Daniel Dauber

R for Non-Programmers: A Guide for Social Scientists

Table of contents

References **47**

Welcome

Welcome to *R for Non-Programmers: A Guide for Social Scientists (R4NP)*. This book provides a helpful resource for anyone looking to use *R* for their research projects, regardless of their level of programming or statistical analysis experience. Each chapter presents essential concepts in data analysis in a concise, thorough, and user-friendly manner. *R4NP* will guide you through your first steps on a possibly endless journey full of 'awe and wonder'.

This book is designed to be accessible to beginners while also providing valuable insights for experienced analysts looking to transition to *R* from other computational software. You don't need any prior knowledge or programming skills to get started. *R4NP* provides a comprehensive entry into R programming, regardless of your background.

1

`Readme.` *before you get started*

1.1 A starting point and reference book

My journey with *R* began rather suddenly one night. After many months of contemplating learning *R* and too many excuses not to get started, I put all logic aside and decided to use *R* on my most significant project to date. Admittedly, at the time, I had some knowledge of other programming languages and felt maybe overconfident learning a new tool. This is certainly not how I would recommend learning *R*. In hindsight, though, it enabled me to do things that pushed the project much further than I could have imagined. However, I invested a lot of additional time on top of my project responsibilities to learn this programming language. In many ways, *R* opened the door to research I would not have thought of a year earlier. Today, I completely changed my style of conducting research, collaborating with others, composing my blog posts and writing my research papers.

It is true what everyone told me back then: *R* programming has a steep learning curve and is challenging. To this date, I would agree with this sentiment if I ignored all the available resources. The one thing I wish I had to help me get started was a book dedicated to analysing data from a Social Scientist perspective and which guides me through the analytical steps I partially knew already. Instead, I spent hours searching on different blogs and YouTube to find solutions to problems in my code. However, at no time I was tempted to revert to my trusted statistics software of choice, i.e. SPSS. I certainly am an enthusiast of learning programming languages, and I do not expect that this is true for everyone. Nevertheless, the field of Social Sciences is advancing rapidly and learning at least one programming language can take you much further than you think.

Thus, the aim of this book is narrowly defined: A springboard into the world of *R* without having to become a full-fledged *R* programmer or possess abundant knowledge in other programming languages. This book will guide you through the most common challenges in empirical research in the Social Sciences. Each chapter is dedicated to a common task we have to achieve to answer our research questions. At the end of each chapter, exercises are provided to hone your skills and allow you to revisit key aspects. In addition, the appendix offers

several in-depth case studies that showcase how a research project would be carried out from start to finish in R using real datasets.

This book likely caters to your needs irrespective of whether you are a seasoned analyst and want to learn a new tool or have barely any knowledge about data analysis. However, it would be wrong to assume that the book covers everything you possibly could know about *R* or the analytical techniques covered. There are dedicated resources available to you to dig deeper. Several of these resources are cited in this book or are covered in Chapter @ref(next-steps). The primary focal point of the book is on learning *R* in the context of Social Sciences research projects. As such, it serves as a starting point on what hopefully becomes an enriching, enjoyable, adventurous, and lasting journey.

1.2 Download the companion R package

The learning approach of this book is twofold: Convey knowledge and offer opportunities to practice. Therefore, more than 50% of the book are dedicated to examples, case studies, exercises and code you can directly use yourself. To facilitate this interactive part, this book is accompanied by a so-called 'R package' (see Chapter @ref(r-packages)), which contains all datasets used in this book and enables you to copy and paste any code snippet[1] and work along with the book.

Once you worked through Chapter @ref(setting-up-r-and-rstudio) you can easily download the package `r4np` using the following code snippet in your console (see Chapter @ref(the-console-window)):

```
devtools::install_github("ddauber/r4np")
```

1.3 A 'tidyverse' approach with some basic R

As you likely know, every language has its flavours in the form of dialects. This is no different to programming languages. The chosen 'dialect' of *R* in this book is the `tidyverse` approach. Not only is it a modern way of programming in *R*, but it is also a more accessible entry point. The code written with `tidyverse` reads almost like a regular sentence and, therefore, is much easier to

[1] A *code snippet* is a piece of programming code that can be used directly as is.

read, understand, and remember. Unfortunately, if you want a comprehensive introduction to learning the underlying basic *R* terminology, you will have to consult other books. While it is worthwhile to learn different ways of conducting research in *R*, basic *R* syntax is much harder to learn, and I opted against covering it as an entry point for novice users. After working through this book, you will find exploring some of the *R* functions from other 'dialects' relatively easy, but you likely miss the ease of use from the `tidyverse` approach.

1.4 Understanding the formatting of this book

The formatting of this book carries special meaning. For example, you will find actual *R* code in boxes like these.

```
name <- "Daniel"
food <- "Apfelstrudel"

paste("My name is ", name, ", and I love ", food, ".", sep = "")
```

```
[1] "My name is Daniel, and I love Apfelstrudel."
```

You can easily copy this *code chunk* by using the button in the top-right corner. Of course, you are welcome to write the code from scratch, which I would recommend because it accelerates your learning.

Besides these blocks of code, you sometimes find that certain words are formatted in a particular way. For example, datasets, like `imdb_top_250`, included in the *R* package `r4np`, are highlighted. Every time you find a highlighted word, it refers to one of the following:

- A dataset,
- A variable,
- A data type,
- The output of code,
- The name of an *R* package,
- The name of a function or one of its components.

This formatting style is consistent with other books and resources on *R* and, therefore, easy to recognise when consulting other content, such as those covered in Chapter @ref(next-steps).

2

Why learn a programming language as a non-programmer?

'R', it is not just a letter you learn in primary school, but a powerful programming language. While it is used for a lot of quantitative data analysis, it has grown over the years to become a powerful tool that excels (#no-pun-intended) in handling data and performing customised computations with quantitative and qualitative data.

