

# Introduction to Data Analysis

*Michael Franke*

*last rendered at: 2019-12-03 15:42:30*



# Contents

|  |           |
|--|-----------|
| <b>Preface</b>   | <b>7</b>  |
| <b>I. Preliminaries</b>  | <b>9</b>  |
| <b>1. General Introduction</b>                                 | <b>11</b> |
| 1.1. Learning goals . . . . .                                  | 11        |
| 1.2. Course structure . . . . .                                | 11        |
| 1.3. Tools and topics covered (and not covered) here . . . . . | 12        |
| 1.4. Data sets covered . . . . .                               | 13        |
| 1.5. Installation . . . . .                                    | 14        |
| <b>2. Basics of R</b>  | <b>19</b> |
| 2.1. First steps . . . . .                                     | 20        |
| 2.2. Data types . . . . .                                      | 27        |
| 2.3. Functions . . . . .                                       | 39        |
| 2.4. Loops and maps . . . . .                                  | 42        |
| 2.5. Piping . . . . .  | 44        |
| 2.6. Rmarkdown . . . . .                                       | 45        |
| <b>II. Data</b>  | <b>47</b> |
| <b>3. Data, variables &amp; experimental designs</b>           | <b>49</b> |
| 3.1. Different kinds of data . . . . .                         | 49        |
| 3.2. On the notion of “variables” . . . . .                    | 52        |
| 3.3. Basics of experimental design . . . . .                   | 54        |
| <b>4. Data Wrangling</b>                                       | <b>59</b> |
| 4.1. Data in, data out . . . . .                               | 59        |
| 4.2. Tidy data . . . . .                                       | 60        |
| 4.3. Data manipulation: the basics . . . . .                   | 63        |
| 4.4. Grouped operations . . . . .                              | 72        |
| 4.5. Case study: the King of France . . . . .                  | 74        |
| <b>5. Summary statistics</b>                                   | <b>81</b> |
| 5.1. Counts and proportions . . . . .                          | 82        |

## *Contents*

|  |            |
|--|------------|
| 5.2. Central tendency and dispersion . . . . .                                     | 86         |
| 5.3. Co-variance & correlation . . . . .   | 100        |
| <b>6. Data Visualization</b>   | <b>105</b> |
| 6.1. Motivating example: Anscombe’s quartet . . . . .                              | 105        |
| 6.2. Visualization: the good, the bad and the info-graphic . . . . .               | 109        |
| 6.3. Basics of ggplot . . . . .  | 115        |
| 6.4. A rendezvous with popular geoms . . . . .                                     | 133        |
| 6.5. Faceting . . . . .  | 154        |
| 6.6. Customization etc. . . . .  | 157        |
| <b>III. Models and inferences</b>  | <b>169</b> |
| <b>7. Basics of Probability Theory</b>   | <b>171</b> |
| 7.1. Probability . . . . .   | 171        |
| 7.2. Structured events & marginal distributions . . . . .                          | 173        |
| 7.3. Conditional probability . . . . .   | 174        |
| 7.4. Random variables . . . . .  | 176        |
| 7.5. Expected value & variance . . . . .   | 182        |
| <b>8. Two approaches to statistical inference</b>                                  | <b>183</b> |
| 8.1. Overview . . . . .  | 183        |
| 8.2. Two notions of probability (revisited) . . . . .                              | 183        |
| <b>9. Models</b>   | <b>185</b> |
| 9.1. Likelihood, Prior, & Posterior . . . . .                                      | 185        |
| 9.2. Modeling . . . . .  | 190        |
| 9.3. Further examples . . . . .  | 198        |
| 9.4. Further elaboration on modeling (in anticipation of the topic “estimation”) . | 204        |
| <b>10. Parameter inference</b>   | <b>209</b> |
| <b>11. Hypothesis Testing</b>  | <b>211</b> |
| <b>12. Model Comparison</b>  | <b>213</b> |
| <b>13. Bayesian hypothesis testing</b>   | <b>215</b> |
| <b>14. Model criticism</b>   | <b>217</b> |
| <b>IV. Applied (generalized) linear modeling</b>                                   | <b>219</b> |
| <b>15. Simple linear regression</b>  | <b>221</b> |

|  |            |
|--|------------|
| <b>16. Logistic regression</b>   | <b>223</b> |
| <b>17. Multinomial regression</b>  | <b>225</b> |
| <b>18. Ordinal regression</b>  | <b>227</b> |
| <b>19. Hierarchical regression</b>   | <b>229</b> |
| <b>A. Further useful material</b>  | <b>231</b> |
| A.1. Material on <i>Introduction to Probability</i> : . . . . .                | 231        |
| A.2. Material on <i>Bayesian Data Analysis</i> : . . . . .                     | 231        |
| A.3. Material on <i>frequentist statistics</i> : . . . . .                     | 231        |
| A.4. Material on <i>R, tidyverse, etc.</i> : . . . . .                         | 231        |
| A.5. Further information for RStudio . . . . .                                 | 232        |
| A.6. Resources on WebPPL . . . . .   | 232        |
| <b>B. Common probability distributions</b>                                     | <b>233</b> |
| B.1. Selected continuous distributions of random variables . . . . .           | 233        |
| B.2. Selected discrete distributions of random variables . . . . .             | 255        |
| B.3. Understanding distributions as random variables . . . . .                 | 267        |
| <b>C. Exponential Family and Maximum Entropy</b>                               | <b>275</b> |
| C.1. An important family: The Exponential Family . . . . .                     | 275        |
| C.2. Excursos: “Information Entropy” and “Maximum Entropy Principal” . . . . . | 276        |
| <b>D. Data sets used in the book</b>   | <b>285</b> |
| D.1. Mental Chronometry . . . . .  | 285        |
| D.2. Simon Task . . . . .  | 295        |
| D.3. World Values Survey (wave 6   2010-2014) . . . . .                        | 305        |
| D.4. King of France . . . . .  | 305        |
| D.5. Bio-Logic Jazz-Metal (and where to consume it) . . . . .                  | 314        |
| D.6. Avocado prices . . . . .  | 320        |



# Preface

This book is the basic reading material for the course “Introduction to Data Analysis”, held at the University of Osnabrück in the winter term of 2019, as part of the BSc *Cognitive Science* program. It introduces key concepts of data analysis from a frequentist and a Bayesian tradition. It uses R to handle, plot and analyze data. It relies on simulation to illustrate selected statistical concepts.



**Part I.**

## **Preliminaries**



# 1. General Introduction

This chapter lays out the learning goals of this book (Section 1.1) and describes how these goals are to be achieved (Section 1.2). Section 1.3 details which technical tools and methods are covered here, and which are not. There will be some information on the kinds of data sets we will use during the course in Section 1.4. Finally, Section 1.5 provides information about how to install the necessary tools for this course.

## 1.1. Learning goals

At the end of this course students should:

- feel confident to pick up any data set to explore in a hypothesis-driven manner
- have gained the competence to
  - understand complex data sets,
  - manipulate a data set, so as to
  - plot aspects of it in ways that are useful for answering a given research question
- understand the general logic of statistical inference in frequentist and Bayesian approach
- be able to independently evaluate statistical analyses based on their adequacy for a given research question and data set
- be able to critically assess the adequacy of analyses commonly found in the literature

Notice that this is, although a lot of hard work already, still rather modest! It doesn't actually say that we necessarily aim at the competence to *do it* or even to *do it flawlessly!* **Our main goal is understanding**, because that is the foundation of practical success *and* the foundation of an ability to learn more in the future. **We do not teach tricks!** **We do not share recipes!**

## 1.2. Course structure

The course consists of four parts. After giving a more detailed overview of the course, Part I introduces R the main programming language that we will use. Part II covers

## 1. General Introduction

what is often called *descriptive statistics*. It also gives us room to learn more about R when we massage data into shape, compute summary statistics and plot various different data types in various different ways.

Part III is the main theoretical part. It covers what is often called *inferential statistics*. Two aspects distinguish this course from the bulk of its cousins out there. First, we use a **dual-pronged approach**, i.e., we are going to introduce both the frequentist and the Bayesian approach to statistical inference side by side. The motivation for this is that seeing the contrast between the two approaches will aid our understanding of either one. Second, we will use a **computational approach**, i.e., we foster understanding of mathematical notions with computer simulations or other variants of helpful code.

Part IV covers applications of what we have learned so far. It focuses on **generalized linear models**, a class of models that have become the new standard for analyses of experimental data in the social and psychological sciences, but are also very useful for data exploration in other domains (such as machine learning).

There are also appendices with additional information:

- Further useful material (textbooks, manuals, etc.) is provided in Appendix A.
- Appendix B covers the most important probability distributions used in this book.
- An excursion providing more information about the important Exponential Family of probability distributions and the Maximum Entropy Principle is given in Appendix C.
- The data sets which reoccur throughout the book as “running examples” are succinctly summarized in Appendix D.

### 1.3. Tools and topics covered (and not covered) here

The main programming language used in this course is R (R Core Team 2018). We will make heavy use of the *tidyverse* package (Wickham 2017), which provides a unified set of functions and conventions that deviate (sometimes: substantially) from basic R. We will also be using the probabilistic programming language WebPPL (Goodman and Stuhlmüller 2014), but only “passively” in order to quickly obtain results from probabilistic calculations that we can experiment with directly in the browser. We will not learn to write WebPPL code from scratch.

We will rely on the R package **brms** (Bürkner 2017) for running Bayesian generalized regression models, which itself relies on the probabilistic programming language **Stan** (Carpenter et al. 2017). We will, however, not learn about **Stan** in this course. Instead of **Stan** we will use the package **greta** (Golding 2019) to write our models and do inference with them. This is because, for current learning purposes, the language in which **greta** formulates its models is much closer to R and so, let’s hope, easier to learn.

Section 1.5 gives information about how to install these, and other, tools necessary for this course.

The main topics that this course will cover are:

- **data preparation:** how to clean up, and massage a data set into shape for plotting and analysis
- **data visualization:** how to select aspects of data to visualize in informative and useful ways
- **statistical models:** what that is, and why it's beneficial to think in terms of models, not tests
- **statistical inference:** what that is, and how it's done in frequentist and Bayesian approaches
- **hypothesis testing:** how Frequentists and Bayesians test scientific hypotheses
- **generalized regression:** how to apply GRMs to different types of data sets

There is, obviously, a lot that we will *not* cover in this course. We will, for instance, not dwell at any length on the specifics of algorithms for computing statistical inferences or model fits. We will also deal with the history and the philosophy of science of statistics only to the extent that it helps understand the theoretical notions and practical habits that are important in the context of this course. We will also not do heavy math.

Data analysis can be quite varied, because data itself can be quite varied. We try to sample some variation, but since this is an introductory course with lots of other ground to cover, we will be slightly conservative in the kind of data that we analyze. There will, for example, not be any pictures, sounds, dates or time points in any of the material covered here.

There are at least two different motivations for data analysis, and it is important to keep them apart. This course focuses on **data analysis for explanation**, i.e., routines that help us understand reality through the inspection and massaging of empirical data. We will only glance at the alternative approach, which is **data analysis for prediction**, i.e., using models to predict future observations, as commonly practiced in machine learning and its applications. In sloppy slogan form, this course treats data science for scientific knowledge gain, not the engineers' applications.

## 1.4. Data sets covered

We want to learn how to do data analysis. This is impossible without laying hands (keys?) on several data sets. But switching from one data set to another is mentally taxing. It is also difficult to focus and really care about any-old data set. This is why this course relies on a small selection of recurring data sets that are, hopefully, generally interesting for the target audience: students of cognitive science. Appendix D gives an overview of the most important, recurring data sets used in this course.

## *1. General Introduction*

Most of the data sets that we will use repeatedly in this class come from various psychological experiments. To make this even more emersive, these experiments are implemented as browser-based experiments, using `_magpie`. This makes it possible for students of this course to do the exact experiments whose data we are analyzing (and maybe generate some more intuitions, maybe generate some hypotheses) about the very data at hand. But it also makes it possible that we will analyze ourselves. That's why part of the exercises for this course will run additional analyses on data collected from the aspiring data analysts themselves. If you want to become an analyst, you should also have undergone analysis yourself, so to speak.

## **1.5. Installation**

This course relies on a few different pieces of software. Primarily, we'll be using R, but we'll need installations of Python and C++ in the background.

There are two options for installing. The simplest method, described in Section 1.5.1 is to install VirtualBox and use our provided Ubuntu virtual machine (link provided in class), which has the required software pre-installed and tested. When using this method, you will be working in a virtualized Linux environment. Alternatively, you can go through a manual installation tailored to your own OS, as described in Section 1.5.2. Manual installation is recommended if you do not wish to use a virtualized Linux environment. The VirtualBox method can be a fall-back option if manual installation fails. Both methods of installation are detailed below. Finally, Section 1.5.3 explains how to update the R package that wraps all R packages needed for this course and provides some extra convenience functions.

### **1.5.1. VirtualBox Setup**

#### **1.5.1.1. Step 1. Install VirtualBox**

Follow the instructions here to download and install for your platform.

#### **1.5.1.2. Step 2. Download our Ubuntu image**

Download the provided VirtualBox Disk Image and move it into a folder such as “VMs”. The link for the VirtualBox Disk Image is provided on StudIP.

#### **1.5.1.3. Step 3. Create a new virtual machine and add the downloaded image**

- Open VirtualBox and click New
- Give your new virtual machine a name, e.g. “IDA2019”

- Change the Machine Folder to the folder where you put the disk image
- Change the Type to “Linux” and Version to “Ubuntu (64-bit)” and click Next to proceed to memory allocation
- Allocate about half of your available memory and click Next to proceed to hard disk selection
- Choose “Use an existing virtual hard disk file” and use the file selection icon to add our provided disk image
- Click Create

#### **1.5.1.4. Step 4. Give your virtual machine more processing power**

- Select your virtual machine on the right panel and click Settings on the top
- Navigate to System -> Processor
- Increase the Processor(s) to about half of what your computer can provide

#### **1.5.1.5. Step 5. Boot your virtual machine**

- Select your virtual machine and click Start
- The username of the system is “user” and the password is “password”

#### **1.5.1.6. Step 6. Install further packages**

The virtual machine includes most of the packages required, but not all.

- First, run the following commands in ‘Terminal’:

```
sudo apt update  
sudo apt install r-cran-devtools r-cran-boot r-cran-extradistr r-cran-ggsignif  
r-cran-naniar
```

- Then, open RStudio and run the following command in the R console:

```
devtools::install_github("n-kall/IDA2019-package")
```

#### **1.5.1.7. Troubleshooting**

- If the virtual machine does not boot, you may need to ensure that ‘virtualization’ is enabled in your computer’s BIOS. Talk to a tutor if you’re having difficulties doing so.

## 1. General Introduction

### 1.5.2. Manual installation

If you don't want to use the virtual machine, or it doesn't work for you, the following six steps describe how to get the main components installed manually. Depending on your operating system (e.g. macOS, Linux, Windows), you might need to follow slightly different instructions, which are specified. Depending on the exact setup of your computer, the results may vary. The virtual machine has been tested with the required software, so if you can, we recommend using that.

#### 1.5.2.1. Step 1. Install Python

##### Windows and macOS:

We recommend installing miniconda from here

##### Linux:

You can install miniconda or you can just use the preinstalled Python (which saves time and space). Make sure you have pip installed, e.g. for Ubuntu `apt install python3-pip`

#### 1.5.2.2. Step 2. Install the required Python packages

We have provided files that list the required Python packages. They can be installed automatically with the following commands in the terminal.

##### For Anaconda:

Download this environment file

```
conda env create -f environment.yml
```

##### For Linux users who are using pip:

Download this requirements file

```
pip3 install -r requirements.txt
```

#### 1.5.2.3. Step 3. Install R

##### Windows and macOS:

Download and install R from here

##### Linux users:

We need to have at least version 3.5 of R. This may be available in your distribution's repository. e.g. if you are using a recent version of Ubuntu (18.10 or later), you can install R with `apt install r-base`. Otherwise, follow these instructions for your version.

#### 1.5.2.4. Step 4. Install RStudio

##### All platforms:

Download and install the latest version of RStudio from here

#### 1.5.2.5. Step 5. Install a C++ toolchain

For the Stan language, which will be interfaced through an R package called brms, you'll need a working C++ compiler.

##### Windows:

Download and install the latest version of RTools from here

##### macOS:

Download and install the latest version of RTools for macOS from here

Note: you may need to register for an Apple Developer account (free of charge).

##### Linux (e.g. Ubuntu):

You can install a compiler and toolchain with `apt install build-essential`.

#### 1.5.2.6. Step 6. Install R packages

We have created an R package that, when installed, will prompt the installation of the packages we will use in this course. In this step, you'll install this package (and automatically install its dependencies).

##### For Windows and macOS:

Open RStudio and run the following two commands in the console.

```
install.packages("devtools")
devtools::install_github("n-kall/IDA2019-package")
```

##### For Linux (e.g. Ubuntu):

It's much faster (and less error-prone) if you install devtools from the app repository via terminal: `apt install r-cran-devtools` and continue to the second line in the R console. But this will only work if you are using a recent version of Ubuntu (18.10 or later).

If you had to manually update your R to version in step 3, you'll need to install the following from terminal:

```
apt install libssl-dev libxml2-dev libcurl4-openssl-dev
```

and then install via the the R console:

## *1. General Introduction*

```
install.packages("devtools")
devtools::install_github("n-kall/IDA2019-package")
```

### **1.5.3. Updating the course package**

Occasionally, we might have to add packages or functionality as we go through this course. In that case, you will have to update the package `IDA2019-package` that ships all of this for this course. To update, use:

```
devtools::install_github("n-kall/IDA2019-package")
```

## 2. Basics of R

R is a specialized programming language for data science. Though old, it is heavily supported by an active community. New tools for data handling, visualization, and statistical analysis are provided in the form of **packages**.<sup>1</sup> While other programming languages specialized for scientific computing, like Python or Julia, also lend themselves beautifully for data analysis, the choice of R in this course is motivated because R's *raison d'être* is data analysis. Some of the R packages that this course will use provide cutting-edge methods which are not as conveniently available in other programming languages (yet).

In a manner of speaking, there are two flavors of R. We should distinguish **base R** from the **tidyverse**. Base R is what you have when you do not load any packages. We enter the tidyverse by loading the package `tidyverse` (see below for information on how to do that). The tidyverse consists of several components (which are actually stand-alone packages that can be loaded separately if needed) all of which supply extra functionality for data analysis, based on a unifying philosophy and representation format. While eventually interchangeable, the look-and-feel of base R and the tidyverse is quite different. Figure 2.1 lists a selection of packages from the tidyverse in relation to their role at different stages of the process of data analysis. The image is taken from this introduction to the tidyverse.

The course will also introduce Rmarkdown in Section 2.6. Rmarkdown is a nice way of documenting your data analyses in a reproducible form. Participants will use Rmarkdown to prepare their homework assignments.

Make sure to have completely installed everything of relevance for this course, as described in Section 1.5. Unless you have strong opinions or an unassailable favorite, we recommend trying RStudio as an IDE for R.

The official documentation for base R is An Introduction to R. The standard reference for using the tidyverse is R for Data Science (R4DS). There are some very useful cheat sheets which you should definitely check out! There are pointers to further material in Appendix ??.

The learning goals for this chapter are:

---

<sup>1</sup>Packages live in the official package repository CRAN, or are supplied in less standardized forms, e.g., via open repositories, such as GitHub.

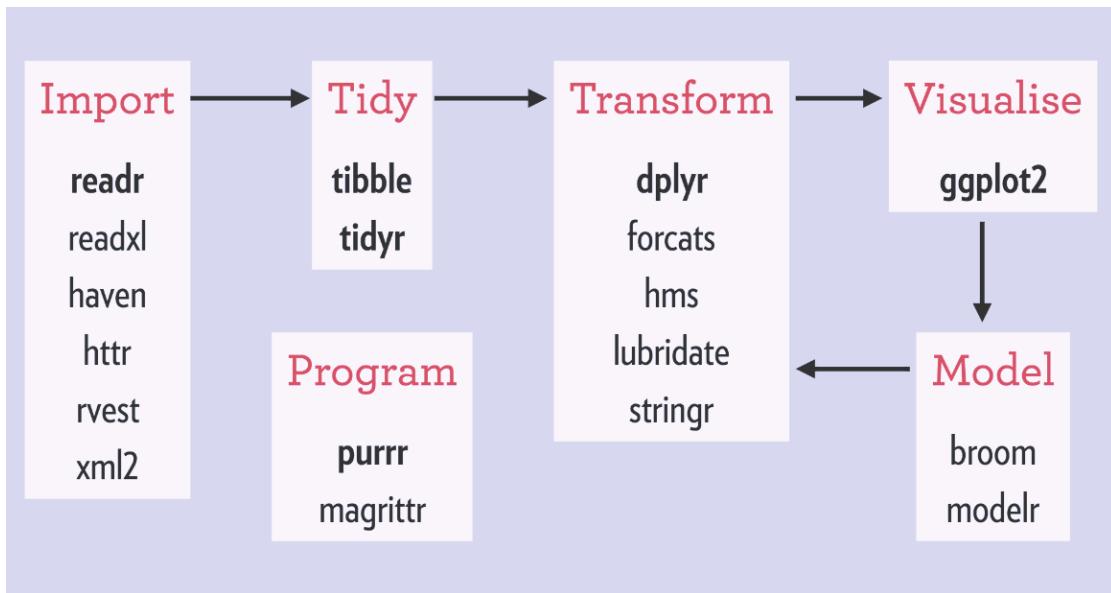


Figure 2.1.: Overview of selected packages from the tidyverse.

- become familiar with R, its syntax and basic notions
- become familiar with the key functionality from the tidyverse
- understand and write simple R scripts
- be able to write documents in Rmarkdown

## 2.1. First steps

R is an interpreted language. This means that you do not have to compile it. You can just evaluate it line by line, in a so-called **session**. The session stores the current values of all variables. If you do not want to retype, you can store your code in a **script**.<sup>2</sup>

Try this out by either typing `r` to open an R session in a terminal or load RStudio.<sup>3</sup> You can immediately calculate stuff:

```
6 * 7
```

```
## [1] 42
```

---

<sup>2</sup>Line-by-line execution of code is useful for quick development and debugging. Make sure to learn about keyboard shortcuts to execute single lines or chunks of code in your favorite editor, e.g., check the RStudio Cheat Sheet for information on its keyboard shortcuts.

<sup>3</sup>When starting a session in a terminal, you can exit a running R session by typing `quit()` or `q()`.

### 2.1.1. Functions

R has many built-in functions. The most common situation is that the function is called by its name using **prefix notation**, followed by round brackets which enclose the function's arguments (separated by commata if multiple). For example, the function **round** takes a number and, per default, returns the closest integer:

```
# the function `round` takes a number as argument and
# returns the closest integer (default)
round(0.6)

## [1] 1
```

Actually, **round** allows several arguments. It takes as input the number **x** to be rounded, and another integer number **digits** which gives the number of digits after the comma to which **x** should be rounded. We can then specify these arguments in a function call of **round** by providing the named arguments.

```
# rounds the number `x` to the number `digits` of digits
round(x = 0.138, digits = 2)

## [1] 0.14
```

When providing all arguments with names, the order of arguments does not matter. When providing at least one non-named argument, all non-named arguments have to be presented in the right order (as expected by the function; to find out what that is use **help**, as explained below in 2.1.6) after subtracting the named arguments from the ordered list of arguments.

```
round(x = 0.138, digits = 2)    # works as intended
round(digits = 2, x = 0.138)    # works as intended
round(0.138, digits = 2)        # works as intended
round(0.138, 2)                # works as intended
round(x = 0.138, 2)             # works as intended
round(digits = 2, 0.138)         # works as intended
round(2, x = 0.138)             # works as intended
round(2, 0.138)                # does not work as intended (returns 2)
```

Functions can have default values for some or all of their arguments. In the case of **round** the default is **digits = 0**. There is obviously no default for **x** in the function **round**.

Some functions can take an arbitrary number of arguments. The function **sum**, which sums up numbers is a point in case.

## 2. Basics of R

```
# adds all of its arguments together  
sum(1,2,3)
```

```
## [1] 6
```

Selected functions can also be called in **infix notation**. This applies to frequently recurring operations, such as mathematical operations or logical comparisons.

```
# both of these calls sum 1, 2, and 3 together  
sum(1,2,3)      # prefix notation  
1 + 2 + 3      # prefix notation
```

Section 2.3 will list some of the most important built-in functions. It will also explain how to define your own functions.

### 2.1.2. Variables

You can assign values to variables using three assignment operators: `->`, `<-` and `=`, like so:

```
x <- 6      # assigns 6 to variable x  
7 -> y      # assigns 7 to variable y  
z = 3        # assigns 3 to variable z  
x * y / z    # returns 6 * 7 / 3 = 14
```

```
## [1] 14
```

Use of `=` is discouraged.<sup>4</sup>

It is good practice to use a consistent naming scheme for variables. This book uses `snake_case_variable_names` and tends towards using `long_and_excessively_informative_names` for important variables, and short variable names, like `i`, `j` or `x`, for local variables, indices etc.

### 2.1.3. Literate coding

It is good practice to document code with short but informative comments. Comments in R are demarcated with `#`.

---

<sup>4</sup>You can produce `<-` in RStudio with Option-- (on Mac) and Alt-- (on Windows/Linux). For other useful keyboard shortcuts, see [here](#).

```
x <- 4711 # a nice number from Cologne
```

Since everything on a line after an occurrence of `#` is treated as a comment, it is possible to break long function calls across several lines, and to add comments to each line:

```
round(          # call the function `round`  
  x = 0.138,    # number to be rounded  
  digits = 2    # number of after-comma digits to round to  
)
```

In RStudio, you can use **Command+Shift+C** (on Mac) and **Ctrl+Shift+C** (on Windows/Linux) to comment or uncomment code, and you can use comments to structure your scripts. Any comment followed by `----` is treated as a (foldable) section.

```
# SECTION: variable assignments ----  
x <- 6  
y <- 7  
# SECTION: some calculations ----  
x * y
```

## 2.1.4. Objects

Strictly speaking, all entities in R are *objects* but that is not always apparent or important for everyday practical purposes see the manual for more information. R supports an object-oriented programming style, but we will not make (explicit) use of this functionality. In fact, this course heavily uses and encourages a functional programming style (see Section 2.4).

Some functions (e.g., optimizers or fitting functions for statistical models) return objects, however, and we will use this output in various ways. For example, if we run a linear regression model on some data set, the output is an object.

```
# you do not need to understand this code  
model_fit = lm(formula = speed~dist, data = cars)  
# just notice that the function `lm` returns an object  
is.object(model_fit)  
  
## [1] TRUE  
  
# printing an object on the screen usually gives you summary information  
print(model_fit)
```

## 2. Basics of R

```
##  
## Call:  
## lm(formula = speed ~ dist, data = cars)  
##  
## Coefficients:  
## (Intercept)      dist  
##     8.2839      0.1656
```

### 2.1.5. Packages

Much of R's charm unfolds through the use of packages. CRAN has the official package repository. To install a new package from a CRAN mirror use the `install.packages` function. For example, to install the package `devtools`, you would use:

```
install.packages("devtools")
```

Once installed, you need to load your desired packages for each fresh session, using:

```
library(devtools)
```

Once loaded all functions, data etc. that ship with a package are available without additional reference to the package name. If you want to be careful or courteous to an admirer of your code, you can reference the package a function comes from explicitly. For example, the following code calls the function `install_github` from the package `devtools` explicitly (so that you would not need to load the package beforehand, for example):

```
devtools::install_github("SOME-URL")
```

Indeed, the `install_github` function allows you to install bleeding-edge packages from github. You can install all of the relevant packages using (after installing the `devtools` package, as described in Section 1.5):

```
devtools::install_github("n-kall/IDA2019-package")
```

After this installation, you can load all packages for this course simply by using:

```
library(IDA2019)
```

In RStudio, there is a special tab in the pane with information on “files”, “plots” etc. to show all installed packages. This also shows which packages are currently loaded.

### 2.1.6. Getting help

If you encounter a function like `lm` that you do not know about, you can access its documentation with the `help` function or just typing `?lm`. For example, the following call summons the documentation for `lm`, the first parts are shown in Figure 2.2.

```
help(lm)
```

```
knitr:::include_graphics("visuals/R-doc-example.png")
```

The screenshot shows the R Documentation page for the `lm` function. At the top left is the package name `lm {stats}`. At the top right is the title `R Documentation`. Below the title is the section `Fitting Linear Models`. Under `Description`, it says: "lm is used to fit linear models. It can be used to carry out regression, single stratum analysis of variance and analysis of covariance (although `aov` may provide a more convenient interface for these)." Under `Usage`, there is a code snippet: 

```
lm(formula, data, subset, weights, na.action,
    method = "qr", model = TRUE, x = FALSE, y = FALSE, qr = TRUE,
    singular.ok = TRUE, contrasts = NULL, offset, ...)
```

. Under `Arguments`, there are two entries: `formula` (an object of class "`formula`" (or one that can be coerced to that class); a symbolic description of the model to be fitted. The details of model specification are given under 'Details') and `data` (an optional data frame, list or environment (or object coercible by `as.data.frame` to a data frame) containing the variables in the model. If not found in `data`, the variables are taken from `environment(formula)`, typically the environment from which `lm` is called).

Figure 2.2.: Excerpt from the documentation of the ‘`lm`’ function.

If you are looking for help on a more general topic, use the function `help.search`. It takes a regular expression as input and outputs a list of occurrences in the available documentation. A useful shortcut for `help.search` is just to type `??` followed by the (unquoted) string to search for. For example, calling either of the following lines might produce a display like in Figure 2.3.

```
# two equivalent ways for obtaining help on search term 'linear'
help.search("linear")
??linear
```

```
knitr:::include_graphics("visuals/R-doc-search-example.png")
```

The top entries in Figure 2.3 are **vignettes**. These are compact manuals or tutorial on particular topics or functions, and they are directly available in R. If you want to browse through the vignettes available on your machine (which depend on which packages you have installed), go ahead:

## 2. Basics of R

Search Results 

---



---

**Vignettes:**

|   |  |                      |                        |                        |
|---|--|----------------------|------------------------|------------------------|
| <a href="#">brms::brms_nonlinear</a>              | Estimating Non-Linear Models with brms                           | <a href="#">HTML</a> | <a href="#">source</a> | <a href="#">R code</a> |
| <a href="#">knitr::docco-linear</a>               | R Markdown with the Docco Linear Style                           | <a href="#">HTML</a> | <a href="#">source</a> | <a href="#">R code</a> |
| <a href="#">lme4::lmer</a>                        | Fitting Linear Mixed-Effects Models using lme4                   | <a href="#">PDF</a>  | <a href="#">source</a> | <a href="#">R code</a> |
| <a href="#">RcppEigen::RcppEigen-Introduction</a> | RcppEigen-intro  | <a href="#">PDF</a>  | <a href="#">source</a> | <a href="#">R code</a> |
| <a href="#">SparseM::SparseM</a>                  | An Introduction to the SparseM Package for Sparse Linear Algebra | <a href="#">PDF</a>  | <a href="#">source</a> | <a href="#">R code</a> |

**Code demonstrations:**

|  |   |                                       |
|--|---|---------------------------------------|
| <a href="#">quantreg::Frank</a>        | Demo of nonlinear in parameters fitting of Frank copula model   | <a href="#">(Run demo in console)</a> |
| <a href="#">SparseM::LeastSquares</a>  | Least Squares Linear Regression   | <a href="#">(Run demo in console)</a> |
| <a href="#">SparseM::LinearAlgebra</a> | Basic Linear Algebra for Sparse Matrices  | <a href="#">(Run demo in console)</a> |
| <a href="#">SparseM::Solve</a>         | Linear Equation Solving   | <a href="#">(Run demo in console)</a> |
| <a href="#">stats::lm.glm</a>          | Some linear and generalized linear modelling examples from 'An Introduction to Statistical Modelling' by Annette Dobson | <a href="#">(Run demo in console)</a> |
| <a href="#">stats::nlm</a>             | Nonlinear least-squares using nlm()   | <a href="#">(Run demo in console)</a> |

**Help pages:**

|   |   |
|---|---|
| <a href="#">arrayhelpers::colMeans.array-method</a> | Row and column sums and means for numeric arrays. |
|---|---|

Figure 2.3.: Result of calling ‘help.search‘ for the term ‘linear’.

```
browseVignettes()
```

## 2.2. Data types

Let's briefly go through the data types that are most important for our later purposes. We can assess the type of an object stored in variable `x` with the function `typeof(x)`.

```
typeof(3)          # returns type "double"
typeof(TRUE)       # returns type "logical"
typeof(cars)       # returns 'list' (includes data.frames, tibbles, objects, ...)
typeof("huhu")    # return 'character' (= string)
typeof(mean)        # return 'closure' (= function)
typeof(c)          # return 'builtin' (= deep system internal stuff)
typeof(round)      # returns type "special" (= well, special stuff?)
```

To learn more about an object, it can help to just print it out as a string:

```
str(lm)
```

```
## function (formula, data, subset, weights, na.action, method = "qr", model = TRUE,
##   x = FALSE, y = FALSE, qr = TRUE, singular.ok = TRUE, contrasts = NULL,
##   offset, ...)
```

It is sometimes possible to cast objects of one type into another type `XXX` using functions `as.XXX` in base R or `as_XXX` in the tidyverse.

```
# casting Boolean value `TRUE` into number format
as.numeric(TRUE) # returns 1

## [1] 1
```

R is essentially an array-based language. Arrays are arbitrary but finite dimensional matrices. We will discuss what is usually referred to as vectors (= one-dimensional arrays), matrices (= two-dimensional arrays) and arrays (= more-than-two-dimensional) in the following section on numeric information. But it is important to keep in mind that arrays can contain objects of other types than numeric information (as long as all objects in the array are of the same type).

## 2. Basics of R

### 2.2.1. Numeric vectors & matrices

#### 2.2.1.1. Numeric information

Standard number format in R is double.

```
typeof(3)
```

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

We can also represent numbers as integers and complex.

```
typeof(as.integer(3))      # returns 'integer'
```

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

```
typeof(as.complex(3))      # returns 'complex'
```

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

#### 2.2.1.2. Numeric vectors

As a generally useful heuristic, expect every numerical information to be treated as a vector (or higher-order: matrix, array, ... ; see below), and to expect any (basic, mathematical) operation in R to (most likely) apply to the whole vector, matrix, array, collection.<sup>5</sup> This makes it possible to ask for the length of a variable to which we assing a single number, for instance:

```
x <- 7  
length(x)
```

```
## [1] 1
```

We can even index such a variable:

```
x <- 7  
x[1]      # what is the entry in position 1 of the vector x?
```

---

<sup>5</sup>If you are familiar with Python's *scipy* and *numpy* packages, this is R's default mode of treating numerical information.

```
## [1] 7
```

Or assign a new value to a hitherto unused index:

```
x[3] <- 6    # assign the value 6 to the 3rd entry of vector x
x           # notice that the 2nd entry is undefined, or "NA", not available

## [1] 7 NA 6
```

Vectors in general can be declared with the built-in function `c()`. To memorize this, think of *concatenation* or *combination*.

```
x <- c(4, 7, 1, 1)    # this is now a 4-place vector
x

## [1] 4 7 1 1
```

There are also helpful functions to generate sequences of numbers:

```
1:10                                # returns 1, 2, 3, ..., 10
seq(from = 1, to = 10, by = 1)        # returns 1, 2, 3, ..., 10
seq(from = 1, to = 10, by = 0.5)       # returns 1, 1.5, 2, ..., 9.5, 10
seq(from = 0, to = 1, length.out = 11)  # returns 0, 0.1, ..., 0.9, 1
```

Indexing in R starts with 1, not 0!

```
x <- c(4, 7, 1, 1)    # this is now a 4-place vector
x[2]

## [1] 7
```

And now we see what is meant above when we said that (almost) every mathematical operation can be expected to apply to a vector:

```
x <- c(4, 7, 1, 1)    # 4-placed vector as before
x + 1

## [1] 5 8 2 2
```

### 2.2.1.3. Numeric matrices

Matrices are declared with the function `matrix`. This function takes, for instance, a vector as an argument.

## 2. Basics of R

```
x <- c(4, 7, 1, 1)      # 4-placed vector as before
(m <- matrix(x))       # cast x into matrix format

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

Notice that the result is a matrix with a single column. This is important. R uses so-called *column-major mode*.<sup>6</sup> This means that it will fill columns first. For example, a matrix with three columns based on a six-placed vector 1, 2, ..., 6 will be built by filling the first column from top to bottom, then the second column top to bottom, and so on.<sup>7</sup>

```
m <- matrix(1:6, ncol = 3)
m
```

```
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

In line with column-major mode, vectors are treated as column vectors in matrix operations:

```
x = c(1,0,1)    # 3-place vector
m %*% x         # dot product with previous matrix 'm'

##      [,1]
## [1,]    6
## [2,]    8
```

As usual, and independently of column- or row-major mode, matrix indexing starts with the row index:

```
m[1,]    # produces first row of matrix 'm'

## [1] 1 3 5
```

---

<sup>6</sup>Python, on the other hand, uses the reverse *row-major mode*.

<sup>7</sup>It is in this sense that the “first index moves fastest” in column-major mode, which is another frequently given explanation of column-major mode.

### 2.2.1.4. Arrays

Arrays are simply higher-dimensional matrices. We will not make use of arrays in this course.

### 2.2.1.5. Names for vectors, matrices and arrays

The positions in a vector can be given names. This is extremely useful for good “literate coding” and therefore highly recommended. The names of vector `x`’s positions are retrieved and set by the `names` function:

```
students <- c("Jax", "Jamie", "Jason") # names of students
grades <- c(1.3, 2.7, 2.0)           # a vector of grades
names(grades)                      # retrieve names: with no names so far

## NULL

names(grades) <- students          # assign names
names(grades)                      # retrieve names again: names assigned

## [1] "Jax"   "Jamie" "Jason"

grades                         # output shows names

##   Jax Jamie Jason
##   1.3   2.7   2.0
```

We can also set the names of a vector directly during construction:<sup>8</sup>

```
# names of students (this is a character vector, see below)
students <- c("Jax", "Jamie", "Jason")
# constructing a vector with names directly assigned
grades <- c(1.3, 2.7, 2.0, names = students)
```

Names for matrices are retrieved or set with functions `rownames` and `colnames`.

---

<sup>8</sup>Notice that we can create strings (actually called ‘characters’ in R) with double quotes

## 2. Basics of R

```
# declare matrix
m <- matrix(1:6, ncol = 3)
# assign row and column names, using function
# `str_c` which is described below
rownames(m) <- str_c("row", 1:nrow(m), sep = "_")
colnames(m) <- str_c("col", 1:ncol(m), sep = "_")
m

##      col_1 col_2 col_3
## row_1     1     3     5
## row_2     2     4     6
```

### 2.2.2. Booleans

There are built-in names for Boolean values “true” and “false”, predictably named TRUE and FALSE. Equivalent shortcuts are T and F. If we attempt to do math with Boolean vectors, the outcome is what any reasonable logician would expect:

```
x <- c(T,F,T)
1 - x
```

```
## [1] 0 1 0
```

```
x + 3
```

```
## [1] 4 3 4
```

Boolean vectors can be used as index sets to extract elements from other vectors.

```
# vector 1, 2, ..., 5
number_vector <- 1:5
# index of odd numbers set to `TRUE`
boolean_vector <- c(T,F,T,F,T)
# returns the elemnts from number vector, for which
# the corresponding element in the Boolean vector is true
number_vector[boolean_vector]

## [1] 1 3 5
```

### 2.2.3. Special values

There are a couple of keywords reserved in R for special kinds of objects:

- NA: “not available”; represents missing values in data
- NaN: “not a number”; e.g., division zero by zero
- Inf or -Inf: infinity and negative infinity; returned when number is too big or devision by zero
- NULL: the NULL object; often returned when function is undefined for input

### 2.2.4. Characters (= strings)

Strings are called characters in R. We will be stubborn and call them strings for most of the time here. We can assign a string value to a variable by putting the string in double quotes:

```
x <- "huhu"
typeof(x)

## [1] "character"
```

We can create vectors of characters in the obvious way:

```
chr_vector <- c("huhu", "hello", "huhu", "ciao")
chr_vector

## [1] "huhu"  "hello"  "huhu"  "ciao"
```

The package **stringr** from the tidyverse also provides very useful and, in comparison to base R, more uniform functions for string manipulation. The cheat sheet for the **stringr** package is highly recommended for a quick overview. Below are some examples.

Function **str\_c** concatenates strings:

```
str_c("Hello", "Hi", "Hey", sep = "! ")
## [1] "Hello! Hi! Hey"
```

We can find the indeces of matches in a character vector with **strwhich**:

## 2. Basics of R

```
chr_vector <- c("huhu", "hello", "huhu", "ciao")
str_which(chr_vector, "hu")

## [1] 1 3
```

Similarly, `str_detect` gives a Boolean vector of matching:

```
chr_vector <- c("huhu", "hello", "huhu", "ciao")
str_detect(chr_vector, "hu")

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

If we want to get the strings matching a pattern, we can use `str_subset`:

```
chr_vector <- c("huhu", "hello", "huhu", "ciao")
str_subset(chr_vector, "hu")

## [1] "huhu" "huhu"
```

Replacing all matches with another string works with `str_replace_all`:

```
chr_vector <- c("huhu", "hello", "huhu", "ciao")
str_replace_all(chr_vector, "h", "B")

## [1] "BuBu" "Bello" "BuBu" "ciao"
```

For data preparation we often need to split strings by a particular character. For instance, a set of reaction times could be separated by a character line “|”. We can split this string representation to get individual measurements like so:

```
# three measures of reaction time in a single string
reaction_times <- "123|234|345"
# notice that we need to doubly (!) escape character |
# notice also that the results is a list (see below)
str_split(reaction_times, "\\|", n = 3)

## [[1]]
## [1] "123" "234" "345"
```

### 2.2.5. Factors

Factors are special vectors, which treat its elements as ordered or unorderd categories. This is useful for representing data from experiments, e.g., of categorical or ordinal variables (see Chapter 3). To create a factor, we can use the function `factor`. The following code creates an *unorderd factor*:

```
chr_vector <- c("huhu", "hello", "huhu", "ciao")
factor(chr_vector)
```

```
## [1] huhu hello huhu ciao
## Levels: ciao hello huhu
```

*Ordered factors* also register the order of the categories:

```
chr_vector <- c("huhu", "hello", "huhu", "ciao")
factor(
  chr_vector,      # the vector to treat as factor
  ordered = T,    # make sure its treated as ordered factor
  levels = c("huhu", "ciao", "hello") # specify order of levels
)
```

```
## [1] huhu hello huhu ciao
## Levels: huhu < ciao < hello
```

We will see that ordered factors are important, for example, in plotting when they determine the order in which different parts of data are arranged on the screen. They are also important for statistical analysis, because they help determine how categories are compared to one another.

Factors are trickier to work with than mere lists, because they are rigid about the represented factor levels. Adding an item that does not belong to any of a factor's levels, leads to trouble:

```
chr_vector <- c("huhu", "hello", "huhu", "ciao")
my_factor <- factor(
  chr_vector,      # the vector to treat as factor
  ordered = T,    # make sure its treated as ordered factor
  levels = c("huhu", "ciao", "hello") # specify order of levels
)
my_factor[5] <- "huhu" # adding a "known category" is okay
my_factor[6] <- "moin" # adding an "unknown category" does not work
my_factor
```

## 2. Basics of R

```
## [1] huhu hello huhu ciao huhu <NA>
## Levels: huhu < ciao < hello
```

The **forcats** package from the tidyverse helps dealing with factors. You should check the Cheat Sheet for more helpful functionality. Here is an example of how to expand the levels of a factor:

```
chr_vector <- c("huhu", "hello", "huhu", "ciao")
my_factor <- factor(
  chr_vector,      # the vector to treat as factor
  ordered = T,     # make sure its treated as ordered factor
  levels = c("huhu", "ciao", "hello") # specify order of levels
)
my_factor[5] <- "huhu" # adding a "known category" is okay
my_factor <- fct_expand(my_factor, "moin") # add new category
my_factor[6] <- "moin" # adding new item now works
my_factor

## [1] huhu hello huhu ciao huhu moin
## Levels: huhu < ciao < hello < moin
```

It is sometimes useful (especially for plotting) to flexibly reorder the levels of an ordered factor. Here are some useful functions from the **forcats** package:

```
my_factor           # original factor

## [1] huhu hello huhu ciao huhu moin
## Levels: huhu < ciao < hello < moin

fct_rev(my_factor)    # reverse level order

## [1] huhu hello huhu ciao huhu moin
## Levels: moin < hello < ciao < huhu

fct_relevel(          # manually supply new level order
  my_factor,
  c("hello", "ciao", "huhu")
)

## [1] huhu hello huhu ciao huhu moin
## Levels: hello < ciao < huhu < moin
```

### 2.2.6. Lists, data frames & tibbles

Lists are key-value pairs. They are created with the built-in function `list`. The difference between a list and a named vector is that in the latter all elements must be of the same type. In a list, the elements can be of arbitrary type. They can also be vectors or even lists themselves. For example:

```
my_list <- list(
  single_number = 42,
  chr_vector    = c("huhu", "ciao"),
  nested_list    = list(x = 1, y = 2, z = 3)
)
my_list

## $single_number
## [1] 42
##
## $chr_vector
## [1] "huhu" "ciao"
##
## $nested_list
## $nested_list$x
## [1] 1
##
## $nested_list$y
## [1] 2
##
## $nested_list$z
## [1] 3
```

To access a list element by its name (=key), we can use the `$` sign followed by the unquoted name, double square brackets `[[ "name" ]]` with the quoted name inside, or indices in double brackets, like so:

```
# all of these return the same list element
my_list$chr_vector

## [1] "huhu" "ciao"

my_list[["chr_vector"]]

## [1] "huhu" "ciao"
```

## 2. Basics of R

```
my_list[[2]]  
  
## [1] "huhu" "ciao"
```

Lists are very important in R because almost all structured data that belongs together is stored as lists. Objects are special kinds of lists. Data is stored in special kinds of lists, so-called *data frames* or so-called *tibbles*.

A data frame is base R's standard format to store data in. A data frame is a list of vectors of equal length. Data sets are instantiated with the function `data.frame`:

```
# fake experimental data  
exp_data <- data.frame(  
  trial = 1:5,  
  condition = factor(  
    c("C1", "C2", "C1", "C3", "C2"),  
    ordered = T  
,  
  response = c(121, 133, 119, 102, 156)  
)  
exp_data  
  
##   trial condition response  
## 1      1        C1     121  
## 2      2        C2     133  
## 3      3        C1     119  
## 4      4        C3     102  
## 5      5        C2     156
```

We can access columns of a data frame, just like we access elements in a list. Additionally, we can also use index notation, like in a matrix:

```
# gives the value of the cell in row 2, column 3  
exp_data[2,3] # return 133  
  
## [1] 133
```

In RStudio, you can inspect data in data frames (and tibbles (see below)) with the function `View`.

*Tibbles* are the tidyverse counterpart of data frames. We can cast a data frame into a tibble, using `as_tibble`.

```
as_tibble(exp_data)

## # A tibble: 5 x 3
##   trial condition response
##   <int>    <ord>     <dbl>
## 1      1     C1       121
## 2      2     C2       133
## 3      3     C1       119
## 4      4     C3       102
## 5      5     C2       156
```

But we can also create a tibble directly with the keyword `tibble`. Indeed, creation of tibbles is conveniently more flexible than the creation of data frames: the former allow dynamic look-up of previously defined elements.

```
my_tibble <- tibble(x = 1:10, y = x^2)      # dynamic construction possible
my_dataframe <- data.frame(x = 1:10, y = x^2) # ERROR :/
```

Another important difference between data frames and tibbles concerns default treatment of character (=string) vectors. When reading in data from a CSV file as a data frame (using function `read.csv`) each character vector is treated as a factor per default. But when using `read_csv` to read CSV data into a tibble character vector are not treated as factors.

## 2.3. Functions

### 2.3.1. Some important built-in functions

Many helpful functions are defined in base R or supplied by packages. We recommend browsing the Cheat Sheets every now and then to pick up more useful stuff for your inventory. Here are some functions that are very basic and generally useful.

#### 2.3.1.1. Standard logic

- `&`: “and”
- `|`: “or”
- `!`: “not”
- `negate()`: a pipe-friendly ! (see Section 2.5 for more on piping)
- `all()`: returns true of a vector if all elements are T
- `any()`: returns true of a vector if at least one element is T

## 2. Basics of R

### 2.3.1.2. Comparisons

- <: smaller
- >: greater
- ==: equal (you can also use `near()` instead of == e.g. `near(3/3,1)` returns TRUE)
- >=: greater or equal
- <=: less or equal
- !=: not equal

### 2.3.1.3. Set theory

- %in%: whether an element is in a vector
- `union(x,y)`: union of x and y
- `intersect(x,y)`: intersection of x and y
- `setdiff(x,y)`: all elements in x that are not in y

### 2.3.1.4. Sampling and combinatorics

- `runif()`: random number from unit interval [0;1]
- `sample(x, size, replace)`: take `size` samples from `x` (with replacement if `replace` is T)
- `choose(n,k)`: number of subsets of size `n` out of a set of size `k` (binomial coefficient)

## 2.3.2. Defining your own functions

If you find yourself in a situation in which you would like to copy-paste some code, possibly with minor amendments, this usually means that you should wrap some recurring operations into a custom-defined function.

There are two ways of defining your own functions: as a named function, or an anonymous function.

### 2.3.2.1. Named functions

The special operator supplied by base R to create new functions is the keyword `function`. Here is an example of defining a new function with two input variables `x` and `y` that returns a computation based on these numbers. We assign this newly created function to the variable `cool_function`, so that we can use this name to call the function later. Notice that the use of the `return` keyword is optional here. If it is left out, the evaluation of the last line is returned.

```
# define a new function
# takes two numbers x & y as argument
# return x * y + 1
cool_function <- function(x, y) {
  return(x * y + 1)
}

# apply `cool_function` to some numbers:
cool_function(3,3)      # return 10
cool_function(1,1)      # return 2
cool_function(1:2,1)    # returns vector [2,3]
cool_function(1)        # throws error: 'argument "y" is missing, with no default'
cool_function()         # throws error: 'argument "x" is missing, with no default'
```

We can give default values for the parameters passed to a function:

```
# same function as before but with
# default values for each argument
cool_function_2 <- function(x = 2, y = 3) {
  return(x * y + 1)
}

# apply `cool_function_2` to some numbers:
cool_function_2(3,3)      # return 10
cool_function_2(1,1)      # return 2
cool_function_2(1:2,1)    # returns vector [2,3]
cool_function_2(1)        # returns 4 (= 1 * 3 + 1)
cool_function_2()         # returns 7 (= 2 * 3 + 1)
```

### 2.3.2.2. Anonymous functions

Notice that we can feed functions as parameters to other functions. This is an important ingredient of a functional-style of programming, and something that we will rely on heavily in this course (see Section 2.4). When supplying a function as an argument to another function, we might not want to name the function that is passed. Here's a (stupid, but hopefully illustrating) example:

```
# define a function that takes a function as argument
new_applier_function <- function(input, function_to_apply) {
  return(function_to_apply(input))
}
```

## 2. Basics of R

```
# sum vector with built-in & named function
new_applier_function(
  input = 1:2,                      # input vector
  function_to_apply = sum    # built-in & named function to apply
)  # returns 3

# sum vector with anonymous function
new_applier_function(
  input = 1:2,                      # input vector
  function_to_apply = function(input) {
    return(input[1] + input[2])
}
)  # returns 3 as well
```

## 2.4. Loops and maps

For iteratively performing computation steps, R has a special syntax for `for` loops. Here is an example of a (stupid, but illustrative) example of a `for` loop in R:

```
# fix a vector to transform
input_vector  <- 1:6

# create output vector for memory allocation
output_vector  <- integer(length(input_vector))

# iterate over length of input
for (i in 1:length(input_vector)) {
  # multiply by 10 if even
  if (input_vector[i] %% 2 == 0) {
    output_vector[i] = input_vector[i] * 10
  }
  else {
    output_vector[i] = input_vector[i]
  }
}

output_vector

## [1]  1 20  3 40  5 60
```

R also provides functional iterators (e.g., `apply`), but we will use the functional iterators from the `purrr` package. The main functional operator from `purrr` is `map` which takes

a vector and a function, applies the function to each element in the vector and returns a list with the outcome. There are also versions of `map`, written as `map_dbl` (double), `map_lgl` (logical) or `map_df` (data frame), which return a vector of doubles, Booleans or a data frame. Here is a first example of how this code looks in a functional style using the functional iterator `map_dbl`:

```
map_dbl(
  input_vector,
  function(i)  {
    if (input_vector[i] %% 2 == 0) {
      return (input_vector[i] * 10)
    }
    else {
      return (input_vector[i])
    }
  }
)

## [1] 1 20 3 40 5 60
```

We can write this even shorter, using `purrr`'s short-hand notation for functions:

```
map_dbl(
  input_vector,
  ~ ifelse( .x %% 2 == 0, .x * 10, .x)
)

## [1] 1 20 3 40 5 60
```

The trailing `~` indicates that we define an anonymous function. It therefore replaces the usual `function(...)` call which indicates which arguments the anonymous function expects. To make up for this, after the `~` we can use `.x` for the first (and only) argument of our anonymous function.

To apply a function to more than one input vector, element per element, we can use `pmap` and its derivatives, like `pmap_dbl` etc. `pmap` takes a list of vectors and a function. In short-hand notation we can define an anonymous function with `~` and integers like `..1`, `..2` etc, for the first, second ... argument. For example:

```
x <- 1:3
y <- 4:6
z <- 7:9
```

## 2. Basics of R

```
pmap_dbl(  
  list(x, y, z),  
  ~ ..1 - ..2 + ..3  
)  
  
## [1] 4 5 6
```

## 2.5. Piping

When we use a functional style of programming, piping is your best friend. Consider the standard example of applying functions in what linguists would call “center-embedding”. We start with the input (written inside the inner-most bracketing), then apply the first function `round`, then the second `mean`, writing each next function call “around” the previous.

```
# define input  
input_vector <- c(0.4, 0.5, 0.6)  
  
# first round, then take mean  
mean(round(input_vector))  
  
## [1] 0.3333333
```

Things quickly get out of hand when more commands are nested. A common practice is to store intermediate results of computations in new variables which are only used to pass the result into the next step.

```
# define input  
input_vector <- c(0.4, 0.5, 0.6)  
  
# rounded input  
rounded_input <- round(input_vector)  
  
# mean of rounded input  
mean(rounded_input)  
  
## [1] 0.3333333
```

Piping let’s you pass the result of a previous function call into the next. The `magrittr` package supplies a special infix operator `%>%` for piping.<sup>9</sup> The pipe `%>%` essentially takes

---

<sup>9</sup>The pipe symbol `%>%` can be inserted in RStudio with Ctrl+Shift+M (Win/Linux) or Cmd+Shift+M (Mac).

what results from evaluating the expression on its left-hand side and inputs it as the first argument in the function on its right-hand side. So `x %>% f` is equivalent to `f(x)`. Or, to continue the example from above, we can now write:

```
input_vector %>% round %>% mean  
## [1] 0.3333333
```

The functions defined as part of the tidyverse are all constructed in such a way that the first argument is the most likely input you would like to pipe into them. But if you want to pipe the left-hand side into another argument slot than the first, you can do that by using the `.` notation to mark the slot where the left-hand side should be piped into: `y %>% f(x, .)` is equivalent to `f(x,y)`.

## 2.6. Rmarkdown

Howework assignments will be issued, filled and submitted in Rmarkdown. To get familiar with Rmarkdown, please follow this tutorial.



# **Part II.**

# **Data**



# 3. Data, variables & experimental designs

The focus of this course is on data from behavioral psychology experiments.<sup>1</sup> In a sense, this is perhaps the most “well-behaved” data to analyze and therefore an excellent starting point into data analysis. However, we should not lose sight of the rich and diverse guises of data that are relevant for scientific purposes. The current chapter therefore starts, in Section 3.1, with sketching some of that richness and diversity. But it then hones in on some basic distinctions of the kinds of data we will frequently deal with in the cognitive sciences in Section 3.2. We also pick up a few relevant concepts from standard experimental design in Section 3.3.

The learning goals for this chapter are:

- appreciate the diversity of data
- distinguish different kinds of variables
  - dependent vs independent
  - nominal vs ordinal vs metric
- get familiar with basic aspects of experimental design
  - factorial designs
  - within- vs between subjects design
  - repeated measures
  - randomization, fillers and controls
  - sample size
- understand the notion of “tidy data”

## 3.1. Different kinds of data

Some say we live in the **data age**. But what is data actually? Purist pedants say: “The plural of datum” and add that a datum is just an observation. But when we say “data”

---

<sup>1</sup>A *behavioral experiment* is an experiment that records participants’ behavioral choices, such as button clicks or linguistic responses in the form of text or speech. This contrasts with, say, *neurological experiments* in which participants’ brain activity is recorded, such as fMRI or EEG, or, e.g., in a psycholinguistic context, *processing-related experiments* in which secondary measures of cognitive activity are measured, such as eye-movements, pupil dilation or galvanic skin responses.

### 3. Data, variables & experimental designs

we usually mean a bit more than a bunch of observations. The Merriam-Webster offers the following:

Factual information (such as measurements or statistics) used as a basis for reasoning, discussion, or calculation.

This is a teleological definition in the sense that it refers to the purpose of the thing: “used as basis for reasoning, discussion, or calculation”. So, what we mean by “data” is in large part defined by what we intend to do with it. Another important aspect of this definition is that we usually consider data to be systematically structured in some way or another. Even when we speak of “raw data”, we expect there to be some structure (maybe labels, categories etc.) that distinguishes data from uninterpretable noise (e.g., the notion of a “variable”, discussed in Section 3.2). In sum, we can say that **data is a representation of information stored in a systematic way for the purpose of inference, argument or decision making**.

There are different kinds of data. Figure 3.1 shows some basic distinctions, represented in a conceptual hierarchy.

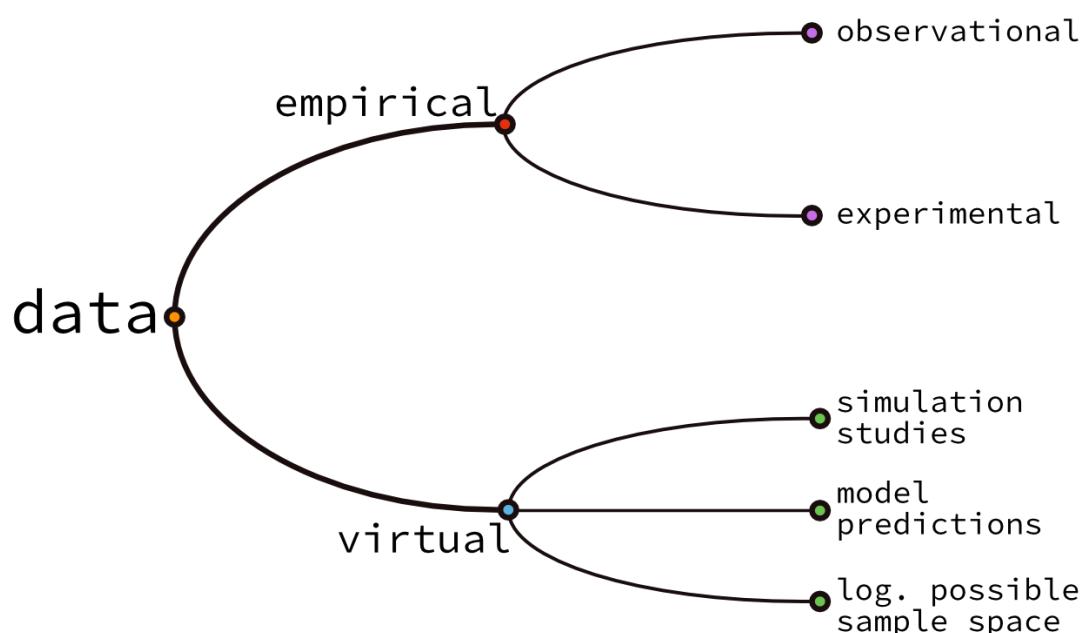


Figure 3.1.: Hierarchy of different kinds of data relevant for 'data science'.

It is easy but wrong to think that data always has to be information based on observations of the world. It is easy to think this because **empirical data**, i.e., data obtained from empirical observation, is the most common form of data (given that it is, arguably, most relevant for decision making and argument). But it is wrong to think this because we

### 3.1. Different kinds of data

Table 3.1.: Comparison of pro's and cons's of observational data and experimental data.

| observational                               | experimental                                       |
|---|--|
| ecological valid                            | possibly artificial                                |
| easy/easier to obtain                       | hard/harder to obtain                              |
| correlation & causation hard to tease apart | may yield information on causation vs. correlation |

can just as well look at **virtual data**. For example, virtual data which is of interest to a data analyst could be **data obtained from computer simulation studies**, e.g., from, say, a 1 billion runs of a multi-agent simulation intended to shed light on the nature of cooperative interaction. It makes sense to analyse such data with the same tools as data from an experiment. We might find out that some parameter constellations in the simulation run are (statistically) most conducive of producing cooperative behavior among our agents, for instance. Another example of virtual data is **data generated as predictions of a model**, which we can use to test whether that model is any good, in so-called model criticism (see Section 14).<sup>2</sup> Finally, we should also include **logically possible sample data** in this list, because of its importance to central ideas of statistical inference (especially *p*-values, see Section 11). Logically possible sample data is that was neither observed, nor predicted by a model, but something that could have been observed hypothetically, something that it is merely logically possible to observe, even if it would almost never happen in reality or would not be predicted by any serious model.

The most frequent form of data, **empirical data** about the actual world, comes in two major variants. **Observational data** is data gathered by (passively) observing and recording what would have happened even if we had not been interested in it, so to speak. Examples of observational data are collections of socio-economic variables, like gender, education, income, number of children etc. In contrast, **experimental** data is data recorded in a strict regime of manipulation-and-observation, i.e., a scientific experiment. Some pieces of information can only be recorded in an observational study (annual income) and others can only be obtained through experimentation (memory span). Both methods of data acquisition have their own pros and cons. Here are some of the more salient ones:

No matter what kind of data we have at hand, there are at least two prominent purposes for which data can be useful: **explanation** and **prediction**. Though related, it is useful to keep these purposes cleanly apart. Data analysis for explanation uses the data to better understand the source of the data (the world, a computer simulation, a model, etc.). Data analysis for prediction tries to extract regularities from the data gathered so far to make predictions (as accurately as possible) about future or hitherto unobserved data.

---

<sup>2</sup>We will later speak of **prior/posterior predictions** for this kind of data. Other applicable terms are **repeat data** or sometimes **fake data**.

### 3. Data, variables & experimental designs

## 3.2. On the notion of “variables”

Data used for data analysis, even if it is “raw data”, i.e., data before preprocessing and cleaning, is usually structured or labelled in some way or other. Even if the whole data we have is a vector of numbers, we would usually know what these numbers represent. For instance, we might just have a quintuple of numbers, but we would (usually, ideally) know that these represent the results of an IQ test.

```
# a simple data vector of IQ-scores
IQ_scores <- c(102, 115, 97, 126, 87)
```

Or we might have a Boolean vector with the information of whether each of five students passed an exam. But even then we would (usually/ideally) know the association between names and test results, as in a table like this:

```
# who passed the exam
exam_results <-
  tribble(
    ~student,    ~pass,
    "Jax",      TRUE,
    "Jason",    FALSE,
    "Jamie",    TRUE
  )
```

Association of information, as between different columns in a table like the one above, is crucial. Most often we have more than one kind of observation that we care about. Most often, we care about systematic relationships between different observables in the world. For instance, we might want to look at pass/fail results from an exam in co-occurrence with information about the proportion of attendance of the course’s tutorial sessions:

```
# proportion of tutorials attended and exam pass/fail
exam_results <-
  tribble(
    ~student,    ~tutorial_proportion,    ~pass,
    "Jax",        0.0,                      TRUE,
    "Jason",     0.78,                     FALSE,
    "Jamie",     0.39,                     TRUE
  )
exam_results

## # A tibble: 3 x 3
##   student tutorial_proportion pass
##   <chr>          <dbl>    <lgl>
```

Table 3.2.: Common / natural formats for representing data of different kinds in R.

| variable type    | representation in R |
|------------------|---------------------|
| nominal / binary | unordered factor    |
| Boolean          | logical vector      |
| ordinal          | ordered factor      |
| metric           | numeric vector      |

```
## 1 Jax          0    TRUE
## 2 Jason        0.78 FALSE
## 3 Jamie        0.39 TRUE
```

Data of this kind is also called **rectangular data**, i.e., data that fits into a rectangular table (More on the structure of rectangular data in Section 4.2.). In the example above, every column represents a **variable** of interest. A (*data*) **variable** stores the observations that are of the same kind.<sup>3</sup>

Common kinds of variables are distinguished based on the structural properties of the kinds of observations that they contain. Common types of empirical data collected are:

- **nominal variable**: each observation is an instance of a (finite) set of clearly distinct categories, lacking a natural ordering;
- **binary variable**: special case of nominal variable when there are only two categories;
- **Boolean variable**: special case of a binary variable when the two categories are Boolean values “true” and “false”;
- **ordinal variable**: each observation is an instance of a (finite) set of clearly distinct and naturally ordered categories, but there is no natural meaning of distance between categories (i.e., it makes sense to say that A is “more” than B but not that A is three times “more” than B);
- **metric variable**: each observation is isomorphic to a subset of the reals, and interval-scaled (i.e., it makes sense to say that A is three times “more” than B);

Examples of some different kinds of variables is shown in Figure 3.2 and Table 3.2 lists common and/or natural ways of representing different kinds of (data) variables in R.

In experimental data we also distinguish the **dependent variable(s)** from the **independent variables**. The dependent variables are the variables that we do not control or manipulate in the experiment, but the ones that we are curious to record (e.g., whether a patient recovered from an illness within a week). Dependent variables are also called

---

<sup>3</sup>This sense of “data variable” is not to be confused with the notion of a “random variable”, a concept we will introduce later in Section 7.4. The term “data variable” is not commonly used; the common term is merely “variable”.

### 3. Data, variables & experimental designs

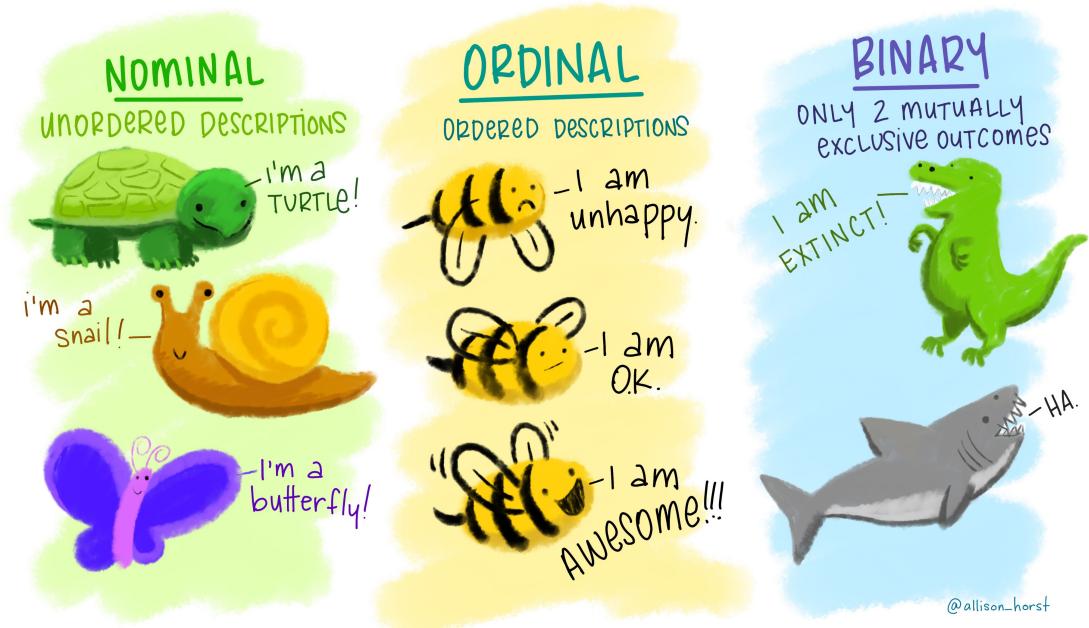


Figure 3.2.: Examples of different kinds of (data) variables.

**to-be-explained variables.** The independent variables are the variables in the experiment that we manipulate (e.g., which drug to administer), usually with the intention of seeing a particular effect on the dependent variables. Independent variables are also called **explanatory variables**.

### 3.3. Basics of experimental design

The most basic template for an experiment, inspired clearly by the natural sciences, is to just measure a quantity of interest (the dependent variable), without taking into account any kind of variation in any kind of independent variables. For instance, we measure the time it takes for an object, with specific shape and weight, to hit the ground when dropped from exactly 2 meters height. To filter out **measurement noise** we do not just record one observation, but take a fair amount. We use the observed times, for instance, to test a theory about acceleration and gravity. Data from such a simple measurement experiment would be just a single vector of numbers.

A more elaborate kind of experiment would allow for at least one independent variable. Another archetypical example of an empirical experiment would be a medical study, e.g., one in which we are interested in the effect of a particular drug on the blood pressure of patients. We would then randomly allocate each participant to one of two groups. One group, the **treatment group**, receives the drug in question, the other group, **the**

**control group**, receives a placebo (and nobody, not even the experimenter, knows who receives what). After a pre-defined exposure to either drug or placebo, blood pressure (for simplicity just systolic blood pressure) is measured. The interesting question is whether there is a difference between the measurements across groups. This is a simple example of a **one-factor design**. The factor in question is which group any particular measurement belongs to. Data from such an experiment could look like this:

```
tribble(
  ~subj_id,      ~group,      ~systolic,
  1,             "treatment", 118,
  2,             "control",   132,
  3,             "control",   116,
  4,             "treatment", 127,
  5,             "treatment", 122
)

## # A tibble: 5 x 3
##   subj_id group     systolic
##       <dbl> <chr>     <dbl>
## 1       1 treatment    118
## 2       2 control     132
## 3       3 control     116
## 4       4 treatment    127
## 5       5 treatment    122
```

For the purposes of this course, which is not a course on experimental design, just a few key concepts of experimental design are important to be aware of. We will go through some of these issues in the following.

### 3.3.1. What to analyze? – Dependent variables

To begin with, it is important to realize that there is quite some variation in what counts as a dependent variable. Not only can there be more than one dependent variable, each dependent variable can also be of quite a different type (nominal, ordinal, metric, ...), as discussed in the previous section. Moreover, we need to carefully distinguish between the actual measurement/observation and the dependent variable itself. The dependent variable is (usually) what we plot, analyze and discuss, but very often we measure much more or something else. The dependent variable (of analysis) could well just be one part of the measurement. For example, a standard measure of blood pressure has a number for systolic and another for diastolic pressure. Focussing on just one of these numbers is a (hopefully: theoretically motivated; possibly: arbitrary; in the worst case: result-oriented) decision of the analyst. More interesting examples of such **data preprocessing** arise frequently in the cognitive sciences, for example:

### 3. Data, variables & experimental designs

- **eye-tracking:** the measured data are triples consisting of a time-point and two spatial coordinates, but what might be analyzed is just the relative proportion of looks at a particular spatial region of interest (some object on the screen) in a particular temporal region of interest (up to 200 ms after the image appeared);
- **EEG:** individual measurements obtained by EEG are very noisy, so that the dependent measure in many analyses is an aggregation over the mean voltage recorded by selected electrodes, where averages are taken for a particular subject over many trials of the same condition (repeated measures) that this subject has seen;

But we do not need to go fancy in our experimental methods, to see how issues of data processing affect data analysis at its earliest stages, namely by selecting the dependent variable (that which is to be analyzed). Just take the distinction between **closed questions** and **open questions** in text-based surveys. In closed questions, participants select an answer from a finite (usually) small number of choices. In open questions, however, they can write text freely, or they can draw, sing, pronounce, gesture etc. Open response formats are great and naturalistic, but they, too, often require the analyst to carve out a particular aspect of the (rich, natural) observed reality to enter analysis.

#### 3.3.2. Conditions, trials, items

A **factorial design** is an experiment with at least two independent variables all of which are (ordered or unordered) factors.<sup>4</sup> Factorial designs are often described in terms of short abbreviations. For example, an experiment described as a “ $2 \times 3$  factorial design” would have two factors of interest, the first of which has two levels, the second of which has three levels (such as a distinction between control and treatment group, and an orthogonal distinction of gender in categories ‘male’, ‘female’ and ‘non-binary’). Many psychological studies are factorial designs. Whole batteries of analyses techniques have been developed specifically tuned to these kinds of experiments.

For a  $2 \times 2 \times 3$  factorial design, there are  $2 * 2 * 3 = 12$  different **experimental conditions** (also sometimes called **design cells**). An important distinction in experimental design is whether all participants contribute data to all of the experimental conditions, or whether each only contributes to a part of it. If participants only contribute data to a part of all experimental conditions, this is called a **between-subjects design**. If all participants contribute data to all experimental conditions, we speak of a **within-subjects design**. Clearly, sometimes the nature of a design factor determines whether the study can be within-subjects. For example, switching gender for the purpose of a medical study on blood pressure drugs is perhaps a tad much to ask of a participant (though certainly a very enlightening experience). If there is room for the experimenter’s choice of study type, it pays to be aware of some of the clear advantages and draw-backs of either method, as listed in Table 3.3.

---

<sup>4</sup>The archetypical medical experiment discussed above is a *one-factor design*. In contrast, the term ‘factorial design’ is usually used to refer to what is also often called a **full factorial design**. These are designs with at least two independent variables.

### 3.3. Basics of experimental design

Table 3.3.: Comparison of pro's and cons's of between- and within-subjects designs.

| between-subjects                         | within-subjects                                 |
|--|---|
| no confound between conditions           | possible cross-contamination between conditions |
| more participants needed                 | fewer participants needed                       |
| less associated information for analysis | more associated data for analysis               |

No matter whether we are dealing with a between- or within-subjects design, another important question is whether each participant gives us only one or more than one observation per cell. If participants contribute more than one observation to a design cell, we speak of a *repeated measures* design. Such designs are useful as they help separate the signal from the noise (recall the initial example of time measurement from physics). They are also economic because getting several observations worth of relevant data from a single participant for each condition means that we have to get fewer people to do the experiment (normally).

However, exposing a participant to the same experimental condition repeatedly can be detrimental to an experiment's purpose. Participants might recognize the repetition and develop quick coping strategies to deal with the boredom, for example. For this reason repeated measures designs usually include different kinds of trials:

- **critical trials** belong to, roughly put, the actual experiment, e.g., one of the experiment's design cells;
- **filler trials** are packaged around the critical trials to prevent blatant repetition, predictability or recognition of the experiment's purpose
- **control trials** are trials whose data is used not for statistics inference but for checking the quality of the data (e.g., attention checks or tests of whether a participant understood the task correctly)

When participants are exposed to several different kinds of trials and even several instances of the same experimental condition, it is also often important to introduce some variability between the instances of the same types of trials. Often psychological experiments therefore use different **items**, i.e., different (theoretically exchangeable) instantiations of the same (theoretically important) pattern. For example, if a careful psycholinguist designs a study on the processing of garden-path sentences, she will include not just one example ("The horse raced past the barn fell") but several (e.g., "Since Jones frequently jogs a mile is a short distance to her"). Item-variability is important also for statistical analyses, as we will see when we talk about hierarchical modeling in Section 19.

In longer experiments, especially within-subjects repeated measures designs in which participants encounter a lot of different items for each experimental condition, clever regimes of **randomization** are important to minimize the possible effect of carry-over artifacts, for example. A frequent method is **pseudo-randomization** where the trial

### *3. Data, variables & experimental designs*

sequence is not completely arbitrary but arbitrary within certain constraints, such as a particular **block design**, where each block presents an identical number of trials of each type, but each block shuffles the sequence of its types completely at random.

The complete opposite of a within-participants repeated measures design is a so-called **single-shot experiment** in which any participant gives exactly one data point for one experimental condition.

#### **3.3.3. Sample size**

A very important question for experimental design is that of the **sample size**: how many data points do we need (per experimental condition)? We will come back to this issue only much later in this course, when we talk about statistical inference. This is because the decision of how many, say, participants to invite for a study, should ideally be influenced, not by the available time and money, but also by statistical considerations of the kind: how many data points do I need in order to obtain a reasonable level of confidence in the resulting statistical inferences I care about?

# 4. Data Wrangling

The information relevant for our analysis goals is not always directly accessible. Sometimes we must first uncover it effortfully from an inconvenient representation. Also, sometimes data must be cleaned (ideally: by *a priori* specified criteria) by removing data points that are deemed of insufficient quality for a particular goal. All of this, and more, is the domain of **data wrangling**: preprocessing, cleaning, reshaping, renaming etc. Section 4.1 describes how to read data from and write data to files. Section 4.2 introduces the concept of **tidy data**. We then look at a few common tricks of data manipulation in Section 4.3. We will learn about grouping operations in Section 4.4. Finally, we look at a concrete application in Section 4.5.

The learning goals for this chapter are:

- be able to read from and write data to files
- understand notion of *tidy data*
- be able to solve common problems of data preprocessing

## 4.1. Data in, data out

The `readr` package handles the reading and writing of data stored in text files.<sup>1</sup> Here is a cheat sheet on the topic: data I/O cheat sheet. In this course will mostly deal with data stored in CSV files.

Reading a data set from a CSV file works with the `read_csv` function:

```
fresh_raw_data <- read_csv("PATH/FILENAME_RAW_DATA.csv")
```

Writing to a csv file can be done with the `write_csv` function:

```
write_csv(processed_data, "PATH/FILENAME_PROCESSED_DATA.csv")
```

If you want to use a different delimiter (between cells) than a comma, you can use `read_delim` and `write_delim` for example, which take an additional argument `delim` to be set to the delimiter in question.

---

<sup>1</sup>Other packages help with reading data from and writing data to other file types, such as excel sheets. Look at the data I/O cheat sheet for more information.

## 4. Data Wrangling

```
# reading data from a file where cells are (unconventionally) delimited by string
data_from_weird_file <- read_delim("WEIRD_DATA_FILE.TXT", delim = "|")
```

### 4.2. Tidy data

The same data can be represented in multiple ways. There is even room for variance in the class of rectangular representations of data. Some manners of representations are more useful for certain purposes than for others. For data analysis (plotting, statistical analyses) we prefer to represent our data as (rectangular) **tidy data**.

#### 4.2.1. Running example

Consider the example of student grades for two exams in a course. A compact way of representing the data for visual digestion is the following representation:

```
exam_results_visual <- tribble(
  ~exam,          ~"Rozz",    ~"Andrew",    ~"Siouxsie",
  "midterm",     "1.3",      "2.0",        "1.7",
  "final" ,      "2.3",      "1.7",        "1.0"
)
exam_results_visual

## # A tibble: 2 x 4
##   exam    Rozz  Andrew Siouxsie
##   <chr>  <chr> <chr>  <chr>
## 1 midterm 1.3   2.0    1.7
## 2 final   2.3   1.7    1.0
```

This is how such data would frequently be represented, e.g., in tables in a journal. Indeed, Rmarkdown helps us present this data in an appetizing manner, e.g., in Table 4.1, which is produced by the code below:

```
knitr::kable(
  exam_results_visual,
  caption = "Fictitious exam results of fictitious students.",
  booktabs = TRUE
)
```

Though highly perspicuous, this representation of the data is not tidy, in the special technical sense we endorse here. A tidy representation of the course results could be this:

Table 4.1.: Fictitious exam results of fictitious students.

| exam    | Rozz | Andrew | Siouxsie |
|---------|------|--------|----------|
| midterm | 1.3  | 2.0    | 1.7      |
| final   | 2.3  | 1.7    | 1.0      |

```

exam_results_tidy <- tribble(
  ~student,      ~exam,      ~grade,
  "Rozz",        "midterm",   1.3,
  "Andrew",      "midterm",   2.0,
  "Siouxsie",    "midterm",   1.7,
  "Rozz",        "final",     2.3,
  "Andrew",      "final",     1.7,
  "Siouxsie",    "final",     1.0
)
exam_results_tidy

## # A tibble: 6 x 3
##   student  exam  grade
##   <chr>    <chr> <dbl>
## 1 Rozz     midterm 1.3
## 2 Andrew    midterm 2
## 3 Siouxsie midterm 1.7
## 4 Rozz     final   2.3
## 5 Andrew    final   1.7
## 6 Siouxsie final   1

```

### 4.2.2. Definition of *tidy* data

Following Wickham (2014), a tidy representation of (rectangular) data is defined as one where:

1. each variable forms a column,
2. each observation forms a row, and
3. each type of observational unit forms a table.

Any data set that is not tidy is **messy data**. Messy data that satisfies the first two constraints, but not the third will be called **almost tidy data** in this course. We will work, wherever possible, with data that is at least almost tidy. Figure 4.1 shows a graphical representation of the concept of tidy data.

## 4. Data Wrangling

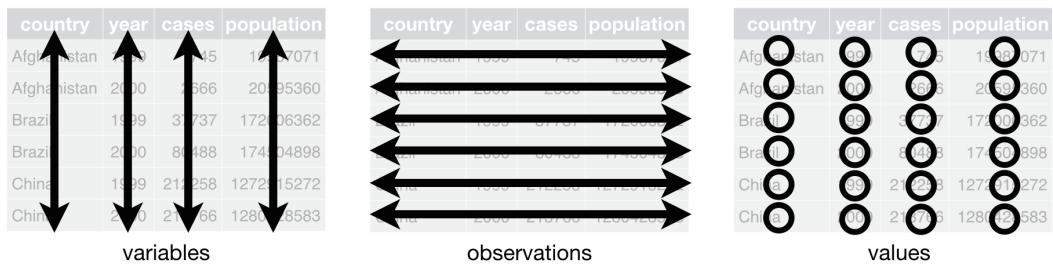


Figure 4.1.: Organization of tidy data (taken from @wickham2016).

### 4.2.3. Excursion: non-redundant data

The final condition in the definition of tidy data is not particularly important for us here (since we will make do with ‘almost tidy data’), but to understand it nonetheless consider the following data set:

```
exam_results_overloaded <- tribble(
  ~student,      ~stu_number,      ~exam,        ~grade,
  "Rozz",        "666",           "midterm",    1.3,
  "Andrew",      "1969",          "midterm",    2.0,
  "Siouxsie",    "3.14",          "midterm",    1.7,
  "Rozz",        "666",           "final",      2.3,
  "Andrew",      "1969",          "final",      1.7,
  "Siouxsie",    "3.14",          "final",      1.0
)
exam_results_overloaded

## # A tibble: 6 x 4
##   student  stu_number exam   grade
##   <chr>     <chr>     <chr>   <dbl>
## 1 Rozz      666       midterm  1.3
## 2 Andrew    1969      midterm  2
## 3 Siouxsie 3.14      midterm  1.7
## 4 Rozz      666       final   2.3
## 5 Andrew    1969      final   1.7
## 6 Siouxsie 3.14      final   1
```

This table is not tidy in an intuitive sense because it includes redundancy. Why list the student numbers twice, once with each observation of exam score? The table is not tidy in the technical sense that not every observational unit forms a table, i.e., the

### 4.3. Data manipulation: the basics

observation of student numbers and the observation of exam scores should be stored independently in different tables, like so:

```
# same as before
exam_results_tidy <- tribble(
  ~student,      ~exam,        ~grade,
  "Rozz",       "midterm",   1.3,
  "Andrew",     "midterm",   2.0,
  "Siouxsie",   "midterm",   1.7,
  "Rozz",       "final",     2.3,
  "Andrew",     "final",     1.7,
  "Siouxsie",   "final",     1.0
)
# additional table with student numbers
student_numbers <- tribble(
  ~student,      ~student_number,
  "Rozz",        "666",
  "Andrew",      "1969",
  "Siouxsie",    "3.14"
)
```

Notice that, although the information is distributed over two tibbles, it is linked by the common column `student`. If we really need to bring all of the information together, the tidyverse has a quick and elegant solution:

```
full_join(exam_results_tidy, student_numbers, by = "student")

## # A tibble: 6 x 4
##   student  exam    grade student_number
##   <chr>    <chr>   <dbl>   <chr>
## 1 Rozz     midterm  1.3    666
## 2 Andrew   midterm  2      1969
## 3 Siouxsie midterm  1.7    3.14
## 4 Rozz     final   2.3    666
## 5 Andrew   final   1.7    1969
## 6 Siouxsie final   1      3.14
```

## 4.3. Data manipulation: the basics

### 4.3.1. Pivoting

The tidyverse strongly encourages the use of tidy data, or at least almost tidy data. If your data is (almost) tidy, you can be reasonably sure that you can plot and analyze the

## 4. Data Wrangling

data without additional wrangling. If your data is not (almost) tidy because it is too wide or too long (see below), what is required is a joyful round of pivoting. There are two directions of pivoting: making data longer, and making data wider.

### 4.3.1.1. Making too wide data longer with `pivot_longer`

Consider the previous example of messy data again:

```
exam_results_visual <- tribble(
  ~exam,      ~"Rozz",    ~"Andrew",    ~"Siouxsie",
  "midterm",   "1.3",     "2.0",       "1.7",
  "final" ,   "2.3",     "1.7",       "1.0"
)
exam_results_visual

## # A tibble: 2 x 4
##   exam    Rozz  Andrew Siouxsie
##   <chr>  <chr> <chr>  <chr>
## 1 midterm 1.3   2.0    1.7
## 2 final   2.3   1.7    1.0
```

This data is “too wide”. We can make it longer with the function `pivot_longer` from the `tidyverse` package. Check out the example below before we plunge into a description of `pivot_longer`.

