Computational Analysis for Bioscientists

Data Analysis in R and what they forgot to teach you about computers!

Emma Rand

2023-03-01

Table of contents

Welcome!

front page stuff

1 About this book

:::: status ::: callout-important You are reading a work in progress. This page is a dumping ground for ideas and not really readable. ::: ::::

Who is this book for

bioscience

undergrads

It is in sections

part 1 what they forgot to teach you

focus on what causes problems for people learning to code.

part 2 Getting started with data. give summary

part 3 Data Analysis, improve name, give summary (babs 2)

When you see.. Try to answer before looking at the code

Your turn! Assign the value of `4` to a variable called `y`:

```
y <- 4
```

conventions used in this book

Part I

What they forgot to teach you about computers

> **!** Important
>
> You are reading a work in progress. This page is a dumping ground for ideas and not really readable. :::

Why this part

give a summary of contents

2 Operating Systems

:::: status ::: callout-important You are reading a work in progress. This page is a dumping ground for ideas and not really readable. :::

2.1 what is an operating system

2.2 types of operating system

include windows, mac, unix, tablets, android, apple

2.3 differences in how you use them

keyboard keys and characters

For RStudio, the section on Keyboard Shortcuts and tips willhelp.

- enter / return
- control / command
- alt / option

Finder and Explorer

installing software

3 Understanding file systems

> **!** Important
>
> You are reading a work in progress. This page is a dumping ground for ideas and not really readable. :::

A file is a unit of storage on a computer with a name that uniquely identifies it. Files can be of different types depending on the sort of information held in them. The file name very often consists of two parts, separated by a dot:

- name - the base name of the file
- extension that should indicate the format or content of the file.

Some examples are report.doc, analysis.R, culture.csv and readme.txt. The relationship between the file extension and the file type

One of the simplest types of file is a "text file" which contains text characters without formatting such as bold or italics and no images or colours. Plain text files can be opened in any text editor like Windows Notepad or Mac's TextEdit.

Data is commonly held in text files because they can be read by many programs

files of file plain text, markup and markdown

file extensions

the relationship between file extensions and programs

A file system contains files and folders

files systems are hierarchical

folder is a directory getwd(), dir() in R, cd, pwd in unix, os.getcwd() in Python