R is now one of my core tools to perform various types of work, for example,

- Statistical analysis,
- Corpus analysis,
- Development of online dashboards for interactive data visualisations and exploration,
- Connection to social media APIs for data collection,
- Creation of reporting systems to provide individualised feedback to research participants,
- Writing research articles, books, and blog posts,
- etc.

> *Learning R is like learning a foreign language. If you enjoy learning languages, then 'R' is just another one.*

While *R* has become a comprehensive tool for data scientists, it has yet to find its way into the mainstream field of Social Sciences. Why? Well, learning programming languages is not necessarily something that feels comfortable to everyone. It is not like Microsoft Word, where you can open the software and explore it through trial and error. Learning a programming language is like learning a foreign language: You have to learn vocabulary, grammar and syntax. Similar to learning a new language, programming languages also have steep learning curves and require quite some commitment.

For this reason, most people do not even dare to learn it because it is time-consuming and often not considered a *'core method'* in Social Sciences disciplines. Apart from that, tools like SPSS have very intuitive interfaces, which seem much easier to use (or not?). However, the feeling of having *'mastered' R* (although one might never be able to claim this) can be extremely rewarding.

I guess this introduction was not necessarily helpful in convincing you to learn any programming language. However, despite those initial hurdles, there are a series of advantages to consider. Below I list some good reasons to learn a programming language as they pertain to my own experiences.

2.1 Learning new tools to analyse your data is always essential

Theories change over time, and new insights into certain social phenomena are published every day. Thus, your knowledge might get outdated quite quickly. This is not so much the case for research methods knowledge. Typically, analytical techniques remain over many years. We still use the mean, mode, quartiles, standard deviation, etc., to describe our quantitative data. However, there are always new computational methods that help us to crunch the numbers even more. *R* is a tool that allows you to venture into new analytical territory because it is open source. Thousands of developers provide cutting-edge research methods free of charge for you to try with your data. You can find them on platforms like GitHub[1]. *R* is like a giant supermarket, where all products are available for free. However, to read the labels on the product packaging and understand what they are, you have to learn the language used in this supermarket.

2.2 Programming languages enhance your conceptual thinking

While I have no empirical evidence for this, I am very certain it is true. I would argue that my conceptual thinking is quite good, but I would not necessarily say that I was born with it. Programming languages are very logical. Any error in your code will make you fail to execute it properly. Sometimes you face challenges in creating the correct code to solve a problem. Through creative abstract thinking (I should copyright this term), you start to approach your

[1] https://github.com

problems differently, whether it is a coding problem or a problem in any other context. For example, I know many students enjoy the process of qualitative coding. However, they often struggle to detach their insights from the actual data and synthesise ideas on an abstract and more generic level. Qualitative researchers might refer to this as challenges in '*second-order deconstruction of meaning'*. This process of abstraction is a skill that needs to be honed, nurtured and practised. From my experience, programming languages are one way to achieve this, but they might not be recognised for this just yet. Programming languages, especially functions (see Chapter @ref(functions)), require us to generalise from a particular case to a generic one. This mental mechanism is also helpful in other areas of research or work in general.

2.3 Programming languages allow you to look at your data from a different angle

There are commonly known and well-established techniques regarding how you should analyse your data rigorously. However, it can be quite some fun to try techniques outside your discipline. This does not only apply to programming languages, of course. Sometimes, learning about a new research method enables you to look at your current tools in very different ways too. One of the biggest challenges for any researcher is to reflect on one's own work. Learning new and maybe even *'strange'* tools can help with this. Admittedly, sometimes you might find out that some new tools are also a dead-end. Still, you likely have learned something valuable through the process of engaging with your data differently. So shake off the rust of your analytical routine and blow some fresh air into your research methods.

2.4 Learning any programming language will help you learn other programming languages.

Once you understand the logic of one language, you will find it relatively easy to understand new programming languages. Of course, if you wanted to, you could become the next '*Neo'* (from 'The Matrix')[2] and change the reality of your research forever. On a more serious note, though, if you know any programming language already, learning *R* will be easier because you have accrued some basic understanding of these particular types of languages.

[2] https://www.imdb.com/title/tt0133093/?ref_=ext_shr_lnk

Having considered everything of the above, do you feel ready for your next foreign language?

3

Setting up R *and RStudio*

Every journey starts with gathering the right equipment. This intellectual journey is not much different. The first step that every *R* novice has to face is to set everything up to get started. There are essentially two strategies:

- Install *R*[1] and RStudio[2]

or

- Run RStudio in a browser via RStudio Cloud[3]

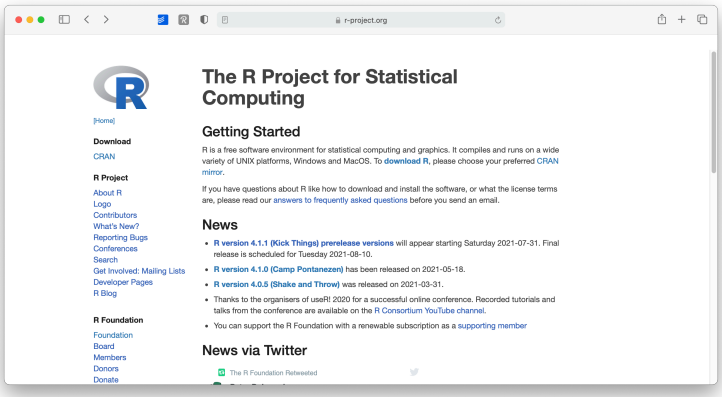
While installing *R* and RStudio requires more time and effort, I strongly recommend it, especially if you want to work offline or make good use of your computer's CPU. However, if you are unsure whether you enjoy learning *R*, you might wish to look at RStudio Cloud first. Either way, you can follow the examples of this book no matter which choice you make.

