Introduction to Data Analysis

Michael Franke

last rendered at: 2019-11-07 12:04:03

Contents

Preface

This book is the basic reading material for the course "Introduction to Data Analysis", held at the Universtity 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, sill 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 fours 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 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 focues 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 covers

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.

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 an 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 conenience 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.

1.5.2. 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 enviroment 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:

```
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**.[1] While other programming languages specialized for scientific computing, like Python of 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 interchangable, 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 analyes 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 A.1.

The learning goals for this chapter are:

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

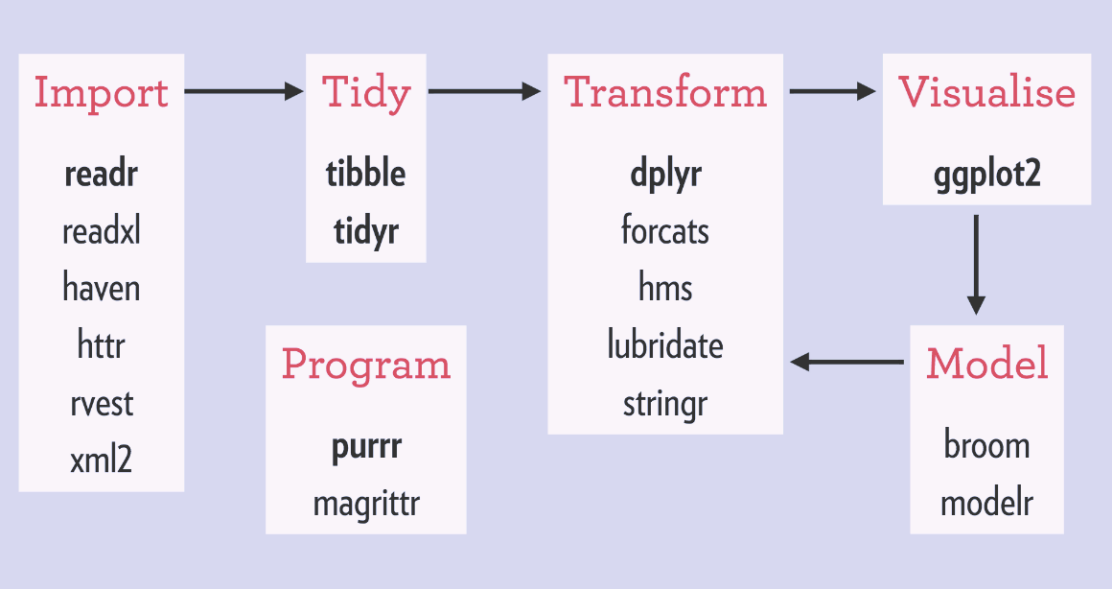


Figure 2.1.: Overview of selected packages from the tidyverse.

- 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**.[2]

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

```
6 * 7
```

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

x### Functions

R has many built-in functions. The most common situation is that the function is

[2] 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.

[3] When starting a session in a terminal, you can exit a running R session by typing `quit()` or `q()`.

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 2.1.5) 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.

```
# 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.

2.1.1. 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.[4]

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.

2.1.2. Literate coding

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

```
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

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

your scripts. Any comment followed by `----` is treated as a (foldable) section.

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

2.1.3. 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)
```

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

2.1.4. 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 curteous 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 informtion on "files", "plots" etc. to show all installed packages. This also shows which packages are currently loaded.

2.1.5. Getting help

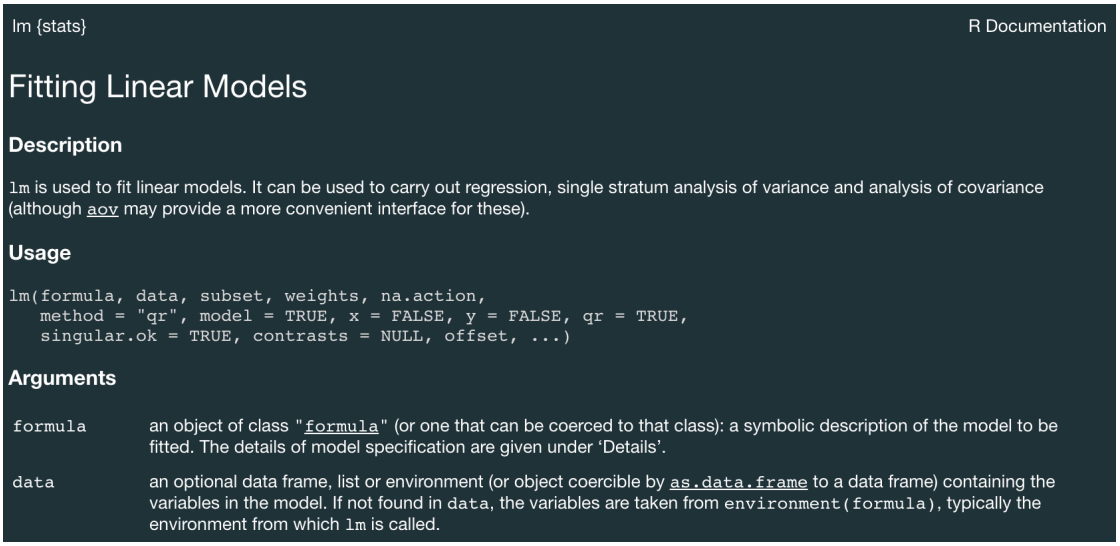
If you encounter a function like `lm` that you do not know about, you can access its documentation with the `help` function or just typing `?lm`. Ror example, the following call summons the documentation for `lm`, the first parts are shown in Figure 2.2.

```
help(lm)
```

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

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
```


lm {stats} R Documentation

Fitting Linear Models

Description

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).

Usage

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

Arguments

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'.
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.

```
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:

```
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:

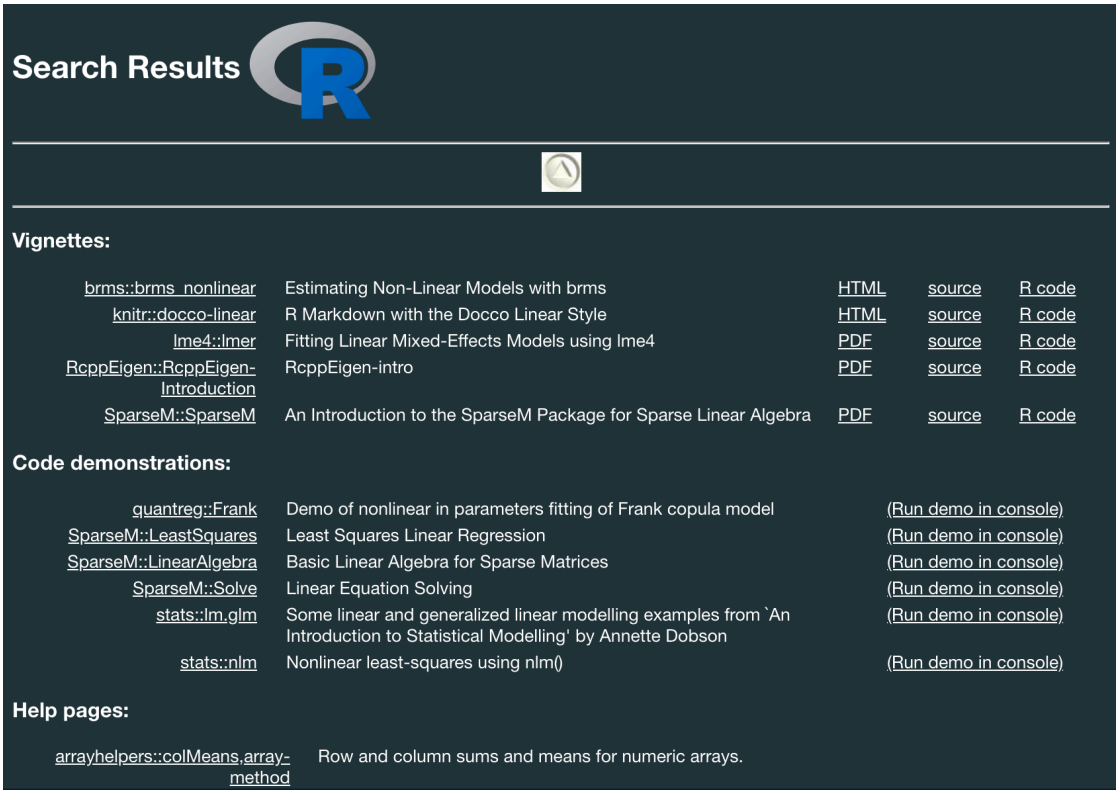


Figure 2.3.: Result of calling ‘help.search‘ for the term ’linear’.

```
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)
```

```
## [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).

2.2.1. 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.[5] 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?
```

```
## [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
```

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

```
## [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.

```
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*.[6] This means that it will fill columns first. For example, a matrix with three columns based on a six-placed vector $1, 2, \ldots, 6$ will be built by filling the first column from top to bottom, then the second column top to bottom, and so on.[7]

```
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
```

[6] Python, on the other hand, uses the reverse *row-major mode*.

[7] 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 nanmes 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:[8]

```
# 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`.

```
# 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
```

[8] Notice that we can create strings (actually called 'characters' in R) with double quotes

```
## 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 availables"; 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 `stingr` 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 `str_which`:

```
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
```

```
## [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 arbitraty 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"
```

```
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]
```

```
## [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 on element is `T`

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%`: wheter 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
# returnt 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-stlye 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))
}

# 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

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.[9] The pipe `%>%` essentially takes 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)`.

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

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.[1] 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 and learn about what makes data tidy in Section 3.4.

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"

[1] 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.

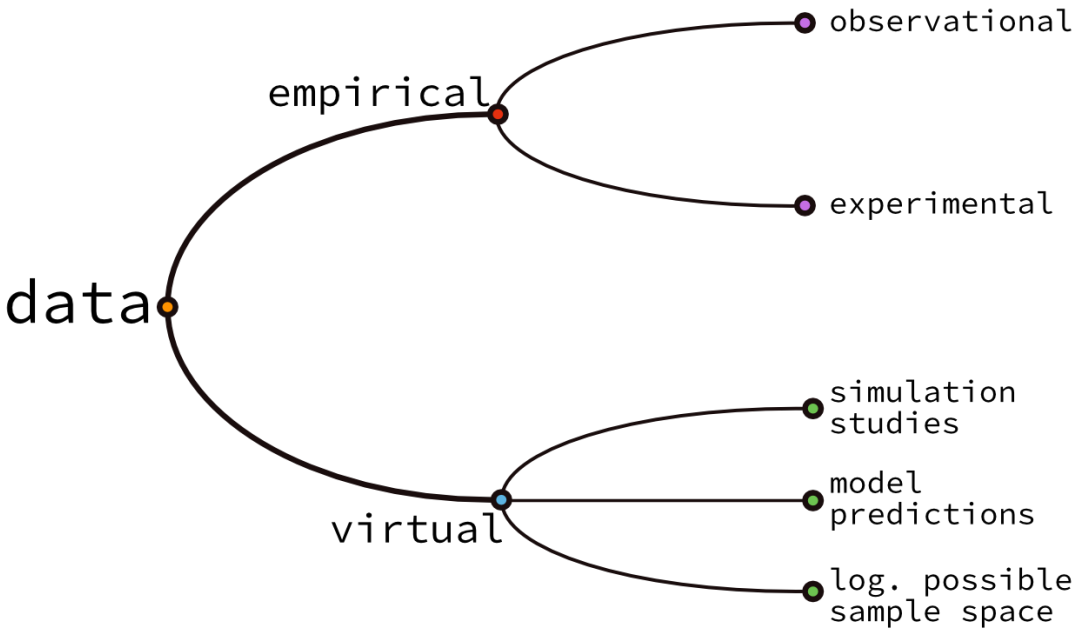


Figure 3.1.: Hierarchy of different kinds of data relevant for 'data science'.

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". This is important, "data" is defined by what we do with it. Another important aspect of this definition is fairly backgrounded, but just as important. 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 (see below the notion of a "variable", discussed in Section 3.2). In sum, we can think of data as a represenation 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.

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

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

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

we can just as well look at **virtual data**. For example, *virtual data* could be **data 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).[2] Finally, **logically possible sample data** is data 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 anyone, 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 observation 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 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 about future data that are as accurate as possible.

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

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 IQ scores or a Boolean vector of whether five students passed the exam; we then still (usually) know what these data points represent:

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

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

Most often, however, we have more than one kind of observation that we care about. Most often, we care about systematic relationships and so we care about co-occurrences of observations of different kind. For instance, we might want to know look at pass/fail results from exam in co-occurrence with information about who attended tutorials:

```
# 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>
## 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 3.4.) In the example above, every column represents a **variable** of interest. A *(data) variable* stores the observations

that are of the same kind.[3]

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)

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 **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

- open/closed questions
- factorial designs
- within- vs between-subjects design
- repeated measures
- randomization, fillers and controls
- sample size

3.4. Tidy data

- stages: preprocesssing & cleaning

[3] 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 @(Chap-03-01-probability-random-variables). The term "data variable" is not commonly used; it is merely "variable".

- wrangling
- long, wide & tidy formats

4. Data Wrangling

- data wrangling (practical R)
  - tidy data (read, write, …)
  - `pivot_longer`, `pivot_wider`, `join` …
    - check `vignette("pivot")`
  - nested tibbles
  - `group_by`, `summarize`, …

5. Summary statistics

- simple summary statistics (theory)
  - mean/mode/median/quantiles/bootstraped CI of mean
  - variance, standard deviation
  - ...
- exploring dependencies in data
  - plotting schemes
  - correlation
    - Bravais-Pearson
    - Spearman/Kendall
    - ...
  - multiple correlations
    - show `pairs` plots
    - find a good data set (also for later in simple linear regressions)

5.1. Exploring numerical data in a vector {#02-02-exploring}

The first part of the Mental Chronometry experiment is a simple reaction time task.

```
# read a data set of reaction times (as a tibble)
# RTs <- read_csv("https://tinyurl.com/y4jo8mox")
RT <- read_csv("data_sets/ch-03-set-01-MC-RTs.csv") %>% pull(RT)

# check its content
glimpse(RT)
```

```
##  num [1:3980] 5.61 5.32 5.36 5.67 5.37 ...
```

5.1.1. Bootstrapped 95% confidence intervals

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 **??**. 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 hypthetical 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^{rep}[1]
2. calculate the mean $\mu(D^{\text{rep}})$ of the newly sampled data
3. repeat steps 1 and 2 to gather r means of different resamples of D; call the result vector μ_{sampled}
4. the boundaries of the 95% inner quantile of μ_{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)
```

[1] D^{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^{obs} for clarity in comparison to D^{rep}.

```
  )
}
```

Applying this method to the vector of reaction times from above we get:

```
bootstrapped_CI(RT)
```

```
## # A tibble: 1 x 3
##   lower  mean upper
##   <dbl> <dbl> <dbl>
## 1  5.72  5.73  5.74
```

Notice that, since `RT` has length 3980, i.e., we have $k = 3980$ observations in the data, the bootstrapped 95% confidence interval is rather narrow. Compare this against a case of $k = 30$:

```
# 30 samples from a standard normal
smaller_data = rnorm(n = 30, mean = 0, sd = 1)
bootstrapped_CI(smaller_data)
```

```
## # A tibble: 1 x 3
##    lower    mean upper
##    <dbl>   <dbl> <dbl>
## 1 -0.364 -0.0487 0.285
```

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 on the Mental Chronometry data set (see Appendix D.1:

```
mc_data_cleaned = read_csv("data_sets/mental-chrono-data_cleaned.csv")
mc_data_cleaned %>%
  group_by(block) %>%
  nest() %>%
  summarise(
    CIs = map(data, function(d) bootstrapped_CI(d$RT))
  ) %>%
  unnest(CIs)
```

```
## # A tibble: 3 x 4
##   block          lower  mean upper
##   <chr>          <dbl> <dbl> <dbl>
## 1 discrimination  480.  488.  495.
## 2 goNoGo          420.  427.  435.
## 3 reaction        296.  300.  304.
```

6. Data Plotting

6.1. Motivating example: Anscombe's quartet

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.