using a file explorer, showing file extensions

Paths

root directory

typical structure on windows and mac

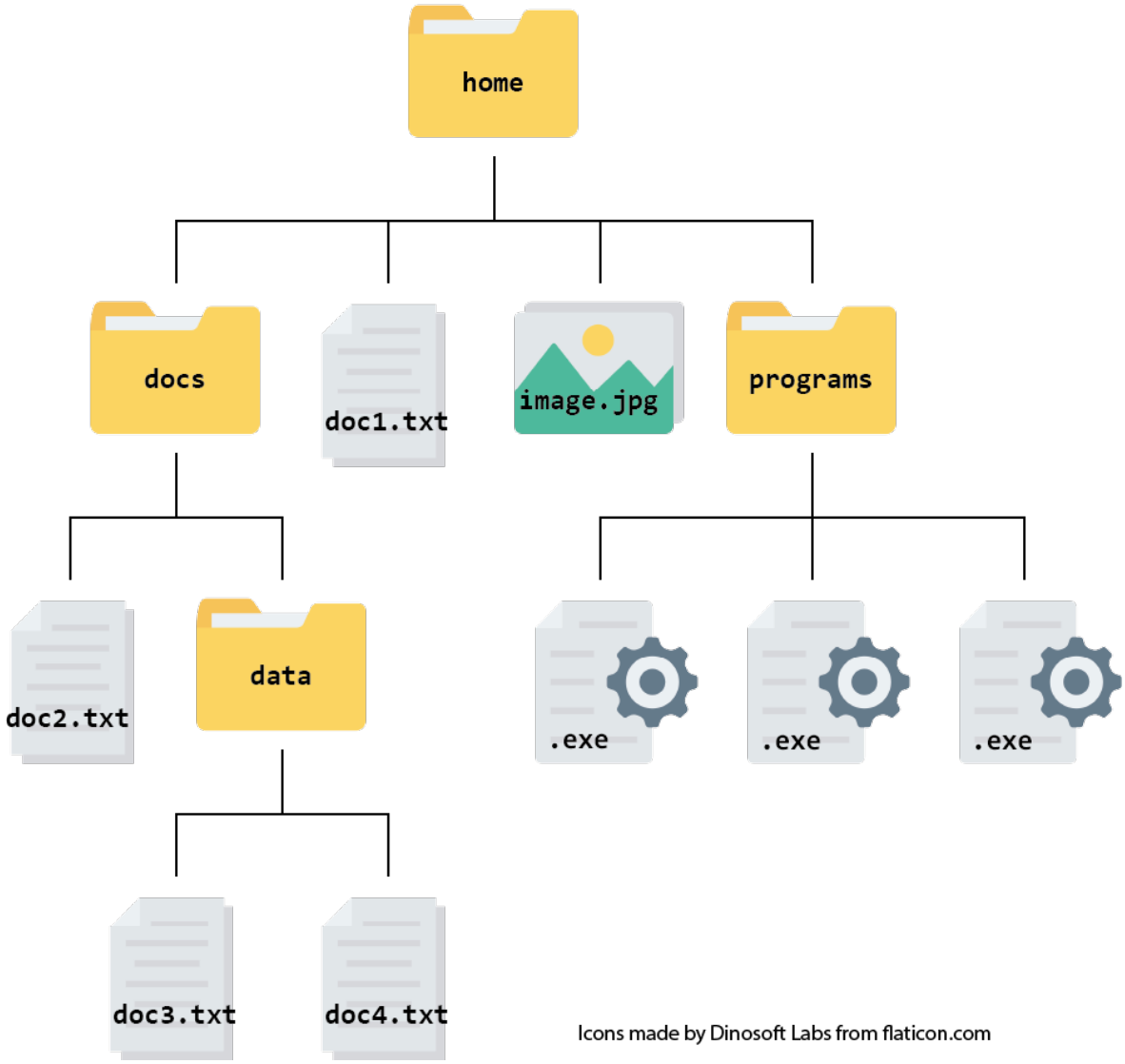


Figure 3.1: A file hierarchy containing 4 levels of folders and files

Working directory

Relative and absolute paths

save files fromthe internet chrome://settings/downloads

4 Organising your work

:::: status ::: callout-important You are reading a work in progress. This page is a dumping ground for ideas and not really readable. :::

use folder

consistency

naming things

::: {.quarto-book-part}

Part II

Getting started with data

> **!** Important
>
> You are reading a work in progress. This page is a dumping ground for ideas and not really readable. :::

why this part

summary of the chapters

general ideas about data and data types

first steps with rstudio

working with data in RStudio

5 Ideas about data

> **!** Important
>
> You are reading a work in progress. This page is a dumping ground for ideas and not really readable. :::

This chapter covers some important concepts data. Data is made up of properties we have measured or recorded, known as variables, and observations, the individual things with those properties. Data is most commonly (and helpfully) organised with variables in columns and each observation on a row.

We can define a variable in two main ways:

1. by what kinds of value it can take and how frequently each of its possible values occur
2. by what role the variable takes in analysis

Both of these determine how we summarise, plot and analyse data.

5.1 Role in analysis

When we do research, we typically have variables that we choose or set and variables that we measure. The variables we choose or set are called independent or explanatory variables. The variables we measure are called dependent or response variables.

TODO: examples

5.2 Kinds of value: data types

The types of values a variable can take determines how we summarise, plot and analyse them. Sometimes this is obvious - when you can recorded the colour of an observation you can't find the mean colour of the sample but you can report the most common colour.

An important distinction is between discrete and continuous types of data. Continuous variables are measurements that can take any value in their range. Discrete variables can take only specific values.

5.2.1 Discrete data

Discrete variables can take only specific values, like genotype or the number of leaves

5.2.1.1 Nominal and Ordinal

Nominal and ordinal data are categorical and often act as explanatory variables.
Nominal variable have no particular order, for example, the eye colour of Drosophila or the genotype of a mouse. When summarising data on eye colour, it wouldn't matter what order the information was given or plotted. Ordinal variables have an order. The Likert scale used in questionnaires is one example. The possible responses are Strongly agree, Agree, Disagree and Strongly disagree; these have an order that you would use when plotting them.

Summarising nominal or ordinal data The most appropriate way to summarise nominal or ordinal data is to report the most frequent values or tabulate the number of each value.

5.2.1.2 Counts

Counts are one of the most common data types. They are quantitative but discrete because they can take only specific values