3.1 Installing R

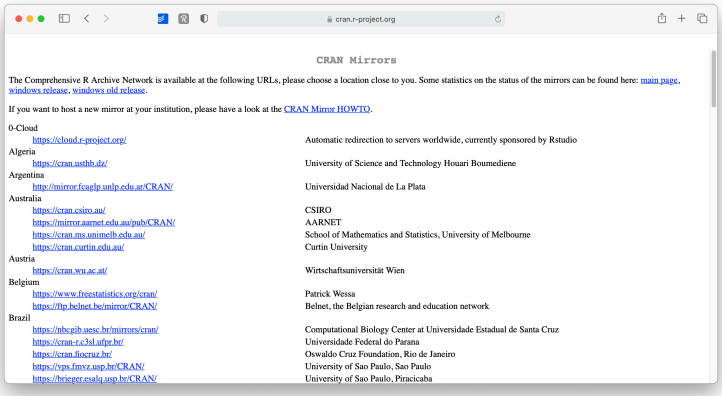
The core module of our programming is *R* itself, and since it is an open-source project, it is available for free on Windows, Mac and Linux computers. So, here is what you need to do to install it properly on your computer of choice:

1. Go to www.r-project.org[4]

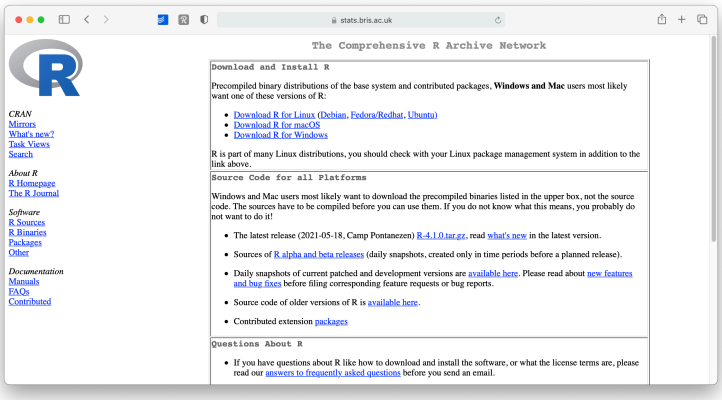
[1] https://www.r-project.org
[2] https://www.rstudio.com
[3] https://rstudio.cloud
[4] https://www.r-project.org%5D(https://www.r-project.org)



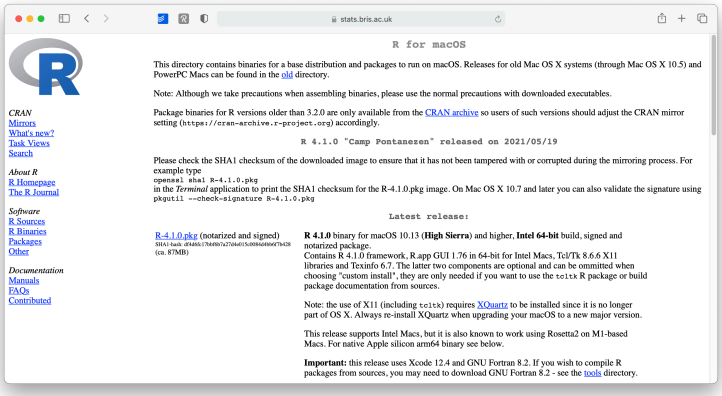
2. Click on `CRAN` where it says `Download`.
3. Choose a server in your country (all of them work, but downloads will perform quicker if you choose your country or one that is close to where you are).



4. Select the operating system for your computer, for example `Download R for macOS`.



5. Select the version you want to install (I recommend the latest version)



6. Open the downloaded file and follow the installation instructions. I recommend leaving the suggested settings as they are.

This was relatively easy. You now have *R* installed. Technically you can start using *R* for your research, but there is one more tool I strongly advise installing: RStudio.

4

The RStudio Interface

The RStudio interface is composed of quadrants, each of which fulfils a unique purpose:

- The `Console` window,
- The `Source` window,
- The `Environment / History / Connections / Tutorial` window, and
- The `Files / Plots / Packages / Help / Viewer` window

Sometimes you might only see three windows and wonder where the `Source` window has gone in your version of RStudio. In order to use it you have to either open a file or create a new one. You can create a new file by selecting `File > New File > R Script` in the menu bar, or use the keyboard shortcut `Ctrl + Shift + N` on PC and `Cmd + Shift + N` on Mac.

I will briefly explain the purpose of each window/pane and how they are relevant to your work in *R*.

5

R *Basics: The very fundamentals*

After a likely tedious installation of *R* and RStudio, as well as a somewhat detailed introduction to the RStudio interface, you are finally ready to *'do'* things. By *'doing'*, I mean *'coding'*. The term *'coding'* in itself can instil fear in some of you, but you only need one skill to do it: Writing. As mentioned earlier, learning coding or programming means learning a new language. However, once you have the basic grammar down, you already can communicate quite a bit. In this section, we will explore the fundamentals of *R*. These build the foundation for everything that follows. After that, we dive right into some analysis.

5.1 Basic computations in *R*

The most basic computation you can do in *R* is arithmetic operations. In other words, addition, subtraction, multiplication, division, exponentiation and extraction of roots. In short, *R* can be used like your pocket calculator, or more likely the one you have on your phone. For example, in Chapter @ref(the-console-window) we already performed an addition. Thus, it might not come as a surprise how their equivalents work in *R*. Let's take a look at the following examples:

```
# Addition
10 + 5
```

```
[1] 15
```

6

Starting your R *projects*

Every new project likely fills you with enthusiasm and excitement. And it should. You are about to find answers to your research questions, and you hopefully come out more knowledgeable due to it. However, there are likely certain aspects of data analysis that you find less enjoyable. I can think of two:

- Keeping track of all the files my project generates
- Data wrangling

While we cover data wrangling in great detail in the next Chapter (Chapter @ref(data-wrangling)), I would like to share some insights from my work that helped me stay organised and, consequently, less frustrated. The following applies to small and large research projects, which makes it very convenient no matter the situation and the scale of the project. Of course, feel free to tweak my approach to whatever suits you. However, consistency is king.