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
## $ 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, 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...
## $ y2 <dbl> 9.14, 8.14, 8.74, 8.77, 9.26, 8.10, 6.13, 3.10, 9.13, 7.26,...
## $ y3 <dbl> 7.46, 6.77, 12.74, 7.11, 7.81, 8.84, 6.08, 5.39, 8.15, 6.42...
## $ y4 <dbl> 6.58, 5.76, 7.71, 8.84, 8.47, 7.04, 5.25, 12.50, 5.56, 7.91...
```

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, after the last section on data wrangling, we can tidy up quickly:

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

```
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),
    r_squared = cor(x,y)^2
  )
```

```
## # A tibble: 4 x 8
##   grp     mean_x mean_y min_x min_y max_x max_y r_squared
##   <chr>    <dbl>  <dbl> <dbl> <dbl> <dbl> <dbl>     <dbl>
## 1 Group 1      9   7.50     4  4.26    14 10.8       0.667
## 2 Group 2      9   7.50     4  3.1     14  9.26      0.666
## 3 Group 3      9   7.5      4  5.39    14 12.7       0.666
## 4 Group 4      9   7.50     8  5.25    19 12.5       0.667
```

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:

```
tidy_anscombe %>%
  ggplot(aes(x, y)) +
    geom_smooth(method = lm, se = F, color = "black") +
    geom_point(color = project_colors[3], size = 2) +
    scale_y_continuous(breaks = scales::pretty_breaks()) +
```

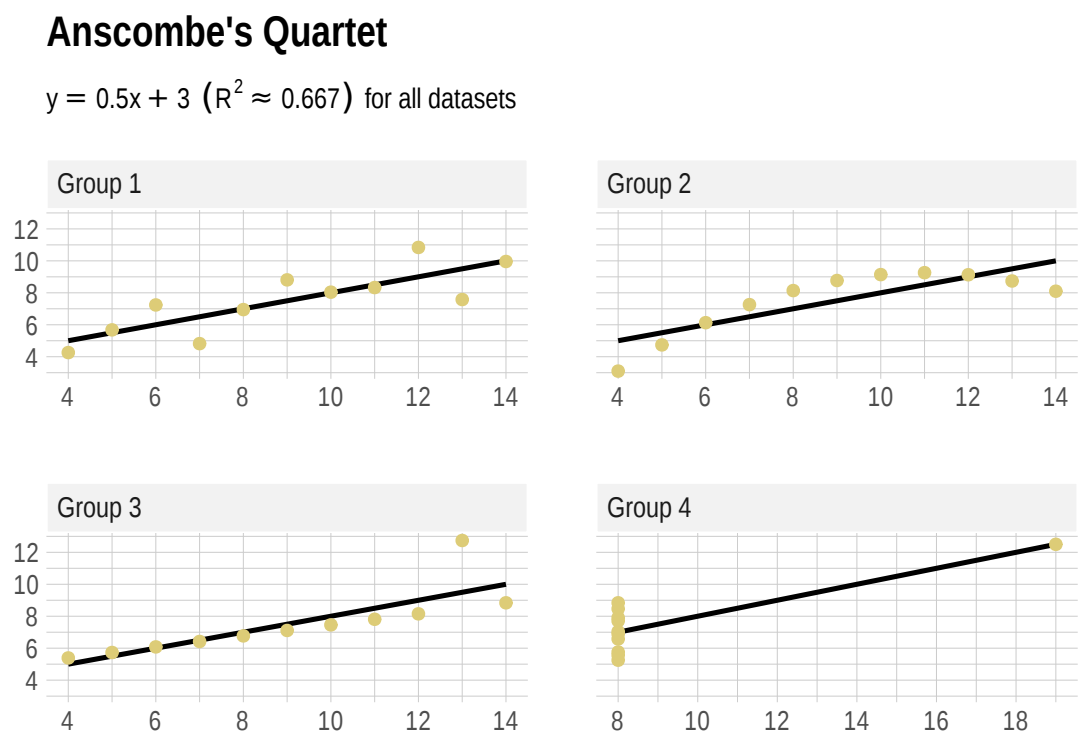


Figure 6.1.: Anscombe's Quartet: four different data sets all of which receive the same correlation score.

```
  scale_x_continuous(breaks = scales::pretty_breaks()) +
  labs(title = "Anscombe's Quartet", x = NULL, y = NULL,
       subtitle = bquote(y == 0.5 * x + 3 ~ (R^2 %~~% .667) ~ "for all datasets")) +
  facet_wrap(~grp, ncol = 2, scales = "free_x") +
  theme(strip.background = element_rect(fill = "#f2f2f2", colour = "white"))
```

6.2. Plotting for 'Mental Chronometry'

```
# read a data set of reaction times (as a tibble)
# RTs <- read_csv("https://tinyurl.com/y4jo8mox")
RTs <- read_csv("data_sets/ch-03-set-01-MC-RTs.csv")