5.2.2 Continuous data

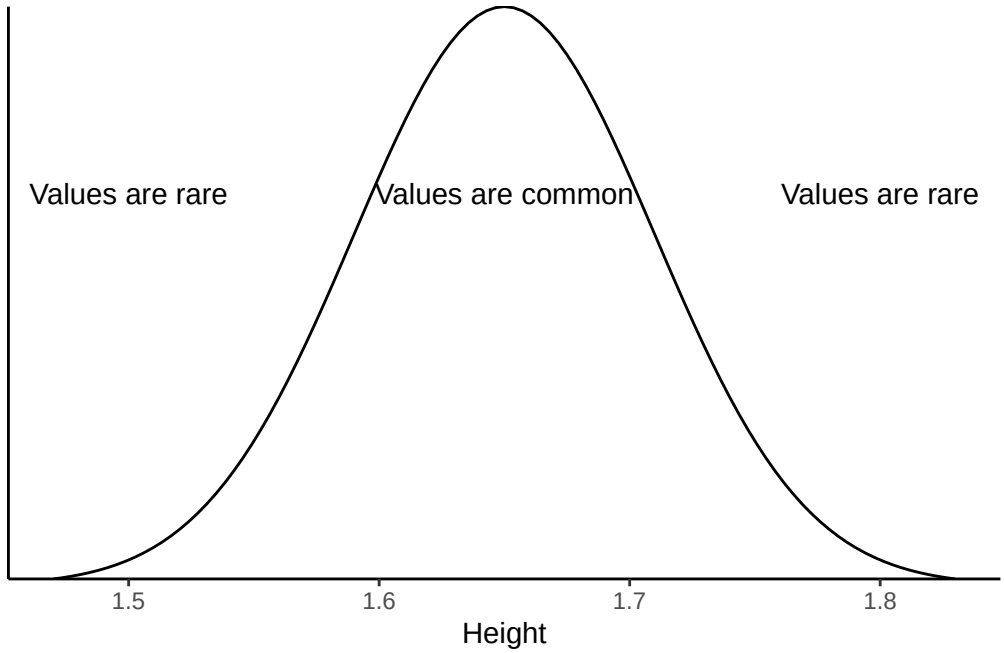
Continuous variables are measurements that can take *any* value in their range so there are an infinite number of possible values. The values have decimal places. Variables like the length and mass of an organism, the volume and optical density of a solution, or the colour intensity of an image are continuous. Many response variables are continuous but continuous variables can also be explanatory. For example,

5.3 Distributions

The distribution of a variable describes the types of values it can take and the likelihood of each value occurring. For example, for a variable like human height values of 1.65 metres occur more often than values of 2 metres and values of 3 metres never occur.

```
m <- 1.65
sd <- 0.06
ggplot(data = data.frame(Height = c(m - 3 * sd, m + 3 * sd)), aes(Height)) +
  stat_function(fun = dnorm, n = 101,
```

```
              args = list(mean = m, sd = sd)) +
scale_y_continuous(breaks = NULL, name = "",
                   expand = c(0, 0)) +
annotate("text", x = 1.5, y = 4.5,
         label = "Values are rare") +
  annotate("text", x = 1.65, y = 4.5,
         label = "Values are common") +
  annotate("text", x = 1.8, y = 4.5,
         label = "Values are rare") +
theme_classic()
```



5.3.1 The normal distribution

5.3.2 Distribution of counts

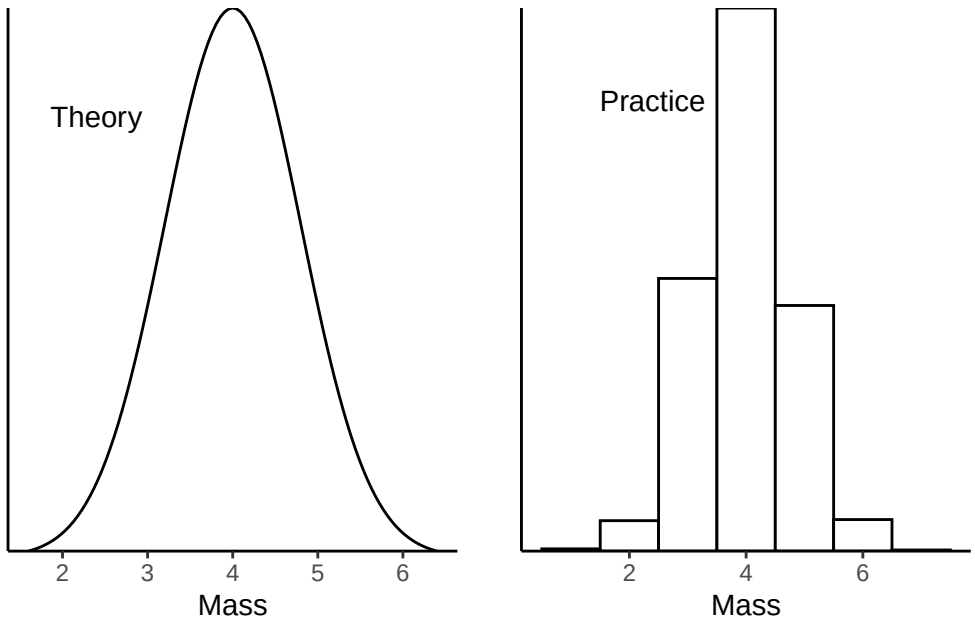
5.4 Theory and practice

The distinction between continuous and discrete values is clear in theory but in practice, the actual values you have might differ from what we would expect for a particular variable. For example, we would expect the mass of cats to be continuous but if our scales only measure to the nearest kilogram then

