in your R Studio instance that looks similar to the following image:



8.2 Working with Included Data

The first step to begin working with your chosen dataset is to load it into your environment by using the “data” function with the quoted name of your dataset inside the parentheses. In the following example, we’ll attach the “iris” dataset to our environment.

i Note

It may not be necessary for you to load your dataset via the “data” function prior to using it. Additionally, some datasets may require you to add them to your search path by using the “attach” function (conversely, you can remove datasets from your search path by using the “detach” function).

```
data("iris")
```

This action will then give us a variable with the same name as our dataset which we can call as we would with any other data structure. Let’s preview the “iris” dataset by using the “head”

function.

```
head(iris)
```

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa

8.3 Resources

- List of datasets available in Base R <https://www.rdocumentation.org/packages/datasets/versions/3.6.2>

9 Import from Spreadsheets

Most R users will have to work with spreadsheets at some point in their careers. This chapter will teach you how to import data from spreadsheets whether it's in a .csv or .xlsx file and get the imported data into a format that's easy to work with. Additionally, this chapter will demonstrate how to import multiple files at once and combine them all into a single dataframe.

9.1 Import from .csv Files

R has a function called “read.csv” which allows you to read a csv file directly to a dataframe. The following code snippet is a simple example of how to execute this function.

Note

It's worth noting that it isn't necessary to store the file path as a variable before calling the function; however, this habit may save you time down the road.

```
input <- "C:/File Location/example.csv"
df <- read.csv(input)
```

Alternatively, if you have multiple files from the same directory that need to be imported, you could do something more like the following code snippet.

```
directory <- "C:/File Location/"
first_file <- paste(directory, "first_file.csv", sep="")
second_file <- paste(directory, "second_file.csv", sep="")
first_df <- read.csv(first_file)
second_df <- read.csv(second_file)
```

9.2 Import from .xlsx Files

Excel files are handled very similarly to CSV files with the exception being that you will need to use the “read_excel” function from the “readxl” library. The following code snippet

demonstrates how to import an Excel file into R.

```
library(readxl)
input <- "C:/File Location/example.xlsx"
df <- read_excel(input)
```

9.3 Import and Combine Multiple Files

You may come across a situation where you have multiple CSV files in a folder which you need combined into a single data frame. The following function from a package I personally created will do just that.

This package exists only on github (rather than being distributed through CRAN) so you'll have to perform an extra step to load the library.

```
install.packages("remotes")
remotes::install_github("TrevorFrench/trevoR")
```

Now that you have the package loaded, you can specify the folder that contains your files and use the “combineFiles” function.

```
wd <- "C:/YOURWORKINGDIRECTORY"
combineFiles(wd)
```

To take this one step further, you can immediately assign the output of the function to a variable name as follows.

```
df <- combineFiles(wd)
```

You now have a dataframe titled “df” which contains all of the data from your .csv files combined!

i Note

All of the headers must match in your CSV files must match exactly for this function to work as expected.

9.4 Resources

- trevoR package documentation <https://github.com/TrevorFrench/trevoR>

10 Working with APIs

API stands for Application Programming Interface. These sorts of tools are commonly used to programmatically pull data from a third party resource. This chapter demonstrates how one can begin to leverage these tools in their own workflows.

10.1 Install Packages

```
install.packages(c('httr', 'jsonlite'))
```

10.2 Require Packages

```
library('httr')  
library('jsonlite')
```

10.3 Make Request

Pass a URL into the 'GET' function and store the response in a variable called 'res'.

```
res = GET("https://api.helium.io/v1/stats")  
print(res)
```

```
Response [https://api.helium.io/v1/stats]  
Date: 2022-08-04 01:25  
Status: 200  
Content-Type: application/json; charset=utf-8  
Size: 922 B
```

10.4 Parse & Explore Data

Use the ‘fromJSON’ function from the ‘jsonlite’ package to parse the response data and then print out the names in the resulting data set.

```
data = fromJSON(rawToChar(res$content))  
  
names(data)  
  
[1] "data"
```

Go one level deeper into the data set and print out the names again.

```
data = data$data  
  
names(data)  
  
[1] "token_supply"      "election_times"    "counts"            "challenge_counts"  "block_times"
```

Alternatively, you can loop through the names as follows.

```
for (name in names(data)){print(name)}  
  
[1] "token_supply"  
[1] "election_times"  
[1] "counts"  
[1] "challenge_counts"  
[1] "block_times"
```

Get the ‘token_supply’ field from the data.

```
token_supply = data$token_supply  
  
print(token_supply)  
  
[1] 124675821
```


10.5 Adding Parameters to Requests

Add 'min_time' and 'max_time' as parameters on a different endpoint and print the resulting 'fee' data.

```
res = GET("https://api.helium.io/v1/dc_burns/sum",
          query = list(min_time = "2020-07-27T00:00:00Z"
                        , max_time = "2021-07-27T00:00:00Z"))

data = fromJSON(rawToChar(res$content))
fee = data$data$fee
print(fee)
```

```
[1] 10112755000
```

10.6 Adding Headers to Requests

Execute the same query as above except this time specify headers. This will likely be necessary when working with an API which requires an API Key.

```
res = GET("https://api.helium.io/v1/dc_burns/sum",
          query = list(min_time = "2020-07-27T00:00:00Z"
                        , max_time = "2021-07-27T00:00:00Z"),
          add_headers(`Accept`='application/json', `Connection`='keep-live'))

data = fromJSON(rawToChar(res$content))
fee = data$data$fee
print(fee)
```

```
[1] 10112755000
```

10.7 Resources

- Blog post by Trevor French <https://medium.com/trevor-french/api-calls-in-r-136290ead81d>

10.7.1 Helpful APIs

Exercises

Questions

text

x

Answers

text

x

Part IV

Part III: Data Preparation

Should we add a feature engineering chapter to this part?