# check its content
glimpse(RTs)

## Observations: 3,980
```

```
## Variables: 1
## $ RT <dbl> 5.605802, 5.318120, 5.361292, 5.669881, 5.370638, 5.327876,...
```

Part III.

Models and inferences

7. Basics of Probability Theory

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 Ω of all **elementary outcome** $\omega_1, \omega_2, \ldots$ of a process or event whose execution is (partially) random or unknown. Elementary outcomes are mutually exclusive. The set Ω exhausts all possibilities.[1]

Example. The set of elementary outcomes of a single coin flip is $\Omega_{\text{coin flip}} = \{\text{heads}, \text{tails}\}$. The elementary outcomes of tossing a six-sided die is $\Omega_{\text{standard die}} = \{\ ,\ ,\ ,\ ,\ ,\ \}$.[2]

An **event** A is a subset of Ω. 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 $A = \{\ ,\ ,\ \}$, i.e., observing an 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

[1] For simplicity of exposure, we gloss over subtleties arising when dealing with infinite sets Ω. 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.

[2] Think of Ω 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. Ω is our, the modellers', first crude simplification of nature, abstracting away aspects we currently do not care about.

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 Ω is a function $P : \mathfrak{P}(\Omega) \to \mathbb{R}$ 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[3]

Occasionally we encounter notation $P \in \Delta(\Omega)$ to express that P is a probability distribution over Ω. (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 Ω 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.

[3] A3 is the axiom of *countable additivity*. Finite additivity may be enough for finite or countable sets Ω, 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: $\Omega_{\text{our urn}} = \{\text{white}, \text{black}\}$. 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[4]) $P(\text{black}) = \frac{k}{N}$.

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:

$$\Omega_{\text{flip-and-draw}} = \{\langle \text{heads}, \text{black} \rangle, \langle \text{heads}, \text{white} \rangle, \langle \text{tails}, \text{black} \rangle, \langle \text{tails}, \text{white} \rangle\} \ .$$

The probability of event $\langle \text{heads}, \text{black} \rangle$ 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 $P(\langle \text{heads}, \text{black} \rangle) = 0.5 \times 0.2 = 0.1$. 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.)

[4] If probabilities are subjective beliefs, a rational agent is, in a sense, normatively required to assign exactly this probability.

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 $\Omega_{flip-\&-draw} = \Omega_{flip} \times \Omega_{draw}$,[5] we can use notation $P(\text{heads}, \text{black})$ as shorthand for $P(\langle \text{heads}, \text{black} \rangle)$. More generally, if $\Omega = \Omega_1 \times \ldots \Omega_n$, 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 $\Omega = \Omega_1 \times \ldots \Omega_n$, the **marginal distribution** over subspace Ω_i, $1 \leq i \leq n$ is the probability distribution that assigns to all $A_i \subseteq \Omega_i$ the probability:[6]

$$P(A_i) = \sum_{A_1 \subseteq \Omega_1, \ldots, A_{i-1} \subseteq \Omega_{i-1}, A_{i+1} \subseteq \Omega_{i+1}, \ldots, A_n \subseteq \Omega_n} P(A_1, \ldots, A_{i-1}, A_i, A_{i+1}, \ldots A_n)$$

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 $P(\text{black}) = 0.7$.[7]

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 $P(A \mid B)$, gives the probability of A on the assumption that B is true.[8] It is defined like so:

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)}$$

Conditional probabilities are only defined when $P(B) > 0$.[9]

[5] With $\Omega_{\text{flip}} = \{\text{heads}, \text{tails}\}$ and $\Omega_{\text{draw}} = \{\text{black}, \text{white}\}$.

[6] This notation, using $\sum$, assumes that subspaces are countable. In other cases, a parallel definition with integrals can be used.

[7] The term "marginal distribution" derives from such probability tables, where traditionally the sum of each row/column was written in the margins.

[8] We also verbalize this as "the conditional probability of A conditioned on B."

[9] 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 $B = \{\ ,\ ,\ \}$ that the tossed number is odd has probability $P(B) = \frac{1}{2}$. The probability of event $A = \{\ ,\ ,\ ,\ \}$ 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 $P(A \cap B) = P(\{\ ,\ \}) = \frac{1}{3}$. The conditional probability of tossing a number that is bigger than two, when we know that the toss is even, is $P(A \mid B) = \frac{1/3}{1/2} = \frac{2}{3}$.

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 $P(A \mid B)$ 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 $P(\text{black} \mid \text{heads})$ of drawing a black ball, given that the initial coin flip showed heads, can be calculated as follows:

$$P(\text{black} \mid \text{heads}) = \frac{P(\text{black}, \text{heads})}{P(\text{heads})} = \frac{0.1}{0.5} = 0.2$$

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:

$$P(B \mid A) = \frac{P(A \mid B) \times P(B)}{P(A)}$$

Bayes rule is straightforward corollary of the definition of conditional probabilities, according to which $P(A \cap B) = P(A \mid B) \times P(B)$, so that:

$$P(B \mid A) = \frac{P(A \cap B)}{P(A)} = \frac{P(A \mid B) \cdot P(B)}{P(A)}$$

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*,[10] 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 Ω of elementary outcomes. A special case occurs when we are interested in a space of numeric outcomes.

A **random variable** is a function $X : \Omega \to \mathbb{R}$ 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.

[10] 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 $P(\{\omega \in \Omega \mid X(\omega) = 2\})$ 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:

$$F_X(x) = P(X \leq x) = \sum_{x' \in \{\text{Rng}(X) | x' \leq x\}} P(X = x)$$

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 θ 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:

$$F_X(x) = P(X \leq x)$$

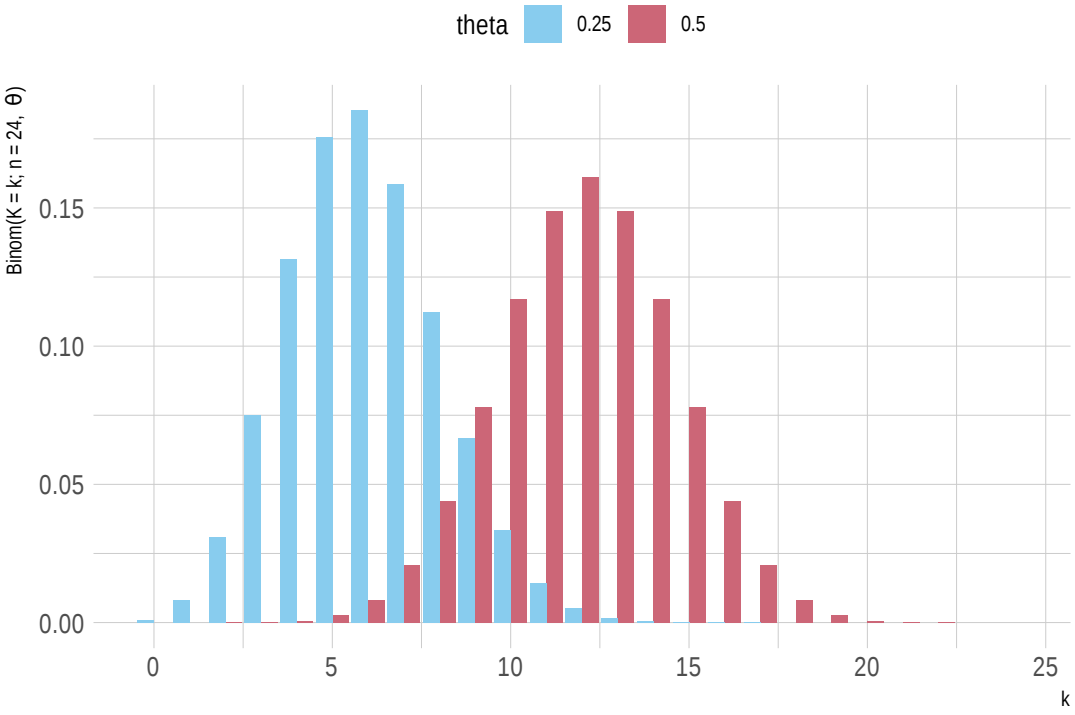


Figure 7.1.: Examples of the Binomial distribution. The y-axis give the probability of seeing k heads when flipping a coin $n = 24$ times with a bias of either $\theta = 0.25$ or $\theta = 0.5$.

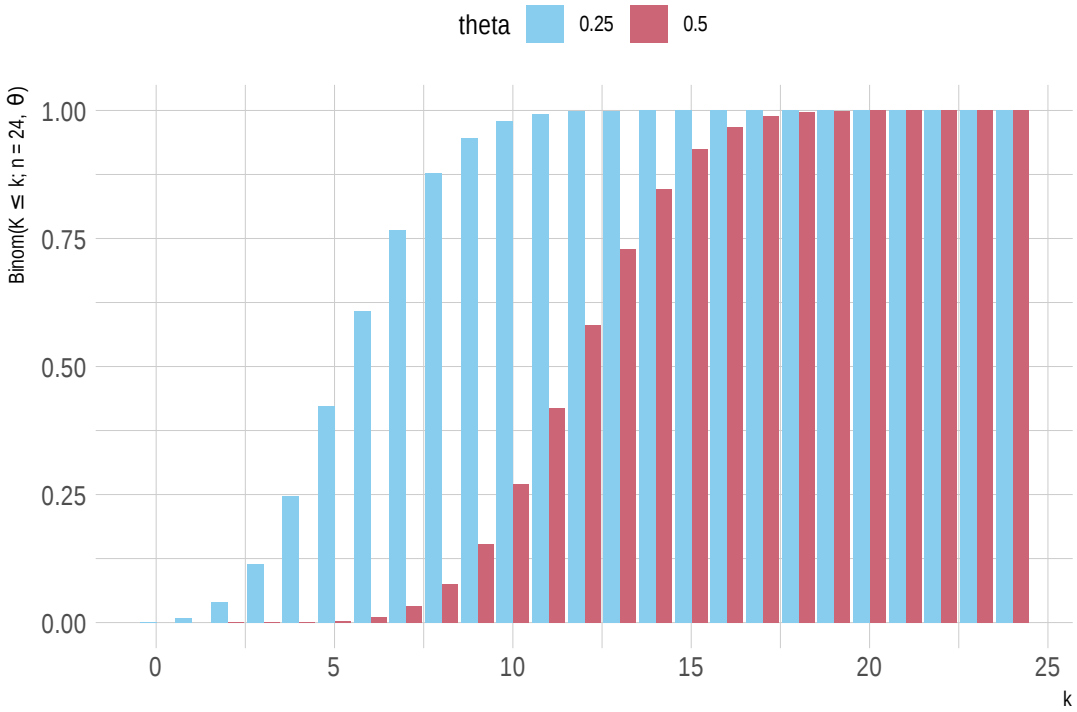


Figure 7.2.: Examples of the cumulative distribution of the Binomial. The y-axis gives the probability of seeing k or less outcomes of heads when flipping a coin $n = 24$ times with a bias of either $\theta = 0.25$ or $\theta = 0.5$.

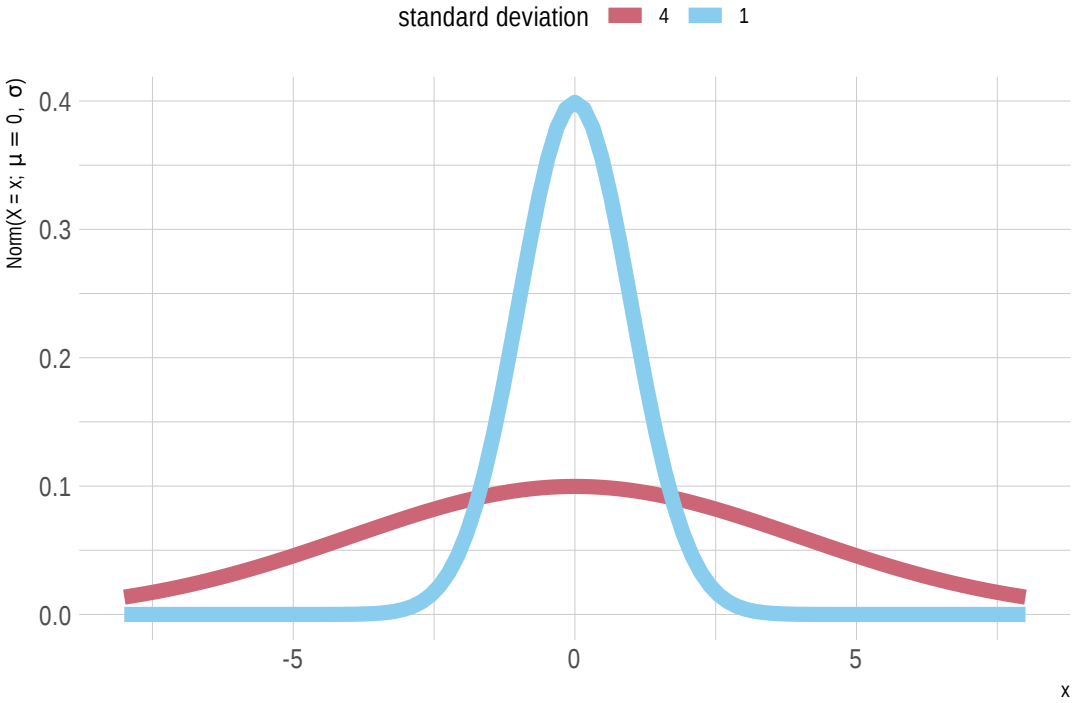


Figure 7.3.: Examples of the Normal distribution. In both cases $\mu = 0$, once with $\sigma = 1$ and once with $\sigma = 4$

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 μ is the *mean*, the central tendency, and parameter σ 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.

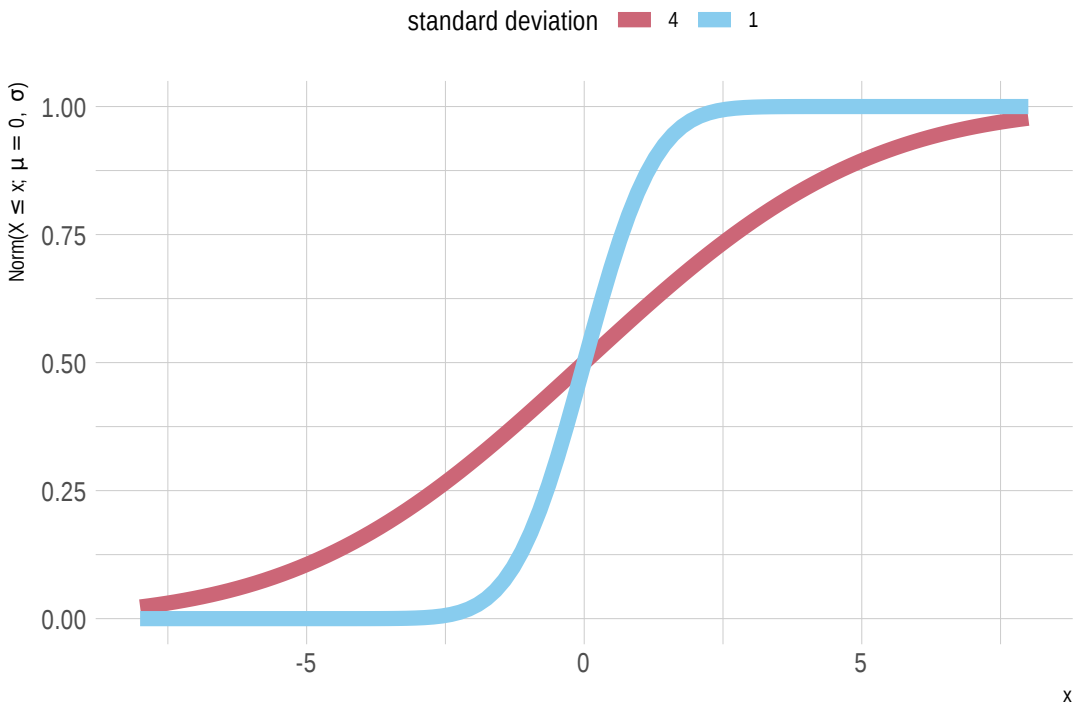


Figure 7.4.: Examples of the cumulative normal distribution corresponding to the previous probability density functions.

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:

$$\mathbb{E}_X = \int x \times f_X(x)\ \mathrm{d}x$$

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:

$$\mathrm{Var}(X) = \sum_x (\mathbb{E}_X - x)^2 \times f_X(x)$$

If X is continuous, it is:

$$\mathrm{Var}(X) = \int (\mathbb{E}_X - x)^2 \times f_X(x)\ \mathrm{d}x$$

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.[11] 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 μ and its variance is σ^2.

[11] 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 θ, the probability of interest, is the value of the relative frequency when n becomes infinitely large. Consequently, the parameter θ 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:

$$P(H|Data) = \frac{P(Data|H) * P(H)}{P(Data)}$$

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 θ exist:

$$p([X = x]|\theta) = \theta^{[x]}(1-\theta)^{(1-[x])}$$

where

- θ 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 θ 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 θ by a given data set. Then θ is unknown and the data are observed. Treating θ 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 θ 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:

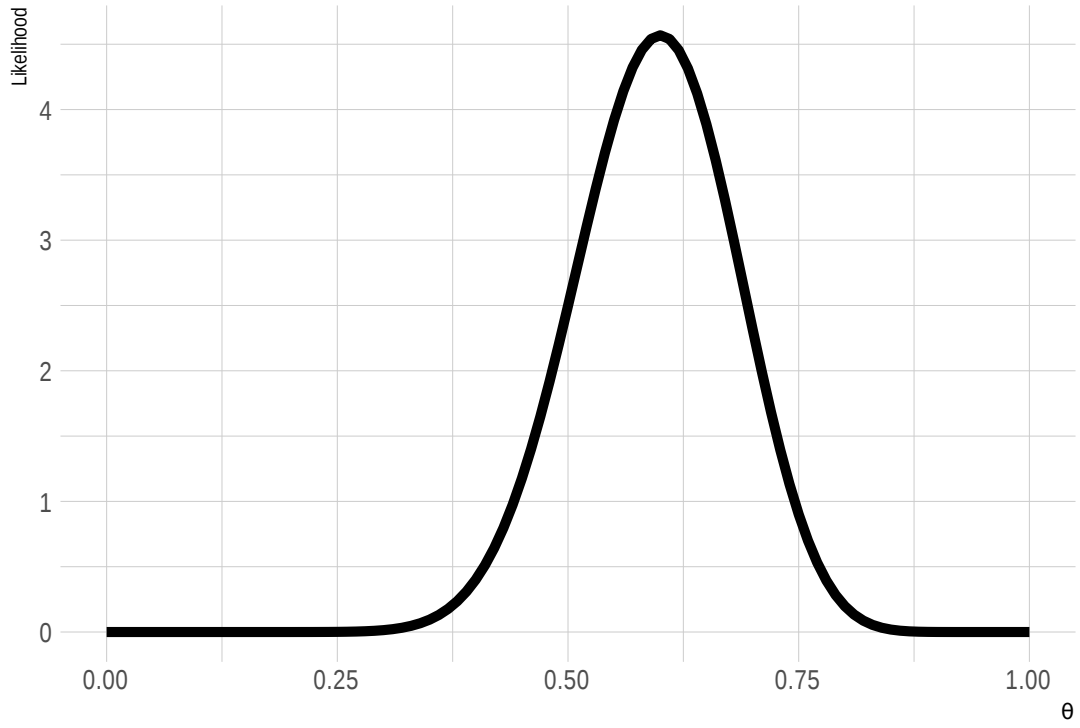


Figure 9.1.: Bernoulli likelihood for simulated observed coin-flip data.

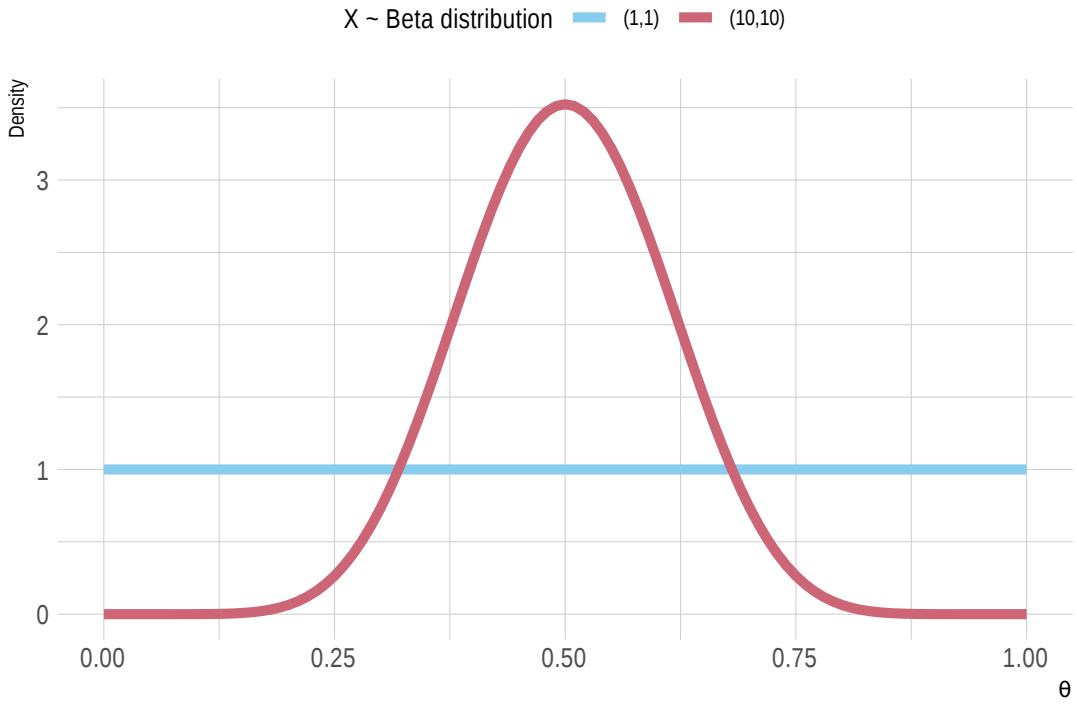


Figure 9.2.: Uninformative vs. informative beta prior distribution.

$$Posterior \propto Prior \cdot Likelihood,$$

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: $\theta \sim Beta(a, b)$ 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).

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

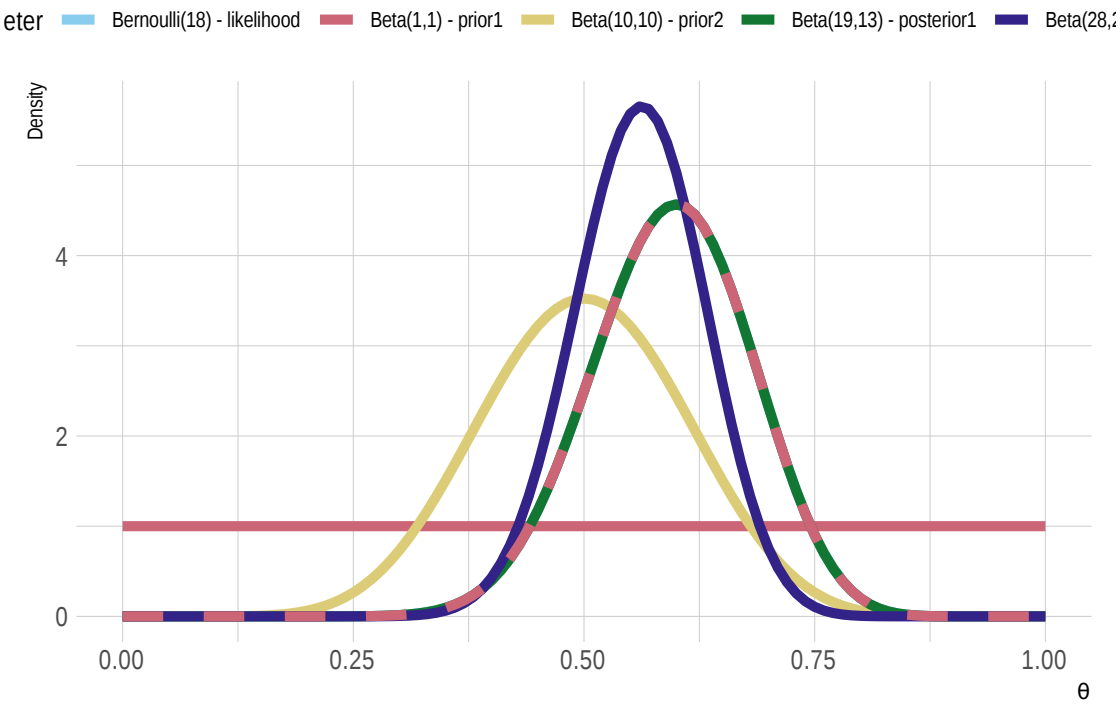


Figure 9.3.: Beta posterior distributions for different beta prior distributions and bernoulli likelihood.

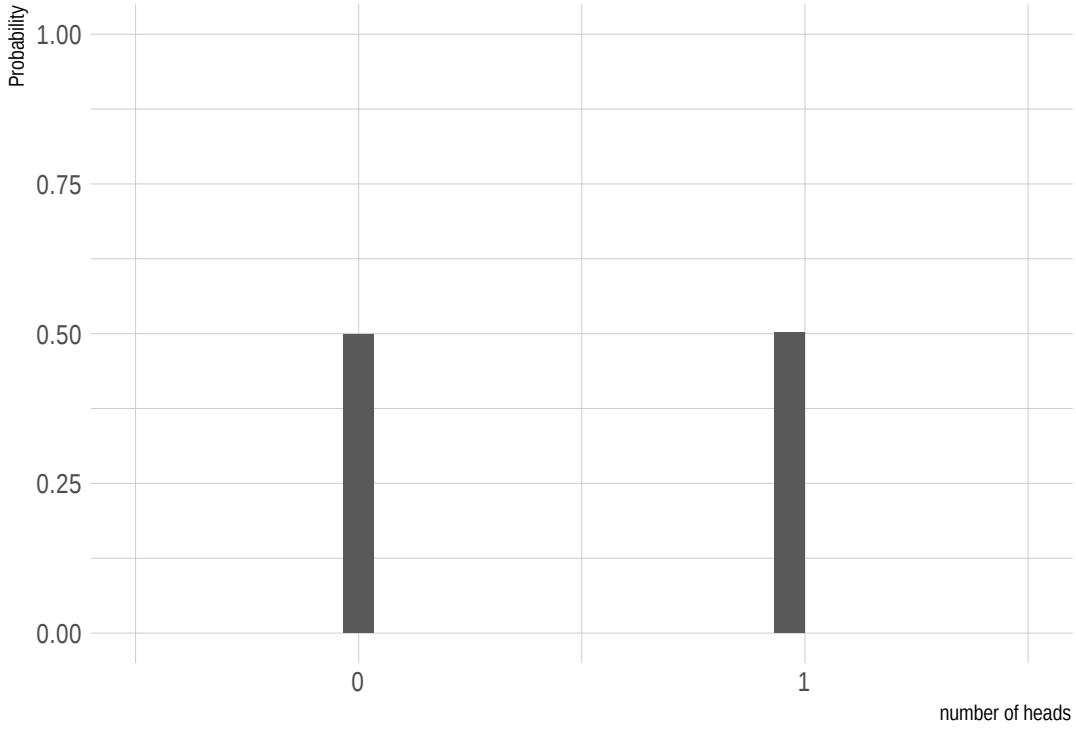


Figure 9.4.: Prior predictive distribution with Beta(1,1) prior for coin flip model

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 θ.

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:

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    14    30
##  2     2    19    30
##  3     3    10    30
##  4     4    19    30
##  5     5    14    30
##  6     6    15    30
##  7     7    14    30
##  8     8    16    30
##  9     9    12    30
## 10    10    10    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 θ is assumed, indicating the probability of heads coming up $p(y = 1)$. The probability that the outcome is head, given a value of parameter θ, is the value of θ (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 θ as unknow 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 θ before seeing the data? Assuming that no expectation about θ exists a priori, indicating that all values of θ 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.

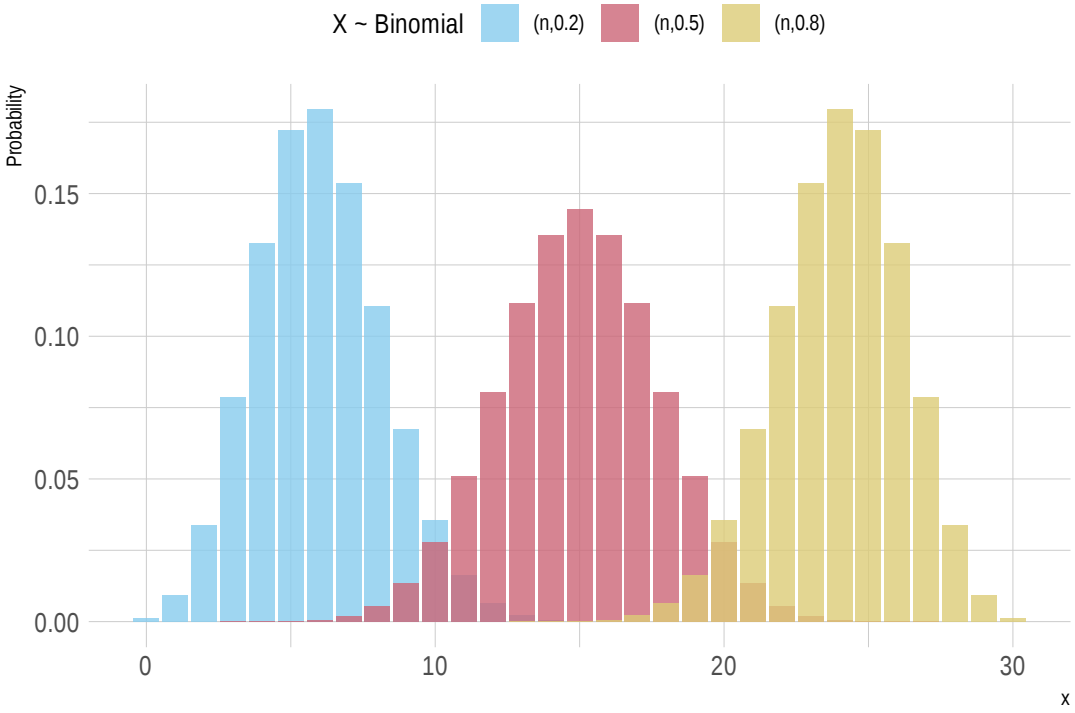


Figure 9.5.: Binomial-distribution for different coin biases

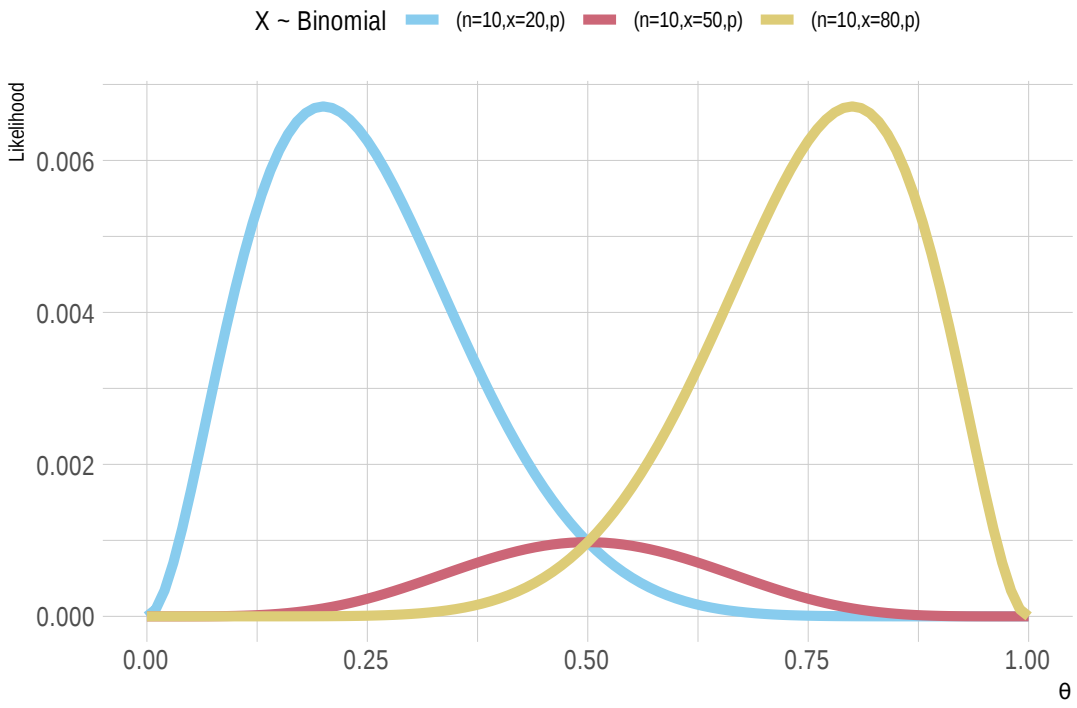


Figure 9.6.: Binomial likelihoods for different observed coin flip outcomes.

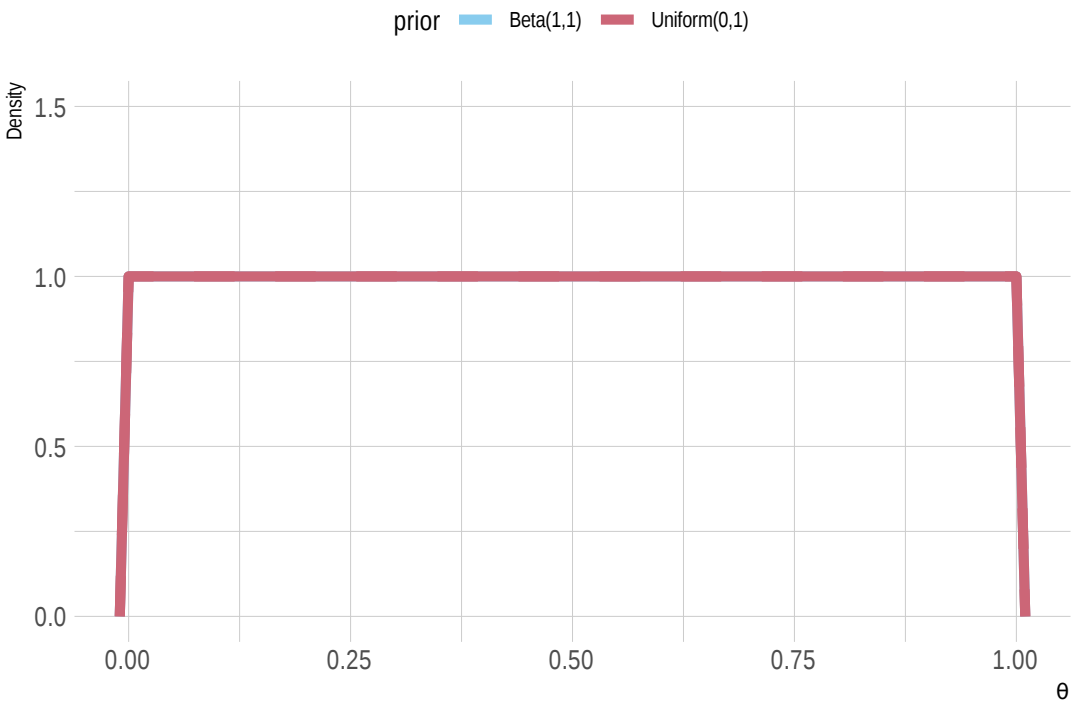


Figure 9.7.: Using uninformative prior distributions: The Uniform(0,1) and Beta(1,1) prior.

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 θ 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 θ. 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 Wagenmakers 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 Wagenmakers 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 (θ),
- observed variables: n and k,

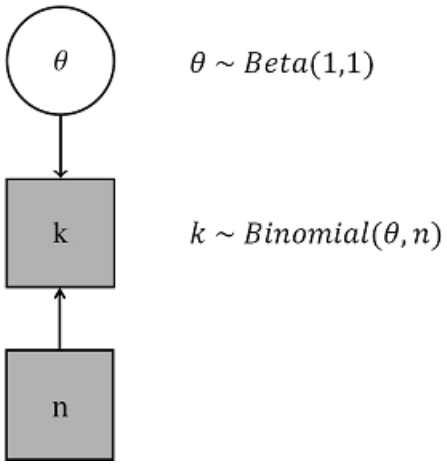


Figure 9.8.: Graphical notation Beta-Binomial Model - One group

- unobserved variables: θ,
- continuous variable: θ,
- 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 θ as well as on the number of trials n.

Finally, the graphical structure together with the textual notation can be represented:

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 θ. Further, θ 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:

$\omega = \frac{a-1}{a+b-2}$,

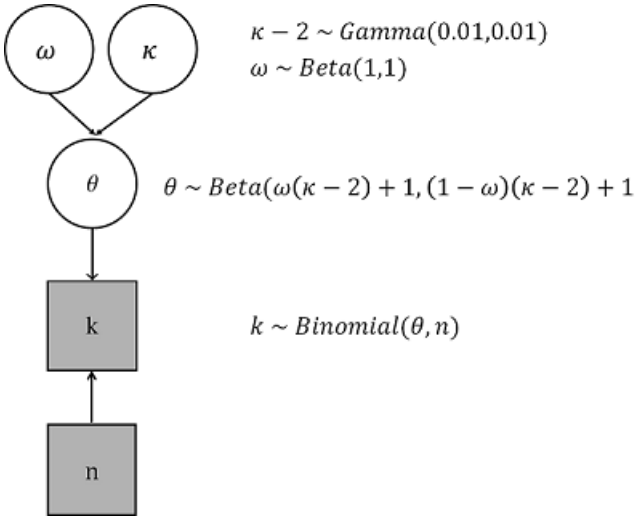


Figure 9.9.: Graphical notation hierarchical Beta-Binomial Model - One group

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 ω and κ, such that:

$$Beta(a, b) = Beta\left(\omega(\kappa - 2) + 1, (1 - \omega)(\kappa - 2) + 1\right).$$

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 θ are fixed: $a = 1$ and $b = 1$. Assume that we get further information: The manufacturing process of the coins has a bias near ω (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 ω.

Such that the model can be assumed as follows:

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:

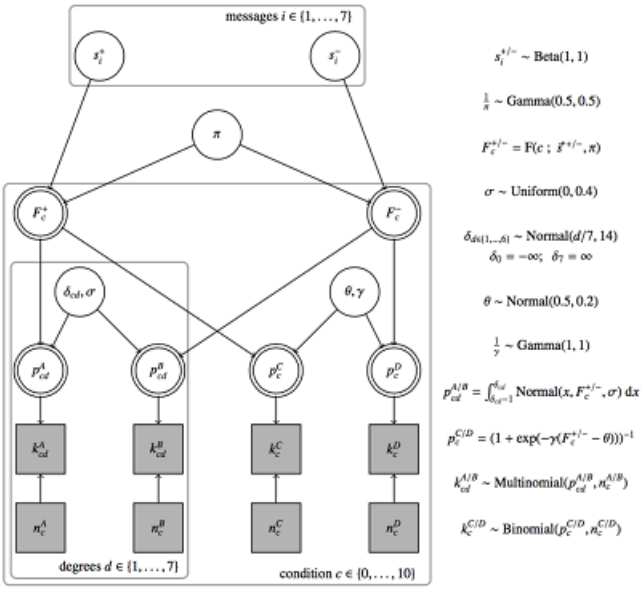


Figure 9.10.: Graphical notation hierarchical Beta-Binomial Model - One group

9.3. Further examples

9.3.1. Difference between two groups

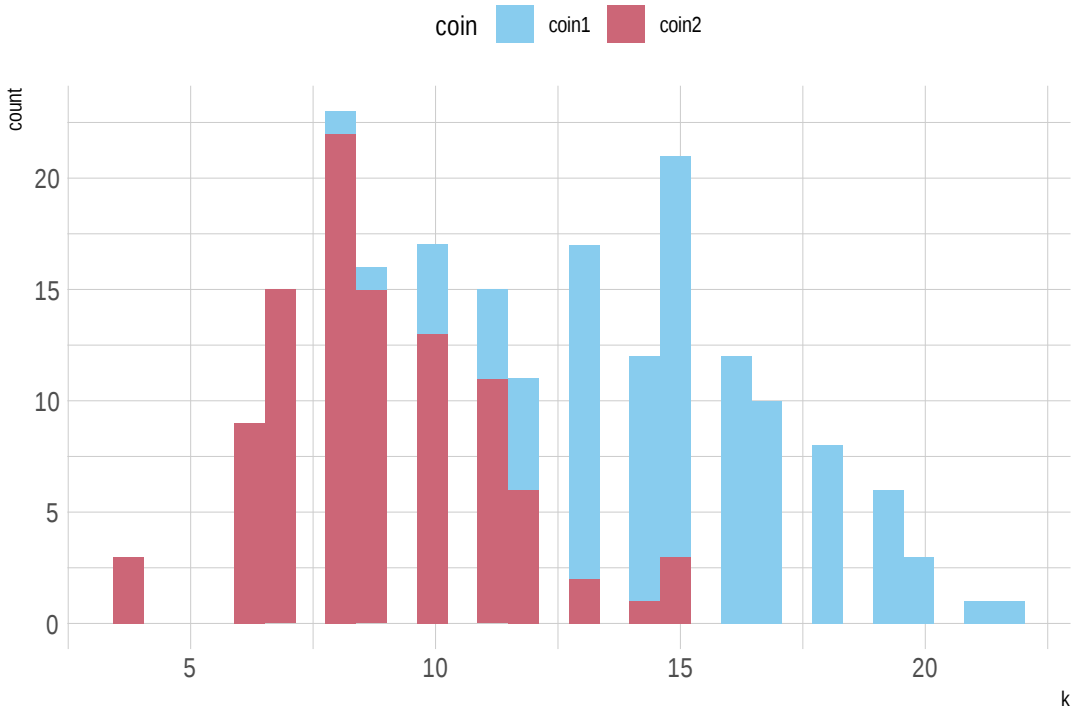
In the introductory example we asked for the underlying probability θ of a single coin that was flipped repeatedly. Consider now, that a second coin y_2 is introduced. One question that arises might be for example: *How different are the biases of the two coins?*