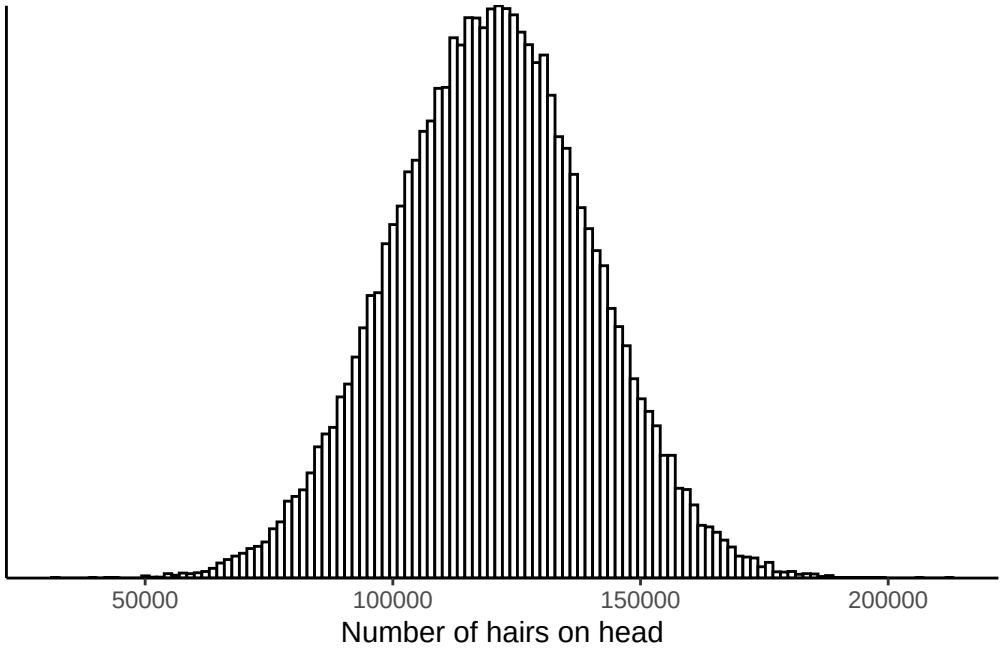
```r
m <- 4
sd <- 0.8
set.seed(1234)
a <- ggplot(data = data.frame(Mass = c(m - 3 * sd, m + 3 * sd)), aes(Mass)) +
  stat_function(fun = dnorm, n = 101,
                args = list(mean = m, sd = sd)) +
  scale_y_continuous(breaks = NULL, name = "",
                     expand = c(0, 0)) +
  annotate("text", x = m - 2 * sd, y = 0.4,
           label = "Theory") +
  theme_classic()

b <- ggplot(data = data.frame(Mass = round(rnorm(1000, m, sd), 0)), aes(Mass)) +
  geom_histogram(binwidth = 1, colour = "black", fill = "white") +
  scale_y_continuous(breaks = NULL, name = "",
                     expand = c(0, 0)) +
  annotate("text", x = m - 2 * sd, y = 400,
           label = "Practice") +
  theme_classic()

a + b
```



```r
m <- 120000
sd <- 20000
set.seed(12)
ggplot() +
  geom_histogram(data = data.frame(hairs = round(rnorm(60000, m, sd), 0)),
                 aes(hairs),
                 bins = 120, colour = "black", fill = "white") +
  scale_y_continuous(breaks = NULL, name = "",
                     expand = c(0, 0)) +
  scale_x_continuous("Number of hairs on head") +
  theme_classic()
```



6 First Steps in RStudio

:::: status ::: callout-important You are reading a work in progress. This page almost readable but is a first draft and may substantial edits. :::

This chapter starts by explaining what R and RStudio are and how you can install them on your own machine. We introduce you to working in RStudio, changing its appearance to suit you and to the key things you need to know about R.

6.1 What are R and Rstudio?

6.1.1 What is R?

R is a programming language and environment for statistical computing and graphics which is free and open source. It is widely used in industry and academia. It is what is known as a "domain-specific" language meaning that it is designed especially for doing data analysis and visualisation rather than a "general-purpose" programming language like Python and C++. It makes doing the sorts of things that bioscientists do a bit easier than in a general purpose-language.

6.1.2 What is RStudio?

RStudio is what is known as an "integrated development environment" (IDE) for R made by Posit. IDEs have features that make it easier to do coding like syntax highlighting, code completion and viewers for files, code objects, packages and plots. You don't have to use RStudio to use R but it is very helpful.