7

Data Wrangling

> **Data wrangling** — also called **data cleaning**, **data remediation**, or **data munging**—refers to a variety of processes designed to transform raw data into more readily used formats. The exact methods differ from project to project depending on the data you're leveraging and the goal you're trying to achieve. (Stobierski 2021)

You collected your data over months (and sometimes years), and all you want to know is whether your data makes sense and reveals something nobody would have ever expected. However, before we can truly go ahead with our analysis, it is essential to understand whether our data is 'tidy'. What I mean is that our data is at the quality we would expect it to be and can be reliably used for data analysis. After all, the results of our research are only as good as our data and the methods we deploy to achieve the desired insights.

8

Descriptive Statistics

The best way to understand how participants in your study have responded to various questions or experimental treatments is to use *descriptive statistics*. As the name indicates, their main purpose is to 'describe'. Most of the time, we want to describe the composition of our sample and how the majority (or minority) of participants performed.

In contrast, we use *inferential statistics* to make predictions. In Social Sciences, we are often interested in predicting how people will behave in certain situations and scenarios. We aim to develop models that help us navigate the complexity of social interactions that we all engage in but might not fully understand. We cover *inferential statistics* in later chapters of this book.

In short, descriptive statistics are an essential component to understand your data. To some extent, one could argue that we were already describing our data when we performed various data wrangling tasks (see Chapter @ref(data-wrangling)). The following chapters focus on essential descriptive statistics, i.e. those you likely want to investigate in 99.9 out of 100 research projects.

This book takes a 'visualised' approach to data analysis. Therefore, each section will entail data visualisations and statistical computing. A key learning outcome of this chapter is to plot your data using the package `ggplot2` and present your data's characteristics in different ways. Each chapter will ask questions about our dataset that we aim to answer visually and computationally. However, first, we need to understand how to create plots in *R*.

8.1 Plotting in *R* with `ggplot2`

Plotting can appear intimidating at first but is very easy and quick once you understand the basics. The `ggplot2` package is a very popular package to generate plots in *R*, and many other packages are built upon it. This makes it a very flexible tool to create almost any data visualisation you could imagine. If you want to see what is possible with `ggplot2`, you might want to consider

looking at #tidytuesday[1] on Twitter, where novices and veterans share their data visualisations every week.

To generate any plot, we need to define three components at least:

- a dataset,
- variables we want to plot, and
- a function to indicate how we want to plot them, e.g. as lines, bars, points, etc.

Admittedly, this is a harsh oversimplification, but it will serve as a helpful guide to get us started. The function `ggplot()` is the one responsible for creating any type of data visualisation. The generic structure of a `ggplot()` looks like this:

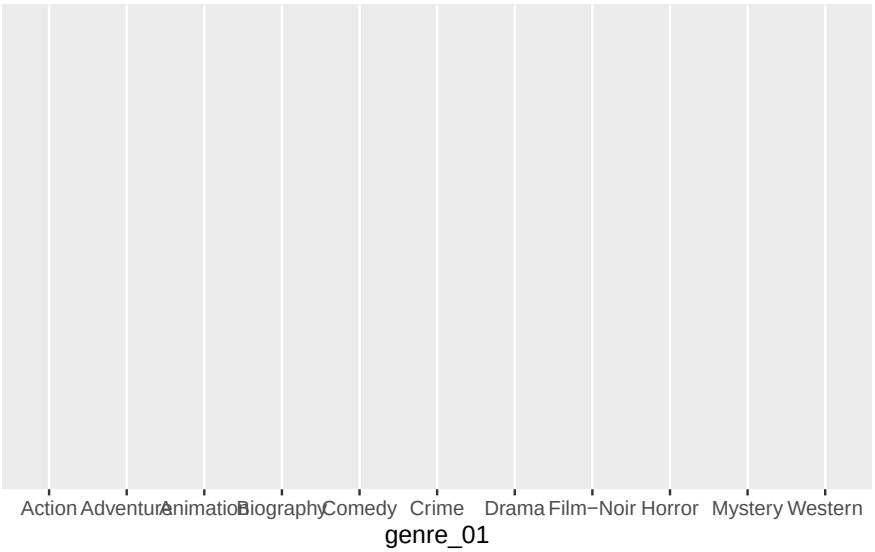
ggplot(data, aes(x = variable_01, y = variable_02))

In other words, we first need to provide the dataset `data`, and then the aesthetics (`aes()`). Think of `aes()` as the place where we define our variables (i.e. `x` and `y`). For example, we might be interested to know which movie genre is the most popular among the top 250 IMDb movies. The dataset `imdb_top_250` from the `r4np` package allows us to find an answer to this question. Therefore we define the components of the plot as follows:

- our data is `imdb_top_250`, and
- our variable of interest is `genre_01`.

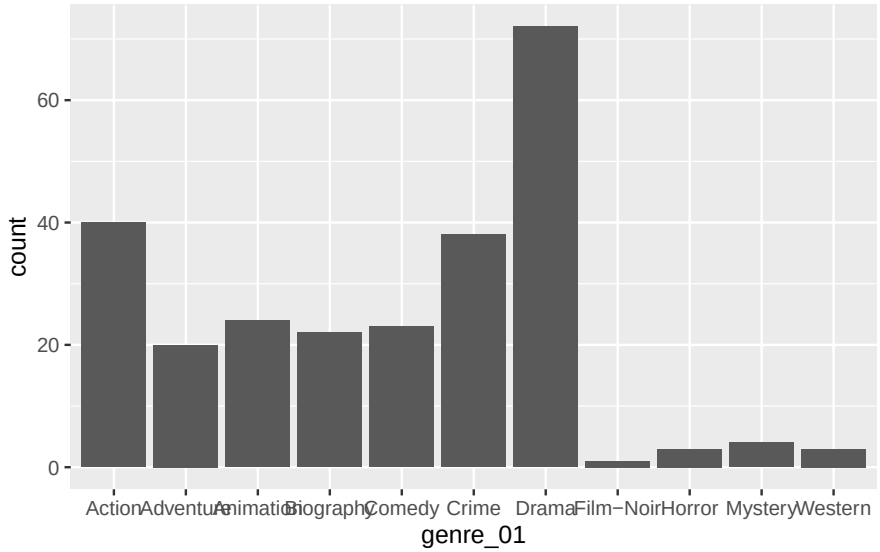
```
ggplot(imdb_top_250, aes(x = genre_01))
```