```
exam_results_visual %>%
  pivot_longer(
    # pivot every column except the first
    cols = - 1,
    # name of new column which contains the
    # names of the columns to be "gathered"
    names_to = "student",
    # name of new column which contains the values
    # of the cells which now form a new column
    values_to = "grade"
  ) %>%
  # optional reordering of columns (to make
  # the output exactly like `exam_results_tidy`)
  select(student, exam, grade)

## # A tibble: 6 x 3
##   student  exam    grade
##   <chr>    <chr>  <dbl>
```

```
## <chr> <chr> <chr>
## 1 Rozz    midterm 1.3
## 2 Andrew   midterm 2.0
## 3 Siouxsie midterm 1.7
## 4 Rozz    final   2.3
## 5 Andrew   final   1.7
## 6 Siouxsie final   1.0
```

What `pivot_longer` does, in general, is take a bunch of columns and gather the values of all cells in these columns into a single, new column, the so-called *value column*, i.e., the column with the values of the cells to be gathered. If `pivot_longer` stopped here, we would lose information about which cell values belonged to which original column. Therefore, `pivot_longer` also creates a second new column, the so-called *name column*, i.e., the column with the names of the original columns that we gathered together. Consequently, in order to do its job, `pivot_longer` minimally needs three pieces of information:<sup>2</sup>

1. which columns to spin around (function argument `cols`)
2. the name of the to-be-created new value column (function argument `values_to`)
3. the name of the to-be-created new name column (function argument `names_to`)

For different ways of selecting columns to pivot around, see Section 4.3.3 below.

#### 4.3.1.2. Making too long data wider with `pivot_wider`

Consider the following example of data which is untidy because it is too long:

```
mixed_results_too_long <-
  tibble(student = rep(c('Rozz', 'Andrew', 'Siouxsie'), times = 2),
         what = rep(c('grade', 'participation'), each = 3),
         howmuch = c(2.7, 2.0, 1.0, 75, 93, 33))
mixed_results_too_long

## # A tibble: 6 x 3
##   student  what      howmuch
##   <chr>    <chr>     <dbl>
## 1 Rozz     grade      2.7
## 2 Andrew   grade      2
## 3 Siouxsie grade      1
## 4 Rozz     participation 75
```

---

<sup>2</sup>There are alternative possibilities for specifying names of the value and name column, which allow for more dynamic construction of strings. We will not cover all of these details here, but we will use some of these alternative specifications in subsequent examples.

#### 4. Data Wrangling

```
## 5 Andrew participation 93
## 6 Siouxsie participation 33
```

This data is untidy because it lumps two types of different measurements (a course grade, and the percentage of participation) in a single column. These are different variables, and so should be represented in different columns.

To fix a data representation that is too long, we can make it wider with the help of the `pivot_wider` function from the `tidyverse` package. We look at an example before looking at the general behavior of the `pivot_wider` function.

```
mixed_results_too_long %>%
  pivot_wider(
    # column containing the names of the new columns
    names_from = what,
    # column containing the values of the new columns
    values_from = howmuch
  )

## # A tibble: 3 x 3
##   student  grade participation
##   <chr>    <dbl>        <dbl>
## 1 Rozz      2.7          75
## 2 Andrew     2            93
## 3 Siouxsie  1            33
```

In general, `pivot_wider` picks out two columns, one column of values to distribute into new to-be-created columns, and one vector of names or groups which contains the information about the, well, names of the to-be-created new columns. There are more refined options for `pivot_wider` some of which we will encounter in the context of concrete cases of application.

##### 4.3.2. Subsetting row & columns

If a data set contains too much information for your current purposes, you can discard irrelevant (or unhelpful) rows and columns. The function `filter` takes a Boolean expression and returns only those rows of which the Boolean expression is true:

```
exam_results_tidy %>%
  # keep only entries with grades better than 1.7
  filter(grade <= 1.7)
```

```
## # A tibble: 4 x 3
##   student    exam    grade
##   <chr>     <chr>    <dbl>
## 1 Rozz      midterm    1.3
## 2 Siouxsie final     1.7
## 3 Andrew    final     1.7
## 4 Siouxsie final      1
```

To select rows by an index or a vector of indeces, use the `slice` function:

```
exam_results_tidy %>%
  # keep only entries from rows with an even index
  dplyr::slice(c(2,4,6)) # explicit call to avoid name clash (package 'greta')

## # A tibble: 3 x 3
##   student    exam    grade
##   <chr>     <chr>    <dbl>
## 1 Andrew    midterm    2
## 2 Rozz      final     2.3
## 3 Siouxsie final      1
```

The function `select` allows to pick out a subset of columns. Interestingly, it can also be used to reorder columns, because the order in which column names are specified matches the order in the returned tibble.

```
exam_results_tidy %>%
  # select columns `grade` and `name`
  select(grade, exam)

## # A tibble: 6 x 2
##   grade exam
##   <dbl> <chr>
## 1 1.3   midterm
## 2 2     midterm
## 3 1.7   midterm
## 4 2.3   final
## 5 1.7   final
## 6 1     final
```

### 4.3.3. Tidy selection of column names

To select the colums in several functions within the tidyverse, such as `pivot_longer` or `select`, there are useful helper functions from the `tidyselect` package. Here are some

#### 4. Data Wrangling

examples:<sup>3</sup>

```
# bogus code for illustration of possibilities!
SOME_DATA %>%
  select( ... # could be one of the following
         # all columns indexed 2, 3, ..., 10
         2:10
         # all columns except the one called "COLNAME"
         - COLNAME
         # all columns with names starting with "STRING"
         ... starts_with("STRING")
         # all columns with names ending with "STRING"
         ... ends_with("STRING")
         # all columns with names containing "STRING"
         ... contains("STRING")
         # all columns with names of the form "Col_i" with i = 1, ..., 10
         ... num_range("Col_", 1:10)
  )
```

##### 4.3.4. Adding, changing and renaming columns

To add a new column, or to change an existing one use function `mutate`, like so:

```
exam_results_tidy %>%
  mutate(
    # add a new column called 'passed' depending on grade
    # [NB: severe passing conditions in this class!!]
    passed = grade <= 1.7,
    # change an existing column; here: change
    # character column 'exam' to ordered factor
    exam = factor(exam, ordered = T)
  )

## # A tibble: 6 x 4
##   student  exam    grade passed
##   <chr>    <ord>   <dbl> <lgl>
## 1 Rozz     midterm  1.3 TRUE
## 2 Andrew   midterm  2    FALSE
## 3 Siouxsie midterm  1.7 TRUE
## 4 Rozz     final    2.3 FALSE
## 5 Andrew   final    1.7 TRUE
## 6 Siouxsie final    1    TRUE
```

---

<sup>3</sup>The helpers from the `tidyselect` package also accept regular expressions.

### 4.3. Data manipulation: the basics

If you want to rename a column, function `rename` is what you want:

```
exam_results_tidy %>%
  # rename existing column "student" to new name "participant"
  # [NB: rename takes the new name first]
  rename(participant = student)

## # A tibble: 6 x 3
##   participant exam     grade
##   <chr>        <chr>    <dbl>
## 1 Rozz         midterm   1.3
## 2 Andrew       midterm   2
## 3 Siouxsie    midterm   1.7
## 4 Rozz         final     2.3
## 5 Andrew       final     1.7
## 6 Siouxsie    final     1
```

#### 4.3.5. Splitting and uniting columns

Here is data from course homework:

```
homework_results_untidy <-
  tribble(
    ~student,      ~results,
    "Rozz",        "1.0,2.3,3.0",
    "Andrew",       "2.3,2.7,1.3",
    "Siouxsie",    "1.7,4.0,1.0"
  )
```

This is not a useful representation format. Results of three homework sets are mashed together in a single column. Each value is separated by a comma, but it's all stored as a character vector.

To disentangle information in a single column, use the `separate` function:

```
homework_results_untidy %>%
  separate(
    # which column to split up
    col = results,
    # names of the new column to store results
    into = str_c("HW_", 1:3),
    # separate by which character / reg-exp
    sep = ",",
```

#### 4. Data Wrangling

```
# automatically (smart-)convert the type of the new cols
convert = T
)

## # A tibble: 3 x 4
##   student    HW_1    HW_2    HW_3
##   <chr>     <dbl>   <dbl>   <dbl>
## 1 Rozz       1      2.3     3
## 2 Andrew     2.3    2.7    1.3
## 3 Siouxsie  1.7    4      1
```

If you have reason to perform the reverse operation, i.e., join together several columns, use the `unite` function.

#### 4.3.6. Sorting a data set

If you want to indicate a fixed order of the reoccurring elements in a (character) vector, e.g., for plotting in a particular order, you should make this column an ordered factor. But if you want to order a data set along a column, e.g., for inspection or printing as a table, then you can do that using the `arrange` function. You can specify several columns to sort alpha-numerically in ascending order, and also indicate a descending order using the `desc` function:

```
exam_results_tidy %>%
  arrange(desc(student), grade)
```

```
## # A tibble: 6 x 3
##   student   exam   grade
##   <chr>     <chr>   <dbl>
## 1 Siouxsie final     1
## 2 Siouxsie midterm  1.7
## 3 Rozz      midterm  1.3
## 4 Rozz      final    2.3
## 5 Andrew    final    1.7
## 6 Andrew    midterm  2
```

#### 4.3.7. Combining tibbles

There are frequently occasions on which data from two separate variables needs to be combined. The simplest case is where two entirely disjoint data sets merely need to be glued together, either horizontally (binding columns together with function `cbind`) or vertically (binding rows together with function `rbind`).

```

new_exam_results_tidy <- tribble(
  ~student,      ~exam,      ~grade,
  "Rozz",        "bonus",    1.7,
  "Andrew",      "bonus",    2.3,
  "Siouxsie",    "bonus",    1.0
)
rbind(
  exam_results_tidy,
  new_exam_results_tidy
)

## # A tibble: 9 x 3
##   student   exam   grade
##   <chr>     <chr>   <dbl>
## 1 Rozz     midterm  1.3
## 2 Andrew    midterm  2
## 3 Siouxsie midterm  1.7
## 4 Rozz     final    2.3
## 5 Andrew    final    1.7
## 6 Siouxsie final    1
## 7 Rozz     bonus    1.7
## 8 Andrew    bonus    2.3
## 9 Siouxsie bonus    1

```

If two data sets have information in common, and the combination should respect that commonality, the `join` family of functions is of great help. Consider the case of distributed information again that we looked at to understand the third constraint of the concept of “tidy data”. There are two tibbles, both of which contain information about the same students. They share the column `student` (this does not necessarily have to be in the same order!) and we might want to join the information from both sources into a single (messy but almost tidy) representation, using `full_join`. We have seen an example already, which is repeated here:

```

# same as before
exam_results_tidy <- tribble(
  ~student,      ~exam,      ~grade,
  "Rozz",        "midterm",  1.3,
  "Andrew",      "midterm",  2.0,
  "Siouxsie",    "midterm",  1.7,
  "Rozz",        "final",    2.3,
  "Andrew",      "final",    1.7,
  "Siouxsie",    "final",    1.0
)

```

#### 4. Data Wrangling

```
# additional table with student numbers
student_numbers <- tribble(
  ~student,      ~student_number,
  "Rozz",        "666",
  "Andrew",      "1969",
  "Siouxsie",    "3.14"
)
full_join(exam_results_tidy, student_numbers, by = "student")

## # A tibble: 6 x 4
##   student   exam     grade student_number
##   <chr>     <chr>    <dbl>   <chr>
## 1 Rozz     midterm   1.3    666
## 2 Andrew   midterm   2      1969
## 3 Siouxsie midterm   1.7    3.14
## 4 Rozz     final     2.3    666
## 5 Andrew   final     1.7    1969
## 6 Siouxsie final     1      3.14
```

If two data sets are to be joined by a column that is not exactly shared by both sets (one contains entries in this columns that the other doesn't) then a `full_join` will retain all information from both. If that is not what you want, check out alternative functions like `right_join`, `semi_join` etc. using the data wrangling cheat sheet.

### 4.4. Grouped operations

A frequently occurring problem in data analysis is to obtain a summary statistic (see Section 5) for different subsets of data. For example, we might want to calculate the average score for each student in our class. We could do that by filtering like so (notice that `pull` gives you the column vector specified):

```
# extracting mean grade for Rozz
mean_grade_Rozz <- exam_results_tidy %>%
  filter(student == "Rozz") %>% pull(grade) %>% mean
mean_grade_Rozz

## [1] 1.8
```

But then we need to do that two more times, so we shouldn't copy-paste code, so we write a function and use `mutate` to add a mean for each student:

```

get_mean_for_student = function(student_name) {
  exam_results_tidy %>%
    filter(student == student_name) %>% pull(grade) %>% mean
}

map_dbl(
  exam_results_tidy %>% pull(student) %>% unique,
  get_mean_for_student
)

## [1] 1.80 1.85 1.35

```

Also not quite satisfactory, clumsy and error-prone. Enter, grouping in the tidyverse. If we want to apply a particular operation to all combinations of levels of different variables (no matter whether they are encoded as factors or not when we group), we can do this with the function `group_by`, followed by either a call to `mutate` or `summarise`. Check this example:

```

exam_results_tidy %>%
  group_by(student) %>%
  summarise(
    student_mean = mean(grade)
  )

## # A tibble: 3 x 2
##   student  student_mean
##   <chr>        <dbl>
## 1 Andrew      1.85
## 2 Rozz        1.8
## 3 Siouxsie   1.35

```

The function `summarise` returns a single row for each combination of levels of grouping variables. If we use the function `mutate` instead, the summary statistic is added (repeatedly) in each of the original rows:

```

exam_results_tidy %>%
  group_by(student) %>%
  mutate(
    student_mean = mean(grade)
  )

```

#### 4. Data Wrangling

```
## # A tibble: 6 x 4
## # Groups: student [3]
##   student exam   grade student_mean
##   <chr>    <chr>   <dbl>      <dbl>
## 1 Rozz     midterm  1.3       1.8
## 2 Andrew   midterm  2         1.85
## 3 Siouxsie midterm  1.7       1.35
## 4 Rozz     final    2.3       1.8
## 5 Andrew   final    1.7       1.85
## 6 Siouxsie final    1         1.35
```

The latter can sometimes be handy, for example when overlaying a plot of the data with grouped means, for instance.

It may be important to remember that after a call of `group_by`, the resulting tibbles retains the grouping information for *all* subsequent operations. To remove grouping information, use the function `ungroup`.

### 4.5. Case study: the King of France

Let's go through one case study of data preprocessing. We look at the example introduced and fully worked out in Appendix D.4. (Please read Section D.4.1 to find out more about where this data set is coming from.)

The raw data set is stored in the GitHub repository that also hosts this web-book. It can be loaded using:

```
data_KoF_raw <- read_csv(url('https://raw.githubusercontent.com/michael-franke/intro...'))
```

We can then get a glimpse at the data using:

```
glimpse(data_KoF_raw)
```

```
## Observations: 2,813
## Variables: 16
## $ submission_id  <dbl> 192, 192, 192, 192, 192, 192, 192, 192, 192, 19...
## $ RT              <dbl> 8110, 35557, 3647, 16037, 11816, 6024, 4986, 13019, ...
## $ age              <dbl> 57, 57, 57, 57, 57, 57, 57, 57, 57, 57, 57, 57, 57, 57, ...
## $ comments          <chr> NA, ...
## $ item_version     <chr> "none", "none", "none", "none", "none", "none", ...
## $ correct_answer    <lgl> FALSE, TRUE, FALSE, TRUE, TRUE, TRUE, FALSE, FALSE, ...
## $ education         <chr> "Graduated College", "Graduated College", "Graduated...
```

#### 4.5. Case study: the King of France

```
## $ gender          <chr> "female", "female", "female", "female", "female", "f...
## $ languages       <chr> "English", "English", "English", "English", "English...
## $ question        <chr> "World War II was a global war that lasted from 1914...
## $ response         <lgl> FALSE, TRUE, FALSE, TRUE, TRUE, FALSE, FALSE, ...
## $ timeSpent       <dbl> 39.48995, 39.48995, 39.48995, 39.48995, 39.48995, 39...
## $ trial_name      <chr> "practice_trials", "practice_trials", "practice_tria...
## $ trial_number    <dbl> 1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12...
## $ trial_type      <chr> "practice", "practice", "practice", "practice", "pra...
## $ vignette         <chr> "undefined", "undefined", "undefined", "undefined", ...
```

The variables in this data set are:

- `submission_id`: unique identifier for each participant
- `RT`: the reaction time for each decision
- `age`: the (self-reported) age of the participant
- `comments`: the (optional) comments each participant may have given
- `item_version`: the condition which the test sentence belongs to (only given for trials of type `main` and `special`)
- `correct_answer`: for trials of type `filler` and `special` what the true answer should have been
- `education`: the (self-reported) education level with options `Graduated College`, `Graduated High School`, `Higher Degree`
- `gender`: (self-reported) gender
- `languages`: (self-reported) native languages
- `question`: the sentence to be judged true or false
- `response`: the answer ("TRUE" or "FALSE") on each trial
- `trial_name`: whether the trial is a main or practice trials (levels `main_trials` and `practice_trials`)
- `trial_number`: consecutive numbering of each participant's trial
- `trial_type`: whether the trial was of the category `filler`, `main`, `practice` or `special`, where the latter encodes the "background checks"
- `vignette`: the current item's vignette number (applies only to trials of type `main` and `special`)

Let's have brief look at the comments (sometimes helpful, usually entertaining) and the self-reported native languages:

```
data_KoF_raw %>% pull(comments) %>% unique
```

```
## [1] NA
## [2] "I hope I was right most of the time!"
## [3] "My level of education is Some Highschool, not finished. So I couldn't input what was
## [4] "It was interesting, and made re-read questions to make sure they weren't tricks. I ha
## [5] "Worked well"
```

#### 4. Data Wrangling

```
## [6] "A surprisingly tricky study! Thoroughly enjoyed completing it, despite sev
## [7] "n/a"
## [8] "Thank you for the opportunity."
## [9] "this was challenging"
## [10] "I'm not good at learning history so i might of made couple of mistakes. I
## [11] "Interesting survey - thanks!"
## [12] "no"
## [13] "Regarding the practice question - I'm aware that Alexander Bell invented t
## [14] "Fun study!"
## [15] "Fun stuff"

data_KoF_raw %>% pull(languages) %>% unique

## [1] "English"           "english"          "English, Italian"
## [4] "English/ ASL"      "English and Polish" "Chinese"
## [7] "English, Mandarin" "Polish"            "Turkish"
## [10] NA                 "English, Sarcasm"   "English, Portuguese"
```

We might wish to exclude people who do not include “English” as one of their native languages in some studies. Here, we do not since we also have strong, more specific filters on comprehension (see below). Since we are not going to use this information later on, we might as well discard it now:

```
data_KoF_raw <- data_KoF_raw %>%
  select(-languages, - comments, -age, - RT, - education, - gender)
```

But even after pruning irrelevant columns, this data set is still not ideal. We need to preprocess it more thoroughly to make it more intuitively manageable. For example, the information in column `trial_name` does not give the trial’s name in an intuitive sense, but its type: whether it is a practice or a main trial. But this information, and more, is also represented in the column `trial_type`. The column `item_version` contains information about the experimental condition. To see this (mess) the code below prints the selected information from the main trials of only one participant in an order that makes it easier to see what is what.

```
data_KoF_raw %>%
  # ignore practice trials for the moment
  # focus on one participant only
  filter(trial_type != "practice", submission_id == 192) %>%
  select(trial_type, item_version, question) %>%
  arrange(desc(trial_type), item_version) %>%
  print(n = Inf)
```

#### 4.5. Case study: the King of France

```
## # A tibble: 24 x 3
##   trial_type item_version question
##   <chr>     <chr>      <chr>
## 1 special    none       The Pope is currently not married.
## 2 special    none       Germany has volcanoes.
## 3 special    none       France has a king.
## 4 special    none       Canada is a democracy.
## 5 special    none       Belgium has rainforests.
## 6 main       0          The volcanoes of Germany dominate the landscape.
## 7 main       1          Canada has an emperor, and he is fond of sushi.
## 8 main       10         Donald Trump, his favorite nature spot is not the Be-
## 9 main       6          The King of France isn't bald.
## 10 main      9          The Pope's wife, she did not invite Angela Merkel fo-
## 11 filler    none       The Solar System includes the planet Earth.
## 12 filler    none       Vatican City is the world's largest country by land ~
## 13 filler    none       Big Ben is a very large building in the middle of Pa-
## 14 filler    none       Harry Potter is a series of fantasy novels written b-
## 15 filler    none       Taj Mahal is a mausoleum on the bank of the river in-
## 16 filler    none       James Bond is a spanish dancer from Madrid.
## 17 filler    none       The Pacific Ocean is a large ocean between Japan and~
## 18 filler    none       Australia has a very large border with Brazil.
## 19 filler    none       Steve Jobs was an American inventor and co-founder o-
## 20 filler    none       Planet Earth is part of the galaxy 'Milky Way'.
## 21 filler    none       Germany shares borders with France, Belgium and Denm-
## 22 filler    none       Antarctica is a continent covered almost completely ~
## 23 filler    none       The Statue of Liberty is a colossal sculpture on Lib-
## 24 filler    none       English is the main language in Australia, Britain a-
```

We see that the information in `item_version` specifies the critical condition. To make this more intuitively manageable, we would like to have a column called `condition` and it should, ideally, also contain useful information for the cases where `trial_type` is not `main` or `special`. That is why we will therefore remove the column `trial_name` completely, and create an informative column `condition` in which we learn of every row whether it belongs to one of the 5 experimental conditions, and if not whether it is a filler or a “background check” (= special) trial.

```
data_KoF_processed <- data_KoF_raw %>%
  # drop redundant information in column `trial_name`
  select(-trial_name) %>%
  # discard practice trials
  filter(trial_type != "practice") %>%
  mutate(
    # add a 'condition' variable
```

## 4. Data Wrangling

```
condition = case_when(
  trial_type == "special" ~ "background check",
  trial_type == "main" ~ str_c("Condition ", item_version),
  TRUE ~ "filler"
) %>%
  # make the new 'condition' variable a factor
  factor(
    ordered = T,
    levels = c(
      str_c("Condition ", c(0, 1, 6, 9, 10)),
      "background check", "filler"
    )
)
# write_csv(data_KoF_processed, "data_sets/king-of-france_data_processed.csv")
```

### 4.5.1. Cleaning the data

We clean the data in two consecutive steps:

1. Remove all data from any participant who got more than 50% of the answer to filler material wrong.
2. Remove individual main trials if the corresponding “background check” question was answered wrongly.

#### 4.5.1.1. Cleaning by-participant

```
# look at error rates for filler sentences by subject
# mark every subject as an outlier when they
# have a proportion of correct responses of less than 0.5
subject_error_rate <- data_KoF_processed %>%
  filter(trial_type == "filler") %>%
  group_by(submission_id) %>%
  summarise(
    proportion_correct = mean(correct_answer == response),
    outlier_subject = proportion_correct < 0.5
  ) %>%
  arrange(proportion_correct)
```

Apply the cleaning step:

```
# add info about error rates and exclude outlier subject(s)
d_cleaned <-
  full_join(data_KoF_processed, subject_error_rate, by = "submission_id") %>%
  filter(outlier_subject == FALSE)
```

#### 4.5.1.2. Cleaning by-trial

```
# exclude every critical trial whose 'background' test question was answered wrongly
d_cleaned <-
  d_cleaned %>%
  # select only the 'background question' trials
  filter(trial_type == "special") %>%
  # is the background question answered correctly?
  mutate(
    background_correct = correct_answer == response
  ) %>%
  # select only the relevant columns
  select(submission_id, vignette, background_correct) %>%
  # right join lines to original data set
  right_join(d_cleaned, by = c("submission_id", "vignette")) %>%
  # remove all special trials, as well as main trials with incorrect background check
  filter(trial_type == "main" & background_correct == TRUE)

# write_csv(d_cleaned, "data_sets/king-of-france_data_cleaned.csv")
```



## 5. Summary statistics

A **summary statistics** is a single number which represents one aspect of a possibly much more complex chunk of data. This single number might, for example, indicate the maximum or minimum value of a vector of a billion observations. The large data set (1 billion observations) is reduced to a single number which represents one aspect of that data. Summary statistics are, as a general (but violable) rule, many-to-one (surjections). They compress complex information into a simpler, compressed representation.

Summary statistics are useful for understanding the data at hand, for communication about a data set, but also for subsequent statistical analyses. As we will see later on, many statistical tests look at a summary statistic  $x$ , which is a single value derived from data set  $D$ , and compare  $x$  to an expectation of what  $x$  should be like if the process that generated  $D$  really had a particular property. For the moment, however, we use summary statistics only to get comfortable with data: understanding it better, and gaining competence to manipulate it.

Chapter 5.1 first uses the Bio-Logic Jazz-Metal data set to look at a very intuitive class of summary statistics for categorical data, namely counts and proportions. Chapter 5.2 introduces summary statistics for simple, one dimensional vectors with numeric information. In doing so we will also learn about *nested tibbles*. Chapter 5.3 looks at measures of relation between two numerical vectors, namely *covariance* and *correlation*. These last two chapters use the avocado data set.

The learning goals for this chapter are:

- become able to compute counts and frequencies for categorical data
- understand and be able to compute summary statistics for one-dimensional metric data:
  - measures of central tendency
    - \* mean/mode/median/
  - measures of dispersion
    - \* variance, standard deviation, quantiles
  - non-parametric estimates of confidence
    - \* bootstrapped CI of mean
- understand and be able to compute for two-dimensional metric data:

## 5. Summary statistics

- covariance
- Bravais-Pearson correlation

### 5.1. Counts and proportions

Very familiar instances of summary statistics are counts and frequencies. While there is no conceptual difficulty in understanding these numerical measures, we have yet to see how to obtain counts for categorical data in R. The Bio-Logic Jazz-Metal data set provides nice material for doing so. If unfamiliar with the data and the experiment that generated it, please have a look at Appendix Chapter D.5.

#### 5.1.1. Loading and inspecting the data

We load the preprocessed data immediately (Appendix Chapter D.5 for details how this preprocessing was performed).

```
data_BLJM_processed <- read_csv(url('https://raw.githubusercontent.com/michael-fran...
```

The preprocessed data lists, for each participant (in column `submission_id`) the binary choice (in column `response`) given for a particular condition (in column `condition`).

```
head(data_BLJM_processed)

## # A tibble: 6 x 3
##   submission_id condition response
##       <dbl>     <chr>    <chr>
## 1         379     BM      Beach
## 2         379     LB      Logic
## 3         379     JM      Metal
## 4         378     JM      Metal
## 5         378     LB      Logic
## 6         378     BM      Beach
```

#### 5.1.2. Obtaining counts with `n`, `count` and `tally`

To obtain counts, the `dplyr` package offers functions `n`, `count` and `tally`, among others.<sup>1</sup> The function `n` does not take arguments and is useful for counting rows. It works inside of `summary` and `mutate` and is usually applied to grouped data sets. For example, we can get a count of how many observations the data in `data_BLJM_processed` contains

---

<sup>1</sup>Useful base R functions for obtaining counts are `table` and `prop.table`.

### 5.1. Counts and proportions

for each condition by first grouping by variable `condition` and then calling `n` (without arguments) inside of `summarise`:

```
data_BLJM_processed %>%
  group_by(condition) %>%
  summarise(nr_observation_per_condition = n()) %>%
  ungroup()

## # A tibble: 3 x 2
##   condition nr_observation_per_condition
##   <chr>                <int>
## 1 BM                  102
## 2 JM                  102
## 3 LB                  102
```

Notice that calling `n` without grouping just gives you the number of rows in the data set:

```
data_BLJM_processed %>% summarize(n_rows = n())

## # A tibble: 1 x 1
##   n_rows
##   <int>
## 1     306
```

This can also be obtained simply by (although in a different output format!):

```
nrow(data_BLJM_processed)

## [1] 306
```

Counting can be helpful also when getting acquainted with a data set, or when checking whether the data is complete. For example, we can verify that every participant in the experiment contributed three data points like so:

```
data_BLJM_processed %>%
  group_by(submission_id) %>%
  summarise(nr_data_points = n())
```

## 5. Summary statistics

```
## # A tibble: 102 x 2
##   submission_id nr_data_points
##       <dbl>           <int>
## 1 278                 3
## 2 279                 3
## 3 280                 3
## 4 281                 3
## 5 282                 3
## 6 283                 3
## 7 284                 3
## 8 285                 3
## 9 286                 3
## 10 287                3
## # ... with 92 more rows
```

The functions `tally` and `count` are essentially just convenience wrappers around `n`. While `tally` expects that the data is already grouped in the relevant way, `count` takes a column specification as an argument and does the grouping (and final ungrouping) implicitly.

For instance, the following code blocks produce the same output, one using `n`, the other using `count`, namely the total number of times a particular response has been given in a particular condition:

```
data_BLJM_processed %>%
  group_by(condition, response) %>%
  summarise(n = n())

## # A tibble: 6 x 3
## # Groups:   condition [3]
##   condition response     n
##   <chr>      <chr>    <int>
## 1 BM         Beach     44
## 2 BM         Mountains 58
## 3 JM         Jazz      64
## 4 JM         Metal     38
## 5 LB         Biology   58
## 6 LB         Logic     44

data_BLJM_processed %>%
  # function`count` is masked by another package, must call explicitly
  dplyr::count(condition, response)
```

```
## # A tibble: 6 x 3
##   condition response     n
##   <chr>      <chr>    <int>
## 1 BM         Beach     44
## 2 BM         Mountains 58
## 3 JM         Jazz      64
## 4 JM         Metal     38
## 5 LB         Biology   58
## 6 LB         Logic     44
```

So, these counts suggest that there is an overall preference for mountains over beaches, Jazz over Metal and Biology over Logic. Who would have known!?

These counts are overall numbers. They do not tell us anything about any potentially interesting relationship between preferences. So, let's have a closer look at the number of people who selected which music-subject pair. We collect these counts in variable `BLJM_associated_counts`. We first need to pivot the data, using `pivot_wider`, to make sure each participant's choices are associated with each other, and then take the counts of interest:

```
BLJM_associated_counts <- data_BLJM_processed %>%
  select(submission_id, condition, response) %>%
  pivot_wider(names_from = condition, values_from = response) %>%
  # drop the Beach-vs-Mountain condition
  select(-BM) %>%
  dplyr::count(JM,LB)
BLJM_associated_counts

## # A tibble: 4 x 3
##   JM     LB     n
##   <chr> <chr> <int>
## 1 Jazz   Biology 38
## 2 Jazz   Logic   26
## 3 Metal  Biology 20
## 4 Metal  Logic   18
```

We can also produce a table of proportions from this, simply by dividing the column called `n` by the total number of observations, i.e., by `sum(n)`. We can also flip the table around into a more convenient (though messy) representation:

```
BLJM_associated_counts %>%
  # look at relative frequency, not total counts
  mutate(n = n / sum(n)) %>%
  pivot_wider(names_from = LB, values_from = n)
```

## 5. Summary statistics

```
## # A tibble: 2 x 3
##   JM     Biology Logic
##   <chr>    <dbl> <dbl>
## 1 Jazz     0.373  0.255
## 2 Metal    0.196  0.176
```

Eye-balling this table of relative frequencies, we might indeed hypothesis that preference for musical style are not independent of preference for academic subject. The impression is corroborated by looking at the plot in Figure 5.1. More on this later!

### Proportion of choices of each music+subject pair



Figure 5.1.: Proportions of jointly choosing a musical style and an academic subfield in the Bio-Logic Jazz-Metal data set.

## 5.2. Central tendency and dispersion

This section will look at two types of summary statistics: measures of central tendency and measures of dispersion.

**Measures of central tendency** map a vector of observations onto a single number that represents, roughly put, “the center”. Since what counts as a “center” is ambiguous, there are several measures of central tendencies. Different measures of central tendencies

## 5.2. Central tendency and dispersion

can be more or less adequate for one purpose or another. The type of variable (nominal, ordinal or metric, for instance) will also influence the choice of measure. We will visit three prominent measures of central tendency here: (*arithmetic*) *mean*, *median* and *mode*.

**Measures of dispersion** indicate how much the observations are spread out around, let's say, "a center". We will visit three prominent measures of dispersion: the *variance*, the *standard deviation* and *quantiles*.

To illustrate these ideas, consider the case of a numeric vector of observations. Central tendency and dispersion together describe a (numeric) vector by giving indicative information about the point around which the observations spread, and how far away from that middle point they tend to lie. Fictitious examples of observation vectors with higher or lower central tendency and higher or lower dispersion are given in Figure 5.2.

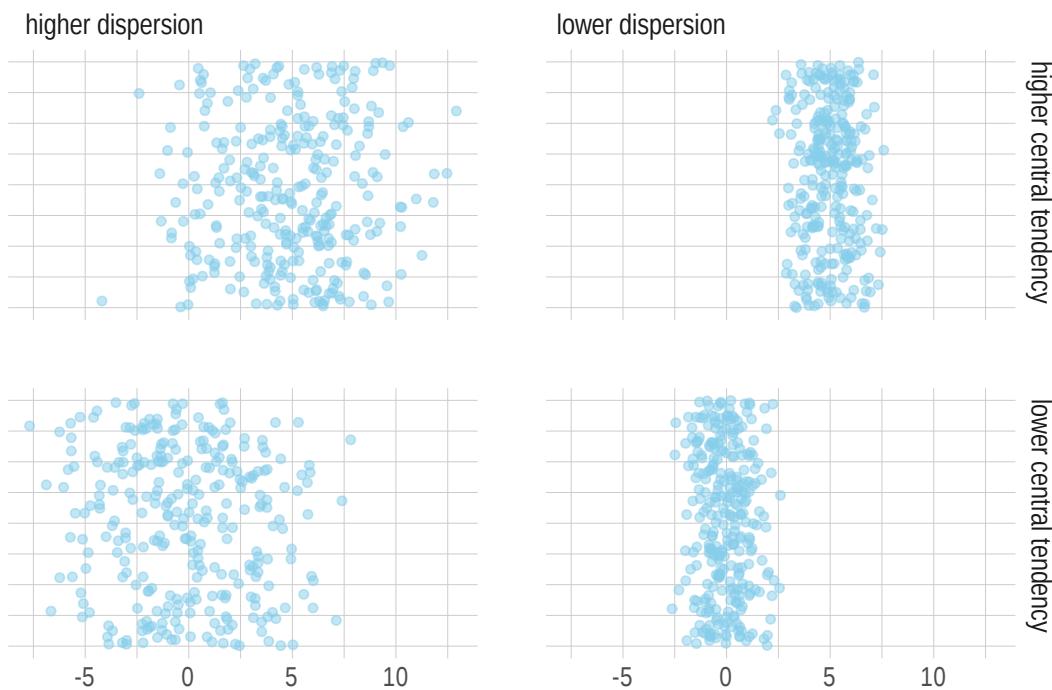


Figure 5.2.: Fictitious data points with higher/lower central tendencies and higher/lower dispersion. NB: points are 'jittered' along the vertical dimension for better visibility; only the horizontal dimension is relevant here.

## 5. Summary statistics

### 5.2.1. The data for the remainder of the chapter

In the remainder of this chapter we will use the avocado data set, a very simple and theory-free example in which we can explore two metric variables: the average price at which avocados were sold during specific intervals of time and the total amount of avocados sold. Please check Appendix Chapter D.6 for more information on this data set.

We load the data into a variable named `avocado_data` but also immediately rename some of the columns to have more convenient handles (see also Appendix Chapter D.6):

```
avocado_data <- read_csv(url('https://raw.githubusercontent.com/michael-franke/intro-to-data-science/master/datasets/avocado.csv')) %>%  
  # remove currently irrelevant columns  
  select(-X1, -contains("Bags"), -year, -region) %>%  
  # rename variables of interest for convenience  
  rename(  
    total_volume_sold = `Total Volume`,  
    average_price = `AveragePrice`,  
    small = '4046',  
    medium = '4225',  
    large = '4770',  
  )
```

We can then take a glimpse:

```
avocado_data
```

```
## # A tibble: 18,249 x 7  
##   Date      average_price total_volume_sold small  medium large type  
##   <date>            <dbl>           <dbl>     <dbl> <dbl> <dbl> <chr>  
## 1 2015-12-27        1.33          64237. 1037. 54455. 48.2 conventional  
## 2 2015-12-20        1.35          54877. 674. 44639. 58.3 conventional  
## 3 2015-12-13        0.93          118220. 795. 109150. 130. conventional  
## 4 2015-12-06        1.08          78992. 1132. 71976. 72.6 conventional  
## 5 2015-11-29        1.28          51040. 941. 43838. 75.8 conventional  
## 6 2015-11-22        1.26          55980. 1184. 48068. 43.6 conventional  
## 7 2015-11-15        0.99          83454. 1369. 73673. 93.3 conventional  
## 8 2015-11-08        0.98          109428. 704. 101815. 80 conventional  
## 9 2015-11-01        1.02          99811. 1022. 87316. 85.3 conventional  
## 10 2015-10-25       1.07          74339. 842. 64757. 113 conventional  
## # ... with 18,239 more rows
```

The columns that will interest us the most in this chapter are:

## 5.2. Central tendency and dispersion

- `average_price` - average price of a single avocado
- `total_volume_sold` - Total number of avocados sold
- `type` - whether the price/amount is for a conventional or an organic avocado

In particular, we will look at summary statistics for the `average_price` and `total_volume_sold`, either for the whole data set or independently for each type of avocado. Notice that both of these variables are numeric. They are vectors of numbers, each representing an observation.

### 5.2.2. Measures of central tendency

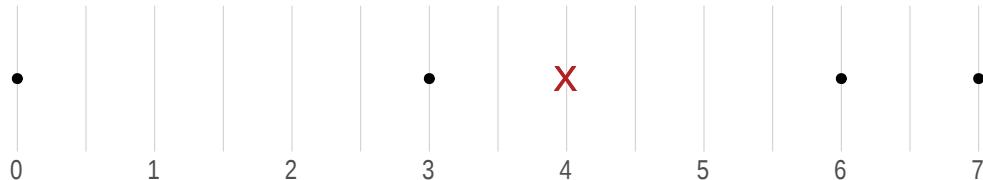
#### 5.2.2.1. The (arithmetic) mean

If  $\vec{x} = \langle x_1, \dots, x_n \rangle$  is a vector of  $n$  observations with  $x_i \in \mathbb{R}$  for all  $1 \leq i \leq n$ , the (arithmetic) **mean** of  $x$ , written  $\mu_{\vec{x}}$ , is defined as

$$\mu_{\vec{x}} = \frac{1}{n} \sum_{i=1}^n x_i .$$

The arithmetic mean can be understood intuitively as **the center of gravity**. If we place a marble on a wooden board for every  $x_i$  such that every marble is equally heavy and the differences between all data measurements are identical to the distances between the marbles, the arithmetic mean is where you can balance the board with the tip of your finger.

**Example.** The mean of the vector  $\vec{x} = \langle 0, 3, 6, 7 \rangle$  is  $\mu_{\vec{x}} = \frac{0+3+6+7}{4} = \frac{16}{4} = 4$ . The black dots in the graph below show the data observations and the red cross indicates the mean. Notice that the mean is clearly *not* the mid-point between the maximum and the minimum (which here would be 3.5).



To calculate the mean of a large vector, R has a built-in function `mean`, which we have in fact used frequently before. Let's use it to calculate the mean of the variable `average_price` for different types of avocados:

## 5. Summary statistics

```
avocado_data %>%
  group_by(type) %>%
  summarise(
    mean_price = mean(average_price)
  )

## # A tibble: 2 x 2
##   type      mean_price
##   <chr>        <dbl>
## 1 conventional     1.16
## 2 organic          1.65
```

Unsurprisingly, the overall mean of the observed prices is (numerically) higher for organic avocados than for conventional ones.

**Excursion.** It is also possible to conceptualize the arithmetic mean as the **expected value** when sampling from the observed data. This is useful for linking the mean of a data sample to the expected value of a random variable, a concept we will introduce in Chapter 7. Suppose you have gathered the data  $\vec{x} = \langle 0, 3, 6, 7 \rangle$ . What is the expected value that you think you will obtain if you sample from this data vector once? – Wait! What does that even mean? Expected value? Sampling once?

Suppose that some joker from around town invites you for a game. The game goes like this. The joker puts a ball in an urn, one for each data observation. The joker writes the observed value on the ball corresponding to that value. You pay the joker a certain amount of money to be allowed to draw one ball from the urn. The balls are indistinguishable and the process of drawing is entirely fair. You receive the number corresponding to the ball you drew paid out in silver coins. (For simplicity, we assume that all numbers are non-negative, but that is not crucial. If a negative number is drawn, you just have to pay the joker that amount.)

How many silver coins would you maximally pay to play one round? Well, of course, no more than four (unless you value gaming on top of silver)! This is because 4 is the expected value of drawing once. This, in turn, is because every ball has a chance of 0.25 of being drawn. So you can expect to earn 0 silver with a 25% chance, 3 with a 25% chance, 6 with a 25% chance and 7 with a 25% chance. In this sense, the mean is the expected value of sampling once from the observed data.

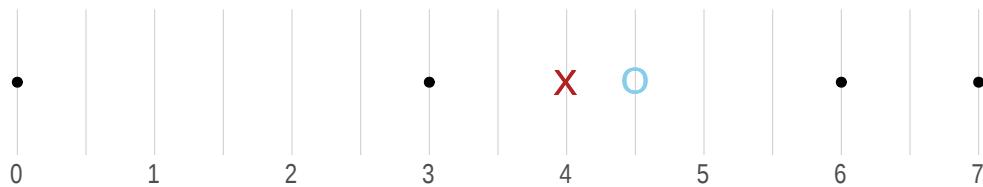
### 5.2.2.2. The median

If  $\vec{x} = \langle x_1, \dots, x_n \rangle$  is a vector of  $n$  data observations from an at least ordinal measure and if  $\vec{x}$  is ordered such that for all  $1 \leq i < n$  we have  $x_i \leq x_{i+1}$ , the **median** is the

## 5.2. Central tendency and dispersion

value  $x_i$  such that the number of data observations that are bigger or equal to  $x_i$  and the number of data observations that are smaller or equal to  $x_i$  are equal. Notice that this definition may yield no unique median. In that case different alternative strategies are used, depending on the data type at hand (ordinal or metric). (See also the example below.) The median corresponds to the 50% quartile, a concept introduced below.

**Example.** The median of the vector  $\vec{x} = \langle 1 = 0, 3, 6, 7 \rangle$  does not exist by the definition given above. However, for metric measures, where distances between measurements are meaningful, it is customary to take the two values “closest to where the median should be” and average them. In the example at hand, this would be  $\frac{3+6}{2} = 4.5$ . The plot below shows the data points in black, the mean as a red cross (as before) and the median as a blue circle



The function `median` from base R computes the median of a vector. It also takes an ordered factor as an argument.

```
median(c(0, 3, 6, 7))

## [1] 4.5
```

To please the avocados, let's also calculate the median price of both types of avocados and compare these to the means we calculated earlier already:

```
avocado_data %>%
  group_by(type) %>%
  summarise(
    mean_price = mean(average_price),
    median_price = median(average_price)
  )

## # A tibble: 2 x 3
##   type      mean_price median_price
##   <chr>        <dbl>       <dbl>
## 1 conventional     1.16       1.13
## 2 organic          1.65       1.63
```

## 5. Summary statistics

### 5.2.2.3. The mode

The **mode** is the value that occurred most frequently in the data. While the mean is only applicable to metric variables, and the median only to variables that are at least ordinal, the mode is only reasonable for variables that have a finite set of different possible observations (nominal or ordinal).

There is no built-in function in R to return the mode of a (suitable) vector, but it is easily retrieved by obtaining counts.

## 5.2.3. Measures of dispersion

### 5.2.3.1. Variance

Variance is a widely used and very useful measure of dispersion for metric data. The variance  $\text{Var}(\vec{x})$  of a vector of metric observations  $\vec{x}$  of length  $n$  is defined as the average of the squared distances from the mean:

$$\text{Var}(\vec{x}) = \frac{1}{n} \sum_{i=1}^n (x_i - \mu_{\vec{x}})^2$$

**Example.** The variance of the vector  $\vec{x} \langle 0, 3, 6, 7 \rangle$  is computed as:

$$\begin{aligned} \text{Var}(\vec{x}) &= \frac{1}{4} ((0-4)^2 + (3-4)^2 + (6-4)^2 + (7-4)^2) = \\ &= \frac{1}{4} (16 + 1 + 4 + 9) = \frac{30}{4} = 7.5 \end{aligned}$$

Figure 5.3 shows a geometric interpretation of variance for the running example of the vector  $\vec{x} \langle 0, 3, 6, 7 \rangle$ .

We can calculate the variance in R explicitly:

```
x <- c(0, 3, 6, 7)
sum((x - mean(x))^2) / length(x)

## [1] 7.5
```

There is also a built-in function `var` from base R. Using this we get a different result though:

```
x <- c(0, 3, 6, 7)
var(x)
```

### Geometric visualization of variance

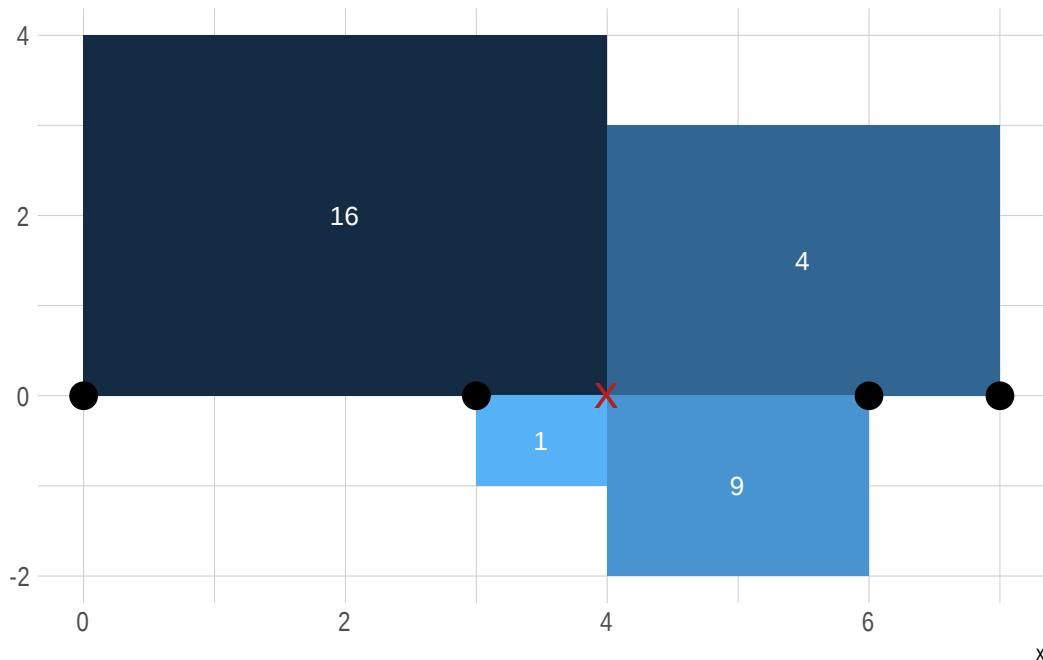


Figure 5.3.: Geometrical interpretation of variance. Four data points (black dots) and their mean (red cross) are shown, together with the squares whose sides are the differences between the observed data points and the mean. The numbers in white give the area of each square, which is also indicated by the coloring of each rectangle.

## 5. Summary statistics

```
## [1] 10
```

This is because `var` computes the variance by a slightly different formula to obtain an **unbiased estimator** of the variance for the case that the mean is not known but also estimated from the data. The formula for the unbiased estimator that R uses simply replaces the  $n$  in the denominator by  $n - 1$ :<sup>2</sup>

$$\text{Var}(\vec{x}) = \frac{1}{n-1} \sum_{i=1}^n (x_i - \mu_{\vec{x}})^2$$

### 5.2.3.2. Standard deviation

The standard deviation  $\text{SD}(\vec{x})$  or numeric vector  $\vec{x}$  is just the square root of the variance:

$$\text{SD}(\vec{x}) = \sqrt{\text{Var}(\vec{x})} = \sqrt{\frac{1}{n} \sum_{i=1}^n (x_i - \mu_{\vec{x}})^2}$$

Let's calculate the (biased) variance and standard deviation for the `average_price` of different types of avocados:

```
avocado_data %>%
  group_by(type) %>%
  summarize(
    variance_price = var(average_price),
    stddev_price   = sd(average_price),
  )

## # A tibble: 2 x 3
##   type      variance_price stddev_price
##   <chr>        <dbl>       <dbl>
## 1 conventional 0.0692      0.263
## 2 organic     0.132       0.364
```

### 5.2.3.3. Quantile

For a vector  $\vec{x}$  of at least ordinal measures, we can generalize the concept of a median to an arbitrary quantile. An  $k\%$  quantile is the element  $x_i$  in  $\vec{x}$  such that  $k\%$  of the data in  $\vec{x}$  lies below  $x_i$ . If this definition does not yield a unique element for some  $k\%$  threshold, similar methods to what we saw for the median are applied.

---

<sup>2</sup>For current purpose it is not important what biased or unbiased estimator is and why this subtle change in the formula matters. We will come back to the issue of estimation in Chapter 10.

## 5.2. Central tendency and dispersion

We can use the base R function `quantile` to obtain the 10%, 25%, 50% and 85% quantiles (just arbitrary picks) for the `average_price` in the avocado data set:

```
quantile(  
  # vector of observations  
  x = avocado_data$average_price,  
  # which quantiles  
  probs = c(0.1, 0.25, 0.5, 0.85)  
)  
  
## 10% 25% 50% 85%  
## 0.93 1.10 1.37 1.83
```

This tells us, for instance, that only about ten percent of the data observations had prices lower than \$0.93.

### 5.2.4. Quantifying confidence with bootstrapping

Bootstrapping is an elegant way to obtain measures of confidence for summary statistics. These measures of confidence can be used for parameter inference, too. We will discuss parameter inference at length in Section 10. In this course, we will not use bootstrapping as an alternative approach to parameter inference. We will, however, follow a common practice (at least in some areas of Cognitive Psychology) to use **bootstrapped 95% confidence intervals of the mean** as part of descriptive statistics, i.e., in summaries and plots of the data.

The bootstrap is a method from a more general class of algorithms, namely so-called **resampling methods**. The general idea is, roughly put, that we treat the data at hand as the true representation of reality. We then imagine that we run an experiment on that (restricted, hypothetical) reality. We then ask ourselves: what would we estimate (e.g., as a mean) in any such hypothetical experiment. The more these hypothetical measures derived from hypothetical experiments based on a hypothetical reality differ, the less confident we are in the estimate. Sounds weird, but is mindblowingly elegant.

An algorithm for constructing a 95% confidence interval of the mean of vector  $D$  of numeric data with length  $k$  looks as follows:

1. take  $k$  samples from  $D$  with replacement, call this  $D^{\text{rep}}$ <sup>3</sup>
2. calculate the mean  $\mu(D^{\text{rep}})$  of the newly sampled data

---

<sup>3</sup> $D^{\text{rep}}$  is short for “repeated Data”. We will use this concept more later on. The idea is that we consider “hypothetical data” which we have not perceived, but which we might have. Repeated data is (usually) of the same shape and form as the original, observed data, which is also sometimes noted as  $D^{\text{obs}}$  for clarity in comparison to  $D^{\text{rep}}$ .

## 5. Summary statistics

3. repeat steps 1 and 2 to gather  $r$  means of different resamples of  $D$ ; call the result vector  $\mu_{\text{sampled}}$
4. the boundaries of the 95% inner quantile of  $\mu_{\text{sampled}}$  are the bootstrapped 95% confidence interval of the mean

The higher  $r$ , i.e., the more samples we take, the better the estimate. The higher  $k$ , i.e., the more observations we have to begin with, the less variable the means  $\mu(D^{\text{rep}})$  of the resampled data will usually be. Hence, usually, the higher  $k$  the smaller the bootstrapped 95% confidence interval of the mean.

Here is a convenience function that we will use throughout the book to produce bootstrapped 95% confidence intervals of the mean:

```
## takes a vector of numbers and returns bootstrapped 95% ConfInt
## for the mean, based on `n_resamples` re-samples (default: 1000)
bootstrapped_CI <-  function(data_vector, n_resamples = 1000) {
  resampled_means <- map_dbl(1:n_resamples, function(i) {
    mean(sample(x = data_vector,
                 size = length(data_vector),
                 replace = T)
        )
  }
)
tibble(
  'lower' = quantile(resampled_means, 0.025),
  'mean'  = mean(data_vector),
  'upper' = quantile(resampled_means, 0.975)
)
}
```

Applying this method to the vector of average avocado prices we get:

```
bootstrapped_CI(avocado_data$average_price)
```

```
## # A tibble: 1 x 3
##   lower  mean upper
##   <dbl> <dbl> <dbl>
## 1 1.40  1.41  1.41
```

Notice that, since `average_price` has length 18249, i.e., we have  $k = 18249$  observations in the data, the bootstrapped 95% confidence interval is rather narrow. Compare this against a case of  $k = 300$  obtained by only looking at the first 300 entries in `average_price`:

```
# first 300 observations of `average price` only
smaller_data <- avocado_data$average_price[1:300]
bootstrapped_CI(smaller_data)

## # A tibble: 1 x 3
##   lower  mean upper
##   <dbl> <dbl> <dbl>
## 1 1.14  1.16  1.17
```

The mean is different (because we are looking at earlier time points) but, importantly, the interval is larger because with only 300 observations we have less confidence in the estimate.

#### 5.2.4.1. Excursion: Summary functions with multiple outputs, using nested tibbles

To obtain summary statistics for different groups of a variable, we can use the function `bootstrapped_CI` conveniently in concert with **nested tibbles**, as demonstrated here:

```
avocado_data %>%
  group_by(type) %>%
  # nest all columns except grouping-column 'type' in a tibble
  # the name of the new column is 'price_tibbles'
  nest(.key = "price_tibbles") %>%
  # collect the summary statistics for each nested tibble
  # the outcome is a new column with nested tibbles
  summarise(
    CIs = map(price_tibbles, function(d) bootstrapped_CI(d$average_price))
  ) %>%
  # unnest the newly created nested tibble
  unnest(CIs)

## # A tibble: 2 x 4
##   type      lower  mean upper
##   <chr>     <dbl> <dbl> <dbl>
## 1 conventional 1.15  1.16  1.16
## 2 organic      1.65  1.65  1.66
```

Using nesting in this case is helpful because we only want to run the bootstrap function once, but we need both of the numbers it returns. The following explains nesting based on this example.

## 5. Summary statistics

To understand what is going on with nested tibbles, notice that the `nest` function in this example creates a nested tibble with just two rows, one for each value of the variable `type`, each of which contains a tibble that contains all the data. The column `price_tibbles` in the first row contains the whole data for all observations for conventional avocado:

```
avocado_data %>%
  group_by(type) %>%
  # nest all columns except grouping-column 'type' in a tibble
  # the name of the new column is 'price_tibbles'
  nest(.key = "price_tibbles") %>%
  # extract new column with tibble
  pull(price_tibbles) %>%
  # peak at the first entry in this vector
  .[1] %>% head()

## <list_of<
##   tbl_df<
##     Date          : date
##     average_price : double
##     total_volume_sold: double
##     small         : double
##     medium        : double
##     large         : double
##   >
## >[1]>
## [[1]]
## # A tibble: 9,126 x 6
##       Date     average_price total_volume_sold small medium large
##     <date>           <dbl>            <dbl>    <dbl>   <dbl>  <dbl>
## 1 2015-12-27      1.33        64237.  1037.  54455.  48.2
## 2 2015-12-20      1.35        54877.  674.   44639.  58.3
## 3 2015-12-13      0.93       118220.  795.   109150. 130.
## 4 2015-12-06      1.08       78992.  1132.  71976.  72.6
## 5 2015-11-29      1.28       51040.  941.   43838.  75.8
## 6 2015-11-22      1.26       55980.  1184.  48068.  43.6
## 7 2015-11-15      0.99       83454.  1369.  73673.  93.3
## 8 2015-11-08      0.98       109428. 704.   101815. 80
## 9 2015-11-01      1.02       99811. 1022.  87316.  85.3
## 10 2015-10-25     1.07       74339.  842.   64757. 113
## # ... with 9,116 more rows
```

After nesting, we call the custom function `bootstrapped_CI` on the variable `average_price` inside of every nested tibble, so first for the conventional, then

## 5.2. Central tendency and dispersion

the organic avocados. The results is a nested tibble. If we now look inside the new column CI, we see that it's cells contain tibbles with the output of each call of `bootstrapped_CI`:

```
avocado_data %>%
  group_by(type) %>%
  # nest all columns except grouping-column 'type' in a tibble
  # the name of the new column is 'price_tibbles'
  nest(.key = "price_tibbles") %>%
  # collect the summary statistics for each nested tibble
  # the outcome is a new column with nested tibbles
  summarise(
    CIs = map(price_tibbles, function(d) bootstrapped_CI(d$average_price)))
  ) %>%
  # extract new column vector with nested tibbles
  pull(CIs) %>%
  # peak at the first entry
  .[1] %>% head()

## [[1]]
## # A tibble: 1 x 3
##   lower  mean upper
##   <dbl> <dbl> <dbl>
## 1 1.15  1.16  1.16
```

Finally, we unnest the new column `CIs` to obtain the final result (code repeated from above):

```
avocado_data %>%
  group_by(type) %>%
  # nest all columns except grouping-column 'type' in a tibble
  # the name of the new column is 'price_tibbles'
  nest(.key = "price_tibbles") %>%
  # collect the summary statistics for each nested tibble
  # the outcome is a new column with nested tibbles
  summarise(
    CIs = map(price_tibbles, function(d) bootstrapped_CI(d$average_price)))
  ) %>%
  # unnest the newly created nested tibble
  unnest(CIs)

## # A tibble: 2 x 4
##   type      lower  mean upper
##   <fct>    <dbl> <dbl> <dbl>
```

## 5. Summary statistics

```
## <chr>     <dbl> <dbl> <dbl>
## 1 conventional 1.15  1.16  1.16
## 2 organic      1.65  1.65  1.66
```

## 5.3. Co-variance & correlation

### 5.3.1. Covariance

Let  $\vec{x}$  and  $\vec{y}$  are two vectors of numeric data of the same length, such that all pairs of  $x_i$  and  $y_i$  are associated observation. For example: the vectors `avocado_data$total_volume_sold` and `avocado_data$average_price` would be such vectors. The covariance between  $\vec{x}$  and  $\vec{y}$  measures, intuitively put, the degree to which changes in one vector correspond with changes in the other. Formally, covariance is defined as follows (notice that we use  $n - 1$  in the denominator, to obtain an unbiased estimator if the means are unknown):

$$\text{Cov}(\vec{x}, \vec{y}) = \frac{1}{n-1} \sum_{i=1}^n (x_i - \mu_{\vec{x}}) (y_i - \mu_{\vec{y}})$$

There is a visually intuitive geometric interpretation of covariance. To see this, let's look at a short contrived example.

```
contrived_example <-
  tribble(
    ~x,    ~y,
    2,    2,
    2.5,  4,
    3.5,  2.5,
    4,    3.5
  )
```

First, notice that the mean of x and y is 3:

```
means_contr_example <- map_df(contrived_example, mean)
means_contr_example

## # A tibble: 1 x 2
##       x     y
##   <dbl> <dbl>
## 1     3     3
```

We can then compute the covariance as follows:

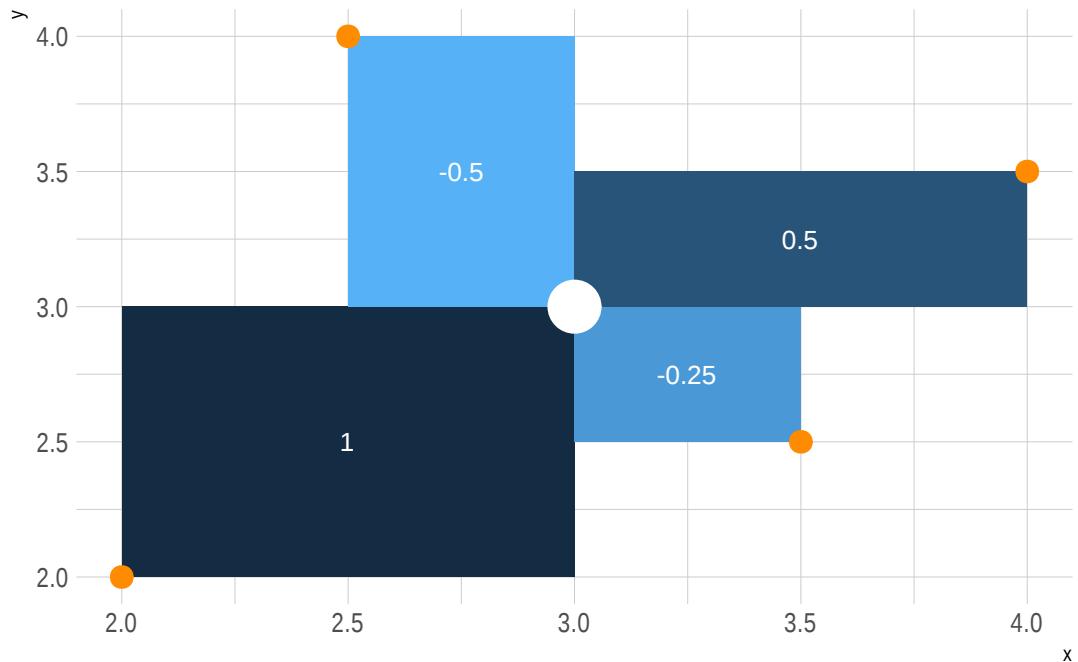
```
contrived_example <-
  contrived_example %>%
  mutate(
    area_rectangle = (x - mean(x)) * (y - mean(y)),
    covariance     = 1 / (n() - 1) * sum((x - mean(x)) * (y - mean(y)))
  )
contrived_example

## # A tibble: 4 x 4
##       x     y area_rectangle covariance
##   <dbl> <dbl>      <dbl>      <dbl>
## 1     2     2        1        0.25
## 2     2.5    4       -0.5        0.25
## 3     3.5    2.5      -0.25       0.25
## 4     4     3.5        0.5        0.25
```

Similar to what we did with variance, we can give a geometrical interpretation of covariance. Figure ?? shows the four summands contributing to the covariance of the `contrived_example`. What this graph clearly shows is that summands can have different signs. If  $x_i$  and  $y_i$  are both bigger than the mean, or if both are smaller than the mean, then the corresponding summand is positive. Otherwise, the corresponding summand is negative. This means that covariance captures the degree to which pairs  $x_i$  and  $y_i$  tend to deviate from the mean in the same general direction. A positive covariance is indicative of a positive general association between  $\vec{x}$  and  $\vec{y}$ , while a negative covariance suggests that as you increase  $x_i$  the associated  $y_i$  becomes smaller.

## 5. Summary statistics

### Rectangular areas contributing to the computation of covariance



We can, of course, also calculate covariance just with a built-in base R function, `cov`:

```
with(contrived_example, cov(x,y))  
  
## [1] 0.25
```

And, using this function we can calculate the covariance between `total_volume_sold` and `average_price` in the avocado data:

```
with(avocado_data, cov(log(total_volume_sold), average_price))  
  
## [1] -0.5388084
```

Interestingly, the negative covariance in this example suggests that that across all associated data pairs, the larger `total_volume_sold`, the lower `average_price`. It is important that this is a descriptive statistic, and that this is not to be interpreted as evidence or a causal relation between the two measures of interest. Not in this example, not in any other. Covariance describes associated data points; it is not evidence for causal relationships.

### 5.3.2. Correlation

The problem with covariance is that it is not invariant under linear transformation. Consider the `contrived_example` from above once more. The original data had the following covariance:

```
with(contrived_example, cov(x,y))

## [1] 0.25
```

But if we just linearly transform, say, vector  $y$  to  $1000 * y + 500$  (e.g., because we switch to an equivalent, but numerically different measuring scale), we obtain:

```
with(contrived_example, cov(x,1000 * y + 500))

## [1] 250
```

To compensate for this problem, we can look at **Bravais-Pearson correlation**, which is covariance standardized by standard deviations:

$$r_{\vec{x}\vec{y}} = \frac{\text{Cov}(\vec{x}, \vec{y})}{\text{SD}(\vec{x}) \text{ SD}(\vec{y})}$$

Let's check invariance under linear transformation, using the built-in function `cor`. The correlation coefficient for the original data is:

```
with(contrived_example, cor(x,y))

## [1] 0.3
```

The correlation coefficient for the data with linearly transformed  $y$  is:

```
with(contrived_example, cor(x,1000 * y + 500))

## [1] 0.3
```

Indeed, the correlation coefficient is nicely bounded to lie between -1 and 1. A correlation coefficient of 0 is to be interpreted as the absence of any correlation. A correlation coefficient of 1 is a perfect positive correlation (the higher  $x_i$ , the higher  $y_i$ ) and -1 indicates a perfect negative correlation (the higher  $x_i$  the lower  $y_i$ ). Again, pronounced

## 5. Summary statistics

positive or negative correlations are *not* to be confused with strong evidence for a causal relation. It is just a descriptive statistic on properties of associated measurements.

In the avocado data, the logarithm of `total_volume_sold` shows a noteworthy correlation with `average_price`. This is also visible in Figure 5.4.