6.1.3 Why is it better to use R than Excel, googlesheets or some other spreadsheet program?

Spreadsheet programs are not statistical packages so although you can carry out some analysis tasks in them they are limited, get things wrong (known about since 1994) and teach you bad data habits. Spreadsheets encourage you to do things that are going to make analysis difficult.

6.1.4 Why is it better to use R than SPSS, Minitab or some other menu-driven statistics program?

- R is free and open source which it will always be available to you .
- Carrying out data analysis using coding makes everything you do reproducible
- The skills and expertise you gain through learning R are highly transferable – much more so than those acquired using SPSS.
- See Thomas Mock's demonstration of doing some data analysis in R including "The Kick Ass Curve": https://rstudio.com/resources/webinars/a-gentle-introduction-to-tidy-statistics-in-r/

There are other good options such as Julia and Python and you are encouraged to explore these. We chose R in part because of the R community which is one of R's greatest assets, being vibrant, inclusive and supportive of users at all levels. https://ropensci.org/blog/2017/06/23/community/

6.2 Installing R and Rstudio

You will need to install both R and RStudio to use them on your own machine. Installation is normally straightforward but you can follow a tutorial here: https://learnr-examples.shinyapps.io/ex-setup-r/#section-welcome

6.2.1 Installing R

Go to https://cloud.r-project.org/ and download the "Precompiled binary distributions of the base system and contributed packages" appropriate for your machine.

6.2.1.1 For Windows

Click "Download R for Windows", then "base", then "Download R 4.#.# for Windows". This will download an `.exe` file. Once downloaded, open (double click) that file to start the installation.

6.2.1.2 For Mac

Click "Download R for (Mac) OS X", then "R-4.#.#.pkg" to download the installer. Run the installer to complete installation.

6.2.1.3 For Linux

Click "Download R for Linux". Instructions on installing are given for Debian, Redhat, Suse and Ubuntu distributions. Where there is a choice, install both `r-base` and `r-base-dev`.

6.2.2 Installing R Studio

Go to https://posit.co/download/rstudio-desktop/

6.3 Install the `tidyverse` package

Install `tidyverse`:

```
install.packages("tidyverse")
```

6.4 Introduction to RStudio

In this section we will introduce you to working in RStudio. We will explain the windows that you see when you first open RStudio and how to change its appearance to suit you. Then we will see how we use R as a calculator and how assign values to R objects.

6.4.1 Changing the appearance

When you first open RStudio it will display three panes and have a white background Figure **??**

We will talk more about these three panes soon but first, let's get into character - the character of a programmer! You might have noticed that people comfortable around computers are often using dark backgrounds. A dark background reduces eye strain and often makes "code syntax" more obvious making it faster to learn and understand at a glance. Code syntax is the set of rules that define what the various combinations of symbols mean. It takes time to learn these rules and you will learn best by repeated exposure to writing, reading and copying code. You have done this before when you learned your first spoken language. All languages have syntax rules governing the order of words and we rarely think about these consciously, instead relying on what sounds and looks right. And what sounds and looks right grows out repeated exposure. For example, 35% of languages, including English, Chinese, Yoruba and Polish use the Subject-Verb-Object syntax rule:

- English: Emma likes R

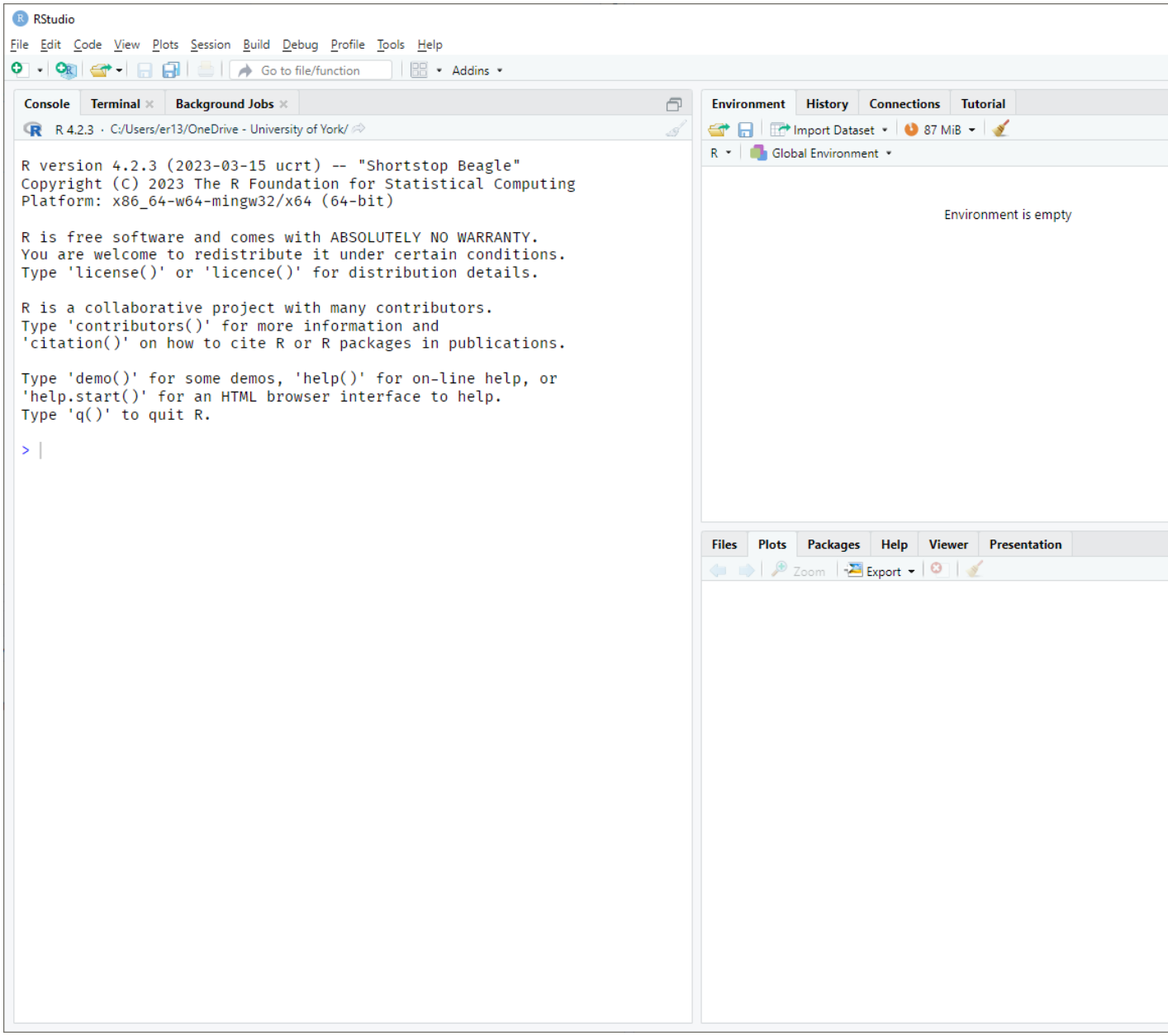


Figure 6.1: When you first open RStudio it will be white with three panes

- Chinese: R Emma xǐhuān R
- Yoruba: Emma fẹran R
- Polish: Emma lubi R

and 40% use Subject-Object-Verb including Turkish and Korean

- Turkish: Emma R'yi seviyor
- Korean: R emmaneun Reul joh-ahanda

You learned this rule in your language very early, long before you were conscious of it, just by being exposed to it frequently. In this book I try to tell you the syntax rules, but you will learn most from looking at, and copying code. Because of this, it is well worth tinkering with the appearance of RStudio to see what Editor theme makes code elements most obvious to you.

There is a tool bar at the top of RStudio. Choose the `Tools` option and then `Global options`. This will open a window where many options can be changed Figure **??**.

Go to the `Appearance` Options and choose and Editor theme you like, followed by OK.

The default theme is Textmate. You will notice that all the Editor themes have syntax highlighting so that keywords, variable names, operators, etc are coloured but some themes have stronger contrasts than others. For beginners, I recommend Vibrant Ink, Chaos or Merbivore rather than Dreamweaver or Gob which have little contrast between some elements. However, individuals differ so experiment for yourself. I tend to vary between Solarised light and dark.

You can also turn one Screen Reader Support in the Accessibility Options in Tools | Global Options.

Back to the Panes. You should be looking at three windows: One on the left and two on the right[1].

The window on the left, labelled Console, is where R commands are executed. In a moment we will start by typing commands in this window. Over on the right hand side, at the top, have several tabs, with the Environment tab showing. This is where all the objects and data that you create will be listed. Behind the Environment tab is the History and later you will be able to view this to see a history of all your commands.

On the bottom right hand side, we have a tab called Plots which is where your plots will go, a tab called Files which is a file explorer just like Windows Explorer or Mac Finder, and a Packages tab where you can see all the packages that are installed. The Packages tab also provides a way to install additional packages. The Help tab has access to all the manual pages.

Right, let's start coding!

[1] If this is not a fresh install of RStudio, you might be looking at fours windows, two on the left and two on the right. That's fine - we will al be using four shortly. For the time being, you might want to close the "Script" window using the small cross next to "Untitled1".