Most data will not be received in the precise format you need to begin your analysis. The process of data preparation is where you will structure and add features to your data.

- **Data Cleaning-** This chapter will cover the basics of cleaning your data including renaming variables, splitting text, replacing values, dropping columns, and dropping rows. These basic actions will be essential to preparing your data prior to developing insights.
- **Handling Missing Data-** You may encounter situations while analysing data that some of your data are missing. This chapter will cover best practices in regards to handling these situations as well as the technical details on how to remedy the data.
- **Outliers-** Outliers are observations that fall outside the expected scope of the dataset. It's important to identify outliers in your data and determine the necessary treatment for them before moving into the next analysis phase.
- **Organizing Data-** This chapter will focus on sorting, filtering, and grouping your datasets.

11 Data Cleaning

This chapter will cover the basics of cleaning your data including renaming variables, splitting text, replacing values, dropping columns, and dropping rows. These basic actions will be essential to preparing your data prior to developing insights.

11.1 Renaming Variables

Let's begin by creating a dataset we can use to work through some examples. In our case, we'll take the first few rows from the "iris" dataset and create a new dataframe called "df".

```
df <- head(iris)
print(df)
```

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa

Now, let's adjust our column names (otherwise known as variables) to be "snake case" (a method to name variables in which all words are lowercase and separated by underscores). We'll do this through the use of the "colnames" function. In the following example, we are renaming each column individually by specifying what number column to adjust.

```
colnames(df)[1] <- "sepal_length"
colnames(df)[2] <- "sepal_width"
colnames(df)[3] <- "petal_length"
colnames(df)[4] <- "petal_width"
colnames(df)[5] <- "species"
```

sepal_length	sepal_width	petal_length	petal_width	species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa

Let's adjust our column names again but to be "camel case" this time. Camel casing requires the first word in a name to be lowercase with all subsequent words having the first letter capitalized. Instead of using the column number though, this time we'll use the actual name of the column we want to adjust.

```
colnames(df)[colnames(df) == "sepal_length"] <- "sepalLength"
colnames(df)[colnames(df) == "sepal_width"] <- "sepalWidth"
colnames(df)[colnames(df) == "petal_length"] <- "petalLength"
colnames(df)[colnames(df) == "petal_width"] <- "petalWidth"
```

sepalLength	sepalWidth	petalLength	petalWidth	species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa

Alternatively, you can use the "rename" function from the "dplyr" library.

```
library(dplyr)
df <- rename(df, "plantSpecies" = "species")
```

sepalLength	sepalWidth	petalLength	petalWidth	plantSpecies
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa

11.2 Splitting Text

If you’ve worked in a spreadsheet application before, you’re likely familiar with the “text-to-columns” tool. This tool allows you to split one column of data into multiple columns based on a delimiter. This same functionality is also achievable in R through functions such as the “separate” function from the “tidyr” library.

To test this function out, let’s first require the “tidyr” library and then create a test dataframe for us to use.

```
library(tidyr)
df <- data.frame(person = c("John_Doe", "Jane_Doe"))
```

person
John_Doe
Jane_Doe

We now have a dataframe with one column which contains a first name and a last name combined by an underscore. Let’s now split the two names into their own separate columns.

```
df <- df %>% separate(person, c("first_name", "last_name"), "_")
```

first_name	last_name
John	Doe
Jane	Doe

Let’s break down what just happened. We first declared that “df” was going to be equal to the output of the function that followed by typing “df <-”. Next we told the separate function that it would be altering the existing dataframe called “df” by typing “df %>%”.

We then gave the separate function three arguments. The first argument was the column we were going to be editing, “person”. The second argument was the names of our two new columns, “first_name” and “last_name”. Finally, the third argument was our desired delimiter, “_”.

11.3 Replace Values

We’ll next go over how you can replace specific values in a dataset. Let’s begin by creating a dataset to work with. The following example will create a dataframe which contains student

names and their respective grades on a test.

```
students <- c("John", "Jane", "Joe", "Janet")
grades <- c(83, 97, 74, 27)
df <- data.frame(student = students, grade = grades)
```

student	grade
John	83
Jane	97
Joe	74
Janet	27

Now that our dataset is assembled, let's decide that we're going to institute a minimum grade of 60. To do this we're going to need to replace any grade lower than 60 with 60. The following example demonstrates one way you could accomplish that.

```
df[which(df$"grade" < 60), "grade"] <- 60
```

student	grade
John	83
Jane	97
Joe	74
Janet	60

11.4 Drop Columns

Let's use the "mtcars" dataset to demonstrate how to drop columns

```
df <- head(mtcars)
print(df)
```

	mpg	cyl	displacement	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160	110	3.90	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108	93	3.85	2.320	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258	110	3.08	3.215	19.44	1	0	3	1

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Hornet	18.7	8	360	175	3.15	3.440	17.02	0	0	3	2
Sportabout											
Valiant	18.1	6	225	105	2.76	3.460	20.22	1	0	3	1

Next, we can either drop columns by specifying the columns we want to keep or by specifying the ones we want to drop. The following example will get rid of the “carb” column by specifying that we want to keep every other column.

```
df <- subset(df, select = c(mpg, cyl, disp, hp, drat, wt, qsec, vs, am, gear))
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear
Mazda RX4	21.0	6	160	110	3.90	2.620	16.46	0	1	4
Mazda RX4 Wag	21.0	6	160	110	3.90	2.875	17.02	0	1	4
Datsun 710	22.8	4	108	93	3.85	2.320	18.61	1	1	4
Hornet 4 Drive	21.4	6	258	110	3.08	3.215	19.44	1	0	3
Hornet Sportabout	18.7	8	360	175	3.15	3.440	17.02	0	0	3
Valiant	18.1	6	225	105	2.76	3.460	20.22	1	0	3

Alternatively, let’s try getting rid of the “gear” column directly. We can do this by putting a “-” in front of the “c” function.