```
with(avocado_data, cor(log(total_volume_sold), average_price))  
  
## [1] -0.5834087
```

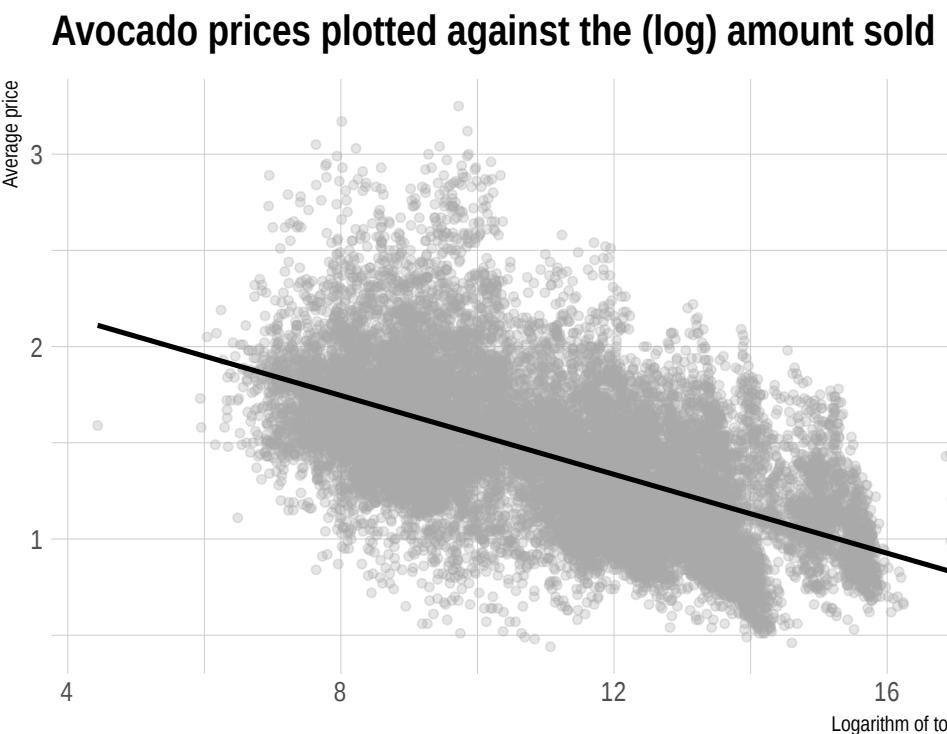


Figure 5.4.: Scatter plot of avocado prices, plotted against (logarithms of) the total amount sold. The black line is a linear regression line indicating the (negative) correlation between these measures (more on this later).

# 6. Data Visualization

Numerical summaries of complex data always incur information loss. Still lossy, but less so (if done well), is visualization. Any serious data analysis should start with a process in which the analyst becomes intimate with the data at hand. Visualization is an integral part of data-intimacy.

Section 6.1 demonstrates how summary statistics can be misleading and how a simple visualization can be much more revealing. Section 6.2 offers some reflection on what makes a data visualization successful. Section 6.3 introduces the basics of data visualization with the `ggplot` package, an integral part of the tidyverse, based on a scatter plot for the avocado price data. Going beyond scatter plots, section 6.4 looks at some common types of plots and how to realize them using the `geom_` family of functions in `ggplot`.

The learning goals for this chapter are:

- obtain a basic understanding of better/worse plotting
  - understand the idea of *hypothesis-driven visualization*
- develop a basic understanding of the ‘grammar of graphs’
- get familiar with frequent visualization strategies
  - barplots, densities, violins, error bars etc.
- be able to fine-tune graphs for better visualization

## 6.1. Motivating example: Anscombe’s quartet

To see how summary statistics can be highly misleading, and how a simple plot can reveal a lot more, consider a famous dataset available in R (Anscombe 1973):

```
glimpse(anscombe %>% as_tibble)

## #> #> Observations: 11
## #> #> Variables: 8
## #> #> $ x1 <dbl> 10, 8, 13, 9, 11, 14, 6, 4, 12, 7, 5
```

## 6. Data Visualization

```
## $ x2 <dbl> 10, 8, 13, 9, 11, 14, 6, 4, 12, 7, 5
## $ x3 <dbl> 10, 8, 13, 9, 11, 14, 6, 4, 12, 7, 5
## $ x4 <dbl> 8, 8, 8, 8, 8, 8, 19, 8, 8, 8
## $ y1 <dbl> 8.04, 6.95, 7.58, 8.81, 8.33, 9.96, 7.24, 4.26, 10.84, 4.82, 5.68
## $ y2 <dbl> 9.14, 8.14, 8.74, 8.77, 9.26, 8.10, 6.13, 3.10, 9.13, 7.26, 4.74
## $ y3 <dbl> 7.46, 6.77, 12.74, 7.11, 7.81, 8.84, 6.08, 5.39, 8.15, 6.42, 5.73
## $ y4 <dbl> 6.58, 5.76, 7.71, 8.84, 8.47, 7.04, 5.25, 12.50, 5.56, 7.91, 6.89
```

There are four pairs of  $x$  and  $y$  coordinates. Unfortunately, these are stored in long format with two pieces of information buried inside of the column name: for instance, the name `x3` contains the information that this column contains the  $x$  coordinates for the 3rd pair. This is rather untidy. But, using tools from the `dplyr` package, we can tidy up quickly:

```
tidy_anscombe <- anscombe %>% as_tibble %>%
  pivot_longer(
    ## we want to pivot every column
    everything(),
    ## use reg-exps to capture 1st and 2nd character
    names_pattern = "(.)(.)",
    ## assign names to new cols, using 1st part of
    ## what reg-exp captures as new column names
    names_to = c(".value", "grp")
  ) %>%
  mutate(grp = paste0("Group ", grp))
tidy_anscombe

## # A tibble: 44 x 3
##       grp         x     y
##   <chr>   <dbl> <dbl>
## 1 Group 1     10  8.04
## 2 Group 2     10  9.14
## 3 Group 3     10  7.46
## 4 Group 4      8  6.58
## 5 Group 1      8  6.95
## 6 Group 2      8  8.14
## 7 Group 3      8  6.77
## 8 Group 4      8  5.76
## 9 Group 1     13  7.58
## 10 Group 2    13  8.74
## # ... with 34 more rows
```

Here are some summary statistics for each of the four pairs:

### 6.1. Motivating example: Anscombe's quartet

```

tidy_anscombe %>%
  group_by(grp) %>%
  summarise(
    mean_x      = mean(x),
    mean_y      = mean(y),
    min_x       = min(x),
    min_y       = min(y),
    max_x       = max(x),
    max_y       = max(y),
    crrltn     = cor(x,y)
  )

## # A tibble: 4 x 8
##   grp      mean_x mean_y min_x min_y max_x max_y crrltn
##   <chr>    <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
## 1 Group 1     9    7.50     4   4.26    14  10.8   0.816
## 2 Group 2     9    7.50     4    3.1     14   9.26   0.816
## 3 Group 3     9    7.5      4   5.39    14  12.7   0.816
## 4 Group 4     9    7.50     8   5.25    19  12.5   0.817

```

These numeric indicators suggest that each pair of  $x$  and  $y$  values is very similar. Only the ranges seem to differ. A brilliant example of how misleading numeric statistics can be, as compared to a plot of the data:<sup>1</sup>

```

tidy_anscombe %>%
  ggplot(aes(x, y)) +
  geom_smooth(method = lm, se = F, color = "darkorange") +
  geom_point(color = project_colors[3], size = 2) +
  scale_y_continuous(breaks = scales::pretty_breaks()) +
  scale_x_continuous(breaks = scales::pretty_breaks()) +
  labs(
    title = "Anscombe's Quartet", x = NULL, y = NULL,
    subtitle = bquote(y == 0.5 * x + 3 ~ (r %~~% .82) ~ "for all groups")
  ) +
  facet_wrap(~grp, ncol = 2, scales = "free_x") +
  theme(strip.background = element_rect(fill = "#f2f2f2", colour = "white"))

```

---

<sup>1</sup>It is not important to understand this code when you first read this chapter. But at the end of the chapter you should be able to understand (passively) what is going on here.

## 6. Data Visualization

### Anscombe's Quartet

$$y = 0.5x + 3 \quad (r \approx 0.82) \text{ for all groups}$$

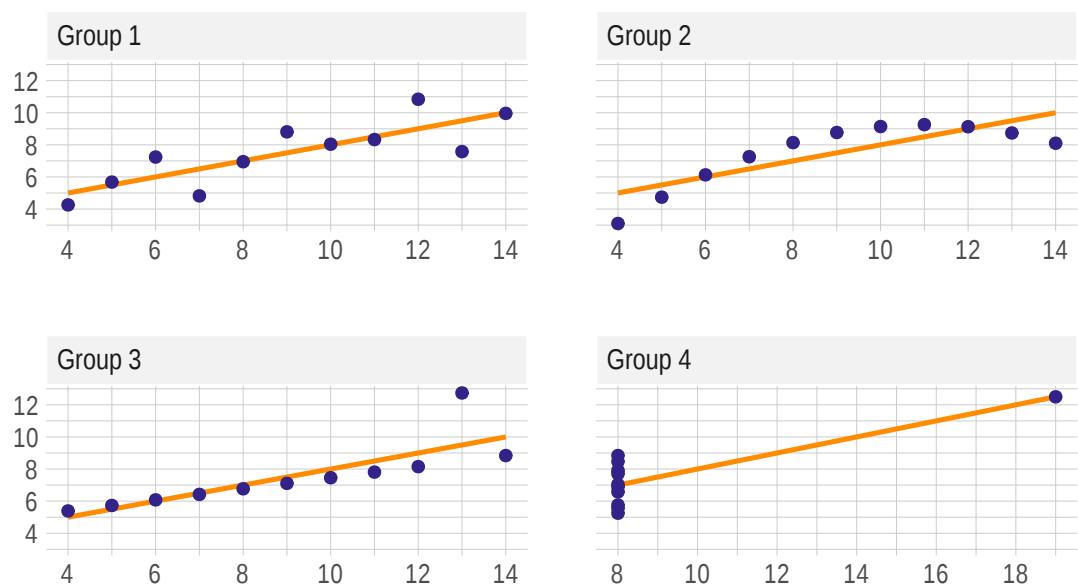


Figure 6.1.: Anscombe's Quartet: four different data sets all of which receive the same correlation score.

## 6.2. Visualization: the good, the bad and the info-graphic

Producing good data visualization is very difficult. There are no uncontroversial criteria for what a good visualization should be. There are, unfortunately, quite clear examples of really bad visualizations. We will look at some of these examples in the following.

An absolute classic on data visualization is an early book by Edward Tufte (1983) entitled “The Visual Display of Quantitative Information”. A distilled and over-simplified summary of Tufte’s proposal is that we should eliminate *chart junk* and increase the *data-ink ratio*, a concept which Tufte defines formally. The more information (=data) a plot conveys, the higher the data-ink ratio. The more ink it requires, the lower it is.

However, not all information in the data is equally relevant. Also, spending extra ink to reduce the recipients mental effort of retrieving the relevant information can be justified. Essentially, I would here propose to consider a special case of data visualization, common to scientific presentations. I want to speak of **hypothesis-driven visualization** as a way of communicating a clear message, the message we care most about at the current moment of (scientific) exchange. Though merely a special instances of all the goals one could pursue with data visualization, focusing on this special case is helpful because it allows us to formulate a (defeasible) rule of thumb for good visualization in analogy to how natural language ought to be used in order to achieve optimal cooperative information flow (at least as conceived by authors such as ...)

### The vague & defeasible rule of thumb of good data visualization (according to the author).

“Communicate a maximal degree of relevant true information in a way that minimizes the recipient’s effort of retrieving this information.”

Interestingly, just like natural language also needs to rely on a conventional medium for expressing ideas which might put additional constraints on what counts as optimal communication (e.g., we might not be allowed to drop a pronoun in English even though it is clearly recoverable from the context, and Italian speakers would happily omit it), so do certain unarticulated conventions in each specific scientific field.<sup>2</sup>

Here are a few examples of bad plotting.<sup>3</sup> To begin with, check out this fictitious data set:

```
large_contrast_data <- tribble(
  ~group, ~treatment, ~measurement,
```

---

<sup>2</sup>If your community only understands scatter plots and bar plots, it will not help communication but only mark you as a pompous show-off if you communicate in any other way, no matter how much better you think this is.

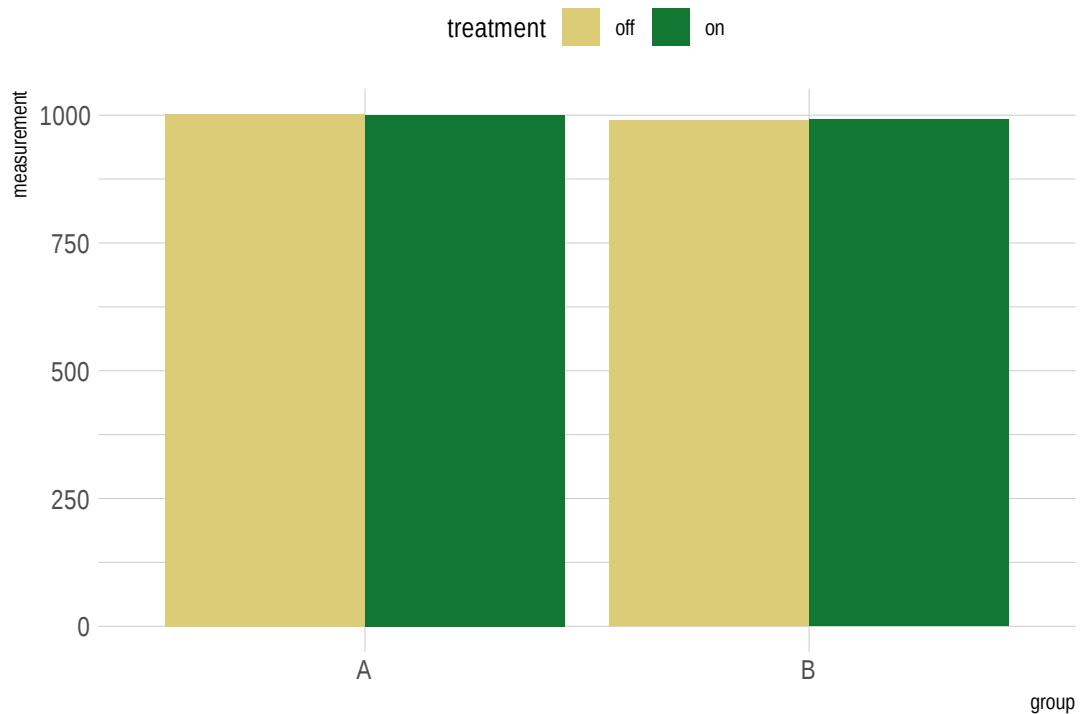
<sup>3</sup>For more disinspiration, see for example this curated list of delightfully bad visualizations from actual publications.

## 6. Data Visualization

```
"A",     "on",      1000,  
"A",     "off",     1002,  
"B",     "on",      992,  
"B",     "off",     990  
)
```

If we are interested in any potential influence of variables `group` and `treatment` on the measurement in question, the following graph is ruinously unhelpful because the large size of the bars renders the relatively small differences between them almost entirely unspottable.

```
large_contrast_data %>%  
  ggplot(aes(x = group, y = measurement, fill = treatment)) +  
  geom_bar(stat = "identity", position = "dodge")
```



A better visualization would be this:

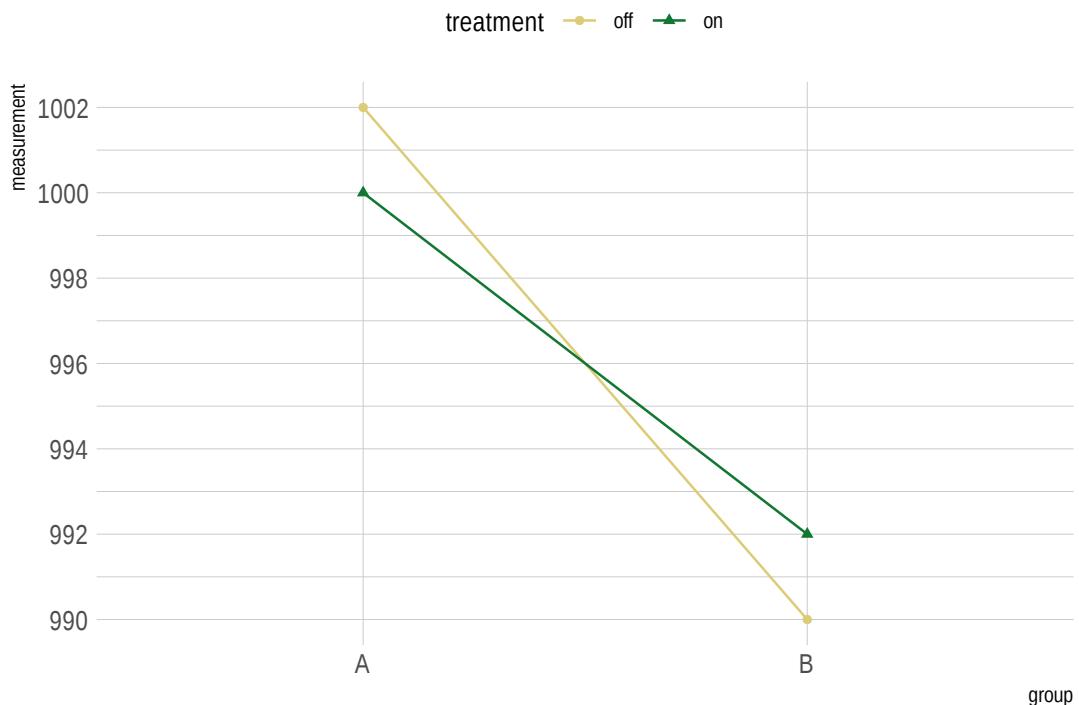
```
large_contrast_data %>%  
  ggplot(aes(  
    x = group,
```

## 6.2. Visualization: the good, the bad and the info-graphic

```

y = measurement,
shape = treatment,
color = treatment,
group = treatment
)
) +
geom_point() +
geom_line() +
scale_y_continuous(breaks = scales::pretty_breaks())

```



The following examples use the Bio-Logic Jazz-Metal data set, in particular the following derived table of counts or the derived table of proportions:

`BLJM_associated_counts`

```

## # A tibble: 4 x 3
##   JM     LB       n
##   <chr> <chr>   <int>
## 1 Jazz   Biology    38
## 2 Jazz   Logic      26

```

## 6. Data Visualization

```
## 3 Metal Biology    20
## 4 Metal Logic      18
```

It is probably hard to believe but Figure 6.2 was obtained without further intentional uglification just by choosing a default 3D bar plot display in Microsoft's Excel. It does actually show the relevant information but it is entirely useless for a human observer without a magnifying glass, professional measuring tools and a calculator.

**Counts of music-subject choice pairs**

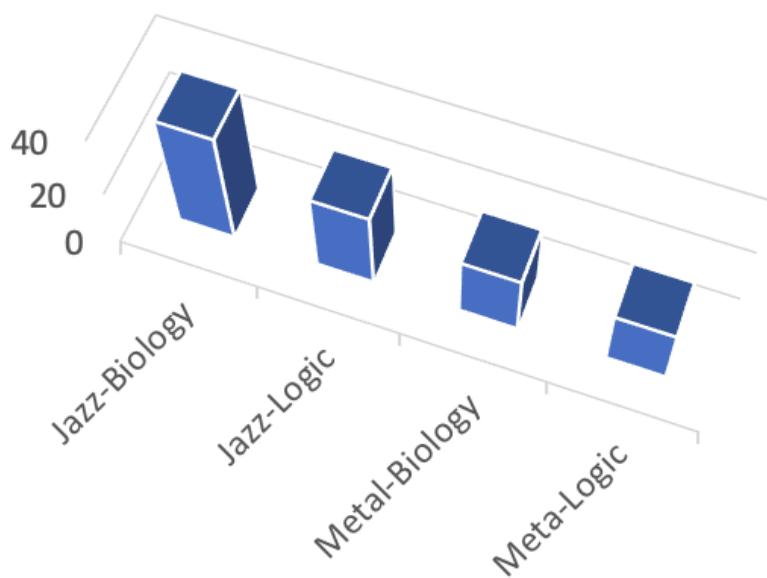


Figure 6.2.: Example of a frontrunner for the prize of today's most complete disaster in the visual communication of information.

It gets slightly better with the following pie chart of the same numerical information, also generated with Microsoft's Excel. Subjectively, Figure 6.3 is pretty much anything but pretty. Objectively, it is better than the previous visualization in terms of 3D bar plots shown in Figure 6.2 but the pie chart is still not useful for answering the question which we care about, namely whether logicians are more likely to prefer Jazz over Metal than biologists.

We can produce a much more useful representation with the code below. (A similar visualization also appeared as Figure 5.1 in the previous chapter.)

```
BLJM_associated_counts %>%
  ggplot(
    aes(
```

### Proportions of music-subject choice pairs

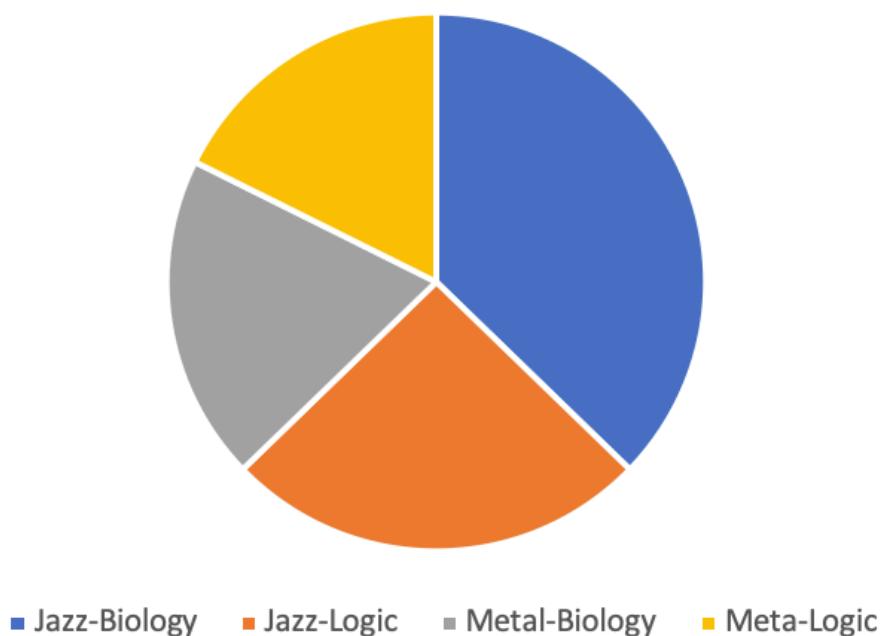


Figure 6.3.: Example of a rather unhelpful visual representation of the BLJM data (when the research question is whether logicians are more likely to prefer Jazz over Metal than biologists).

## 6. Data Visualization

```
x = LB,  
y = n,  
color = JM,  
shape = JM,  
group = JM  
)  
) +  
geom_point(size = 3) + geom_line() +  
labs(  
  title = "Counts of choices of each music+subject pair",  
  x = "",  
  y = ""  
)
```

**Counts of choices of each music+subject pair**



**Info-Graphics.** Scientific communication with visualized data is different from other modes of communication with visualized data. These other contexts come with different requirements for good data visualization. Good examples of highly successful *info-graphics* are produced by the famous illustrator Nigel Holmes, for instance. Figure 6.4 is an example from Holmes' website showing different amounts of energy consumption for different household appliances. The purpose of this visualization is not (only) to

communicate information about which of the listed household appliances is most energy intensive. It's main purpose is to raise awareness for the unexpectedly large energy consumption of household appliances in general (in stand-by mode).<sup>4</sup>

## 6.3. Basics of `ggplot`

In this section we will work towards a first plot with `ggplot`. It will be a scatter plot (more on different kinds of plots in Section 6.4) for the avocado price data. Check out the `ggplot` cheat sheet for a quick overview of the nuts and bolts of `ggplot`.

The following introduces the following key concepts of `ggplot`:

- **incremental composition:** adding elements or changing attributes of a plot incrementally
- **convenience functions & defaults:** a closer look at high-level convenience functions (like `geom_point`) and what they actually do
- **layers:** seeing how layers are stacked when we call, e.g. different `geom_` functions in sequence
- **grouping:** what happens when we use grouping information (e.g., for color, shape or in facets)

The section finishes with a first full example of plot that has different layers, uses grouping, and customizes a few other things.

### 6.3.1. Incrementally composition of a plot

The “gg” in the package name `ggplot` is short for “grammar of graphs”. It provides functions for describing scientific data plots in a compositional manner, i.e., for dealing with different recurrent elements in a plot in an additive way. As a results of this approach, we will use the symbol `+` to *add* more and more elements (or to override the implicit defaults in previously evoked elements) to build a plot. For example, we can obtain a scatter plot for the avocado price data simply by first calling the function `ggplot`, which just creates an empty plot:

```
incrementally_built_plot <- ggplot()
```

The plot stored in variable `incrementally_built_plot` is very boring. Take a look:

```
incrementally_built_plot
```

---

<sup>4</sup>Image retrieved from Nigel Holmes' website website on November 25 2019.

## 6. Data Visualization

### Vampire Energy

Even when household appliances are turned off, most are still using some electricity. Appliances are either in passive standby mode (the clock on the microwave is still ticking) or active standby mode (the VCR is off, but programmed to record something).

These numbers are for average standby modes, showing how much electricity is sucked out annually, in kilowatt hours, and what it costs you—assuming 11 cents per kilowatt hour. Red lines show passive standby mode; blue lines show active standby mode.

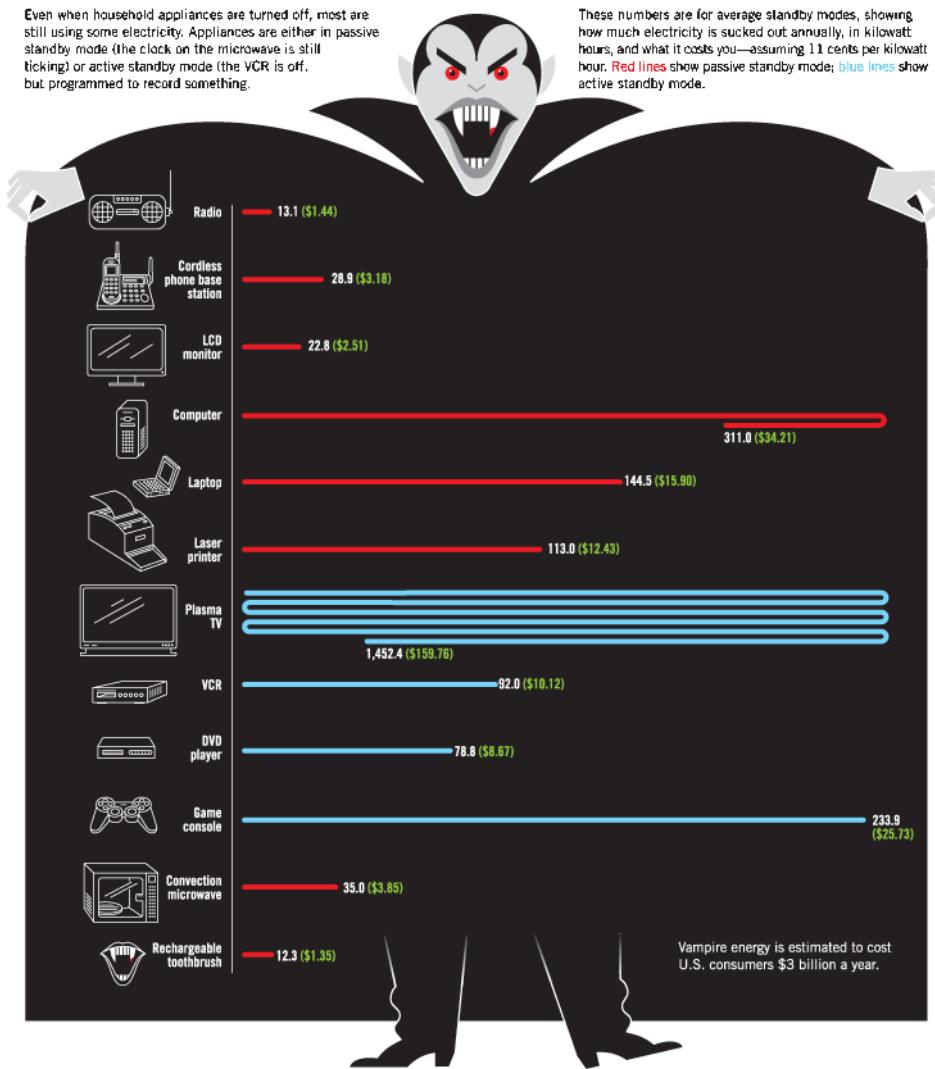


Figure 6.4.: Example of an infographic. While possibly considered 'chart junk' in a scientific context, the eye-catching and highly memorable (and pretty!) artwork serve a strong secondary purpose in contexts other than scientific ones where hypothesis-driven precise communication with visually presented data is key.

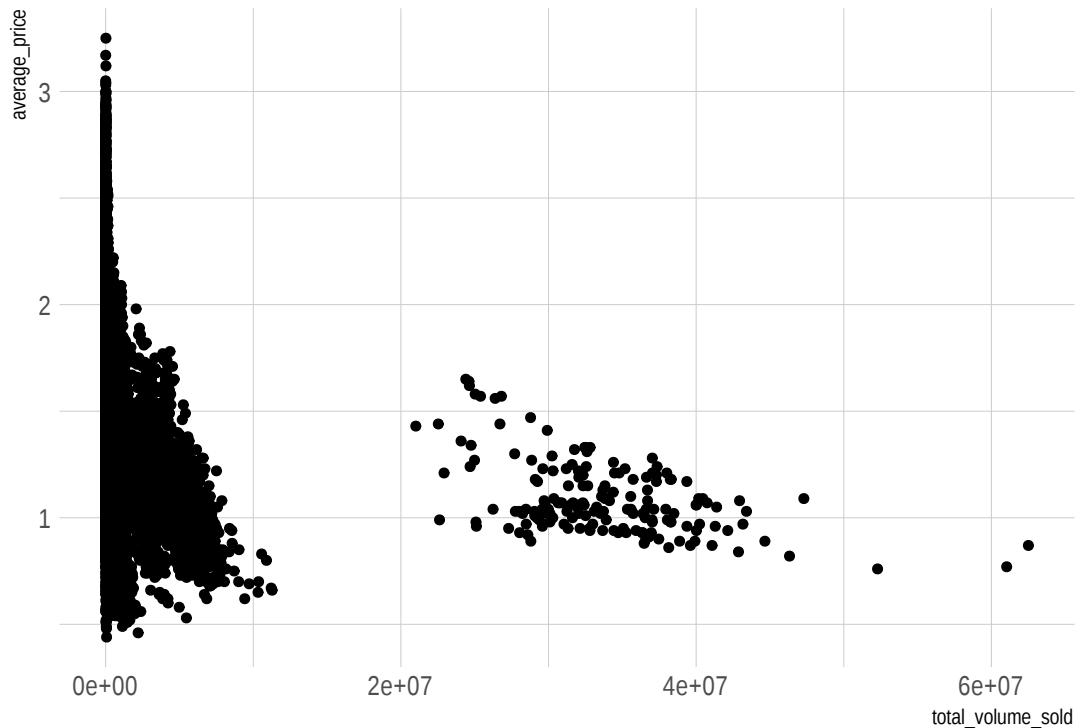
As you can see, you do not see anything except a (white) canvas. But we can add some stuff. Don't get hung up on the details right now, just notice that we use `+` to add stuff to our plot:<sup>5</sup>

```
incrementally_built_plot +
  # add a geom of type `point` (=> scatter plot)
  geom_point(
    # what data to use
    data = avocado_data,
    # supply a mapping (in the form of an 'aesthetic' (see below))
    mapping = aes(
      # which variable to map onto the x-axis
      x = total_volume_sold,
      # which variable to map onto the y-axis
      y = average_price
    )
  )
```

---

<sup>5</sup>If you run this code for yourself, the output is likely to look different from what is shown here. This is because this web-book uses a default theme for all of its plots. We will come back to customization with themes later.

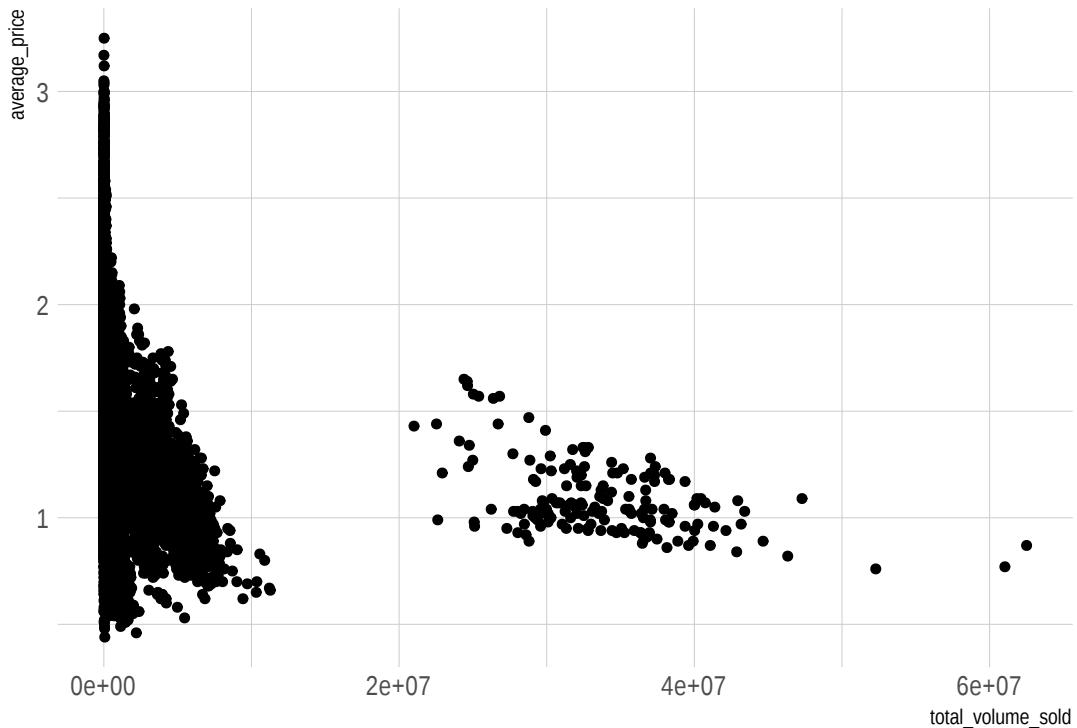
## 6. Data Visualization



You see that the function `geom_point` is what makes the points appear. You tell it which data to use and which mapping of variables from the data set to elements in the plot you like. That's it, at least to begin with.

We can also supply the information about the data to use and the aesthetic mapping in the call to function `ggplot`. Doing so will make this information the default for any subsequently added layer. Notice also that the `data` argument in function `ggplot` is the first argument, so we will frequently make use of piping, like in the following code which is equivalent to the previous in terms of output:

```
avocado_data %>%
  ggplot(aes(x = total_volume_sold, y = average_price)) +
  geom_point()
```



### 6.3.2. Elements in the layered grammar of graphs

Let's take a step back. Actually, the function `geom_point` is a convenience function that does a lot of things automatically for us. It helps understanding subsequent code if we peek under the hood at least for a brief moment initially, if only to just realize where some of the terminology in and round the “grammar of graphs” comes from.

The `ggplot` package defines a **layered grammar of graphs** (Wickham 2010). This is a structured description language for plots (relevant for data science). It uses a smart system of defaults so that it suffices to often just call a convenience wrapper like `geom_point`. But underneath there is the possibility of tinkering with (almost?) all of the (layered) elements and to change the defaults if need be.

The process of mapping data onto a visualization essentially follows this route:

data -> statistical transformation -> geom. object -> aesthetics

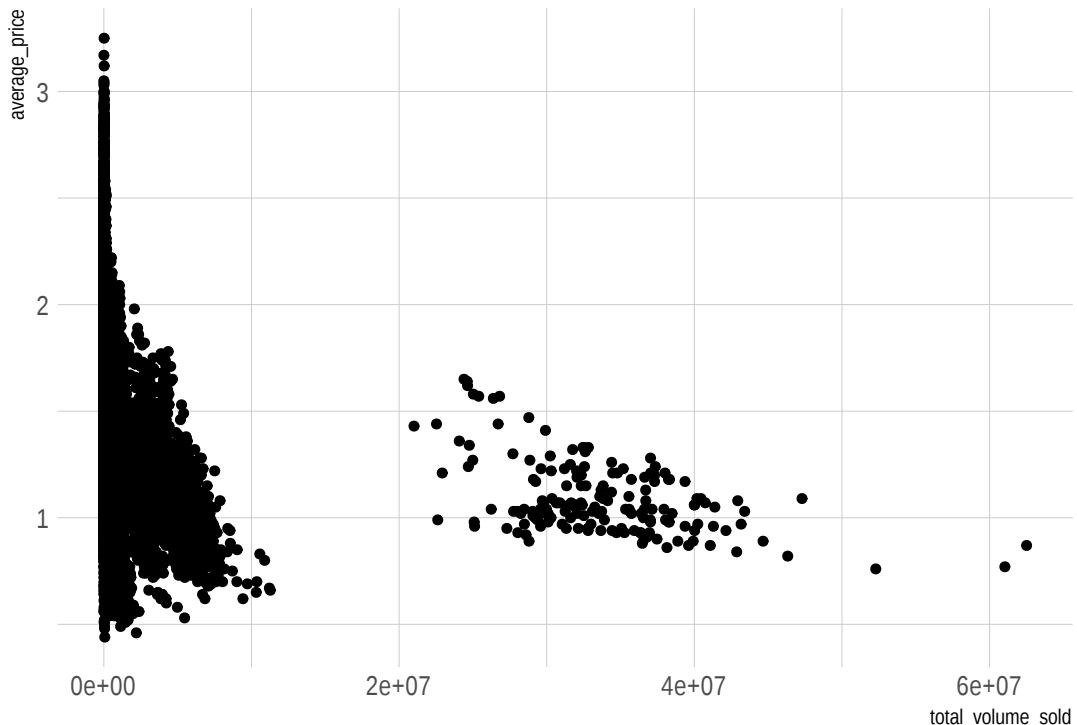
You supply (tidy) data. The data is then transformed (e.g., by computing a summary statistic) in some way or another. This could just be an “identity map” in which case you will visualize the data exactly as it is. The resulting data representation is mapped onto some spatial (geometric) appearance, like a line, a dot, or a geometric shape. Finally, there is room to alter the specific asthetics of this mapping from data to visual object, like adjusting the size or the color of a geometric object, possibly depending on some other

## 6. Data Visualization

properties it has (e.g., whether it is an observation for a conventional or an organically grown avocado).

To make explicit the steps which are implicitly carried out by `geom_points` in the example above, here is a fully verbose but output-equivalent sequence of commands that builds the same plot by defining all the basic components manually:

```
avocado_data %>%
  ggplot() +
    # plot consists of layers (more on this soon)
    layer(
      # how to map columns onto ingredients in the plot
      mapping = aes(x = total_volume_sold, y = average_price),
      # what statistical transformation should be used? - here: none
      stat = "identity",
      # how should the transformed data be visually represented? - here: as points
      geom = "point",
      # should we tinker in any other way with the positioning of each element?
      # - here: no, thank you!
      position = "identity"
    ) +
    # x & y axes are non-transformed continuous
    scale_x_continuous() +
    scale_y_continuous() +
    # we use a cartesian coordinate system (not a polar or a geographical map)
    coord_cartesian()
```



In this explicit call, we still need to specify the data and the mapping (which variable to map onto which axis). But we need to specify much more. We tell `ggplot` that we want standard (e.g., not log-transformed axis). We also tell it that our axis are continuous, that the data should not be transformed and that the visual shape (=geom) to which the data is to be mapped is a point (hence the name `geom_point`).

It is not important to understand all of these components right now. It is important to have seen them once, and to understand that `geom_point` is a wrapper around this call which assumes reasonable defaults (such as non-transformed axis, points for representation etc.).

### 6.3.3. Layers and groups

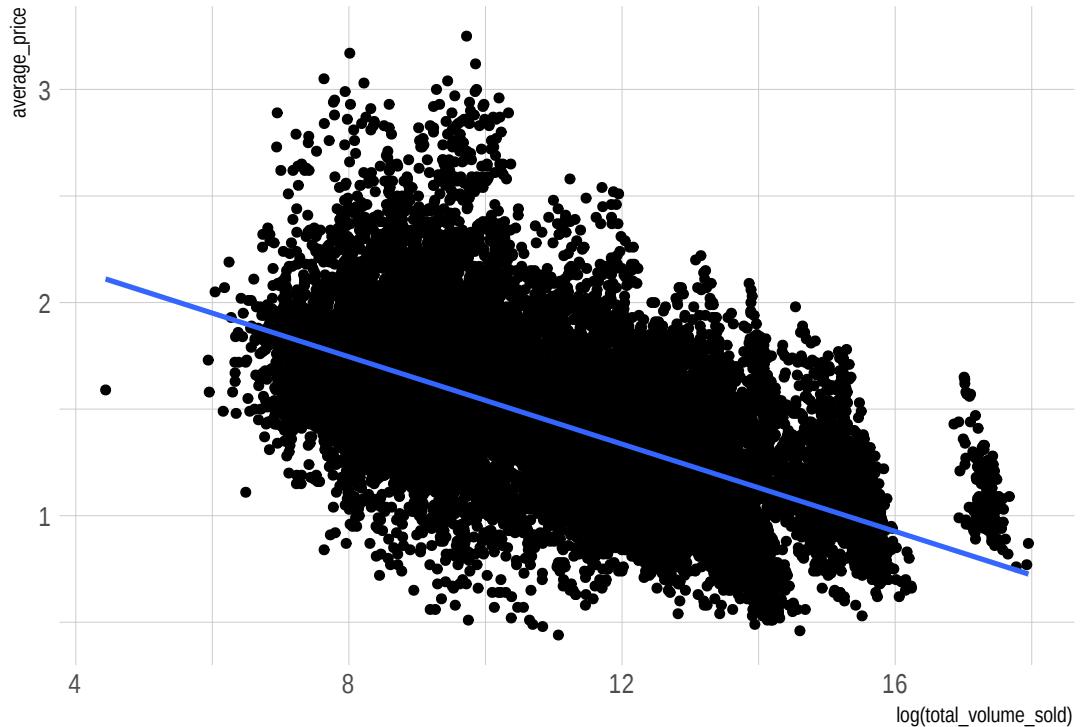
`ggplot` is the “grammar of *layered* graphs”. Plots are compositionally built by combining different layers, if need be. For example, we can use another function from the `geom_` family of functions to display a different visualization derived from the same data on top of our previous scatter plot.<sup>6</sup>

---

<sup>6</sup>Notice that, as soon as we add the linear regression line, it makes sense to use the logarithm of the `total_volume_sold` because otherwise the fit is quite ridiculous. The logarithm helps to spread out the large number of data points where `total_volume_sold` is very low, and to “bring back to the flock” the data points where the `total_volume_sold` is outliers high. It can be quite useful to use

## 6. Data Visualization

```
avocado_data %>%
  ggplot(
    mapping = aes(
      # notice that we use the log (try without it to understand why)
      x = log(total_volume_sold),
      y = average_price
    )
  ) +
  # add a scatter plot
  geom_point() +
  # add a linear regression line
  geom_smooth(method = "lm")
```

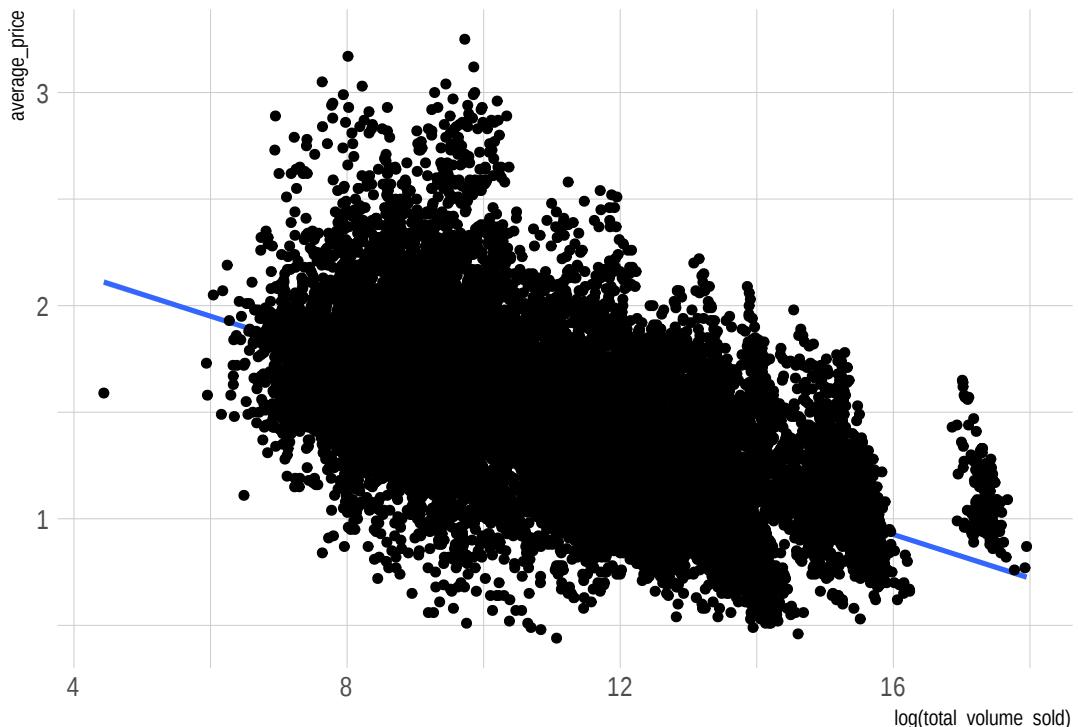


Notice that layering is really sequential. To see this, just check what happens when we reverse the calls of the `geom_` functions in the previous example:

---

such transformations, if they are well understood. It is controversial whether such transformations should precede statistical analyses, but that is not important right now.

```
avocado_data %>%
  ggplot(
    mapping = aes(
      # notice that we use the log (try without it to understand why)
      x = log(total_volume_sold),
      y = average_price
    )
  ) +
  # FIRST: add a linear regression line
  geom_smooth(method = "lm") +
  # THEN: add a scatter plot
  geom_point()
```

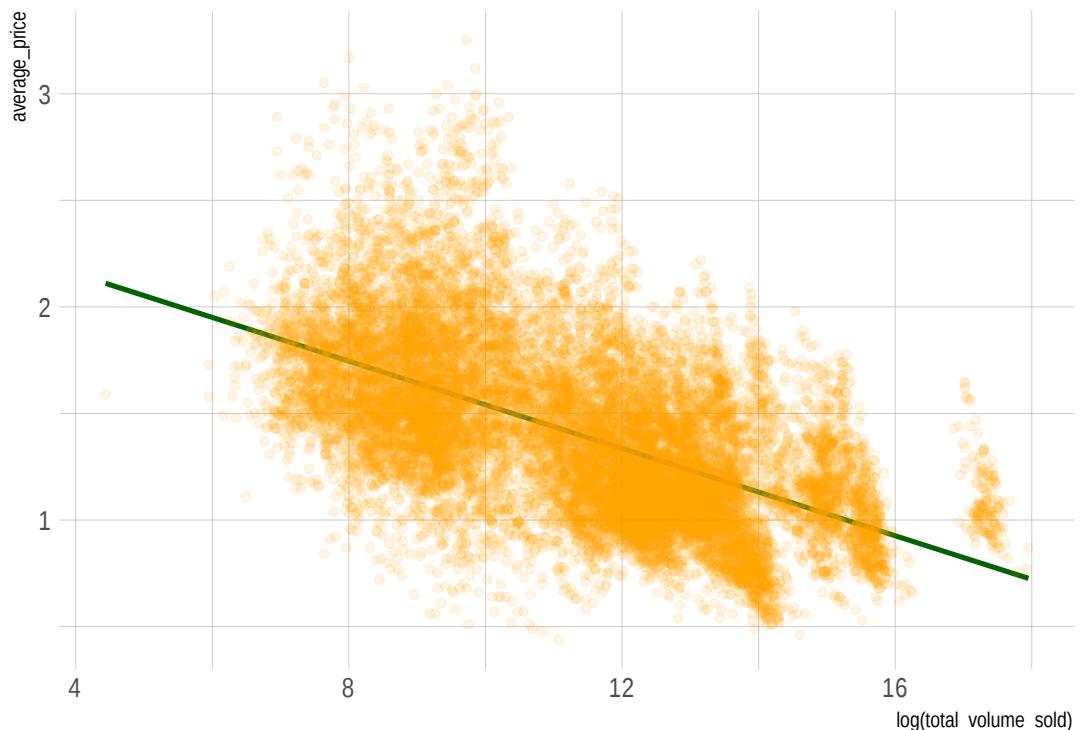


If you want lower layers to be visible behind layers added later, one possibility is to tinker with opacity, via the `alpha` parameter. Notice that the example below also changes the colors. The result is quite toxic, but at least you see the line underneath the semi-transparent points.

```
avocado_data %>%
  ggplot(
```

## 6. Data Visualization

```
mapping = aes(  
    # notice that we use the log (try without it to understand why)  
    x = log(total_volume_sold),  
    y = average_price  
)  
) +  
# FIRST: add a linear regression line  
geom_smooth(method = "lm", color = "darkgreen") +  
# THEN: add a scatter plot  
geom_point(alpha = 0.1, color = "orange")
```



The aesthetics defined in the initial call to `ggplot` are global defaults for all layers to follow, unless they are overwritten. This also holds for the data supplied to `ggplot`. For example, we can create a second layer using another call to `geom_point` from a second data set (e.g., with a summary statistic), like so:

```
# create a small tibble with the means of both  
# variables of interest  
avocado_data_means <-  
  avocado_data %>%
```

```

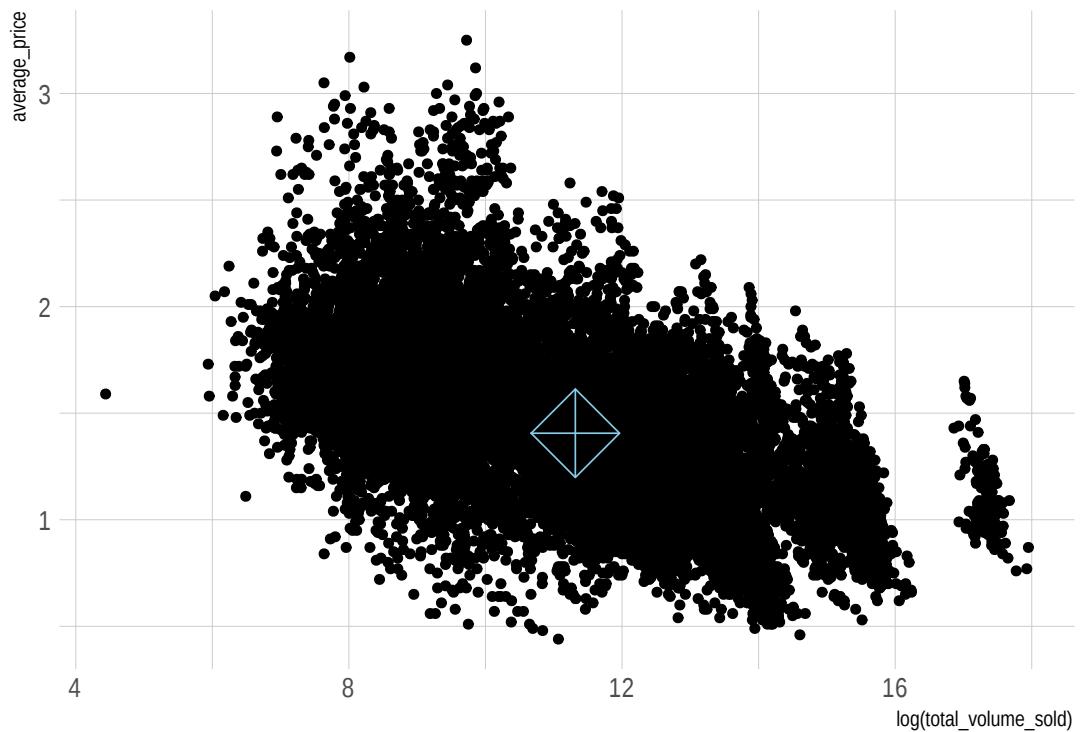
summarize(
  mean_volume = mean(log(total_volume_sold)),
  mean_price  = mean(average_price)
)
avocado_data_means

## # A tibble: 1 x 2
##   mean_volume mean_price
##       <dbl>      <dbl>
## 1        11.3      1.41

avocado_data %>%
  ggplot(
    aes(x = log(total_volume_sold),
        y = average_price)
  ) +
  # first layer uses globally declared data & mapping
  geom_point() +
  # second layer uses different data set & mapping
  geom_point(
    data = avocado_data_means,
    mapping = aes(
      x = mean_volume,
      y = mean_price
    ),
    # change shape of element to display (see below)
    shape = 9,
    # change size of element to display
    size = 12,
    color = "skyblue"
  )

```

## 6. Data Visualization

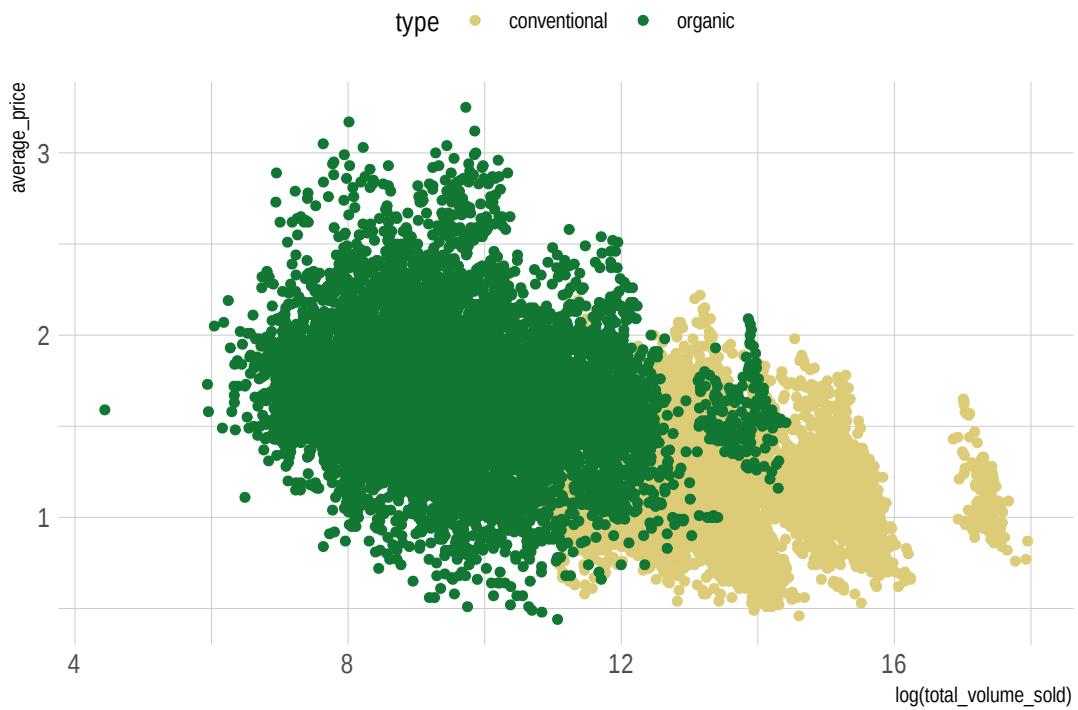


### 6.3.4. Grouping

Categorical distinction are frequently important in data analysis. Just think of the different combinations of factor levels in a factorial design, or the difference between conventionally grown and organically grown avocados. `ggplot` understands grouping very well and acts on appropriately, if you tell it to in the right way.

Grouping can be relevant for different aspects of a plot: the color of points or lines, their shape, or even whether to plot everything together or separately. For instance, we might want to display different types of avocados in different color. We can do this like so:

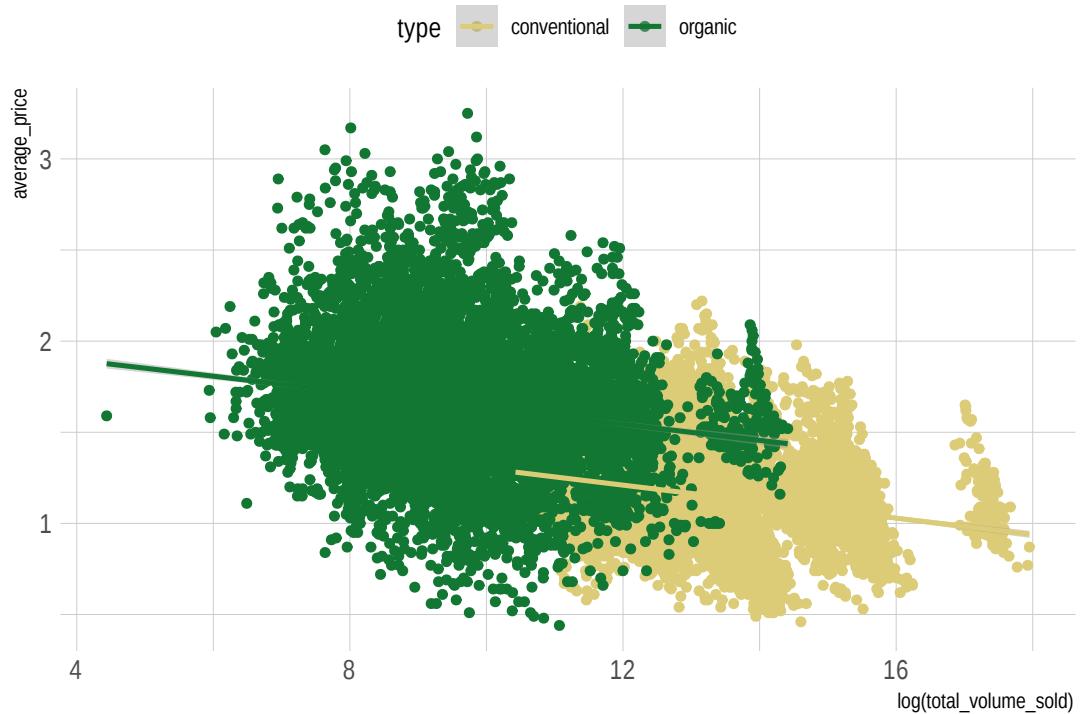
```
avocado_data %>%
  ggplot(
    aes(
      x = log(total_volume_sold),
      y = average_price,
      # use a different color for each type of avocado
      color = type
    )
  ) +
  geom_point(aes(color = type))
```



Notice that we added the grouping information inside of `aes` to the call of `ggplot`. This way the grouping is the global default for the whole plot. Check what happens when we then add another layer, like `geom_smooth`:

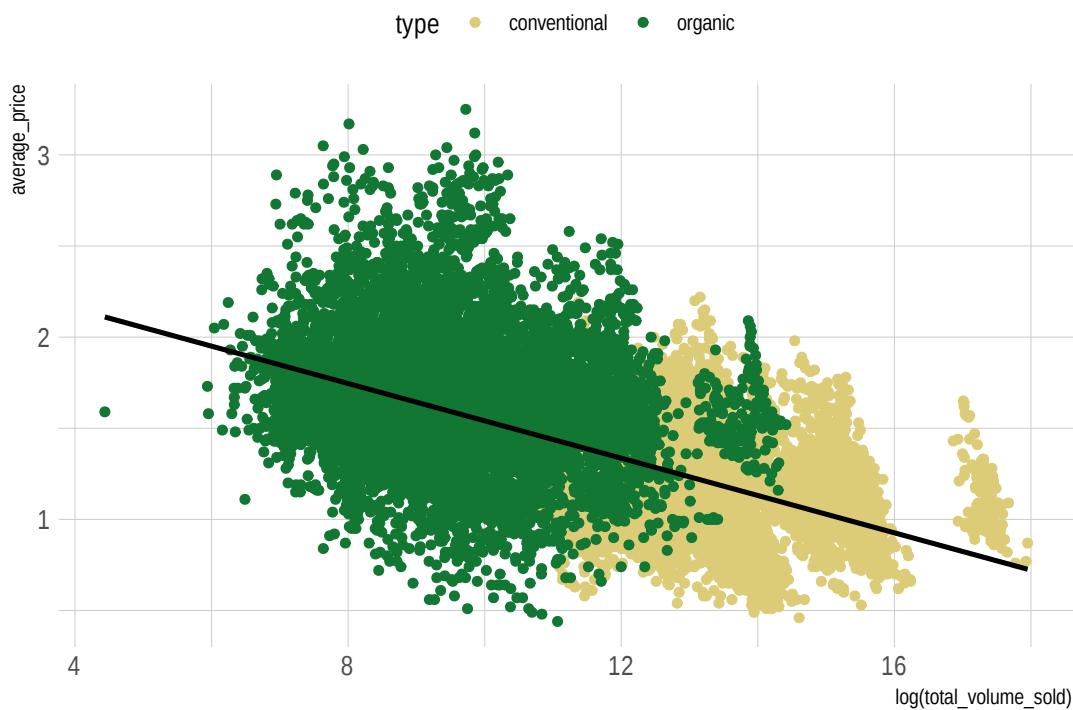
```
avocado_data %>%
  ggplot(
    aes(
      x = log(total_volume_sold),
      y = average_price,
      # use a different color for each type of avocado
      color = type
    )
  ) +
  geom_point(aes(color = type)) +
  geom_smooth(method = "lm")
```

## 6. Data Visualization



The regression lines will also be shown in the colors of the underlying scatter plot. We can change this by overwriting the `color` attribute locally, but then we loose the grouping information:

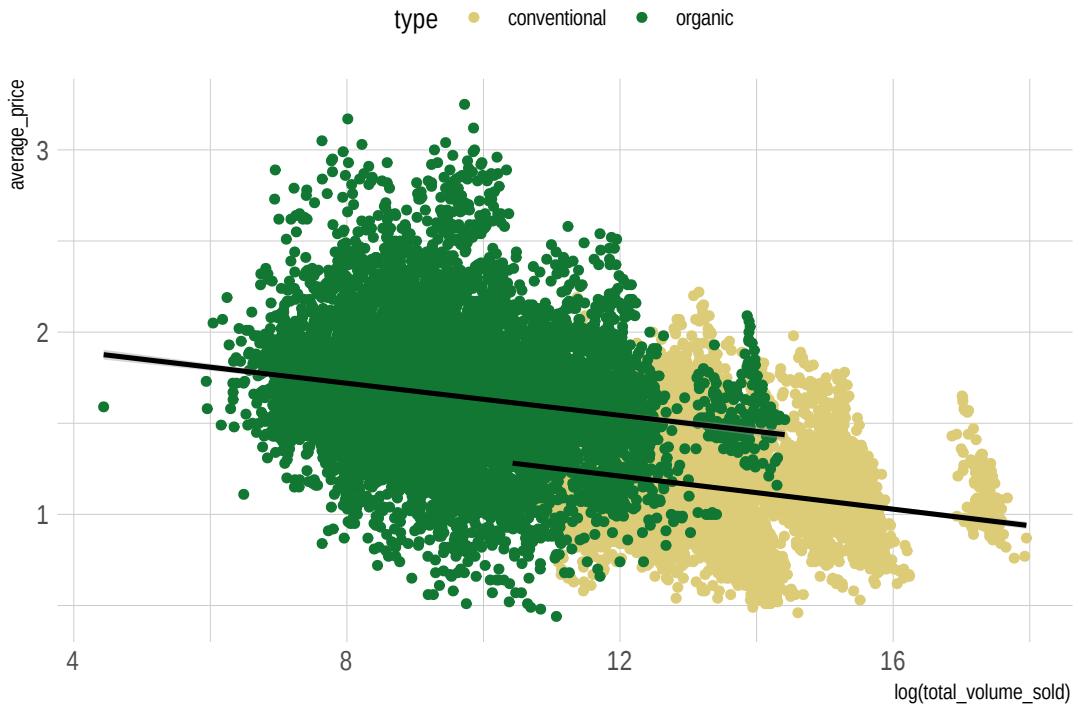
```
avocado_data %>%
  ggplot(
    aes(
      x = log(total_volume_sold),
      y = average_price,
      # use a different color for each type of avocado
      color = type
    )
  ) +
  geom_point(aes(color = type)) +
  geom_smooth(method = "lm", color = "black")
```



To retrieve the grouping information, we can change the explicit keyword `group` (which just treats data from the relevant factor levels differently without directly changing their appearance):

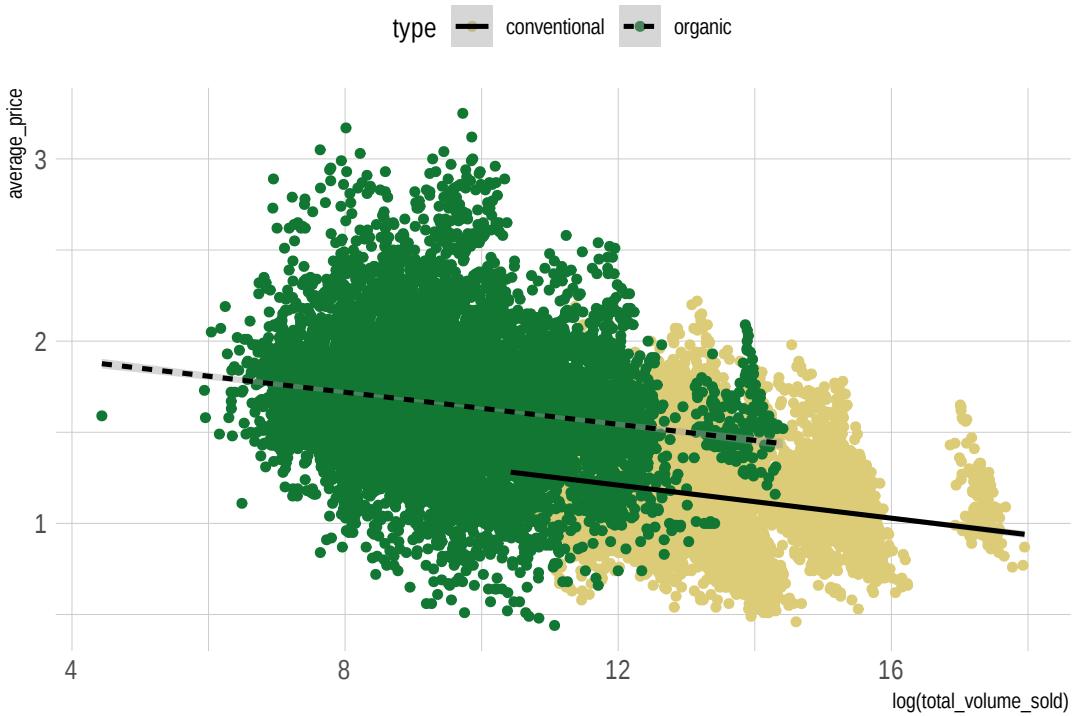
```
avocado_data %>%
  ggplot(
    aes(
      x = log(total_volume_sold),
      y = average_price,
      # use a different color for each type of avocado
      color = type
    )
  ) +
  geom_point(aes(color = type)) +
  geom_smooth(
    # tell the smoother to deal with avocados types separately
    aes(group = type),
    method = "lm",
    color = "black"
  )
```

## 6. Data Visualization



Finally, we see that the lines are not uniquely associative with the avocado type, so we can also change the regression line's `shape` attribute conditional on avocado type:

```
avocado_data %>%
  ggplot(
    aes(
      x = log(total_volume_sold),
      y = average_price,
      # use a different color for each type of avocado
      color = type
    )
  ) +
  geom_point(aes(color = type)) +
  geom_smooth(
    # tell the smoother to deal with avocados types separately
    aes(group = type, linetype = type),
    method = "lm",
    color = "black"
  )
```



### 6.3.5. Example of a customized plot

If done with the proper mind and heart, plots intended to share (and to communicate a point, following the idea of hypothesis-driven visualization) will usually require a lot of tweaking. We will cover some of the most frequently relevant tweaks in Section 6.6.

To nevertheless get a feeling of where the journey is going, at least roughly, here is an example of a plot of the avocado data which is much more tweaked and honed. No claim is intended regarding the false idea that this plot is in any sense optimal. There is not even a clear hypothesis or point to communicate. This just showcases some functionality. Notice, for instance, that this plot uses two layers, invoked by `geom_point` which shows the scatter plot of points and `geom_smooth` which layers on top the point cloud regression lines (one for each level in the grouping variable).

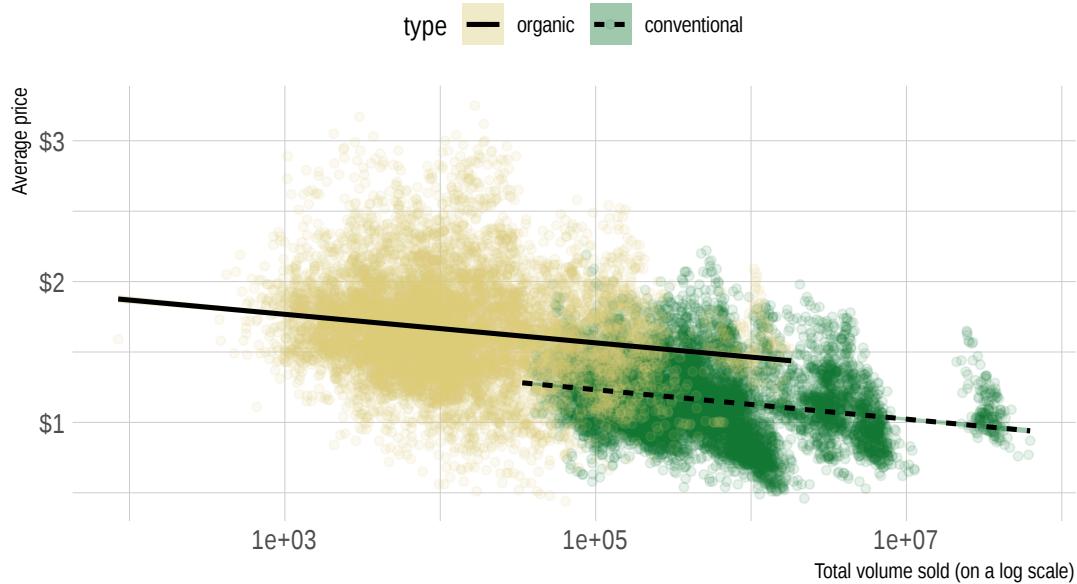
```
# pipe data set into function `ggplot`  
avocado_data %>%  
  # reverse factor level so that horizontal legend entries align with  
  # the majority of observations of each group in the plot  
  mutate(  
    type = fct_rev(type)  
  ) %>%
```

## 6. Data Visualization

```
# initialize the plot
ggplot(
  # defined mapping
  mapping = aes(
    # which variable goes on the x-axis
    x = total_volume_sold,
    # which variable goes on the y-axis
    y = average_price,
    # which groups of variables to distinguish
    group = type,
    # color and fill to change by grouping variable
    fill = type,
    linetype = type,
    color = type
  )
) +
# declare that we want a scatter plot
geom_point(
  # set low opacity for each point
  alpha = 0.1
) +
# add a linear model fit (for each group)
geom_smooth(
  color = "black",
  method = "lm"
) +
# change the default (normal) of x-axis to log-scale
scale_x_log10() +
# add dollar signs to y-axis labels
scale_y_continuous(labels = scales::dollar) +
# change axis labels and plot title & subtitle
labs(
  x = 'Total volume sold (on a log scale)',
  y = 'Average price',
  title = "Avocado prices against amount sold",
  subtitle = "With linear regression lines"
)
```

## Avocado prices against amount sold

With linear regression lines



## 6.4. A rendezvous with popular geoms

In the following we will cover some of the more basic `geom_` functions relevant for our present purposes. It might be useful to read this section top-to-bottom at least once, not to think of it as a mere reference list. More information is provided by the ggplot cheat sheet.

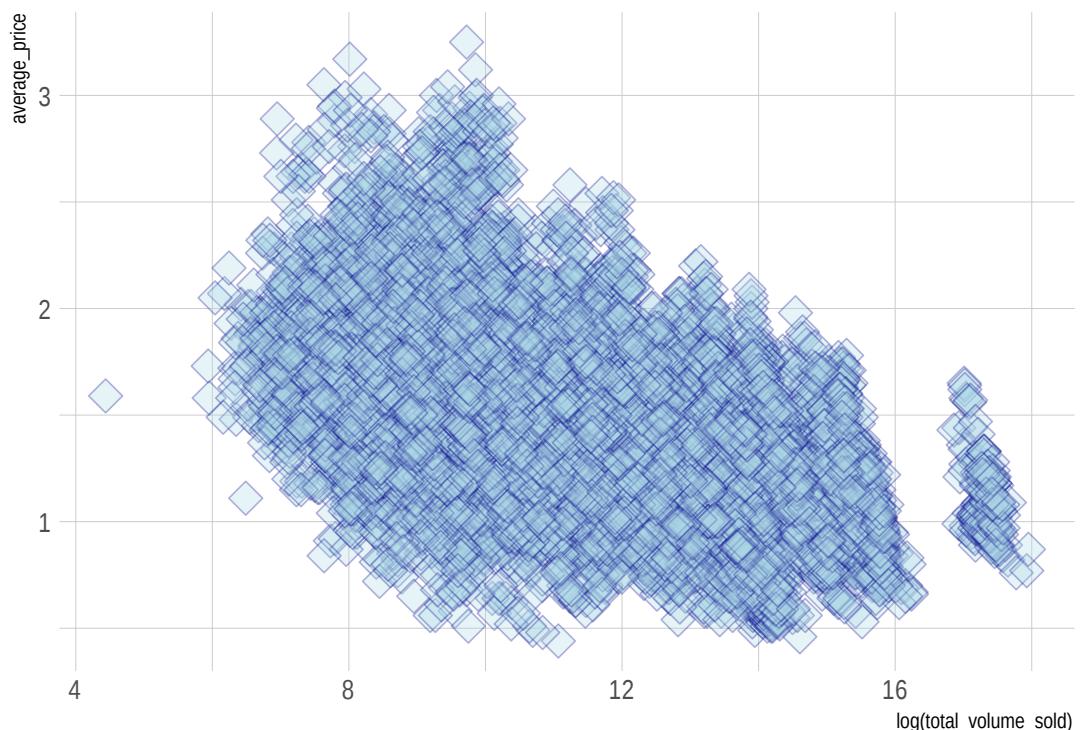
### 6.4.1. Scatter plots with `geom_point`

Scatter plots visualize pairs of associated observations as points in space. We have seen this for the avocado prize data above. Let's look at some of the further arguments we can use to tweak the presentation by `geom_point`. The following example changes the shape of the objects displayed to tilted rectangles (sometimes called diamonds, e.g., in LaTeX \diamond) away from the default circles, the color of the shapes, their size and opacity.

```
avocado_data %>%
  ggplot(aes(x = log(total_volume_sold), y = average_price)) +
  geom_point()
```

## 6. Data Visualization

```
# shape to display is number 23 (tilted rectangle, see below)
shape = 23,
# color of the surrounding line of the shape (for shapes 21-24)
color = "darkblue",
# color of the interior of each shape
fill = "lightblue",
# size of each shape (default is 1)
size = 5,
# level of opacity for each shape
alpha = 0.3
)
```



How do you know which shape is which number? - By looking at the picture in Figure 6.5, for instance.

### 6.4.2. Smooth

The `geom_smooth` function operates on two-dimensional metric data and outputs a smoothed line, using different kinds of fitting functions. It is possible to show an indicator of certainty for the fit. We will deal with model fits in later parts of the book. For illustration just enjoy a few examples here:

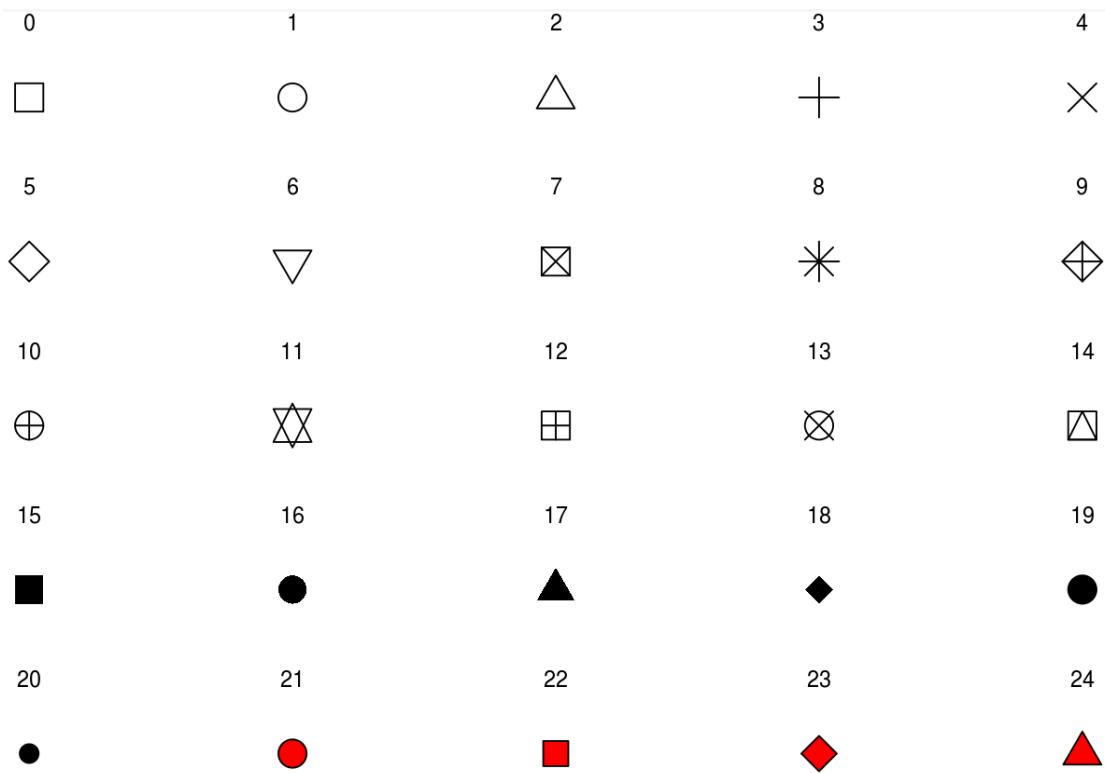
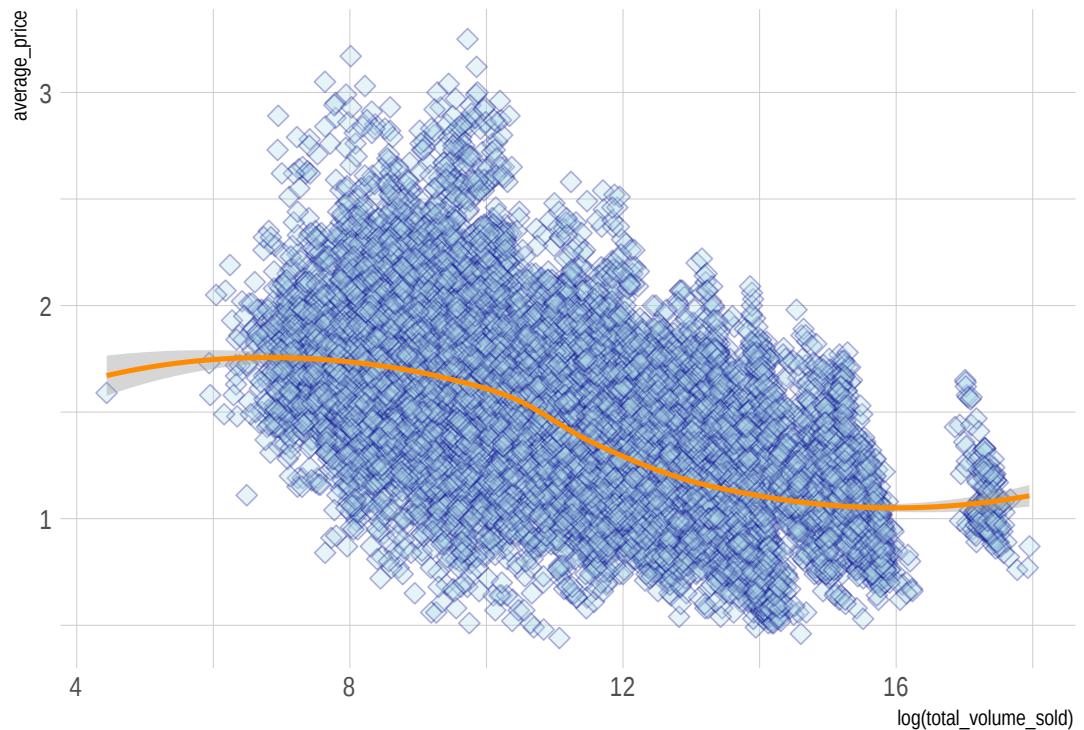


Figure 6.5.: The numerical coding of different shapes in ‘ggplot’ Notice that objects 21-24 are sensitive to both ‘color’ and ‘fill’, but the others are only sensitive to ‘color’.

## 6. Data Visualization

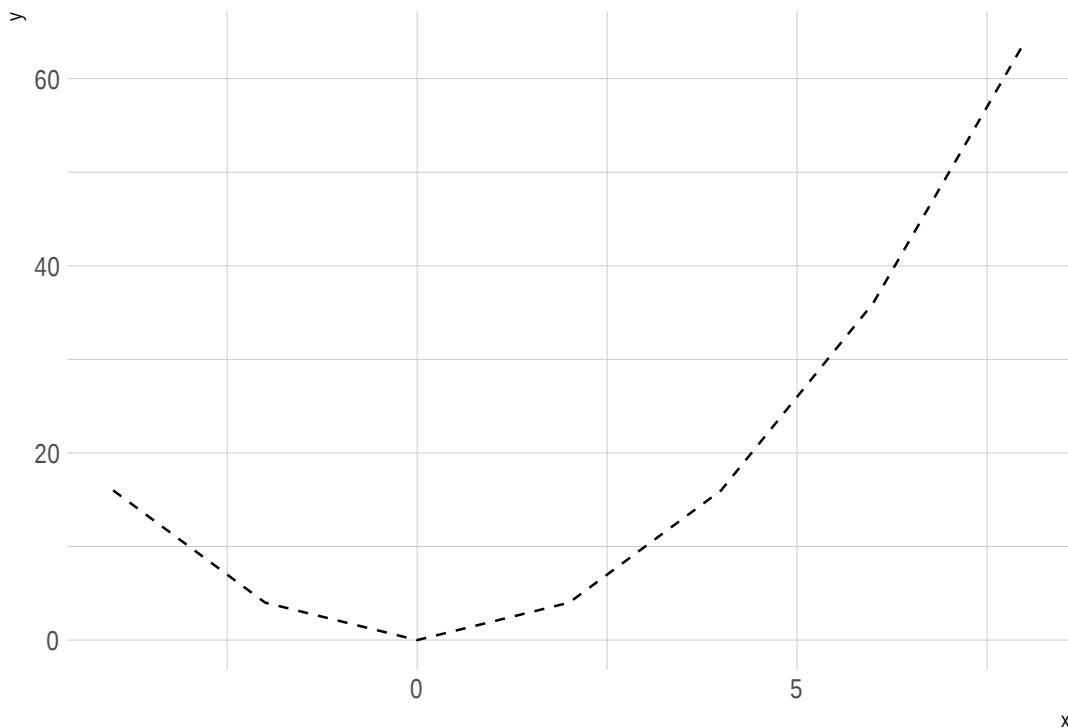
```
avocado_data %>%
  ggplot(aes(x = log(total_volume_sold), y = average_price)) +
  geom_point(
    shape = 23,
    color = "darkblue",
    fill = "lightblue",
    size = 3,
    alpha = 0.3
  ) +
  geom_smooth(
    # fitting a smoothed curve to the data
    method = "loess",
    # display standard error around smoothing curve
    se = T,
    color = "darkorange"
  )
```



### 6.4.3. Line

Use `geom_line` to display a line for your data if that data has associated (ordered) matrix values. You can use argument `linetype` to specify the kind of line to draw.