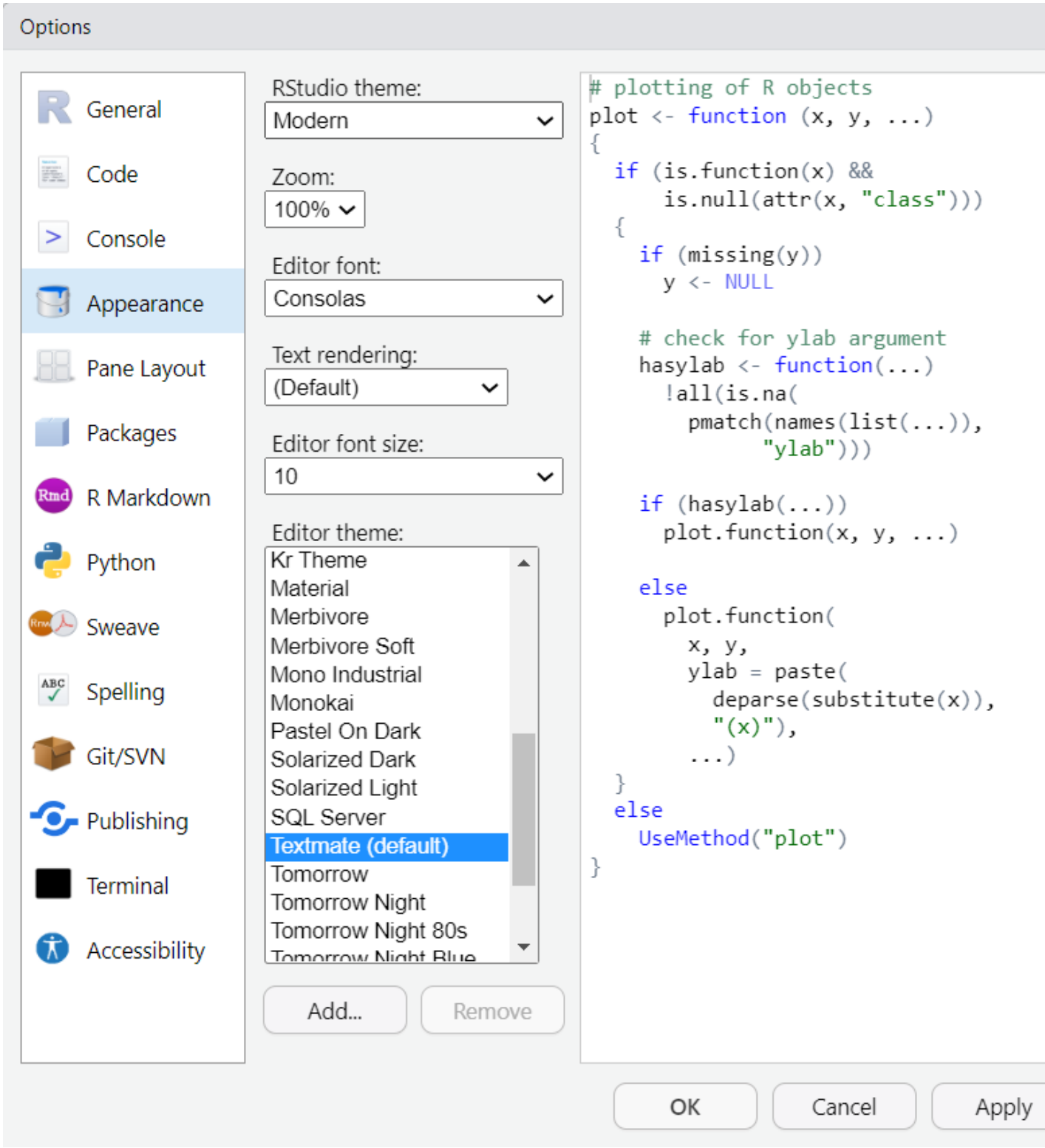


Figure 6.2: Tools | Global Options opens a window. One of the options is Appearance

6.4.2 Your first piece of code

We can use R just like a calculator. Put your cursor after the `>` in the Console, type `3 + 4` and Enter to send that command:

```
3+4
```

```
[1] 7
```

The `>` is called the "prompt". You do not have to type it, it tells you that R is ready for input.

Where I've written `3+4`, I have no spaces. However, you *can* have spaces, and in fact, it's good practice to use spaces around your operators because it makes your code easier to read. So a better way of writing this would be:

```
3 + 4
```

```
[1] 7
```

In the output we have the number `7`, which, obviously, is the answer. From now on, you should assume commands typed at the console should be followed by Enter to send them.