```
#simulate flips of two coins
sample.space <- c(0,1)
##First coin:
theta1 <- 0.5               # probability of a success (here: head)
X1 <- 30                    # number of trials in the experiment
n1 <- 100                   # number of observations
k1 <- 0                     # number of heads [initialization]

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               # probability of a success (here: head)
X2 <- 30                    # number of trials in the experiment
n2 <- 100                   # number of observations
```

```
k2 <- 0                    # number of heads [initialization]

## repeat experiment N-times
for (i in 1: n2) {
  k2[i] <- sum(sample(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    15    30
##  2 coin1     2    13    30
##  3 coin1     3    15    30
##  4 coin1     4    15    30
##  5 coin1     5    21    30
##  6 coin1     6    17    30
##  7 coin1     7    12    30
##  8 coin1     8    17    30
##  9 coin1     9    14    30
## 10 coin1    10    14    30
## # ... with 190 more rows
```

```
#Plotting the observed results
ggplot(data=coin.flip2,mapping = aes(x=k, fill=coin ))+
          geom_histogram()
```



9.3.1.1. Conceptual steps for modeling

We suppose that the underlying probabilities of the two coins correspond to *different* latent variables θ_1 and θ_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 θ_1 and θ_2.

Furthermore, from a conceptional perspective, we are interested in the *difference between the coin biases*. Therefore a further variable will be introduced δ, 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:

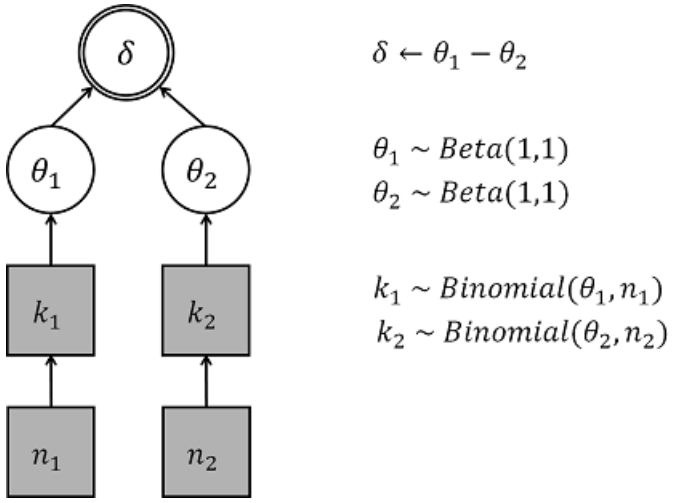


Figure 9.11.: Graphical notation Beta-Binomial Model - Two groups

9.3.1.2. 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)
```

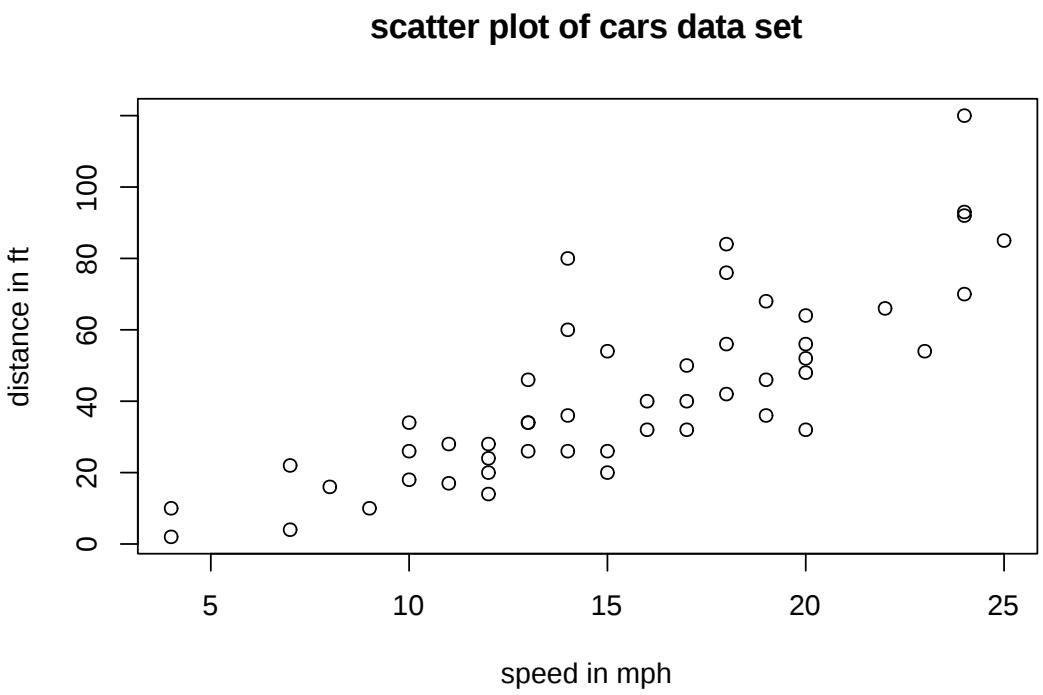
```
## 'data.frame':    50 obs. of  2 variables:
##  $ speed: num  4 4 7 7 8 9 10 10 10 11 ...
##  $ dist : num  2 10 4 22 16 10 18 26 34 17 ...
```

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 *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 i 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 σ. 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 knowlege** (before seeing the data) has to be defined. The parameters of the current model are the predicted value μ and the standard deviation σ. For the parameter μ a normal distribution can be assumend 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 σ a uniform distribution is assumed:

$$\sigma \sim Uniform(0, 40)$$

9.3.2.2. Excursus: 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:

$$y_i \overset{\text{iid}}{\sim} Normal(\mu, \sigma).$$

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 dependency relevant for the model? If yes, this information has to be added in the model (e.g. in form of a hierarchical model). Janyes 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 do 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 usefull to assume iid (McElreath 2015).

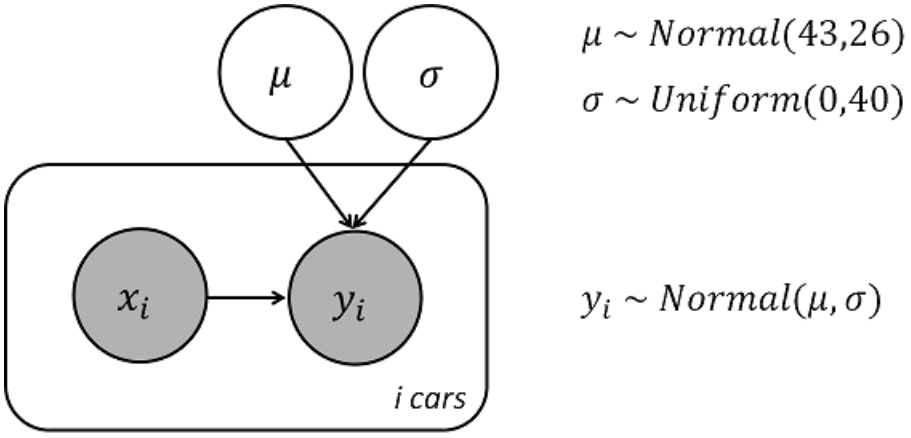


Figure 9.12.: Graphical notation Simplre Regression model

9.3.3. 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 θ for observing head as outcome of a coin flip has been discussed. But the estimation of θ 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 θ *parameter(s)* are needed. When it comes to estimation exactly this/these parameter(s) will be the result(s), therefore is is important to see already the connection to the models that were developed in this chapter.