```
df <- subset(df, select = -c(gear))
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am
Mazda RX4	21.0	6	160	110	3.90	2.620	16.46	0	1
Mazda RX4 Wag	21.0	6	160	110	3.90	2.875	17.02	0	1
Datsun 710	22.8	4	108	93	3.85	2.320	18.61	1	1
Hornet 4 Drive	21.4	6	258	110	3.08	3.215	19.44	1	0
Hornet Sportabout	18.7	8	360	175	3.15	3.440	17.02	0	0
Valiant	18.1	6	225	105	2.76	3.460	20.22	1	0

One other way you could drop columns if you wanted to use index numbers rather than column names is demonstrated below.

```
df <- df[, -c(1, 3:7)]
```

	cyl	vs	am
Mazda RX4	6	0	1
Mazda RX4 Wag	6	0	1
Datsun 710	4	1	1
Hornet 4 Drive	6	1	0
Hornet Sportabout	8	0	0
Valiant	6	1	0

As you can see, we used the square brackets to select a subset of our dataframe and then pasted our values after the comma to declare that we were choosing columns rather than rows. After that we used the “-” symbol to say that we were choosing columns to drop rather than columns to keep. Finally, we chose to drop columns 1 as well as columns 3 through 7.

11.5 Drop Rows

We are also able to drop rows with the same method we just used to drop columns with the difference being that we would place our values in front of the comma rather than after the comma. For example, if we wanted to drop the first two rows (otherwise known as observations) from our previous dataframe, we could do the following.

```
df <- df[-c(1:2),]
```

	cyl	vs	am
Datsun 710	4	1	1
Hornet 4 Drive	6	1	0
Hornet Sportabout	8	0	0
Valiant	6	1	0

11.6 Resources

- Separate function documentation <https://tidyr.tidyverse.org/reference/separate.html>

12 Handling Missing Data

You may encounter situations while analysing data that some of your data are missing. This chapter will cover best practices in regards to handling these situations as well as the technical details on how to remedy the data. mena input constant val whow to determine the right way to impute/drop columns

12.1 Replacing Nulls/NA's (constant value?)

Explain what Null and NA mean Grades makes sense to replace with 0 because student may have skipped class blah blah (constant value imputation)

Many datasets you encounter will likely be missing data. The temptation may be to immediately disregard these observations; however, it's important to consider what missing data represents in the context of your dataset as well as the context of what your analysis is hoping to achieve. For example, say you are a teacher and you are trying to determine the average test scores of your students. You have a dataset which lists your students names along with their respective test scores. However, you find that one of your students has an "NA" value in place of a test score.

Depending on the context, it may make sense for you to ignore this observation prior to calculating the average score. It could also make sense for you to assign a value of "0" to this student's test score.

```
students <- c("John", "Jane", "x")
scores <- c(1, 2, NA)
df <- data.frame(score = scores, student = students)

print(df)
```

	score	student
1	1	John
2	2	Jane
3	NA	x

12.2 Mean Imputation

Come up with a data example where it makes sense to fill in blanks with an average (business logic where you claim or revert to average) Alternatively perform median imputation (measure of central tendency)

12.3 Multiple Imputation?

12.4 Resources

13 Outliers

Outliers are observations that fall outside the expected scope of the dataset. It's important to identify outliers in your data and determine the necessary treatment for them before moving into the next analysis phase.

might be necessary to impute value, remove row or may be necessary to keep the data besides extreme value ## Finding Outliers

One common first step many people employ when looking for outliers is visualizing their datasets so that extreme values can be quickly identified. This section will briefly cover several common visualizations used to identify outlier; however, each of these plots will be explored more in-depth later in the book.

13.0.1 Box and Whisker Plot

boxplot()

scatter plot bell curve & standard deviation ## Removing Outliers

13.1 Resources

14 Organizing Data

This chapter will focus on sorting, filtering, and grouping your datasets.

14.1 Sorting

Student Scores dataset sort highest to lowest, alphabetical, lowest to highest, random sort.

sort() vs order() vs rank()

```
head(ChickWeight)
```

weight	Time	Chick	Diet
42	0	1	1
51	2	1	1
59	4	1	1
64	6	1	1
76	8	1	1
93	10	1	1

```
df <- ChickWeight  
df <- df[order(df$weight),]
```

	weight	Time	Chick	Diet
196	35	2	18	1
26	39	2	3	1
195	39	0	18	1
293	39	0	27	2
305	39	0	28	2
317	39	0	29	2

```
df <- ChickWeight
df <- df[order(-df$weight),]
```

	weight	Time	Chick	Diet
400	373	21	35	3
399	361	20	35	3
388	341	21	34	3
398	332	18	35	3
232	331	21	21	2
387	327	20	34	3

14.2 Filtering

Filter by grades higher or lower than a certain number, where name starts with or is equal to a value, do an or/and statement.

14.3 Grouping

Add a column that indicates whether each student is male or female and group by gender. Then run stats based off of grouping.

14.4 Resources

Exercises

Questions

text

x

Answers

text

x

Part V

Part IV: Developing Insights

“A learning organization is an organization skilled at creating, acquiring, and transferring knowledge, and at modifying its behavior to reflect new knowledge and insights.” -David A. Garvin (Garvin 1993)

Once your data is prepared, you can now begin to make sense of your data and develop insights about its meaning. For many, this is where the data analysis process becomes the most fulfilling. This is the point where you get to reap what you’ve sown in the previous phases of the data analysis lifecycle.