[1] https://twitter.com/search?q=%23tidytuesday



Running this line of code will produce an empty plot. We only get labels for our x-axis since we defined it already. However, we have yet to tell `ggplot()` how we want to represent the data on this canvas. Your choice for how you want to plot your data is usually informed by the type of data you use and the statistics you want to represent. For example, plotting the mean of a factor is not meaningful, e.g. computing the mean of movie genres. On the other hand, we can count how often specific genres appear in our dataset. One way of representing a factor's count (or frequency) is to use a bar plot. To add an element to `ggplot()`, i.e. bars, we use `+` and append the function `geom_bar()`, which draws bars. The `+` operator works similar to `%>%` and allows to chain multiple functions one after the other as part of a `ggplot()`.

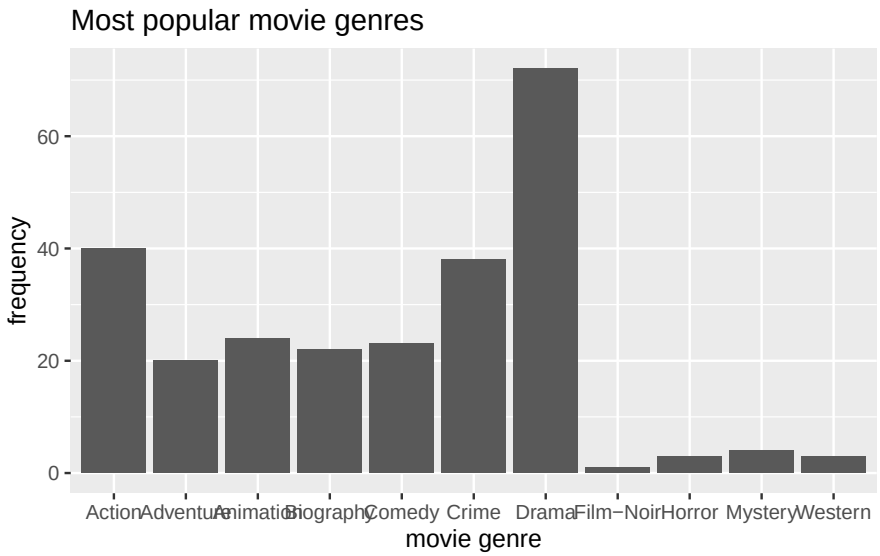
```
ggplot(imdb_top_250, aes(x = genre_01)) +
  geom_bar()
```



With only two lines of coding, we created a great looking plot. We can see that `Drama` is by far the most popular genre, followed by `Action` and `Crime`. Thus, we successfully found an answer to our question. Still, there are more improvements necessary to use it in a publication.

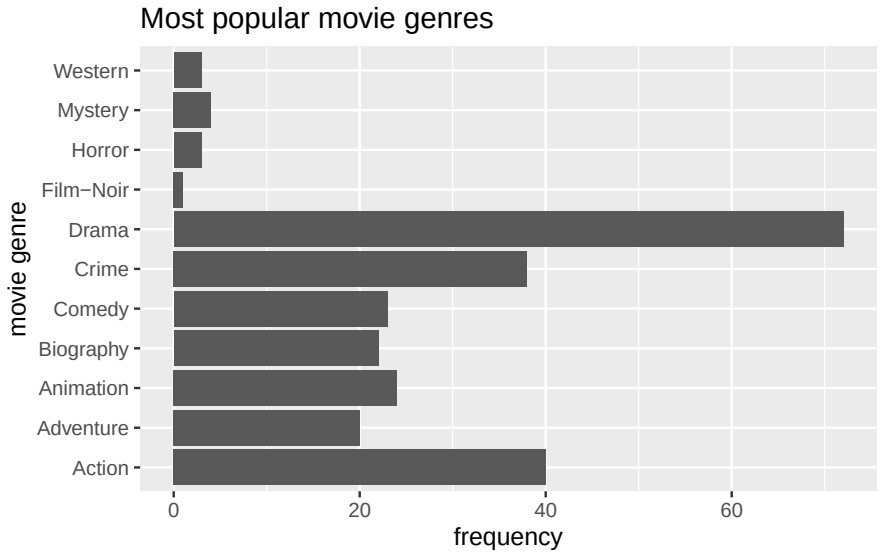
We can use `+` to add other elements to our plot, such as a title and proper axes labels. Here are some common functions to further customise our plot:

```
ggplot(imdb_top_250, aes(x = genre_01)) +
  geom_bar() +
  ggtitle("Most popular movie genres") +  # Add a title
  xlab("movie genre") +                   # Rename x-axis
  ylab("frequency")                       # Rename y-axis
```



When working with factors, the category names can be rather long. In this plot, we have lots of categories, and the labels `Adventure`, `Animation`, and `Biography` are a bit too close to each other for my taste. This might be an excellent opportunity to use `coord_flip()`, which rotates the entire plot by 90 degrees, i.e. turning the x-axis into the y-axis and vice versa. This makes the labels much easier to read.

```
ggplot(imdb_top_250, aes(x = genre_01)) +
  geom_bar() +
  ggtitle("Most popular movie genres") +
  xlab("movie genre") +
  ylab("frequency") +
  coord_flip()
```



Our plot is almost perfect, but we should take one more step to make reading and understanding this plot even easier. At the moment, the bars are ordered alphabetically by movie genre. Unfortunately, this is hardly ever a useful way to order your data. Instead, we might want to sort the data by frequency, showing the most popular genre at the top. To achieve this, we could either sort the movies by hand (see Chapter @ref(reordering-factor-levels)) or slightly amend what we have coded so far.