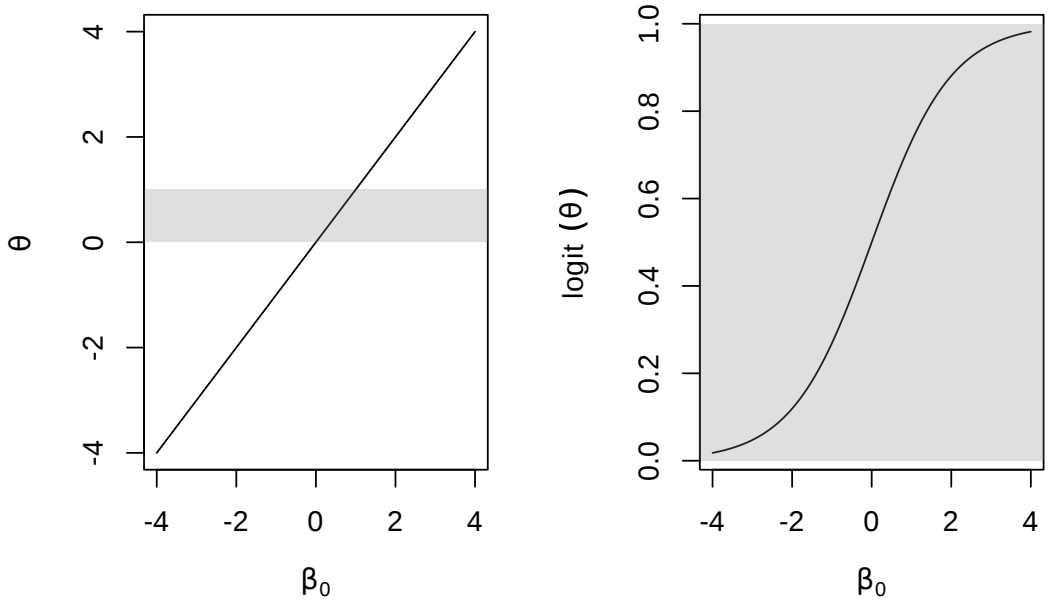
For the coin flip example the value of interest is the underlying probability, thus, only one parameter is needed:β_0. (Note: Latin letters are used when we refer to the sample, Greek letters are used when we refer to the population.)

How is β_0 linked to the latent variable θ?

Considering for example the simplest case: a *linear relationship* (see next plot left side). The problem which arises at this point is that θ 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 β_0 can take any value while θ is bounded to the range 0-1. One transformation that offers exactly this possibility is the *logit link function* (see above plot right side)

$$logit(\theta) = \beta_0.$$

As the underlying assumption maps the parameter to the latent variable θ (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:

$$\theta = logistic(\beta_0).$$

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 β_0 is estimated from the given sample. It defines the latent variable θ. The parameter is mapped to θ by a logistic link function. The underlying probability θ 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 δ 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:

$$\theta_j = \beta_0 + \beta_1 * X_{Group_j},$$
$$\text{with } X_{Group_j} = \begin{cases} 0, \text{if coin 2,} \\ 1, \text{if coin 1.} \end{cases}$$

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:

$$\theta_1 = \beta_0 + \beta_1.$$

The proportion of coming up head for coin 1 has to be calculated by summing up the *intercept* β_0 and the *slope* β_1.

Taken together: *What is the interpretation of the slope β_1?* The difference $\delta = \theta_1 - \theta 2$ is

$$\theta_1 - \theta_2 = (\beta_0 + \beta_1) - \beta_0 = \beta_1 = \delta,$$

the slope β_1, thus, we can see that this parameterization enables us to estimate the difference between two groups. When in 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.

Appliied (generalized) linear modeling

15. Simple linear regression

- "multiple" = "more than one predictor"
  - interactions
  - collinearity
- categorical predictors
  - relation to t-test and ANOVA
  - different coding shemes
- robust regression

16. Logistic regression

to do

17. Multinomial regression

todo

18. Ordinal regression

todo

19. Hierarchical regression

todo

A. Further useful material

Section A.1 lists further study material in the form of books, papers, web resources etc.

A.1. Reading material

A.1.1. 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.1.2. Material on *Baysian Data Analysis*:

- "Doing Bayesian data analysis: A tutorial with R, JAGS, and Stan" by J. Kruschke (Kruschke 2015)
- "Baysian 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.1.3. Material on *frequentist statistics*:

- "Statistics for LinguistsL: An introduction using R", by B. Winter (Winter 2019-)

A.1.4. 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.1.5. Further information for RStudio

- *Keyboard shortcuts* for Windows and Mac in RStudio: "Tools -> Keyboard Shortcuts Help" or also on the RStudio support site

A.1.6. 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 μ and σ 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 μ and its width is scaled by choice of σ. The distribution is symmetric with most observations lying aroung the central peak μ and more extreme values are further away depending on σ.

$$X \sim Normal(\mu, \sigma^2)$$

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",
```

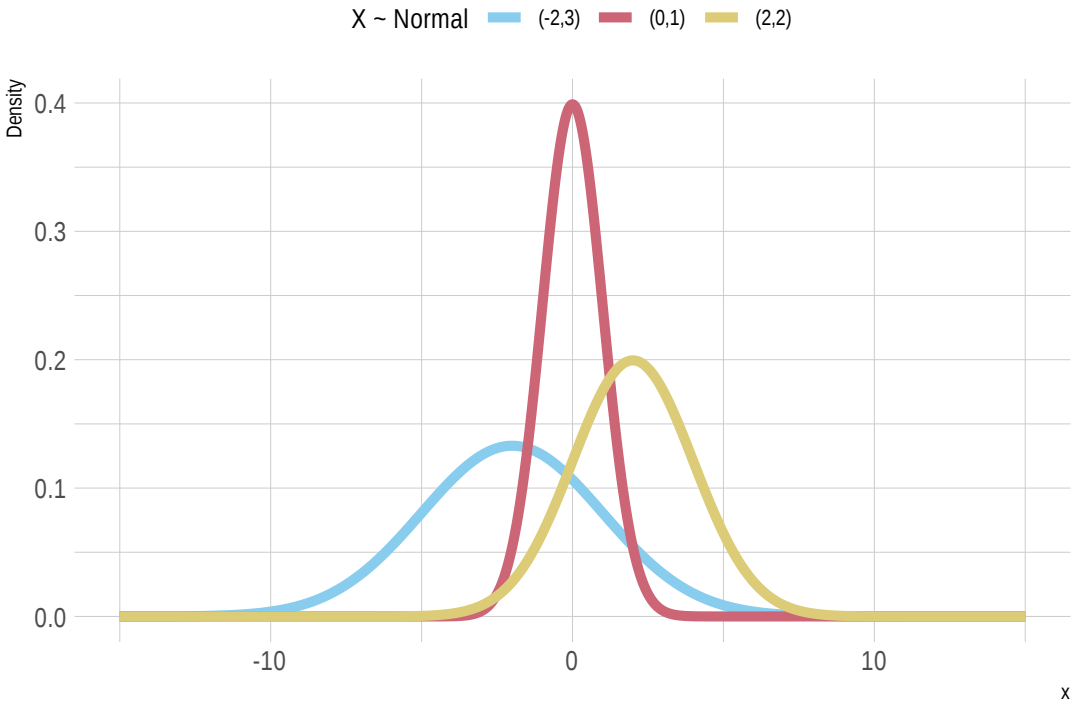


Figure B.1.: Examples of probability density function of normal distributions.

```
                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")

rv_normal %>%
  group_by(parameter) %>%
  mutate(
    cum_y = cumsum(y)/sum(y)
  ) %>%
  ungroup() %>%
  ggplot(aes(x, cum_y, color = parameter)) +
```

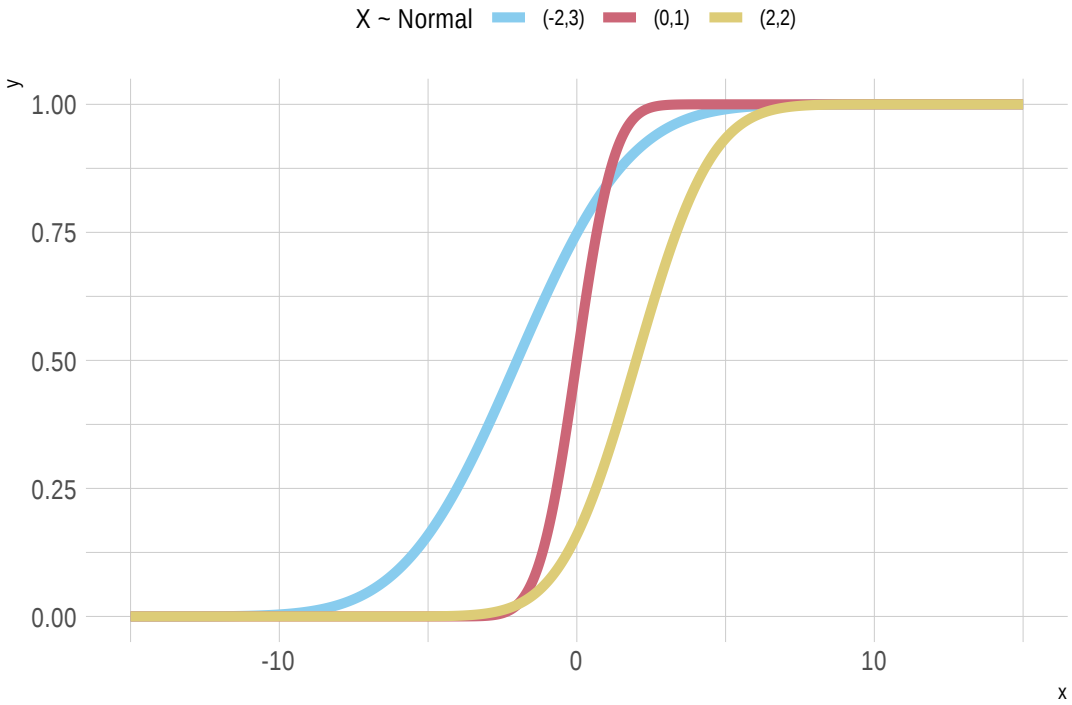


Figure B.2.: Examples of the cumulative distribution function of normal distributions corresponding to the previous probability density functions.

```
geom_line(size = 2) +
labs(color = "X ~ Normal", y = "y")
```

A special case of normal distributed random variables is the *standard normal* distributed variable with $\mu = 0$ and $\sigma = 1$: $Y \sim Normal(0, 1)$. 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

$$F(x) = \int_{-\inf}^{x} f(t)dt$$

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 σ away from the mean μ.

- $P(\mu - \sigma \leq X \leq \mu + \sigma) = 0.6827$
- $P(\mu - 2\sigma \leq X \leq \mu + 2\sigma) = 0.9545$
- $P(\mu - 3\sigma \leq X \leq \mu + 3\sigma) = 0.9973$

```
# 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)))
```

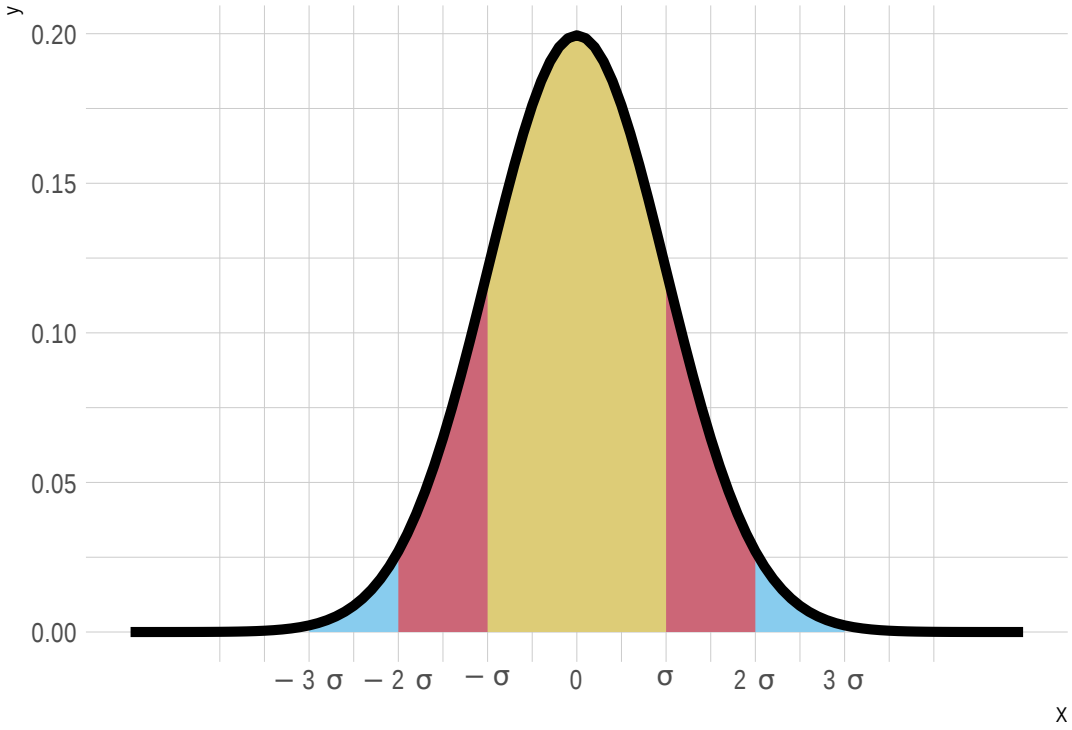


Figure B.3.: Coverage of normal distribution

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 $X \sim Normal(\mu_x, \sigma^2)$ and $Y \sim Normal(\mu_y, \sigma^2)$ 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 χ^2-distribution is widely used in hypothesis testing in inferential statistics, because many test statistics are approximately distributed as χ^2-distribution.

The χ^2-distribution is directly related to the standard normal distribution: The sum of n independent and standard normal distributed random variables $X_1, X_2, ..., X_n$ is distributed according to a χ^2 distribution with n *degrees of freedom*:

$$Y = X_1^2 + X_2^2 + ... + X_n^2.$$

The χ^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),
  y3 = dchisq(x, df = 9)
) %>%
```

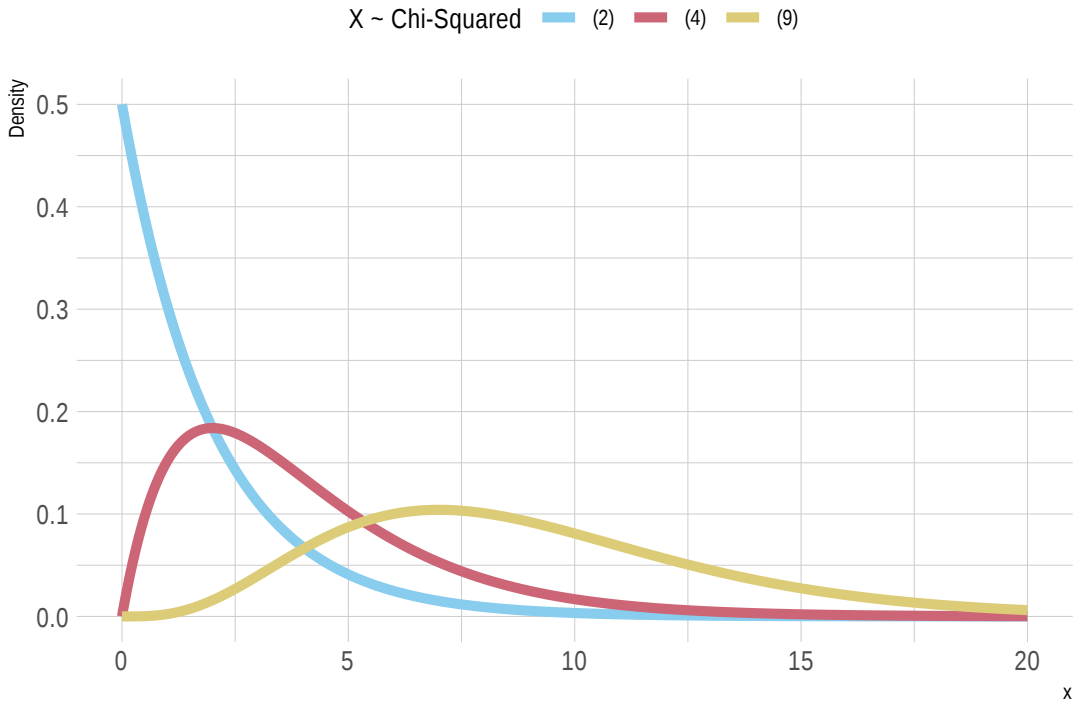


Figure B.4.: Examples of probability density function of chi-squared distributions.

```
  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")

rv_chisq %>%
  group_by(parameter) %>%
  mutate(
    cum_y = cumsum(y)/sum(y)
```

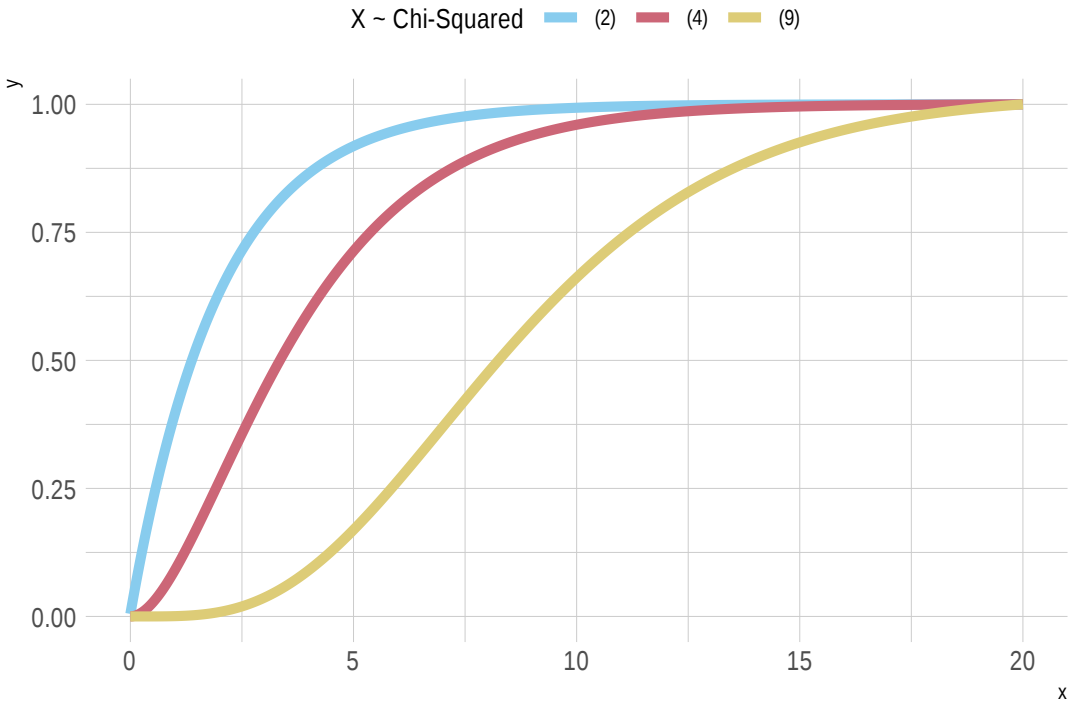


Figure B.5.: Examples of the cumulative distribution function of chi-squared distributions corresponding to the previous probability density functions.

```
) %>%
ungroup() %>%
ggplot(aes(x, cum_y, color = parameter)) +
  geom_line(size = 2) +
  labs(color = "X ~ Chi-Squared", y = "y")
```