```
tibble(
  x = seq(-4,8, by = 2),
  y = x^2
) %>%
  ggplot(aes(x,y)) +
  geom_line(
    linetype = "dashed"
)
```

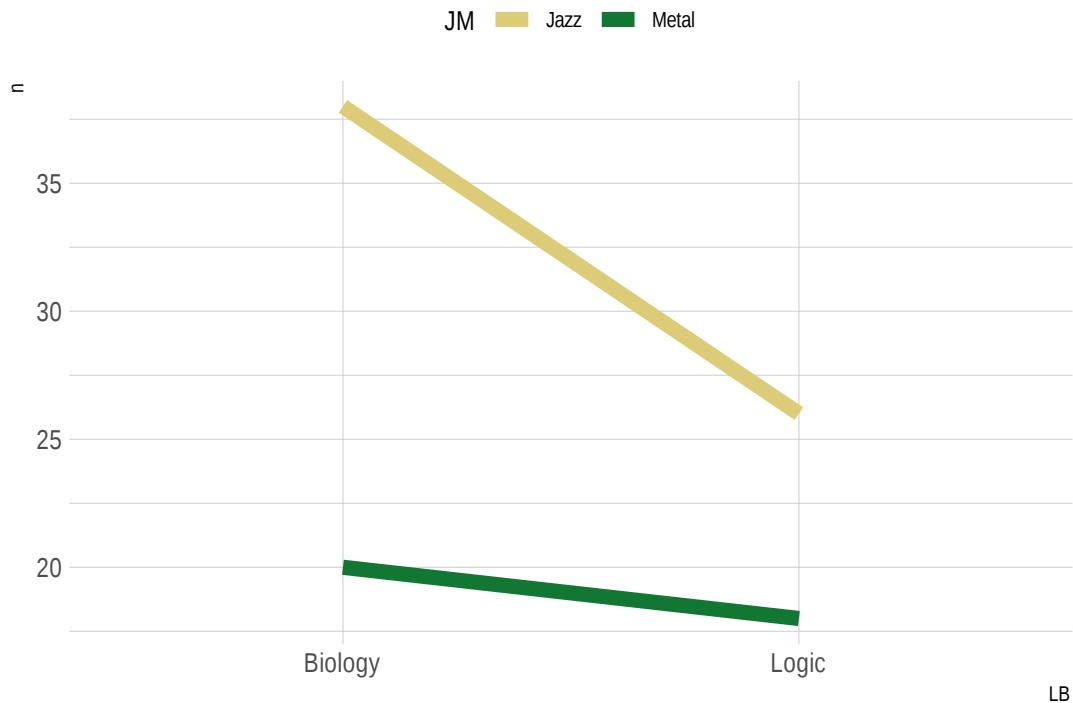


Sometimes you may want to draw lines between items that are grouped:

```
BLJM_associated_counts %>%
  ggplot(
    aes(
      x = LB,
      y = n,
```

## 6. Data Visualization

```
      color = JM,  
      group = JM  
    )  
  ) +  
geom_line(size = 3)
```



### 6.4.4. Barplot

A barplot, plotted with `geom_bar` or `geom_col`, displays a single number for each of several groups for visual comparison by length. The difference between these two functions is that `geom_bar` relies on an implicit counting, while `geom_col` expects the numbers that translate into the lenght of the bars to be supplied for it. This book favors the use of `geom_col` by first wrangling the data to show the numbers to be visualized, since often this is the cleaner approach and the numbers are useful to have access to independently (e.g., for referring to in the text).

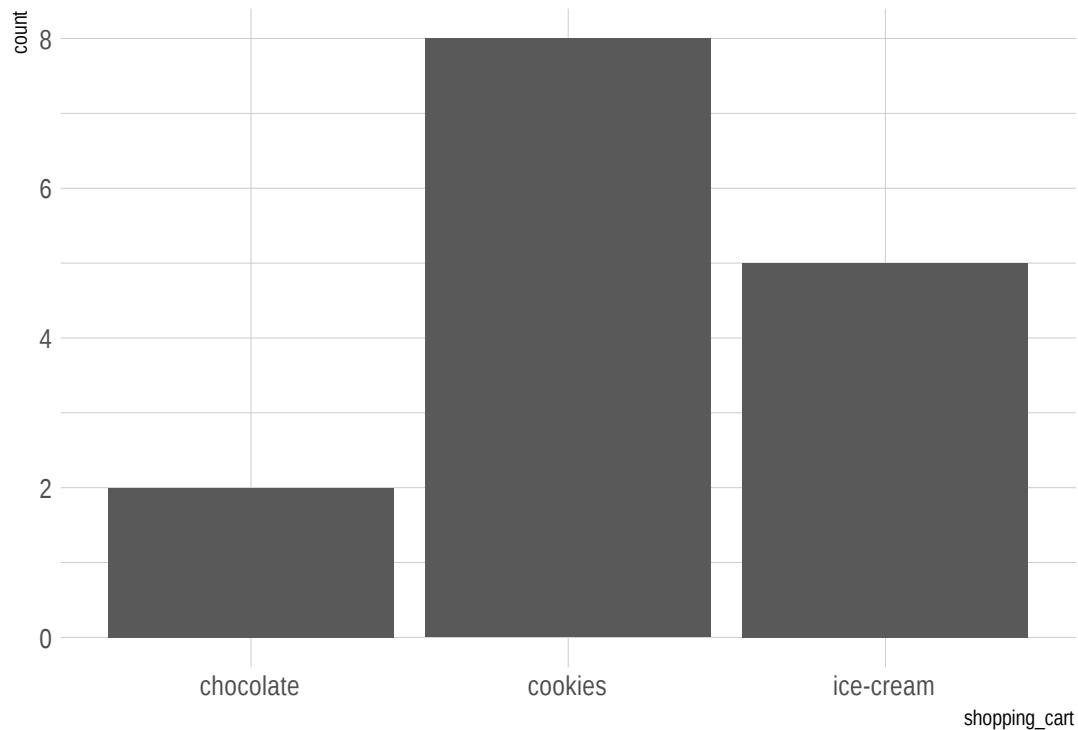
Here's an example of how `bar_plot` works (implicitly counting numbers of occurrences):

```
tibble(  
  shopping_cart = c(
```

```

rep("chocolate", 2),
rep("ice-cream", 5),
rep("cookies", 8)
)
) %>%
ggplot(aes(x = shopping_cart)) +
geom_bar()

```



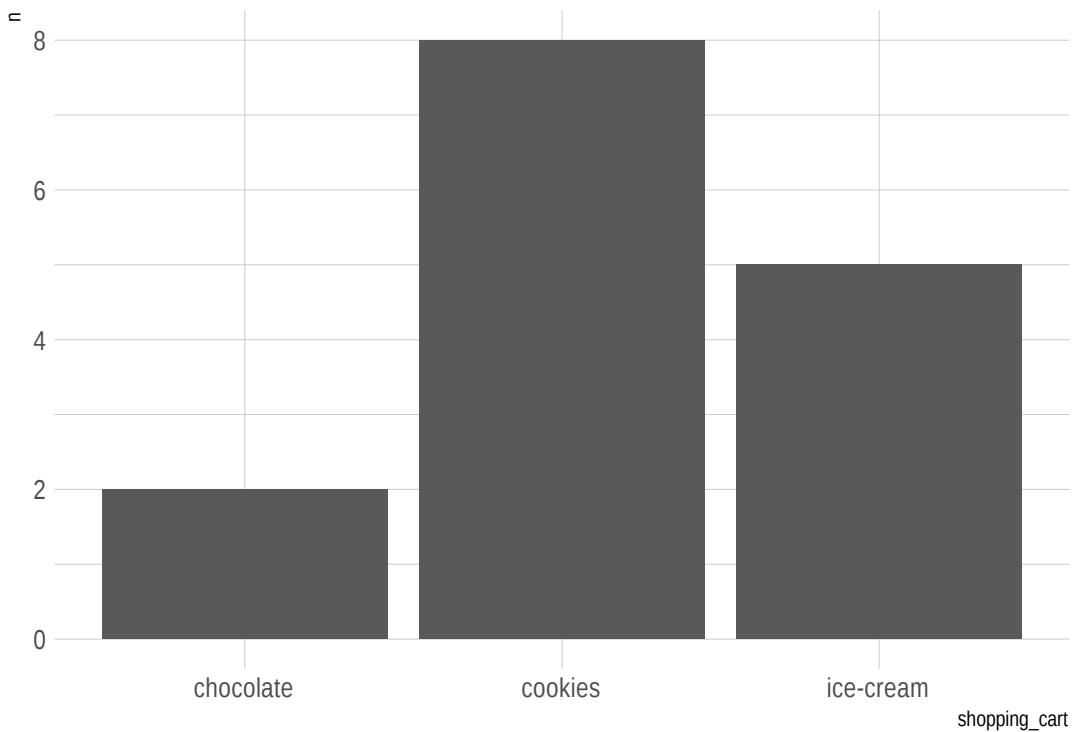
To display this data with `geom_col` we need to count occurrences first ourselves:

```

tibble(
  shopping_cart = c(
    rep("chocolate", 2),
    rep("ice-cream", 5),
    rep("cookies", 8)
  )
) %>%
  dplyr::count(shopping_cart) %>%
  ggplot(aes (x = shopping_cart, y = n) ) +
  geom_col()

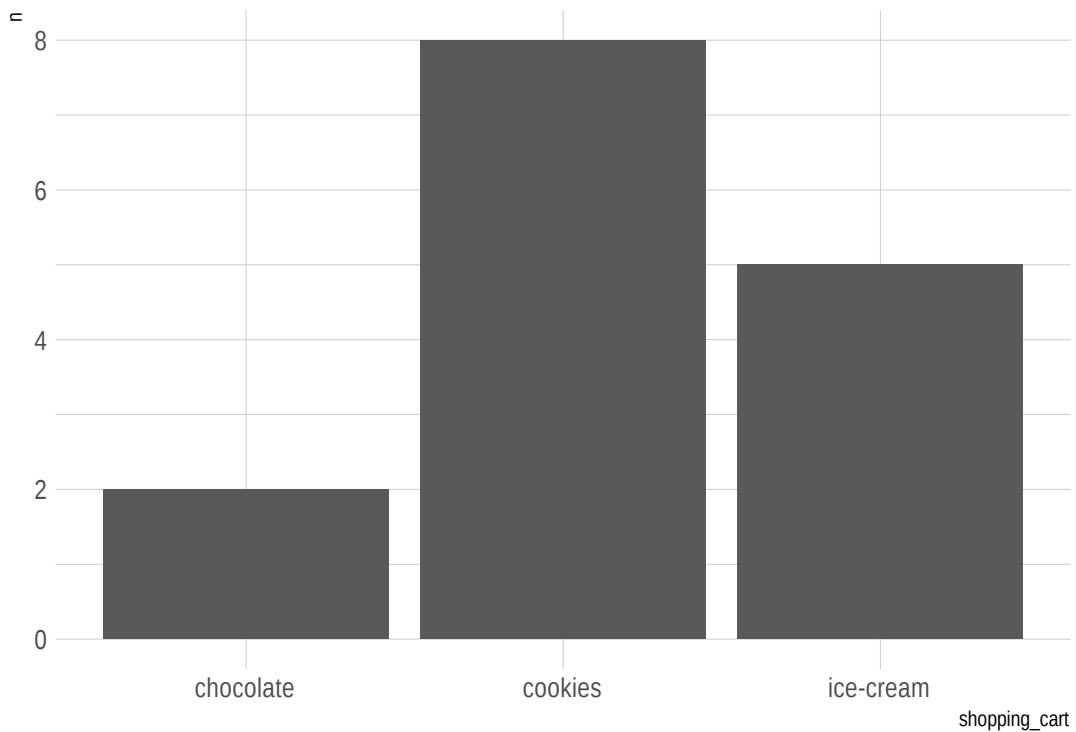
```

## 6. Data Visualization



To be clear, `geom_col` is essentially `geom_bar` when we overwrite the default statistical transformation of counting to “identity”:

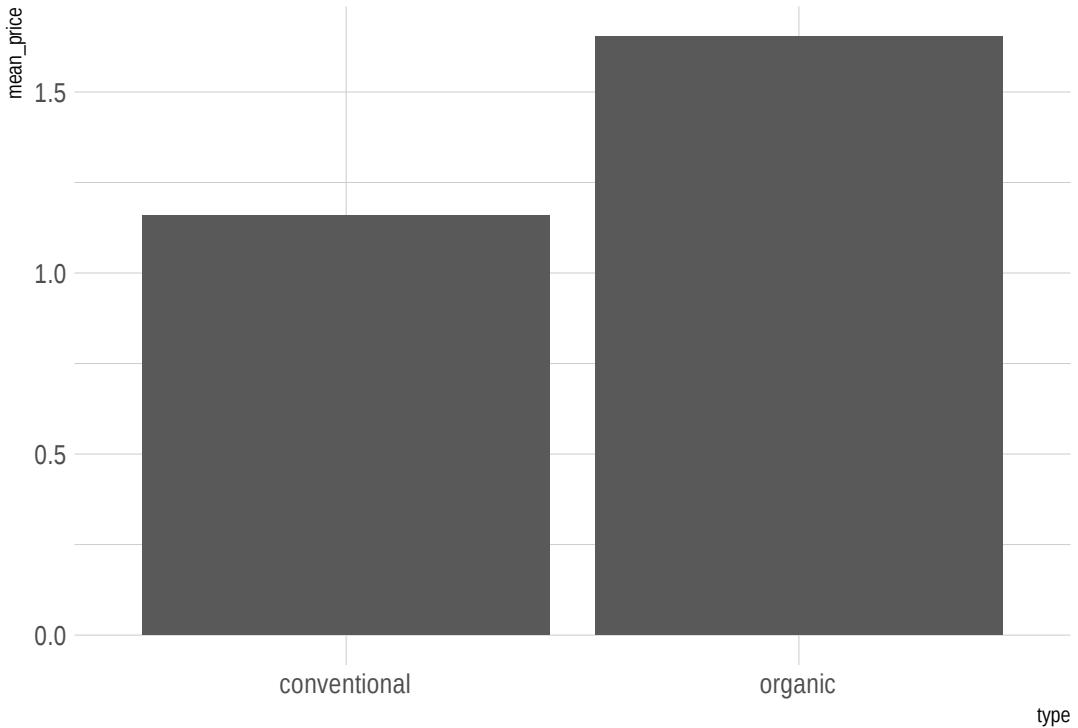
```
tibble(
  shopping_cart = c(
    rep("chocolate", 2),
    rep("ice-cream", 5),
    rep("cookies", 8)
  )
) %>%
  dplyr::count(shopping_cart) %>%
  ggplot(aes(x = shopping_cart, y = n)) +
  geom_bar(stat = "identity")
```



Barplots are a frequent sight in psychology papers. They are also controversial. They often fare badly with respect to the data-ink ratio. Especially, when what is plotted are means of grouped variables. For example, the following plot is rather uninformative (even if the research question is a comparison of means):

```
avocado_data %>%
  group_by(type) %>%
  summarise(
    mean_price = mean(average_price)
  ) %>%
  ggplot(aes(x = type, y = mean_price)) +
  geom_col()
```

## 6. Data Visualization



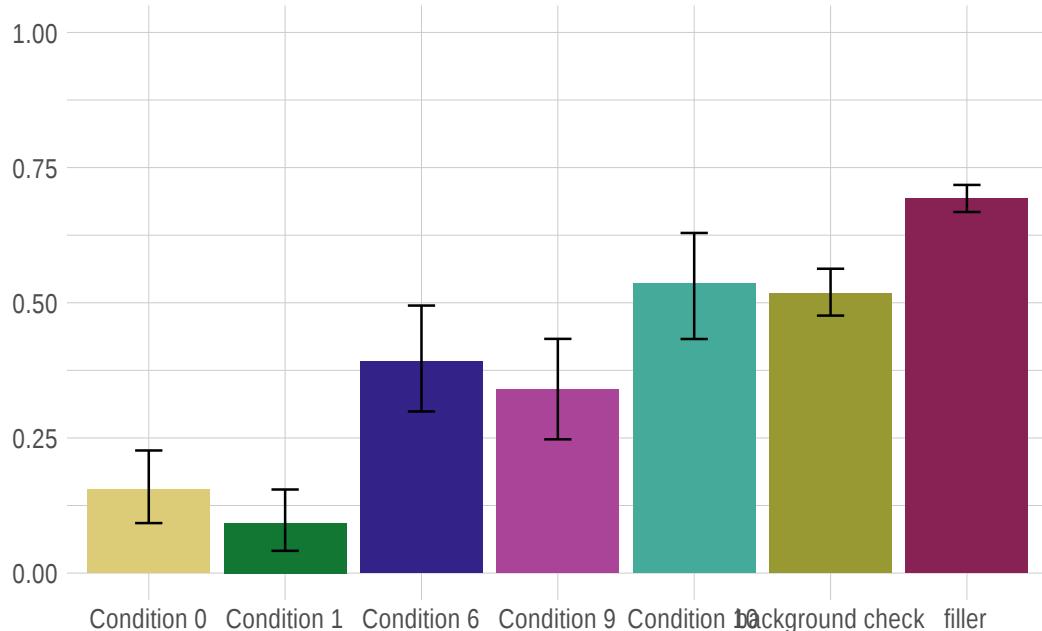
It makes sense to use the available space for a more informative report about the distribution of data points around the means, e.g., by using `geom_violin` or `geom_histogram` etc.

But barplots may also be good enough if there is not more of immediate relevance, such as when we look at counts or proportions. Still, it might help to include a measure of certainty. For instance, using the King of France data set, we can display proportions of ‘true’ answers with 95% bootstrapped confidence intervals like in the plot below. Notice the use of the `geom_errorbar` function to display the intervals in the following example.

```
data_KoF_processed %>%
  # drop unused factor levels
  droplevels() %>%
  # get means and 95% bootstrapped CIs for each condition
  group_by(condition) %>%
  nest() %>%
  summarise(
    CIs = map(data, function(d) bootstrapped_CI(d$response == "TRUE"))
  ) %>%
  unnest(CIs) %>%
  # plot means and CIs
  ggplot(aes(x = condition, y = mean, fill = condition)) +
```

```
geom_col() +
  geom_errorbar(aes(ymin = lower, ymax = upper, width = 0.2)) +
  ylim(0,1) +
  ylab("") + xlab("") + ggtitle("Proportion of 'TRUE' responses per condition") +
  theme(legend.position = "none") +
  scale_fill_manual(values = project_colors)
```

## Proportion of 'TRUE' responses per condition



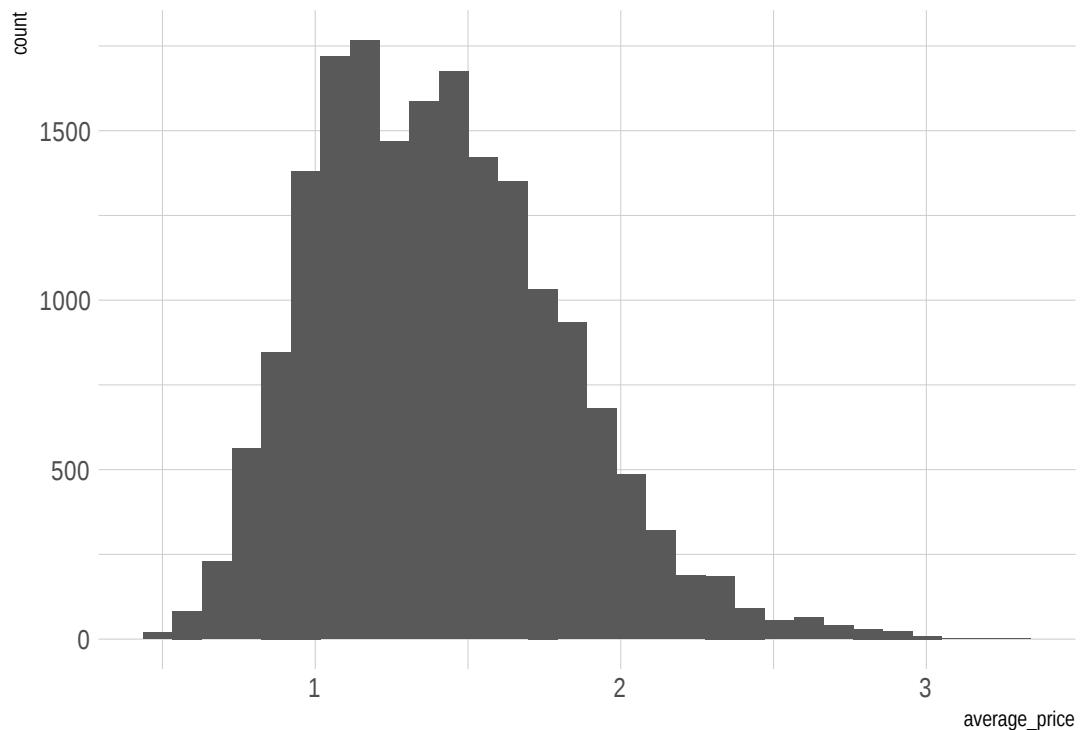
### 6.4.5. Plotting distributions: histograms, boxplots, densities and violins

There are different ways for plotting the distribution of observations in a one-dimensional vector, each with its own advantages and disadvantages: the histogram, a box plot, a density plot, and a violin plot. Let's have a look at each, based on the `average_price` of different types of avocados.

The histogram displays the number of occurrences of observations inside of prespecified bins. By default the function `geom_histogram` uses 30 equally spaced bins to display counts of your observations.

```
avocado_data %>%
  ggplot(aes(x = average_price)) +
  geom_histogram()
```

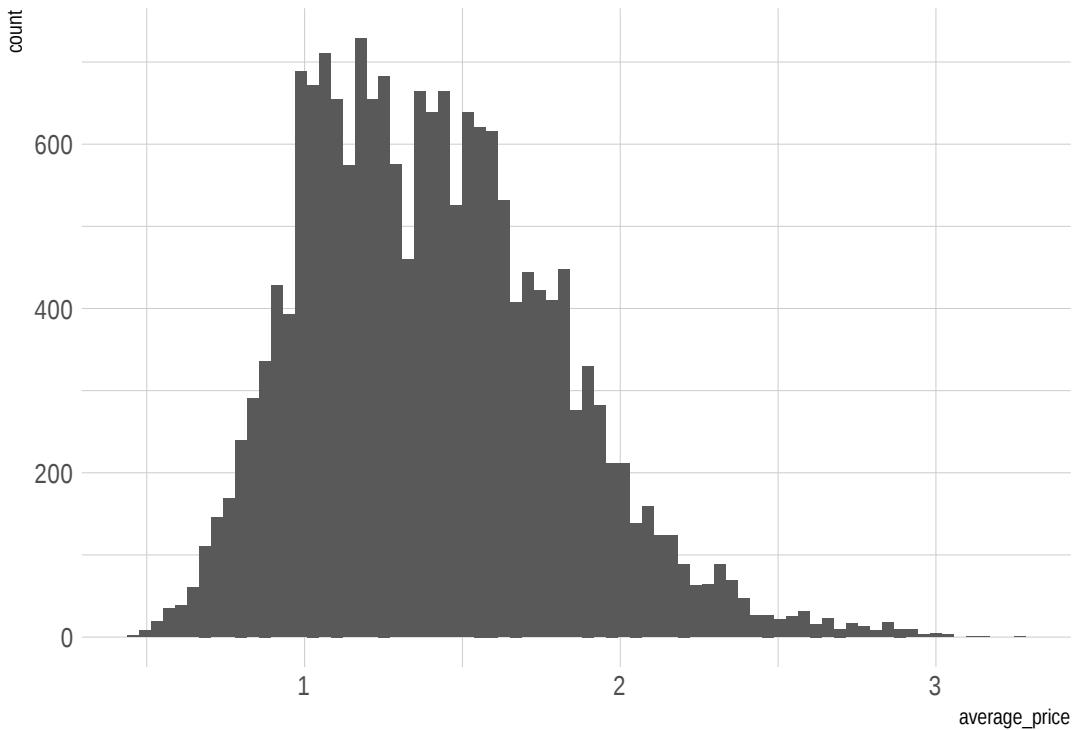
## 6. Data Visualization



If we specify more bins, we get a more fine-grained picture. (But notice that such a high number of bins works for the present data set, which has many observations, but it would not necessarily work for a small data set.)

```
avocado_data %>%
  ggplot(aes(x = average_price)) +
  geom_histogram(bins = 75)
```

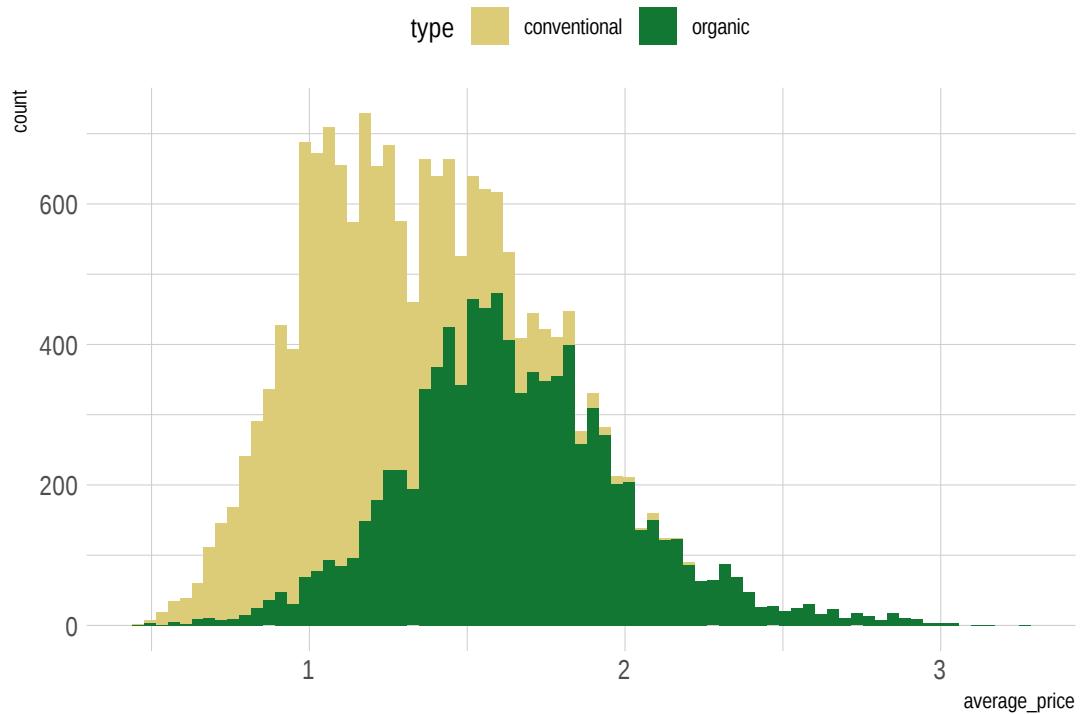
#### 6.4. A rendezvous with popular geoms



We can also layer histograms but this is usually a bad idea (even if we tinker with opacity) because a higher layer might block important information from a lower layer:

```
avocado_data %>%
  ggplot(aes(x = average_price, fill = type)) +
  geom_histogram(bins = 75)
```

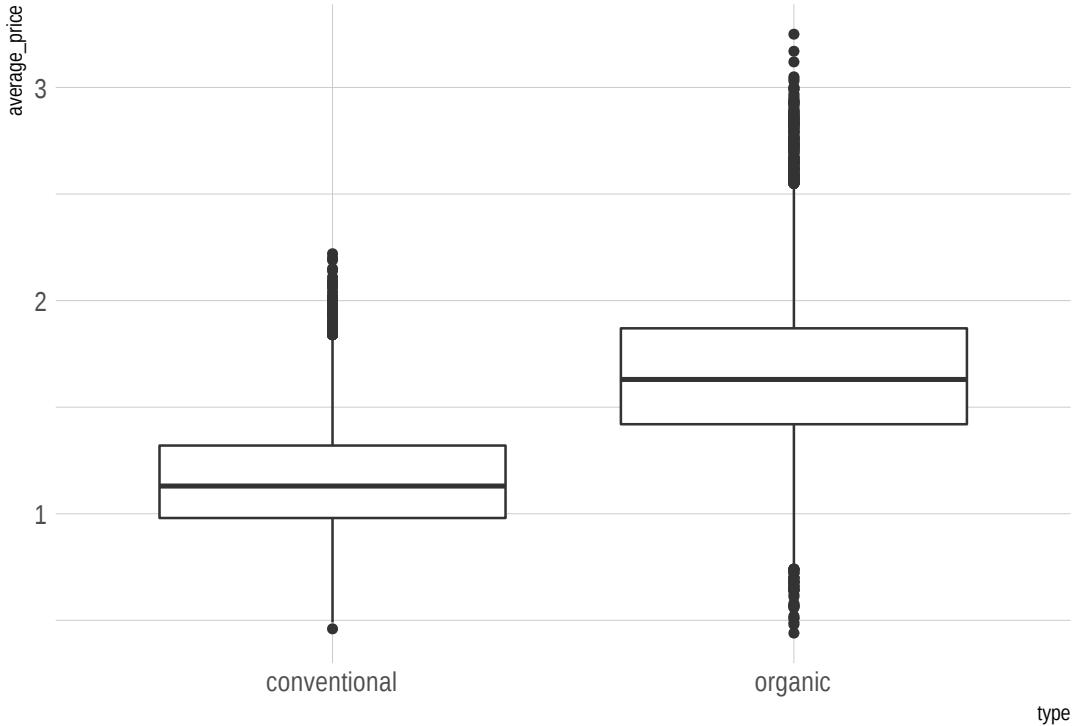
## 6. Data Visualization



An alternative display of distributional metric information is a **box plot**. Box plots are classics, also called *box-and-whiskers plots*, and they basically visually report key summary statistics of your metric data. These do work much better than histograms for direct comparison:

```
avocado_data %>%
  ggplot(aes(x = type , y = average_price)) +
  geom_boxplot()
```

#### 6.4. A rendezvous with popular geoms

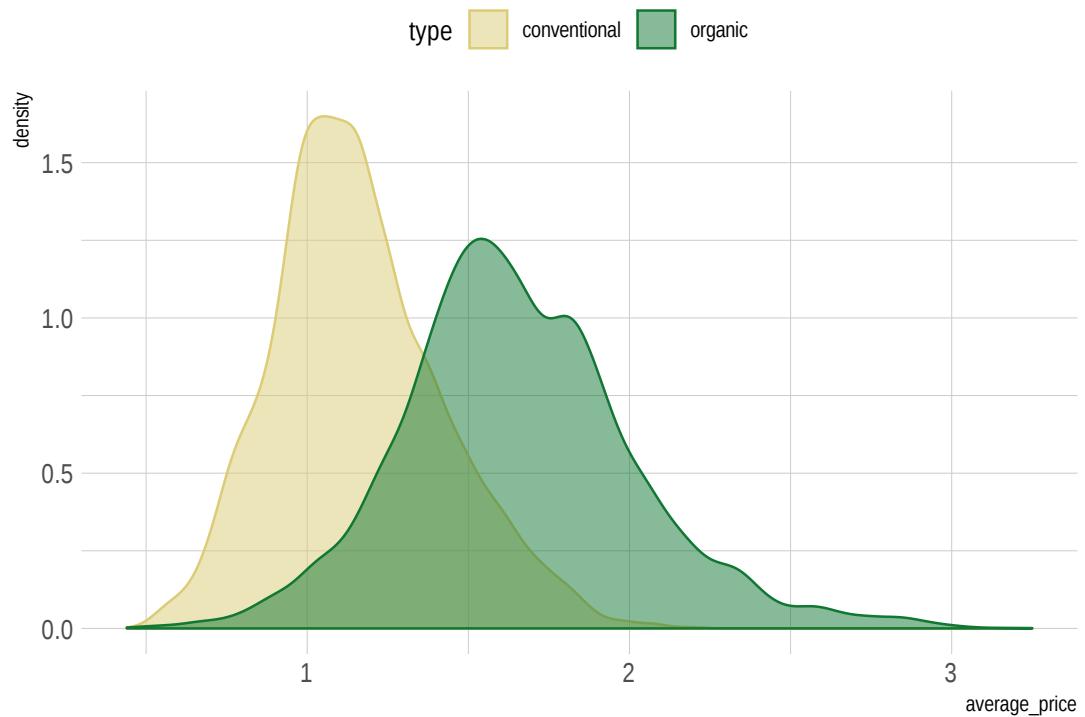


What we see here is the median for each group (thick black line) and the 25% and 75% quantiles (boxes). The straight lines show the range from the 25% or 75% quantiles to the values given by  $\text{median} + 1.58 * \text{IQR} / \sqrt{n}$ , where the IQR is the “interquartile range”, i.e., the range between the 25% and 75% quantiles (boxes).

To get a better picture of the shape of the distribution, `geom_density` uses a kernel estimate to show a smoothed line, roughly indicating ranges of higher density of observations with higher numbers. Using opacity, `geom_density` is useful also for the close comparison of distributions across different groups:

```
avocado_data %>%
  ggplot(aes(x = average_price, color = type, fill = type)) +
  geom_density(alpha = 0.5)
```

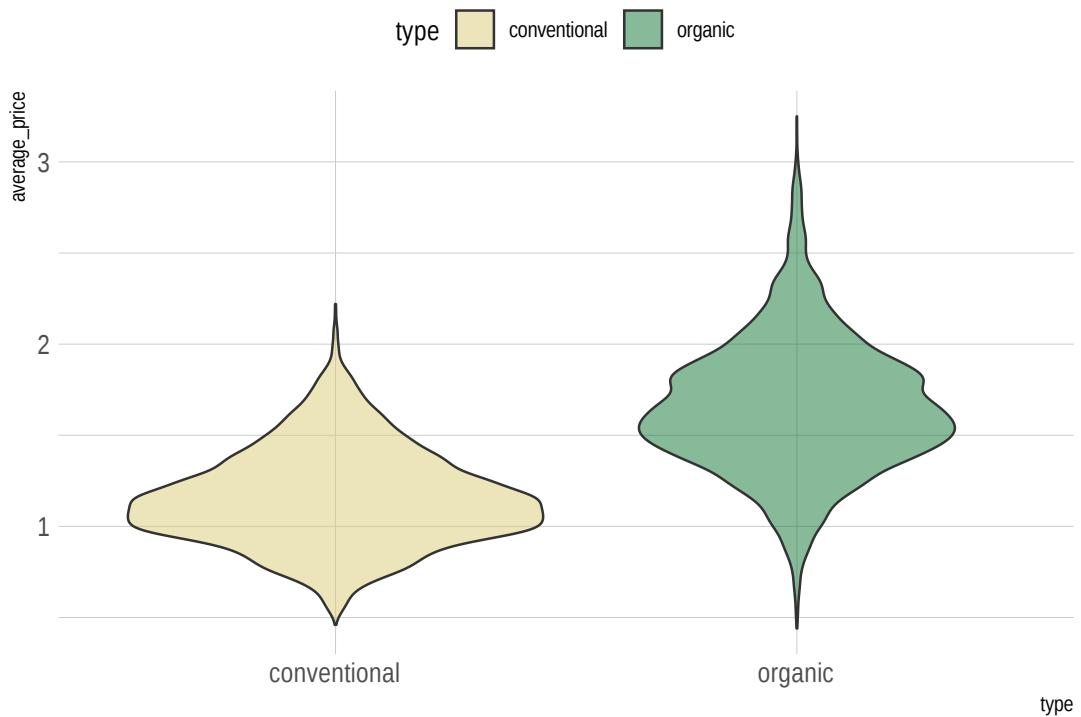
## 6. Data Visualization



For many groups to compare, density plots can become cluttered. **Violin plots** are like mirrored density plots and are better for comparison of multiple groups:

```
avocado_data %>%
  ggplot(aes(x = type, y= average_price, fill = type)) +
  geom_violin(alpha = 0.5)
```

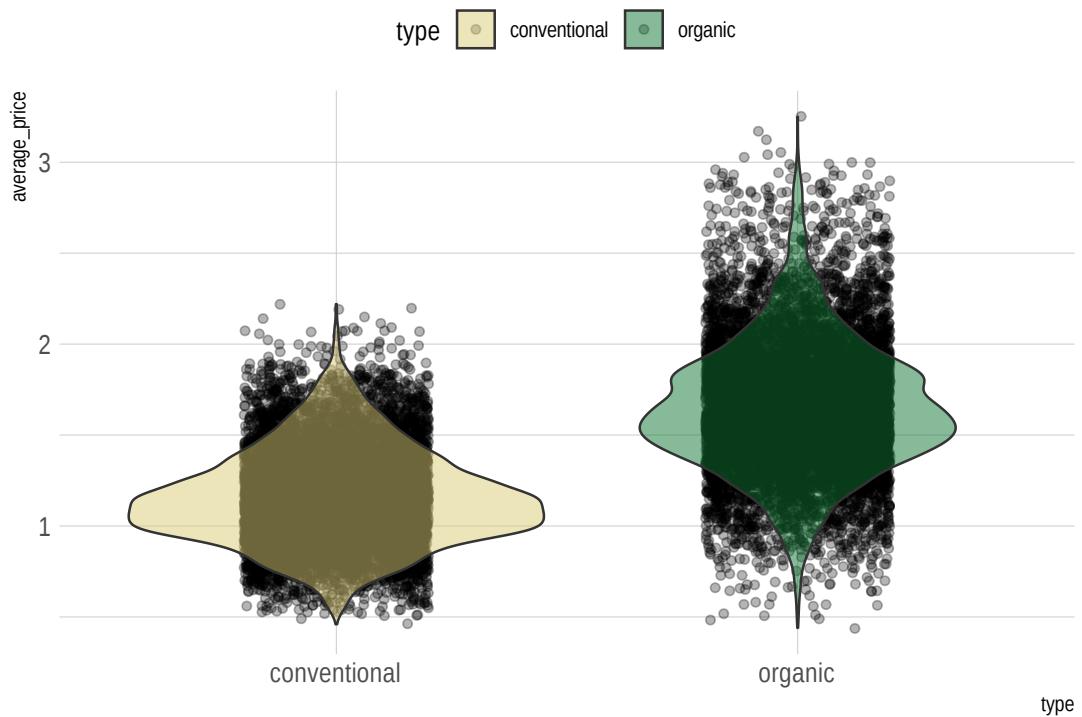
#### 6.4. A rendezvous with popular geoms



A frequently seen method of visualization is to layer a jittered distribution of points under a violin plot, like so:

```
avocado_data %>%
  ggplot(aes(x = type, y= average_price, fill = type)) +
  geom_jitter(alpha = 0.3, width = 0.2) +
  geom_violin(alpha = 0.5)
```

## 6. Data Visualization



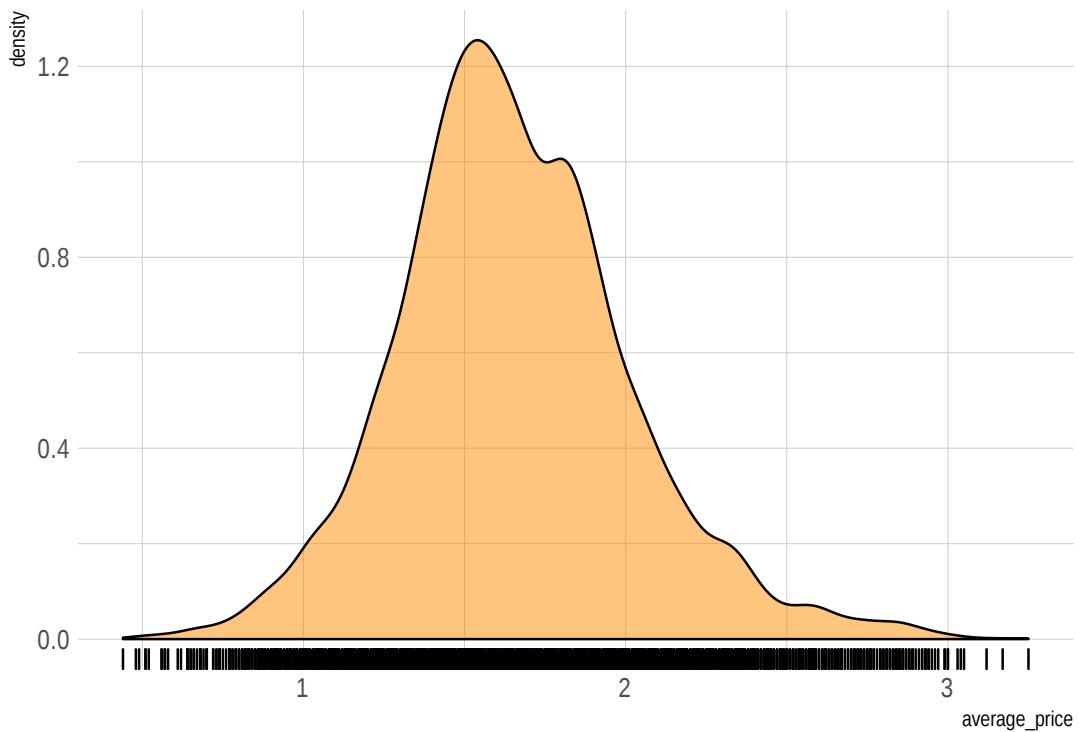
### 6.4.6. Rugs

Since plotting distributions, especially with high-level abstract smoothing as in `geom_density` and `geom_violin` fails to give information about the actual quantity of the data points, rug plots are useful additions to such plots. `geom_rug` add marks along the axes where different points lie.

Here is an example of `geom_rug` combined with `geom_density`:

```
avocado_data %>%
  filter(type == "organic") %>%
  ggplot(aes(x = average_price)) +
  geom_density(fill = "darkorange", alpha = 0.5) +
  geom_rug()
```

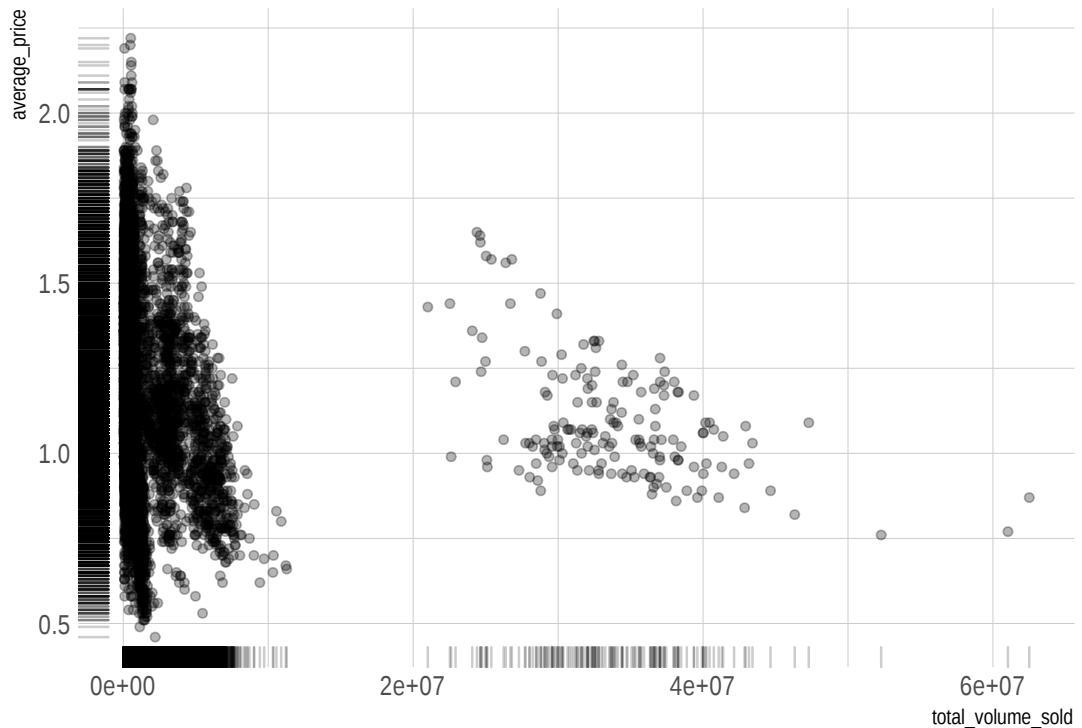
#### 6.4. A rendezvous with popular geoms



Here are rugs on a two-dimensional scatter plot:

```
avocado_data %>%
  filter(type == "conventional") %>%
  ggplot(aes(x = total_volume_sold, y = average_price)) +
  geom_point(alpha = 0.3) +
  geom_rug(alpha = 0.2)
```

## 6. Data Visualization



### 6.4.7. Annotation

It can be useful add further elements to a plot. We might want to add text, or specific geometrical shapes to highlight aspects of data. The most general function for doing this is `annotate`. The function `annotate` takes as a first argument a `geom` argument, e.g., `text` or `rectangle`. It is therefore not a wrapper function in the `geom_` family of functions, but the underlying function around which convenience functions like `geom_text` or `geom_rectangle` are wrapped. The further arguments that `annotate` expects depend on the geom it is supposed to realize.

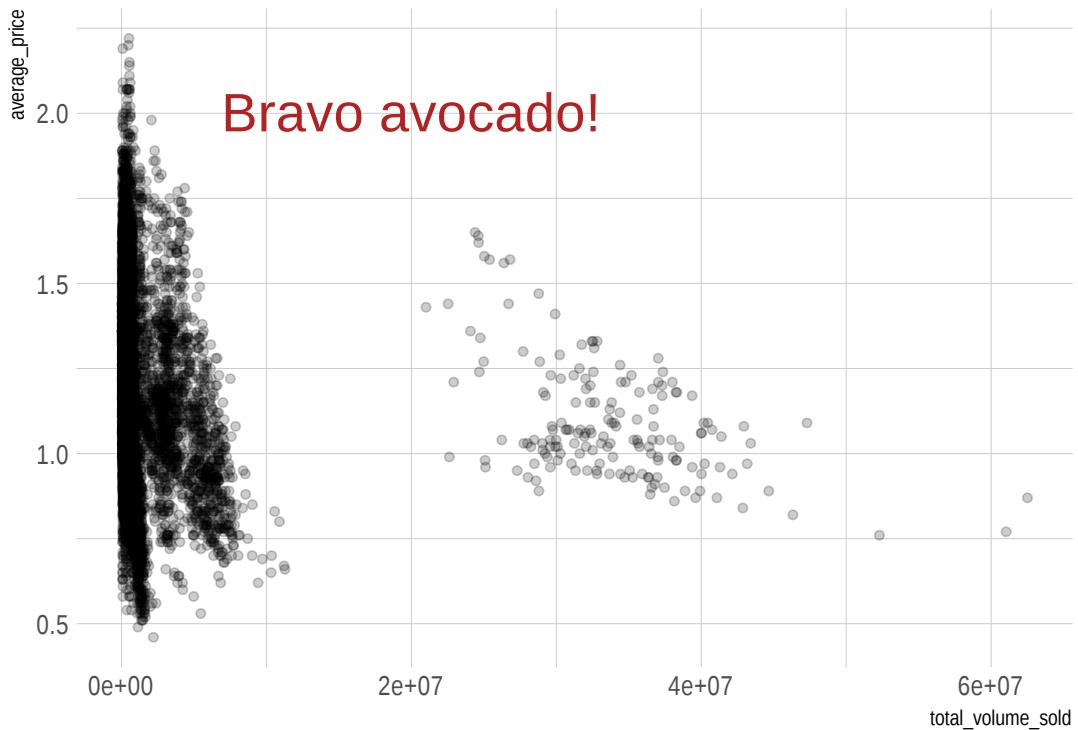
Suppose we want to add textual information at a particular coordinate. We can do this with `annotate` as follows:

```
avocado_data %>%
  filter(type == "conventional") %>%
  ggplot(aes(x = total_volume_sold, y = average_price)) +
  geom_point(alpha = 0.2) +
  annotate(
    geom = "text",
    # x and y coordinates for the text
    x = 2e7,
```

```

y = 2,
# text to be displayed
label = "Bravo avocado!",
color = "firebrick",
size = 8
)

```



We can also single out some data points, like so:

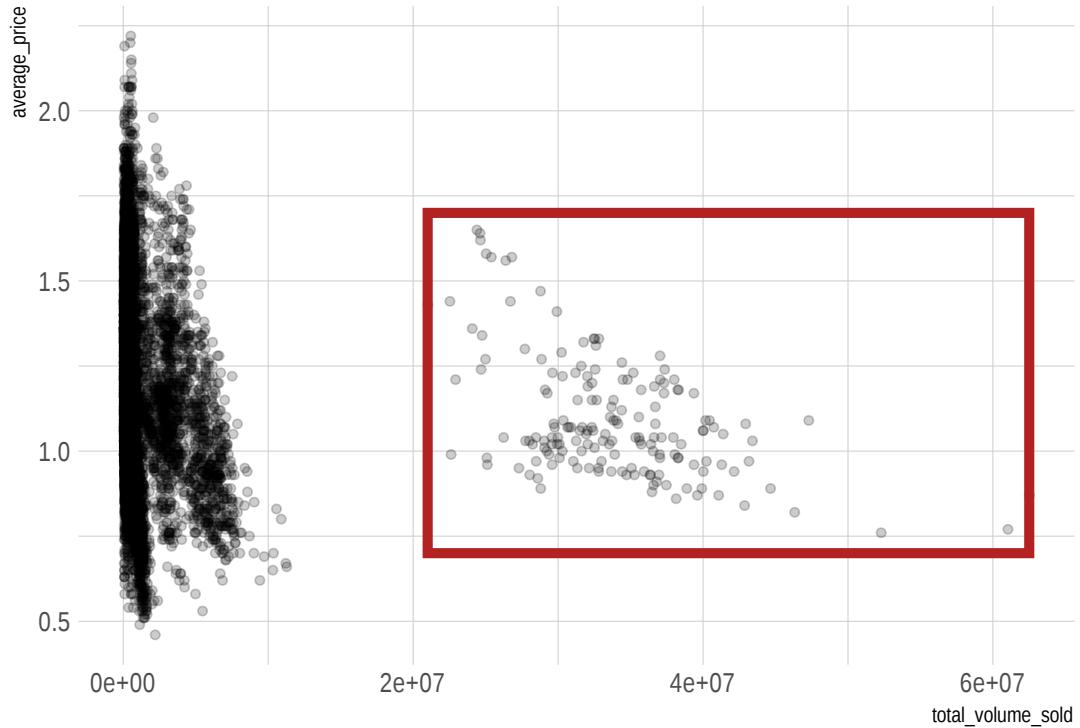
```

avocado_data %>%
  filter(type == "conventional") %>%
  ggplot(aes(x = total_volume_sold, y = average_price)) +
  geom_point(alpha = 0.2) +
  annotate(
    geom = "rect",
    # coordinates for the rectangle
    xmin = 2.1e7,
    xmax = max(avocado_data$total_volume_sold) + 100,
    ymin = 0.7,
    ymax = 1.7,
  )

```

## 6. Data Visualization

```
color = "firebrick",
fill = "transparent",
size = 2
)
```



## 6.5. Faceting

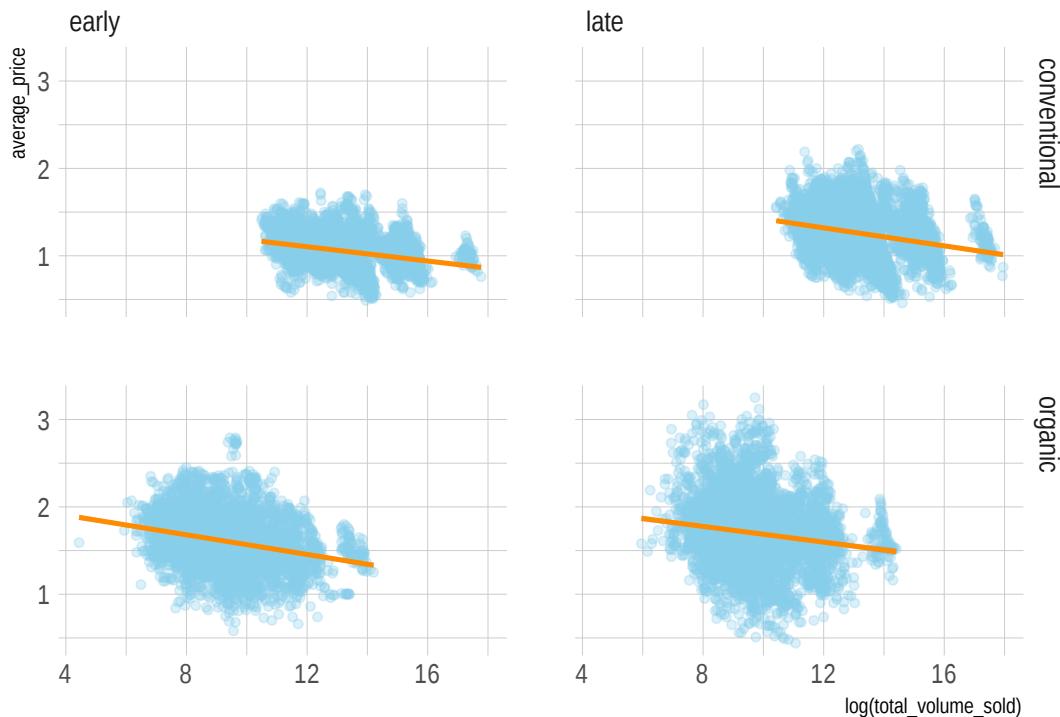
If we have grouping information, sometimes it can just get too much to put all of the information in a single plot, even if we use colors, shapes or linetypes for disambiguation. Facets are a great way to separately repeat the same kind of plot for different levels of relevant factors.

The functions `facet_grid` and `facet_wrap` are used for faceting. They both expect a formula-like syntax (we have not yet introduced formulas) using the notation `~` to separate factors. The difference between these functions shows most clearly when we have more than two factors. So let's introduce a new factor `early` to the avocado price data, representing whether a recorded measurement was no later than the median date or not.

```
avocado_data_early_late <- avocado_data %>%
  mutate(early = ifelse(Date <= median(Date), "early", "late"))
```

Using `facet_grid` we get a two-dimensional grid, and we can specify along which axis of this grid the different factor levels are to range by putting the factors in the formula notation like this: `row_factor ~ col_factor`.

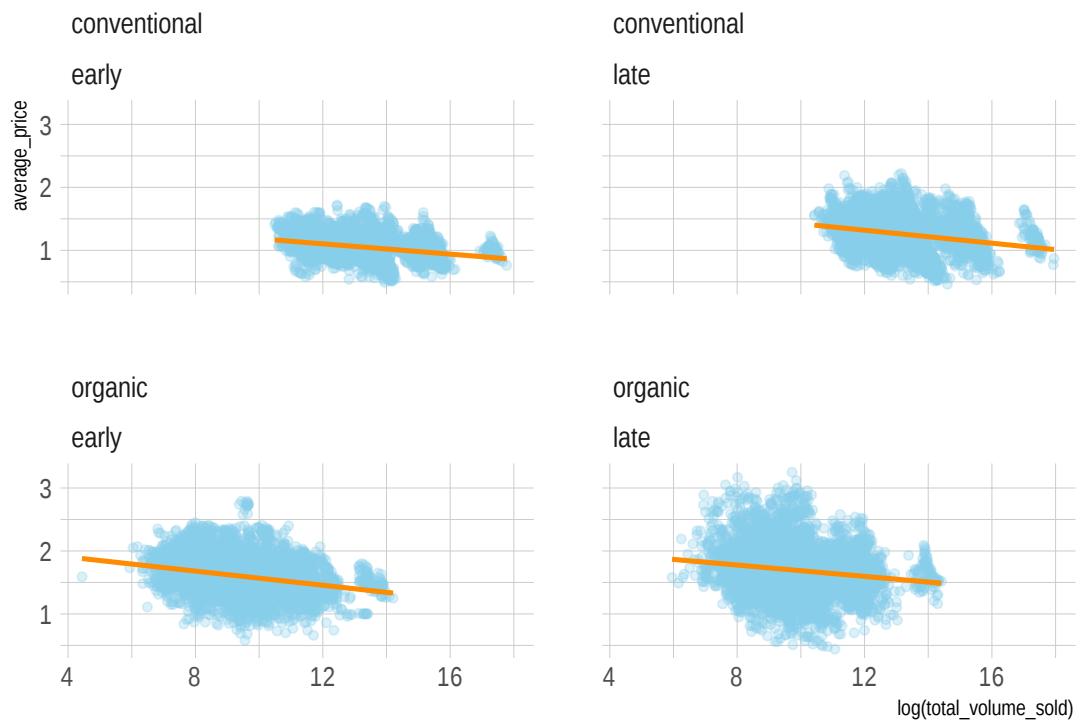
```
avocado_data_early_late %>%
  ggplot(aes(x = log(total_volume_sold), y = average_price)) +
  geom_point(alpha = 0.3, color = "skyblue") +
  geom_smooth(method = "lm", color = "darkorange") +
  facet_grid(type ~ early)
```



The same kind of plot realized with `facet_wrap` looks slightly different. The different factor level combinations are mashed together into a pair.

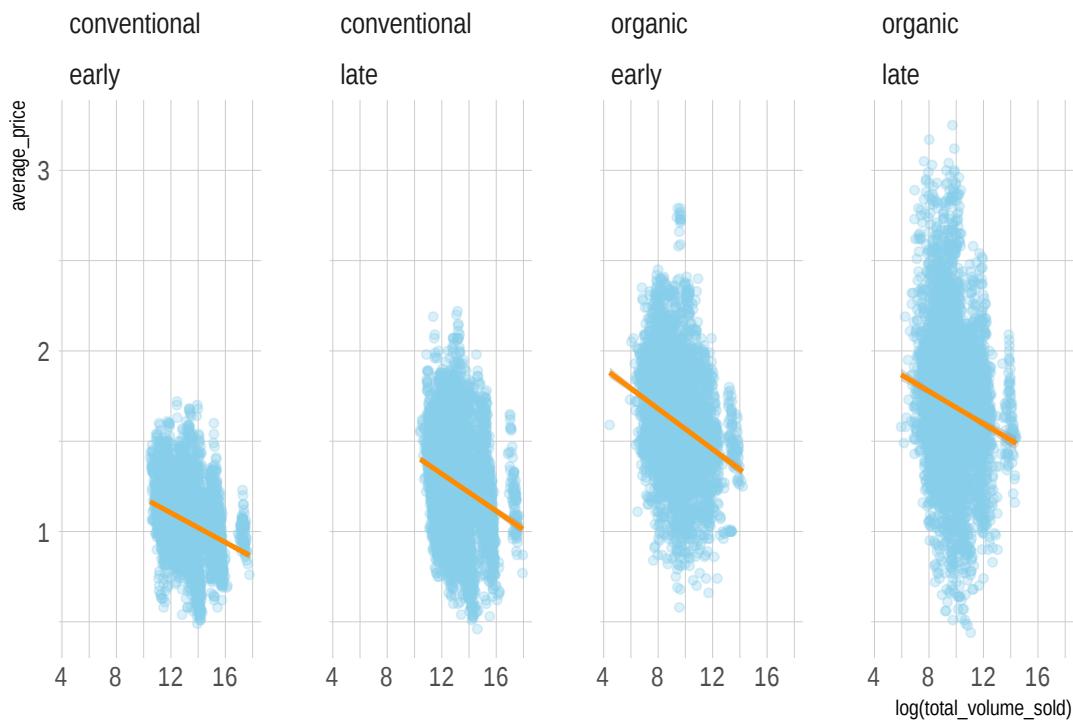
```
avocado_data_early_late %>%
  ggplot(aes(x = log(total_volume_sold), y = average_price)) +
  geom_point(alpha = 0.3, color = "skyblue") +
  geom_smooth(method = "lm", color = "darkorange") +
  facet_wrap(type ~ early)
```

## 6. Data Visualization



With `facet_wrap` it is possible to specify the desired number of columns or rows:

```
avocado_data_early_late %>%
  ggplot(aes(x = log(total_volume_sold), y = average_price)) +
  geom_point(alpha = 0.3, color = "skyblue") +
  geom_smooth(method = "lm", color = "darkorange") +
  facet_wrap(type ~ early, nrow = 1)
```



## 6.6. Customization etc.

There are many ways in which graphs can (and often: ought to) be tweaked further. The following can only cover a small, but hopefully useful selection.

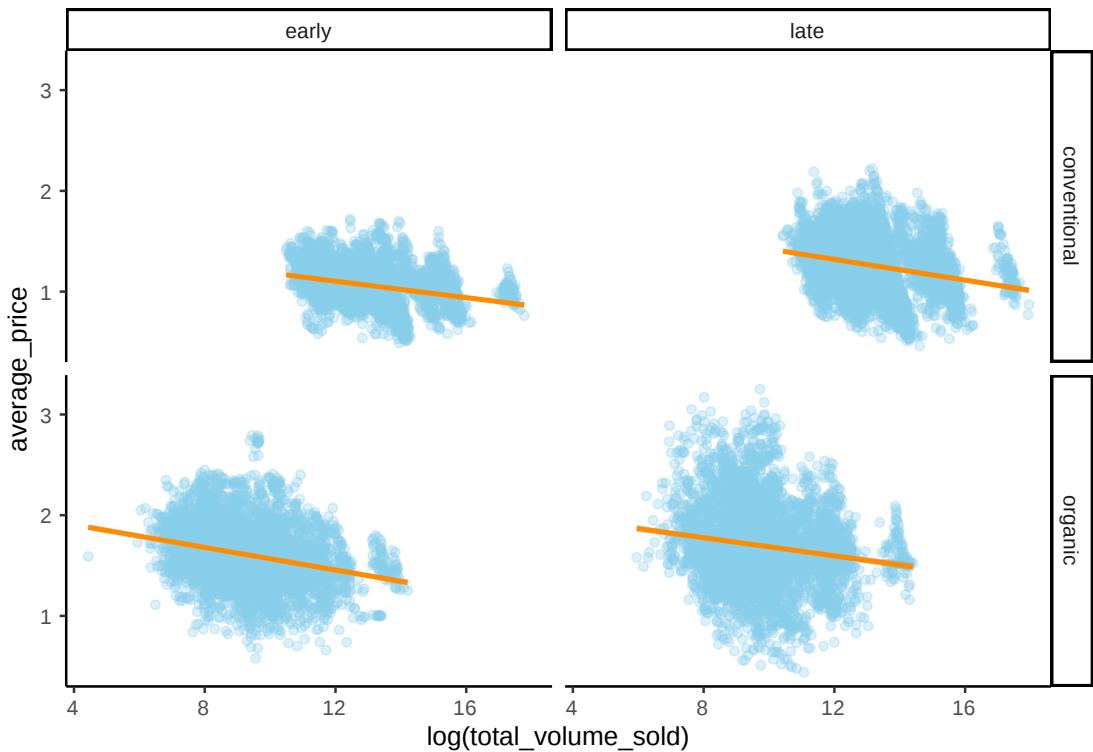
### 6.6.1. Themes

The general appearance of a plot is governed by its **theme**. There are many ready-made themes already in the `ggplot` package, as listed here, and there are more in several other packages. If we store a plot in a variable we can look at how different themes affect it.

```
avocado_grid_plot <- avocado_data_early_late %>%
  ggplot(aes(x = log(total_volume_sold), y = average_price)) +
  geom_point(alpha = 0.3, color = "skyblue") +
  geom_smooth(method = "lm", color = "darkorange") +
  facet_grid(type ~ early)

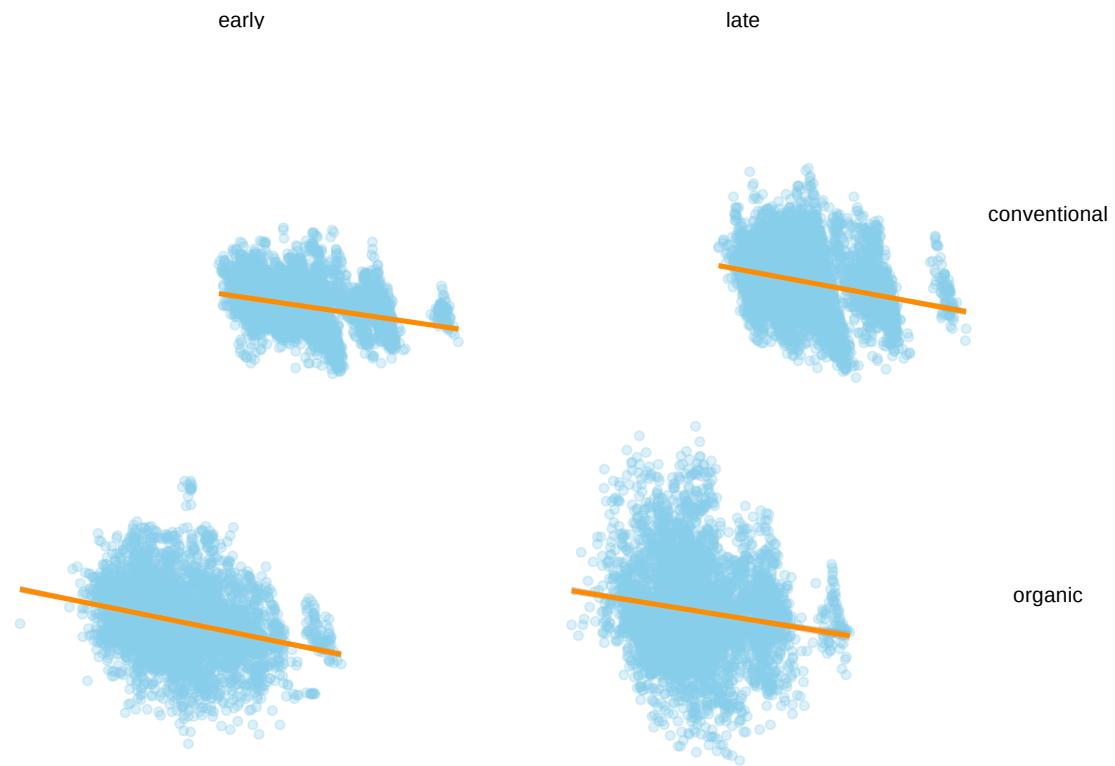
avocado_grid_plot + theme_classic()
```

## 6. Data Visualization



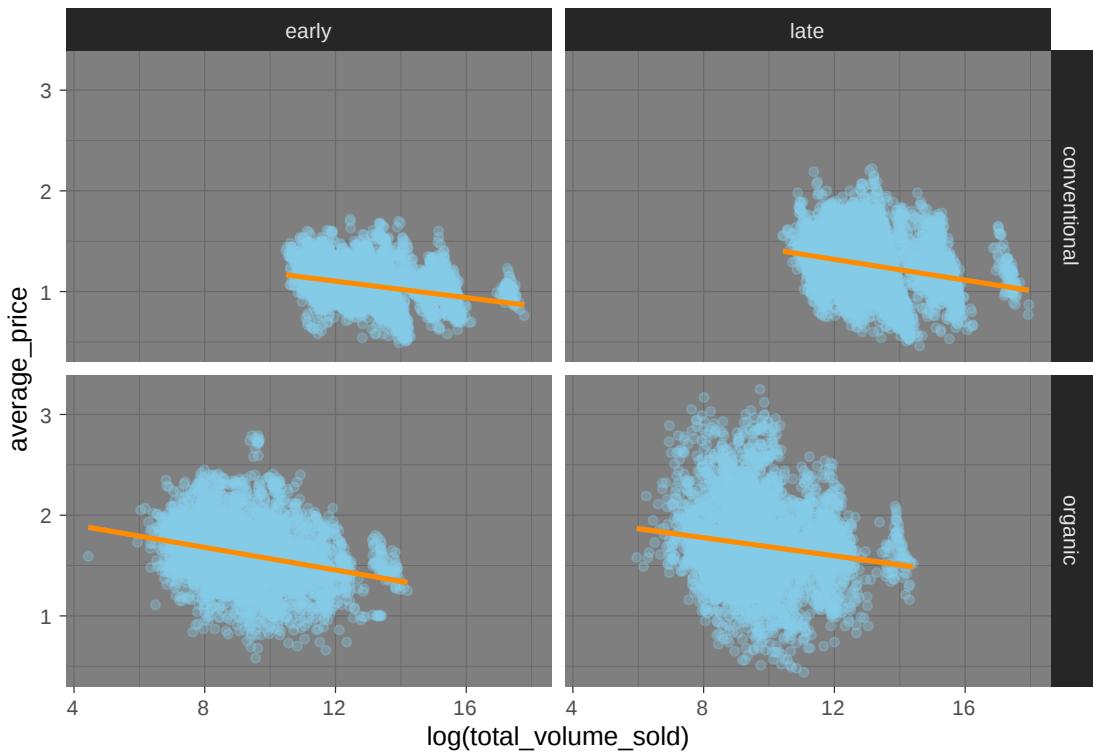
```
avocado_grid_plot + theme_void()
```

## 6.6. Customization etc.



```
avocado_grid_plot + theme_dark()
```

## 6. Data Visualization

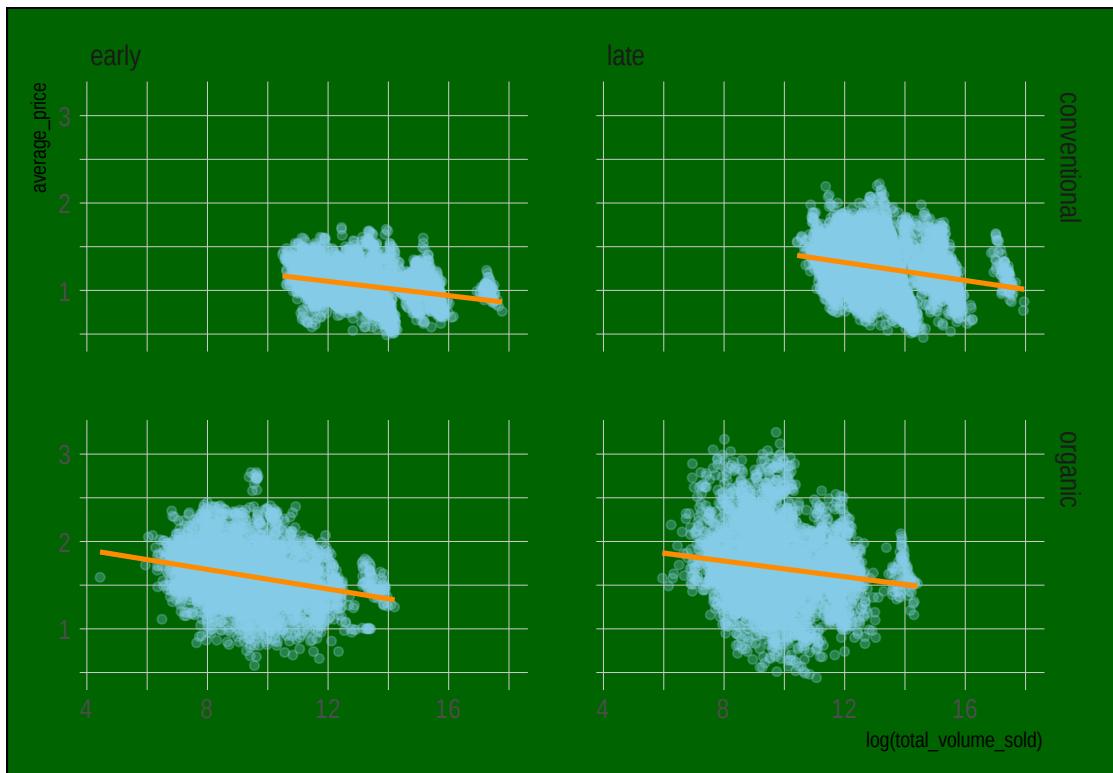


The plots in this book use the theme `hrbrthemes::theme_ipsum` from the `hrbrthemes` package as a default. You can set the default theme for all subsequent plots using a command like this:

```
# set the 'void' theme as global default
theme_set(
  theme_void()
)
```

More elaborate tweaking of a plot's layout can be achieved by the `theme` function. There are many options. Some let you do crazy things:

```
avocado_grid_plot + theme(plot.background = element_rect(fill = "darkgreen"))
```

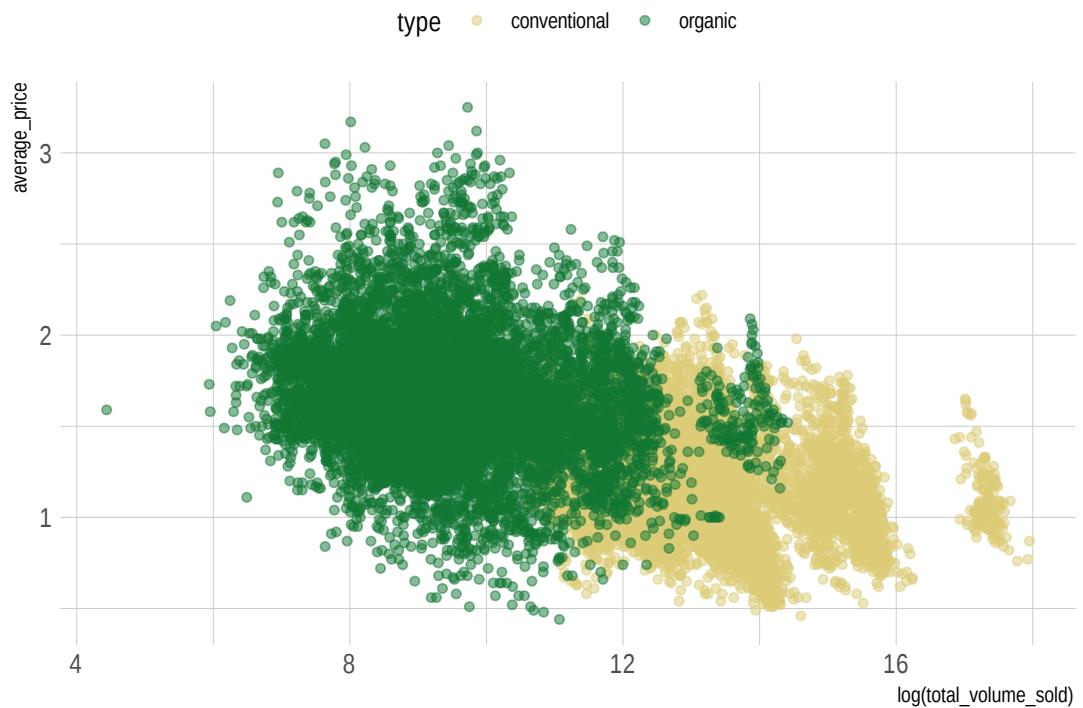


## 6.6.2. Guides

When using grouped variables (by color, shape, linetype, group, ...) `ggplot` creates a legend automatically.

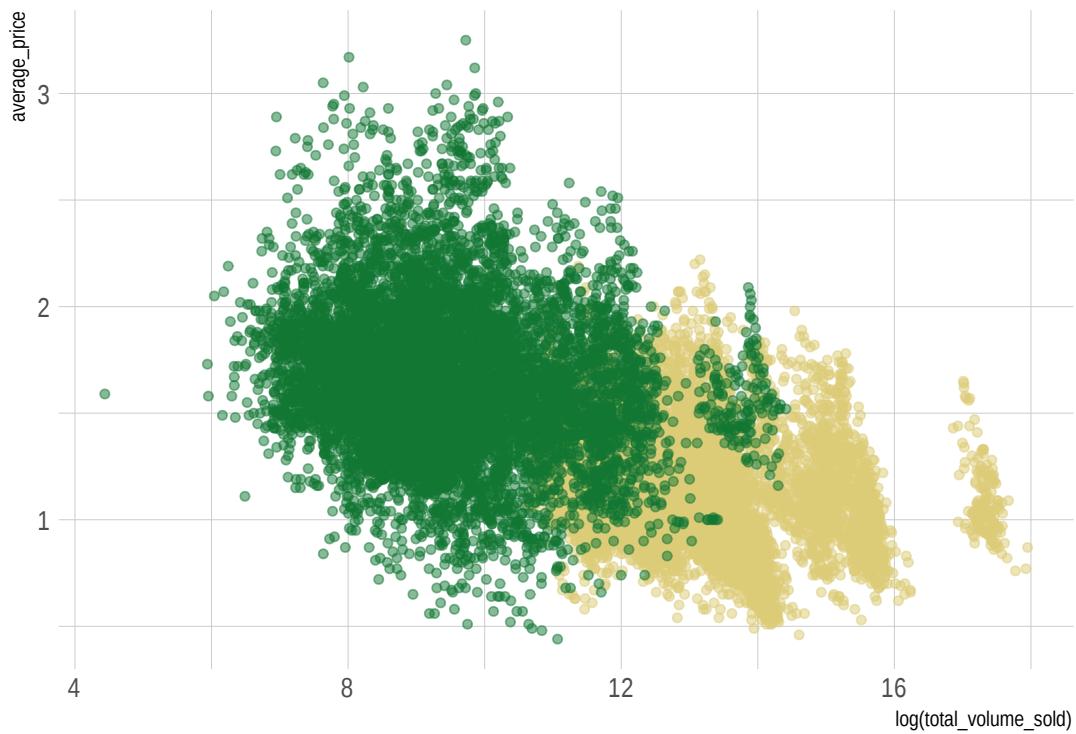
```
avocado_data %>%
  ggplot(
    mapping = aes(
      x = log(total_volume_sold),
      y = average_price,
      color = type
    )
  ) +
  geom_point(alpha = 0.5)
```

## 6. Data Visualization



The legend can be suppressed with the `guides` command. It takes as arguments the different types of grouping variables (like `color`, `group`, etc,)

```
avocado_data %>%
  ggplot(
    mapping = aes(
      x = log(total_volume_sold),
      y = average_price,
      color = type
    )
  ) +
  geom_point(alpha = 0.5) +
  # no legned for grouping by color
  guides(color = "none")
```

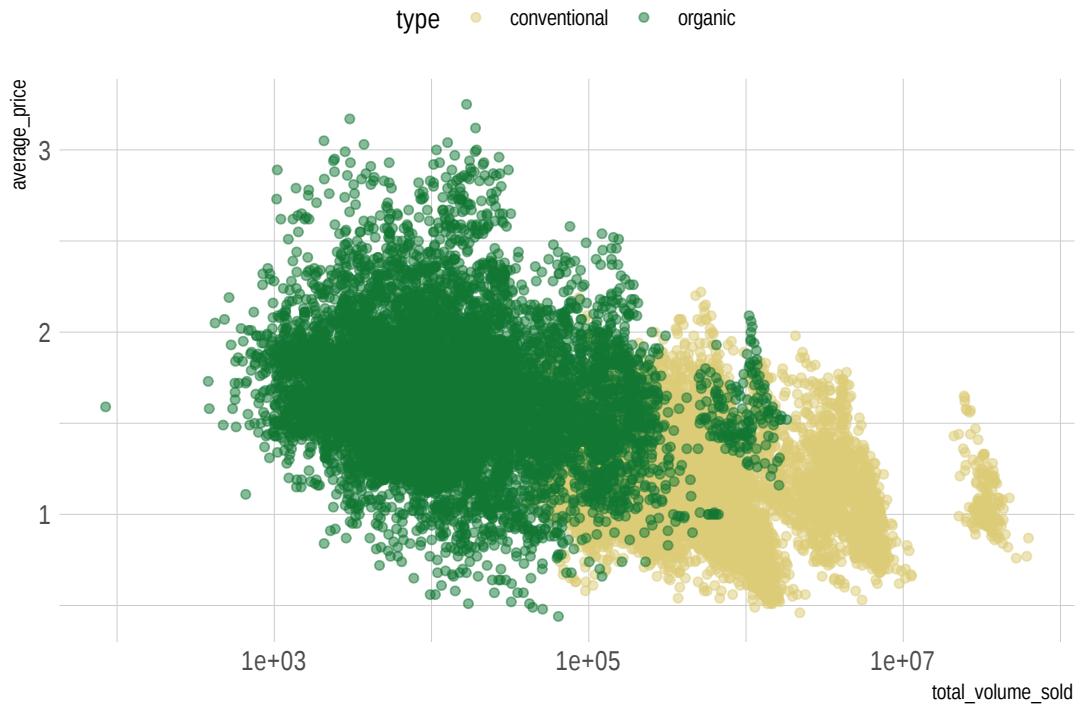


### 6.6.3. Axes, ticks and tick labels

If you need to use a non-standard (Cartesian) axis, you can do so, e.g., to change the *x*-axis to a log scale (with base 10):

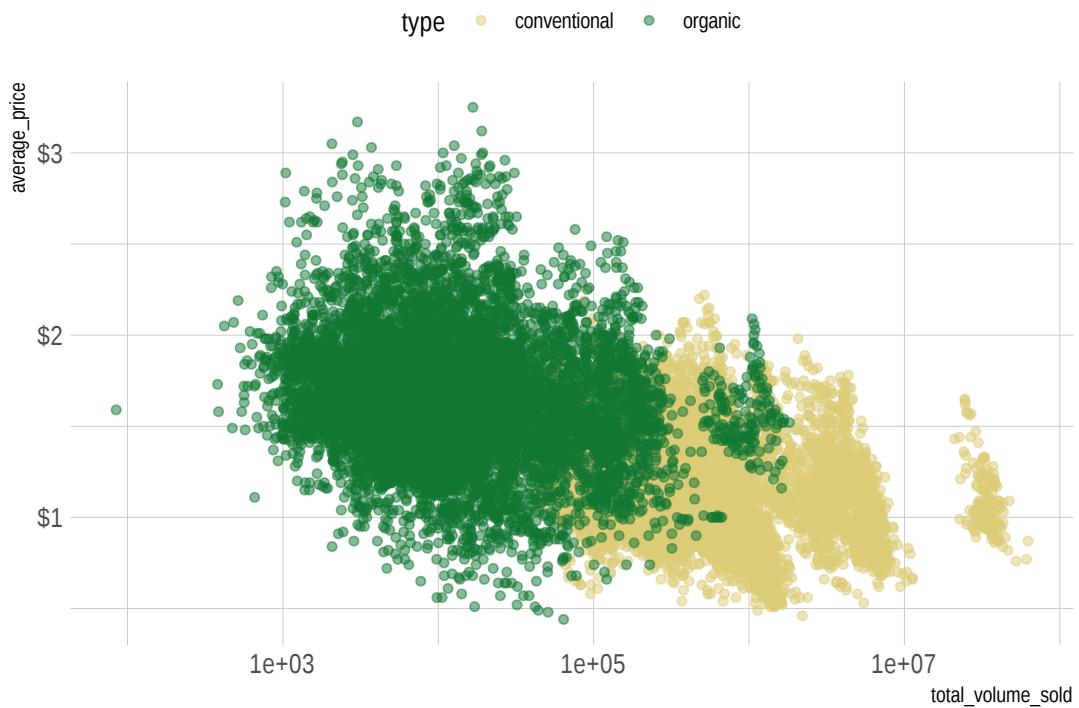
```
avocado_data %>%
  ggplot(
    mapping = aes(
      x = total_volume_sold,
      y = average_price,
      color = type
    )
  ) +
  geom_point(alpha = 0.5) +
  scale_x_log10()
```

## 6. Data Visualization



The `scales` package has a number of nice convenience functions for tweaking axis ticks (the places where axes are marked and possibly labelled) and tick labels (the labels applied to the tick marks). For example, we can add dollar signs to the price information, like so:

```
avocado_data %>%
  ggplot(
    mapping = aes(
      x = total_volume_sold,
      y = average_price,
      color = type
    )
  ) +
  geom_point(alpha = 0.5) +
  scale_x_log10() +
  scale_y_continuous(labels = scales::dollar)
```



#### 6.6.4. Labels

To change any other kind of labeling information (aside from tick mark labels on axes), the `labs` function can be used. It is rather self-explanatory:

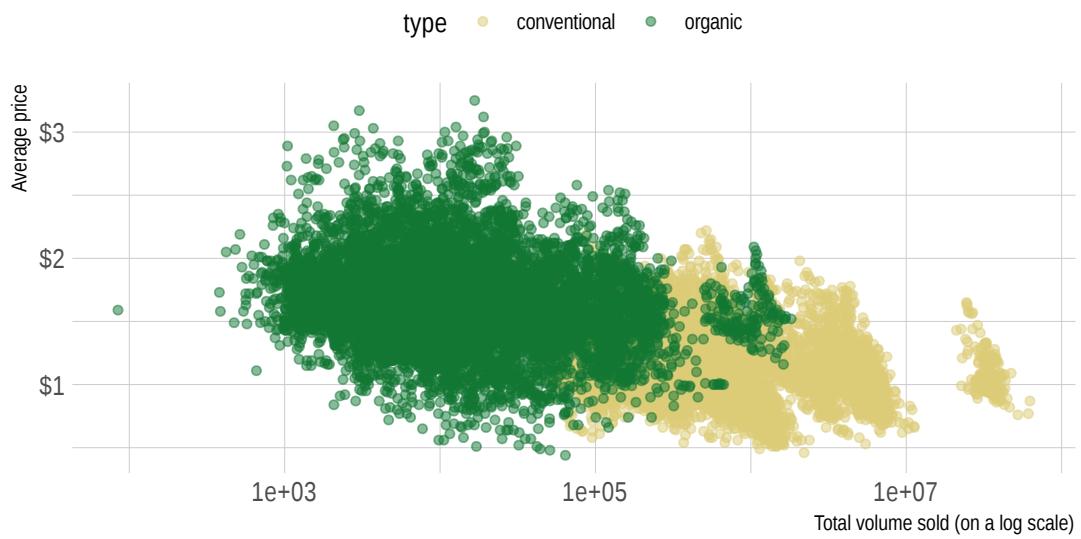
```
avocado_data %>%
  ggplot(
    mapping = aes(
      x = total_volume_sold,
      y = average_price,
      color = type
    )
  ) +
  geom_point(alpha = 0.5) +
  scale_x_log10() +
  scale_y_continuous(labels = scales::dollar) +
  # change axis labels and plot title & subtitle
  labs(
    x = 'Total volume sold (on a log scale)',
    y = 'Average price',
    title = "Avocado prices plotted against the amount sold per type",
  )
```

## 6. Data Visualization

```
    subtitle = "With linear regression lines",
    caption = "This plot shows the total volume of avocados sold against the average
)
```

### Avocado prices plotted against the amount sold per type

With linear regression lines



*This plot shows the total volume of avocados sold against the average price for many different points in time.*

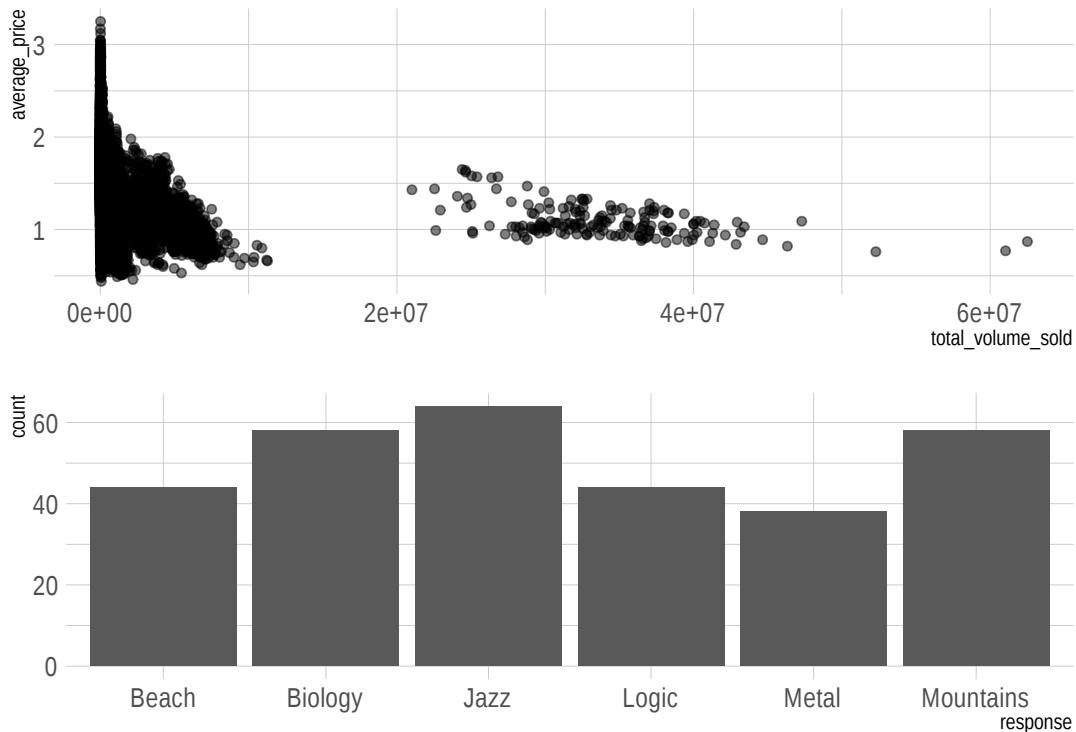
#### 6.6.5. Combining & arranging plots

Presenting visual information in a tightly packed spatial arrangement can be helpful for the spectator. Everything is within a single easy saccade, so to speak. Therefore it can be useful to combine different plots into a single combined plot. The `cowplot` package helps with this, in particular the function `cowplot::plot_grid` as shown here:

```
# create an avocado plot
avocado_plot <- avocado_data %>%
  ggplot(aes(x = total_volume_sold, y = average_price)) +
  geom_point(alpha = 0.5)

# create a BLJM bar plot
BLJM_plot <- data_BLJM_processed %>%
  ggplot(aes(x = response)) +
  geom_bar()
```

```
# combine both into one
cowplot::plot_grid(
  # plots to combine
  avocado_plot,
  BLJM_plot,
  # number columns
  ncol = 1
)
```



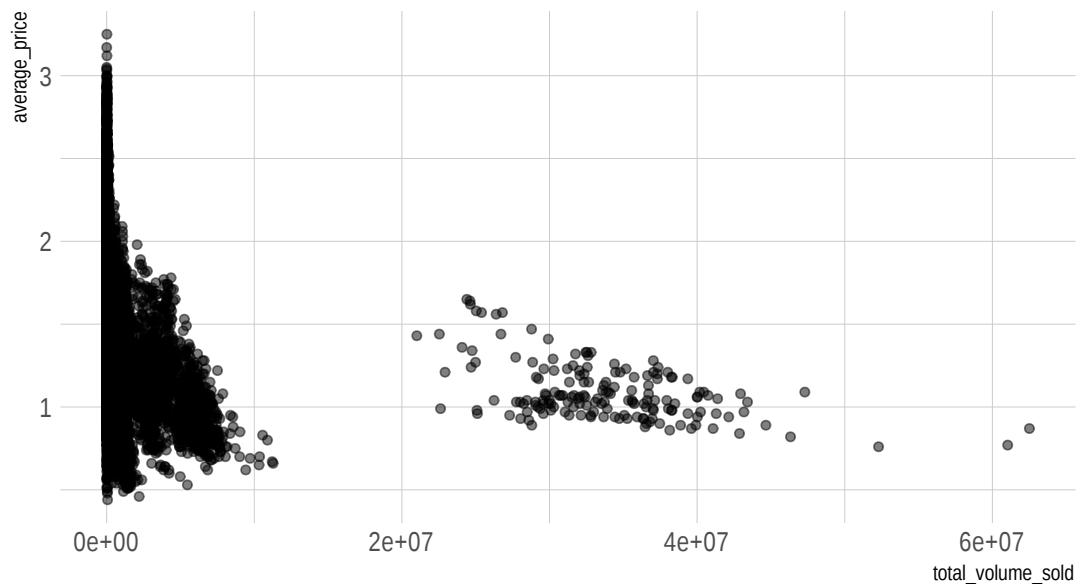
### 6.6.6. LaTeX expressions in plot labels

If you are enthusiastic about LaTeX, you can also use it inside of plot labels. The `latex2exp` package is useful here, which provides the function `latex2exp::TeX` to allow you to include LaTeX formulas. Just make sure that you double all backslashes, as in the following example:

```
avocado_data %>%
  ggplot(aes(x = total_volume_sold, y = average_price)) +
  geom_point(alpha = 0.5) +
  labs(title = latex2exp::TeX("We can use $\\LaTeX$ here: $\\sum_{i = 0}^n \\alpha^i$"))
```

## 6. Data Visualization

We can use L<sup>A</sup>T<sub>E</sub>X here:  $\sum_{i=0}^n \alpha^i$



**Part III.**

**Models and inferences**



# 7. Basics of Probability Theory

Probability is the basic ingredient of statistical inference.

In this chapter we will ...

**Content covered:** axiomatic definition, interpretation, joint distributions, marginalization, conditional probability & Bayes rule, random variables: discrete and continuous, expected values & variance

**Learning goal:** get comfortable with basic notions of probability theory

## 7.1. Probability

### 7.1.1. Outcomes, events, observations

We are interested in the space  $\Omega$  of all **elementary outcome**  $\omega_1, \omega_2, \dots$  of a process or event whose execution is (partially) random or unknown. Elementary outcomes are mutually exclusive. The set  $\Omega$  exhausts all possibilities.<sup>1</sup>

**Example.** The set of elementary outcomes of a single coin flip is  $\Omega_{\text{coin flip}} = \{\text{heads, tails}\}$ . The elementary outcomes of tossing a six-sided die is  $\Omega_{\text{standard die}} = \{ \text{, , , , , } \}$ .<sup>2</sup>

An **event**  $A$  is a subset of  $\Omega$ . Think of an event as a (possibly partial) observation. We might observe, for instance, not the full outcome of tossing a die, but only that there is a dot in the middle. This would correspond to the event  $A = \{ \text{, , } \}$ , i.e., observing an

---

<sup>1</sup>For simplicity of exposure, we gloss over subtleties arising when dealing with infinite sets  $\Omega$ . We make up for this when we define probability *density* functions for continuous random variables, which is all the uncountable infinity that we will usually be concerned with in applied statistics.

<sup>2</sup>Think of  $\Omega$  as a partition of the space of all possible ways in which the world could be, where we lump together into one partition cell all ways in which the world could be that are equivalent regarding those aspects of reality that we are interested in. We do not care whether the coin lands in the mud or in the sand. It only matters whether it came up heads or tails. Each elementary event can be realized in myriad ways.  $\Omega$  is our, the modellers', first crude simplification of nature, abstracting away aspects we currently do not care about.

## 7. Basics of Probability Theory

odd numbered outcome. The *trivial observation*  $A = \Omega$  and the *impossible observation*  $A = \emptyset$  are counted as events, too. The latter is included for technical reasons.

For any two events  $A, B \subseteq \Omega$ , standard set operations correspond to logical connectives in the usual way. For example, the conjunction  $A \cap B$  is the observation of both  $A$  and  $B$ ; the disjunction  $A \cup B$  is the observation that it is either  $A$  or  $B$ ; the negation of  $A$ ,  $\overline{A} = \{\omega \in \Omega \mid \omega \notin A\}$ , is the observation that it is not  $A$ .

### 7.1.2. Probability distributions

A **probability distribution**  $P$  over  $\Omega$  is a function  $P : \mathfrak{P}(\Omega) \rightarrow \mathbb{R}$  that assigns to all events  $A \subseteq \Omega$  a real number (from the unit interval, see A1 below), such that the following (so-called Kolmogorov axioms) are satisfied:

A1.  $0 \leq P(A) \leq 1$

A2.  $P(\Omega) = 1$

A3.  $P(A_1 \cup A_2 \cup A_3 \cup \dots) = P(A_1) + P(A_2) + P(A_3) + \dots$  whenever  $A_1, A_2, A_3, \dots$  are mutually exclusive<sup>3</sup>

Occasionally we encounter notation  $P \in \Delta(\Omega)$  to express that  $P$  is a probability distribution over  $\Omega$ . (E.g., in physics, theoretical economics or game theory. Less so in psychology or statistics.) If  $\omega \in \Omega$  is an elementary event, we often write  $P(\omega)$  as a shorthand for  $P(\{\omega\})$ . In fact, if  $\Omega$  is finite, it suffices to assign probabilities to elementary outcomes.

A number of rules follow immediately from of this definition (prove this!):

C1.  $P(\emptyset) = 0$

C2.  $P(\overline{A}) = 1 - P(A)$

C3.  $P(A \cup B) = P(A) + P(B) - P(A \cap B)$  for any  $A, B \subseteq \Omega$

### 7.1.3. Interpretations of probability

It is reasonably safe, at least preliminarily, to think of probability, as defined above, as a handy mathematical primitive which is useful for certain applications. There are at least three ways of thinking about where this primitive probability might come from, roughly paraphrasable like so:

1. **Frequentist:** Probabilities are generalizations of intuitions/facts about frequencies of events in repeated executions of a random event.
2. **Subjectivist:** Probabilities are subjective beliefs by a rational agent who is uncertain about the outcome of a random event.
3. **Realist:** Probabilities are a property of an intrinsically random world.

<sup>3</sup>A3 is the axiom of *countable additivity*. Finite additivity may be enough for finite or countable sets  $\Omega$ , but infinite additivity is necessary for full generality in the uncountable case.

Table 7.1.: Joint probability table for the flip-and-draw scenario

|       | heads                  | tails                  |
|-------|------------------------|------------------------|
| black | $0.5 \times 0.2 = 0.1$ | $0.5 \times 0.4 = 0.2$ |
| white | $0.5 \times 0.8 = 0.4$ | $0.5 \times 0.6 = 0.3$ |

### 7.1.4. Urns and frequencies

Think of an urn as a container which contains a number of  $N > 1$  balls. Balls can be of different color. For example, let us suppose that our urn has  $k > 0$  black balls and  $N - k$  white balls. (There is at least one black and one white ball.) For a single random draw from our urn we have:  $\Omega_{\text{our urn}} = \{\text{white, black}\}$ . If we imagine an infinite sequence of single draws from our urn, putting whichever ball we drew back in after every draw, the limiting proportion with which we draw a black ball is  $\frac{k}{N}$ . (If in doubt, execute this experiment. By hand or by computer.) This statement about frequency is what motivates saying that the probability of drawing a black ball on a single trial is (or should be<sup>4</sup>)  $P(\text{black}) = \frac{k}{N}$ .

## 7.2. Structured events & marginal distributions

### 7.2.1. Probability table for a flip-and-draw scenario

Suppose we have two urns. Both have  $N = 10$  balls. Urn 1 has  $k_1 = 2$  black and  $N - k_1 = 8$  white balls. Urn 2 has  $k_2 = 4$  black and  $N - k_2 = 6$  white balls. We sometimes draw from urn 1, sometimes from urn 2. To decide, we flip a fair coin. If it comes up heads, we draw from urn 1; if it comes up tails, we draw from urn 2.

An elementary outcome of this two-step process of flip-and-draw is a pair  $\langle \text{outcome-flip}, \text{outcome-draw} \rangle$ . The set of all possible such outcomes is:

$$\Omega_{\text{flip-and-draw}} = \{\langle \text{heads, black} \rangle, \langle \text{heads, white} \rangle, \langle \text{tails, black} \rangle, \langle \text{tails, white} \rangle\} .$$

The probability of event  $\langle \text{heads, black} \rangle$  is given by multiplying the probability of seeing “heads” on the first flip, which happens with probability 0.5, and then drawing a black ball, which happens with probability 0.2, so that  $P(\langle \text{heads, black} \rangle) = 0.5 \times 0.2 = 0.1$ . The probability distribution over  $\Omega_{\text{flip-draw}}$  is consequently as in Table 7.1. (If in doubt, start flipping & drawing and count your outcomes.)

---

<sup>4</sup>If probabilities are subjective beliefs, a rational agent is, in a sense, normatively required to assign exactly this probability.

## 7. Basics of Probability Theory

### 7.2.2. Structured events and joint-probability distributions

Table 7.1 is an example of a **joint probability distribution** over a structured event space, which here has two dimensions. Since our space of outcomes is the Cartesian product of two simpler outcome spaces, namely  $\Omega_{\text{flip-}\&-\text{draw}} = \Omega_{\text{flip}} \times \Omega_{\text{draw}}$ ,<sup>5</sup> we can use notation  $P(\text{heads, black})$  as shorthand for  $P(\langle \text{heads, black} \rangle)$ . More generally, if  $\Omega = \Omega_1 \times \dots \Omega_n$ , we can think of  $P \in \Delta(\Omega)$  as a joint probability distribution over  $n$  subspaces.

### 7.2.3. Marginalization

If  $P$  is a joint-probability distribution over event space  $\Omega = \Omega_1 \times \dots \Omega_n$ , the **marginal distribution** over subspace  $\Omega_i$ ,  $1 \leq i \leq n$  is the probability distribution that assigns to all  $A_i \subseteq \Omega_i$  the probability:<sup>6</sup>

$$P(A_i) = \sum_{A_1 \subseteq \Omega_1, \dots, A_{i-1} \subseteq \Omega_{i-1}, A_{i+1} \subseteq \Omega_{i+1}, \dots, A_n \subseteq \Omega_n} P(A_1, \dots, A_{i-1}, A_i, A_{i+1}, \dots, A_n)$$

For example, the marginal distribution over coin flips derivable from the joint probability distribution in Table 7.1 gives  $P(\text{heads}) = P(\text{tails}) = 0.5$ , since the sum of each column is exactly 0.5. The marginal distribution over flips derivable from Table 7.1 has  $P(\text{black}) = 0.3$  and  $P(\text{black}) = 0.7$ .<sup>7</sup>

## 7.3. Conditional probability

Fix probability distribution  $P \in \Delta(\Omega)$  and events  $A, B \subseteq \Omega$ . The conditional probability of  $A$  given  $B$ , written as  $P(A | B)$ , gives the probability of  $A$  on the assumption that  $B$  is true.<sup>8</sup> It is defined like so:

$$P(A | B) = \frac{P(A \cap B)}{P(B)}$$

Conditional probabilities are only defined when  $P(B) > 0$ .<sup>9</sup>

---

<sup>5</sup>With  $\Omega_{\text{flip}} = \{\text{heads, tails}\}$  and  $\Omega_{\text{draw}} = \{\text{black, white}\}$ .

<sup>6</sup>This notation, using  $\sum$ , assumes that subspaces are countable. In other cases, a parallel definition with integrals can be used.

<sup>7</sup>The term “marginal distribution” derives from such probability tables, where traditionally the sum of each row/column was written in the margins.

<sup>8</sup>We also verbalize this as “the conditional probability of  $A$  conditioned on  $B$ .”

<sup>9</sup>Updating with events which have probability zero entails far more severe adjustments of the underlying belief system than just ruling out information hitherto considered possible. Formal systems that capture such *belief revision* are studied in formal epistemology Halpern (2003).

**Example.** If a die is unbiased, each of its six faces has equal probability to come up after a toss. The probability of event  $B = \{1, 3, 5\}$  that the tossed number is odd has probability  $P(B) = \frac{1}{2}$ . The probability of event  $A = \{2, 3, 4\}$  that the tossed number is bigger than two is  $P(A) = \frac{2}{3}$ . The probability that the tossed number is bigger than two and odd is  $P(A \cap B) = P(\{3\}) = \frac{1}{3}$ . The conditional probability of tossing a number that is bigger than two, when we know that the toss is even, is  $P(A | B) = \frac{1/3}{1/2} = \frac{2}{3}$ .

Algorithmically, conditional probability first rules out all events in which  $B$  is not true and then simply renormalizes the probabilities assigned to the remaining events in such a way that the relative probabilities of surviving events remains unchanged. Given this, another way of interpreting conditional probability is that  $P(A | B)$  is what a rational agent *should* believe about  $A$  after observing that  $B$  is in fact true and nothing more. The agent rules out, possibly hypothetically, that  $B$  is false, but otherwise does not change opinion about the relative probabilities of anything that is compatible with  $B$ .

### 7.3.1. Bayes rule

Looking back at the joint-probability distribution in Table 7.1, the conditional probability  $P(\text{black} | \text{heads})$  of drawing a black ball, given that the initial coin flip showed heads, can be calculated as follows:

$$P(\text{black} | \text{heads}) = \frac{P(\text{black}, \text{heads})}{P(\text{heads})} = \frac{0.1}{0.5} = 0.2$$

This calculation, however, is quite spurious. We knew that already from the way the flip-and-draw scenario was set up. After flipping heads, we draw from urn 1, which has  $k = 2$  out of  $N = 10$  black balls, so clearly: if the flip is heads, then the probability of a black ball is 0.2. Indeed, in a step-wise random generation process like the flip-and-draw scenario, some conditional probabilities are very clear, and sometimes given by definition. These are, usually, the conditional probabilities that define how the process unfolds forward in time, so to speak.

**Bayes rule** is a way of expressing, in a manner of speaking, conditional probabilities in terms of the “reversed” conditional probabilities:

$$P(B | A) = \frac{P(A | B) \times P(B)}{P(A)}$$

Bayes rule is straightforward corollary of the definition of conditional probabilities, according to which  $P(A \cap B) = P(A | B) \times P(B)$ , so that:

$$P(B | A) = \frac{P(A \cap B)}{P(A)} = \frac{P(A | B) \cdot P(B)}{P(A)}$$

## 7. Basics of Probability Theory

Bayes rule allows for reasoning backwards from observed causes to likely underlying effects. When we have a feed-forward model of how unobservable effects probabilistically constrain observable outcomes, Bayes rule allows us to draw inferences about *latent/unobservable variables* based on the observation of their downstream effects.

Consider yet again the flip-and-draw scenario. But now assume that Jones flipped the coin and drew a ball. We see that it is black. What is the probability that it was drawn from urn 1, equivalently, that the coin landed heads? It is not  $P(\text{heads}) = 0.5$ , the so-called *prior probability* of the coin landing heads. It is a conditional probability, also called the *posterior probability*,<sup>10</sup> namely  $P(\text{heads} \mid \text{black})$ , but one that is not as easy and straightforward to write down as the reverse  $P(\text{black} \mid \text{heads})$  of which we said above that it is an almost trivial part of the set up of the flip-and-draw scenario. It is here that Bayes rule has its purpose:

$$P(\text{heads} \mid \text{black}) = \frac{P(\text{black} \mid \text{heads}) \times P(\text{heads})}{P(\text{black})} = \frac{0.2 \times 0.5}{0.3} = \frac{1}{3}$$

This result is quite intuitive. Drawing a black ball from urn 2 (i.e., after seeing tails) is twice as likely as drawing a black ball from urn 1 (i.e., after seeing heads). Consequently, after seeing a black ball drawn, with equal probabilities of heads and tails, the probability that the coin landed tails is also twice as large as that it landed heads.

## 7.4. Random variables

We have so far define a probability distribution as a function that assigns a probability to each subset of the space  $\Omega$  of elementary outcomes. A special case occurs when we are interested in a space of numeric outcomes.

A **random variable** is a function  $X : \Omega \rightarrow \mathbb{R}$  that assigns to each elementary outcome a numerical value.

**Example.** For a single flip of a coin we have  $\Omega_{\text{coin flip}} = \{\text{heads}, \text{tails}\}$ . A usual way of mapping this onto numerical outcomes is to define  $X_{\text{coin flip}} : \text{heads} \mapsto 1; \text{tails} \mapsto 0$ . Less trivially, consider flipping a coin two times. Elementary outcomes should be individuated by the outcome of the first flip and the outcome of the second flip, so that we get:

$$\Omega_{\text{two flips}} = \{\langle \text{heads}, \text{heads} \rangle, \langle \text{heads}, \text{tails} \rangle, \langle \text{tails}, \text{heads} \rangle, \langle \text{tails}, \text{tails} \rangle\}$$

Consider the random variable  $X_{\text{two flips}}$  that counts the total number of heads. Crucially,  $X_{\text{two flips}}(\langle \text{heads}, \text{tails} \rangle) = 1 = X_{\text{two flips}}(\langle \text{tails}, \text{heads} \rangle)$ . We assign the same numerical value to different elementary outcomes.

---

<sup>10</sup>The terms *prior* and *posterior* make sense when we think about an agent's belief state before (prior to) and after (posterior to) an observation.

### 7.4.1. Notation & terminology

Traditionally random variables are represented by capital letters, like  $X$ . Variables for the numeric values they take on are written as small letters, like  $x$ .

We write  $P(X = x)$  as a shorthand for the probability  $P(\{\omega \in \Omega \mid X(\omega) = x\})$  that an event occurs that is mapped onto  $x$  by random variable  $X$ . For example, if our coin is fair, then  $P(X_{\text{two flips}} = x) = 0.5$  for  $x = 1$  and 0.25 otherwise. Similarly, we can also write  $P(X \leq x)$  for the probability of observing an event that  $X$  maps to a number not bigger than  $x$ .

If the range of  $X$  is countable, we say that  $X$  is **discrete**. For ease of exposition, we may say that if the range of  $X$  is an interval of real numbers,  $X$  is called **continuous**.

### 7.4.2. Cumulative distribution functions, mass & density

For a discrete random variable  $X$ , the **cumulative distribution function**  $F_X$  associated with  $X$  is defined as:

$$F_X(x) = P(X \leq x) = \sum_{x' \in \{\text{Rng}(X) \mid x' \leq x\}} P(X = x')$$

The **probability mass function**  $f_x$  associated with  $X$  is defined as:

$$f_X(x) = P(X = x)$$

**Example.** Suppose we flip a coin with a bias of  $\theta$   $n$  times. What is the probability that we will see heads  $k$  times? If we map the outcome of heads to 1 and tails to 0, this probability is given by the Binomial distribution, as follows:

$$\text{Binom}(K = k; n, \theta) = \binom{n}{k} \theta^k (1 - \theta)^{n-k}$$

Here  $\binom{n}{k} = \frac{n!}{k!(n-k)!}$  is the binomial coefficient. It gives the number of possibilities of drawing an unordered set with  $k$  elements from a set with a total of  $n$  elements. Figure 7.1 gives an example of the Binomial distribution, concretely its probability mass function, for two values of the coin's bias,  $\theta = 0.25$  or  $\theta = 0.5$ , when flipping the coin  $n = 24$  times. Figure 7.2 gives the corresponding cumulative distributions.

For a continuous random variable  $X$ , the probability  $P(X = x)$  will usually be zero: it is virtually impossible that we will see precisely the value  $x$  realized in a random event that can realize uncountably many numerical values of  $X$ . However,  $P(X \leq x)$  does take workable values and so we define the cumulative distribution function  $F_X$  associated with  $X$  as:

$$F_X(x) = P(X \leq x)$$

## 7. Basics of Probability Theory

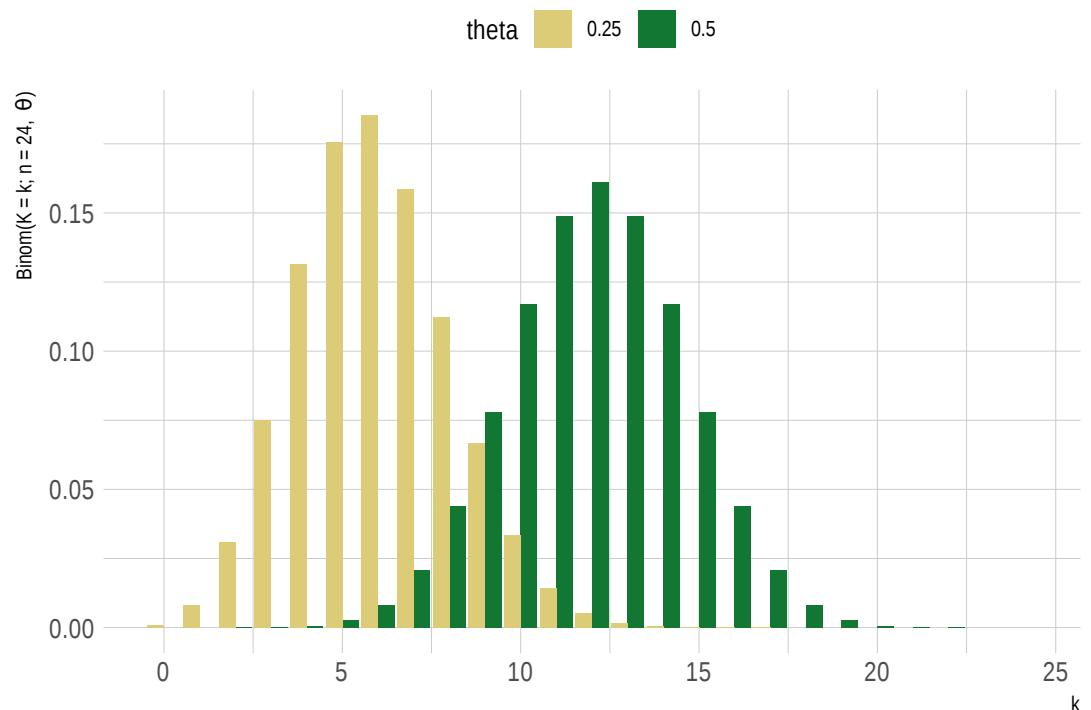


Figure 7.1.: Examples of the Binomial distribution. The  $y$ -axis give the probability of seeing  $k$  heads when flipping a coin  $n = 24$  times with a bias of either  $\theta = 0.25$  or  $\theta = 0.5$ .

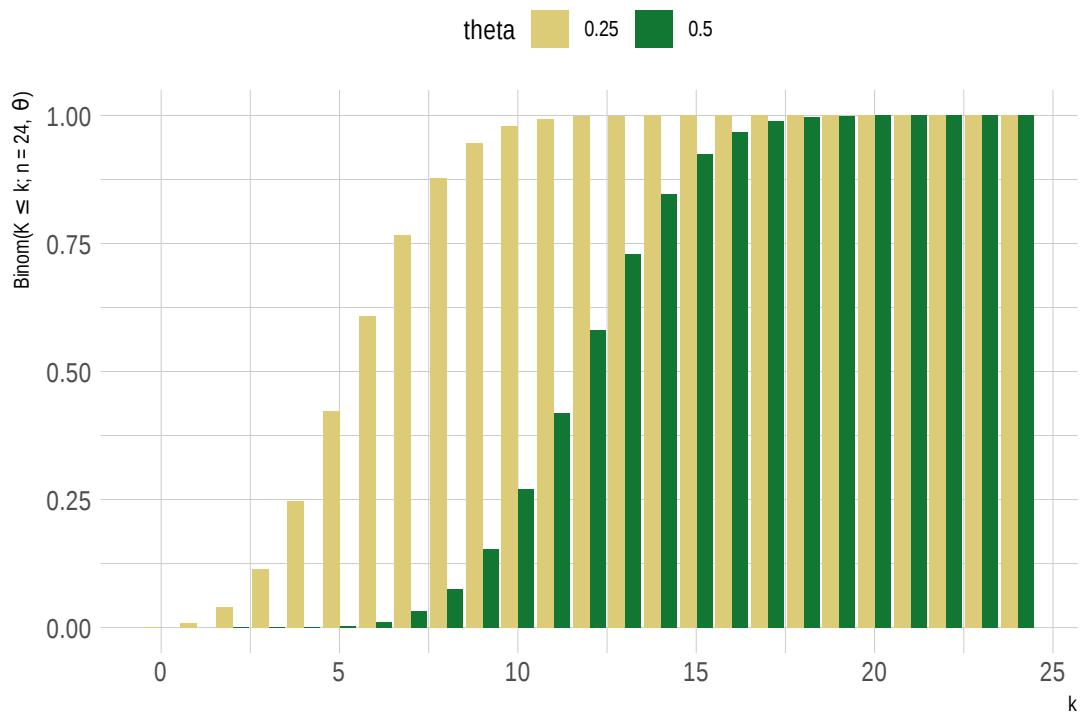


Figure 7.2.: Examples of the cumulative distribution of the Binomial. The  $y$ -axis gives the probability of seeing  $k$  or less outcomes of heads when flipping a coin  $n = 24$  times with a bias of either  $\theta = 0.25$  or  $\theta = 0.5$ .

## 7. Basics of Probability Theory

Instead of a probability **mass** function, we derive a **probability density function** from the cumulative function as:

$$f_X(x) = F'(x)$$

A probability density function can take values greater than one, unlike a probability mass function.

**Example.** The **Gaussian or Normal distribution** characterizes many natural distributions of measurements which are symmetrically spread around a central tendency. It is defined as:

$$\mathcal{N}(X = x; \mu, \sigma) = \frac{1}{\sqrt{2\sigma^2\pi}} \exp\left(-\frac{(x - \mu)^2}{2\sigma^2}\right)$$

where parameter  $\mu$  is the *mean*, the central tendency, and parameter  $\sigma$  is the *standard deviation*. Figure 7.3 gives examples of the probability density function of two normal distributions. Figure 7.4 gives the corresponding cumulative distribution functions.

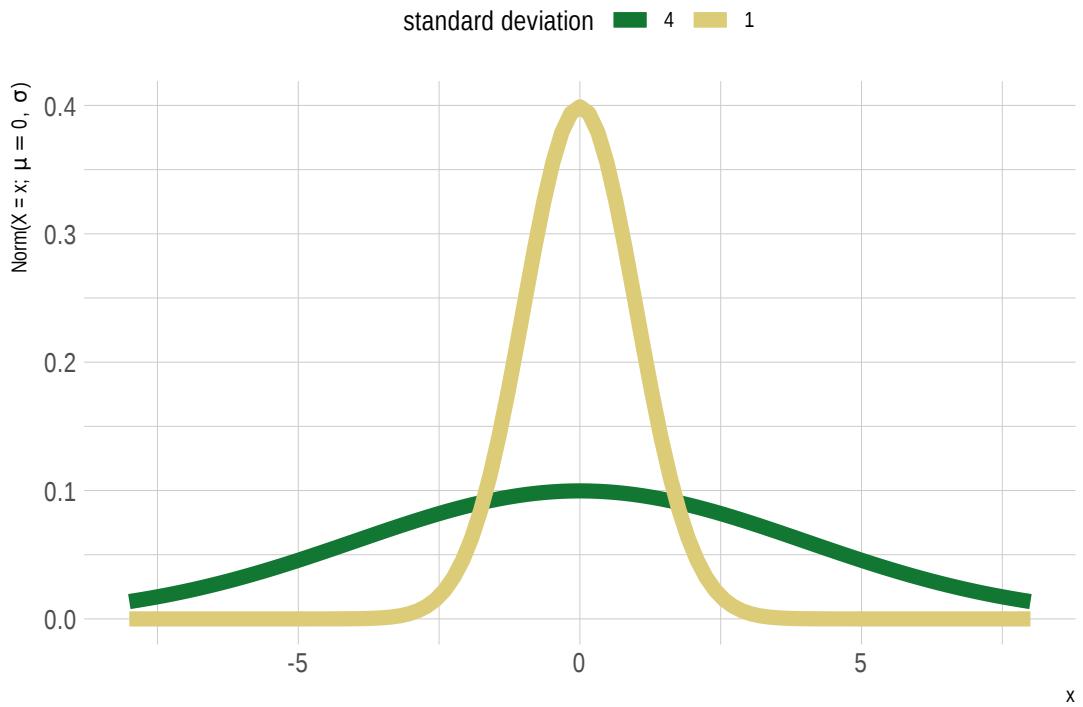


Figure 7.3.: Examples of the Normal distribution. In both cases  $\mu = 0$ , once with  $\sigma = 1$  and once with  $\sigma = 4$

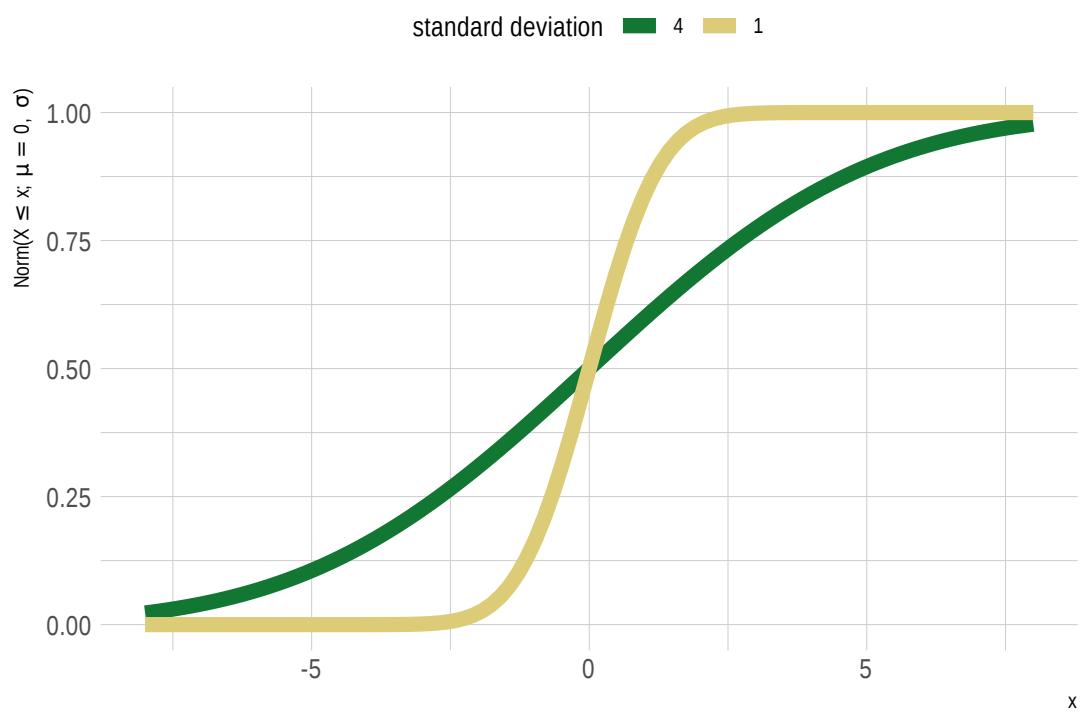


Figure 7.4.: Examples of the cumulative normal distribution corresponding to the previous probability density functions.

## 7.5. Expected value & variance

The **expected value** of a random variable  $X$  is a measure of central tendency. It tells us, like the name suggests, which average value of  $X$  we can expect when repeatedly sampling from  $X$ . If  $X$  is continuous, the expected value is:

$$\mathbb{E}_X = \sum_x x \times f_X(x)$$

If  $X$  is continuous, it is:

$$\mathbb{E}_X = \int x \times f_X(x) \, dx$$

The expected value is also frequently called the **mean**.

The **variance** of a random variable  $X$  is a measure of how much likely values of  $X$  are spread or clustered around the expected value. If  $X$  is discrete, the variance is:

$$\text{Var}(X) = \sum_x (\mathbb{E}_X - x)^2 \times f_X(x)$$

If  $X$  is continuous, it is:

$$\text{Var}(X) = \int (\mathbb{E}_X - x)^2 \times f_X(x) \, dx$$

**Example.** If we flip a coin with bias  $\theta = 0.25$  a total of  $n = 24$ , we expect on average to see  $n \times \theta = 24 \times 0.25 = 6$  outcomes showing heads.<sup>11</sup> The variance is  $n \times \theta \times (1 - \theta) = 24 \times 0.25 \times 0.75 = \frac{24 \times 3}{16} = \frac{18}{4} = 4.5$ .

The expected value of a normal distribution is just its mean  $\mu$  and its variance is  $\sigma^2$ .

---

<sup>11</sup>This is not immediately obvious from our definition, but it is intuitive and you can derive it.

# 8. Two approaches to statistical inference

## 8.1. Overview

In the first section we introduce the *model chapter* based on a short revision of the debate between *Frequentists* and *Bayesians*. After a short overview we formalize the conceptual ideas and the components of the presented model. We finish by discussing further example models.

## 8.2. Two notions of probability (revisited)

What are probabilities? Two viewpoints can be distinguished: Probabilities exist “outside in the world” or they are “subjective beliefs” (Kruschke 2015). Although both notions imply different approaches how to deal with probabilities, the mathematical properties are quite similar (Kruschke 2015).

### 8.2.1. Frequentism — Probabilities as properties of the world

When thinking about probabilities as existing “outside in the world” one has to consider the random process that produces the observed data. A Frequentist imagine this random process being repeated a large number of times so that the probability of an outcome is the number of observed outcomes divided by the total number of observations  $n$  (Dobson and Barnett 2008). Frequentism is an approach that searches for relative frequencies in a large number of trials (Vallverdú 2016). The parameter  $\theta$ , the probability of interest, is the value of the relative frequency when  $n$  becomes infinitely large. Consequently, the parameter  $\theta$  can be estimated from the observed data,  $\hat{\theta}$ , by maximizing the likelihood function (see section “Likelihood, Prior & Posterior”) (Dobson and Barnett 2008).

### 8.2.2. Bayesianism — Probabilities as subjective beliefs

Another notion of probabilities is to think of them as “beliefs” inside one’s mind.

#### 8.2.2.0.1. Bayes’ Theorem

The core of Bayesian methods is Bayes’ theorem which describes how prior belief is combined with observed data:

## 8. Two approaches to statistical inference

$$P(H|Data) = \frac{P(Data|H) * P(H)}{P(Data)}$$

or in plain language,

$$Posterior = \frac{Likelihood * Prior}{Marginal Likelihood},$$

The job of the “marginal Likelihood” in the denominator is to standardize the posterior and thus to ensure it sums up to one (integrates to one). Therefore, the key lesson of Bayes’ theorem is (McElreath 2015):

$$Posterior \propto Likelihood * Prior$$

### 8.2.2.0.2. To summarize up:

A parameter in *Bayesian methods* is conceptualized as a random variable with its own distribution (the posterior) that summarizes the current state of knowledge. The expected value of the posterior is the best guess about the true value of the parameter and its variability reflects the amount of uncertainty (Kline 2013).

In *Frequentist statistics*, a parameter is seen as a constant that should be estimated with sample statistics (Kline 2013).

# 9. Models

## 9.1. Likelihood, Prior, & Posterior

Before the topic of is introduced, some conceptual notions regarding likelihood, prior and posterior are necessary.

### 9.1.1. Probability density function vs. Likelihood function

As already introduced, for a coin flip the probability of each outcome can be modeled with the *Bernoulli distribution*, because two discrete outcomes (head or tail) and a constant probability  $\theta$  exist:

$$p([X = x]|\theta) = \theta^{[x]}(1 - \theta)^{(1-[x])}$$

where

- $\theta$  is the probability of coin-flip-outcome “head”, and
- the bracket [ ] indicates that the particular parameter is treated as unknown.

With the formula above the probability of  $\theta$  is treated as “known”. Accordingly, the distribution of possible outcomes can be derived.

But often the contrary is the case, that is one is interested in the value of  $\theta$  by a given data set. Then  $\theta$  is unknown and the data are observed. Treating  $\theta$  as parameter instead of  $x$  leads to the *likelihood function* — a mathematical formula that specifies the plausibility of the data. It states the probability of any possible observation:

$$p(X = x|[\theta]) = [\theta]^x(1 - [\theta])^{(1-x)}$$

Please be aware that through exchanging the roles of  $x$  and  $\theta$  in the second equation (likelihood function) this function is no longer a probability distribution and thus does not integrate to 1.

### 9.1.2. Prior & Posterior

The clearest difference between frequentist and Bayesian methods is the incorporation of *prior information* in the Bayesian framework. The posterior distribution results by combining the likelihood with the prior information:

*9. Models*

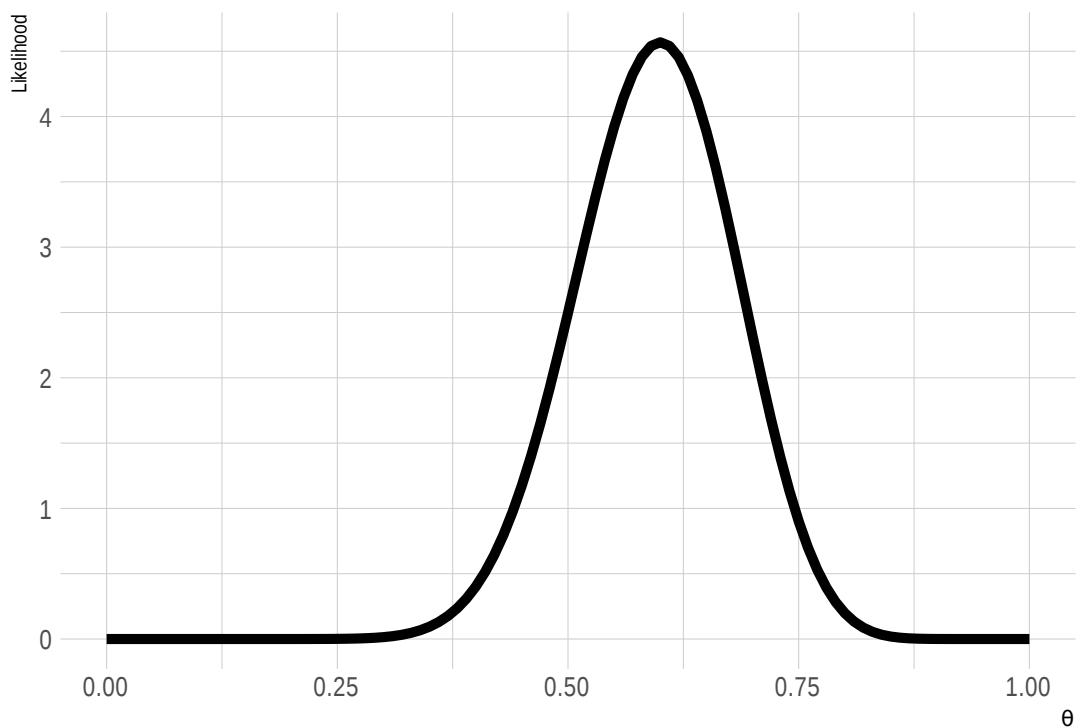


Figure 9.1.: Bernoulli likelihood for simulated observed coin-flip data.

### 9.1. Likelihood, Prior, & Posterior

$$\text{Posterior} \propto \text{Prior} \cdot \text{Likelihood},$$

or

$$P(B|A) \propto P(B) \cdot P(A|B).$$

When an uninformative (e.g. uniform) prior  $P(B)$  is used then the posterior  $P(B|A)$  is completely dependent on the data (likelihood)  $P(A|B)$ . The influence of the prior on the posterior depends on their relative weighting. Remember, the posterior is the conditional distribution of the parameter given the data. The mathematical procedure behind it is depicted by *Bayes' Theorem*.

Consider for example a beta distribution with the parameters  $a$  and  $b$ :  $\theta \sim \text{Beta}(a, b)$  for expressing prior knowledge. Assume further that the observed data are derived from a coin flip experiment, thus, the likelihood function is a bernoulli likelihood. The parameters of a beta distribution can be interpreted as  $a =$ no. of heads and  $b =$ no. of tails, or in other words:  $n = a + b$ . Consequently, Beta(1,1) has a lower weight and thus influence on the posterior than Beta(5,5).

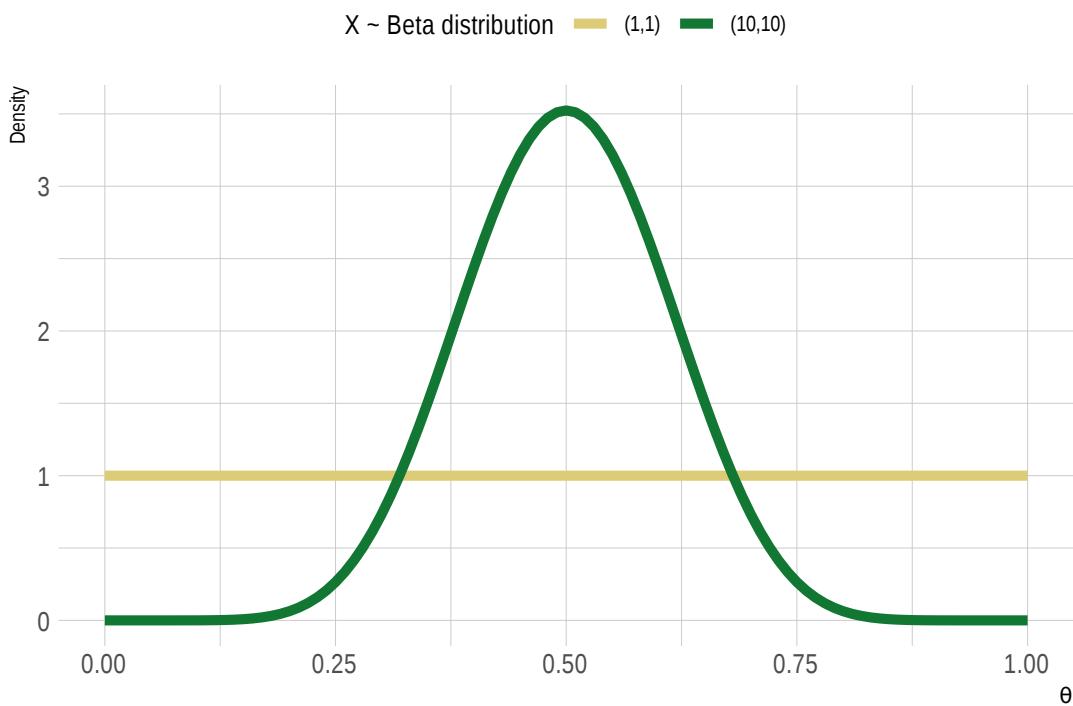


Figure 9.2.: Uninformative vs. informative beta prior distribution.

A lot of effort has been done in the area of “Objective Bayesian data analysis” in order to develop “uninformative prior distributions”, because a lot of people feel uncomfortable

## 9. Models

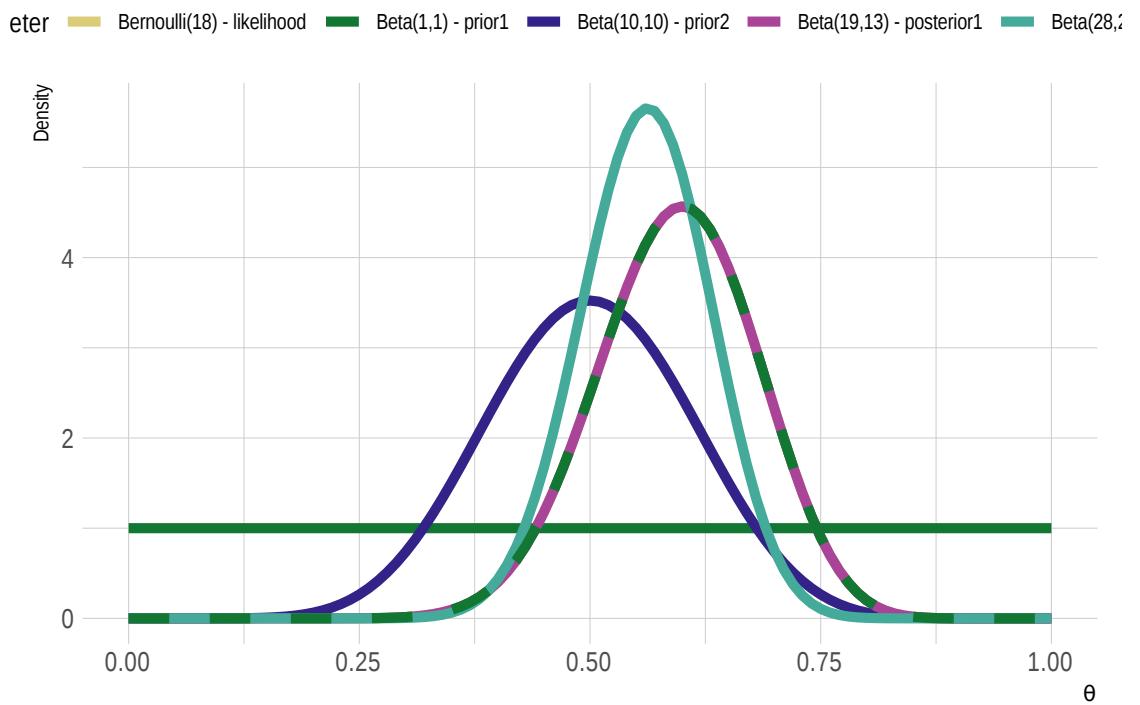


Figure 9.3.: Beta posterior distributions for different beta prior distributions and bernoulli likelihood.

## 9.1. Likelihood, Prior, & Posterior

using informative priors — seeing them as biased or unscientific. On the contrary, most people interpret results based on their prior experiences and in this light, prior information is just a way to quantify this (Dobson and Barnett 2008). It is “just” important that the definition of the prior distributions make sense (at every stage) in the model.

### 9.1.2.1. Exursos: Prior predictive distribution

So far we have seen that the *prior distribution* over parameters captures the initial assumptions or state of knowledge about the psychological variables they represent (Lee and Wagenmakers 2014), in the above example this variable is  $\theta$ .

Considering these initial assumptions in terms of prior distributions allow to make predictions about what data we would expect given the model and current state of knowledge. This distribution is called *prior predictive distribution*. It is a distribution over data, and gives the relative probability of different observable outcomes before any data have been seen (Lee and Wagenmakers 2014).

For the coin flip model we consider a flat prior distribution: Beta(1,1). The prior predictive distribution would therefore look like:

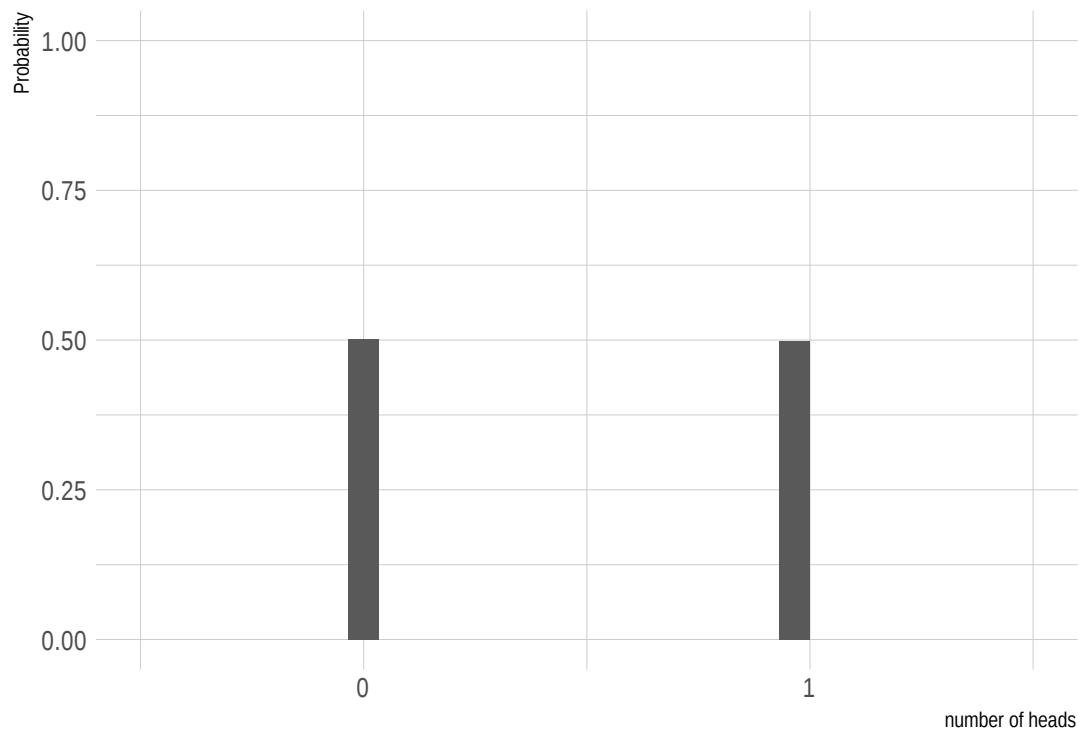


Figure 9.4.: Prior predictive distribution with Beta(1,1) prior for coin flip model

## 9.2. Modeling

### 9.2.1. Introductory example

As introductory example we considered a coin flip experiment and asked if a particular coin is *biased*. In order to investigate this question a coin is flipped  $x$  times (= trials) and the number of success (i.e. numbers of “head”)  $k$  is recorded. This is repeated  $n$  times (=observations).

```
## # A tibble: 10 x 3
##       n     k     x
##   <dbl> <dbl> <dbl>
## 1     1     22    30
## 2     2     15    30
## 3     3     16    30
## 4     4     12    30
## 5     5     14    30
## 6     6     15    30
## 7     7     13    30
## 8     8     14    30
## 9     9     20    30
## 10    10    13    30
```

The above table shows the observed outcome, but how the underlying probability of coming up *heads* can be derived from that data set?

### 9.2.2. Steps of Data Analysis

The approach described here is based on (McElreath 2015; Kruschke 2015). Although the approach is introduced in a Bayesian context, it can be used as a general guideline (with some caveats):

- Identify the relevant variables according to the hypothesis (Measurement scales, predicted vs. predictor variables).
- Define the descriptive model for the relevant variables.
  - likelihood distribution (distribution of each outcome variable that defines the plausibility of individual observations)
  - parameters (define and name all parameters of the model in order to relate the likelihood to the predictor variable(s))
- Bayesian context: Specify prior distribution(s).

Further steps that will be subject of later chapters:

- Inference and interpretation of the results.
- Model checking (Is the defined model adequate?)

In the following we are interested in the question if a certain coin is biased.

**First step** is to identify the relevant variables. For the coin flip experiment a coin is flipped  $n$  times, whereby each observation consists of  $x$  trials. Imagine for example 10 people flip a coin 30 times, then  $n = 10$  and  $x = 30$ . The variable *coin flip*  $Y$  is dichotomous with the possible outcomes “head” and “tail”. For each observation the outcome is recorded: “0” for coming up tail and “1” for coming up head. The data are summarized for each observation. The variable  $k$  indicates the number of heads coming up in  $x$  trials.

In **the second step** a descriptive model for the identified variables has to be defined. An underlying probability  $\theta$  is assumed, indicating the probability of heads coming up  $p(y = 1)$ . The probability that the outcome is head, given a value of parameter  $\theta$ , is the value of  $\theta$  (Kruschke 2015, 109). Formally, this can be written as

$$p(y = 1|\theta) = \theta$$

As only two outcomes of  $Y$  exists, the probability that the outcome is tail is the complementary probability  $1 - \theta$ . Both probabilities can be combined in one probability expression:

$$Pr(Y|n, \theta) = \frac{n!}{y!(n-y)!} \theta^y (1-\theta)^{n-y}.$$

This probability distribution is called the **Binomial distribution**. The fracture at the beginning indicates how many ordered sequences of  $n$  outcomes a count  $y$  have.

When the coin is flipped only once, then the probability can be written as:

$$Pr(Y|\theta) = \theta^y (1-\theta)^{1-y}.$$

This special variant of the Binomial distribution is the so-called **Bernoulli distribution**. To see the connection to the first considerations: When the outcome “head” is observed the equation reduces to  $Pr(y = 1|\theta) = \theta$  and when the outcome “tail” is observed the equation results in  $Pr(y = 0|\theta) = (1-\theta)$ .

Accordingly, for the introductory example it can be noted that the coin flip variable  $Y$  is distributed as Binomial distribution. (Note: For Bayes’ rule the likelihood function is needed. Remember, the likelihood function treats  $\theta$  as unknown and the data as known. This role of parameter is exchanged in a probability distribution.)

**The third step** is solely a *Bayesian idea*, that is the incorporation of prior knowledge. What do we believe about the coin bias  $\theta$  before seeing the data? Assuming that no expectation about  $\theta$  exists a priori, indicating that all values of  $\theta$  between 0 and 1 are equally probable. This can be modeled by a uniform distribution or as already visualized as Beta distribution with parameters  $a=1$  and  $b=1$  (see following figure).

So far, the coin flip model is defined conceptually. In the following some notational considerations have to be made.

## 9. Models

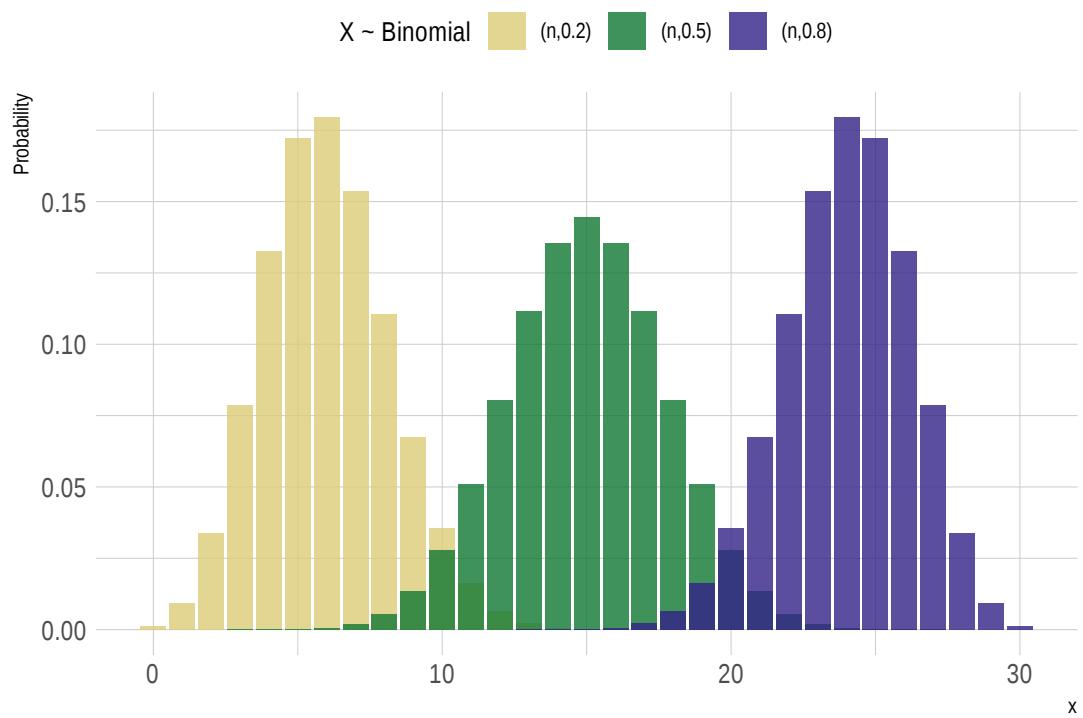


Figure 9.5.: Binomial-distribution for different coin biases

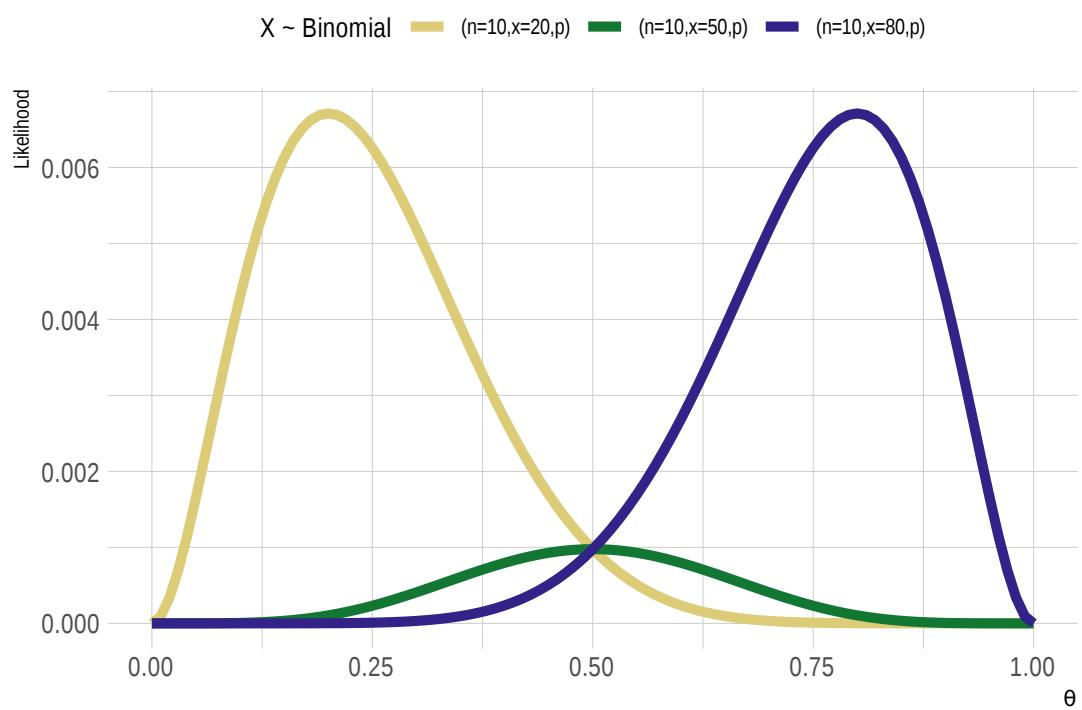


Figure 9.6.: Binomial likelihoods for different observed coin flip outcomes.

*9. Models*

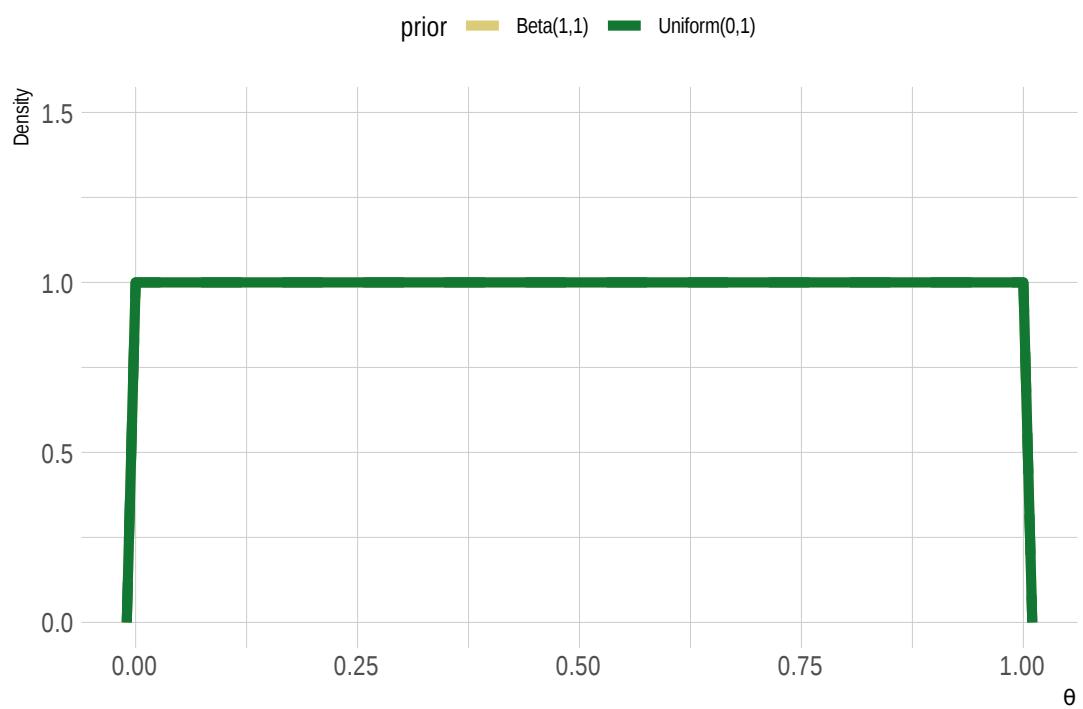


Figure 9.7.: Using uninformative prior distributions: The Uniform(0,1) and Beta(1,1) prior.

### 9.2.3. Notation

#### 9.2.3.1. Textual notation

In the textual notation, first the prior assumptions (if the Bayesian perspective is taken) are described. For the coin flip example this is:

$$\theta \sim Beta(1, 1).$$

The symbol “ $\sim$ ” means “is distributed as”, thus, the above equation says before seeing the data all possible values of  $\theta$  between 0 and 1 are assumed to be equally likely.

Subsequently, the descriptive model for the data has to be defined. As already described in the section above, it is assumed that the observed data (upcoming of heads  $k$ ) are distributed as Binomial distribution with given  $n$  (number of observations) and unknown  $\theta$ . This relation is denoted symbolically as

$$k \sim Binomial(\theta|n).$$

To summarize the current model (whereby the prior knowledge is only considered from a Bayesian perspective):

$$\theta \sim Beta(1, 1),$$

$$k \sim Binomial(\theta|n).$$

#### 9.2.3.2. Graphical notation

When models get very complex and incorporate many parameters it can be difficult to tease out all relations between the model components. In such a situation a graphical notation of a model might be helpful. In the following the convention described in Wagenamakers and Lee's *Bayesian Cognitive Modelling* (2014) is used: The graph structure is used to indicate dependencies between the variables, with children depending on their parents (Lee and Wagenamakers 2014). General conventions:

- Nodes - problem relevant variables,
- shaded nodes - observed variables,
- unshaded nodes - unobserved variables,
- circular nodes - continuous variables,
- square nodes - discrete variables,
- single line - stochastic dependency, and
- double line - deterministic dependency.

For the introductory example this indicates:

- relevant variables: number of trials ( $n$ ), number of success ( $k$ ) and probability for a success ( $\theta$ ),

## 9. Models

- observed variables:  $n$  and  $k$ ,
- unobserved variables:  $\theta$ ,
- continuous variable:  $\theta$ ,
- discrete variables:  $n$  and  $k$ .

In the next step the dependencies have to be determined:

The number of success  $k$  depends on the probability of a success  $\theta$  as well as on the number of trials  $n$ .

Finally, the graphical structure together with the textual notation can be represented:

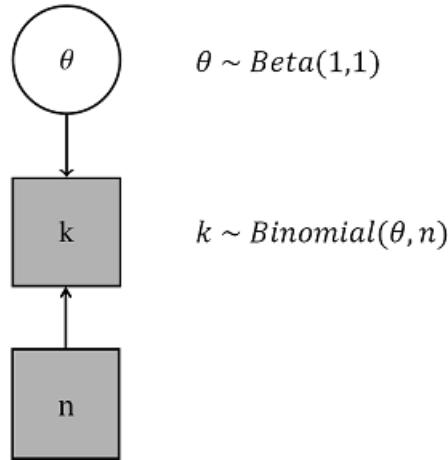


Figure 9.8.: Graphical notation Beta-Binomial Model - One group

### 9.2.4. An outlook: Hierarchical models

Often data can be considered as part of an overall structure. Single observations can be modelled belonging into different groups. These groups in turn are part of a superordinate group etc. Such information are presented in a model in form of a hierarchy.

For example, consider again the coin flip experiment. The outcome of *head* is influenced by the probability  $\theta$ . Further,  $\theta$  is assumed to be distributed as  $Beta(1,1)$ . Remember that the parameter  $a$  and  $b$  of a Beta-distribution can be considered in this context as:  $a =$ number of heads and  $b =$  number of tails, consequently,  $n = a + b$ .

#### 9.2.4.1. Reparameterization of a Beta distribution

Probability distributions can be described by their *central tendency* and *spread* (or dispersion). The *mode* of a Beta distribution is defined as:

$$\omega = \frac{a-1}{a+b-2},$$

and the concentration as:

$$\kappa = a + b.$$

The nice thing is, that the definition of the *mode* as well as of the *concentration* consists solely of the parameters  $a$  and  $b$ . Therefore, it is possible to re-express the parameters of a Beta density in terms of  $\omega$  and  $\kappa$ , such that:

$$\text{Beta}(a, b) = \text{Beta}(\omega(\kappa - 2) + 1, (1 - \omega)(\kappa - 2) + 1).$$

*Why this is useful? And what is its value in connection with hierarchical modeling?*

Return back to the coin flip experiment. So far, the parameters of the prior on  $\theta$  are fixed:  $a = 1$  and  $b = 1$ . Assume that we get further information: The manufacturing process of the coins has a bias near  $\omega$  (example taken from (Kruschke 2015)). But how to incorporate this additional knowledge in the model?

At this point, the hierarchy and the reparameterization come into play. Hierarchy because a further assumption is placed on top of the existing model and reparameterization, because we want to express the prior in terms of the mode  $\omega$ .

Such that the model can be assumed as follows:

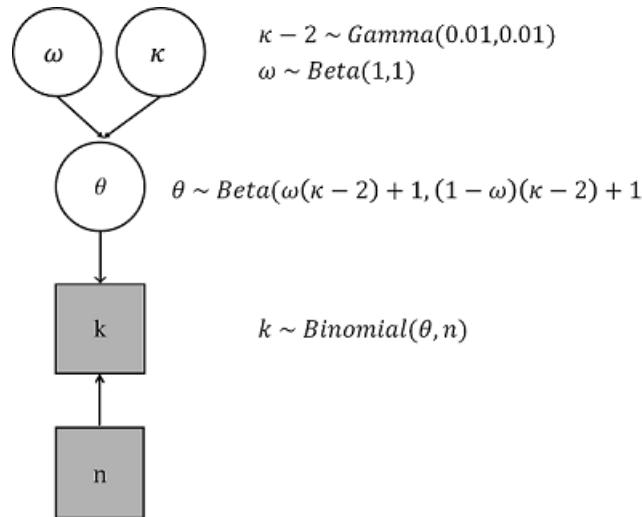


Figure 9.9.: Graphical notation hierarchical Beta-Binomial Model - One group

Now, the parameters of the hyperpriors (Gamma and Beta) are fixed, but they can be treated as parameters as well ... as such hierarchical models can be created with any degree of complexity:

## 9. Models

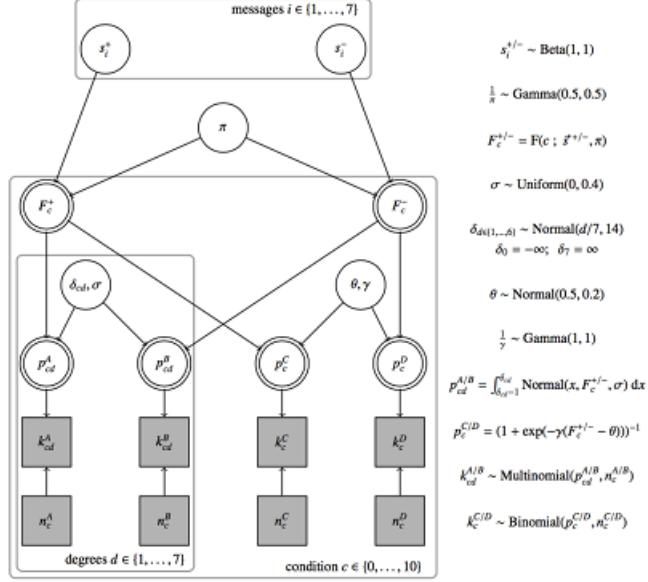


Figure 9.10.: Graphical notation hierarchical Beta-Binomial Model - One group

## 9.3. Further examples

### 9.3.1. Difference between two groups

In the introductory example we asked for the underlying probability  $\theta$  of a single coin that was flipped repeatedly. Consider now, that a second coin  $y_2$  is introduced. One question that arises might be for example: *How different are the biases of the two coins?*

```
#simulate flips of two coins
sample.space <- c(0,1)
##First coin:
theta1 <- 0.5
X1 <- 30
n1 <- 100
k1 <- 0

for (i in 1: n1) {
  k1[i] <- sum(sample(sample.space, size = X1, replace = TRUE,
                      prob = c(theta1, 1 - theta1)))
}
##Second coin:
theta2 <- 0.7
X2 <- 30
n2 <- 100
```

```

k2 <- 0                      # number of heads [initialization]

## repeat experiment N-times
for (i in 1:n2) {
  k2[i] <- sum(sample.space, size = X2, replace = TRUE,
               prob = c(theta2, 1 - theta2)))
}

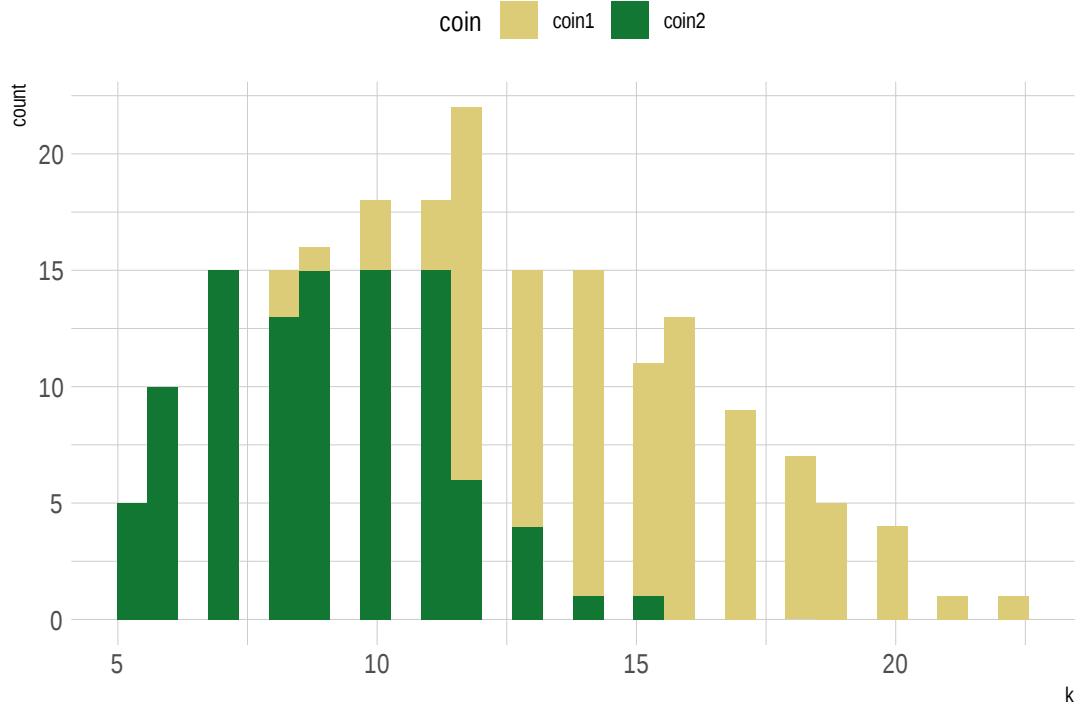
## show results in a tibble
coin.flip2 <- tibble("coin" = c(replicate(n1, "coin1"), replicate(n2, "coin2")),
                     "n" = c(seq(from=1, to=n1, by=1), seq(from=1, to=n2, by=1)),
                     "k" = c(k1,k2),
                     "x" = c(replicate(n1,X1), replicate(n2,X2)))
) %>%
  print()

## # A tibble: 200 x 4
##   coin     n     k     x
##   <chr> <dbl> <dbl> <dbl>
## 1 coin1     1     14    30
## 2 coin1     2     13    30
## 3 coin1     3     18    30
## 4 coin1     4     18    30
## 5 coin1     5     13    30
## 6 coin1     6     14    30
## 7 coin1     7     14    30
## 8 coin1     8     17    30
## 9 coin1     9     20    30
## 10 coin1    10    12    30
## # ... with 190 more rows

#Plotting the observed results
ggplot(data=coin.flip2,mapping = aes(x=k, fill=coin ))+
  geom_histogram()

```

## 9. Models



### 9.3.1.1. Conceptual steps for modeling

We suppose that the underlying probabilities of the two coins correspond to *different* latent variables  $\theta_1$  and  $\theta_2$ .

**First step** is again the *identification of the relevant variables* according to the research question. As already indicated for the “one coin” example we have:

- the observed number of heads  $k_1$  and  $k_2$  (for each coin, respectively), which is influenced by
- the number of observations  $n_1$  and  $n_2$  and by
- the underlying probabilities  $\theta_1$  and  $\theta_2$ .

Furthermore, from a conceptional perspective, we are interested in the *difference between the coin biases*. Therefore a further variable will be introduced  $\delta$ , defined by:

$$\delta = \theta_1 - \theta_2.$$

The *distributional assumptions*, according to the **second and third step**, can be adopted from the “one coin” example, such that the graphical notation (including the textual notation) can be denoted as follows:

### 9.3.1.2. Notation Beta-Binomial Model - Two Groups

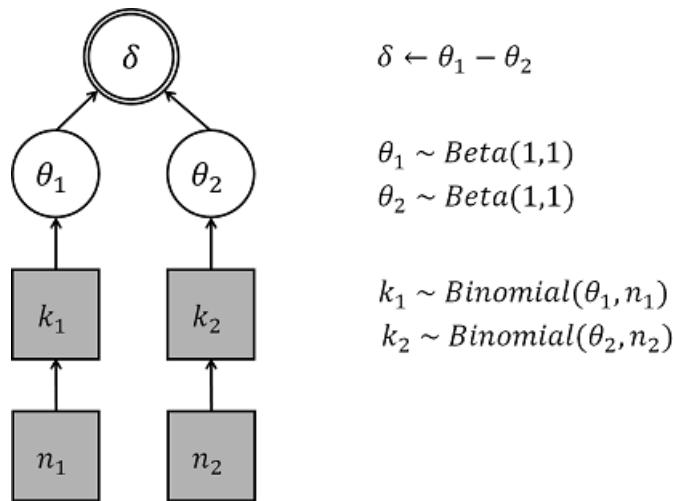


Figure 9.11.: Graphical notation Beta-Binomial Model - Two groups

### 9.3.2. Simple linear regression with one metric predictor

The following example originates from a data set in which speed of cars and the distance taken to stop was recorded. It is a simple data set good for introducing the basic ideas for simple linear regression.