- **Summary Statistics**- Summary statistics are usually where one starts when beginning to develop insights. You may hear the phrase “Exploratory Data Analysis” (sometimes abbreviated “EDA”) throughout your career. This is the point where you try to get a high-level understanding of your data through methods such as summary statistics.
- **Regression**- Regression is a common statistical technique employed by many to make generalizations as well as predictions about data.
- **Plotting**- This chapter will cover the basics of creating plots in R. It will begin by demonstrating the plotting capabilities available in R “out of the box”. This will be followed by an introduction to “ggplot2” which is one of the most common plotting libraries in R.

15 Summary Statistics

Descriptive Statistics alternatively

Summary statistics are usually where one starts when beginning to develop insights. You may hear the phrase “Exploratory Data Analysis” (sometimes abbreviated “EDA”) throughout your career. This is the point where you try to get a high-level understanding of your data through methods such as summary statistics.

15.1 Quantitative Data

```
mean, sd, var, min, max, median, range, and quantile dat <- ggplot2::mpg summary(dat$hwy)
```

15.1.1 Measures of Central Tendency

15.1.2 Scope of Data

15.2 Qualitative Data

15.2.1 Frequency not necessarily qualitative

15.2.2 Word Cloud

15.3 Resources

16 Regression

Regression is a common statistical technique employed by many to make generalizations as well as predictions about data.

16.1 Linear Regression

`lm(y~x)`

A data frame with 272 observations on 2 variables.

[,1] eruptions numeric Eruption time in mins [,2] waiting numeric Waiting time to next eruption (in mins)

```
head(faithful)
```

eruptions	waiting
3.600	79
1.800	54
3.333	74
2.283	62
4.533	85
2.883	55

```
lm <- lm(faithful$eruptions ~ faithful$waiting)
summary(lm)
```

Call:

```
lm(formula = faithful$eruptions ~ faithful$waiting)
```

Residuals:

Min	1Q	Median	3Q	Max
-1.29917	-0.37689	0.03508	0.34909	1.19329

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)	
(Intercept)	-1.874016	0.160143	-11.70	<2e-16	***
faithful\$waiting	0.075628	0.002219	34.09	<2e-16	***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.4965 on 270 degrees of freedom
Multiple R-squared: 0.8115, Adjusted R-squared: 0.8108
F-statistic: 1162 on 1 and 270 DF, p-value: < 2.2e-16

Maybe plot trendline now similar to what was done in my medium article

16.2 Multiple Regression

```
lm(y ~ x + x, data = input)
```

16.3 Logistic Regression

```
glm(y ~ x, family = binomial, data = mtcars)
```

16.4 Resources

- “Visualizing OLS Linear Regression Assumptions in R” by Trevor French <https://medium.com/trevor-french/visualizing-ols-linear-regression-assumptions-in-r-e762ba7afaff>

17 Plotting

This chapter will cover the basics of creating plots in R. It will begin by demonstrating the plotting capabilities available in R “out of the box”. This will be followed by an introduction to “ggplot2” which is one of the most common plotting libraries in R.

17.1 Base R

Address the fact that certain types of plots have already been covered previously in the book.

17.1.1 Scatter Plot

`plot()`

17.1.2 Box Plot

`boxplot()`

17.1.3 Plot Matrix

`pairs()`

17.1.4 Pie Chart

`pie()`

17.1.5 Bar Plot

`barplot()`

17.1.6 Histogram

`hist()`

17.1.7 Density Plot

`density()`

17.1.8 Dot Chart

`dotchart()`

17.2 ggplot2

17.2.1 Different types of plots?

17.3 Resources

- Resource: <https://ggplot2.tidyverse.org/>

Exercises

Questions

text

x

Answers

text

x

Part VI

Part V: Reporting

“It feels like we’re all suffering from information overload or data glut. And the good news is there might be an easy solution to that, and that’s using our eyes more. So, visualizing information, so that we can see the patterns and connections that matter and then designing that information so it makes more sense, or it tells a story, or allows us to focus only on the information that’s important. Failing that, visualized information can just look really cool.” -David McCandless (McCandless 2010)

Finally, it’s important to report on your data in such a way that the information is able to be digested by the people who need to see it when they need to see it.

- **Spreadsheets-** Spreadsheets are common way to communicate information to stakeholders. This chapter will go over how to export .xlsx and .csv files from R, how to format those spreadsheets, and how to add formulas to them.
- **R Markdown-** R Markdown allows you to create documents in a programmatic fashion that lends itself towards reproducibility. This chapter will cover the different formats that are available in R as well as how to create them.
- **R Notebook-** Some more technical audiences may require reporting which includes your methodology. This is where R Notebooks come in. R Notebooks are a subset of R Markdown documents that allow you to display and execute code directly in your document.
- **R Shiny-** R Shiny is a tool used to develop web applications and is commonly deployed in the use of creating dashboards, hosting static reports, and custom tooling.

18 Spreadsheets

Spreadsheets are common way to communicate information to stakeholders. This chapter will go over how to export .xlsx and .csv files from R, how to format those spreadsheets, and how to add formulas to them.

18.1 Export

18.1.1 Export .csv Files

In order to export a dataframe to a CSV file, you can use the “write.csv” function. This function will accept a dataframe followed by the desired output location of your file. Let’s start by creating a sample dataframe to work with.

```
people <- c("John", "Jane", NA)
id <- c(12, 27, 23)
df <- data.frame(id = id, person = people)
```

id	person
12	John
27	Jane
23	NA

Now, let’s specify the location we want to store the CSV file at and execute the “write.csv” function.

```
output <- "C:/File Location/example.csv"
write.csv(df, output)
```

This will give you a file that looks like the following image.

	A	B	C
1		id	person
2	1	12	John
3	2	27	Jane
4	3	23	NA
5			

You'll notice that the first column contains the row numbers of the dataframe. This can be remedied by setting "row.names" to "FALSE" as follows.

```
write.csv(df, output, row.names = FALSE)
```

This will yield the following result.