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 χ^2-distributed random variables $X \sim \chi^2(m)$ and $Y \sim \chi^2(n)$ is again a chi^2-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^2-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^2-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.

```
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),
```

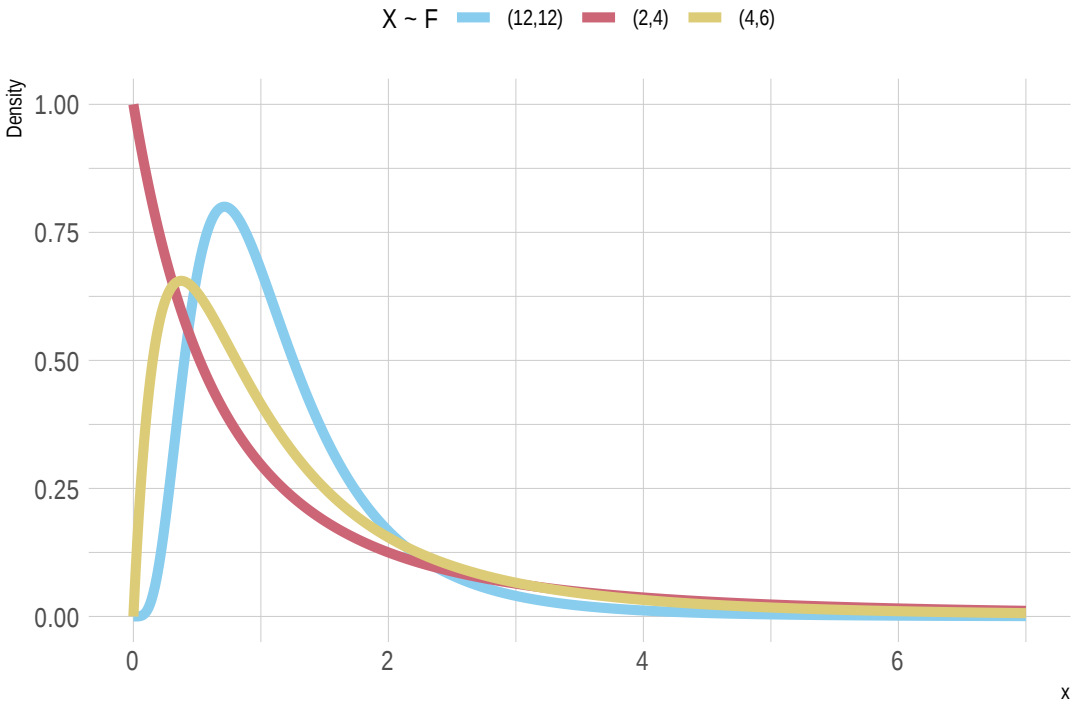


Figure B.6.: Examples of probability density function of F distributions.

```
  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) %>%
```

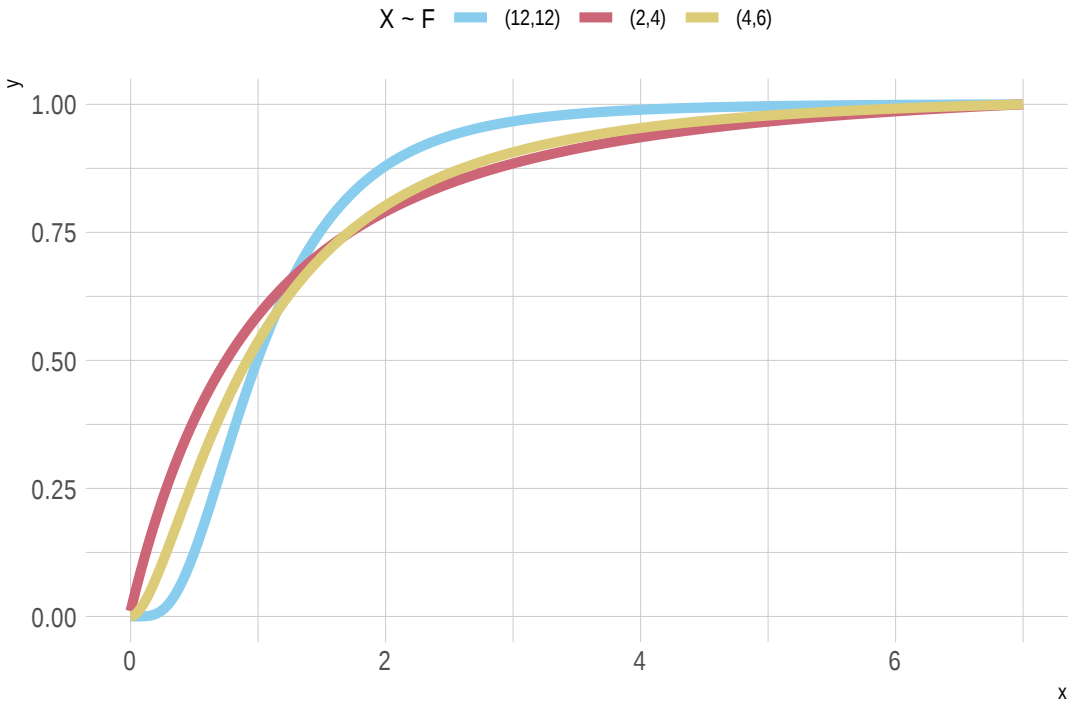


Figure B.7.: Examples of the cumulative distribution function of F distributions corresponding to the previous probability density functions.

```
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:

$$I(z, a, b) = \frac{1}{B(a, b)} \cdot \int_0^z t^{a-1}(1-t)^{b-1} dt.$$

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 ressembles 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 χ^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.

```
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:

$$I(z, a, b) = \frac{1}{B(a, b)} \cdot \int_0^z t^{a-1}(1-t)^{b-1} dt.$$

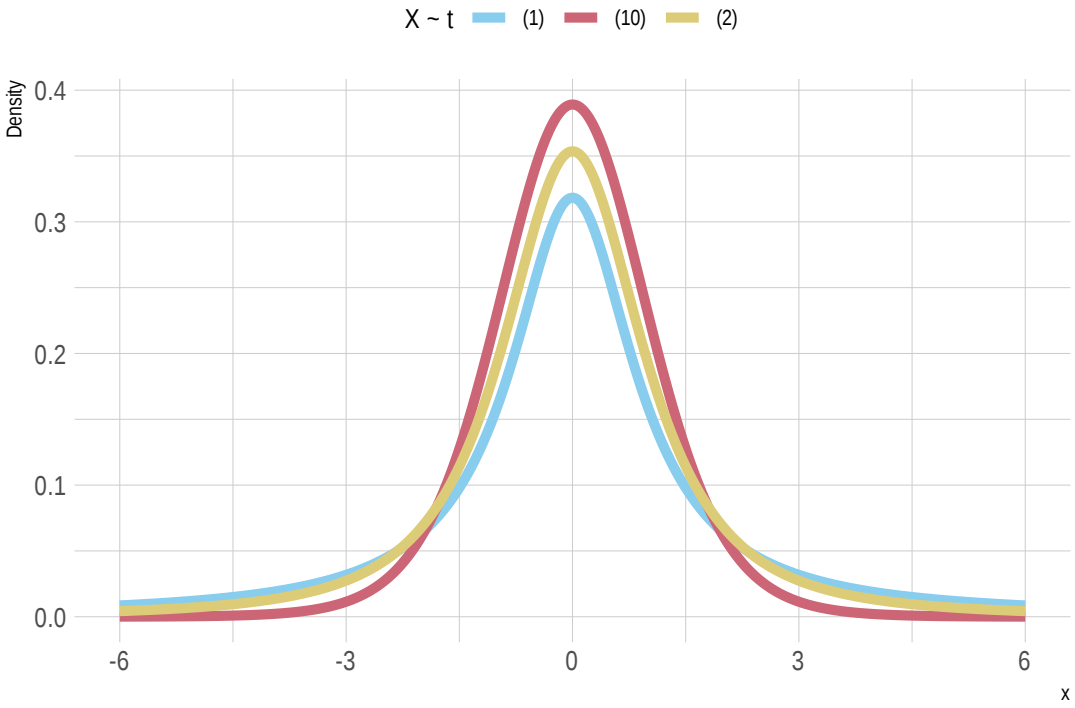


Figure B.8.: Examples of probability density function of t distributions.

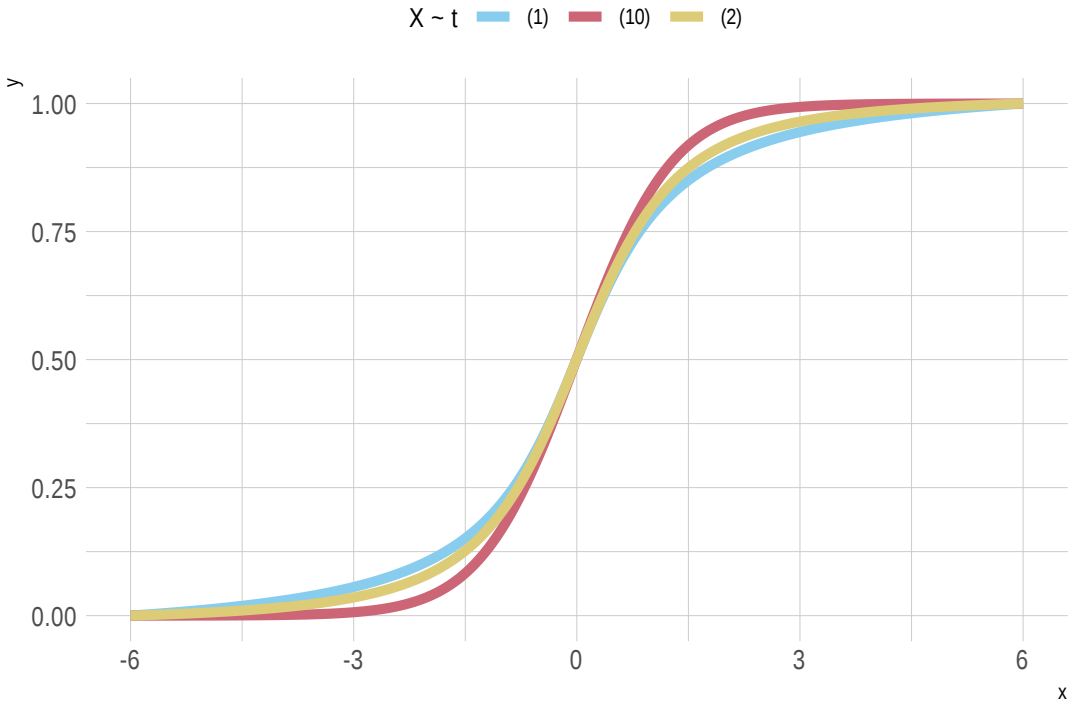


Figure B.9.: Examples of the cumulative distribution function of t distributions corresponding to the previous probability density functions.

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) liklihood 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(
    parameter = case_when(parameter == "y1" ~ "(1,1)",
```

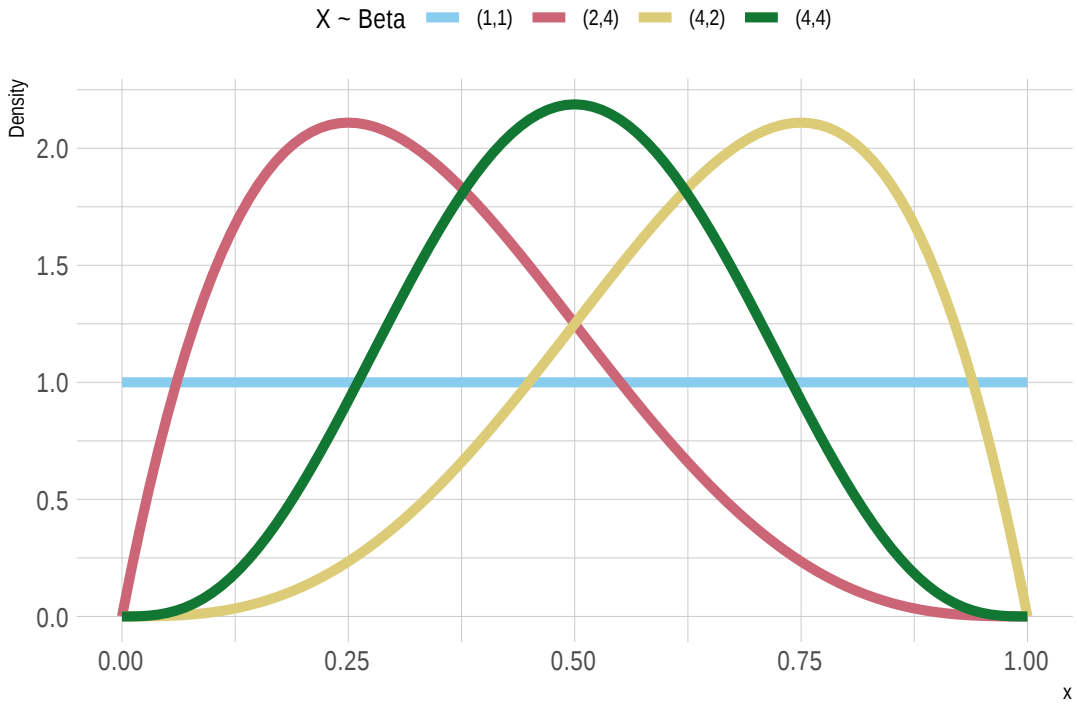


Figure B.10.: Examples of probability density function of beta distributions.

```
                        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")

rv_beta %>%
  group_by(parameter) %>%
  mutate(
    cum_y = cumsum(y)/sum(y)
  ) %>%
  ungroup() %>%
  ggplot(aes(x, cum_y, color = parameter)) +
    geom_line(size = 2) +
```

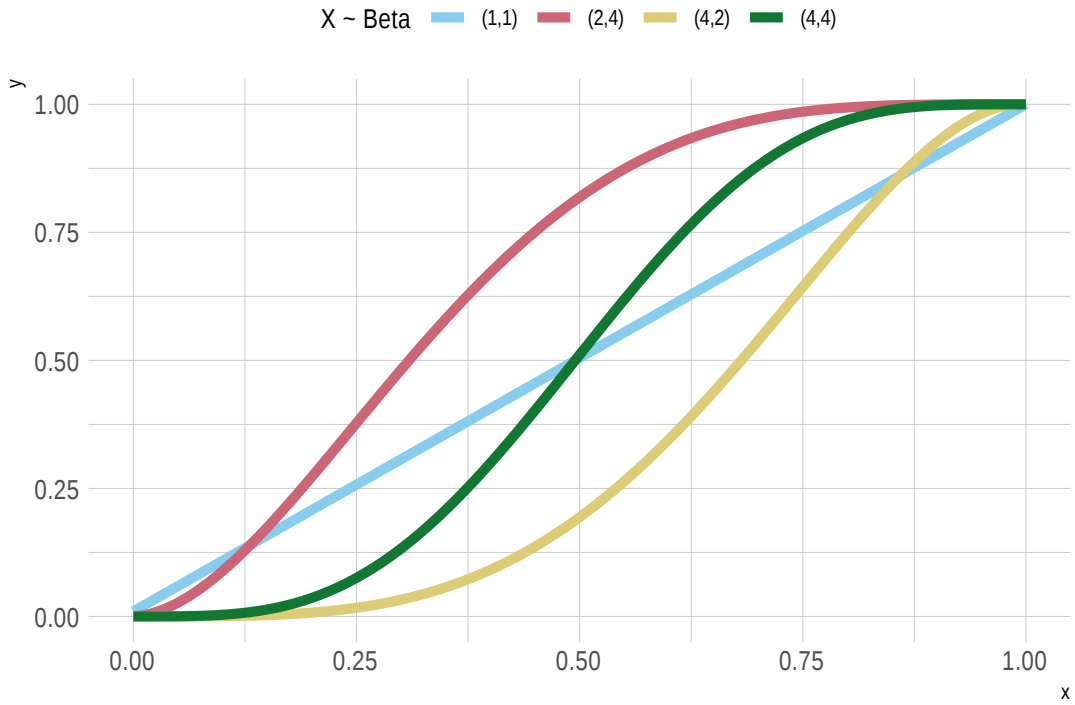


Figure B.11.: Examples of the cumulative distribution function of beta distributions corresponding to the previous probability density functions.

```
labs(color = "X ~ Beta", y = "y")
```

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

$$F(x) = \frac{B(x;a,b)}{B(a,b)},$$

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 ω and concentration κ 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 occurence, 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):

$$X \sim Unif(a, b).$$

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.