```
#The "cars" data set
data(cars)
#take a look at the variables included in the data set
str(cars)

## 'data.frame':   50 obs. of  2 variables:
## $ speed: num  4 4 7 7 8 9 10 10 10 11 ...
## $ dist : num  2 10 4 22 16 10 18 26 34 17 ...
```

One possible question could be how much the stopping distance increases when the speed of a car increases.

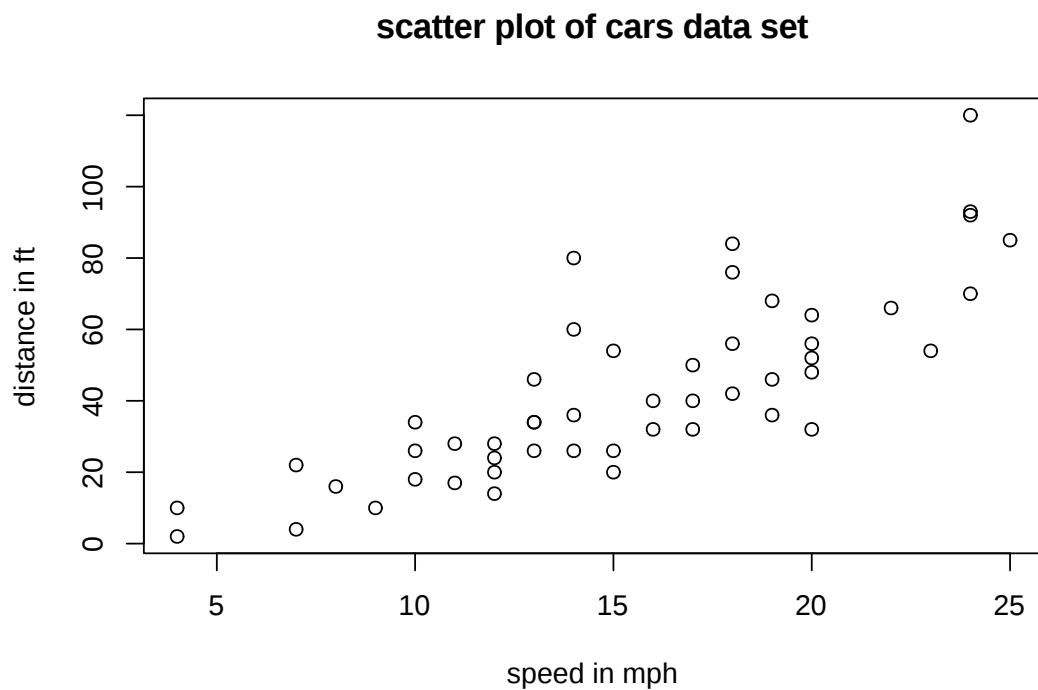
#### 9.3.2.1. Conceptual steps for modeling

First step is to **identify the relevant variables**. In this case these are “speed” measured in mph and “distance” measured in ft, thus, both variables are metric variables. As distance will be predicted from speed. The *predicted variable* is “distance” and the

## 9. Models

*predictor variable* is “speed”. A scatter plot can visualize a possible relationship between both variables.

```
plot(x=cars$speed, y=cars$dist, type="p", main="scatter plot of cars data set",
      ylab="distance in ft", xlab="speed in mph")
```



Next step is to define a **descriptive model of the data**. According to the scatter plot it is not too absurd to think that distance might be proportional to speed. Therefore, a linear relationship between both variables can be assumed, where speed is used in order to predict distance. But how can the distribution of the predicted variable “distance” be described? The following plot shows in blue the density of the actual distance values.

```
#density of distance values in blue
#(in black simulation of a normal distribution)
dens(cars$dist, col="blue", norm.comp = TRUE, main="Distribution of distance",
      xlab="distance in ft")
```

Although the distribution of “distance” values is not identical to the corresponding normal distribution, it can be assumed that the values follow a *normal distribution*. The underlying consideration is that the distance values  $y_i$  are distributed randomly

according to a normal distribution around the predicted value  $\hat{y}$  and with a standard deviation denoted with  $\sigma$ . This can be denoted as:

$$y_i \sim Normal(\mu, \sigma).$$

The index  $i$  indicates each element (i.e. car) of the list  $y$ , which in turn is the list of distances.

In the third step, a Bayesian perspective is taken the **prior knowledge** (before seeing the data) has to be defined. The parameters of the current model are the predicted value  $\mu$  and the standard deviation  $\sigma$ . For the parameter  $\mu$  a normal distribution can be assumed with parameters that reflect the estimated values from the sample.

```
#descriptive statistics from the sample
tibble(variables=c("speed", "distance"),
       mean=c(mean(cars$speed), mean(cars$dist)),
       sigma = c(sd(cars$speed), sd(cars$dist)))

## # A tibble: 2 x 3
##   variables  mean sigma
##   <chr>     <dbl> <dbl>
## 1 speed      15.4  5.29
## 2 distance   43.0  25.8
```

$$\mu \sim Normal(43, 26)$$

For the standard deviation  $\sigma$  a uniform distribution is assumed:

$$\sigma \sim Uniform(0, 40)$$

### 9.3.2.2. Excusus: Identically and independently distributed (*iid*)

The short model description  $y_i \sim Normal(\mu, \sigma)$  incorporates often already an assumption about the distribution of distance-values: They are *identically and independently distributed*. Often the abbreviation *iid* can be found for this assumption:

$$y_i \stackrel{iid}{\sim} Normal(\mu, \sigma).$$

The abbreviation *iid* indicates that each value  $y_i$  has the same probability function, independent of the other  $y$  values and using the same parameters (McElreath 2015). This is hardly ever true (why hierarchical modeling is very attractive). For example, thinking about the cars in the current example data set. Some cars may be of different types or even the same type but different batches. But the question is: Is this underlying

## 9. Models

dependency relevant for the model? If yes, this information has to be added in the model (e.g. in form of a hierarchical model). Jaynes states it as follows: “*The onus is always on the user to make sure that all information, which his common sense tells him is relevant to the problem, is actually incorporated into the equations, (...).*”(Jaynes 2003, 339). But if one does not know any relevant underlying relationships the most conservative distribution to use is *iid*. Note, that the stated assumptions define how the model represents a problem and not how the world should be understood. For example, there might exist underlying correlations but on the overall distribution there influence tends towards zero. In such cases it remains useful to assume *iid* (McElreath 2015).

### 9.3.3. Notation Simple Regression model

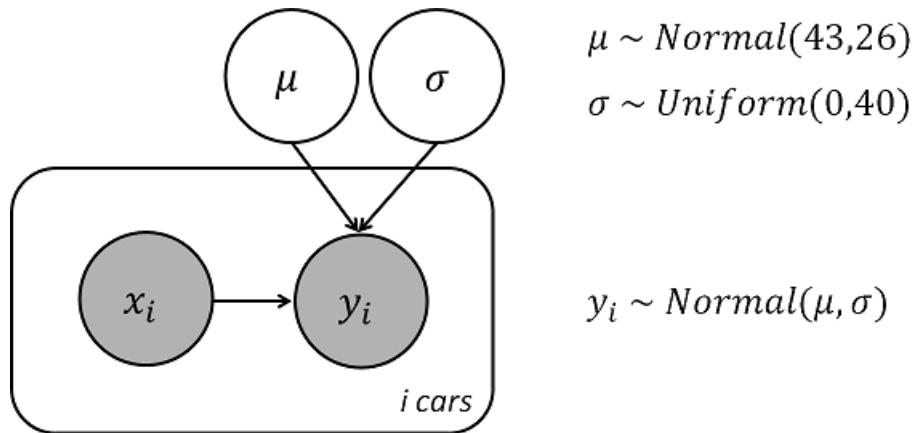


Figure 9.12.: Graphical notation Simple Regression model

## 9.4. Further elaboration on modeling (in anticipation of the topic “estimation”)

### 9.4.1. Beta-Binomial model - one group (revisited)

Sofar the existence of the underlying probability  $\theta$  for observing head as outcome of a coin flip has been discussed. But the estimation of  $\theta$  has been ignored until yet. Although “estimation” will be topic of next chapter, it is helpful at this point to discuss the introduced models further. In order to estimate  $\theta$  parameter(s) are needed. When it comes to estimation exactly this/these parameter(s) will be the result(s), therefore is important to see already the connection to the models that were developed in this chapter.

For the coin flip example the value of interest is the underlying probability, thus, only

#### 9.4. Further elaboration on modeling (in anticipation of the topic “estimation”)

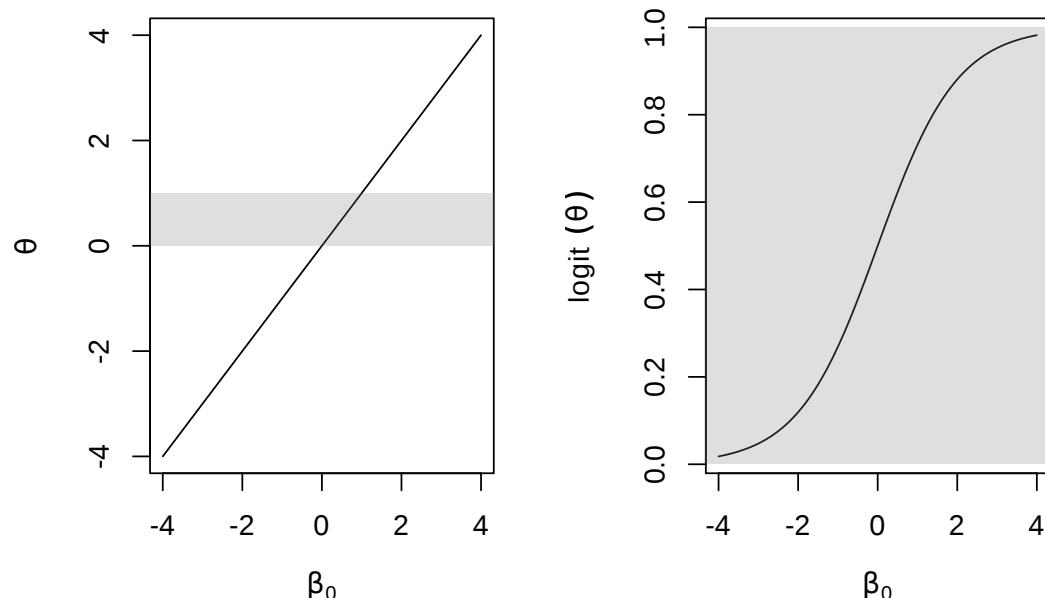
one parameter is needed:  $\beta_0$ . (Note: Latin letters are used when we refer to the sample, Greek letters are used when we refer to the population.)

How is  $\beta_0$  linked to the latent variable  $\theta$ ?

Considering for example the simplest case: a *linear relationship* (see next plot left side). The problem which arises at this point is that  $\theta$  represents a probability, and is therefore bounded to the range 0-1 (grey shaded area).

```
#Different relationships between the parameter and expected value
x <- seq(from=-4, to=4, length.out = 100)
y <- x                         #linear relationship
y.log <- inv_logit_scaled(x)   #logistic relationship

par(mfrow = c(1, 2))    #set both plot beside each other
plot(x, y, type="l", ylab=expression(theta), xlab=expression(beta[0]))
rect(-5, 0, 5, 1, col = rgb(0.5, 0.5, 0.5, 1/4), border = NA)
plot(x, y.log, type="l", ylab=expression(logit~(theta)), xlab=expression(beta[0]))
rect(-5, 0, 5, 1, col = rgb(0.5, 0.5, 0.5, 1/4), border = NA)
```



A mathematical transformation is needed such that the parameter  $\beta_0$  can take any value while  $\theta$  is bounded to the range 0-1. One transformation that offers exactly this possibility is the *logit link function* (see above plot right side)

## 9. Models

$$\text{logit}(\theta) = \beta_0.$$

As the underlying assumption maps the parameter to the latent variable  $\theta$  (and not the other way around) from a conceptional point of view the *inverse link function* is more appropriate, which is the *logistic link* in this case:

$$\theta = \text{logistic}(\beta_0).$$

It is defined as

$$\theta = \frac{\exp(\beta_0)}{1 + \exp(\beta_0)}.$$

Both expression, *logit* and *logistic* link achieve mathematically the same result but it is conceptually just a different matter of emphasis (Kruschke 2015).

### 9.4.1.1. Notation of beta-binomial model - one group (revisited)

The current descriptive model incorporates the idea that parameter  $\beta_0$  is estimated from the given sample. It defines the latent variable  $\theta$ . The parameter is mapped to  $\theta$  by a logistic link function. The underlying probability  $\theta$  designates the observed number of upcoming heads. The number of upcoming heads in turn, is assumed to be distributed as Binomial distribution.

### 9.4.2. Beta-Binomial model - two groups (revisited)

In the above model for two coins the latent variable  $\delta$  was already introduced. It is defined by the difference between the underlying probabilities  $\theta_1 - \theta_2$ . Which parameters should be used in order to estimate the difference between both groups? As we will see, it turns out that the same mathematical form can be used, as one would use for simple linear regression:

$$\theta_j = \beta_0 + \beta_1 * X_{\text{Group}_j},$$

with  $X_{\text{Group}_j} = \begin{cases} 0, & \text{if coin 2,} \\ 1, & \text{if coin 1.} \end{cases}$

Considering *coin 2*, the above equation would result in

$$\theta_2 = \beta_0,$$

which is the *intercept* and indicates the proportion of head coming up for coin 2.

Considering by contrast coin 1, then the equation would result in:

$$\theta_1 = \beta_0 + \beta_1.$$

#### 9.4. Further elaboration on modeling (in anticipation of the topic “estimation”)

The proportion of coming up head for coin 1 has to be calculated by summing up the *intercept*  $\beta_0$  and the *slope*  $\beta_1$ .

Taken together: *What is the interpretation of the slope  $\beta_1$ ?* The difference  $\delta = \theta_1 - \theta_2$  is

$$\theta_1 - \theta_2 = (\beta_0 + \beta_1) - \beta_0 = \beta_1 = \delta,$$

the slope  $\beta_1$ , thus, we can see that this parameterization enables us to estimate the difference between two groups. When it comes to estimation and interpretation the results will be the intercept  $b_0$  and the slope  $b_1$ .

##### 9.4.3. Simple linear regression model (revisited)



## 10. Parameter inference

- MLE vs posterior
- confidence intervals
- credible intervals
- briefly: algorithms for MLE & Bayesian inference



# **11. Hypothesis Testing**

- binomial test
- t-test
- ANOVA
- linear regression



## 12. Model Comparison

- AIC
- likelihood ratio test
- Bayes factor



## **13. Bayesian hypothesis testing**

- testing via Bayesian posterior inference
- testing via model comparison



## 14. Model criticism

- prior and posterior predictives
- visual predictive checks
- prior/posterior predictive  $p$ -values



## **Part IV.**

# **Applied (generalized) linear modeling**



## 15. Simple linear regression

- “multiple” = “more than one predictor”
  - interactions
  - collinearity
- categorical predictors
  - relation to t-test and ANOVA
  - different coding schemes
- robust regression



## **16. Logistic regression**

to do



## 17. Multinomial regression

todo



## **18. Ordinal regression**

todo



## **19. Hierarchical regression**

todo



# A. Further useful material

## A.1. Material on *Introduction to Probability*:

- “Introduction to Probability” by J.K. Blitzstein and J. Hwang (Blitzstein and Hwang 2014)
- “Probability Theory: The Logic of Science” by E.T. Jaynes (Jaynes 2003)

## A.2. Material on *Bayesian Data Analysis*:

- “Doing Bayesian data analysis: A tutorial with R, JAGS, and Stan” by J. Kruschke (Kruschke 2015)
- “Bayesian Data Analysis” by A. Gelman et al. (Gelman et al. 2013)
- “Statistical Rethinking: A Bayesian Course with Examples in R and Stan” by R. McElreath (McElreath 2015)
  - webbook based on McElreath’s book: Statistical Rethinking with brms, ggplot2, and the tidyverse by Solomon Kurz

## A.3. Material on *frequentist statistics*:

- “Statistics for LinguistsL: An introduction using R”, by B. Winter (Winter 2019-)

## A.4. Material on *R, tidyverse, etc.*:

- official R manual: An Introduction to R
- “R for Data Science: Import, Tidy, Transform, Visualize, and Model Data” by H. Wickham and G. Grolemund (Wickham and Grolemund 2016)
- RStudio’s Cheat Sheets
- “Data Visualization” by K. Healy (Healy 2018)
- webbook Learning Statistics with R by Danielle Navarro
- webbook with focus on visualization: Data Science for Psychologists by Hansjörg Neth

## A. Further useful material

### A.5. Further information for RStudio

- *Keyboard shortcuts* for Windows and Mac in RStudio: “Tools -> Keyboard Shortcuts Help” or also on the RStudio support site

### A.6. Resources on WebPPL

- official website
- documentation
- Bayesian Data Analysis using Probabilistic Programs: Statistics as pottery by webbook on BDA with WebPPL by MH Tessler

## B. Common probability distributions

This chapter summarizes common probability distributions, which occur at central places in this book.

### B.1. Selected continuous distributions of random variables

#### B.1.1. Normal distribution

One of the most important distribution families is the *gaussian* or *normal family* because it fits many natural phenomena. Furthermore the sampling distributions of many estimators depend on the normal distribution. On the one hand because they are derived from normally distributed random variables or on the other hand because they can be asymptotically approximated by a normal distribution for large samples (*Central limit theorem*).

Distributions of the normal family are symmetric with range  $(-\infty, +\infty)$  and have two parameters  $\mu$  and  $\sigma$  that are referred to, respectively, as the *mean* and the *standard deviation* of the normal random variable. These parameters are examples of *location* and *scale* parameters. The normal distribution is located at  $\mu$  and its width is scaled by choice of  $\sigma$ . The distribution is symmetric with most observations lying around the central peak  $\mu$  and more extreme values are further away depending on  $\sigma$ .

$$X \sim \text{Normal}(\mu, \sigma^2)$$

Fig.~B.1 shows the probability density function of three normal distributed random variables with different parameters. Fig.~B.2 shows the corresponding cumulative function of the three normal distributions.

```
rv_normal <- tibble(
  x = seq(from = -15, to = 15, by = .01),
  y1 = dnorm(x),
  y2 = dnorm(x, mean = 2, sd = 2),
  y3 = dnorm(x, mean = -2, sd = 3)
) %>%
  pivot_longer(cols = starts_with("y"),
              names_to = "parameter",
```

## B. Common probability distributions

```

values_to = "y") %>%
mutate(
  parameter = case_when(parameter == "y1" ~ "(0,1)",
                         parameter == "y2" ~ "(2,2)",
                         parameter == "y3" ~ "(-2,3)")
)

ggplot(rv_normal, aes(x, y, color = parameter)) +
  geom_line(size = 2) +
  labs(color = "X ~ Normal", y = "Density")

```

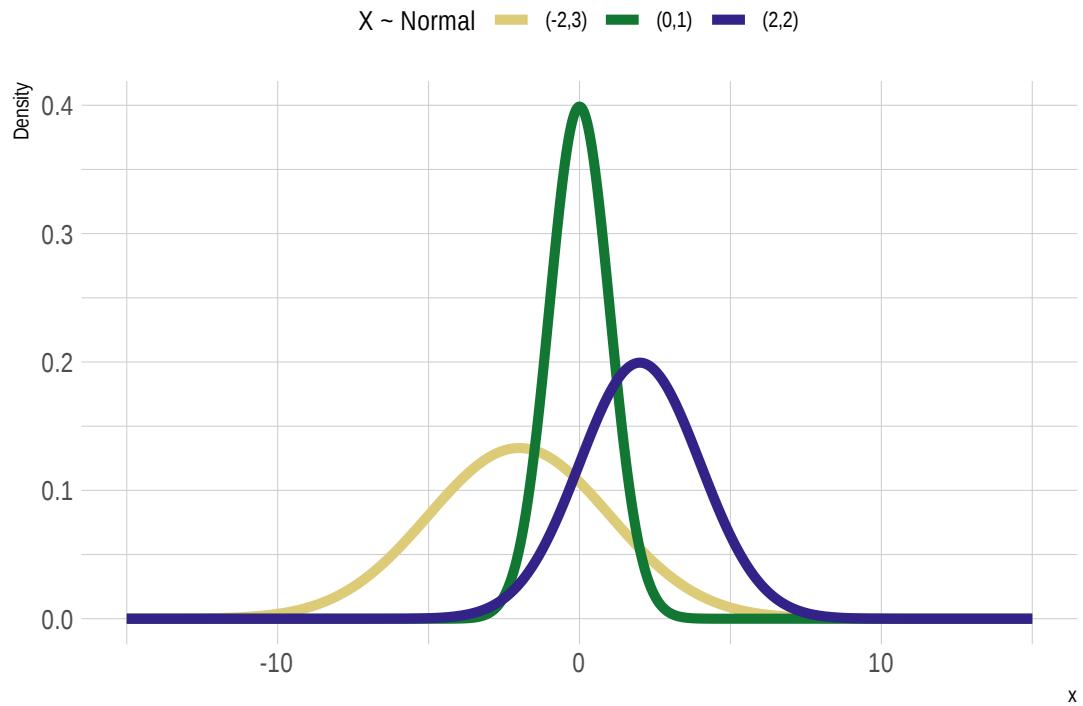


Figure B.1.: Examples of probability density function of normal distributions.

```

rv_normal %>%
  group_by(parameter) %>%
  mutate(
    cum_y = cumsum(y)/sum(y)
  ) %>%
  ungroup() %>%
  ggplot(aes(x, cum_y, color = parameter)) +

```

### B.1. Selected continuous distributions of random variables

```
geom_line(size = 2) +
labs(color = "X ~ Normal", y = "y")
```

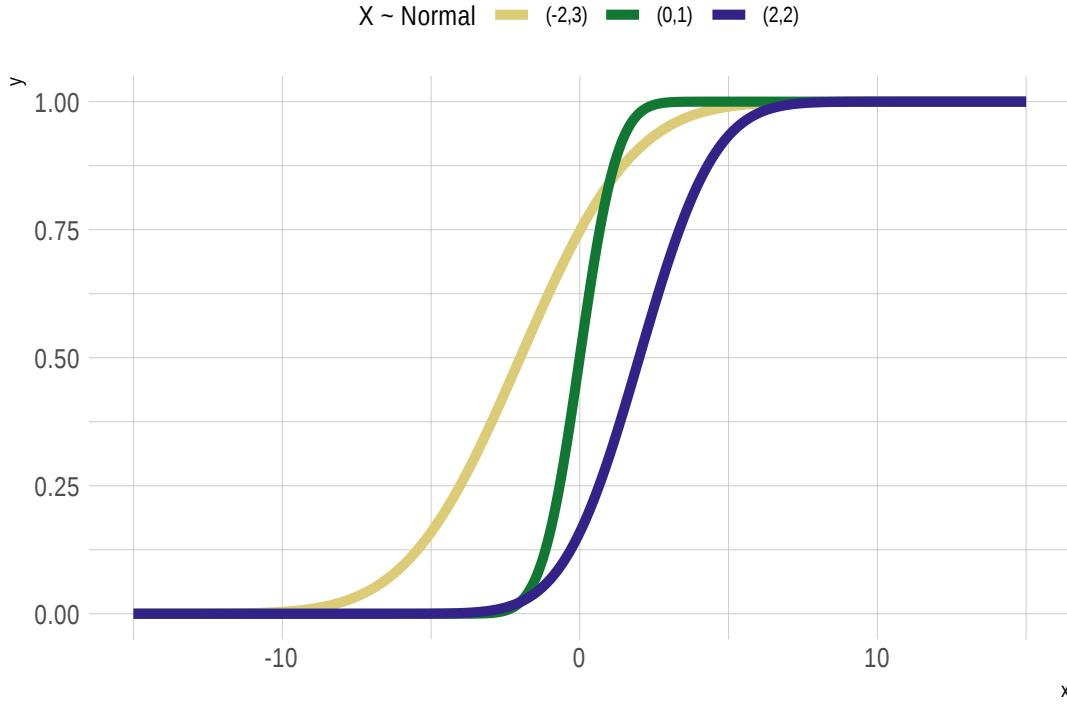


Figure B.2.: Examples of the cumulative distribution function of normal distributions corresponding to the previous probability density functions.

A special case of normal distributed random variables is the *standard normal* distributed variable with  $\mu = 0$  and  $\sigma = 1$ :  $Y \sim \text{Normal}(0, 1)$ . Each normal distribution can be converted into a standard normal distribution by *z-standardization* (see equation below). The advantage of standardization is that values from different scales can be compared, because they become *scale independent* by z-transformation.

#### Probability density function

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-0.5\left(\frac{x-\mu}{\sigma}\right)^2\right)$$

#### Cumulative distribution function

$$F(x) = \int_{-\infty}^x f(t)dt$$

## B. Common probability distributions

**Expected value**  $E(X) = \mu$

**Variance**  $Var(X) = \sigma^2$

**Z-transformation**  $Z = \frac{X-\mu}{\sigma}$

\*\*Deviation and \*Coverage\*\* The normal distribution is often associated with the *68-95-99.7 rule*. The values refer to the probability of a random data point landing within *one, two or three* standard deviations of the mean (Fig.~B.3 depicts these three intervals). For example, about 68% of values drawn from a normal distribution are within one standard deviation  $\sigma$  away from the mean  $\mu$ .

- $P(\mu - \sigma \leq X \leq \mu + \sigma) = 0.6827$
- $P(\mu - 2\sigma \leq X \leq \mu + 2\sigma) = 0.9545$
- $P(\mu - 3\sigma \leq X \leq \mu + 3\sigma) = 0.9973$

```
# plot normal distribution with intervals
ggplot(NULL, aes(x = c(-10, 10))) +
  # plot area under the curve
  stat_function(fun = dnorm, args = list(mean = 0, sd = 2),
                geom = "area",
                fill = project_colors[1],
                xlim = c(-6, 6)) +
  stat_function(fun = dnorm, args = list(mean = 0, sd = 2),
                geom = "area",
                fill = project_colors[2],
                xlim = c(-4, 4)) +
  stat_function(fun = dnorm, args = list(mean = 0, sd = 2),
                geom = "area",
                fill = project_colors[3],
                xlim = c(-2, 2)) +
  # plot the curve
  stat_function(fun = dnorm, args = list(mean = 0, sd = 2),
                geom = "line",
                xlim = c(-10, 10),
                size = 2) +
  # scale x-axis
  xlim(-10, 10) +
  # label x-axis
  xlab("X") +
  # label ticks of x-axis
  scale_x_continuous(breaks = c(-6,-4,-2,0,2,4,6),
                     labels = c(expression(-3~sigma), expression(-2~sigma),
                               expression(-sigma), "0", expression(sigma),
                               expression(2~sigma), expression(3~sigma)))
```

*B.1. Selected continuous distributions of random variables*

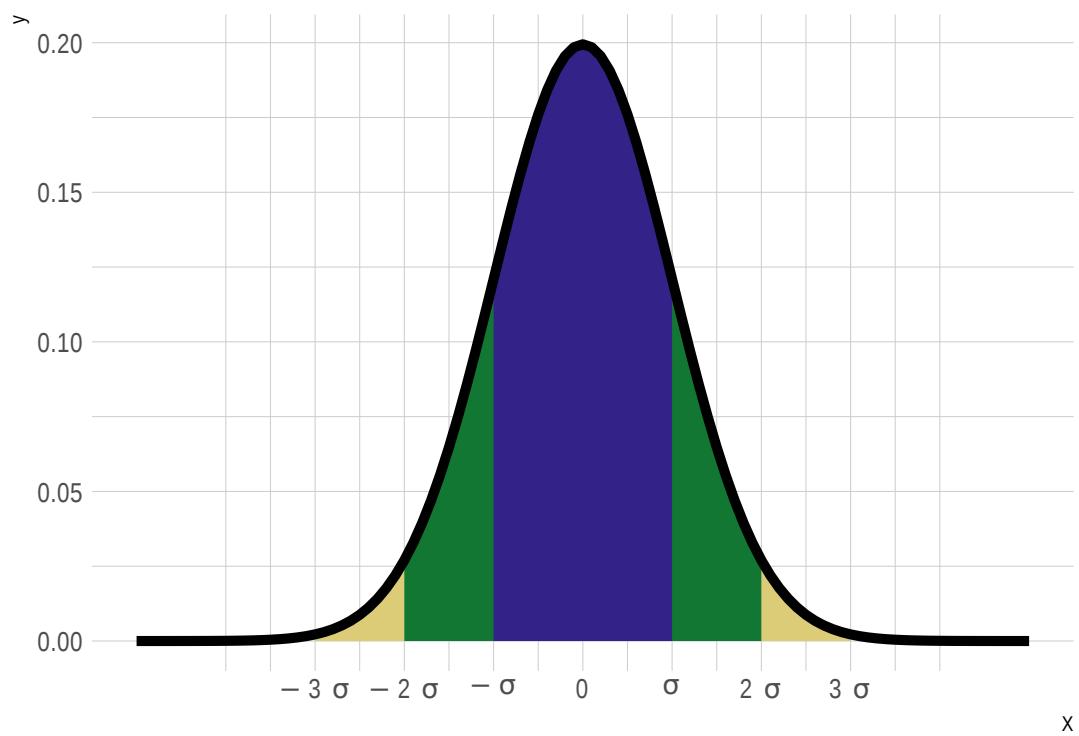


Figure B.3.: Coverage of normal distribution

## B. Common probability distributions

### Linear transformations

1. If  $X \sim Normal(\mu, \sigma^2)$  is linear transformed by  $Y = a*X + b$ , then the new random variable is again normal distributed with  $Y \sim Normal(a\mu + b, a^2\sigma^2)$ .
2. Are  $X \sim Normal(\mu_x, \sigma_x^2)$  and  $Y \sim Normal(\mu_y, \sigma_y^2)$  normal distributed and independent, then their sum is again normal distributed with  $X + Y \sim Normal(\mu_x + \mu_y, \sigma_x^2 + \sigma_y^2)$ .

#### B.1.1.1. Hands On

```
knitr::include_app("https://istats.shinyapps.io/NormalDist/", height = "800px")
```

App taken from <http://www.artofstat.com/webapps.html> (Klingenberg, n.d.)

### B.1.2. Chi-squared distribution

The  $\chi^2$ -distribution is widely used in hypothesis testing in inferential statistics, because many test statistics are approximately distributed as  $\chi^2$ -distribution.

The  $\chi^2$ -distribution is directly related to the standard normal distribution: The sum of  $n$  independent and standard normal distributed random variables  $X_1, X_2, \dots, X_n$  is distributed according to a  $\chi^2$  distribution with  $n$  *degrees of freedom*:

$$Y = X_1^2 + X_2^2 + \dots + X_n^2.$$

The  $\chi^2$  distribution is a skew probability distribution with range  $[0, +\infty)$  and only one parameter:  $n$ , the *degrees of freedom*: (If  $n = 1$ , then  $(0, +\infty)$ .)

$$X \sim \chi^2(n).$$

Fig.~B.4 shows the probability density function of three chi-squared distributed random variables with different values for the parameter. Notice, that with increasing degrees of freedom the chi-squared distribution approximates the normal distribution. For  $n \geq 30$  the chi-squared distribution can be approximated by a normal distribution. Fig.~B.5 shows the corresponding cumulative function of the three chi-squared density distributions.

```
rv_chisq <- tibble(
  x = seq(from = 0, to = 20, by = .01),
  y1 = dchisq(x, df = 2),
  y2 = dchisq(x, df = 4),
```

### B.1. Selected continuous distributions of random variables

```

y3 = dchisq(x, df = 9)
) %>%
pivot_longer(cols = starts_with("y"),
              names_to = "parameter",
              values_to = "y") %>%
mutate(
  parameter = case_when(parameter == "y1" ~ "(2)",
                         parameter == "y2" ~ "(4)",
                         parameter == "y3" ~ "(9)")
)
# dist plot
ggplot(rv_chisq, aes(x, y, color = parameter)) +
  geom_line(size = 2) +
  labs(color = "X ~ Chi-Squared", y = "Density")

```

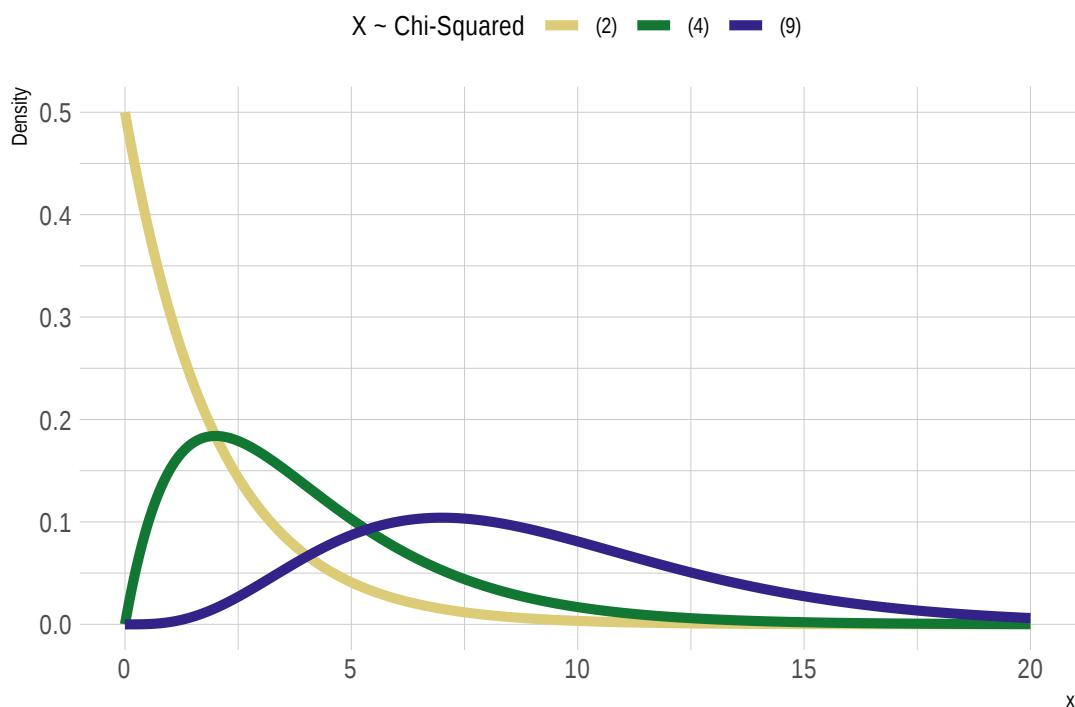


Figure B.4.: Examples of probability density function of chi-squared distributions.

## B. Common probability distributions

```
rv_chisq %>%
  group_by(parameter) %>%
  mutate(
    cum_y = cumsum(y)/sum(y)
  ) %>%
  ungroup() %>%
  ggplot(aes(x, cum_y, color = parameter)) +
  geom_line(size = 2) +
  labs(color = "X ~ Chi-Squared", y = "y")
```

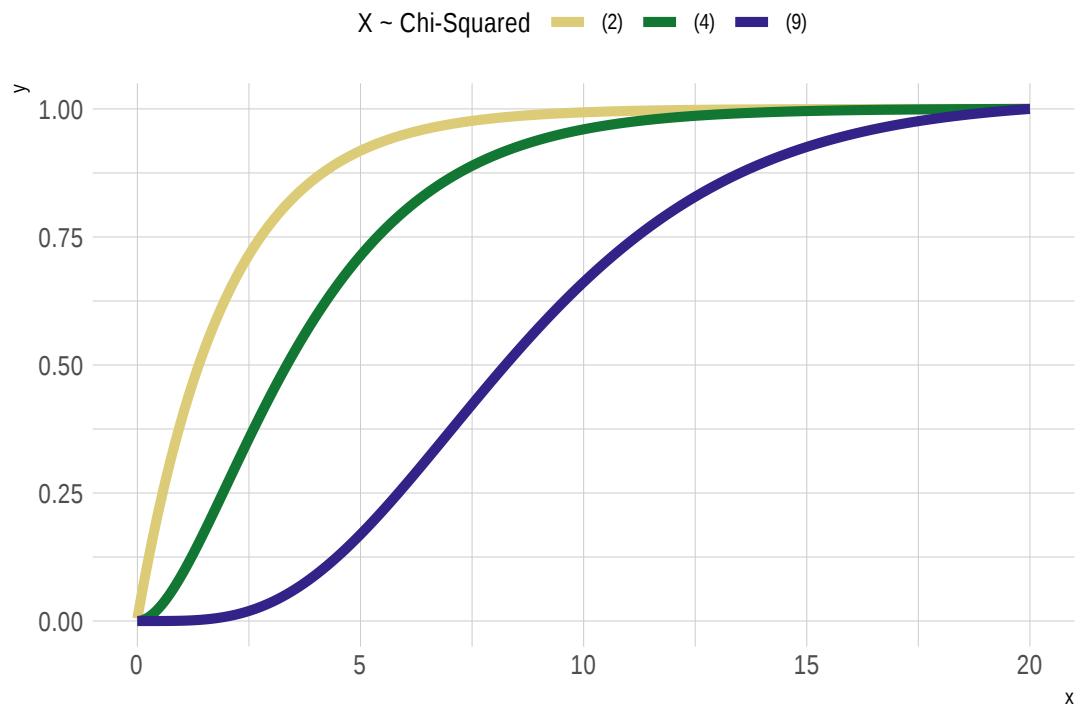


Figure B.5.: Examples of the cumulative distribution function of chi-squared distributions corresponding to the previous probability density functions.

## Probability density function

$$f(x) = \begin{cases} \frac{x^{\frac{n}{2}-1} e^{-\frac{x}{2}}}{2^{\frac{n}{2}} \Gamma(\frac{n}{2})} & \text{for } x > 0, \\ 0 & \text{otherwise.} \end{cases}$$

Where  $\Gamma(\frac{n}{2})$  denotes the Gamma function.

### Cumulative distribution function

$$F(x) = \frac{\gamma(\frac{n}{2}, \frac{x}{2})}{\Gamma_{\frac{n}{2}}}$$

with  $\gamma(s, t)$  being the lower incomplete gamma function:

$$\gamma(s, t) = \int_0^t t^{s-1} e^{-t} dt.$$

**Expected value**  $E(X) = n$

**Variance**  $Var(X) = 2n$

**Transformations** The sum of two  $\chi^2$ -distributed random variables  $X \sim \chi^2(m)$  and  $Y \sim \chi^2(n)$  is again a *chi*<sup>2</sup>-distributed random variable  $X + Y = \chi^2(m + n)$ .

#### B.1.2.1. Hands On

```
knitr:::include_app("https://istats.shinyapps.io/ChisqDist/", height = "800px")
```

App taken from <http://www.artofstat.com/webapps.html> (Klingenberg, n.d.)

### B.1.3. F distribution

The F distribution, named after R.A. Fisher, is used in particular in regression and variance analysis. It is defined by the ratio of two *chi*<sup>2</sup>-distributed random variables  $X \sim \chi^2(m)$  and  $Y \sim \chi^2(n)$ , each divided by its degree of freedom:

$$F = \frac{\frac{X}{m}}{\frac{Y}{n}}.$$

The F distribution is a continuous skew probability distribution with range  $(0, +\infty)$  and two parameters  $m$  and  $n$ , corresponding to the degrees of freedom of the two *chi*<sup>2</sup>-distributed random variables:

$$X \sim F(m, n).$$

Fig.~B.6 shows the probability density function of three F distributed random variables with different parameter values. For a small number of degrees of freedom the density distribution is skewed to the left side. When the number increases, the density distribution gets more and more symmetric. Fig.~B.7 shows the corresponding cumulative function of the three F density distributions.

## B. Common probability distributions

```

rv_F <- tibble(
  x = seq(from = 0, to = 7, by = .01),
  y1 = df(x, df1 = 2, df2 = 4),
  y2 = df(x, df1 = 4, df2 = 6),
  y3 = df(x, df1 = 12, df2 = 12)
) %>%
  pivot_longer(cols = starts_with("y"),
               names_to = "parameter",
               values_to = "y") %>%
  mutate(
    parameter = case_when(parameter == "y1" ~ "(2,4)",
                           parameter == "y2" ~ "(4,6)",
                           parameter == "y3" ~ "(12,12)")
  )

# dist plot
ggplot(rv_F, aes(x, y, color = parameter)) +
  geom_line(size = 2) +
  labs(color = "X ~ F", y = "Density")

rv_F %>%
  group_by(parameter) %>%
  mutate(
    cum_y = cumsum(y)/sum(y)
  ) %>%
  ungroup() %>%
  ggplot(aes(x, cum_y, color = parameter)) +
  geom_line(size = 2) +
  labs(color = "X ~ F", y = "y")

```

### Probability density function

$$F(x) = m^{\frac{m}{2}} n^{\frac{n}{2}} \cdot \frac{\Gamma(\frac{m+n}{2})}{\Gamma(\frac{m}{2})\Gamma(\frac{n}{2})} \cdot \frac{x^{\frac{m}{2}-1}}{(mx+n)^{\frac{m+n}{2}}} \text{ for } x > 0.$$

Where  $\Gamma(x)$  denotes the gamma function.

### Cumulative distribution function

$$F(x) = I\left(\frac{m \cdot x}{m \cdot x + n}, \frac{m}{2}, \frac{n}{2}\right),$$

with  $I(z, a, b)$  being the regularized incomplete beta function:

$$I(z, a, b) = \frac{1}{B(a, b)} \cdot \int_0^z t^{a-1} (1-t)^{b-1} dt.$$

*B.1. Selected continuous distributions of random variables*

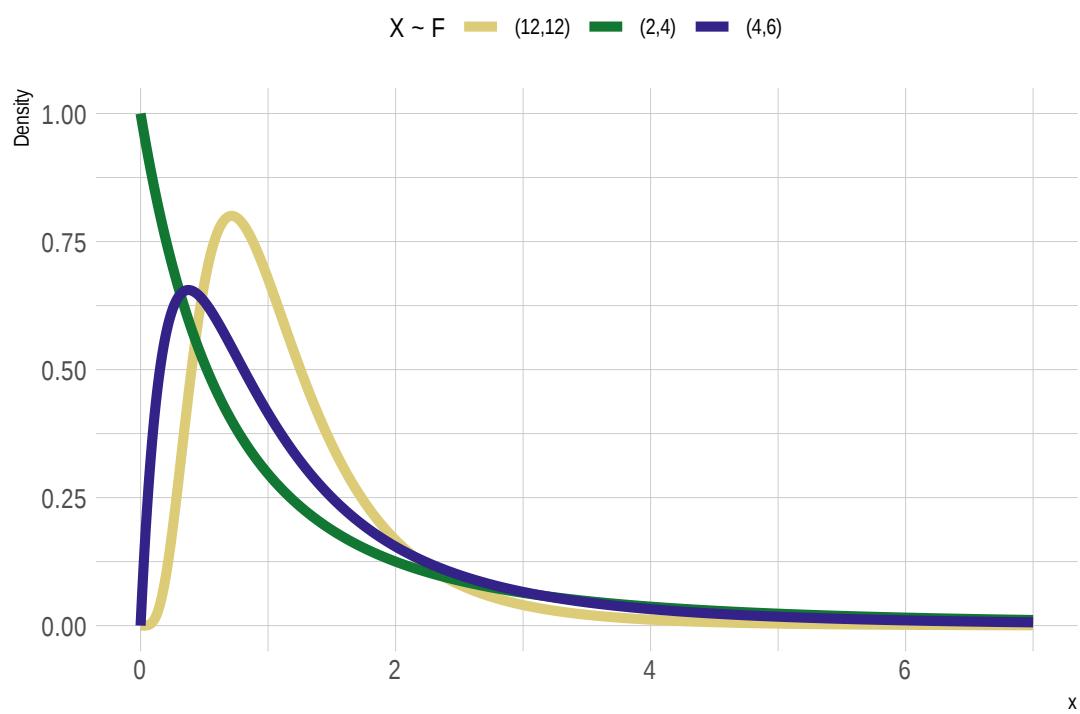


Figure B.6.: Examples of probability density function of F distributions.

*B. Common probability distributions*

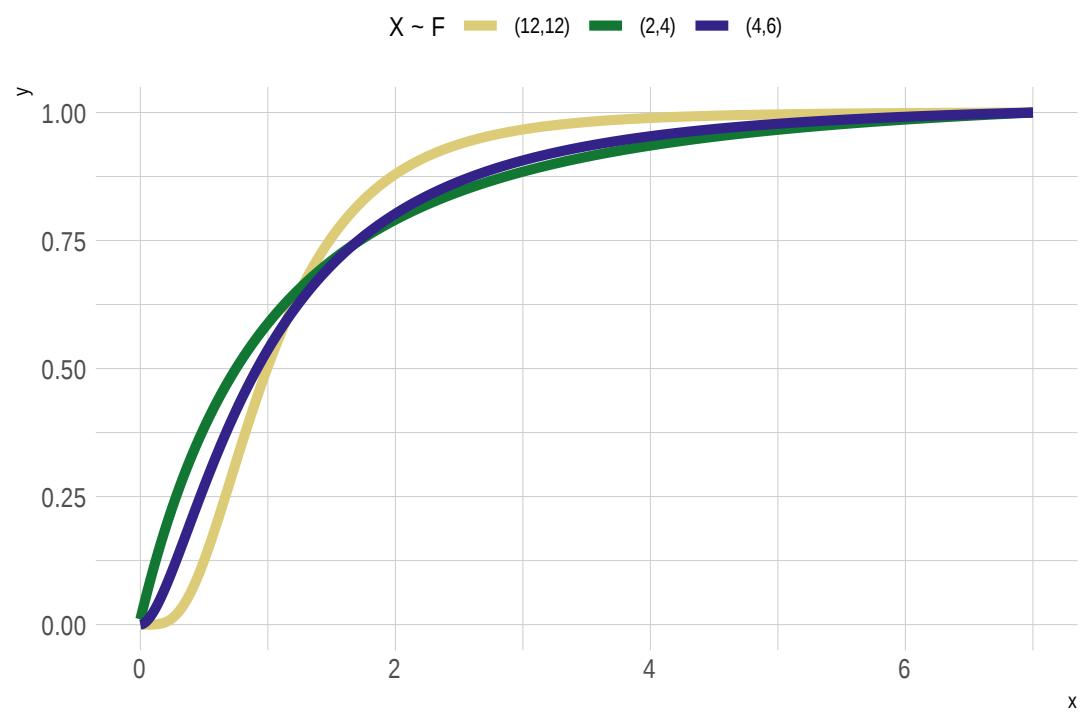


Figure B.7.: Examples of the cumulative distribution function of F distributions corresponding to the previous probability density functions.

## B.1. Selected continuous distributions of random variables

**Expected value**  $E(X) = \frac{n}{n-2}$  (for  $n \geq 3$ )

**Variance**  $Var(X) = \frac{2n^2(n+m-2)}{m(n-4)(n-2)^2}$  (for  $n \geq 5$ )

### B.1.3.1. Hands On

```
knitr::include_app("https://istats.shinyapps.io/FDist/", height = "800px")
```

App taken from <http://www.artofstat.com/webapps.html> (Klingenberg, n.d.)

### B.1.4. Student t-distribution

The t or student-t distribution was discovered by William S. Gosset in 1908 (Vallverdú 2016), who published his work under the pseudonym “student”. He worked at the Guinness factory and had to deal with the problem of small sample sizes. This challenge resulted finally in the t distribution. Accordingly, this distribution is used in particular when the sample size is small and the variance unknown, which is often the case in reality. Its shape resembles the normal bell shape and has a peak at zero, but the t distribution is a bit lower and wider (bigger tails) than the normal distribution.

The t distribution consists of a standard normal distributed random variable  $X \sim Normal(0, 1)$  and a  $\chi^2$ -distributed random variable  $Y \sim \chi^2(n)$  ( $X$  and  $Y$  are independent):

$$T = \frac{X}{\sqrt{\frac{Y}{n}}}.$$

The t distribution has range  $(-\infty, +\infty)$  and one parameter  $n$ , the degrees of freedom. The degrees of freedom can be calculated by the sample size  $n$  minus one:

$$X \sim t(n).$$

Fig.~B.8 shows the probability density function of three t distributed random variables with different parameters. Notice that for small degrees of freedom  $n$ , the t-distribution has bigger tails. This is because the t distribution was specially designed to provide more conservative test results when analyzing small samples. When the degrees of freedom increases, the t distribution approaches a normal distribution. For  $n \geq 30$  this approximation is quite good. Fig.~B.9 shows the corresponding cumulative function of the three t density distributions.

## B. Common probability distributions

```

rv_student <- tibble(
  x = seq(from = -6, to = 6, by = .01),
  y1 = dt(x, df = 1),
  y2 = dt(x, df = 2),
  y3 = dt(x, df = 10)
) %>%
  pivot_longer(cols = starts_with("y"),
               names_to = "parameter",
               values_to = "y") %>%
  mutate(
    parameter = case_when(parameter == "y1" ~ "(1)",
                           parameter == "y2" ~ "(2)",
                           parameter == "y3" ~ "(10)")
  )

# dist plot
ggplot(rv_student, aes(x, y, color = parameter)) +
  geom_line(size = 2) +
  labs(color = "X ~ t", y = "Density")

rv_student %>%
  group_by(parameter) %>%
  mutate(
    cum_y = cumsum(y)/sum(y)
  ) %>%
  ungroup() %>%
  ggplot(aes(x, cum_y, color = parameter)) +
  geom_line(size = 2) +
  labs(color = "X ~ t", y = "y")

```

### Probability density function

$$f(x) = \frac{\Gamma(\frac{n+1}{2})}{\sqrt{n\pi} \cdot \Gamma(\frac{n}{2})} \left(1 + \frac{x^2}{n}\right)^{-\frac{n+1}{2}},$$

with  $\Gamma(x)$  denoting the gamma function.

### Cumulative distribution function

$$F(x) = I\left(\frac{x + \sqrt{x^2 + n}}{2\sqrt{x^2 + n}}, \frac{n}{2}, \frac{n}{2}\right),$$

where  $I(z, a, b)$  denotes the regularized incomplete beta function:

$$I(z, a, b) = \frac{1}{B(a, b)} \cdot \int_0^z t^{a-1} (1-t)^{b-1} dt.$$

*B.1. Selected continuous distributions of random variables*

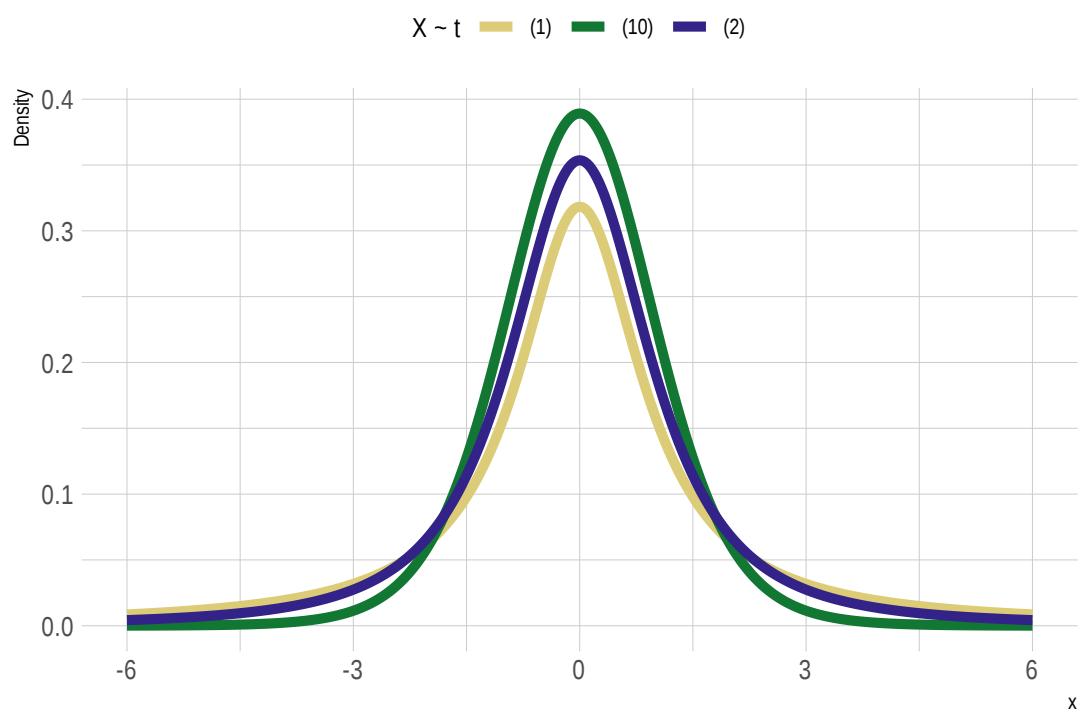


Figure B.8.: Examples of probability density function of t distributions.

*B. Common probability distributions*

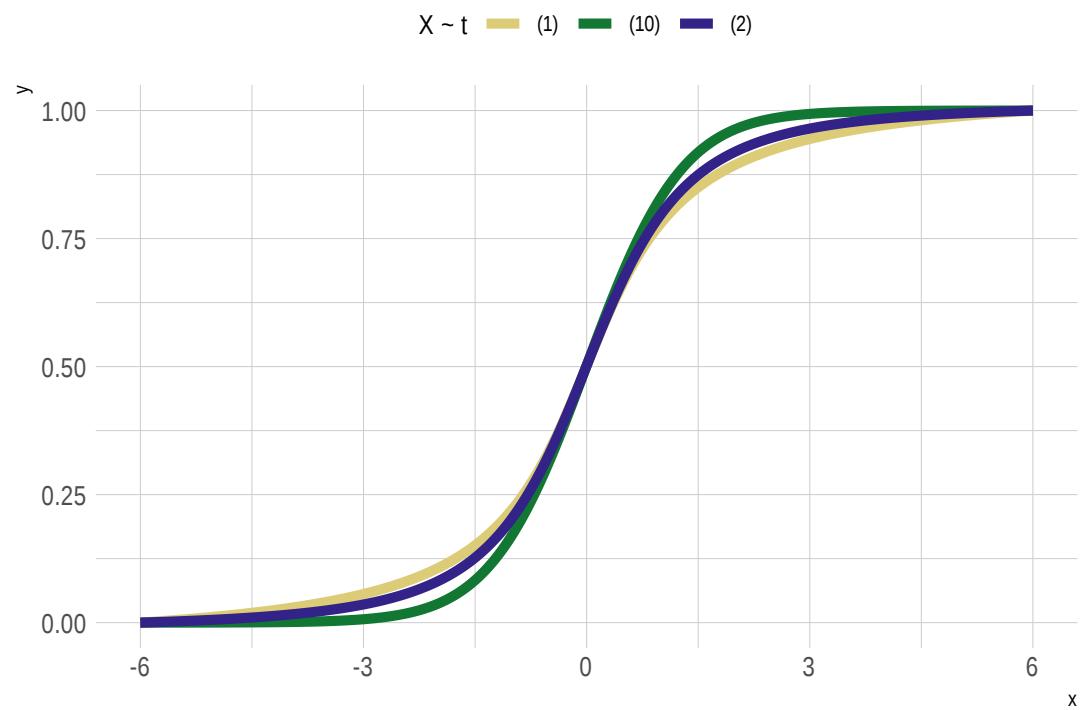


Figure B.9.: Examples of the cumulative distribution function of t distributions corresponding to the previous probability density functions.

**Expected value**  $E(X) = 0$

**Variance**  $Var(X) = \frac{n}{n-2}$  (for  $n \geq 30$ )

#### B.1.4.1. Hands On

```
knitr::include_app("https://istats.shinyapps.io/tdist/", height = "800px")
```

App taken from <http://www.artofstat.com/webapps.html> (Klingenberg, n.d.)

#### B.1.5. Beta distribution

The beta distribution creates a continuous distribution of numbers between 0 and 1, therefore this distribution is useful if the uncertain quantity is bounded by 0 and 1 (or 100%), is continuous, and has a single mode. In Bayesian Data Analysis the beta distribution has a special standing as prior distribution for a bernoulli or binomial (see discrete distributions) likelihood. The reason for this is that a combination of a beta prior and a bernoulli (or binomial) likelihood results in a posterior distribution with the same form as the beta distribution. Such priors are referred to as *conjugate priors*.

A beta distribution has two parameters  $a$  and  $b$ :

$$X \sim Beta(a, b).$$

The two parameters can be interpreted as the number of observations made, such that:  $n = a + b$ . If  $a$  and  $b$  get bigger, the beta distribution gets narrower. If only  $a$  gets bigger the distribution moves rightward and if only  $b$  gets bigger the distribution moves leftward. Thus, the parameters define the shape of the distribution, therefore they are also called *shape parameters*. A Beta(1,1) is equivalent to a uniform distribution. Fig.~B.10 shows the probability density function of four beta distributed random variables with different parameter values. Fig.~B.11 shows the corresponding cumulative functions.

```
rv_beta <- tibble(
  x = seq(from = 0, to = 1, by = .01),
  y1 = dbeta(x, shape1 = 1, shape2 = 1),
  y2 = dbeta(x, shape1 = 4, shape2 = 4),
  y3 = dbeta(x, shape1 = 4, shape2 = 2),
  y4 = dbeta(x, shape1 = 2, shape2 = 4)
) %>%
  pivot_longer(cols = starts_with("y"),
               names_to = "parameter",
               values_to = "y") %>%
  mutate(
```

## B. Common probability distributions

```

parameter = case_when(parameter == "y1" ~ "(1,1)",
                      parameter == "y2" ~ "(4,4)",
                      parameter == "y3" ~ "(4,2)",
                      parameter == "y4" ~ "(2,4)")
)

# dist plot
ggplot(rv_beta, aes(x, y, color = parameter)) +
  geom_line(size = 2) +
  labs(color = "X ~ Beta", y = "Density")

```

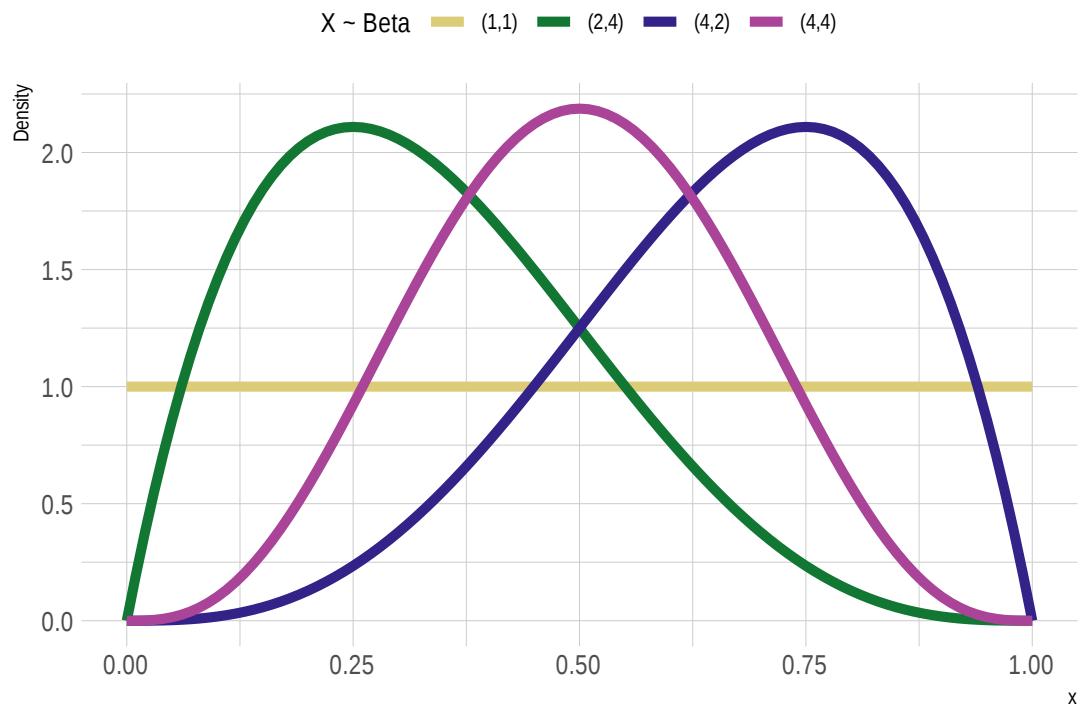


Figure B.10.: Examples of probability density function of beta distributions.

```

rv_beta %>%
  group_by(parameter) %>%
  mutate(
    cum_y = cumsum(y)/sum(y)
  ) %>%
  ungroup() %>%
  ggplot(aes(x, cum_y, color = parameter)) +

```

### B.1. Selected continuous distributions of random variables

```
geom_line(size = 2) +
labs(color = "X ~ Beta", y = "y")
```

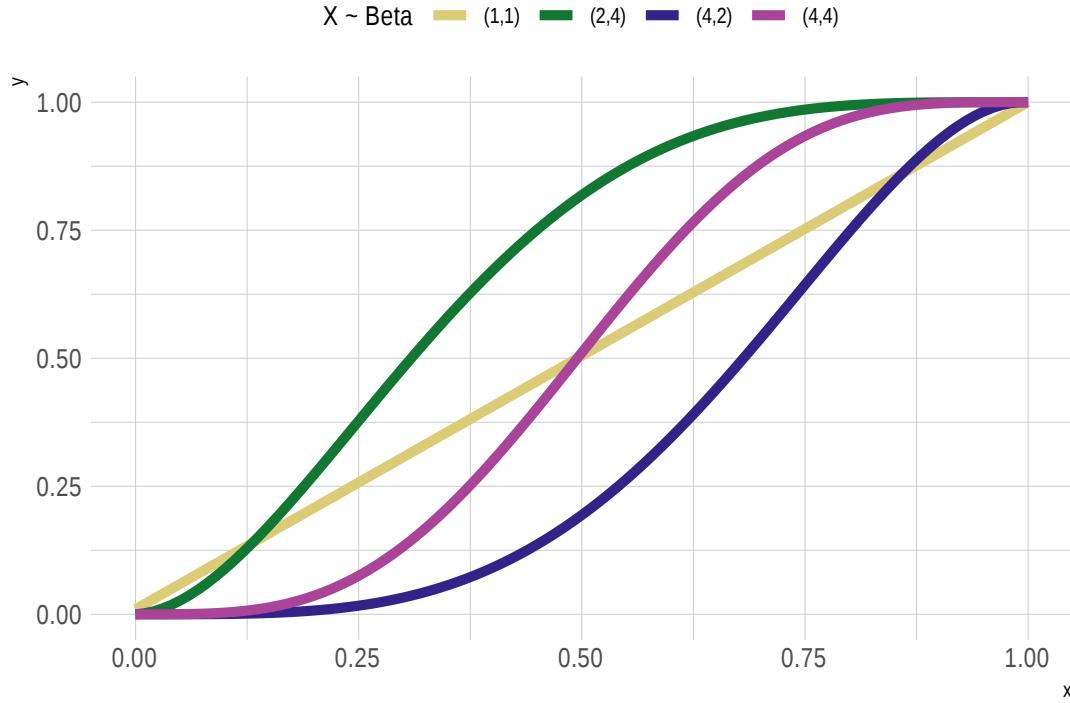


Figure B.11.: Examples of the cumulative distribution function of beta distributions corresponding to the previous probability density functions.

#### Probability density function

$$f(x) = \frac{\theta^{(a-1)}(1-\theta)^{(b-1)}}{B(a,b)},$$

where  $B(a, b)$  is the beta function:

$$B(a, b) = \int_0^1 \theta^{(a-1)}(1-\theta)^{(b-1)} d\theta.$$

#### Cumulative distribution function

$$F(x) = \frac{B(x; a, b)}{B(a, b)},$$

## B. Common probability distributions

where  $B(x; a, b)$  is the *incomplete beta function*:

$$B(x; a, b) = \int_0^x t^{(a-1)}(1-t)^{(b-1)} dt,$$

and  $B(a, b)$  the (complete) *beta function*

$$B(a, b) = \int_0^1 \theta^{(a-1)}(1-\theta)^{(b-1)} d\theta.$$

**Expected value** Mean:  $E(X) = \frac{a}{a+b}$  Mode:  $\omega = \frac{(a-1)}{a+b-2}$

**Variance** Variance:  $Var(X) = \frac{ab}{(a+b)^2(a+b+1)}$  Concentration:  $\kappa = a + b$  (related to variance such that, the bigger  $a$  and  $b$  are, the narrower the distribution)

**Reparameterization of the beta distribution** Sometimes it is helpful (and more intuitive) to write the beta distribution in terms of its mode  $\omega$  and concentration  $\kappa$  instead of  $a$  and  $b$ :

$$Beta(a, b) = Beta(\omega(\kappa - 2) + 1, (1 - \omega)(\kappa - 2) + 1), \text{ for } \kappa > 2.$$

### B.1.6. Uniform distribution

The (continuous) uniform distribution takes values within a specified range  $a$  and  $b$  that have constant probability. Sometimes the distribution is also called rectangular distribution, due to its shape of a rectangle. The uniform distribution is in particular common for random number generation. In Bayesian Data Analysis it is often used as prior distribution to express *ignorance*. This can be thought in the following way: When different events are possible but no (reliable) information exists about their probability of occurrence, the most conservative (and also intuitive) choice would be to assign probability such that all events are equally likely to occur. The uniform distribution model this intuition, it generates a completely random number in some interval  $[a, b]$ .

The distribution is specified by two parameters: the end points  $a$  (minimum) and  $b$  (maximum):

$$X \sim Uni(a, b).$$

When  $a = 0$  and  $b = 1$  the distribution is referred to as *standard* uniform distribution. Fig.~B.12 shows the probability density function of two uniform distributed random variables with different parameter values. Fig.~B.13 shows the corresponding cumulative functions.

### B.1. Selected continuous distributions of random variables

```

rv_unif <- tibble(
  x = seq(from = -.1, to = 4.2, by = .01),
  y1 = dunif(x),
  y2 = dunif(x, min = 2, max = 4)
) %>%
  pivot_longer(cols = starts_with("y"),
               names_to = "parameter",
               values_to = "y") %>%
  mutate(
    parameter = ifelse(parameter == "y1",
                        "(0,1)", "(2,4)")
  )

# dist plot
ggplot(rv_unif, aes(x, y, color = parameter)) +
  geom_line(size = 2) +
  labs(color = "X ~ Uniform", y = "Density")

```

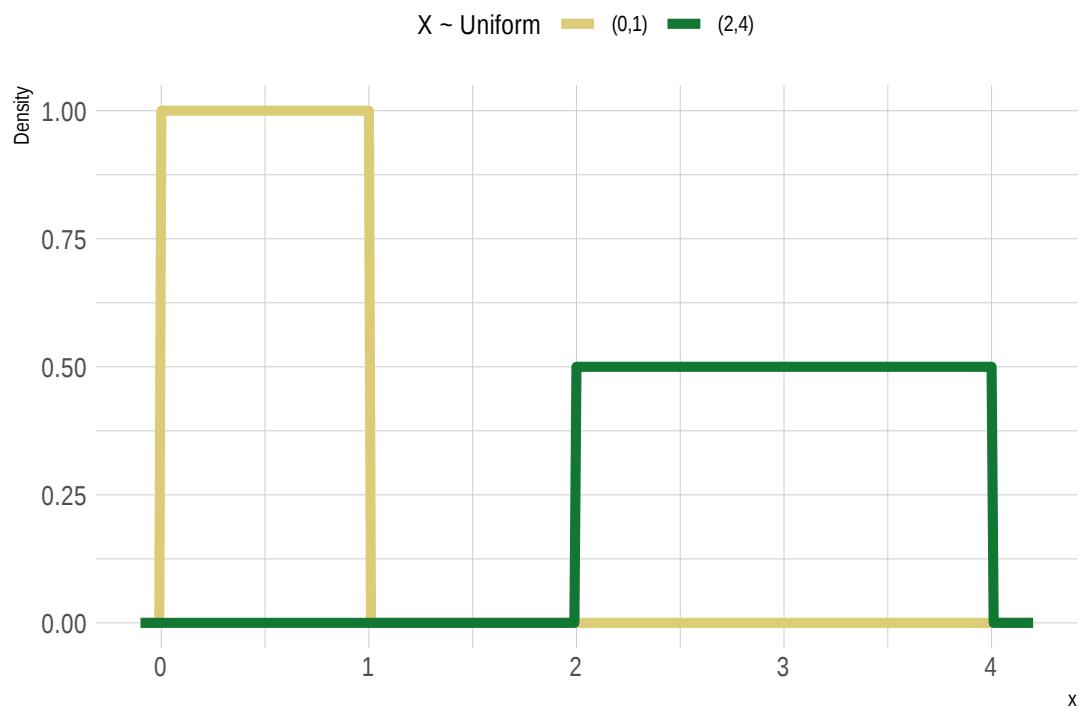


Figure B.12.: Examples of probability density function of uniform distributions.

## B. Common probability distributions

```
rv_unif %>%
  group_by(parameter) %>%
  mutate(
    cum_y = cumsum(y)/sum(y)
  ) %>%
  ungroup() %>%
  ggplot(aes(x, cum_y, color = parameter)) +
  geom_line(size = 2) +
  labs(color = "X ~ Uniform", y = "y")
```

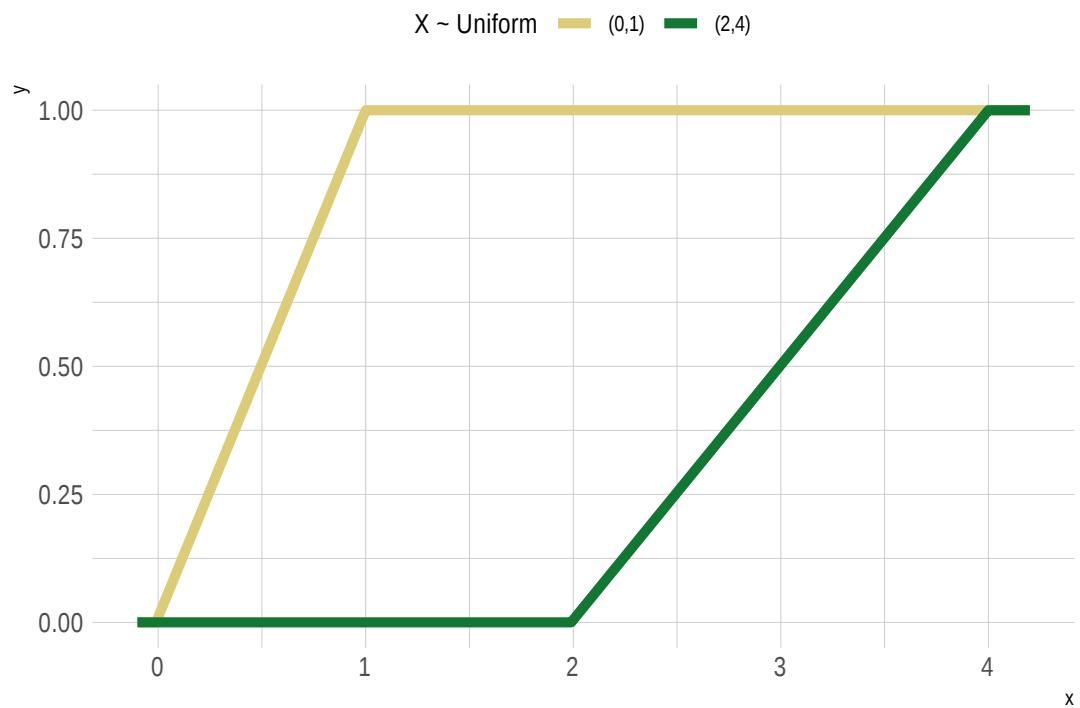


Figure B.13.: Examples of the cumulative distribution function of uniform distributions corresponding to the previous probability density functions.

### Probability density function

$$f(x) = \begin{cases} \frac{1}{b-a} & \text{for } x \in [a, b], \\ 0 & \text{otherwise.} \end{cases}$$

### Cumulative distribution function

$$F(x) = \begin{cases} 0 & \text{for } x < a, \\ \frac{x-a}{b-a} & \text{for } a \leq x < b, \\ 1 & \text{for } x \geq b. \end{cases}$$

**Expected value**  $E(X) = \frac{a+b}{2}$

**Variance**  $Var(X) = \frac{(b-a)^2}{12}$

## B.2. Selected discrete distributions of random variables

### B.2.1. Binomial distribution

The binomial distribution is a useful model for binary decisions where the outcome is a choice between two alternatives (e.g. Yes/No, Left/Right, Present/Absent, Head/Tail, ...). The two outcomes are coded as 0 (failure) and 1 (success). Consequently, let the probability of occurrence of the outcome “success” be  $p$ , then the probability of occurrence of “failure” is  $1-p$ . Consider a coin-flip experiment, with the outcomes “head” and “tail”. If we flip a coin repeatedly, e.g. 30 times, the successive trials are independent of each other and the probability  $p$  is constant, then the resulting binomial distribution is a discrete random variable with outcomes  $\{0, 1, 2, \dots, 30\}$ .

The binomial distribution has two parameters “size” and “prob”, often denoted as  $n$  and  $p$ , respectively. The “size” refers to the number of trials and “prob” to the probability of success:

$$X \sim Binomial(n, p).$$

Fig.~B.14 shows the probability mass function of three binomial distributed random variables with different parameter values. As stated above,  $p$  refers to the probability of success. The higher this probability the more often we will observe the outcome coded with “1”. Therefore the distribution tends toward the right side and vice-versa. The distribution gets more symmetrical if the parameter  $p$  approximates 0.5. Fig.~B.15 shows the corresponding cumulative functions.