The one in parentheses, `[1]`, is an index. It is telling you that the `7` is the first element of the output. We can see this more clear if we create something with more output. For example, `50:100` will print the numbers from 50 to 100.

```
50:100
```

```
 [1]  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68
[20]  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87
[39]  88  89  90  91  92  93  94  95  96  97  98  99 100
```

The numbers in the square parentheses at the beginning of the line give you the index of the first element in the line. R is telling you where you are in the output.

6.4.3 Assigning variables

Very often we want to keep input values or output for future use. We do this with 'assignment' An assignment is a statement in programming that is used to set a value to a variable name. In R, the operator used to do assignment is `<-`. It assigns the value on the right-hand to the value on the left-hand side.

To assign the value `3` to `x` we do:

```
x <- 3
```

and Enter to send that command.

The assignment operator is made of two characters, the `<` and the `-` and there is a keyboard short cut: Alt+- (windows) or Option+- (Mac). Using the shortcut means you'll automatically get spaces. You won't see any output when the command has been executed because there is no output. However, you will see `x` listed under Values in the Environment tab (top right).

Your turn! Assign the value of `4` to a variable called `y`:

```
y <- 4
```

Check you can see `y` listed in the Environment tab.

Type `x` and Enter to print the contents of `x` in the console:

```
x
```

```
[1] 3
```

We can use these values in calculations just like we could in in maths and algebra.

```
x + y
```

```
[1] 7
```

We get the output of `7` just as we expect. Suppose we make a mistake when typing, for example, accidentally pressing the u button instead of the y button:

```
x + u
```

```
Error in eval(expr, envir, enclos): object 'u' not found
```

We get an error. We will probably see this error quite often - it means we have tried to use a variable that is not in our Environment. So when you get that error, have a quick look up at your environment[2].

We made a typo and will want to try again. We usefully have access to all the commands that previously entered when we use the ↑ Up Arrow. This is known as command recall. Pressing the ↑ Up Arrow once recalls the last command; pressing it twice recalls the command before the last one and so on.

Recall the `x + u` command (you may need to use the ↓ Down Arrow to get back to get it) and use the Back space key to remove the `u` and then add a `y`.

A lot of what we type is going to be wrong - that is not because you are a beginner, it is same for everybody! On the whole, you type it wrong until you get it right and then you move to the next part. This means you are going to have to access your previous commands often. The History - behind the Environment tab - contains everything you can see with the ↑ Up Arrow. You can imagine that as you increase the number of commands you run in a session, having access the this record of everything you did is useful. However, the thing about the history is, that it has *everything* you typed, including all the wrong things!

What we really want is a record of everything we did that was right! This is why we use scripts instead of typing directly into the console.

6.4.4 Using Scripts

An R script is a plain text file with a `.R` extension and containing R code. Instead of typing into the console, we normally type into a script and then send the commands to the console when we are ready to run them. Then if we've made any mistakes, we just correct our script and by the end of the session, it contains a record of everything we typed that worked.

You have several options open a new script:

- button: Green circle with a white cross, top left and choose R Script
- menu: File | New File | R Script
- keyboard shortcut: Ctrl+Shift+N (Windows) / Shift+Command+N (Mac)

Open a script and add the two assignments to it:

```
x <- 3
y <- 4
```

To send the first line to the console, we place our cursor on the line (anywhere) and press Ctrl-Enter (Windows) / Command-Return. That line will be executed in the console and in

[2]When we are using scripts, it is very easy to write code but forget to run it. Very often when you see this error it will because you have written the code to create an object but forgotten to execute it.

the script, our cursor will jump to the next line. Now, send the second command to the console in the same way.

From this point forward, you should assume commands should be typed into the script and sent to the console.

Add the incorrect command attempting to sum the two variables:

```
x + u
```

```
Error in eval(expr, envir, enclos): object 'u' not found
```

To correct this, we do not add another line to the script but instead edit the existing command:

```
x + y
```

```
[1] 7
```

In addition to making it easy to keep a record of your work, scripts have another big advantage, you can include 'comments' - pieces of text that describe what the code is doing. Comments are indicated with a `#` in front of them. You can write anything you like after a `#` and R will recognise that it is not code and doesn't need to be run.

```
# This script performs the sum of two values

x <- 3    # can be altered
y <- 4    # can be altered

# perform sum
x + y
```

```
[1] 7
```

The comments should be highlighted in a different colour than the code. They will be italic in some Editor themes.

You have several options save a script:

- button: use the floppy disc icon
- menu: File | Save
- keyboard shortcut: Ctrl+S (Windows) / Command+S (Mac)

You could use a name like `test1.R` - note the `.R` extension wil be added automatically.