	A	B
1	id	person
2	12	John
3	27	Jane
4	23	NA
5		

Finally, you'll notice that one of the names is an "NA" value. You can tell R how to handle these values at the time of exporting your file with the "na" argument. This argument will replace any "NA" values with the value of your choosing. Let's try replacing the "NA" value with "Unspecified".

```
write.csv(df, output, row.names = FALSE, na = "Unspecified")
```

This results in the following output:

	A	B	
1	id	person	
2	12	John	
3	27	Jane	
4	23	Unspecified	
5			

18.1.2 Export .xlsx Files

Excel files are handled very similarly to CSV files with the exception being that you will need to use the “write_excel” function from the “writexl” library. The following code snippet demonstrates how to export your data to an Excel file.

```
library(writexl)
output <- "C:/File Location/example.xlsx"
write_xlsx(df, output)
```

saveworkbook -csvs and xlsx

18.2 Formatting

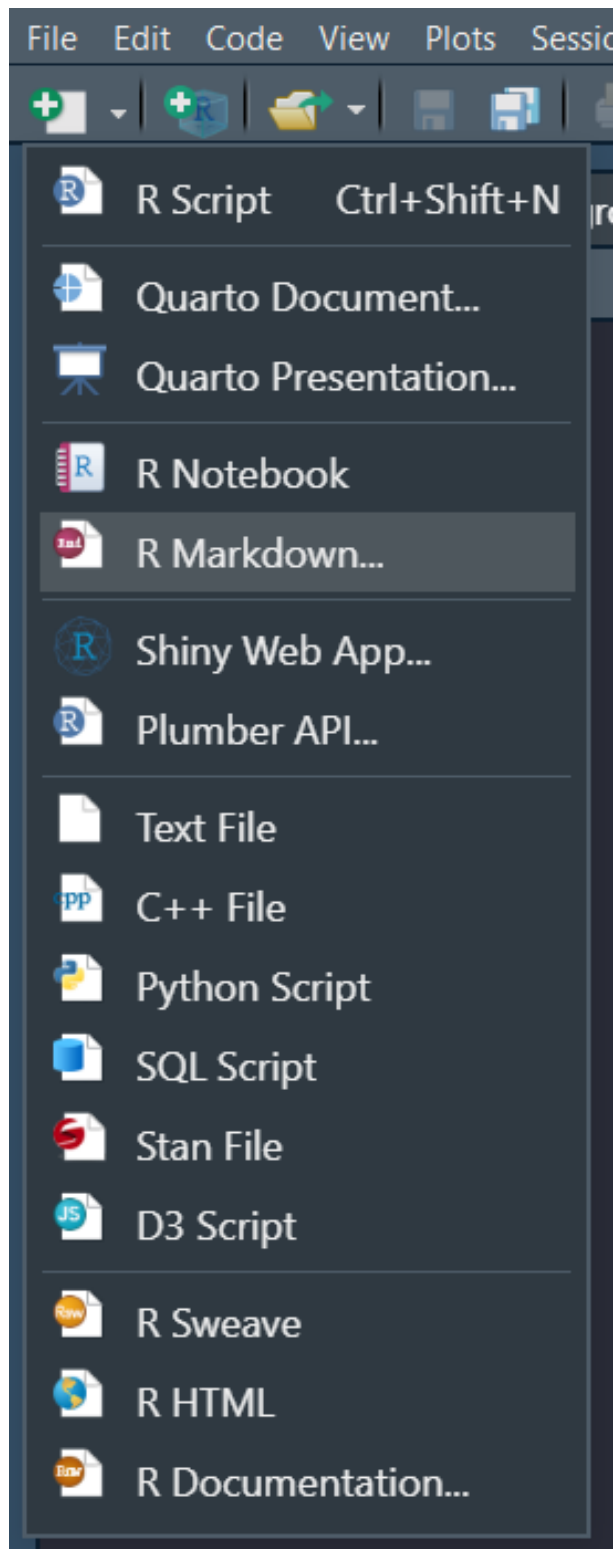
- colors
- tabs
- font
- etc

18.3 Formulas

18.4 Resources

19 R Markdown

R Markdown allows you to create documents in a programmatic fashion that lends itself towards reproducibility. This chapter will cover the different formats that are available in R as well as how to create them.