```
# how many trials
trials = 30

rv_binom <- tibble(
  x = seq(0, trials),
  y1 = dbinom(x, size = trials, p = 0.2),
  y2 = dbinom(x, size = trials, p = 0.5),
  y3 = dbinom(x, size = trials, p = 0.8)
) %>%
  pivot_longer(cols = starts_with("y"),
```

## B. Common probability distributions

```

            names_to  = "parameter",
            values_to = "y") %>%
mutate(
  parameter = case_when(parameter == "y1" ~ "(n,0.2)",
                         parameter == "y2" ~ "(n,0.5)",
                         parameter == "y3" ~ "(n,0.8)")
)

# dist plot
ggplot(rv_binom, aes(x, y, fill = parameter)) +
  geom_col(position = "identity", alpha = 0.8) +
  labs(fill = "X ~ Binomial", y = "Probability")

```

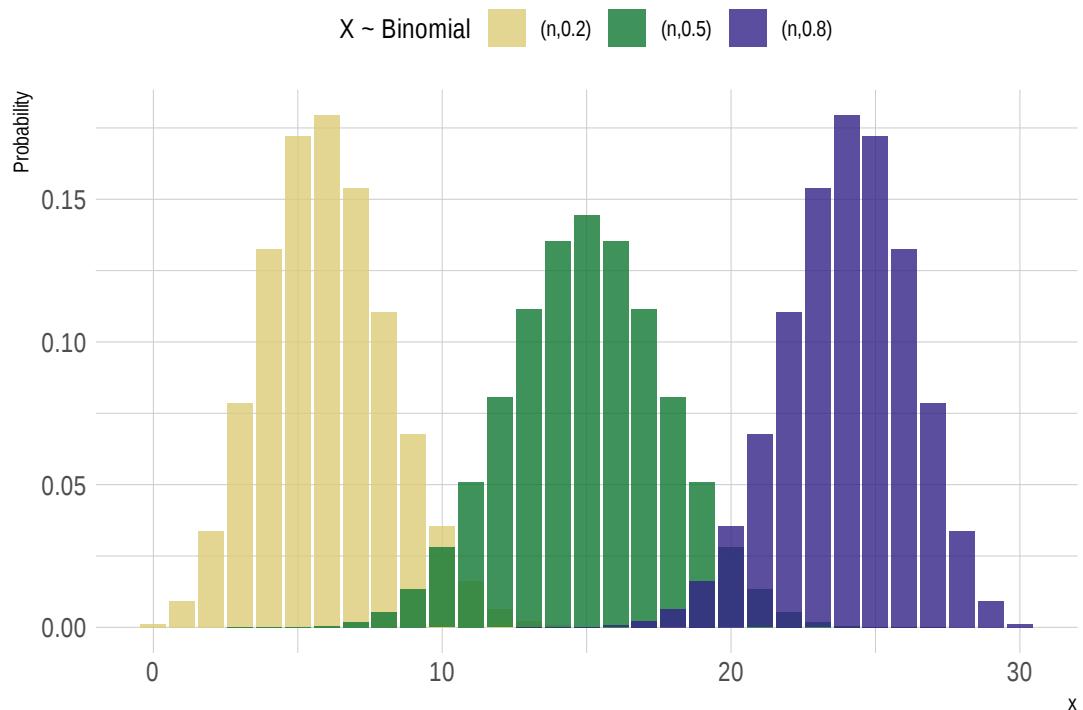


Figure B.14.: Examples of probability mass function of Binomial distributions.

```

rv_binom %>%
  group_by(parameter) %>%
  mutate(
    cum_y = cumsum(y)/sum(y)
  ) %>%

```

```
ungroup() %>%
ggplot(aes(x, cum_y, color = parameter)) +
  geom_step(size = 2) +
  labs(color = "X ~ Binomial", y = "y")
```

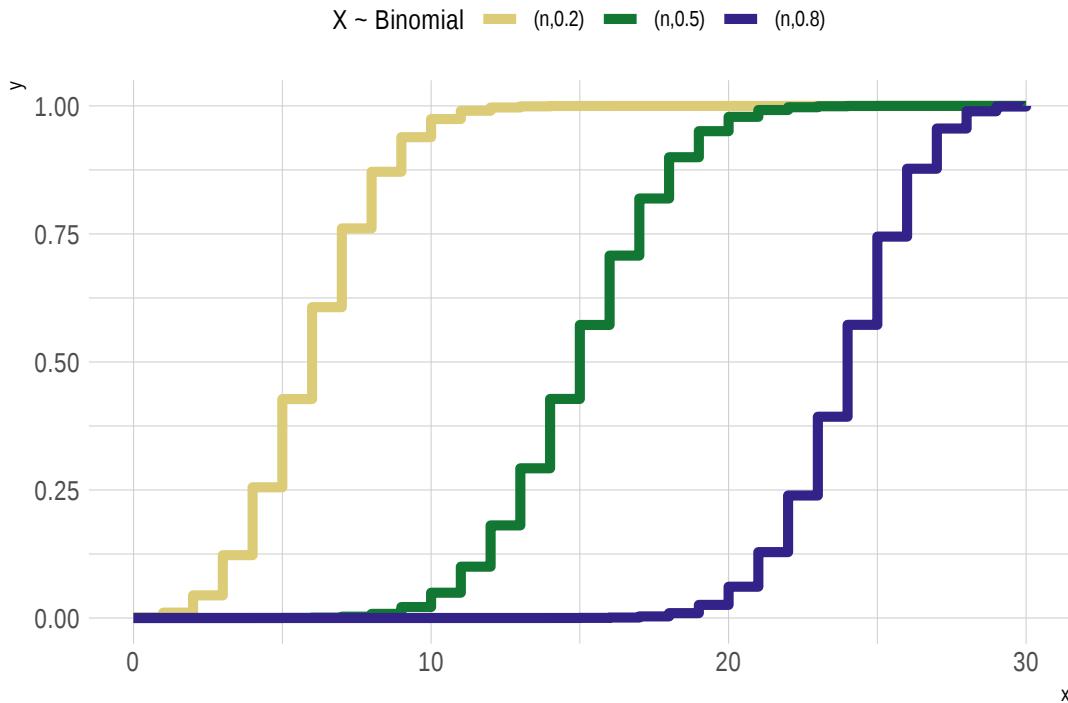


Figure B.15.: Examples of the cumulative distribution function of Binomial distributions corresponding to the previous probability mass functions.

### Probability mass function

$$f(x) = \binom{n}{x} p^x (1-p)^{n-x},$$

where  $\binom{n}{x}$  is the binomial coefficient.

### Cumulative function

$$F(x) = \sum_{k=0}^x \binom{n}{k} p^k (1-p)^{n-k}$$

**Expected value**  $E(X) = n \cdot p$

**Variance**  $Var(X) = n \cdot p \cdot (1-p)$

## B. Common probability distributions

### B.2.1.1. Hands On

```
knitr::include_app("https://istats.shinyapps.io/BinomialDist/", height = "800px")
```

App taken from <http://www.artofstat.com/webapps.html> (Klingenberg, n.d.)

### B.2.2. Bernoulli distribution

The Bernoulli distribution is a special case of the binomial distribution with *size* = 1, therefore the outcome of a bernoulli random variable is either 0 or 1. Apart from that the same information holds as for the binomial distribution. As the “size” parameter is now negligible, the bernoulli distribution has only one parameter, the probability of success  $p$ :

$$X \sim \text{Bern}(p).$$

Fig.B.16 shows the probability mass function of three bernoulli distributed random variables with different parameters. Fig.B.17 shows the corresponding cumulative distributions.

```
rv_bern <- tibble(
  x = seq(from = 0, to = 1),
  y1 = dbern(x, prob = 0.2),
  y2 = dbern(x, prob = 0.5),
  y3 = dbern(x, prob = 0.8)
) %>%
  pivot_longer(cols = starts_with("y"),
               names_to = "parameter",
               values_to = "y") %>%
  mutate(
    parameter = case_when(parameter == "y1" ~ "(0.2)",
                           parameter == "y2" ~ "(0.5)",
                           parameter == "y3" ~ "(0.8)")
  )

# dist plot
ggplot(rv_bern, aes(x, y, fill = parameter)) +
  geom_col(position = "dodge", color = "white") +
  labs(fill = "X ~ Bernoulli", y = "Probability") +
  scale_x_continuous(breaks = c(0.0, 1.0), labels = c("0", "1"), limits = c(-0.5, 1.5))
```

*B.2. Selected discrete distributions of random variables*

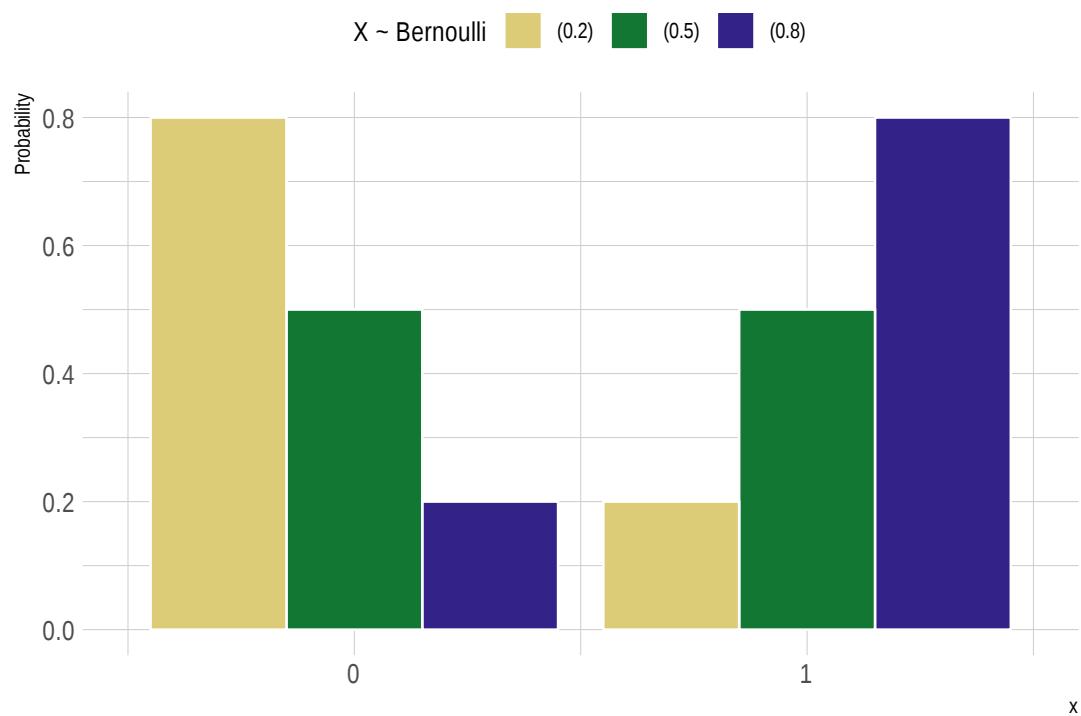


Figure B.16.: Examples of probability mass function of Bernoulli distributions.

## B. Common probability distributions

```

rv_bern %>%
  group_by(parameter) %>%
  mutate(
    cum_y = cumsum(y)/sum(y),
    cum_y2 = cumsum(y)/sum(y)
  ) %>%
  add_column(
    x2 = c(1,1,1,1.5,1.5,1.5)
  ) %>%
  ungroup() %>%
  
```

ggplot(aes(x, cum\_y, color = parameter)) +
 geom\_segment(aes(xend = x2, yend = cum\_y2), size = 1.5, linetype = "dashed") +
 geom\_segment(aes(x = -0.5, y = 0, xend = 0.0, yend = 0), size = 1.5, linetype = "dashed") +
 geom\_point(aes(x, cum\_y), size = 4) +
 labs(color = "X ~ Bernoulli", y = "y") +
 scale\_x\_continuous(breaks = c(0.0,1.0), labels = c("0","1"), limits = c(-0.5,1.5))

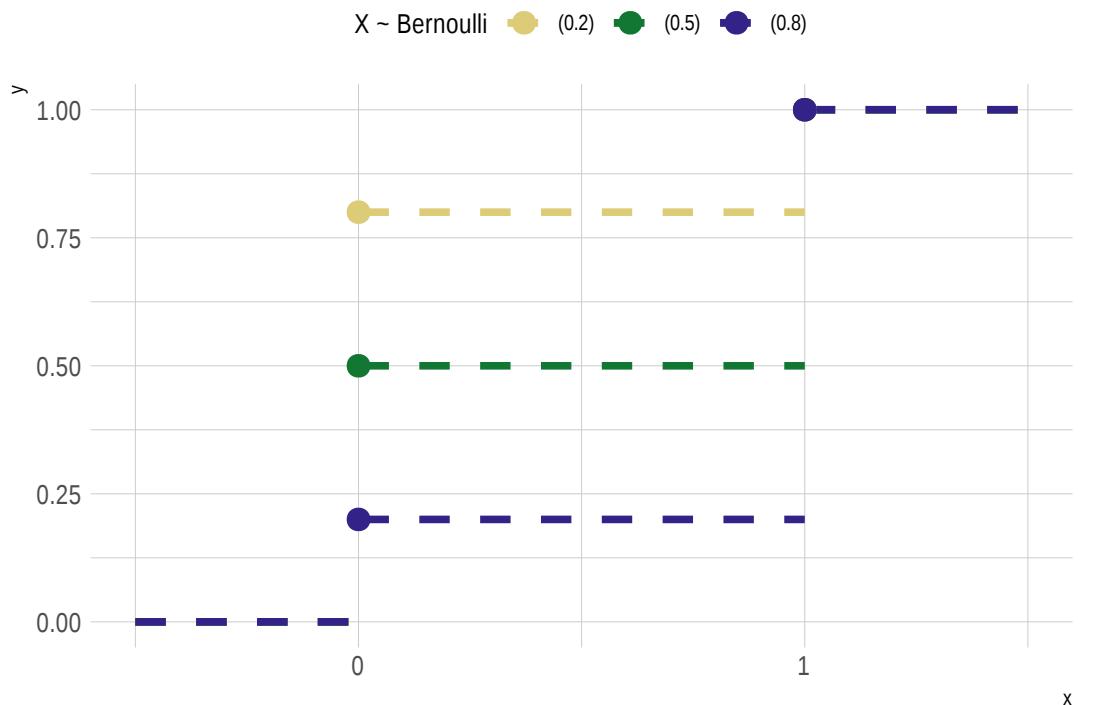


Figure B.17.: Examples of the cumulative distribution function of Bernoulli distributions corresponding to the previous probability mass functions.

### Probability mass function

$$f(x) = \begin{cases} p & \text{if } x = 1, \\ 1 - p & \text{if } x = 0. \end{cases}$$

### Cumulative function

$$F(x) = \begin{cases} 0 & \text{if } x < 0, \\ 1 - p & \text{if } 0 \leq x < 1, \\ 1 & \text{if } x \geq 1. \end{cases}$$

**Expected value**  $E(X) = p$

**Variance**  $Var(X) = p \cdot (1 - p)$

### B.2.3. Beta-Binomial distribution

The beta-binomial distribution, as the name already indicates, is a mixture of a binomial and beta distribution. Remember, a binomial distribution is useful to model a binary choice with outcomes “0” and “1”. The binomial distribution has two parameters  $p$ , the probability of success (“1”), and  $n$ , the number of trials. Furthermore we assume that the successive trials are independent and  $p$  is constant. In a beta-binomial distribution  $p$  is not anymore assumed to be constant (or fixed) but changes from trial to trial. Thus, a further assumption about the distribution of  $p$  is made and here the beta distribution comes into play: the probability  $p$  is assumed to be randomly drawn from a beta distribution with parameters  $a$  and  $b$ . Therefore, the beta-binomial distribution has three parameters  $n$ ,  $a$  and  $b$ :

$$X \sim BetaBinom(n, a, b).$$

For large values of  $a$  and  $b$  the distribution approaches a binomial distribution. When  $a = 1$  and  $b = 1$  the distribution equals a discrete uniform distribution from 0 to  $n$ . When  $n = 1$ , the distribution equals a bernoulli distribution.

Fig.~B.18 shows the probability mass function of three beta-binomial distributed random variables with different parameter values. Fig.~B.19 shows the corresponding cumulative distributions.

```
# how many trials
trials = 30

rv_betabinom <- tibble(
  x = seq(from = 0, to = trials),
  y1 = dbbinom(x, size = trials, alpha = 4, beta = 4),
  y2 = dbbinom(x, size = trials, alpha = 2, beta = 4),
```

## B. Common probability distributions

```

y3 = dbbinom(x, size = trials, alpha = 1, beta = 1)
) %>%
pivot_longer(cols = starts_with("y"),
             names_to = "parameter",
             values_to = "y") %>%
mutate(
  parameter = case_when(parameter == "y1" ~ "(n,4,4)",
                         parameter == "y2" ~ "(n,2,4)",
                         parameter == "y3" ~ "(n,1,1)")
)
# dist plot
ggplot(rv_betaBinom, aes(x, y, fill = parameter)) +
  geom_col(position = "identity", alpha = 0.7) +
  labs(fill = "X ~ Beta-Binomial", y = "Probability")

```

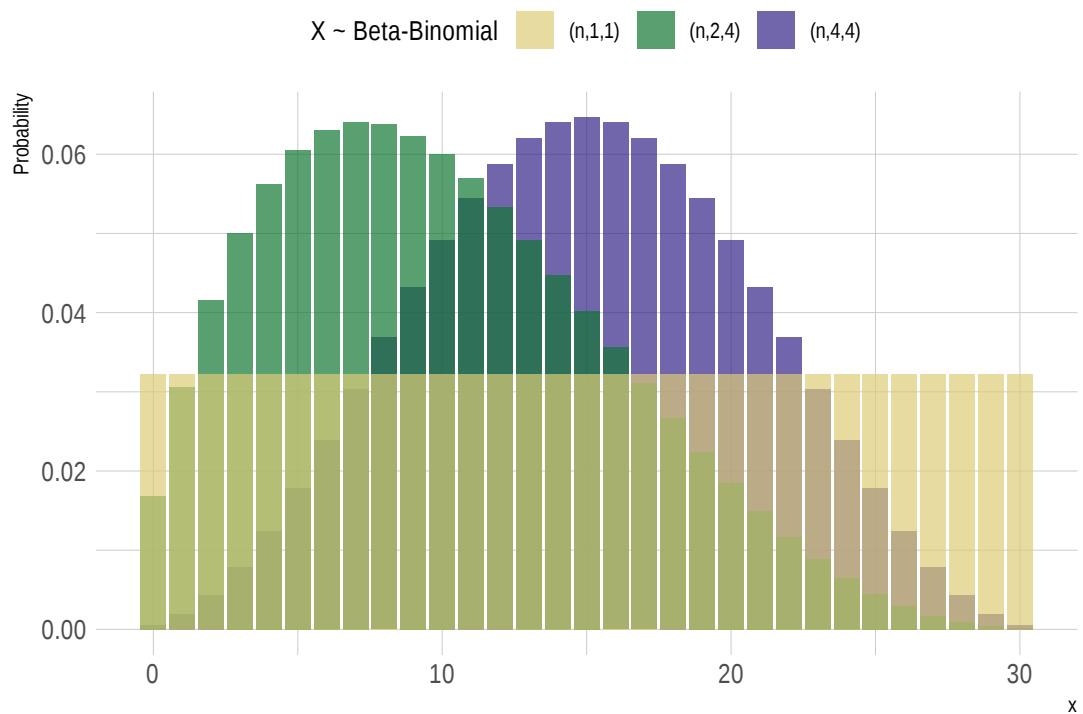


Figure B.18.: Examples of probability mass function of Beta-Binomial distributions.

## B.2. Selected discrete distributions of random variables

```
rv_betabinom %>%
  group_by(parameter) %>%
  mutate(
    cum_y = cumsum(y)/sum(y)
  ) %>%
  ungroup() %>%
  ggplot(aes(x, cum_y, color = parameter)) +
  geom_step(size = 2) +
  labs(color = "X ~ Beta-Binomial", y = "y")
```

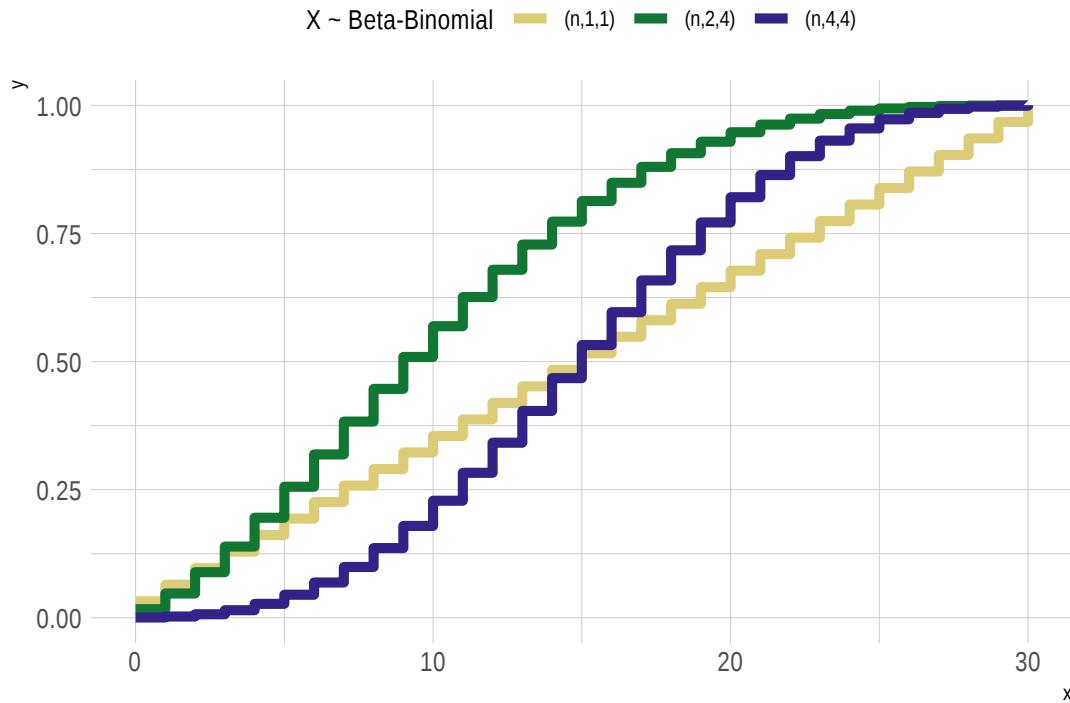


Figure B.19.: Examples of the cumulative distribution function of Beta-Binomial distributions corresponding to the previous probability mass functions.

### Probability mass function

$$f(x) = \binom{n}{x} \frac{B(a+x, b+n-x)}{B(a, b)},$$

where  $\binom{n}{x}$  is the binomial coefficient and  $B(x)$  the beta *function* (see beta distribution).

### Cumulative function

## B. Common probability distributions

$$F(x) = \begin{cases} 0 & \text{if } x < 0, \\ \binom{n}{x} \frac{B(a+x, b+n-x)}{B(a, b)} {}_3F_2(n, a, b) & \text{if } 0 \leq x < n, \\ 1 & \text{if } x \geq n. \end{cases}$$

Where  ${}_3F_2(n, a, b)$  is the generalized hypergeometric function.

**Expected value**  $E(X) = n \frac{a}{a+b}$

**Variance**  $Var(X) = n \frac{ab}{(a+b)^2} \frac{a+b+n}{a+b+1}$

### B.2.4. Poisson distribution

A poisson distributed random variable represents the number of successes occurring in a given *time interval*. It gives the probability of a given number of events happening in a fixed interval of time. The poisson distribution is a limiting case of the binomial distribution when the number of trials becomes very large and the probability of success is small. For example the number of car accidents in Osnabrueck in the next month, the number of typing errors on a page, the number of interruptions generated by a CPU during T seconds, etc. Events described by a poisson distribution must fullfill the following conditions: they occur in non-overlapping intervals, they can not occur simultaneously and each event occurs at a constant rate.

The poisson distribution has one parameter, the rate  $\lambda$ , sometimes also referred to as *intensity*:

$$X \sim Po(\lambda).$$

The parameter  $\lambda$  can be thought of as the expected number of events in the time interval. Consequently, changing the rate parameter changes the probability of seeing different numbers of events in one interval. See Fig.~B.20 for the probability mass function of three poisson distributed random variables with different parameter values. Notice, that the higher  $\lambda$  the more symmetrical gets the distribution. In fact, the poisson distribution can be approximated by a normal distribution for a rate paramter  $\geq 10$ . Fig.~B.21 shows the corresponding cumulative distributions.

```
rv_pois <- tibble(
  x = seq(from = 0, to = 30, by = 1),
  y1 = dpois(x, lambda = 2),
  y2 = dpois(x, lambda = 8),
  y3 = dpois(x, lambda = 15)
) %>%
  pivot_longer(cols = starts_with("y"),
               names_to = "parameter",
               values_to = "y") %>%
  mutate(
```

## B.2. Selected discrete distributions of random variables

```

parameter = case_when(parameter == "y1" ~ "(2)",
                      parameter == "y2" ~ "(8)",
                      parameter == "y3" ~ "(15)")
)

# dist plot
ggplot(rv_pois, aes(x, y, fill = parameter)) +
  geom_col(alpha = 0.7, position = "identity") +
  labs(fill = "X ~ Poisson", y = "Density")

```

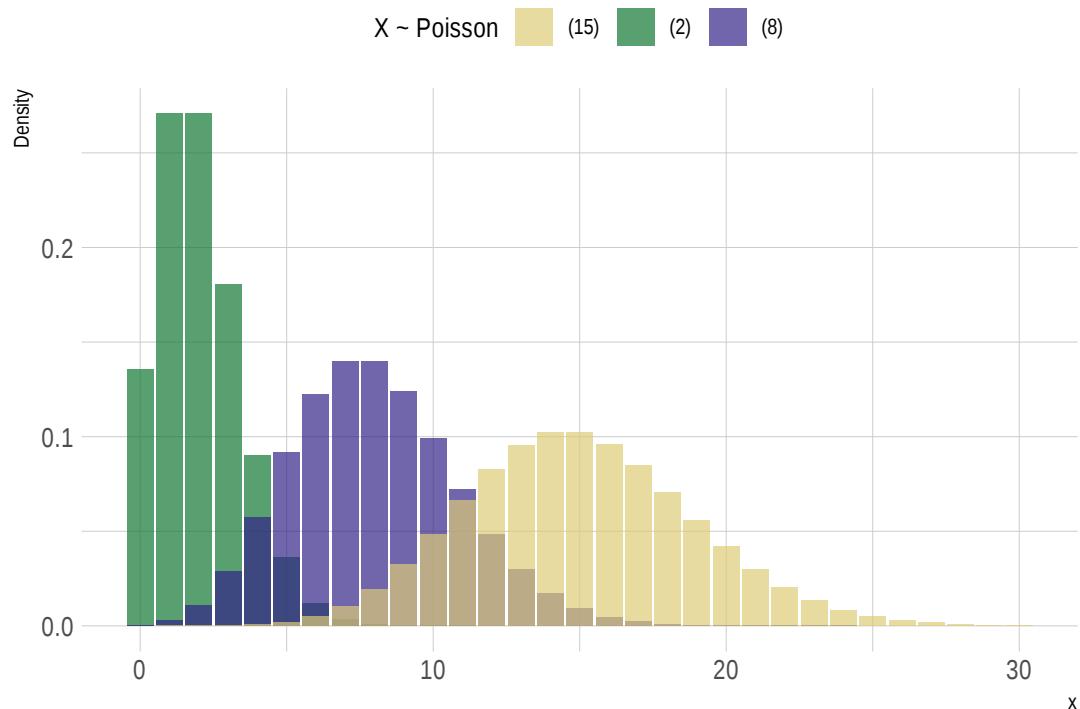


Figure B.20.: Examples of probability mass function of Poisson distributions.

```

# cumdist plot
rv_pois %>%
  group_by(parameter) %>%
  mutate(
    cum_y = cumsum(y)/sum(y)
  ) %>%
  ungroup() %>%
  ggplot(aes(x, cum_y, color = parameter)) +

```

## B. Common probability distributions

```
geom_step(size = 2) +
  labs(color = "X ~ Poisson", y = "y")
```

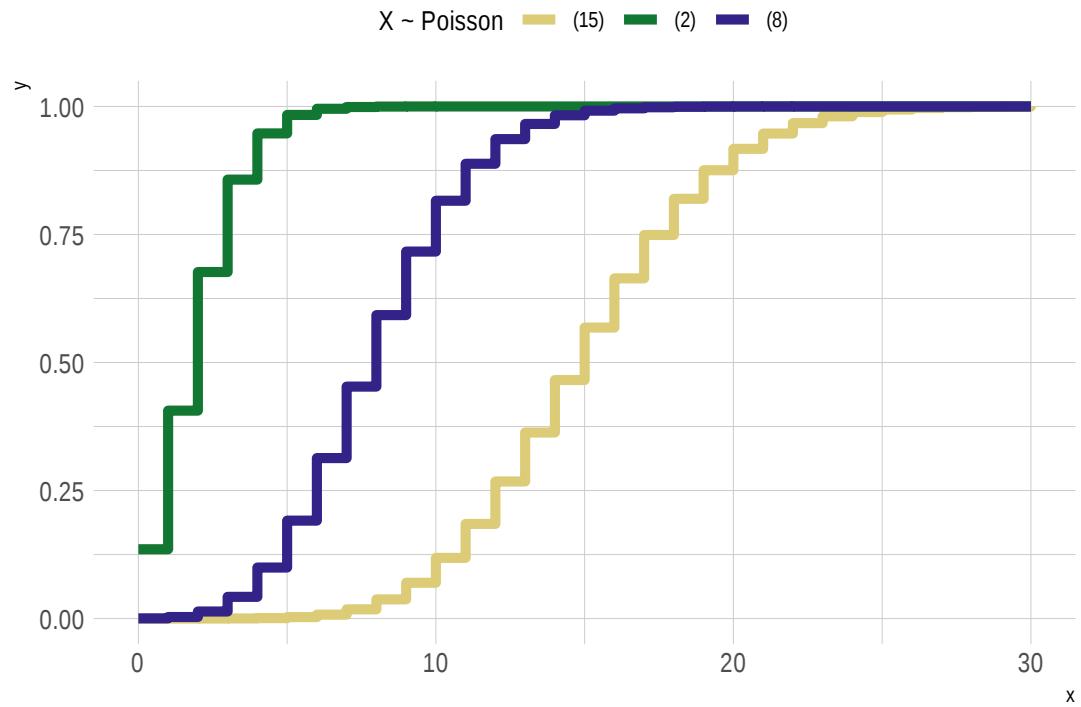


Figure B.21.: Examples of the cumulative distribution function of Poisson distributions corresponding to the previous probability mass functions.

### Probability mass function

$$f(x) = \frac{\lambda^x}{x!} e^{-\lambda}$$

### Cumulative function

$$F(x) = \sum_{k=0}^x \frac{\lambda^k}{k!} e^{-\lambda}$$

**Expected value**  $E(X) = \lambda$

**Variance**  $Var(X) = \lambda$

#### B.2.4.1. Hands On

```
knitr:::include_app("https://istats.shinyapps.io/PoissonDist/", height = "800px")
```

App taken from <http://www.artofstat.com/webapps.html> (Klingenberg, n.d.)

## B.3. Understanding distributions as random variables

```
rv_normal_transform = tibble(
  x_axis = seq(from = -5, to = 5, by = .01),
  rv_norm_x = dnorm(x = x_axis, mean = 1, sd = 2),
  rv_norm_y = dnorm(x = x_axis, mean = 1.75, sd = 2.75),
  rv_linear_transform = 3*rv_norm_x+0.25,
  rv_summ_transform = rv_norm_x + rv_norm_y
) %>%
  pivot_longer(cols = starts_with("rv"),
               names_to = "random_variables",
               values_to = "y") %>%
  mutate(
    random_variables = case_when(random_variables == "rv_norm_x" ~ "X ~ N(1,2)",
                                 random_variables == "rv_norm_y" ~ "Y ~ N(1.75,2.75)",
                                 random_variables == "rv_linear_transform" ~ "3*X+0.25 ~ N(3*1+0.25,3*2.75^2)",
                                 random_variables == "rv_summ_transform" ~ "X+Y ~ N(1+1.75,2^2+2.75^2)"
    )
)

# dist plot
ggplot(rv_normal_transform, aes(x_axis, y, color = random_variables)) +
  geom_line(size = 2) +
  labs(color = "X,Y ~ Normal", y = "Density", x = "x")

rv_norm_transform = tibble(
  rv_norm_X = rnorm(n = 1e6),
  rv_norm_Y = rnorm(n = 1e6),
  rv_norm_Z = rnorm(n = 1e6),
  rv_summ_transform_1 = rv_norm_X^2+rv_norm_Y^2,
  rv_summ_transform_2 = rv_norm_X^2+rv_norm_Y^2+rv_norm_Z^2
) %>%
  pivot_longer(cols = starts_with("rv_norm"),
```

## B. Common probability distributions

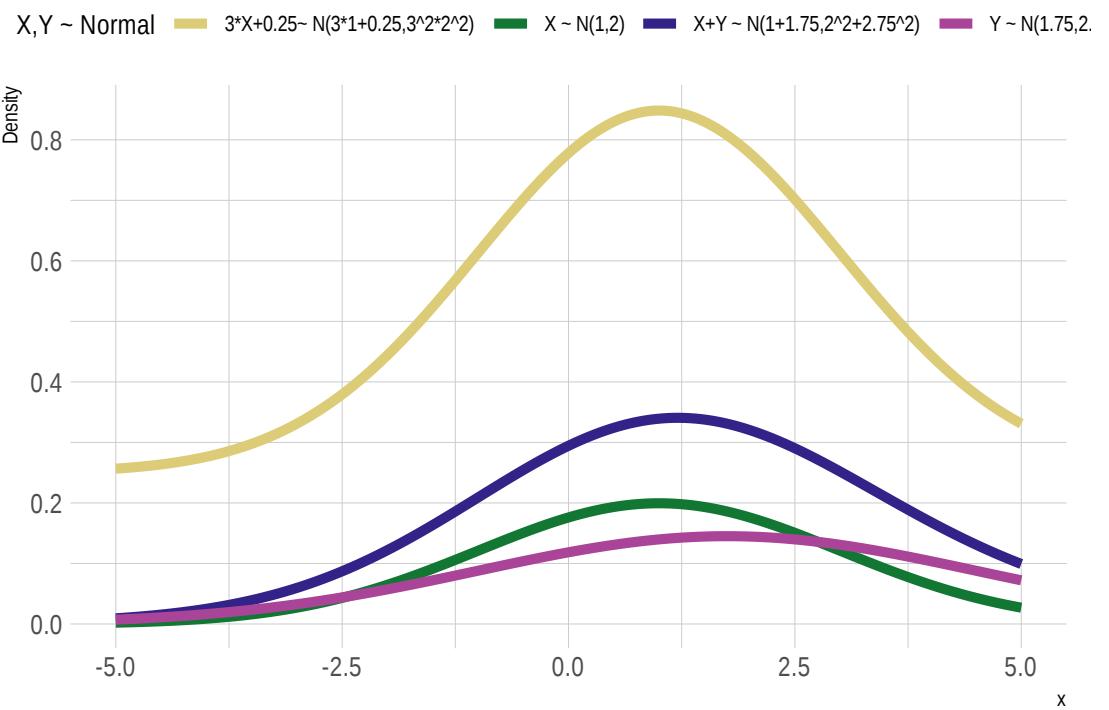


Figure B.22.: Transformation of a normal distributed random variable by addition and linear transformation.

### B.3. Understanding distributions as random variables

```

names_to  = "random_variables",
values_to = "y") %>%
mutate(
RVs = case_when(random_variables == "rv_summ_transform_1" ~ "X^2+Y^2",
                random_variables == "rv_summ_transform_2" ~ "X^2+Y^2+Z^2"
)
)

rv_chi_transform = tibble(
x_axis = seq(from = 0, to = 10, by = .01),
rv_chi_1 = dchisq(x = x_axis, df = 2),
rv_chi_2 = dchisq(x = x_axis, df = 3)
) %>%
pivot_longer(cols = starts_with("rv"),
              names_to  = "random_variables",
              values_to = "y") %>%
mutate(
RVs = case_when(random_variables == "rv_chi_1" ~ "Chi(2)",
                random_variables == "rv_chi_2" ~ "Chi(3)"
)
)

# dist plots
p1 <- ggplot() +
  geom_line(rv_chi_transform, mapping = aes(x_axis, y, color = RVs), size = 2) +
  labs(y = "Density", x = "x")

p2 <- ggplot() +
  geom_line(rv_norm_transform, mapping = aes(y, color = RVs), size = 2, stat = "density") +
  xlim(0,10) +
  ylim(0,0.5) +
  labs(y = "Density", x = "x")

grid.arrange(p1,p2, ncol = 2)

rv_norm_chi_transform = tibble(
  rv_norm_X = rnorm(n = 1e6),
  rv_norm_Y = rnorm(n = 1e6),
  rv_norm_Z = rnorm(n = 1e6),
  rv_chi = rv_norm_Y^2+rv_norm_Z^2,
  rv_transform_t = rv_norm_X/sqrt(rv_chi/2)
) %>%
pivot_longer(cols = starts_with("rv_transform"),

```

*B. Common probability distributions*

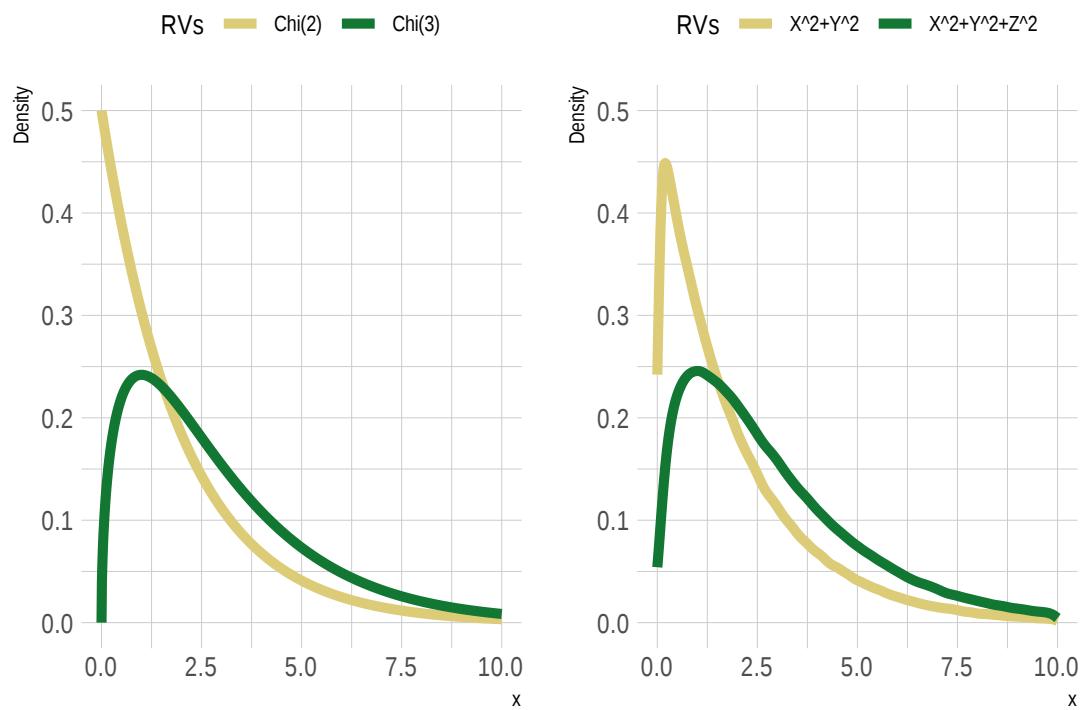


Figure B.23.: Create chi-distributed random variable by summation of standard normal random variables

### B.3. Understanding distributions as random variables

```

names_to  = "random_variables",
values_to = "y") %>%
mutate(
  RVs = case_when(random_variables == "rv_transform_t" ~ "X/sqrt(chi/df)")
)

rv_student = tibble(
  x_axis = seq(from = -5, to = 5, by = .01),
  rv_t_1 = dt(x = x_axis, df = 2)
) %>%
pivot_longer(cols = starts_with("rv"),
              names_to  = "random_variables",
              values_to = "y") %>%
mutate(
  RVs = case_when(random_variables == "rv_t" ~ "t(2)")
)

# dist plots
p1 <- ggplot() +
  geom_line(rv_student, mapping = aes(x_axis, y), size = 2) +
  ylim(0,0.4) +
  labs(y = "Density", x = "x")

p2 <- ggplot() +
  geom_line(rv_norm_chi_transform, mapping = aes(y), size = 2, stat = "density") +
  xlim(-5,5) +
  ylim(0,0.4) +
  labs(y = "Density", x = "x")

grid.arrange(arrangeGrob(p1, top = "RV ~ t(2)",), arrangeGrob(p2, top = "RV ~ std.norm/sqrt(chi/df)"))

rv_norm_chi_transform = tibble(
  rv_norm_V = rnorm(n = 1e6),
  rv_norm_W = rnorm(n = 1e6),
  rv_norm_X = rnorm(n = 1e6),
  rv_norm_Y = rnorm(n = 1e6),
  rv_norm_Z = rnorm(n = 1e6),
  rv_chi_1 = rv_norm_V^2+rv_norm_W^2,
  rv_chi_2 = rv_norm_X^2+rv_norm_Y^2+rv_norm_Z^2,
  rv_transform_F = (rv_chi_1/2)/(rv_chi_2/3)
) %>%
pivot_longer(cols = starts_with("rv_transform"),
              names_to  = "random_variables",
              values_to = "y")

```

*B. Common probability distributions*

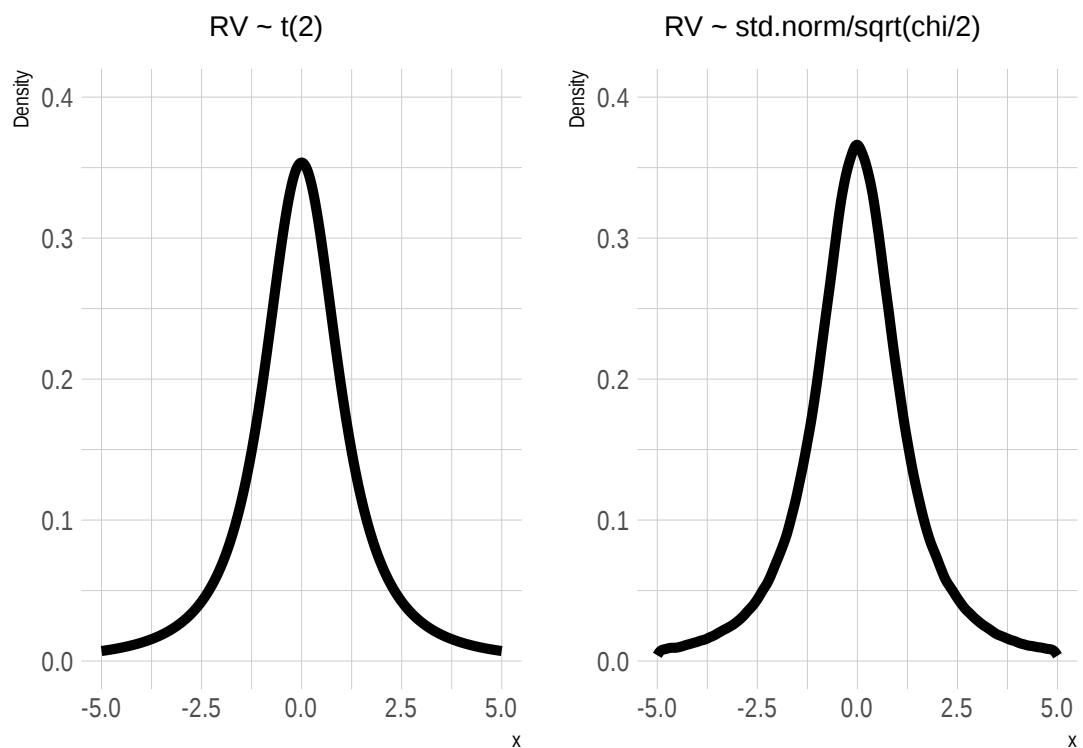


Figure B.24.: Create t-distributed random variable by division of standard normal RV by chi-squared RV

### B.3. Understanding distributions as random variables

```
names_to  = "random_variables",
values_to = "y") %>%
mutate(
RVs = case_when(random_variables == "rv_transform_F" ~ "(rv_chi_1/2)/(rv_chi_2/3)")
)

rv_fisher = tibble(
x_axis = seq(from = 0, to = 20, by = .01),
rv_F = df(x = x_axis, df1 = 2, df2 = 3)
) %>%
pivot_longer(cols = starts_with("rv"),
names_to  = "random_variables",
values_to = "y") %>%
mutate(
RVs = case_when(random_variables == "rv_F" ~ "F(2,3)")
)

# dist plots
p1 <- ggplot() +
geom_line(rv_fisher, mapping = aes(x_axis, y), size = 2) +
ylim(0,1) +
labs(y = "Density", x = "x")

p2 <- ggplot() +
geom_line(rv_norm_chi_transform, mapping = aes(y), size = 2, stat = "density") +
xlim(0,20) +
ylim(0,1) +
labs(y = "Density", x = "x")

grid.arrange(arrangeGrob(p1, top = "RV ~ F(2,3)",), arrangeGrob(p2, top = "RV ~ (rv_chi_1/2)/(rv_chi_2/3)"))
```

*B. Common probability distributions*

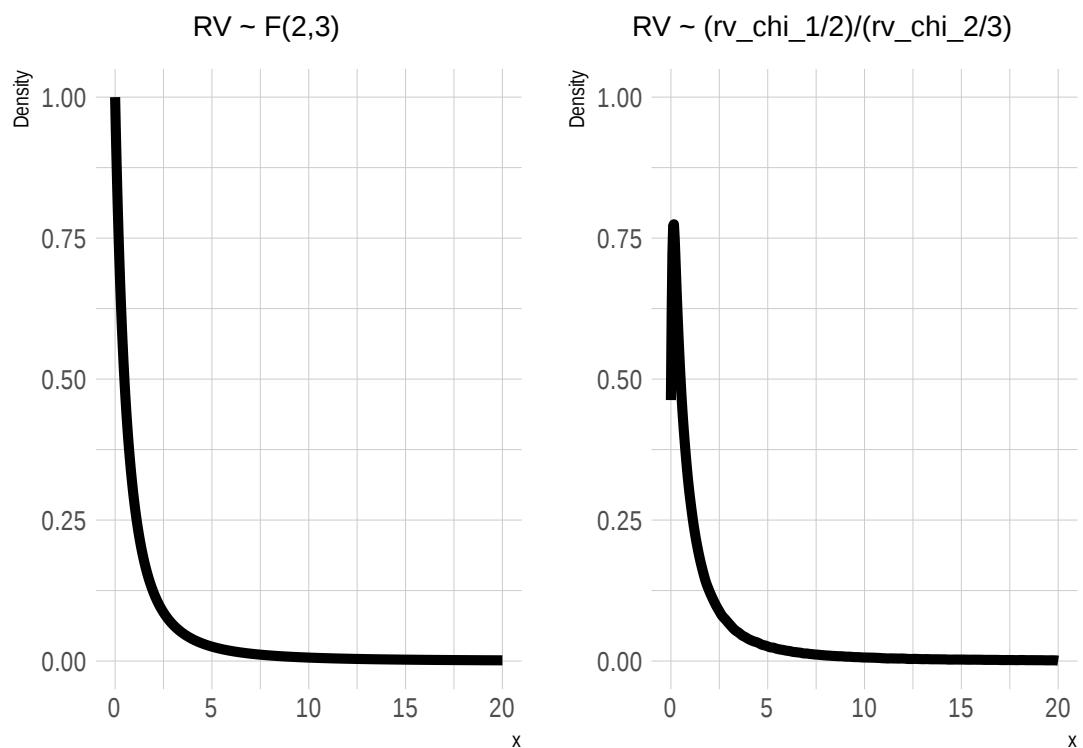


Figure B.25.: Create F-distributed random variable by division of chi-squared RV by (another independent) chi-squared RV

# C. Exponential Family and Maximum Entropy

This chapter deals with the Exponential Family of probability distributions.

## C.1. An important family: The Exponential Family

Most common distributions used in statistical modeling are members of the exponential family. Among others:

- Poisson distribution,
- Bernoulli distribution,
- Normal distribution,
- Chi-Square distribution, and of course the
- Exponential distribution.

In the upcoming section some of these distributions will be described in more detail. But what makes the *exponential family* so *special*? On the one hand, distributions of this family have some convenient mathematical properties which makes them attractive to use in *statistical modeling*. In particular for Bayesian Analysis: For example do all these distributions have a *conjugate prior* and the posterior distribution has a simple form. Furthermore the above example distributions are really just examples. The exponential family encompasses a *wide class of distributions* which makes it possible to model various cases.

On the other hand, the use of distributions from the exponential family is also from a *conceptual perspective* attractive. Consider for example the following situation:

Consider we want to infer a probability distribution subject to certain constraints. For example a coin flip experiment can have only a dichotomous outcome  $\{0,1\}$  and has a constant probability. *Which distribution should be used in order to model this scenario?*

There are several possible distributions that can be used, according to which *criteria* should a distribution be selected? Often one attempts a *conservative choice*, that is to bring as little subjective information into a model as possible. Or in other terms, one goal could be to select the distribution, among all possible distributions, that is *maximal ignorant* and least biased given the constraints.

### C. Exponential Family and Maximum Entropy

Consequently, the question arises how “*ignorance*” can be measured and *distributions compared* according to their “information content”? This will be topic of the upcoming excursions, the key words here are “*entropy*”, which comes from information theory, and “*Maximum Entropy Principal*”.

To briefly anticipate the connection between exponential family and maximum ignorance distributions: The maximum entropy principal starts with constraints that are imposed on a distribution and derives by maximizing entropy a probability density/mass function. Distributions belonging to the exponential family arise as *solutions to the maximum entropy problem* subject to linear constraints.

In the upcoming section selected continuous and discrete distributions will be described in more detail. Followed by a part which motivation is to strengthen the intuition about understanding *distributions as random variables*.

## C.2. Excursions: “Information Entropy” and “Maximum Entropy Principal”

### C.2.1. Information Entropy

*Entropy* is a measure of information content of an outcome of  $X$  such that less probable outcomes convey more information than more probable ones. Thus, entropy can be stated as a *measure of uncertainty*. When the goal is to find a distribution that is as ignorant as possible, then, consequently, entropy should be maximal. Formally, entropy is defined as follows: If  $X$  is a discrete random variable with distribution  $P(X = x_i) = p_i$  then the entropy of  $X$  is

$$H(X) = - \sum_i p_i \log p_i.$$

If  $X$  is a continuous random variable with probability density  $p(x)$  then the differential entropy of  $X$  is

$$H(X) = - \int_{-\infty}^{+\infty} p(x) \log p(x) dx.$$

From which considerations is this *entropy* definition derived? There exist various approaches that finally come to the same answer: the above stated definition of entropy. However, the most cited derivation is Shannon’s theorem. Another and perhaps more intuitive derivation is Wallis derivation. Jaynes (2003) describes both approaches in detail. The following provides a short insight in both derivations and is taken from (Jaynes 2003).

### C.2.1.1. Shannon’s theorem

Shannon’s approach starts by stating conditions that a measure of the *amount of uncertainty*  $H_n$  has to satisfy.

1. It is possible to set up some kind of association between *amount of uncertainty* and real numbers
2.  $H_n$  is a continuous function of  $p_i$ . Otherwise, an arbitrarily small change in the probability distribution would lead to a big change in the amount of uncertainty.
3.  $H_n$  should correspond to common sense in that, when there are many possibilities, we are more uncertain than when there are few. This condition takes the form that in case the  $p_i$  are all equal, the quantity  $h(n)$  is a monotonic increasing function of  $n$ .
4.  $H_n$  is consistent in that, when there is more than one way of working out its value, we must get the same answer for few possible way.

Under these assumptions the resulting unique measure of uncertainty of a probability distribution  $p$  turns out to be just the *average log-probability*:

$$H(p) = - \sum_i p_i \log(p_i).$$

(The interested reader can find a systematic derivation in (Jaynes 2003).) Accepting this interpretation of entropy, it follows that the distribution  $(p_1, \dots, p_n)$  which maximizes the above equation, subject to constraints imposed by the available information, will represent the most *honest* description of what the model *knows* about the propositions  $(A_1, \dots, A_n)$  (Jaynes 2003).

The function  $H$  is called the *entropy*, or the *information entropy* of the distribution  $\{p_i\}$ .

### C.2.1.2. The Wallis derivation

A second and perhaps more intuitive approach of deriving entropy was suggested by G. Wallis. The following description is taken from Jaynes (2003).

We are given information  $I$ , which is to be used in assigning probabilities  $\{p_1, \dots, p_m\}$  to  $m$  different probabilities. We have a total amount of probability

$$\sum_{i=1}^m p_i = 1$$

to allocate among them.

The problem can be stated as follows. Choose some integer  $n \gg m$ , and imagine that we have  $n$  little *quanta* of probabilities, each of magnitude  $\delta = \frac{1}{n}$ , to distribute in an way we see fit.

### C. Exponential Family and Maximum Entropy

Suppose we were to scatter these quanta at random among the  $m$  choices (penny-pitch game into  $m$  equal boxes). If we simply toss these quanta of probability at random, so that each box has an equal probability of getting them, nobody can claim that any box is being unfairly favoured over any other.

If we do this and the first box receives exactly  $n_1$  quanta, the second  $n_2$  quanta etc. we will say the random experiment has generated the probability assignment:

$$p_i = n_i \delta = \frac{n_i}{n}, \text{ with } i = 1, 2, \dots, m.$$

The probability that this will happen is the multinomial distribution:

$$m^{-n} \frac{n!}{n_1! \cdot \dots \cdot n_m!}.$$

Now imagine that we repeatedly scatter the  $n$  quanta at random among the  $m$  boxes. Each time we do this we examine the resulting probability assignment. If it happens to conform to the information  $I$ , we accept it; otherwise we reject it and try again. We continue until some probability assignment  $\{p_1, \dots, p_m\}$  is accepted.

What is the most likely probability distribution to result from this game? It is the one which maximizes

$$W = \frac{n!}{n_1! \cdot \dots \cdot n_m!}$$

subject whatever constraints are imposed by the information  $I$ .

We can refine this procedure by using smaller quanta, i.e. large  $n$ . By using *Sterlings approximation*

$$n! \sim \sqrt{(2\pi n)} \left(\frac{n}{e}\right)^n,$$

and taking the logarithm from it:

$$\log(n!) \sim \sqrt{(2\pi n)} + n \log\left(\frac{n}{e}\right),$$

we have

$$\log(n!) \sim \sqrt{(2\pi n)} + n \log(n) - n.$$

Taking furthermore, also the logarithm from  $W$  and substituting  $\log(n!)$  by Sterlings approximation, finally gives the definition of information entropy, as derived by Shannon's theorem:

## C.2. Excursos: “Information Entropy” and “Maximum Entropy Principal”

$$\frac{1}{n} \log(W) \rightarrow - \sum_{i=1}^m p_i \log(p_i) = H(p_1, \dots, p_m).$$

**To sum it up:** Entropy is a measure of uncertainty. The higher the entropy of a random variable  $X$  the more uncertainty it incorporates. When the goal is to find a maximal ignorance distribution, this goal can be consequently translated into a maximization problem: Find the distribution with maximal entropy subject to existing constraints. This will be topic of the next part of our excursos.

### C.2.2. Deriving Probability Distributions using the Maximum Entropy Principle

The maximum entropy principle is a means of deriving probability distributions given certain constraints and the assumption of maximizing entropy. One technique for solving this maximization problem is the *Lagrange multiplier technique*.

#### C.2.2.1. Lagrangian multiplier technique

Given a multivariable function  $f(x, y, \dots)$  and constraints of the form  $g(x, y, \dots) = c$ , where  $g$  is another multivariable function with the same input space as  $f$  and  $c$  is a constant.

In order to minimize (or maximize) the function  $f$  consider the following steps, assuming  $f$  to be  $f(x)$ :

1. Introduce a new variable  $\lambda$ , called *Lagrange multiplier*, and define a new function  $\mathcal{L}$  with the form:

$$\mathcal{L}(x, \lambda) = f(x) + \lambda(g(x) - c).$$

2. Set the derivative of the function  $\mathcal{L}$  equal to the zero:

$$\mathcal{L}'(x, \lambda) = 0,$$

in order to find the critical points of  $\mathcal{L}$ .

3. Consider each resulting solution within the limits of the made constraints and derive the resulting distribution  $f$ , which gives the minimum (or maximum) one is searching for.

For more details see (Academy 2019)

### C. Exponential Family and Maximum Entropy

#### C.2.2.2. Example 1: Derivation of maximum entropy pdf with no other constraints

For more details see (Finlayson 2017, @keng2017)

Suppose a random variable for which we have absolutely no information on its probability distribution, beside the fact that it should be a pdf and thus, integrate to 1. We ask for the following:

*What type of probability density distribution gives maximum entropy when the random variable is bounded by a finite interval, say  $a \leq X \leq b$ ? (Reza 1994)*

We assume that the maximum ignorance distribution is the one with maximum entropy. It minimizes the prior information in a distribution and is therefore the most conservative choice.

For the continuous case entropy, the measure of uncertainty, is defined as

$$H(x) = - \int_a^b p(x) \log(p(x)) dx,$$

with subject to the mentioned constraint that the sum of all probabilities is one (as it is a pdf):

$$\int_a^b p(x) dx = 1.$$

Rewrite this into the form of *Lagrangian* equation gives

$$\mathcal{L} = - \int_a^b p(x) \log(p(x)) dx + \lambda \left( \int_a^b p(x) dx - 1 \right).$$

The next step is to *minimize* the Lagrangian function. To solve this, we have to use the *calculus of variations*(Keng 2017).

First differentiating  $\mathcal{L}$  with respect to  $p(x)$

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial p(x)} &= 0, \\ -1 - \log(p(x)) + \lambda &= 0, \\ p(x) &= e^{(\lambda-1)}. \end{aligned}$$

Second, the result of  $p(x)$  has to satisfy the stated constraint

$$\int_a^b p(x) dx = 1,$$

## C.2. Excursions: “Information Entropy” and “Maximum Entropy Principal”

$$\int_a^b e^{1-\lambda} dx = 1.$$

Solving this equation with respect to  $\lambda$  gives:

$$\lambda = 1 - \log\left(\frac{1}{b-a}\right).$$

Taking both solutions together we get the following probability density function:

$$p(x) = e^{(1-\lambda)} = e^{(1-(1-\log(\frac{1}{b-a})))},$$

$$p(x) = \frac{1}{b-a}.$$

And this is the *uniform distribution* on the interval  $[a, b]$ . Such that, the answer of the above question is:

*The maximum entropy distribution is associated with a random variable , that is distributed as uniform probability density distribution between a and b.*

This should not be too unexpected. As it is quite intuitive that a uniform distribution is the maximal ignorance distribution (when no other constraints were made). The next example will be more exciting.

### C.2.2.3. Example 2: Derivation of maximum entropy pdf with given mean $\mu$ and variance $\sigma^2$

Suppose a random variable  $X$  with a preassigned standard deviation  $\sigma$  and mean  $\mu$ . Again the question is: *Which function  $p(x)$  gives the maximum of the entropy  $H(x)$ ?*

The Maximum Entropy is defined for the current case as

$$H(X) = - \int_{-\infty}^{\infty} p(x) \log p(x) dx,$$

is subject to the constraint that it should be a pdf

$$\int_{-\infty}^{\infty} p(x) dx = 1,$$

and that  $\mu$  and  $\sigma$  are given (whereby only one constrained is needed, as the  $\mu$  is already included in the definition of  $\sigma$ ):

### C. Exponential Family and Maximum Entropy

$$\int_{-\infty}^{\infty} (x - \mu)^2 p(x) dx = \sigma^2.$$

Accordingly to the above mentioned technique the formulas are summarized in form of the *Lagrangian* equation:

$$\mathcal{L} = - \int_{-\infty}^{\infty} p(x) \log p(x) dx + \lambda_0 \left( \int_{-\infty}^{\infty} p(x) dx - 1 \right) + \lambda_1 \left( \int_{-\infty}^{\infty} (x - \mu)^2 p(x) dx - \sigma^2 \right).$$

Next,  $\mathcal{L}$  will be partially differentiated with respect to  $p(x)$ :

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial p(x)} &= 0, \\ -(1 + \log p(x)) + \lambda_0 + \lambda_1(x - \mu)^2 &= 0, \end{aligned}$$

$$p(x) = e^{\lambda_0 + \lambda_1(x - \mu)^2 - 1}.$$

Further we have to make sure that the result holds for the stated constraints:

$$\int_{-\infty}^{\infty} e^{\lambda_0 + \lambda_1(x - \mu)^2 - 1} dx = 1,$$

and

$$\int_{-\infty}^{\infty} (x - \mu)^2 e^{\lambda_0 + \lambda_1(x - \mu)^2 - 1} dx = \sigma^2.$$

For the first constraint we get

$$e^{\lambda_0 - 1} \sqrt{-\frac{\pi}{\lambda_1}} = 1,$$

and for the second constraint

$$e^{\lambda_0 - 1} = \sqrt{\frac{1}{2\pi}} \frac{1}{\sigma},$$

Thus

$$\lambda_1 = \frac{-1}{2\sigma^2}$$

## C.2. Excursos: “Information Entropy” and “Maximum Entropy Principal”

Taking all together we can write:

$$p(x) = e^{\lambda_0 + \lambda_1(x-\mu)^2 - 1} = e^{\lambda_0 - 1} e^{\lambda_1(x-\mu)^2},$$

substituting the solutions for  $e^{\lambda_0 - 1}$  and  $\lambda_1$ :

$$p(x) = \sqrt{\frac{1}{2\pi}} \frac{1}{\sigma} e^{\frac{-1}{2\sigma^2}(x-\mu)^2},$$

finally we can rearrange the terms a bit and get:

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(\frac{-1}{2}\left(\frac{(x-\mu)^2}{\sigma^2}\right)\right),$$

the *Gaussian probability density distribution*.

**To sum it up:**

If one is to infer a probability distribution given certain constraints, out of all distributions  $\{p_i\}$  compatible with them, one should pick the distribution  $\{p_i^*\}$  having the largest value of  $H$  (De Martino and De Martino 2018). In other terms, a Maximum Entropy distribution is completely undetermined by features that do not appear explicitly in the constraints subject to which it has been computed.

An **overview of Maximum Entropy distributions** can be found on Wikipedia.



## D. Data sets used in the book

Several data sets are used throughout the book as ‘running examples’. They occur in different places to illustrate different things. This chapter centrally describes each data set, together with the most important visualizations and analyses.

### D.1. Mental Chronometry

#### D.1.1. Nature, origin and rationale of the data

Francis Donders is remembered as one of, if not the first experimental cognitive psychologists. He famously introduced the **subtraction logic** which looks at difference in reaction times across different tasks to infer difference in the complexity of the mental processes involved in these tasks. The Mental Chronometry data set presents the results of an online replication of one such subtraction-experiment.

##### D.1.1.1. The experiment

50 participants were recruited using the crowd-sourcing platform Prolific and paid for their participation.

In each experiment trial, participants see either a blue square or a blue circle appear on the screen and are asked to respond as quickly as possible. The experiment consists of three parts, presented to all participants in the same order (see below). The parts differ in the adequate response to the visual stimuli.

###### 1. Reaction task

The participant presses the space bar whenever there is a stimulus (square or circle)

*Recorded:* reaction time

###### 2. Go/No-Go task

The participant presses the space bar whenever their target (one of the two stimuli) is on the screen

*Recorded:* the reaction time and the response

## D. Data sets used in the book

### 3. Discrimination task

The participant presses the **F** key on the keyboard when there is one of the stimuli and the **J** key when there is the other one of the stimuli on the screen.

*Recorded:* the reaction time and the response

The **reaction time** measurement starts from the onset of the visual stimuli to the button press. The **response** variable records whether the reaction was correct or incorrect.

For each participant, the experiment randomly allocates one shape (circle or square) as the target to be used in both the second and the third task.

The experiment was realized using `_magpie` and can be tried out here.

#### D.1.1.2. Theoretical motivation & hypotheses

We expect that reaction times of correct responses are lowest in the reaction task, higher in the Go/No-Go task, and highest in the discrimination task.

#### D.1.2. Loading and preprocessing the data

The raw data produced by the online experiment is not particularly tidy. It needs substantial massages before plotting and analysis.

```
d_raw <- read_csv('data_sets/mental-chromo-data_raw.csv')
glimpse(d_raw)

## Observations: 3,750
## Variables: 32
## $ submission_id <dbl> 8554, 8554, 8554, 8554, 8554, 8554, 8554, 8554, 8554, ...
## $ QUD           <chr> "Press SPACE when you see a shape on the screen", "Pr...
## $ RT            <dbl> 376, 311, 329, 270, 284, 311, 269, 317, 325, 240, 262...
## $ age           <dbl> 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 2...
## $ comments       <chr> NA, N...
## $ correctness    <chr> "correct", "correct", "correct", "correct", "correct"...
## $ education      <chr> "high school / college", "high school / college", "hi...
## $ elemSize        <dbl> 100, 100, 100, 100, 100, 100, 100, 100, 100, 100...
## $ endTime         <dbl> 1.570374e+12, 1.570374e+12, 1.570374e+12, 1.570374e+1...
## $ expected        <chr> NA, N...
## $ experiment_id   <dbl> 68, 68, 68, 68, 68, 68, 68, 68, 68, 68, 68, 68, 6...
## $ f              <chr> NA, N...
## $ focalColor      <chr> "blue", "blue", "blue", "blue", "blue", "blue", "blue...
## $ focalNumber     <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## $ focalShape      <chr> "square", "square", "circle", "square", "circle", "ci...
```

```

## $ gender          <chr> "female", "female", "female", "female", "female", "fe...
## $ j               <chr> NA, N...
## $ key1            <lgl> NA, N...
## $ key2            <chr> NA, N...
## $ key_pressed     <chr> NA, N...
## $ languages        <chr> "Right", "Right", "Right", "Right", "Right", ...
## $ pause             <dbl> 2631, 1700, 1322, 1787, 1295, 2330, 1620, 2460, 1580, ...
## $ response          <chr> "space", "space", "space", "space", "space", ...
## $ sort              <chr> "grid", "grid", "grid", "grid", "grid", "grid...
## $ startDate         <chr> "Sun Oct 06 2019 15:45:19 GMT+0100 (Hora de verão da ...
## $ startTime         <dbl> 1.570373e+12, 1.570373e+12, 1.570373e+12, 1.570373e+1...
## $ stimulus           <chr> "square", "square", "circle", "square", "circle", "ci...
## $ target             <chr> "square", "square", "circle", "square", "circle", "ci...
## $ timeSpent         <dbl> 7.2514, 7.2514, 7.2514, 7.2514, 7.2514, 7.2514, 7.251...
## $ total              <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## $ trial_number       <dbl> 1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, ...
## $ trial_type         <chr> "reaction_practice", "reaction_practice", "reaction_p...

```

The most pressing problem is that entries in the column `trial_type` contain two logically separate pieces of information: the block (reaction, go/no-go, discrimination) *and* whether the data comes from a practice trial (which we want to discard) or a main trial (which we want to analyze). We therefore separate this information, and perform some other massages, to finally select a preprocessed data set for further analysis:

```

block_levels <- c("reaction", "goNoGo", "discrimination") # ordering of blocks for plotting, etc

d_preprocessed <- d_raw %>%
  separate(trial_type, c("block", "stage"), sep = "_", remove = FALSE) %>%
  mutate(comments = ifelse(is.na(comments), "non given", comments)) %>%
  filter(stage == "main") %>%
  mutate(
    block = factor(block, ordered = T, levels = block_levels),
    response = ifelse(is.na(response), "none", response)
  ) %>%
  filter(response != "wait") %>%
  rename(
    handedness = languages, # variable name is simply wrong
    total_time_spent = timeSpent
  ) %>%
  select(
    submission_id,
    trial_number,
    block,
    stage
  )

```

#### D. Data sets used in the book

```
    stimulus,  
    RT,  
    handedness,  
    gender,  
    total_time_spent,  
    comments  
)  
  
# write_csv(d_preprocessed, 'mental-chrono-data_preprocessed.csv')
```

##### D.1.3. Cleaning the data

Remeber that the criteria for data exclusion should ideally be defined before data collection (or at least inspection). They should definitely never be chosen in such a way as to maximize the “desirability” of an analysis. Data cleaning is not a way of making sure that your favorite research hypothesis “wins”.

Although we have not preregistered any data cleaning regime or analyses for this data set, we demonstrate a frequently used cleaning scheme for reaction time data, which does depend on the data in some sense, but does not require precise knowledge of the data. In particular, we are going to do this:

1. We remove remove the data from an individual participant  $X$  if there is an experimental condition  $C$  such that the mean RT of  $X$  for condition  $C$  is more than 2 standard deviations away from the overal mean RT for condition  $C$ .
2. From the remaining data, we then remove any individual trial  $Y$  if the RT of  $Y$  is more than 2 standard deviations away from the mean of experimental condition  $C$  (where  $C$  is the condition of  $Y$ , of course).

Notice that in the case at hand, the experimental conditions are the three types of tasks.

###### D.1.3.1. Cleaning by-participant

Our rule for removing data from outlier participants is this:

We remove remove the data from an individual participant  $X$  if there is an experimental condition  $C$  such that the mean RT of  $X$  for condition  $C$  is more than 2 standard deviations away from the overal mean RT for condition  $C$ . We also remove all trials with reaction times below 100ms.

This procedure is implemented in this code:

```

# summary stats (means) for participants
d_sum_stats_participants <- d_preprocessed %>%
  group_by(submission_id, block) %>%
  summarise(
    mean_P = mean(RT)
  )

# summary stats (means and SDs) for conditions
d_sum_stats_conditions <- d_preprocessed %>%
  group_by(block) %>%
  summarise(
    mean_C = mean(RT),
    sd_C   = sd(RT)
  )

d_sum_stats_participants <-
  full_join(
    d_sum_stats_participants,
    d_sum_stats_conditions,
    by = "block"
  ) %>%
  mutate(
    outlier_P = abs(mean_P - mean_C) > 2 * sd_C
  )

# show outlier participants
d_sum_stats_participants %>% filter(outlier_P == 1) %>% show()

## # A tibble: 1 x 6
## # Groups:   submission_id [1]
##   submission_id block      mean_P  mean_C   sd_C outlier_P
##       <dbl> <ord>      <dbl>  <dbl>   <dbl>   <lgl>
## 1         8505 discrimination 1078.   518.   185.  TRUE

```

When plotting the data for this condition and this participant, we see that the high overall mean is not just caused by a single outlier, but several trials that took longer than 1 second.

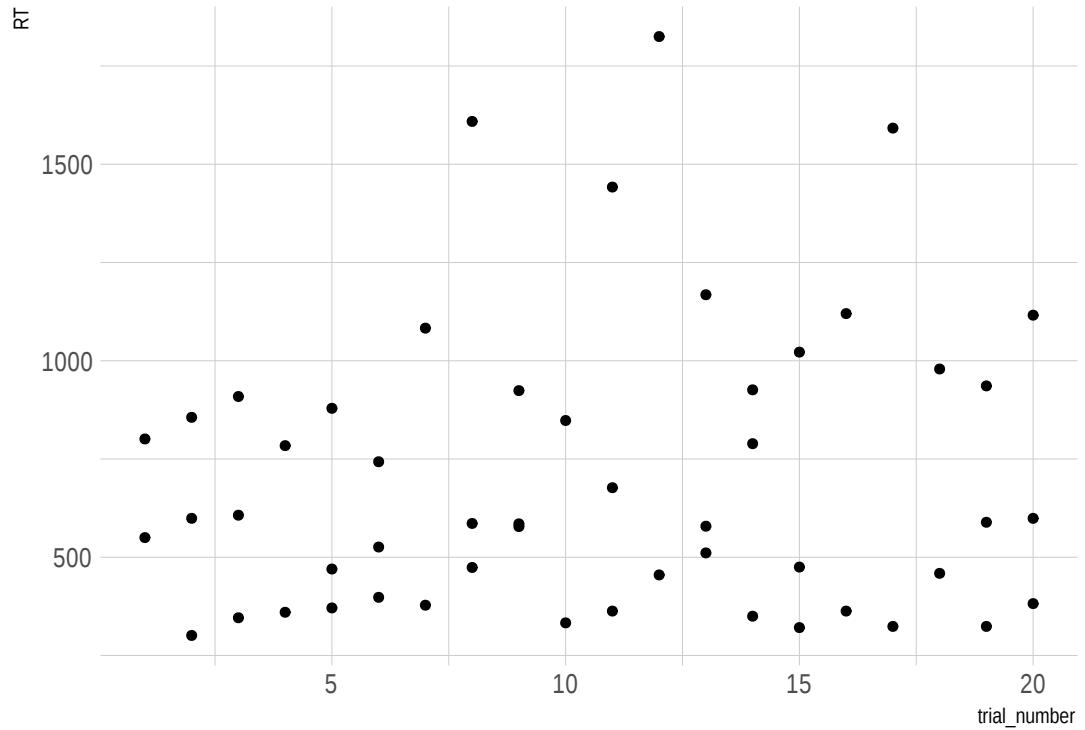
```

d_preprocessed %>%
  semi_join(
    d_sum_stats_participants %>% filter(outlier_P == 1),
    by = c("submission_id")
  )

```

#### D. Data sets used in the book

```
) %>%
ggplot(aes(x = trial_number, y = RT)) +
geom_point()
```



We are then going to exclude this participant's entire data from all subsequent analysis:<sup>1</sup>

```
d_cleaned <- d_preprocessed %>%
filter(submission_id != d_sum_stats_participants$submission_id[1] )
```

##### D.1.3.2. Cleaning by-trial

Our rule for excluding data from individual trials is:

From the remaining data, we then remove any individual trial  $Y$  if the RT of  $Y$  is more than 2 standard deviations away from the mean of experimental condition  $C$  (where  $C$  is the condition of  $Y$ , of course). We also remove all trials with reaction times below 100ms.

---

<sup>1</sup>This may seem a harsh step, but when data acquisition is cheap, it's generally not a bad strategy to be very strict in exclusion criteria, and to apply rules that are not strongly context-dependent.

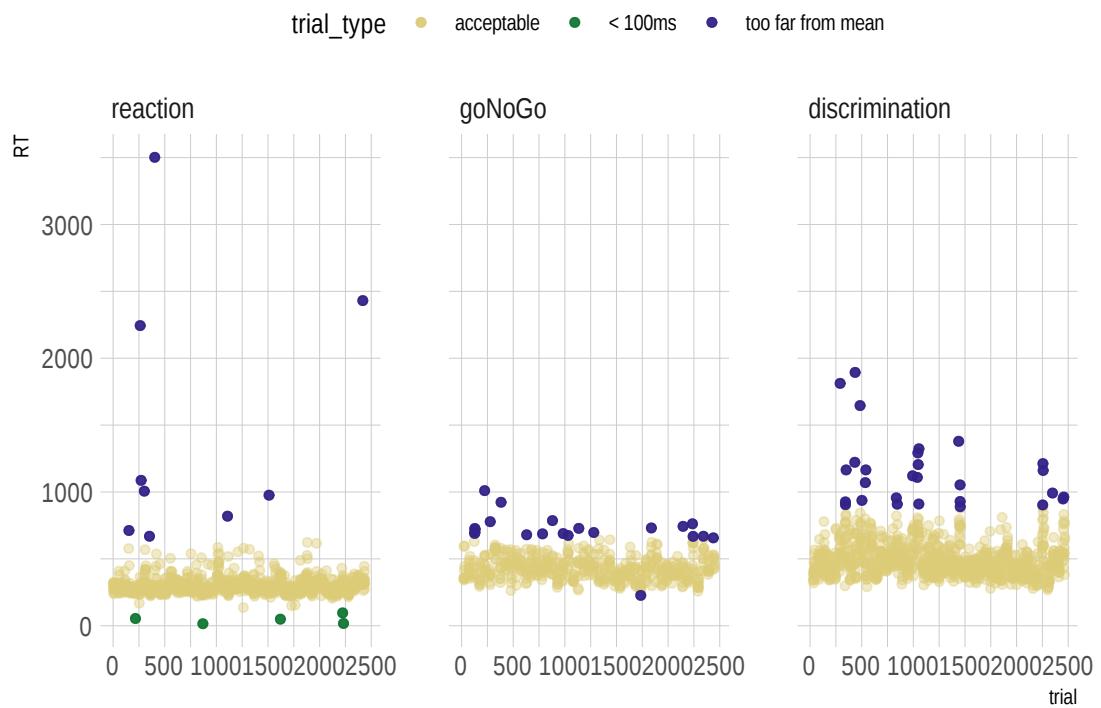
The following code implements this:

```
# mark individual trials as outliers
d_cleaned <- d_cleaned %>%
  full_join(
    d_sum_stats_conditions,
    by = "block"
  ) %>%
  mutate(
    trial_type = case_when(
      abs(RT - mean_C) > 2 * sd_C ~ "too far from mean",
      RT < 100 ~ "< 100ms",
      TRUE ~ "acceptable"
    ) %>% factor(levels = c("acceptable", "< 100ms", "too far from mean")),
    trial = 1:nrow(d_cleaned)
  )

# visualize outlier trials

d_cleaned %>%
  ggplot(aes(x = trial, y = RT, color = trial_type)) +
  geom_point(alpha = 0.4) + facet_grid(~block) +
  geom_point(alpha = 0.9, data = filter(d_cleaned, trial_type != "acceptable"))
```

#### D. Data sets used in the book



So, we remove 63 individual trials.

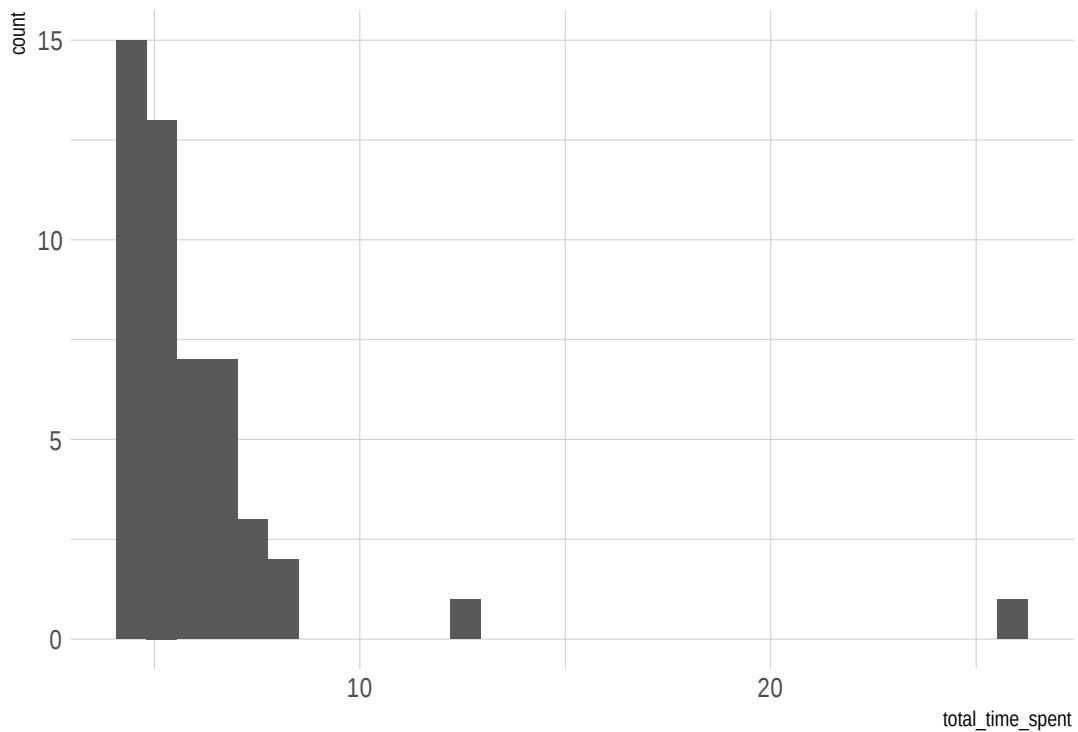
```
d_cleaned <- d_cleaned %>%
  filter(trial_type == "acceptable")

## this version of the data is stored as cleaned
# write_csv(d_cleaned, "data_sets/mental-chrono-data_cleaned.csv")
```

#### D.1.4. Exploration: summary stats & plots

What's the distribution of `total_time_spent`, i.e., the time each participant took to complete the whole study?