```
rv_unif <- tibble(
  x = seq(from = -.1, to = 4.2, by = .01),
  y1 = dunif(x),
  y2 = dunif(x, min = 2, max = 4)
```

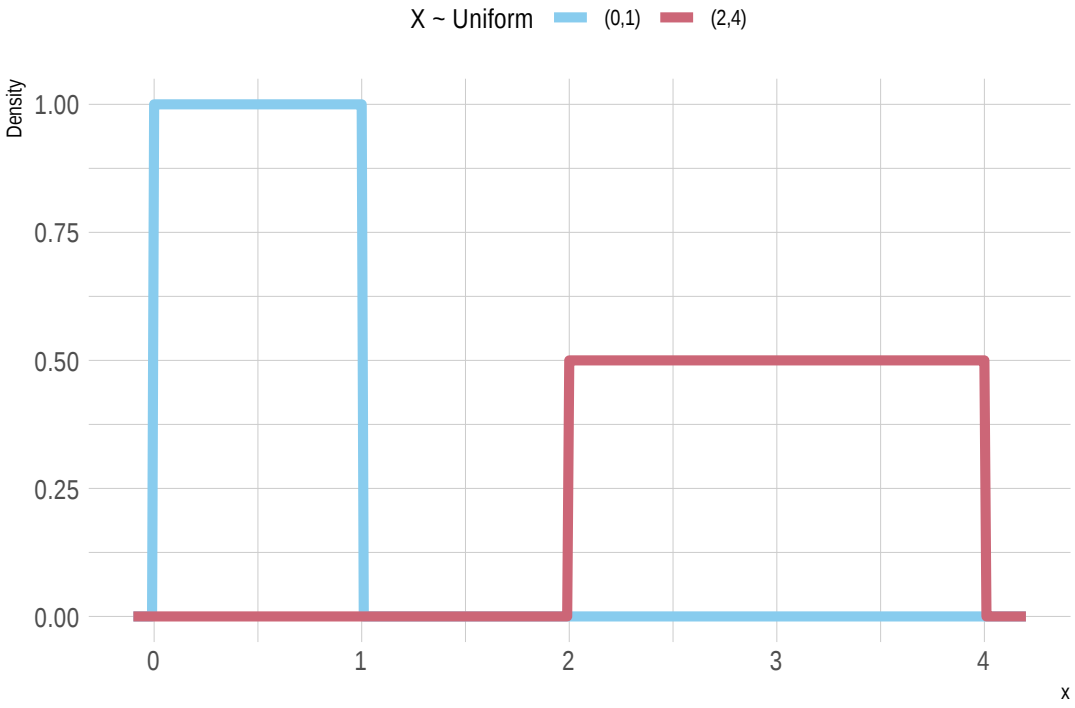


Figure B.12.: Examples of probability density function of uniform distributions.

```
) %>%
  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")

rv_unif %>%
  group_by(parameter) %>%
  mutate(
    cum_y = cumsum(y)/sum(y)
```

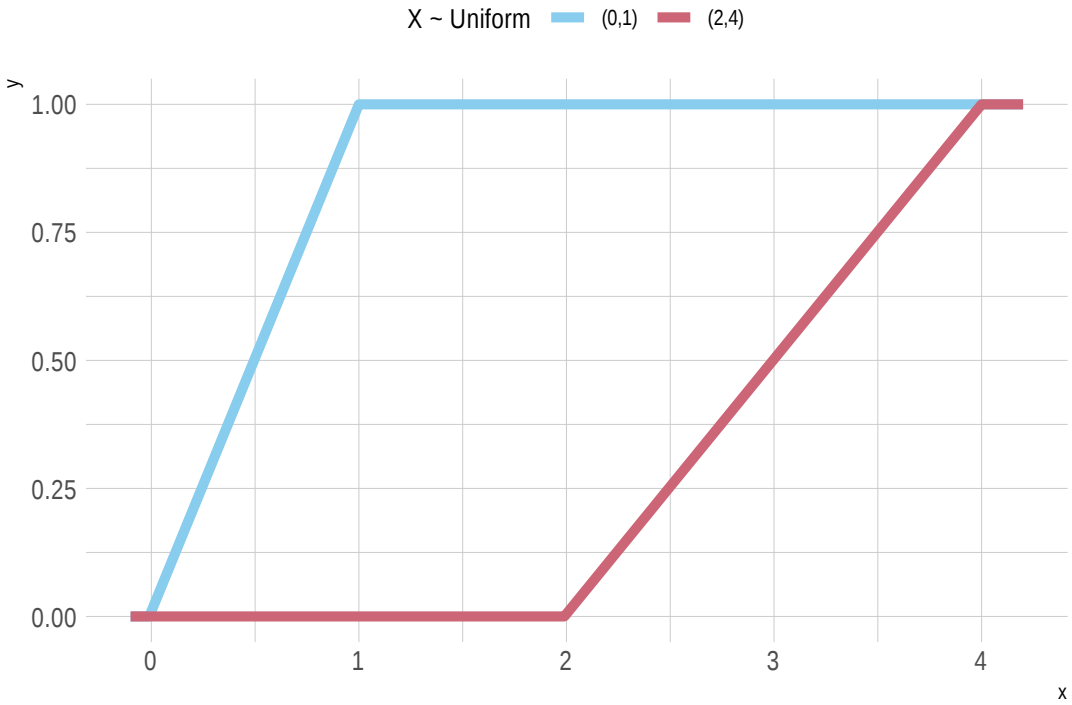


Figure B.13.: Examples of the cumulative distribution function of uniform distributions corresponding to the previous probability density functions.

```
) %>%
ungroup() %>%
ggplot(aes(x, cum_y, color = parameter)) +
  geom_line(size = 2) +
  labs(color = "X ~ Uniform", y = "y")
```

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 occurence of the outcome "success" be p, then the probability of occurence 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 $\{0, 1, 2, ..., 30\}$.
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"),
               names_to  = "parameter",
               values_to = "y") %>%
  mutate(
    parameter = case_when(parameter == "y1" ~ "(n,0.2)",
                          parameter == "y2" ~ "(n,0.5)",
```

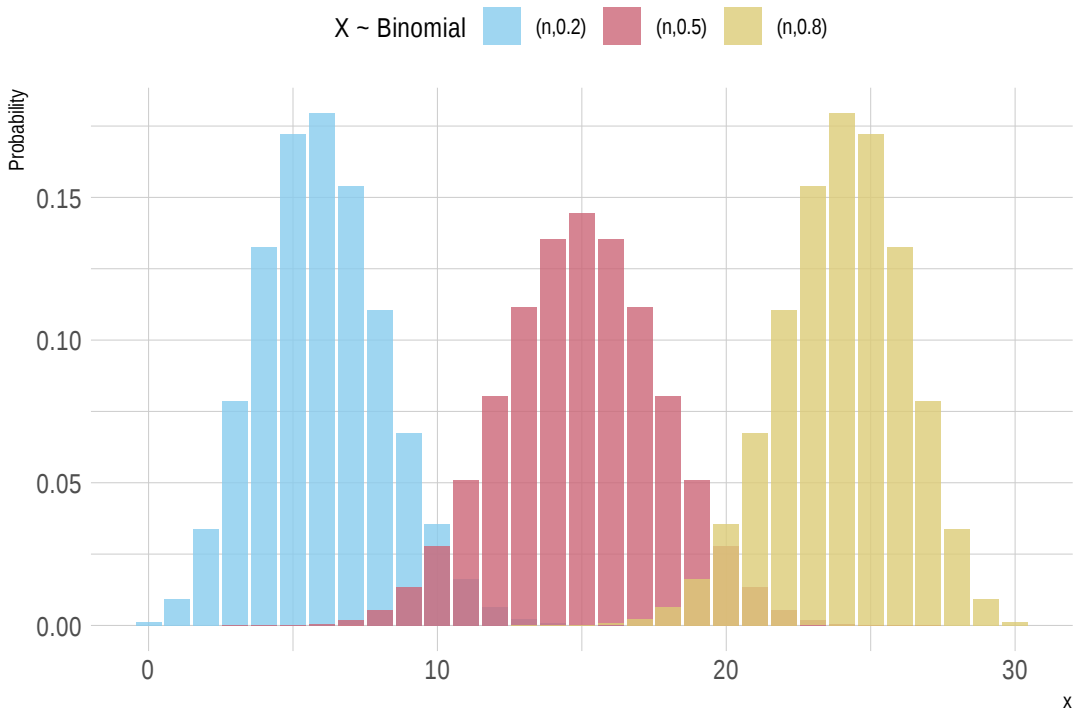


Figure B.14.: Examples of probability mass function of Binomial distributions.

```
                        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")

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")
```

Probability mass function

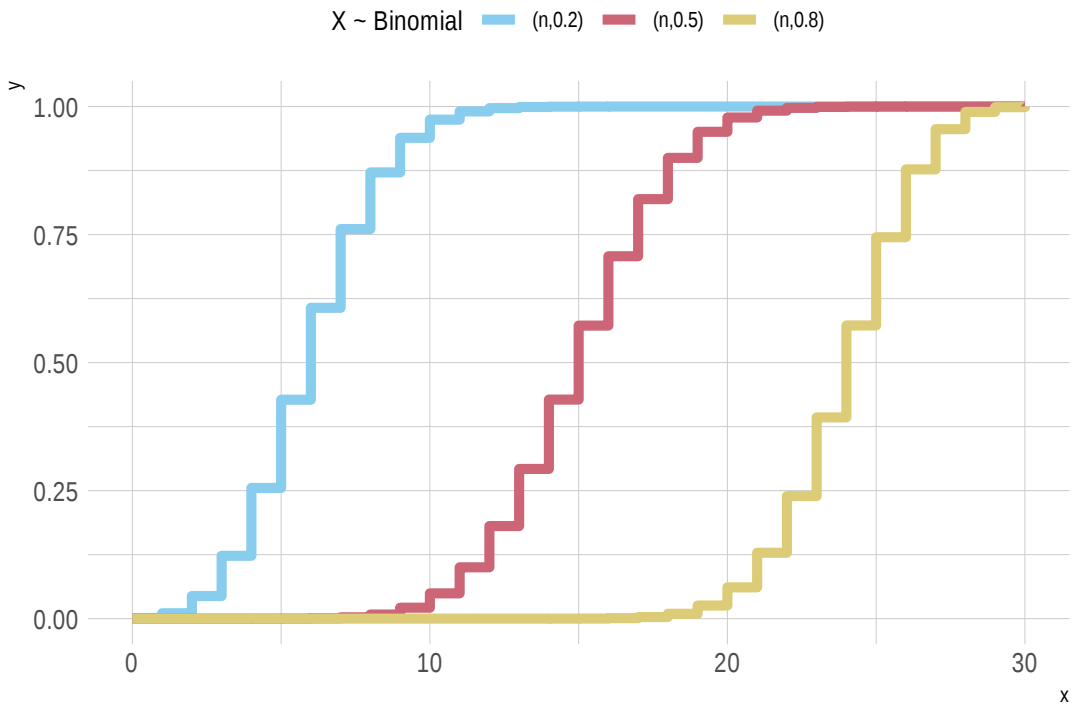


Figure B.15.: Examples of the cumulative distribution function of Binomial distributions corresponding to the previous probability mass functions.

$$f(x) = \binom{n}{x} p^x (1-p)^{n-x},$$

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.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:

$$X \sim Bern(p).$$

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(
```

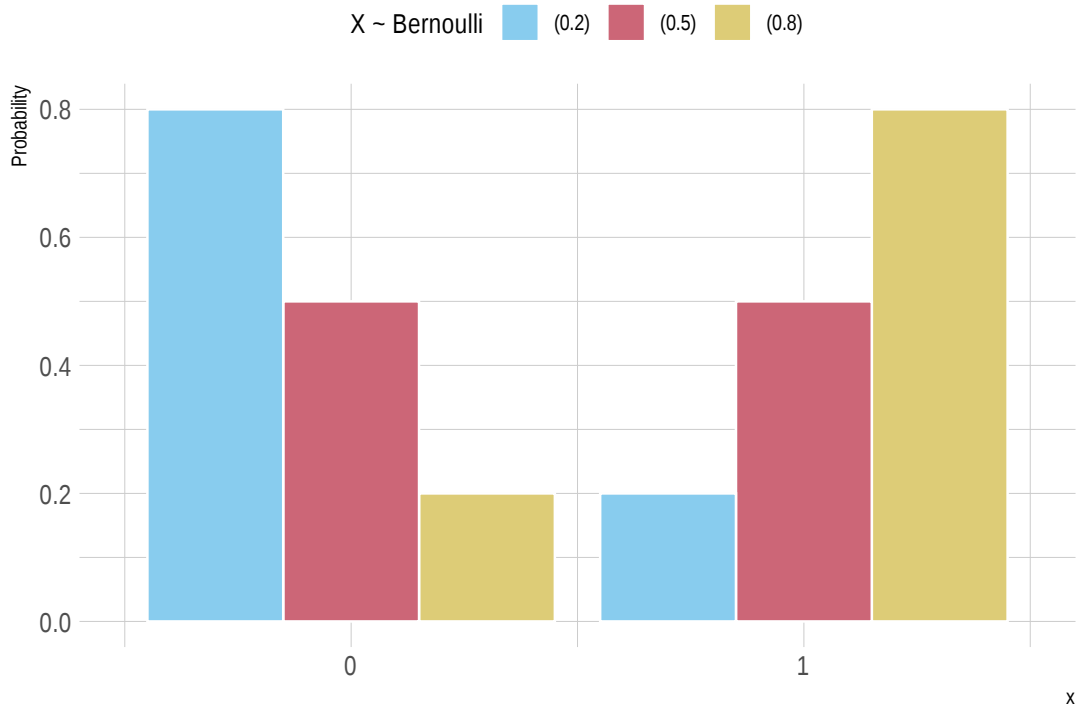


Figure B.16.: Examples of probability mass function of Bernoulli distributions.

```
    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)

rv_bern %>%
  group_by(parameter) %>%
  mutate(
    cum_y = cumsum(y)/sum(y),
    cum_y2 = cumsum(y)/sum(y)
  ) %>%
  add_column(
```

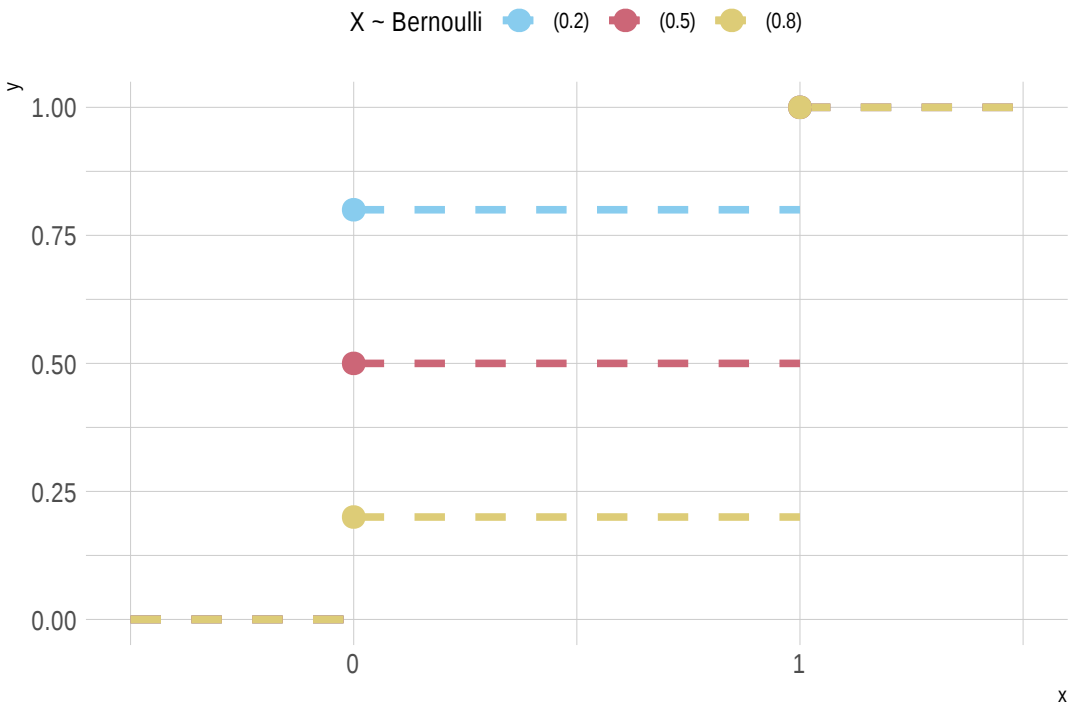


Figure B.17.: Examples of the cumulative distribution function of Bernoulli distributions corresponding to the previous probability mass functions.