19.1 Format Options (All of the output options)

19.2 Example (maybe pdf? and another section for html/powerpoint/etc.)

19.3 Including Plots

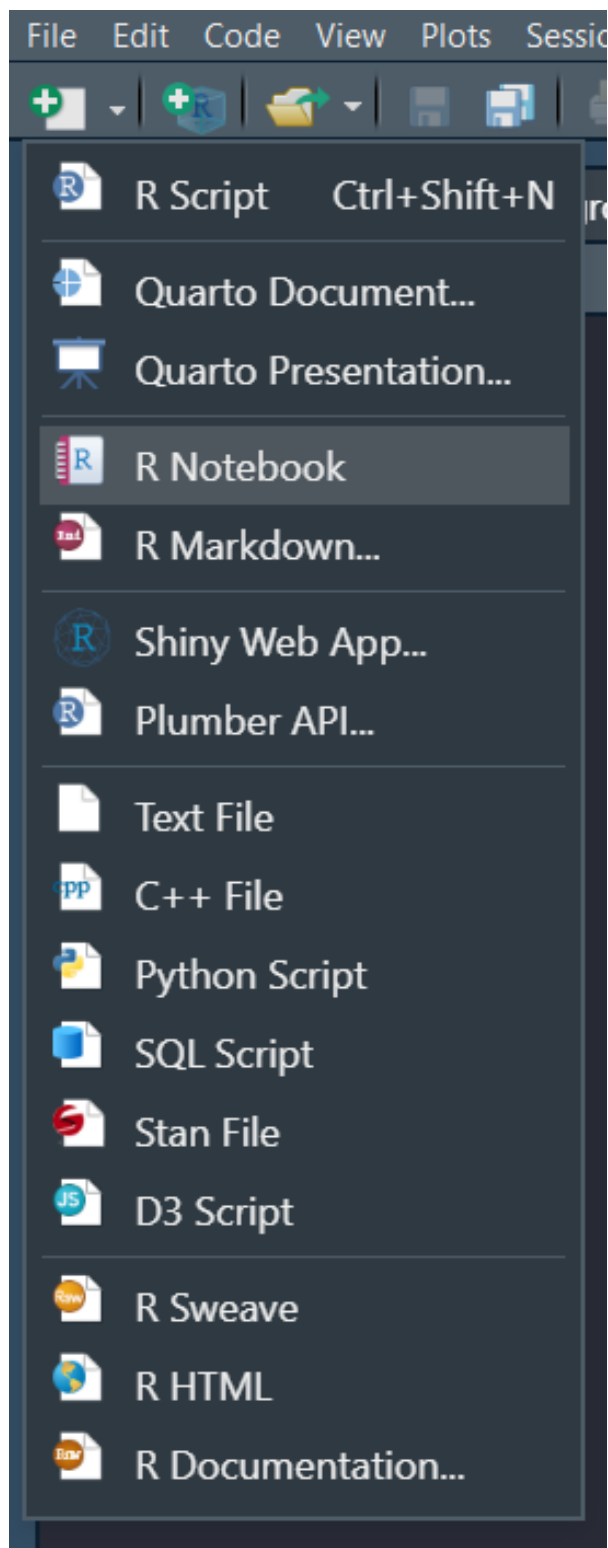
19.4 Resources

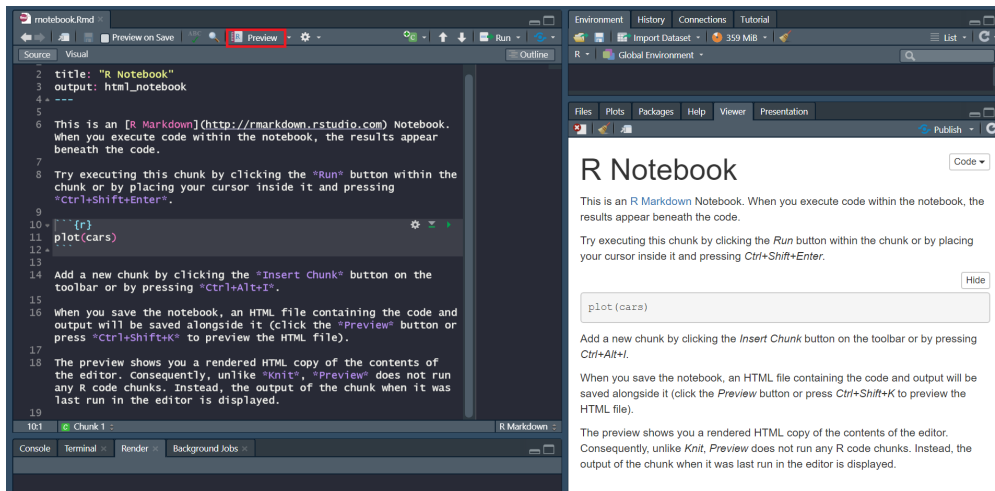
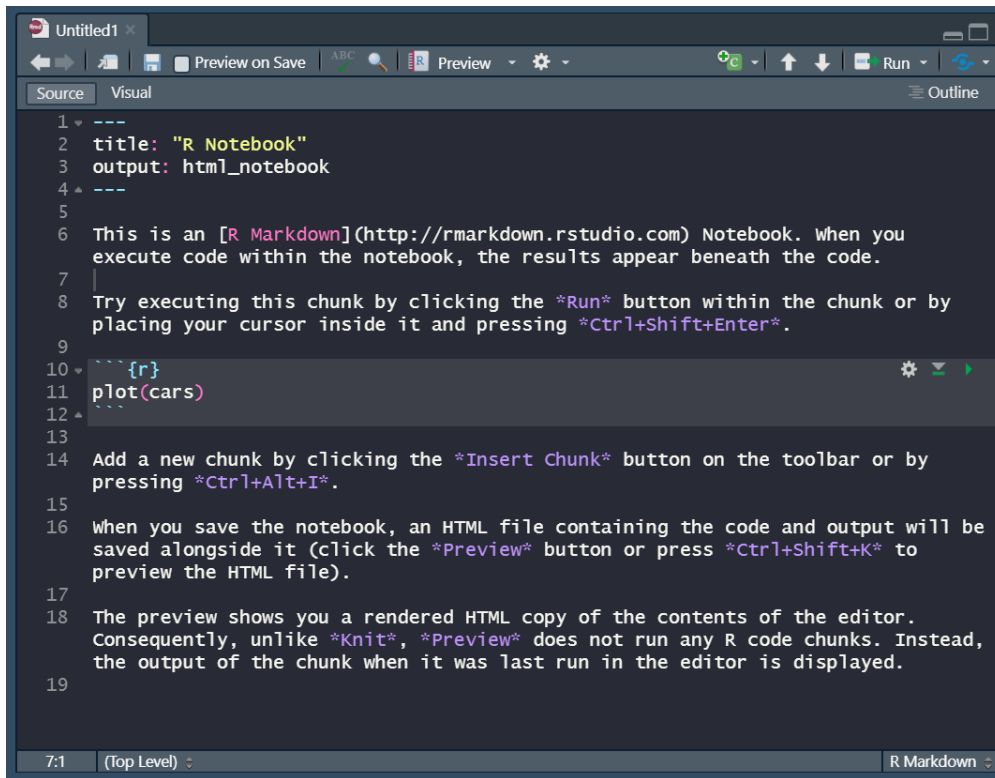
- Formats: <https://rmarkdown.rstudio.com/formats.html>
- Resource: <https://rmarkdown.rstudio.com/>

20 R Notebook

Some more technical audiences may require reporting which includes your methodology. This is where R Notebooks come in. R Notebooks are a subset of R Markdown documents that allow you to display and execute code directly in your document.