The problem you encounter when rearranging a `geom_bar()` with only one variable is that we do not have an explicit value to indicate how we want to sort the bars. Our current code is based on the fact that `ggplot` does the counting for us. So, instead, we need to do two things:

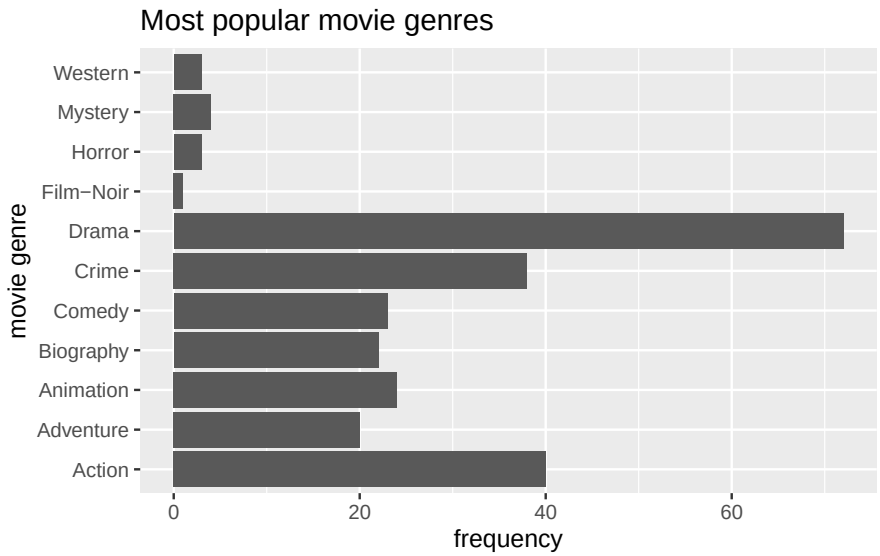
- create a table with all genres and their frequency, and
- use this table to plot the genres by the frequency we computed

```
# Step 1: The frequency table only
imdb_top_250 %>%
  count(genre_01)
```

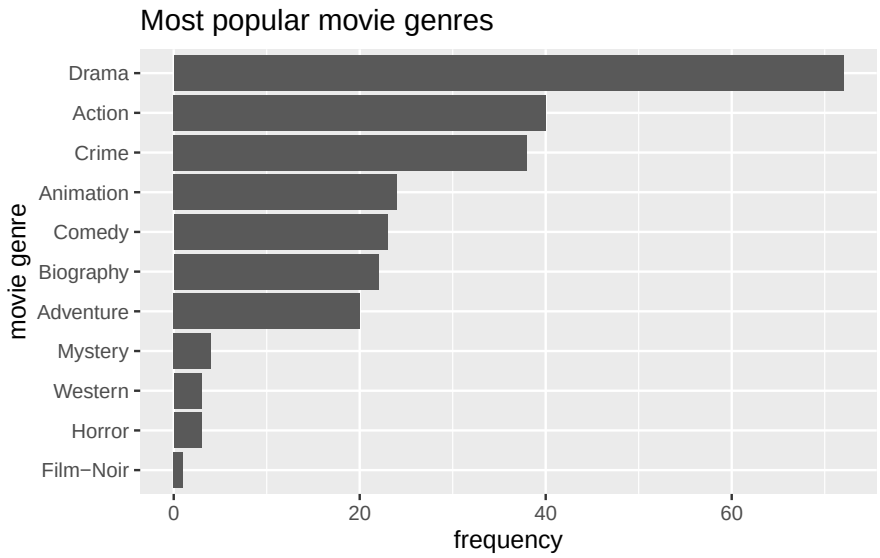
```
# A tibble: 11 x 2
   genre_01      n
   <fct>     <int>
 1 Action       40
 2 Adventure    20
 3 Animation    24
 4 Biography    22
```

```
 5 Comedy       23
 6 Crime        38
 7 Drama        72
 8 Film-Noir     1
 9 Horror        3
10 Mystery       4
11 Western       3
```

```
# Step 2: Plotting a barplot based on the frequency table
imdb_top_250 %>%
  count(genre_01) %>%
  ggplot(aes(x = genre_01, y = n)) +

  # Use geom_col() instead of geom_bar()
  geom_col() +

  # Add titles for plot
  ggtitle("Most popular movie genres") +
  xlab("movie genre") +
  ylab("frequency") +

  # Rotate plot by 180 degrees
  coord_flip()
```



```
#Step 3: reorder() genre_01 by frequency, i.e. by 'n'
imdb_top_250 %>%
  count(genre_01) %>%
  ggplot(aes(x = reorder(genre_01, n), y = n)) +  # Use 'reorder()'
  geom_col() +
  ggtitle("Most popular movie genres") +
  xlab("movie genre") +
  ylab("frequency") +
  coord_flip()
```



Step 3 is the only code you need to create the desired plot. The other two steps only demonstrate how one can slowly build this plot, step-by-step. You might have noticed that I used `dplyr` to chain all these functions together (i.e. `%>%`), and therefore, it was not necessary to specify the dataset in `ggplot()`.

9

Sources of bias: Outliers, normality and other 'conundrums'

'*Bias'* in your analysis is hardly ever a good thing unless you are a qualitative researcher. Whether you consider it positive or negative, we have to be aware of issues preventing us from performing a specific type of analysis. All of the statistical computations discussed in the following chapters can easily be affected by different sources of bias. The lack of certain biases can be an assumption of particular statistical tests. Thus, violating these assumptions would imply that the analytical technique we use will produce wrong results, i.e. biased results. Field (2013) summarises three main assumptions we have to consider:

- Linearity and additivity,
- Independence,
- Normality, and
- Homogeneity of variance, i.e. homoscedasticity.