```
d_cleaned %>%
  select(submission_id, total_time_spent) %>%
  unique() %>%
  ggplot(aes(x = total_time_spent)) +
  geom_histogram()
```



There are two participants who took noticeably longer than all the others, but we need not necessarily be concerned about this, because it is not unusual for participants of online experiments to open the experiment and wait before actually starting.

Here are summary statistics for the reaction time measures for each condition (= block).

```
d_sum_stats_blocks_cleaned <- d_cleaned %>%
  group_by(block) %>%
  nest() %>%
  summarise(
    CIIs = map(data, function(d) bootstrapped_CI(d$RT))
  ) %>%
  unnest(CIIs)

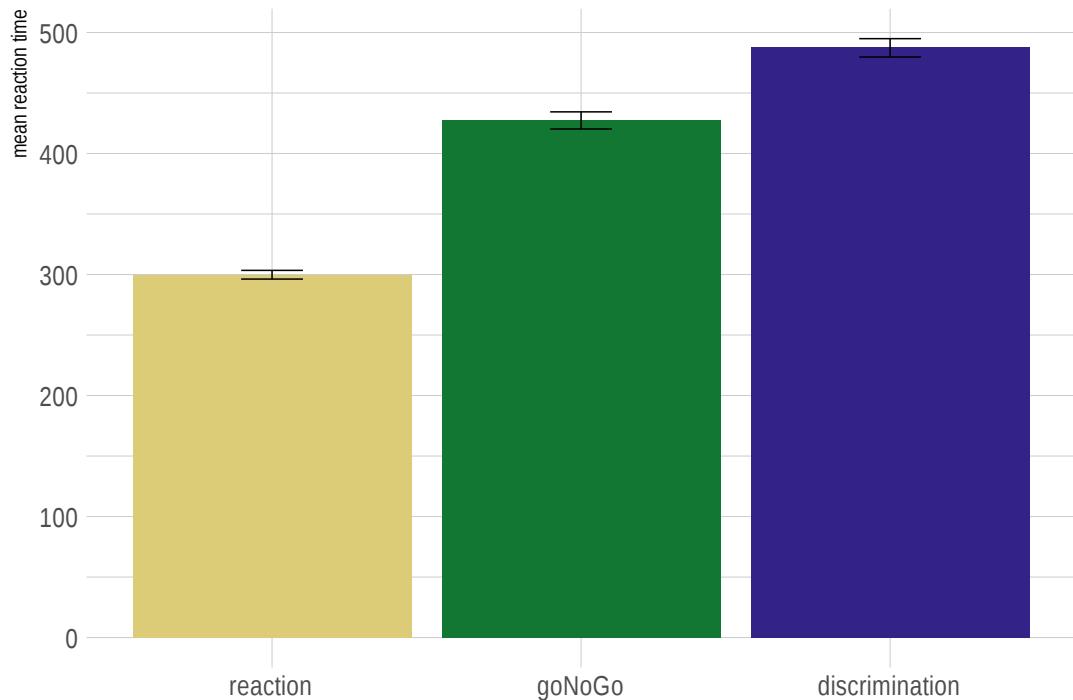
d_sum_stats_blocks_cleaned

## # A tibble: 3 x 4
##   block      lower  mean upper
##   <ord>     <dbl> <dbl> <dbl>
## 1 reaction  296.  300.  303.
## 2 goNoGo   420.  427.  434.
## 3 discrimination 480. 488. 495.
```

#### D. Data sets used in the book

And a plot of the summary:

```
d_sum_stats_blocks_cleaned %>%
  ggplot(aes(x = block, y = mean, fill = block)) +
  geom_col() +
  geom_errorbar(aes(ymin = lower, ymax = upper), size = 0.3, width = 0.2) +
  ylab("mean reaction time") + xlab("") +
  scale_fill_manual(values = project_colors) +
  theme(legend.position = "none")
```



#### D.1.5. Data analysis

We are interested in seeing whether the mean RTs are smallest for the ‘reaction’ task, higher for the ‘go/no-go’ task, and highest for the ‘discrimination’ task. We test this with a hierarchical Bayesian regression model, taking participant-level variation of intercepts and slopes for factor `block` into account. We make ‘go/no-go’ the default level of the `block` factor, so that we can directly test our directed hypothesis, using posterior parameter inference.

```

# making 'go/no-go' the reference level
## TODO : write convenience function for this kind of releveling
reflevel <- "goNoGo"
reflevel_index <- which(levels(d_cleaned$block) == reflevel)
contrasts(d_cleaned$block) <- contr.treatment(
  nlevels(d_cleaned$block),
  reflevel_index
)
colnames(contrasts(d_cleaned$block)) <- str_c("_",levels(d_cleaned$block)[-reflevel_index])

regression_model_ME <- brm(
  formula = RT ~ block + (1 + block | submission_id),
  data = d_cleaned
)

## TODO tidy and concise output
regression_model_ME

```

We see that the value zero lies clearly outside of the 95% credible interval for block ‘reaction’ and for block ‘discrimination’. The deviation from the intercept (zero point) is in the expected direction. We may conclude that, as hypothesized, reaction times in the ‘reaction’ condition are lowest, higher in the ‘go/no-go’ condition, and highest in the ‘discrimination’ condition.

## D.2. Simon Task

CAVEAT: THIS CHAPTER IS A DRAFT; DEFER READING UNTIL LATER

The Simon task is pretty cool. The task is designed to see if responses are faster and/or more accurate when the stimulus to respond to occurs in the same relative location as the response, even if the stimulus location is irrelevant to the task. For example, it is faster to respond to a stimulus presented on the left of the screen with a key that is on the left of the keyboard (e.g. q), than with a key that is on the right of the keyboard (e.g. p).

### D.2.1. Experiment

#### D.2.1.1. Participants

A total of 213 participants took part in an online version of a Simon task. Participants were students enrolled in either “Introduction to Cognitive (Neuro-)Psychology” (N = 166), or “Experimental Psychology Lab Practice” (N = 39) or both (N = 4).

## D. Data sets used in the book

### D.2.1.2. Materials & Design

Each trial started by showing a fixation cross for 200 ms in the center of the screen. Then, one of two geometrical shapes was shown for 500 ms. The **target shape** was either a blue square or a blue circle. The target shape appeared either on the left or right of the screen. Each trial determined uniformly at random which shape (square or circle) to show as target and where on the screen to display it (left or right). Participants were instructed to press keys **q** (left of keyboard) or **p** (right of keyboard) to identify the kind of shape on the screen. The shape-key allocation happened experiment initially, uniformly at random once for each participant and remained constant throughout the experiment. For example, a participant may have been asked to press **q** for square and **p** for circle.

Trials were categorized as either ‘congruent’ or ‘incongruent’. They were congruent if the location of the stimulus was the same relative location as the response key (e.g. square on the right of the screen, and **p** key to be pressed for square) and incongruent if the stimulus was not in the same relative location as the response key (e.g. square on the right and **q** key to be pressed for square).

In each trial, if no key was pressed within 3 seconds after the appearance of the target shape, a message to please respond faster was displayed on screen.

### D.2.1.3. Procedure

Participants were first welcomed and made familiar with the experiment. They were told to optimize both speed and accuracy. They then practiced the task for 20 trials before starting the main task, which consisted of 100 trials. Finally, the experiment ended with a post-test survey in which participants were asked for their student IDs and the class they were enrolled in. They were also able to leave any optional comments.

## D.2.2. Results

### D.2.2.1. Loading and inspecting the data

We load the data into R and show a summary of the variables stored in the tibble:

```
d <- read_csv("data_sets/simon-task.csv")
glimpse(d)

## Observations: 25,560
## Variables: 15
## $ submission_id    <dbl> 7432, 7432, 7432, 7432, 7432, 7432, 7432, 7432, 743...
## $ RT                <dbl> 1239, 938, 744, 528, 706, 547, 591, 652, 627, 485, ...
```

```

## $ condition      <chr> "incongruent", "incongruent", "incongruent", "incon...
## $ correctness    <chr> "correct", "correct", "correct", "correct", "correct", ...
## $ class          <chr> "Intro Cogn. Neuro-Psychology", "Intro Cogn. Neuro-...
## $ experiment_id  <dbl> 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, ...
## $ key_pressed    <chr> "q", "q", "q", "q", "p", "p", "q", "p", "q", "q", ...
## $ p              <chr> "circle", "circle", "circle", "circle", "circle", "circle", ...
## $ pause           <dbl> 1896, 1289, 1705, 2115, 2446, 2289, 2057, 2513, 186...
## $ q              <chr> "square", "square", "square", "square", "square", "square", ...
## $ target_object   <chr> "square", "square", "square", "square", "circle", "circle", ...
## $ target_position <chr> "right", "right", "right", "right", "left", "right", ...
## $ timeSpent       <dbl> 7.565417, 7.565417, 7.565417, 7.565417, 7.565417, 7.565417, ...
## $ trial_number    <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, ...
## $ trial_type      <chr> "practice", "practice", "practice", "practice", "pr...

```

It is often useful to check general properties, such as the mean time participants spent on the experiment:

```
d %>% pull(timeSpent) %>% mean()
```

```
## [1] 21.61656
```

About 21.62 minutes is quite long, but we know that the mean is very susceptible to outliers, so we may want to look at a more informative set of **summary statistics**:

```
d %>% pull(timeSpent) %>% summary()
```

|    | Min.  | 1st Qu. | Median | Mean   | 3rd Qu. | Max.     |
|----|-------|---------|--------|--------|---------|----------|
| ## | 5.648 | 6.905   | 7.692  | 21.617 | 9.113   | 1158.110 |

### D.2.2.2. Summarizing & cleaning the data

We look at outlier-y behavior at the level of individual participants first, then at the level of individual trials.

#### D.2.2.2.1. Individual-level error rates & reaction times

It is conceivable that some participants did not take the task seriously. They may have just fooled around. We will therefore inspect each individual's response patterns and reaction times. If participants appear to have “misbehaved” we discard all of their data. (**CAVEAT:** Notice the researcher degrees of freedom in the decision of what counts as “misbehavior”! It is therefore that choices like these are best committed to in advance, e.g. via pre-registration!)

We can calculate the mean reaction times and the error rates for each participant.

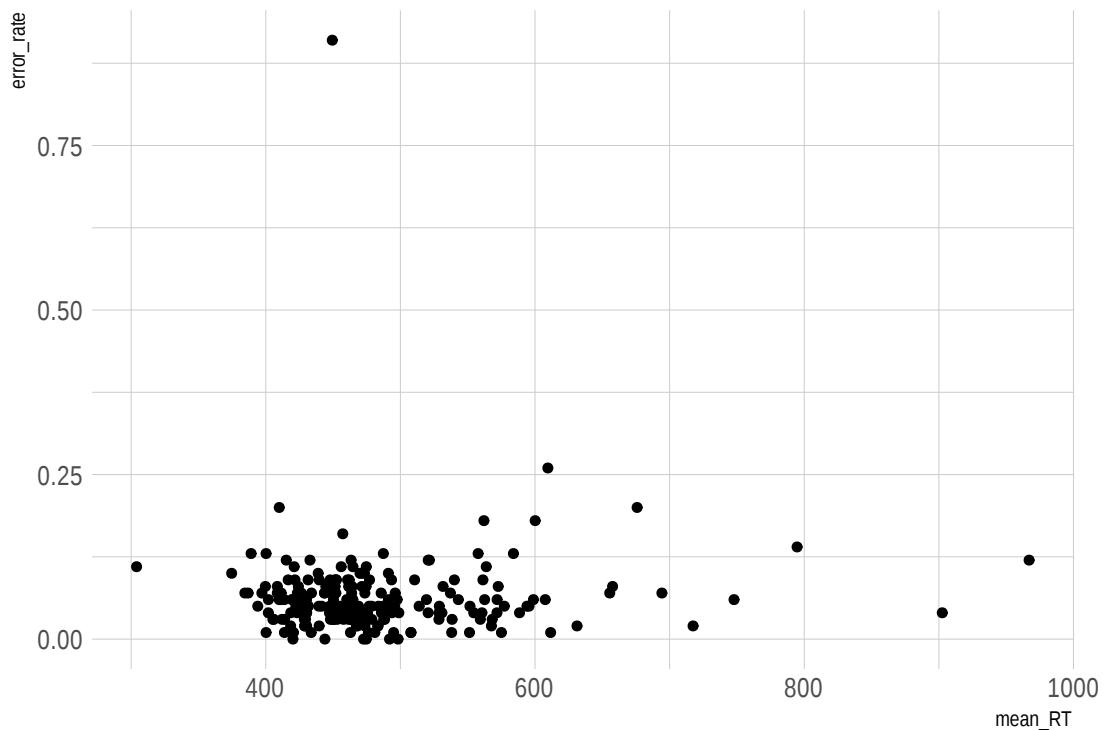
#### D. Data sets used in the book

```
d_individual_summary <- d %>%
  filter(trial_type == "main") %>%      # look at only data from main trials
  group_by(submission_id) %>%            # calculate the following for each individual
  summarize(mean_RT = mean(RT),
            error_rate = 1 - mean(ifelse(correctness == "correct", 1, 0)))
head(d_individual_summary)

## # A tibble: 6 x 3
##   submission_id  mean_RT  error_rate
##       <dbl>     <dbl>      <dbl>
## 1         7432     595.      0.05
## 2         7433     458.      0.04
## 3         7434     531.      0.04
## 4         7435     433.      0.12
## 5         7436     748.      0.06
## 6         7437     522.      0.12
```

Let's plot this summary information:

```
d_individual_summary %>%
  ggplot(aes(x = mean_RT, y = error_rate)) +
  geom_point()
```

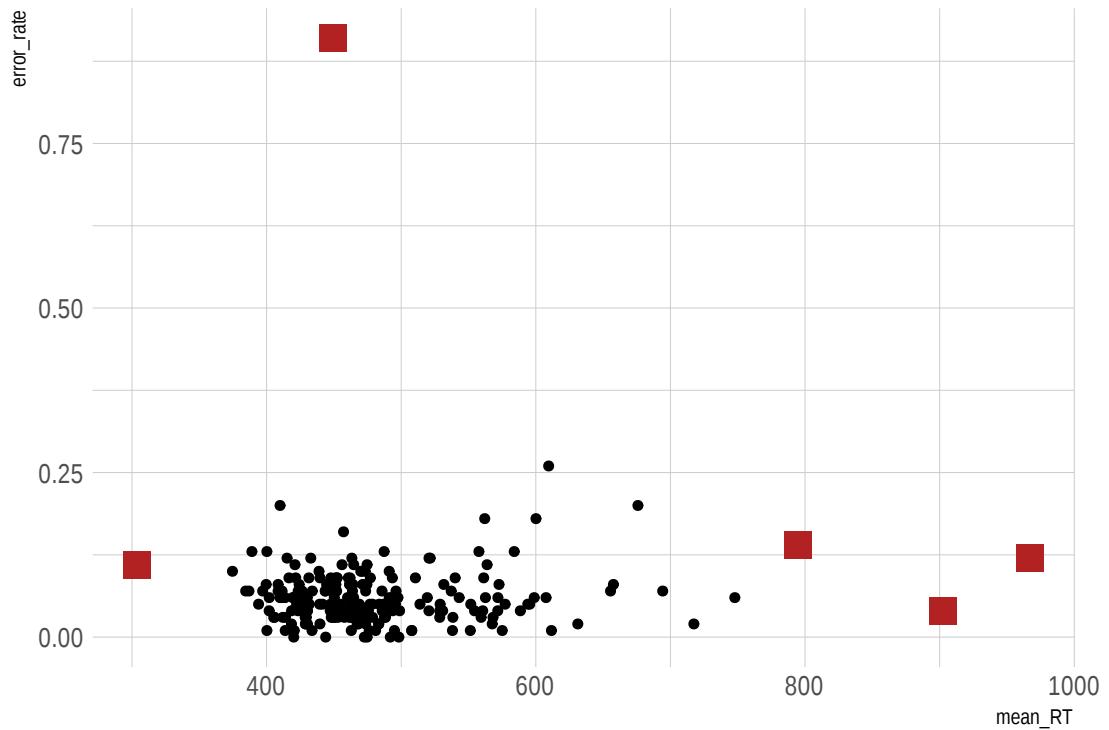


Here's a crude way of branding outlier-participants:

```
d_individual_summary <- d_individual_summary %>%
  mutate(outlier = case_when(mean_RT < 350 ~ TRUE,
                             mean_RT > 750 ~ TRUE,
                             error_rate > 0.5 ~ TRUE,
                             TRUE ~ FALSE))

d_individual_summary %>%
  ggplot(aes(x = mean_RT, y = error_rate)) +
  geom_point() +
  geom_point(data = filter(d_individual_summary, outlier == TRUE),
             color = "firebrick", shape = "square", size = 5)
```

#### D. Data sets used in the book



We then clean the data set in a first step by removing all participants identified as outlier-y:

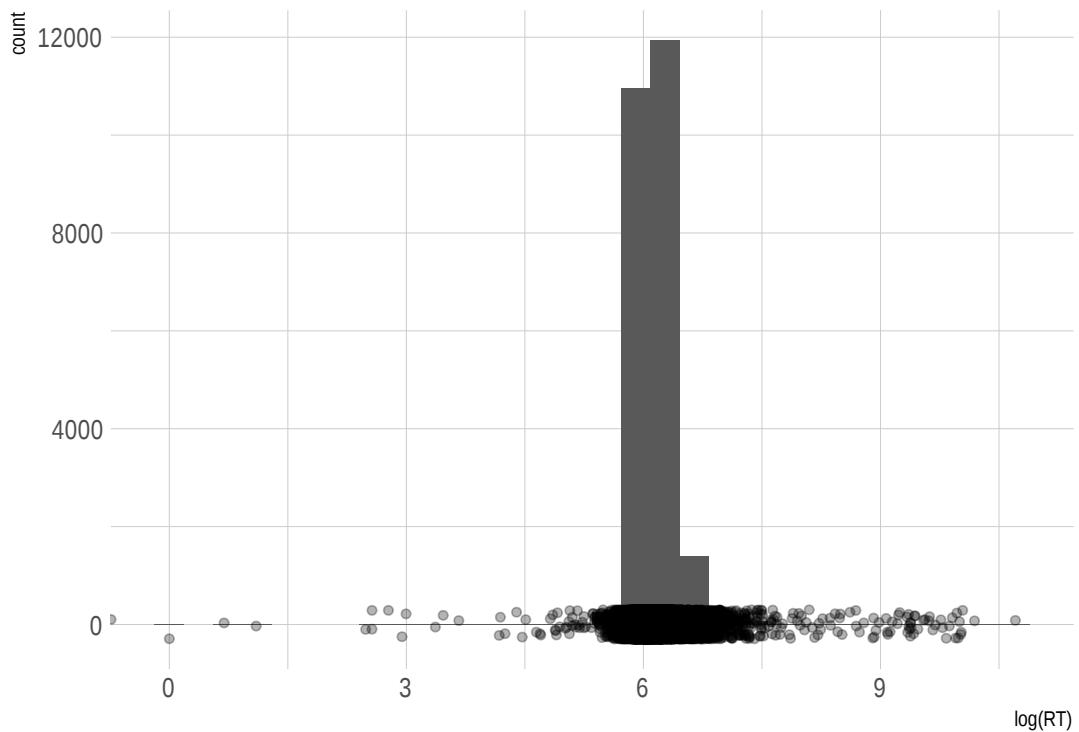
```
d <- full_join(d, d_individual_summary, by = "submission_id") # merge the tibbles
d <- filter(d, outlier == FALSE)
message("We excluded ", sum(d_individual_summary$outlier) , " participants for suspicious mean RTs and higher error rates.")

## We excluded 5 participants for suspicious mean RTs and higher error rates.
```

##### D.2.2.2. Trial-level reaction times

It is also conceivable that individual trials resulted in early accidental key presses or were interrupted in some way or another. We therefore look at the overall distribution of RTs and determine (similarly arbitrarily, but once again this should be planned in advance) what to exclude.

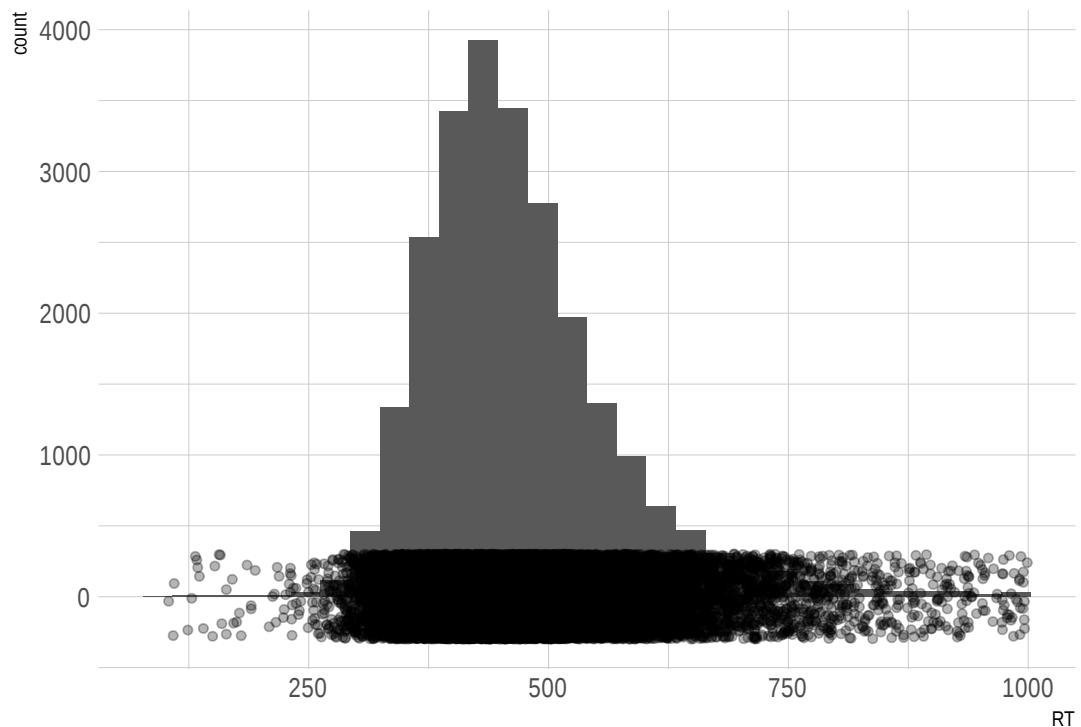
```
d %>% ggplot(aes(x = log(RT))) +
  geom_histogram() +
  geom_jitter(aes(x = log(RT), y = 1), alpha = 0.3, height = 300)
```



Let's decide to exclude all trials that lasted longer than 1 second and also all trials with reaction times under 100 ms.

```
d <- filter(d, RT > 100 & RT < 1000)
d %>% ggplot(aes(x = RT)) +
  geom_histogram() +
  geom_jitter(aes(x = RT, y = 1), alpha = 0.3, height = 300)
```

#### D. Data sets used in the book



##### D.2.2.3. Exploring the (main) data

We are mostly interested in the influence of congruency on the reaction times in the trials where participants gave a correct answer. But here we also look at, for comparison, the reaction times for the incongruent trials.

Here is a summary of the means and standard deviations for each condition:

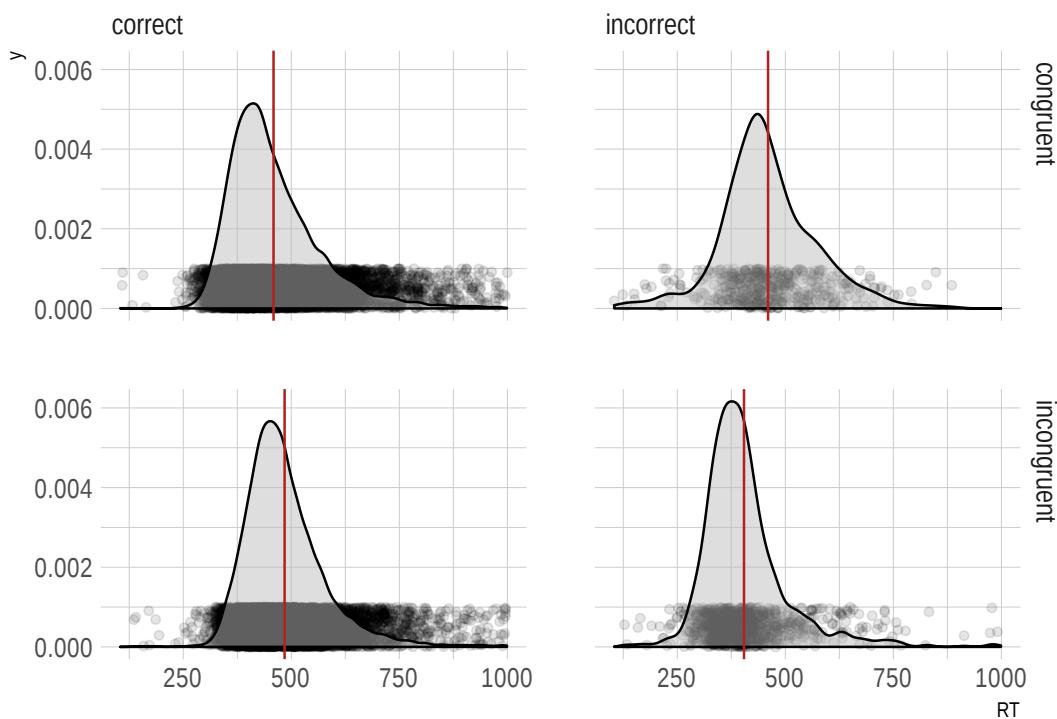
```
d_sum <- d %>%
  group_by(correctness, condition) %>%
  summarize(mean_RT = mean(RT),
            sd_RT = sd(RT))
d_sum

## # A tibble: 4 x 4
## # Groups:   correctness [2]
##   correctness condition   mean_RT   sd_RT
##   <chr>       <chr>        <dbl>    <dbl>
## 1 correct      congruent     459.    105.
## 2 correct      incongruent   484.    91.9
## 3 incorrect    congruent     460.    111.
```

```
## 4 incorrect    incongruent     404.   95.4
```

Here's a plot of the reaction times split up by whether the answer was correct and whether the trial was congruent or incongruent.

```
d %>% ggplot(aes(x = RT)) +
  geom_jitter(aes(y = 0.0005), alpha = 0.1, height = 0.0005) +
  geom_density(fill = "gray", alpha = 0.5) +
  geom_vline(data = d_sum,
             mapping = aes(xintercept = mean_RT),
             color = "firebrick") +
  facet_grid(condition ~ correctness)
```



### D.2.3. Analysis

We are interested in comparing the RTs of correct answers in the congruent and incongruent conditions. We saw a difference in mean reaction times, but we'd like to know if this difference is meaningful. One way of testing this is by running a regression model, which tries to predict RTs as a function of congruency. In the simplest case we would therefore do this:

#### D. Data sets used in the book

```
model_ST_simple = brm(RT ~ condition, filter(d, correctness == "correct"))
summary(model_ST_simple)
```

According to this analysis, there is reason to believe in a difference in RTs between congruent and incongruent groups. The coefficient estimated for the incongruent group is on average ca. 25 ms higher than that of the congruent group.

However, we can also look at the interaction between correctness and condition. As shown in the above graph, there are four different cells in a 2x2 grid.

In the below model, this is coded with ‘dummy coding’ such that the top-left cell (congruent-correct) is the intercept, and each other cell is calculated by the addition of offsets.

```
model_ST_complex <- brm(RT ~ condition * correctness, d)
```

We may want to ask the question: are reaction times to correct-congruent responses shorter than reaction times to incorrect-incongruent responses?

To do this, we first need to extract the posterior samples from our model.

```
post_samples <- posterior_samples(model_ST_complex) %>%
  as_tibble()
```

Then we need to determine the correct offsets to match the correct-congruent and incorrect-incongruent cells in the design matrix.

```
# correct-congruent is the reference cell
correct_congruent <- post_samples$b_Intercept

# incorrect_incongruent is the bottom-right cell
incorrect_incongruent <- post_samples$b_Intercept +
  post_samples$b_conditionincongruent +
  post_samples$b_correctnessincorrect +
  post_samples$b_conditionincongruent:correctnessincorrect`
```

Once we know these, we can calculate the probability that the comparison is in the correct direction.

```
mean(correct_congruent < incorrect_incongruent)
```

## D.3. World Values Survey (wave 6 | 2010-2014)

### D.3.1. Nature, origin and rationale of the data

The World Values Survey (WVS) aims to study *changing values and their impact on social and political life*. The WVS consists of nationally representative surveys conducted in almost 100 countries which contain almost 90 percent of the world's population, using a common questionnaire. The WVS is the largest non-commercial, cross-national, time series investigation of human beliefs and values.

It currently includes interviews with almost 400,000 respondents. Respondents are people in the age 18 and older residing within private households in each country, regardless of their nationality, citizenship or language.

The main method of data collection in the WVS survey is *face-to-face interview* at respondent's home / place of residence.

->

-> ->

->

->

->

-> -> ->

-> -> -> -> -> -> -> ->

-> ->

-> ->

## D.4. King of France

### D.4.1. Nature, origin and rationale of the data

A **presupposition** of a sentence is a piece of information that is necessary for the sentence to make sense, but which is not communicated explicitly. If I say "Jones chained my camel to a tree", this sentence presupposes, somewhat incredibly, that I own a camel. If it is false that I own a camel, the sentence makes no sense. Yet, if I say it and you say: "I disagree" you take issue with my claim about chaining, not about me owning a camel. In this sense, the presupposition is not part of the explicitly contributed content (it is "not at issue content", as the linguists would say).

We here partially replicate a previous study by Abrusán and Szendrői (2013) investigating how sentences with false presuppositions are perceived. The main question of

## D. Data sets used in the book

interest for us is whether sentences with a false presupposition are rather regarded as true or rather as false. We therefore present participants with sentences (see below) and have them rate these as ‘true’ or ‘false’, a so-called **truth-value judgement task**, a common paradigm in experimental semantics and pragmatics. (The original study by Abrusán and Szendrői (2013) also included a third option ‘cannot tell’, which we do not use, since this data set is mainly used for toying around with binary choice data.)

Abrusán and Szendrői (2013) presented their participants with 11 different types of sentences, of which we here only focus on five. Here are examples of the five conditions we test, using the corresponding condition numbers from the experiment by Abrusán and Szendrői (2013).

**C0.** The king of France is bald.

**C1.** France has a king, and he is bald.

**C6.** The King of France isn’t bald.

**C9.** The King of France, he did not call Emmanuel Macron last night.

**C10.** Emmanuel Macron, he did not call the King of France last night.

The presupposition in question is “France has a king”. C0 and C1 differ only with respect to whether this piece of information is presupposed (C1) or explicitly asserted (C0). The variants C0 and C6 differ only with respect to negation in the main (asserted) proposition. Finally, the contrast pair C9 and C10 is interesting because of a particular topic-focus structure and the placement of negation. In C9 the topic is “the King of France” which introduces the presupposition in question. In C10 the topic is “Emmanuel Macron”, but it introduces the presupposition under a negation.

Figure D.1 shows the results reported by Abrusán and Szendrői (2013).

### D.4.1.1. The experiment

#### D.4.1.1.1. Participants

We obtained data from 97 participants via the online crowd-sourcing platform Prolific.<sup>2</sup> All participants were native speakers of English.

#### D.4.1.1.2. Material

The sentence material consisted of five vignettes. Here are the sentences that constitute “condition 1” of each of the five vignettes:

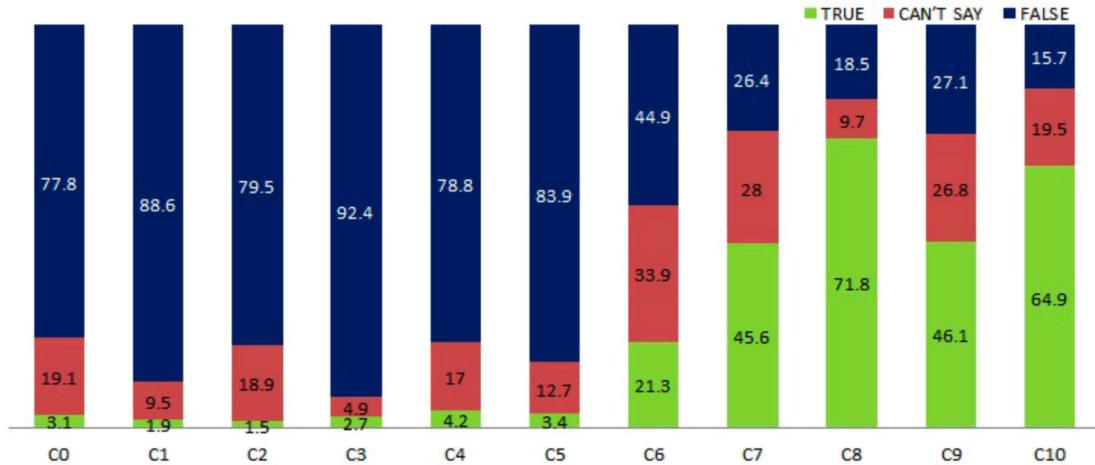
**V1.** The King of France is bald.

**V2.** The Emperor of Canada is fond of sushi.

---

<sup>2</sup>We recruited 100 participants, but the data from three participants was not recorded due to technical problems.

#### D.4. King of France



Proportion of TRUE/ CAN'T SAY/ FALSE responses in each condition

Figure D.1.: Results of @AbrusanSzendroi2013:Experimenting-w .

**V3.** The Pope's wife is a lawyer.

**V4.** The Belgian rainforest provides a habitat for many species.

**V5.** The volcanoes of Germany dominate the landscape.

As every vignette occurred in each of the five conditions, there are a total of 25 critical sentences. Additionally, for each vignette, there is a “background check” sentence which is intended to find out whether participants know whether the relevant presuppositions are true. The “background check” sentences are:

**BC1.** France has a king.

**BC2.** The Pope is currently not married.

**BC3.** Canada is a democracy.

**BC4.** Belgium has rainforests.

**BC5.** Germany has volcanoes.

Finally, there are also 110 filler sentences, which do not have a presupposition, but also require common world knowledge for a correct answer. As each filler has an uncontroversially correct answer, these fillers also serve as a general attention check, to probe into whether participants are reading the sentences carefully enough. Example filler sentences are:

**F1.** William Shakespeare was a famous Italian painter in Rome.

**F2.** There were two world wars in the 20th century.

## D. Data sets used in the book

### D.4.1.1.3. Procedure

Each experimental run started with five practice trials, which used the five additional sentences, which were like the filler material and the same for each participant, presented in random order.

The main part of the experiment presented each participant with five critical sentences, exactly one from each vignette and exactly one from each condition, allocated completely at random. Each participant also saw all of the five “background check” sentences. Each “background check” sentence was presented *after* the corresponding vignette’s critical sentence. All of these test trials were interspersed with 14 random filler sentences.

### D.4.1.1.4. Realization

The experiment was realized using `_magpie` and can be tried out here.

## D.4.1.2. Theoretical motivation & hypotheses

We will be concerned with the following two research questions:<sup>3</sup>

1. Is the overall rate (= aggregating over all vignettes & conditions) of “TRUE” judgements for sentences with presupposition failure different from pure guessing chance of 0.5?
2. Is there a difference in (binary) truth-value judgements (aggregated over all vignettes) between C0 (with presupposition) and C1 (where the presupposition is part of the at-issue / asserted content)?
3. Is there a difference in (binary) truth-value judgements (aggregated over all vignettes) between C0 (the positive sentence) and C6 (the negative sentence)?
4. Is there a difference in (binary) truth-value judgements (aggregated over all vignettes) between C9 (where the presupposition is topical) and C10 (where the presupposition is not topical and occurs under negation)?

## D.4.2. Loading and preprocessing the data

First, load the data:

```
## Observations: 2,813
## Variables: 16
## $ submission_id  <dbl> 192, 192, 192, 192, 192, 192, 192, 192, 192, 19...
## $ RT              <dbl> 8110, 35557, 3647, 16037, 11816, 6024, 4986, 13019, ...
## $ age              <dbl> 57, 57, 57, 57, 57, 57, 57, 57, 57, 57, 57, 57, 57, 57, ...
## $ comments         <chr> NA, ...
```

---

<sup>3</sup>These research questions are a compromise between actual theoretical relevance and practical (= educational) considerations.

```

## $ item_version <chr> "none", "none", "none", "none", "none", "none", "non...
## $ correct_answer <lgl> FALSE, TRUE, FALSE, TRUE, TRUE, TRUE, FALSE, FALSE, ...
## $ education <chr> "Graduated College", "Graduated College", "Graduated...
## $ gender <chr> "female", "female", "female", "female", "female", "f...
## $ languages <chr> "English", "English", "English", "English", "English...
## $ question <chr> "World War II was a global war that lasted from 1914...
## $ response <lgl> FALSE, TRUE, FALSE, TRUE, TRUE, TRUE, FALSE, FALSE, ...
## $ timeSpent <dbl> 39.48995, 39.48995, 39.48995, 39.48995, 39.48995, 39...
## $ trial_name <chr> "practice_trials", "practice_trials", "practice_tria...
## $ trial_number <dbl> 1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12...
## $ trial_type <chr> "practice", "practice", "practice", "practice", "pra...
## $ vignette <chr> "undefined", "undefined", "undefined", "undefined", ...

```

```
data_KoF_raw <- read_csv(url('https://raw.githubusercontent.com/michael-franke/intro-data-anal...'))
```

And then have a glimpse:

```
glimpse(data_KoF_raw)
```

```

## Observations: 2,813
## Variables: 16
## $ submission_id <dbl> 192, 192, 192, 192, 192, 192, 192, 192, 19...
## $ RT <dbl> 8110, 35557, 3647, 16037, 11816, 6024, 4986, 13019, ...
## $ age <dbl> 57, 57, 57, 57, 57, 57, 57, 57, 57, 57, 57, 57, ...
## $ comments <chr> NA, ...
## $ item_version <chr> "none", "none", "none", "none", "none", "none", ...
## $ correct_answer <lgl> FALSE, TRUE, FALSE, TRUE, TRUE, TRUE, FALSE, FALSE, ...
## $ education <chr> "Graduated College", "Graduated College", "Graduated...
## $ gender <chr> "female", "female", "female", "female", "female", "f...
## $ languages <chr> "English", "English", "English", "English", "English...
## $ question <chr> "World War II was a global war that lasted from 1914...
## $ response <lgl> FALSE, TRUE, FALSE, TRUE, TRUE, TRUE, FALSE, FALSE, ...
## $ timeSpent <dbl> 39.48995, 39.48995, 39.48995, 39.48995, 39.48995, 39...
## $ trial_name <chr> "practice_trials", "practice_trials", "practice_tria...
## $ trial_number <dbl> 1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12...
## $ trial_type <chr> "practice", "practice", "practice", "practice", "pra...
## $ vignette <chr> "undefined", "undefined", "undefined", "undefined", ...

```

The most important variables in this data set are:

- `submission_id`: unique identifier for each participant
- `trial_type`: whether the trial was of the category `filler`, `main`, `practice` or `special`, where the latter encodes the “background checks”

#### D. Data sets used in the book

- `item_version`: the condition which the test sentence belongs to (only given for trials of type `main` and `special`)
- `response`: the answer (“TRUE” or “FALSE”) on each trial
- `vignette`: the current item’s vignette number (applies only to trials of type `main` and `special`)

As the variable names used in the raw data are not ideal, we will pre-process the raw data a bit for easier analysis.

```
data_KoF_processed <- data_KoF_raw %>%
  # discard practice trials
  filter(trial_type != "practice") %>%
  mutate(
    # add a 'condition' variable
    condition = case_when(
      trial_type == "special" ~ "background check",
      trial_type == "main" ~ str_c("Condition ", item_version),
      TRUE ~ "filler"
    ) %>%
    factor(
      ordered = T,
      levels = c(str_c("Condition ", c(0, 1, 6, 9, 10)), "background check", "filler")
    )
  )
# write_csv(data_KoF_processed, "data_sets/king-of-france_data_processed.csv")
```

#### D.4.3. Cleaning the data

We clean the data in two consecutive steps:

1. Remove all data from any participant who got more than 50% of the answer to filler material wrong.
2. Remove individual main trials if the corresponding “background check” question was answered wrongly.

##### D.4.3.1. Cleaning by-participant

```
# look at error rates for filler sentences by subject
# mark every subject with < 0.5 proportion correct

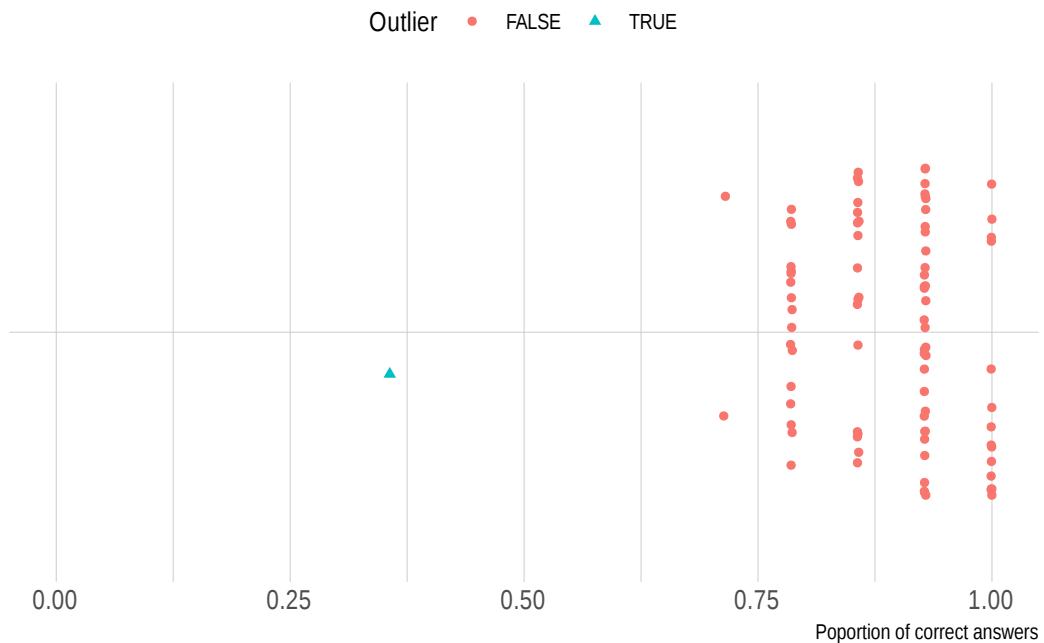
subject_error_rate <- data_KoF_processed %>%
  filter(trial_type == "filler") %>%
```

```
group_by(submission_id) %>%
summarise(
  proportion_correct = mean(correct_answer == response),
  outlier_subject = proportion_correct < 0.5
) %>%
arrange(proportion_correct)
```

Plot the results:

```
# plot by-subject error rates
subject_error_rate %>%
  ggplot(aes(x = proportion_correct, color = outlier_subject, shape = outlier_subject)) +
  geom_jitter(aes(y = ""), width = 0.001) +
  xlab("Poportion of correct answers") + ylab("") +
  ggtitle("Distribution of proportion of correct answers on filler trials") +
  xlim(0,1) +
  scale_color_discrete(name = "Outlier") +
  scale_shape_discrete(name = "Outlier")
```

## Distribution of proportion of correct answers on filler trials



Apply the cleaning step:

#### D. Data sets used in the book

```
# add info about error rates and exclude outlier subject(s)
d_cleaned <-
  full_join(data_KoF_processed, subject_error_rate, by = "submission_id") %>%
  filter(outlier_subject == FALSE)
```

##### D.4.3.2. Cleaning by-trial

```
# exclude every critical trial whose 'background' test question was answered wrongl

d_cleaned <-
  d_cleaned %>%
  # select only the 'background question' trials
  filter(trial_type == "special") %>%
  # is the background question answered correctly?
  mutate(
    background_correct = correct_answer == response
  ) %>%
  # select only the relevant columns
  select(submission_id, vignette, background_correct) %>%
  # right join lines to original data set
  right_join(d_cleaned, by = c("submission_id", "vignette")) %>%
  # remove all special trials, as well as main trials with incorrect background che
  filter(trial_type == "main" & background_correct == TRUE)

# write_csv(d_cleaned, "data_sets/king-of-france_data_cleaned.csv")
```

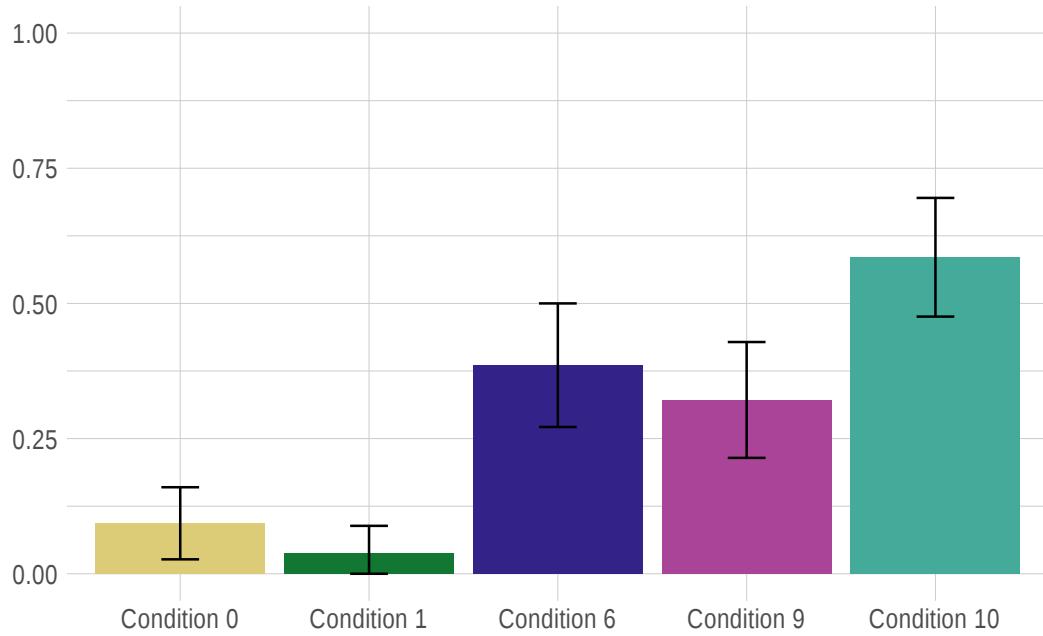
##### D.4.4. Exploration: summary stats & plots

Plot for ratings by condition:

```
d_cleaned %>%
  # drop unused factor levels
  droplevels() %>%
  # get means and 95% bootstrapped CIs for each condition
  group_by(condition) %>%
  nest() %>%
  summarise(
    CIs = map(data, function(d) bootstrapped_CI(d$response == "TRUE")))
  ) %>%
  unnest(CIs) %>%
```

```
# plot means and CIs
ggplot(aes(x = condition, y = mean, fill = condition)) +
  geom_bar(stat = "identity") +
  geom_errorbar(aes(ymin = lower, ymax = upper, width = 0.2)) +
  ylim(0,1) +
  ylab("") + xlab("") + ggtitle("Proportion of 'TRUE' responses per condition") +
  theme(legend.position = "none") +
  scale_fill_manual(values = project_colors)
```

## Proportion of 'TRUE' responses per condition



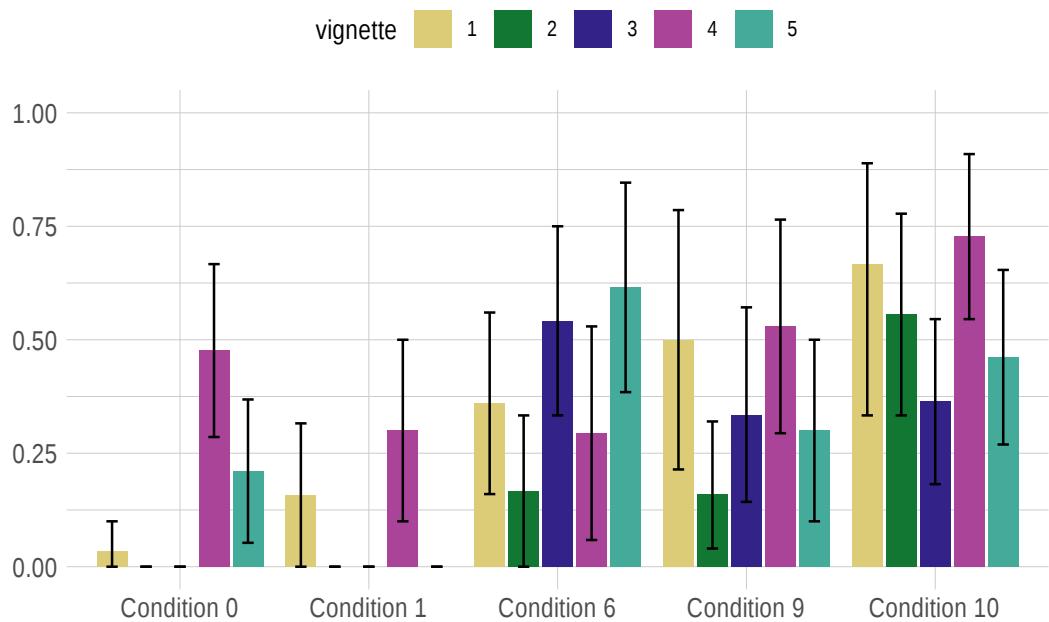
Plot for each condition & vignette:

```
data_KoF_processed %>%
  filter(trial_type == "main") %>%
  droplevels() %>%
  group_by(condition, vignette) %>%
  nest() %>%
  summarise(
    CI = map(data, function(d) bootstrapped_CI(d$response == "TRUE")))
  ) %>%
  unnest(CI) %>%
```

#### D. Data sets used in the book

```
ggplot(aes(x = condition, y = mean, fill = vignette)) +
  geom_bar(stat = "identity", position = "dodge2") +
  geom_errorbar(
    aes(ymin = lower, ymax = upper),
    width = 0.3,
    position = position_dodge(width = 0.9)
  ) +
  ylim(0,1) +
  ylab("") + xlab("") + ggtitle("Proportion of 'TRUE' responses per condition & vignette")
```

### Proportion of 'TRUE' responses per condition & vignette



#### D.4.5. Data analysis

### D.5. Bio-Logic Jazz-Metal (and where to consume it)

#### D.5.1. Nature, origin and rationale of the data

This is a very short and non-serious experiment that asks for just three binary decisions from each participant, namely their spontaneous preference for one of two presented options (biology vs logic, jazz vs metal and mountains vs beach). The data from this

## *D.5. Bio-Logic Jazz-Metal (and where to consume it)*

experiment will be analyzed and plotted. This is supposed to be a useful and hopefully entertaining self-generated data set with which to practice making contingency tables and to apply binomial tests and fun stuff like that.

### **D.5.1.1. The experiment**

#### **D.5.1.1.1. Participants**

We obtained data from 102 participants of this very course.

#### **D.5.1.1.2. Material**

There were three critical trials (and nothing else). All trials had the same trailing question:

If you have to choose between the following two options, which one do you prefer?

Each critical trial then presented two options as buttons, one of which had to be clicked.

1. Biology vs Logic
2. Jazz vs Metal
3. Mountains vs Beach

#### **D.5.1.1.3. Procedure**

Each participant saw all three critical trials (and no other trials) in random order.

#### **D.5.1.1.4. Realization**

The experiment was realized using `_magpie` and can be tried out here.

### **D.5.1.2. Theoretical motivation & hypotheses**

This is a bogus experiment, and no sane person would advance a serious hypothesis about this. Except that, actually, the lecturer is careless enough to conjecture that appreciators of Metal music like logic more than Jazz-enthusiasts would (because Metal is cleaner and more mechanic, while Jazz is fuzzy and organic, obviously).<sup>4</sup>

---

<sup>4</sup>Notice how easy it is to motivate any-old psychological theory. Some other scientific disciplines are much better at smothering nonsensical ideas from the start.

## D. Data sets used in the book

### D.5.2. Loading and preprocessing the data

First, load the data:

```
## Observations: 306
## Variables: 19
## $ submission_id <dbl> 379, 379, 379, 378, 378, 378, 377, 377, 377, 376, 376...
## $ QUD <lgl> NA, N...
## $ RT <dbl> 9230, 9330, 5248, 5570, 2896, 36236, 5906, 4767, 1042...
## $ age <dbl> 30, 30, 30, 29, 29, 29, 20, 20, 20, 21, 21, 21, 21, 23, 2...
## $ comments <chr> NA, N...
## $ education <chr> "Graduated High School", "Graduated High School", "Gr...
## $ endTime <dbl> 1.573751e+12, 1.573751e+12, 1.573751e+12, 1.573738e+1...
## $ experiment_id <dbl> 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, ...
## $ gender <chr> "male", "male", "male", "male", "male", "male", "fema...
## $ languages <chr> "German", "German", "German", "German", "German", "Ge...
## $ option1 <chr> "Mountains", "Biology", "Metal", "Metal", "Biology", ...
## $ option2 <chr> "Beach", "Logic", "Jazz", "Jazz", "Logic", "Beach", "...
## $ question <chr> "If you have to choose between the following two opti...
## $ response <chr> "Beach", "Logic", "Metal", "Metal", "Logic", "Beach", ...
## $ startDate <chr> "Thu Nov 14 2019 18:01:24 GMT+0100 (CET)", "Thu Nov 1...
## $ startTime <dbl> 1.573751e+12, 1.573751e+12, 1.573751e+12, 1.573738e+1...
## $ timeSpent <dbl> 2.3601500, 2.3601500, 2.3601500, 2.1552667, 2.1552667...
## $ trial_name <chr> "forced_choice", "forced_choice", "forced_choice", "f...
## $ trial_number <dbl> 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, ...

data_BLJM_raw <- read_csv(url('https://raw.githubusercontent.com/michael-franke/int...'))
```

Take a peak:

```
glimpse(data_BLJM_raw)
```

```
## Observations: 306
## Variables: 19
## $ submission_id <dbl> 379, 379, 379, 378, 378, 378, 377, 377, 377, 376, 376...
## $ QUD <lgl> NA, N...
## $ RT <dbl> 9230, 9330, 5248, 5570, 2896, 36236, 5906, 4767, 1042...
## $ age <dbl> 30, 30, 30, 29, 29, 29, 20, 20, 20, 21, 21, 21, 21, 23, 2...
## $ comments <chr> NA, N...
## $ education <chr> "Graduated High School", "Graduated High School", "Gr...
## $ endTime <dbl> 1.573751e+12, 1.573751e+12, 1.573751e+12, 1.573738e+1...
## $ experiment_id <dbl> 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, ...
```

## D.5. Bio-Logic Jazz-Metal (and where to consume it)

```
## $ gender <chr> "male", "male", "male", "male", "male", "fema...
## $ languages <chr> "German", "German", "German", "German", "Ge...
## $ option1 <chr> "Mountains", "Biology", "Metal", "Metal", "Biology", ...
## $ option2 <chr> "Beach", "Logic", "Jazz", "Jazz", "Logic", "Beach", ...
## $ question <chr> "If you have to choose between the following two opti...
## $ response <chr> "Beach", "Logic", "Metal", "Metal", "Logic", "Beach", ...
## $ startDate <chr> "Thu Nov 14 2019 18:01:24 GMT+0100 (CET)", "Thu Nov 1...
## $ startTime <dbl> 1.573751e+12, 1.573751e+12, 1.573751e+12, 1.573738e+1...
## $ timeSpent <dbl> 2.3601500, 2.3601500, 2.3601500, 2.1552667, 2.1552667...
## $ trial_name <chr> "forced_choice", "forced_choice", "forced_choice", "f...
## $ trial_number <dbl> 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3,...
```

The most important variables in this data set are:

- `submission_id`: unique identifier for each participant
- `option1` and `option2`: what the choice options were
- `response`: which of the two options was chosen

Notice that there is no convenient column indicating which of the three critical conditions we are dealing with, so we create that information from the information in columns `option1` and `option2`, while also discarding everything we will not need:

```
data_BLJM_processed <-
  data_BLJM_raw %>%
  mutate(
    condition = str_c(str_sub(option2, 1, 1), str_sub(option1, 1, 1))
  ) %>%
  select(submission_id, condition, response)
# write_csv(data_BLJM_processed, "../data_sets/bio-logic-jazz-metal-data-processed.csv")
data_BLJM_processed

## # A tibble: 306 x 3
##   submission_id condition response
##       <dbl> <chr>     <chr>
## 1         379 BM        Beach
## 2         379 LB        Logic
## 3         379 JM        Metal
## 4         378 JM        Metal
## 5         378 LB        Logic
## 6         378 BM        Beach
## 7         377 BM        Mountains
## 8         377 LB        Biology
## 9         377 JM        Jazz
## 10        376 BM        Beach
## # ... with 296 more rows
```

## D. Data sets used in the book

### D.5.3. Exploration: counts & plots

We are interest in some counts. First, let's look at the overal choice rates in each condition:

```
data_BLJM_processed %>%
  # function`count` is masked by another package, must call explicitly
  dplyr::count(condition, response)

## # A tibble: 6 x 3
##   condition response     n
##   <chr>      <chr>    <int>
## 1 BM         Beach     44
## 2 BM         Mountains 58
## 3 JM         Jazz      64
## 4 JM         Metal     38
## 5 LB         Biology   58
## 6 LB         Logic     44
```

Overall it seems that mountains are preferred over beaches, Jazz is preferred over Metal and Biology is preferred over Logic.

The overall counts, however, do not tell us anything about any potentially interesting relationship between preferences. So, let's have a closer look at the lecturer's conjecture that a preference for logic tends to go with a stronger preference for metal than a preference for biology does. To check this, we need to look at different counts, namely the number of people who selected which music-subject pair. We collect these counts in variable BLJM\_associated\_counts:

```
BLJM_associated_counts <- data_BLJM_processed %>%
  select(submission_id, condition, response) %>%
  pivot_wider(names_from = condition, values_from = response) %>%
  select(-BM) %>%
  dplyr::count(JM,LB)
BLJM_associated_counts

## # A tibble: 4 x 3
##   JM     LB     n
##   <chr> <chr> <int>
## 1 Jazz   Biology 38
## 2 Jazz   Logic   26
## 3 Metal  Biology 20
## 4 Metal  Logic   18
```

### D.5. Bio-Logic Jazz-Metal (and where to consume it)

Notice that this representation is tidy, but not as ideal for visual inspection. A more commonly seen format can be obtained by pivoting to a wider representation:

```
# visually attractive table representation
BLJM_associated_counts %>%
  pivot_wider(names_from = LB, values_from = n)

## # A tibble: 2 x 3
##   JM     Biology Logic
##   <chr>    <int> <int>
## 1 Jazz      38     26
## 2 Metal     20     18
```

The tidy representation *is* ideal for plotting, though. Notice, however, that the code below plots proportions of choices, not raw counts:

```
BLJM_associated_counts %>%
  ggplot(aes(x = LB, y = n / sum(n), color = JM, shape = JM, group = JM)) +
  geom_point(size = 3) + geom_line() +
  labs(
    title = "Proportion of choices of each music+subject pair",
    x = "",
    y = "")
```

#### D. Data sets used in the book

### Proportion of choices of each music+subject pair



The lecturer's conjecture might be correct. This does look like there could be an interaction. While Jazz is preferred more generally, the preference for Jazz over Metal seems more pronounced for those participants who preferred Biology than for those who preferred Logic.

## D.6. Avocado prices

### D.6.1. Nature, origin and rationale of the data

This data set has been plucked from kaggle. More information on the origin and composition of this data set can be found on kaggle's website covering the avocado data. The data set includes information about the prices of (Hass) avocados and the amount sold (of different kinds) at different points in time. The data is originally from the Hass Avocado Board where the data is described as follows:

The [data] represents weekly 2018 retail scan data for National retail volume (units) and price. Retail scan data comes directly from retailers' cash registers based on actual retail sales of Hass avocados. Starting in 2013, the table below reflects an expanded, multi-outlet retail data set. Multi-outlet reporting includes an aggregation of the following channels: grocery, mass,

club, drug, dollar and military. The Average Price (of avocados) in the table reflects a per unit (per avocado) cost, even when multiple units (avocados) are sold in bags. The Product Lookup codes (PLU's) in the table are only for Hass avocados. Other varieties of avocados (e.g. greenskins) are not included in this table.

Columns of interest are:

- Date - date of the observation
- AveragePrice - average price of a single avocado
- Total Volume - Total number of avocados sold
- type - whether the price/amount is for conventional or organic
- 4046 - Total number of small avocados sold (PLU 4046)
- 4225 - Total number of medium avocados sold (PLU 4225)
- 4770 - Total number of large avocados sold (PLU 4770)

### D.6.2. Loading and preprocessing the data

We load the data into a variable named `avocado_data` but also immediately rename some of the columns to have more convenient handles:

```
avocado_data <- read_csv(url('https://raw.githubusercontent.com/michael-franke/intro-data-anal...'))
# remove currently irrelevant columns
select( -X1 , - contains("Bags") , - year , - region ) %>%
# rename variables of interest for convenience
rename(
  total_volume_sold = `Total Volume` ,
  average_price = `AveragePrice` ,
  small = '4046' ,
  medium = '4225' ,
  large = '4770' ,
)
```

We can then take a glimpse:

```
glimpse(avocado_data)

## Observations: 18,249
## Variables: 7
## $ Date           <date> 2015-12-27, 2015-12-20, 2015-12-13, 2015-12-06, ...
## $ average_price   <dbl> 1.33, 1.35, 0.93, 1.08, 1.28, 1.26, 0.99, 0.98, 1...
## $ total_volume_sold <dbl> 64236.62, 54876.98, 118220.22, 78992.15, 51039.60...
## $ small          <dbl> 1036.74, 674.28, 794.70, 1132.00, 941.48, 1184.27...
```

#### D. Data sets used in the book

```
## $ medium          <dbl> 54454.85, 44638.81, 109149.67, 71976.41, 43838.39...
## $ large           <dbl> 48.16, 58.33, 130.50, 72.58, 75.78, 43.61, 93.26, ...
## $ type            <chr> "conventional", "conventional", "conventional", "...
```

#### D.6.3. Summary statistics

We are interested in the following summary statistics for the variables `total_amount_sold` and `average_price` for the whole data and for each type of avocado separately:

- mean
- median
- variance
- the bootstrapped 95% confidence interval of the mean

To get these results we define a convenience function that calculates exactly these measures:

```
summary_stats_convenience_fct <- function(numeric_data_vector) {
  bootstrap_results <- bootstrapped_CI(numeric_data_vector)
  tibble(
    CI_lower = bootstrap_results$lower,
    mean = bootstrap_results$mean,
    CI_upper = bootstrap_results$upper,
    median = median(numeric_data_vector),
    var = var(numeric_data_vector)
  )
}
```

We then apply this function once for the whole data set and once for each type of avocado (conventional or organic). We do this using a nested tibble in order to record the joint output of the convenience function (so that we only need to calculate the bootstrapped 95% confidence interval twice).

```
# summary stats for the whole data taken together
avocado_sum_stats_total <- avocado_data %>%
  select(type, average_price, total_volume_sold) %>%
  pivot_longer(
    cols = c(total_volume_sold, average_price),
    names_to = 'variable',
    values_to = 'value'
  ) %>%
  group_by(variable) %>%
  nest() %>%
```

```

summarise(
  summary_stats = map(data, function(d) summary_stats_convenience_fct(d$value))
) %>%
unnest(summary_stats) %>%
mutate(type = "both_together") %>%
# reorder columns: moving `type` to second position
select(1,type,everything())

# summary stats for each type of avocado
avocado_sum_stats_by_type <- avocado_data %>%
  select(type, average_price, total_volume_sold) %>%
  pivot_longer(
    cols = c(total_volume_sold, average_price),
    names_to = 'variable',
    values_to = 'value'
  ) %>%
  group_by(type, variable) %>%
  nest() %>%
  summarise(
    summary_stats = map(data, function(d) summary_stats_convenience_fct(d$value))
  ) %>%
  unnest(summary_stats)

# joining the summary stats in a single tibble
avocado_sum_stats <-
  full_join(avocado_sum_stats_total, avocado_sum_stats_by_type)

# inspect the results
avocado_sum_stats

## # A tibble: 6 x 7
##   variable     type      CI_lower     mean    CI_upper    median     var
##   <chr>       <chr>      <dbl>     <dbl>      <dbl>     <dbl>     <dbl>
## 1 average_price both_together 1.40 1.41e0 1.41e0 1.37e0 1.62e- 1
## 2 total_volume_sold both_together 800444. 8.51e5 9.01e5 1.07e5 1.19e+13
## 3 average_price conventional 1.15 1.16e0 1.16e0 1.13e0 6.92e- 2
## 4 total_volume_sold conventional 1559971. 1.65e6 1.76e6 4.08e5 2.25e+13
## 5 average_price organic 1.65 1.65e0 1.66e0 1.63e0 1.32e- 1
## 6 total_volume_sold organic 45101. 4.78e4 5.07e4 1.08e4 2.03e+10

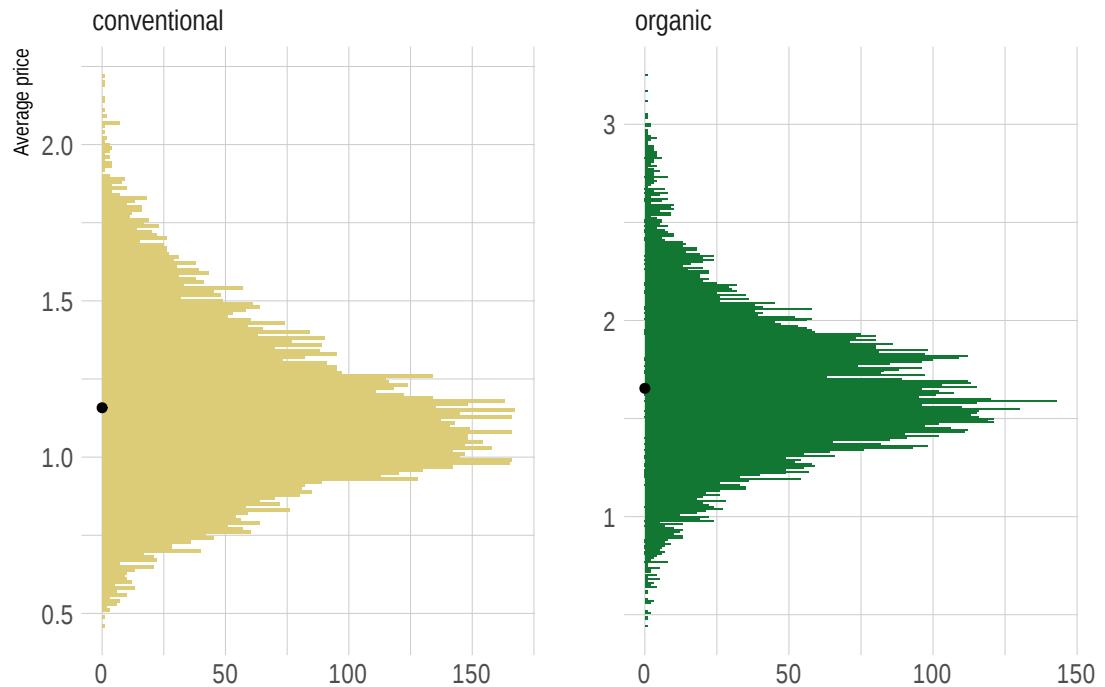
```

#### D.6.4. Plots

Here are plots of the distributions of `average_price` for different types of avocados:

#### D. Data sets used in the book

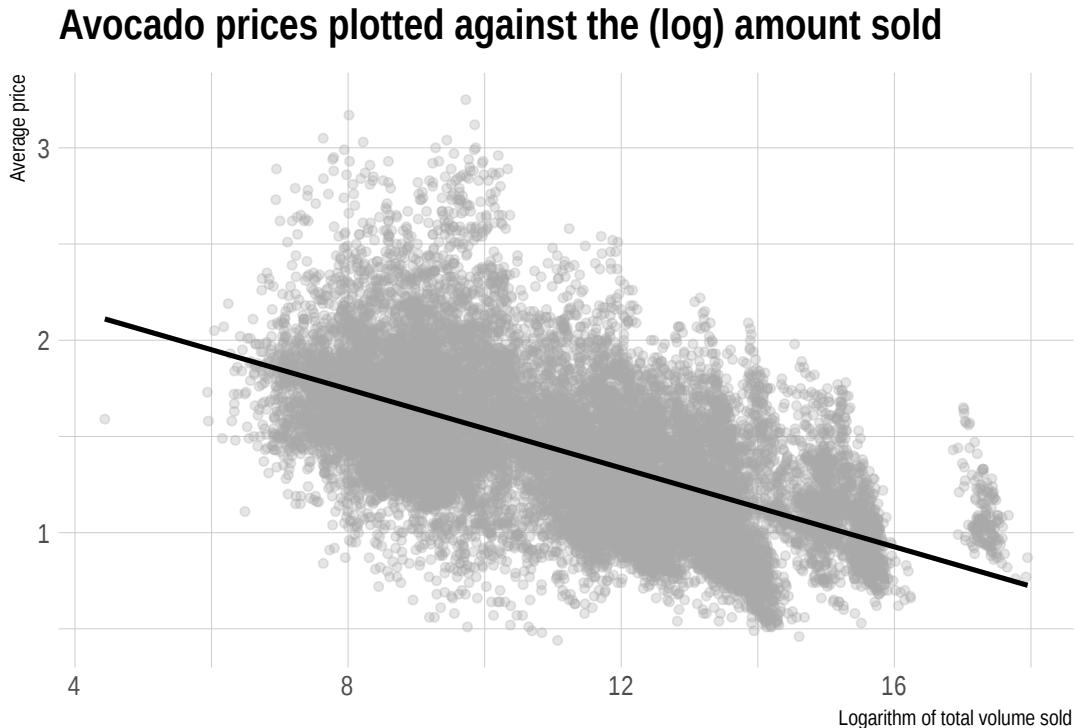
```
avocado_data %>%
  ggplot(aes(x = average_price, fill = type)) +
  geom_histogram(binwidth = 0.01) +
  facet_wrap(~type, scales = "free") +
  coord_flip() +
  geom_point(
    data = avocado_sum_stats_by_type %>% filter(variable == "average_price"),
    aes(y = 0, x = mean)
  ) +
  ylab('') +
  xlab('Average price') +
  theme(legend.position = "none")
```



Here is a scatter plot of the logarithm of `total_volume_sold` against `average_price`:

```
avocado_data %>%
  ggplot(aes(x = log(total_volume_sold), y = average_price)) +
  geom_point(color = "darkgray", alpha = 0.3) +
  geom_smooth(color = "black", method = "lm") +
  xlab('Logarithm of total volume sold') +
```

```
ylab('Average price') +
ggtitle("Avocado prices plotted against the (log) amount sold")
```



And another scatter plot, using a log-scaled  $x$ -axis and distinguishing different types of avocados are:

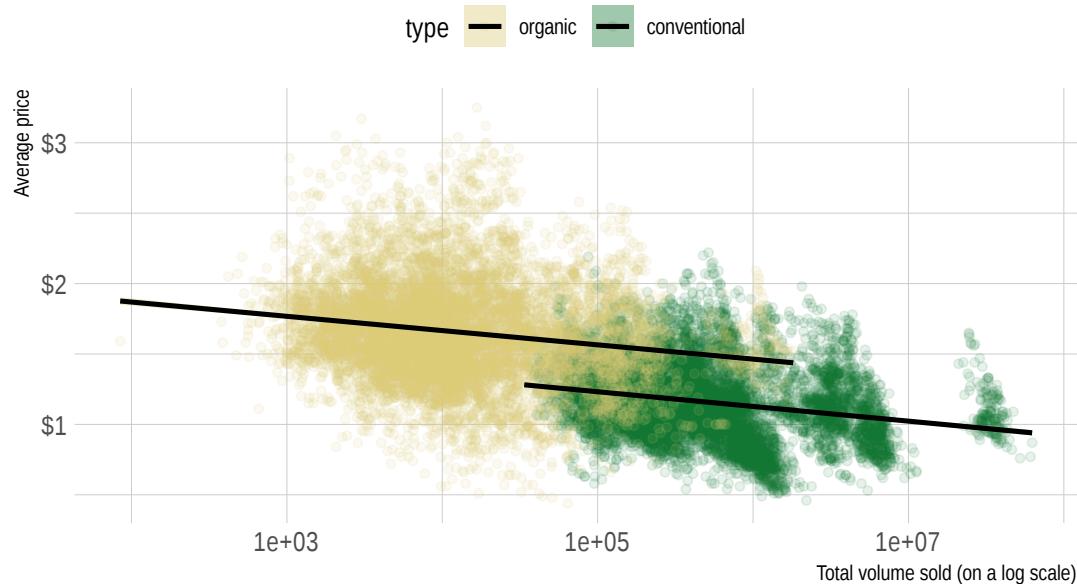
```
# pipe data set into function `ggplot`
avocado_data %>%
  # reverse factor level so that horizontal legend entries align with
  # the majority of observations of each group in the plot
  mutate(
    type = fct_rev(type)
  ) %>%
  # initialize the plot
  ggplot(
    # defined mapping
    mapping = aes(
      # which variable goes on the x-axis
      x = total_volume_sold,
      # which variable goes on the y-axis
```

#### D. Data sets used in the book

```
y = average_price,
# which groups of variables to distinguish
group = type,
# color and fill to change by grouping variable
fill = type,
color = type
)
) +
# declare that we want a scatter plot
geom_point(
  # set low opacity for each point
  alpha = 0.1
) +
# add a linear model fit (for each group)
geom_smooth(
  color = "black",
  method = "lm"
) +
# change the default (normal) of x-axis to log-scale
scale_x_log10() +
# add dollar signs to y-axis labels
scale_y_continuous(labels = scales::dollar) +
# change axis labels and plot title & subtitle
labs(
  x = 'Total volume sold (on a log scale)',
  y = 'Average price',
  title = "Avocado prices plotted against the amount sold per type",
  subtitle = "With linear regression lines"
)
```

## Avocado prices plotted against the amount sold per type

With linear regression lines



Abrusán, Márta, and Kriszta Szendrői. 2013. “Experimenting with the King of France: Topics, Verifiability and Definite Descriptions.” *Semantics & Pragmatics* 6 (1): 1–43.

Academy, Khan. 2019. “Lagrange multipliers, introduction.” 2019. <https://www.khanacademy.org/math/multivariable-calculus/applications-of-multivariable-derivatives/constrained-optimization/a/lagrange-multipliers-single-constraint>.

Anscombe, F. J. 1973. “Graphs in Statistical Analysis.” *The American Statistician* 27 (1). [American Statistical Association, Taylor & Francis, Ltd.]: 17–21. <https://doi.org/10.2307/2682899>.

Blitzstein, Joseph K., and Jessica Hwang. 2014. *Introduction to Probability*. Chapman; Hall/CRC.

Bürkner, Paul-Christian. 2017. “brms: An R Package for Bayesian Multilevel Models Using Stan.” *Journal of Statistical Software* 80 (1): 1–28. <https://doi.org/10.18637/jss.v080.i01>.

Carpenter, Bob, Andrew Gelman, Matthew D. Hoffman, Daniel Lee, Ben Goodrich, Michael Betancourt, Marcus Brubaker, Jiqiang Guo, Peter Li, and Allen Riddell. 2017. “Stan: A Probabilistic Programming Language.” *Journal of Statistical Software* 76 (1). <https://doi.org/10.18637/jss.v076.i01>.

De Martino, Andrea, and Daniele De Martino. 2018. “An Introduction to the Maximum

#### D. Data sets used in the book

- Entropy Approach and its Application to Inference Problems in Biology.” *Heliyon* 4 (4). Elsevier.
- Dobson, Annette J., and Adrian G. Barnett. 2008. *An Introduction to Generalized Linear Models*. Chapman; Hall/CRC.
- Finlayson, Sam. 2017. “Deriving probability distributions using the Principal of Maximum Entropy.” 2017. <https://sgfin.github.io/2017/03/16/Deriving-probability-distributions-using-the-Principle-of-Maximum-Entropy/#introduction>.
- Gelman, Andrew, John B. Carlin, Hal S. Stern, David B. Dunson, Aki Vehtari, and Donald B. Rubin. 2013. *Bayesian Data Analysis*. Chapman; Hall/CRC.
- Golding, Nick. 2019. *greta: Simple and Scalable Statistical Modelling in R*. <https://CRAN.R-project.org/package=greta>.
- Goodman, Noah D, and Andreas Stuhlmüller. 2014. “The Design and Implementation of Probabilistic Programming Languages.” <http://dippl.org>.
- Halpern, Joseph Y. 2003. *Reasoning about Uncertainty*. MIT Press.
- Healy, Kieran. 2018. *Data Visualization*. New Jersey, USA: Princeton University Press.
- Jaynes, Edwin T. 2003. *Probability Theory: The Logic of Science*. Cambridge university press.
- Keng, Brian. 2017. “Maximum Entropy Distributions.” 2017. <http://bjlkeng.github.io/posts/maxim> entropy-distributions/.
- Kline, Rex B. 2013. *Beyond Significance Testing: Statistics Reform in the Behavioral Sciences*. American Psychological Association.
- Klingenberg, Bernhard. n.d. “Art of Stat.” <http://www.artofstat.com/home.html>.
- Kruschke, John. 2015. *Doing Bayesian data analysis: A tutorial with R, JAGS, and Stan*. Academic Press.
- Lee, Michael D., and Eric-Jan Wagenmakers. 2014. *Bayesian cognitive modeling: A practical course*. Cambridge university press.
- McElreath, Richard. 2015. *Statistical Rethinking: A Bayesian Course with Examples in R and Stan*. Chapman; Hall/CRC.
- R Core Team. 2018. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. <https://www.R-project.org/>.
- Reza, Fazlollah M. 1994. *An Introduction to Information Theory*. Courier Corporation.
- Tufte, Edward. 1983. *The Visual Display of Quantitative Information*. Graphics Press.
- Vallverdú, Jordi. 2016. *Bayesians Versus Frequentists: A Philosophical Debate on Statistical Reasoning*. Heidelberg: Springer.
- Wickham, Hadley. 2010. “A Layered Grammar of Graphics.” *Journal of Computational and Graphical Statistics* 19 (1): 3–28.

- . 2014. “Tidy Data.” *Journal of Statistical Software* 59 (10).
- . 2017. *tidyverse: Easily Install and Load the 'Tidyverse'*. <https://CRAN.R-project.org/package=tidyverse>.
- Wickham, Hadley, and Garrett Grolemund. 2016. *R for Data Science: Import, Tidy, Transform, Visualize, and Model Data*. O'Reilly Media, Inc.
- Winter, Bodo. 2019. *Statistics for Linguists: An Introduction Using R*. Routledge.