- Subset of R Markdown





20.1 Resources

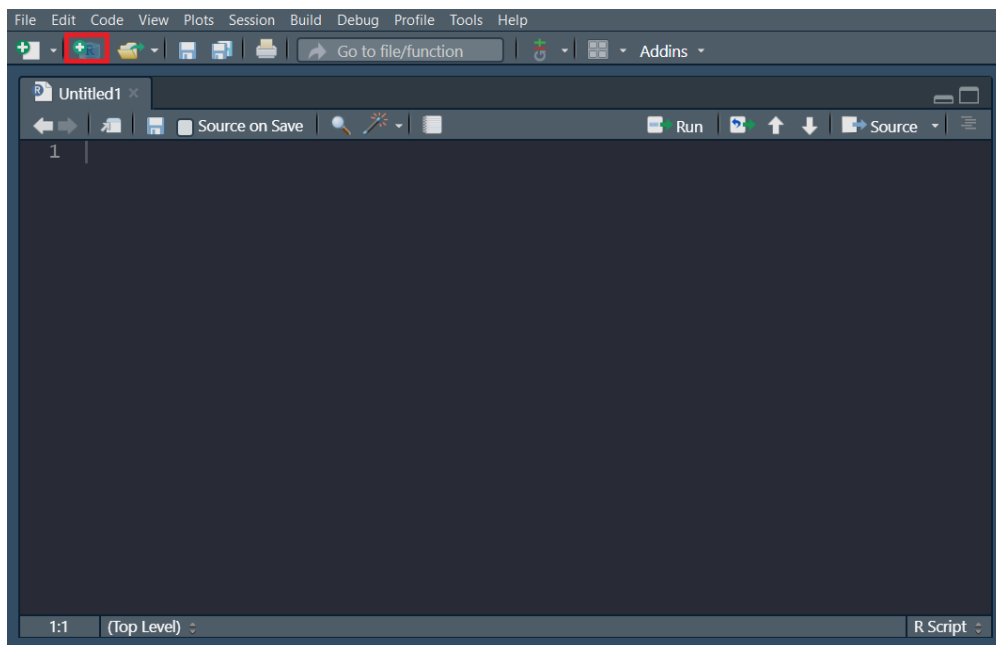
- Resource: <https://rmarkdown.rstudio.com/lesson-10.html>