10

Correlations

Sometimes counting and measuring means, medians, and standard deviations is not enough because they are all based on a single variable. Instead, we might have questions related to the relationship of two or more variables. In this section, we will explore how correlations can (and kind of cannot - see Chapter @ref(simpsons-paradox)) provide insights into the following questions:

- Do movie viewers agree with movie critics regarding the rating of movies?
- Do popular movies receive more votes from users than less popular movies?
- Do movies with more votes also make more money?

11

Power: You either have it or you don't

The most frequently asked question by my students is: How much data do I have to collect? My answer is always the same: *'It depends'*. From a student's perspective, this must be one of the most frustrating answers. Usually, I follow this sentence up with something more helpful to guide students on their way. For example, there is a way to determine how large your sample has to be to detect specific relationships reliably. This procedure is called *power analysis*.

While *power analysis* can be performed before and after data collection, it seems rather pointless to do it afterwards when collecting more data is not only challenging but sometimes impossible. Thus, my explanations in this chapter will primarily focus on the question: How much data is enough to detect relationships between variables reliably.

12

Comparing groups

Social Sciences is about the study of human beings and their interactions. As such, we frequently want to compare two or more groups of human beings, organisations, teams, countries, etc., with each other to see whether they are similar or different from each other. Sometimes we also want to track individuals over time and see how they may have changed in some way or other. In short, comparing groups is an essential technique to make inferences and helps us better understand the diversity surrounding us.

If we want to perform a group comparison, we have to consider which technique is most appropriate for our data. For example, some of it might be related to the type of data we have collected, and other aspects might be linked to the distribution of the data. More specifically, before we apply any statistical technique, we have to consider at least the following:

13

Regression: Creating models to predict future observations

Regressions are an exciting area of data analysis since it enables us to make very specific predictions, incorporating different variables simultaneously. As the name implies, regressions 'regress', i.e., draw on past observations to make predictions about future observations. Thus, any analysis incorporating a regression makes the implicit assumption that the past best explains the future.

I once heard someone refer to regressions as driving a car by looking at the rear-view mirror. As long as the road is straight, we will be able to navigate the car successfully. However, if there is a sudden turn, we might drive into the abyss. This makes it very clear when and how regressions can be helpful. Regressions are also a machine learning method, which falls under *models with supervised learning*. If you find machine learning fascinating, you might find the book *"Hands-on Machine Learning with R"* (Boehmke and Greenwell 2019) very insightful and engaging.

14

Mixed-methods research: Analysing qualitative data in R

Conducting mixed-methods research is challenging for everyone. It requires an understanding of different methods and data types and particular knowledge in determining how one can mix different methods to improve the insights compared to a single method approach. This chapter looks at one possibility of conducting mixed-methods research in *R*. It is likely the most evident use of computational software for qualitative data.

While I consider myself comfortable in qualitative and quantitative research paradigms, this chapter could be somewhat uncomfortable if you are used to only one or the other approach, i.e. only quantitative or only qualitative research. However, with advancements in big data science, it is impossible to code, for example, two million tweets qualitatively. Thus, the presented methodologies should not be considered in isolation from other forms of analysis. For some, they might serve as a screening tool to sift through large amounts of data and find those nuggets of most significant interest that deserve more in-depth analysis. For others, these tools constitute the primary research design to allow to generalise to a larger population. In short: the purpose of your research will dictate the approach.

15

Where to go from here: The next steps in your R *journey*

The first steps have been made, but we can certainly not claim we have reached our destination on this journey. If anything, this book helped us reach the first peak in a series of mountains, giving us an overview of what else there is to explore. To provide some guidance on where to go next, I collated some resources worth exploring as a natural continuation to this book.

15.1 GitHub: A Gateway to even more ingenious *R* packages

If you feel you want more *R* or you are curious to know what other *R* packages can do to complement your analysis, then GitHub brings an almost infinite amount of options for you. Besides the CRAN versions of packages, you find development versions of *R* packages on Github. These usually incorporate the latest features but are not yet available through CRAN. Having said that, if you write your next publication, it might be best to work with packages that are released on CRAN. These are the most stable versions. After all, you don't want the software to fail on you and hinder you from making that world-changing discovery.

Still, more and more *R* packages are released every day that offer the latest advancements in statistical computations and beyond. Methods covered in Chapter @ref(mixed-methods-research) are a great example of what has become possible with advanced programming languages. So if you want to live at the bleeding edge of innovative research methods, then look no further.

However, GitHub also serves another purpose: To back up your research projects and work collaboratively with others. I strongly encourage you to create your own GitHub account, even just to try it. RStudio has built-in features for working with GitHub, making it easy to keep track of your analysis and ensure regular backups. Nobody wants to clean up their data over months and lose it because their cat just spilt the freshly brewed Taiwanese High Mountain Oolong Tea over one's laptop. I use GitHub for many different

things, hosting my blog (The Social Science Sofa[1]), research projects, and this book.

There is much more to say about GitHub that cannot be covered in this book, but if you seek an introduction, you might find GitHub's Youtube Channel[2] of interest or their 'Get started'[3] guide. Best of all, setting up and using your GitHub account is free.

15.2 Books to read and expand your knowledge

There are undoubtedly many more books to read, explore and use. However, my number one recommendation to follow up with this book is 'R for Data Science'[4]. It takes your *R* coding beyond the context of Social Sciences and introduces new aspects, such as *'custom functions'* and *'looping'*. These are two essential techniques if you, for example, have to fit a regression model for +20 subsets of your data, create +100 plots to show results for different countries, or need to create +1000 of individualised reports for training participants. In short, these are very powerful (and efficient) techniques you should learn sooner than later but are less essential for a basic understanding of data analysis in *R* for the Social Sciences.