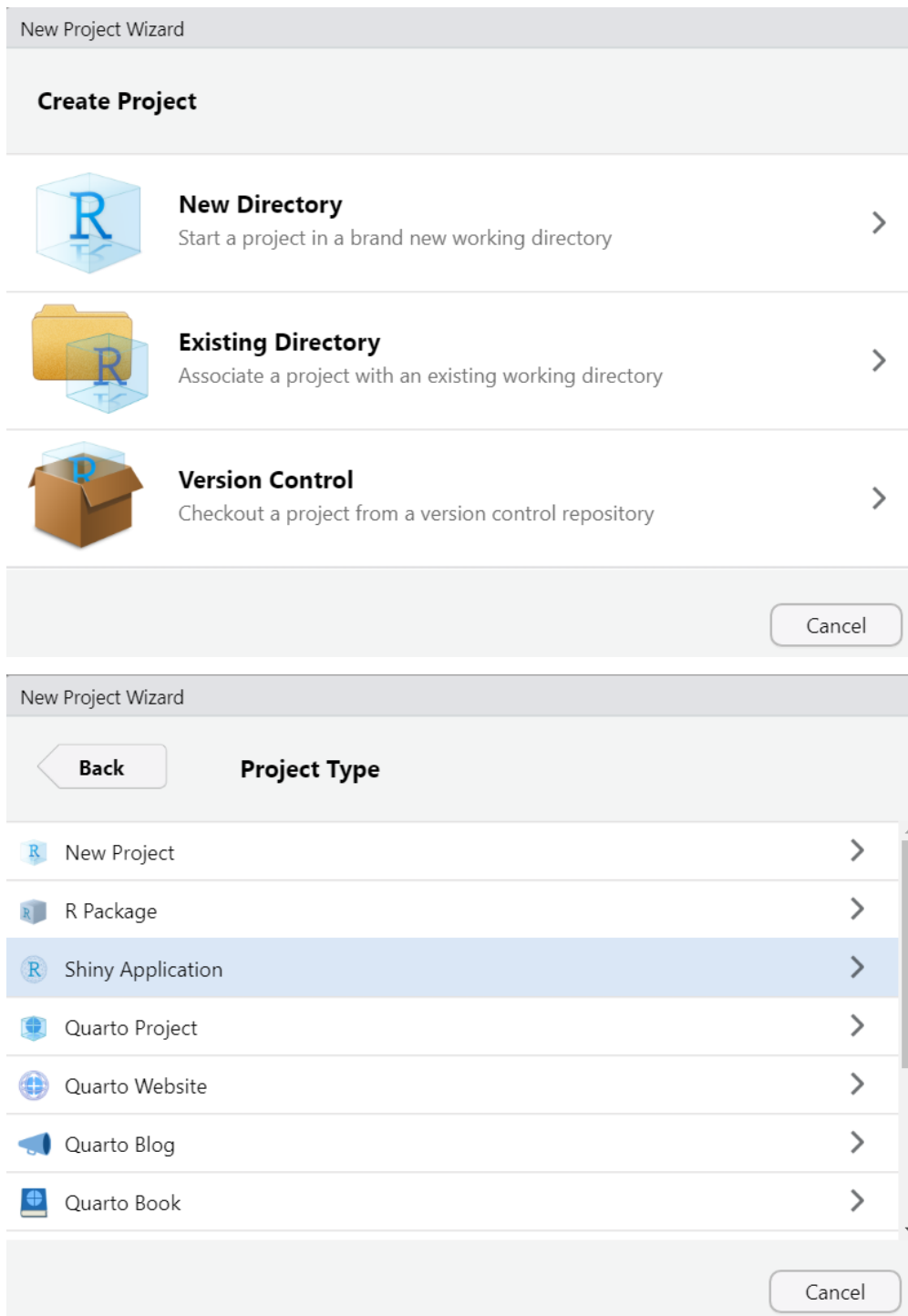
21 R Shiny

R Shiny is a tool used to develop web applications and is commonly deployed in the use of creating dashboards, hosting static reports, and custom tooling.

21.1 Quickstart


Explain what a project is





New Project Wizard

[Back](#) **Create Shiny Application**



Directory name:

Create project as subdirectory of: [Browse...](#)

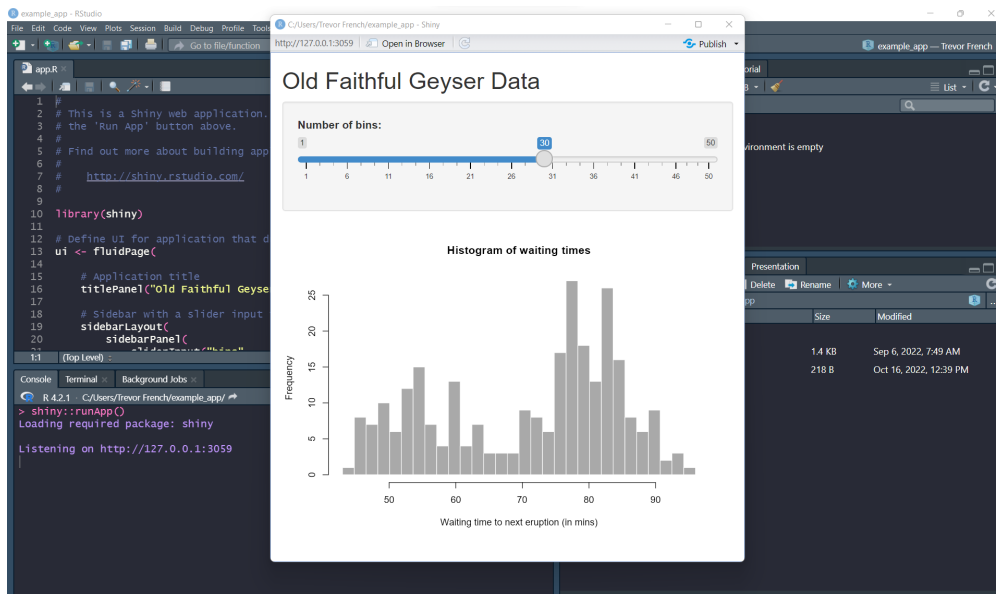
☐ Create a git repository

☐ Use renv with this project

☐ Open in new session

[Create Project](#) [Cancel](#)

```
1 #
2 # This is a Shiny web application. You can run the application by clicking
3 # the 'Run App' button above.
4 #
5 # Find out more about building applications with shiny here:
6 #
7 # http://shiny.rstudio.com/
8 #
9
10 library(shiny)
11
12 # Define UI for application that draws a histogram
13 ui <- fluidPage(
14   # Application title
15   titlePanel("Old Faithful Geyser Data"),
16   # Sidebar with a slider input for number of bins
17   sidebarLayout(
18     sidebarPanel(
19       sliderInput("bins",
20         "Number of bins:",
21         min = 1,
22         max = 50,
23         value = 30)
24     ),
25     # Show a plot of the generated distribution
26     mainPanel(
27       plotOutput("distPlot")
28     )
29   )
30 )
```

21.2 Basic Components of a Shiny Application

21.2.1 Server

21.2.2 UI

21.3 shinyapps.io

- Shiny apps account
- Quick Start

21.4 Deploying Application

21.5 Shinyuieditor?

21.6 Resources

- Resource: <https://shiny.rstudio.com/>

Exercises

Questions

text

x

Answers

text

x

References

- Eremenko, Kirill. 2020. “Hadley Wickham Talks Integration and Future of r and Python [Audio Podcast].” SuperDataScience. <https://www.superdatascience.com/podcast/hadley-wickham-talks-integration-and-future-of-python-and-r>.
- Garvin, David A. 1993. “Building a Learning Organization.” *Harvard Business Review* July-August 1993.
- Hermans, Felienne. 2021. “Hadley Wickham on r and Tidyverse [Audio Podcast].” Software Engineering Radio. <https://www.se-radio.net/2021/03/episode-450-hadley-wickham-on-r-and-tidyverse/>.
- Hofmann, J. R. 1996. *Enlightenment and Electrodynamics*. Cambridge University Press.
- Ihaka, Ross. 1998. “R : Past and Future History.” <https://www.stat.auckland.ac.nz/~ihaka/downloads/Interface98.pdf>.
- McCandless, David. 2010. “The Beauty of Data Visualization.” https://www.ted.com/talks/david_mccandless_the_beauty_of_data_visualization/transcript?language=en.
- Paulson, Josh. 2022. *Navigating Code in the RStudio IDE*. <https://support.rstudio.com/hc/en-us/articles/200710523-Navigating-Code-in-the-RStudio-IDE>.