If you want to expand your repertoire regarding data visualisations, a fantastic starting point represents 'ggplot2: Elegant Graphics for Data Analysis'[5] and 'Fundamentals of Data Visualization'[6]. However, these days I am looking mostly for inspiration and new packages that help me create unique, customised plots for my presentations and publications. Therefore, looking at the source code of plots you like (for example, on GitHub) is probably the best way to learn about `ggplot2` and some new techniques of how to achieve specific effects (see also Chapter @ref(next-steps-twitter)).

If you are a qualitative researcher, you might be more interested in what else you can do with *R* to systematically analyse large amounts of textual data (as was shown in Chapter @ref(mixed-methods-research)). I started with the excellent book 'Text Mining with R: A Tidy Approach'[7], which introduces you in greater depth to sentiment analysis, correlations, n-grams, and topic modelling. The lines of qualitative and quantitative research become increasingly

[1] http://thesocialsciencesofa.com
[2] https://www.youtube.com/user/github
[3] https://docs.github.com/en
[4] https://r4ds.had.co.nz
[5] https://ggplot2-book.org
[6] https://clauswilke.com/dataviz/
[7] https://www.tidytextmining.com

blurred. Thus, learning these techniques will be essential moving forward and pushing the boundaries of what is possible with textual data.

R can do much more than just statistical computing and creating pretty graphs. For example, you can write your papers with it, even if you do not write a single line of code. From Chapter @ref(r-markdown-and-r-notebooks), you might remember that I explained that *R* Markdown files are an alternative to writing R scripts. Suppose you want to deepen your knowledge in this area and finally let go of Microsoft Word, I encourage you to take a peek at the 'R Markdown Cookbook'[8] for individual markdown files and 'bookdown: Authoring Books and Technical Documents with R Markdown'[9] for entire manuscripts, e.g. journal paper submissions or books. I almost entirely abandoned Microsoft Word, even though it served me well for so many years - thanks.

Lastly, I want to make you aware of another open source book that covers What They Forgot to Teach you About R[10] by Jennifer Bryan and Jim Hester. It is an excellent resource to get some additional insights into how one should go about working in *R* and *RStudio*.

15.3 Engage in regular online readings about *R*

A lot of helpful information for novice and expert *R* users is not captured in books but online blogs. There are several that I find inspiring and an excellent learning platform, each focusing on different aspects.

The most comprehensive hub for all things *R* is undoubtedly 'R-bloggers'[11]. It is a blog aggregator which focuses on collecting content related to *R*. I use it regularly to read more about new packages, new techniques, helpful tricks to become more 'fluent' in *R* or simply find inspiration for my own *R* packages. More than once, I found interesting blogs by just reading posts on 'R-bloggers'. For example, the day I wrote this chapter, I learned about `emayili`, which allows you to write your emails from *R* using *R* markdown. So not even the sky is the limit, it seems.

Another blog worth considering is the one from 'Tidyverse'[12]. This blog is hosted and run by RStudio and covers all packages within the `tidyverse`. Posts like 'waldo 0.3.0'[13] made my everyday data wrangling tasks a lot easier

[8] https://bookdown.org/yihui/rmarkdown-cookbook/
[9] https://bookdown.org/yihui/bookdown/
[10] https://rstats.wtf
[11] https://www.r-bloggers.com
[12] https://www.tidyverse.org/blog/
[13] https://www.tidyverse.org/blog/2021/08/waldo-0-3-0/

because finding the differences between two datasets can be like searching a needle in a haystack. For example, it is not unusual to receive two datasets that contain the same measures, but some of their column names are slightly different, which does not allow us to merge them in the way we want quickly. I previously spent days comparing and arranging multiple datasets with over 100 columns. Let me assure you, it is not really fun to do.

The blog 'Data Imaginist'[14] by Thomas Lin Pedersen covers various topics around data visualisations. He is well known for his Generative Art, i.e. art that is computationally generated. But one of my favourite packages, `patchwork`, was written by him and gained immense popularity. It has never been easier to arrange multiple plots with such little code.

Lastly, I want to share with you the blog of Cédric Scherer[15] for those of you who want to learn more about high-quality data visualisations based on `ggplot2`. His website hosts many visualisations with links to the source code on GitHub to recreate them yourself. It certainly helped me improve the visual storytelling of my research projects.

15.4 Join the Twitter community and hone your skills

Learning *R* means you also join a community of like-minded programmers, researchers, hobbyists and enthusiasts. Whether you have a Twitter account or not, I recommend looking at the #RStats[16] community there. Plenty of questions about *R* programming are raised and answered on Twitter. In addition, people like to share insights from their projects, often with source code published on GitHub. Even if you are not active on social media, it might be worth having a Twitter account just to receive news about developments in *R* programming.

As with any foreign language, if we do not use it regularly, we easily forget it. Thus, I would like to encourage you to take part in Tidy Tuesday[17]. It is a weekly community exercise around data visualisation and data wrangling. In short: You download a dataset provided by the community, and you are asked to create a visualisation as simple or complex as you wish. Even if you only manage to participate once a month, it will make you more 'fluent' in writing your code. Besides, there is a lot to learn from others because you are also asked to share the source code. This activity takes place on Twitter, and you can find contributions by using #TidyTuesday[18]. Whether you want to

[14] https://www.data-imaginist.com
[15] https://www.cedricscherer.com/top/dataviz/
[16] https://twitter.com/search?q=%23rstats
[17] https://www.tidytuesday.com
[18] https://twitter.com/search?q=%23tidytuesday

share your plots is up to you, but engaging with this activity will already pay dividends. Besides, it is fun to work with datasets that are not necessarily typical for your field.

References

Boehmke, Brad, and Brandon Greenwell. 2019. *Hands-on Machine Learning with r*. Chapman; Hall/CRC.

Field, Andy. 2013. *Discovering Statistics Using IBM SPSS Statistics*. Sage Publications.

Stobierski, Tim. 2021. "Data Wrangling: What It Is and Why It's Important." Harvard Business School Online. https://online.hbs.edu/blog/post/data-wrangling.