```
    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))
```

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),
  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)",
```

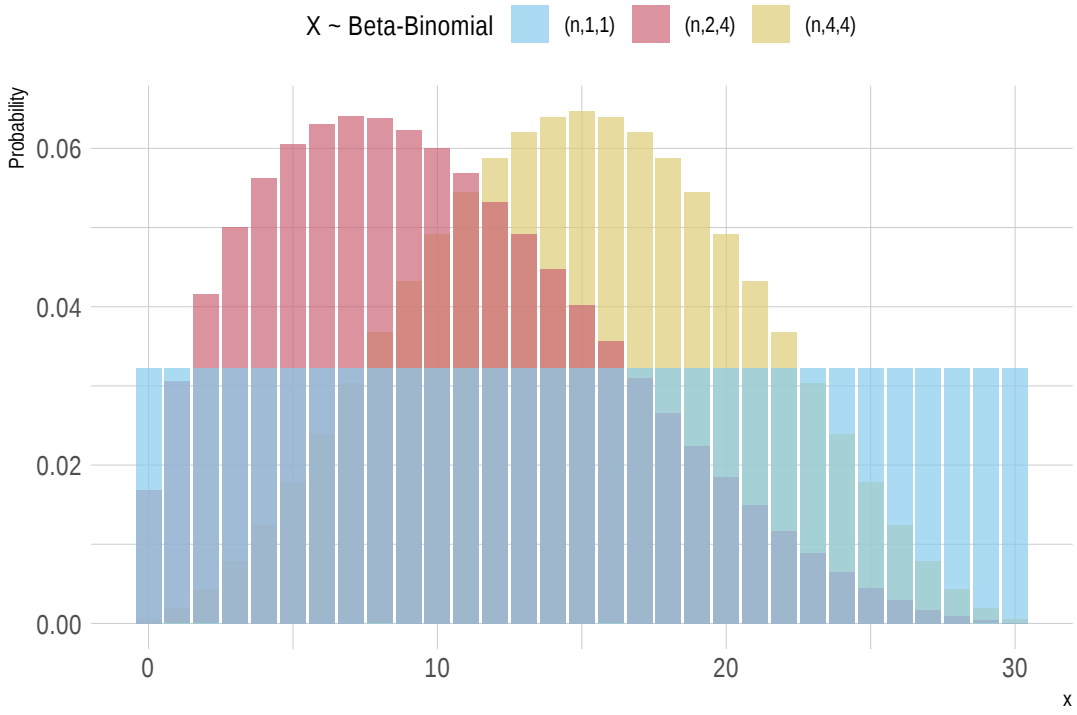


Figure B.18.: Examples of probability mass function of Beta-Binomial distributions.

```
                        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")

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")
```

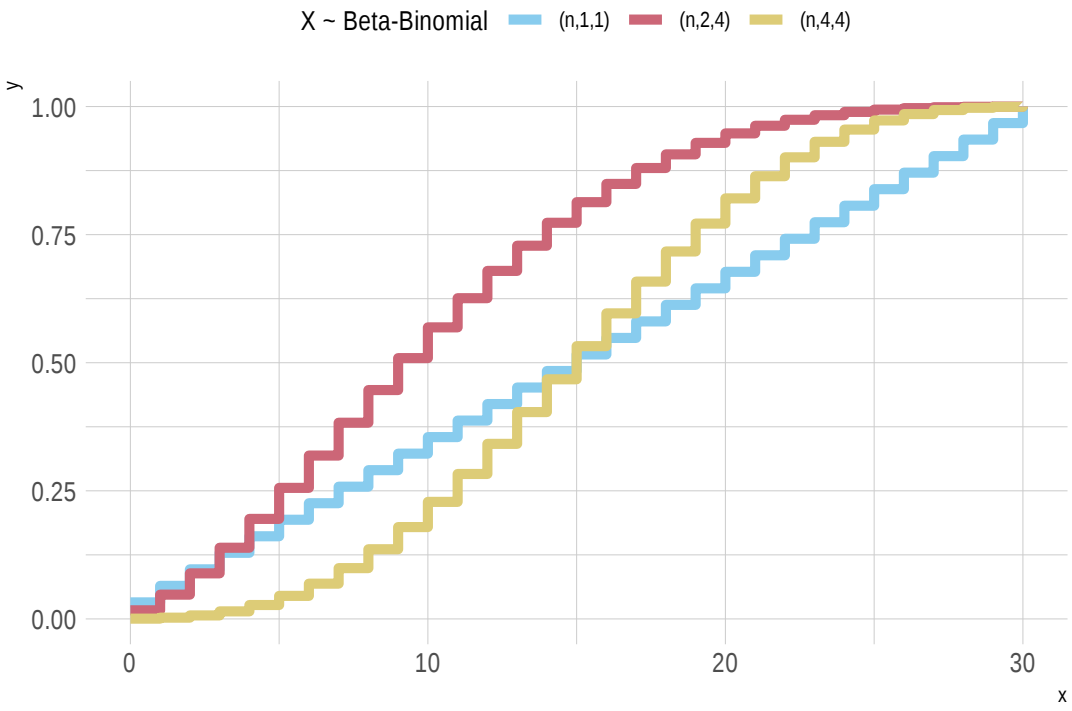


Figure B.19.: Examples of the cumulative distribution function of Beta-Binomial distributions corresponding to the previous probability mass functions.

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

$$F(x) = \begin{cases} 0 & \text{if } x < 0, \\ \binom{n}{x} \frac{B(a+x,b+n-x)}{B(a,b)} {}_3F_2(n,a,b) & \text{if } 0 \leq x < n, \\ 1 & \text{if } x \geq n. \end{cases}$$

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 λ, sometimes also referred to as *intensity*:

$$X \sim Po(\lambda).$$

The parameter λ 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 λ the more symmetrical gets the distribution. In fact, the poisson distribution can be approximated by a normal distribution for a rate paramter ≥ 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(
    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")

# cumdist plot
rv_pois %>%
  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 ~ Poisson", y = "y")
```

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 $Var(X) = \lambda$

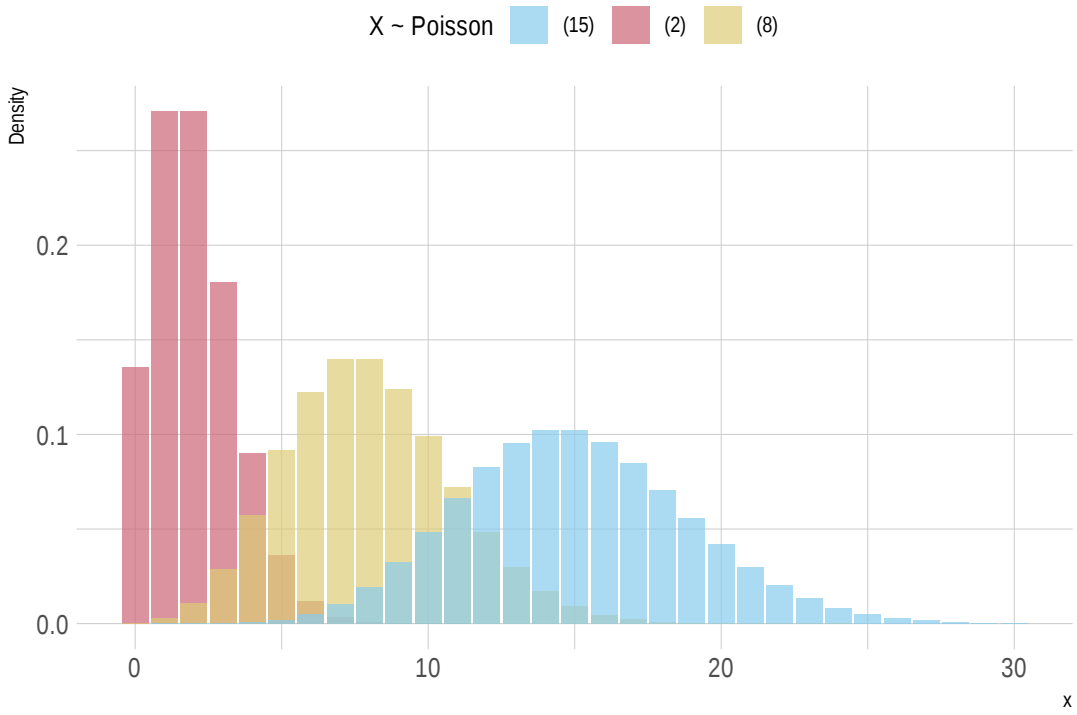


Figure B.20.: Examples of probability mass function of Poisson distributions.

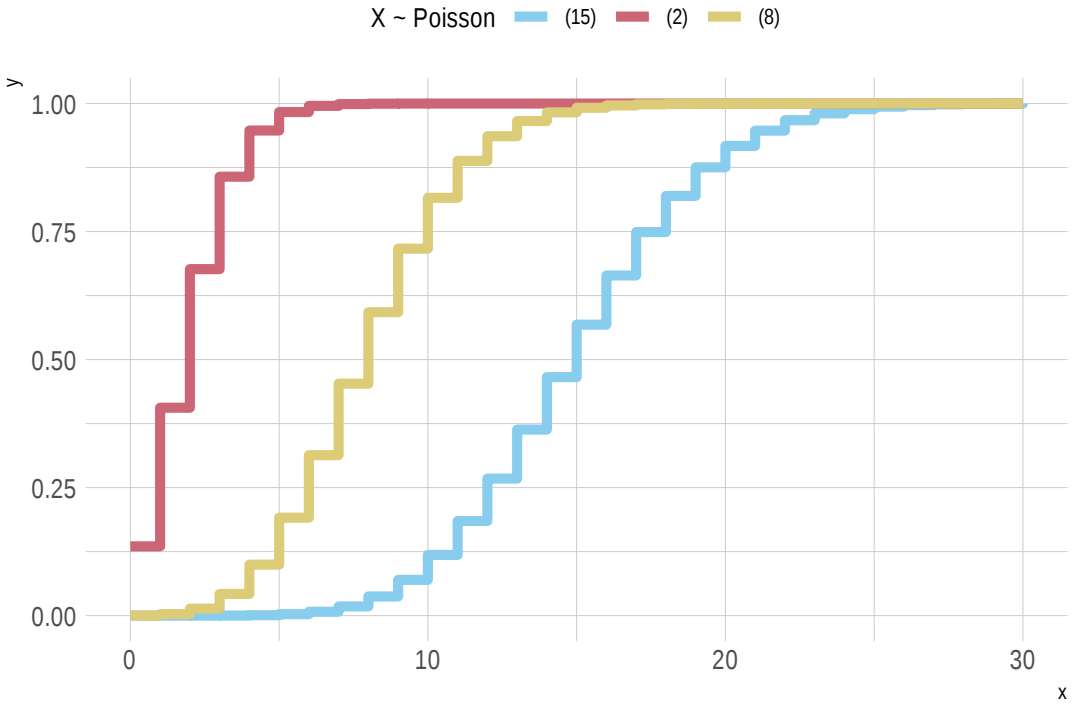


Figure B.21.: Examples of the cumulative distribution function of Poisson distributions corresponding to the previous probability mass functions.

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
                          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_summ"),
               names_to  = "random_variables",
               values_to = "y") %>%
  mutate(
```

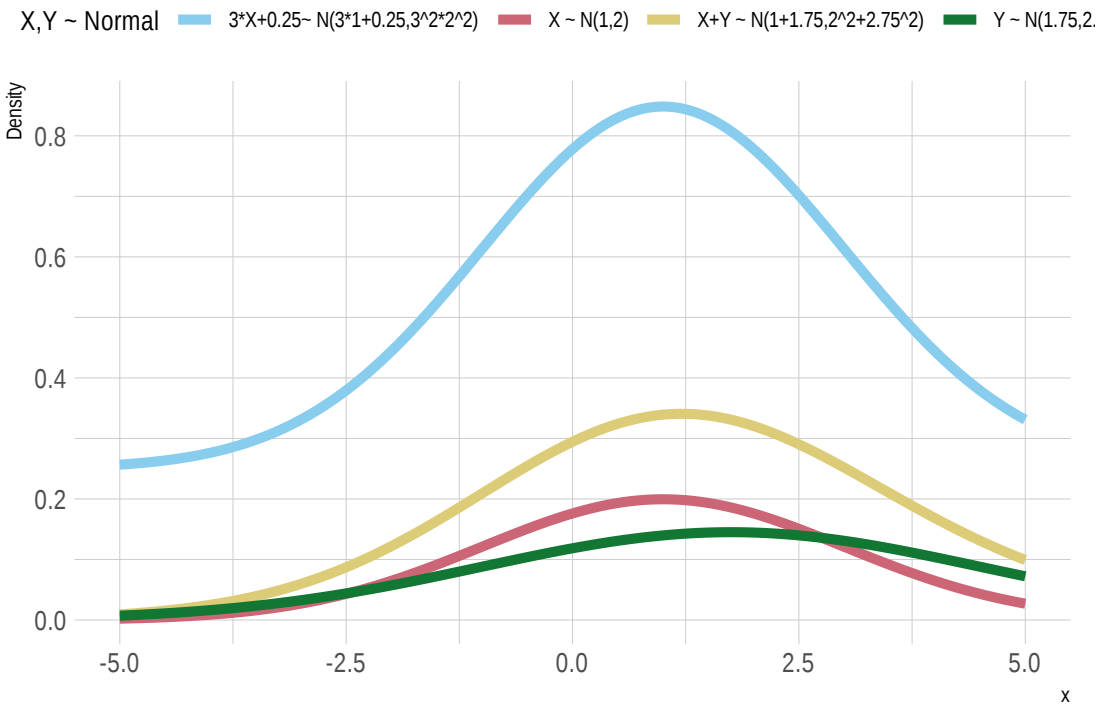


Figure B.22.: Transformation of a normal distributed random variable by addition and linear transformation.

```
    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"),
               names_to  = "random_variables",
               values_to = "y") %>%
  mutate(
```

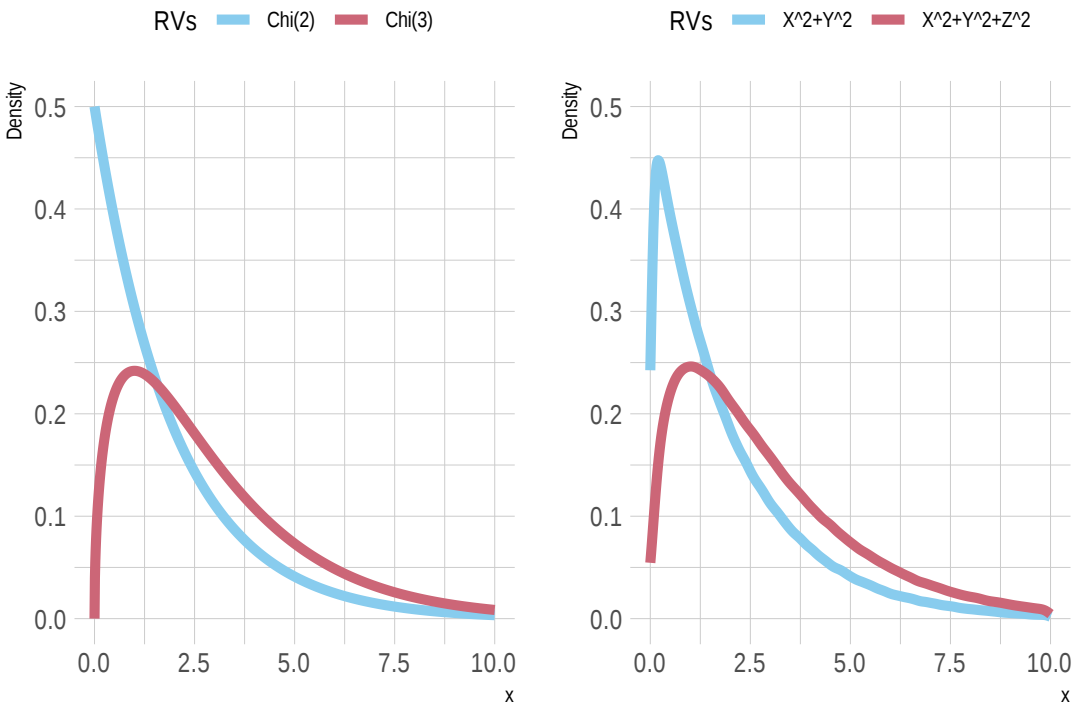


Figure B.23.: Create chi-distributed random variable by summation of standard normal random variables

```
    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

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") %>%
  mutate(
```

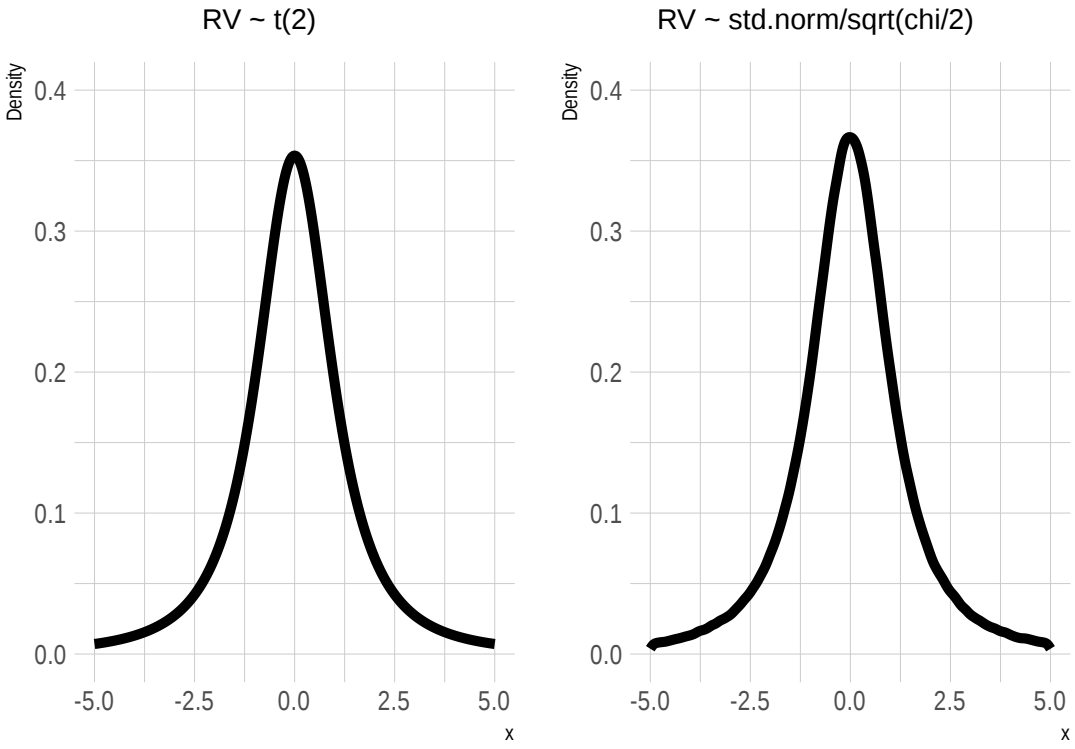


Figure B.24.: Create t-distributed random variable by divison of standard normal RV by chi-squared RV

```
    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)/(r
```

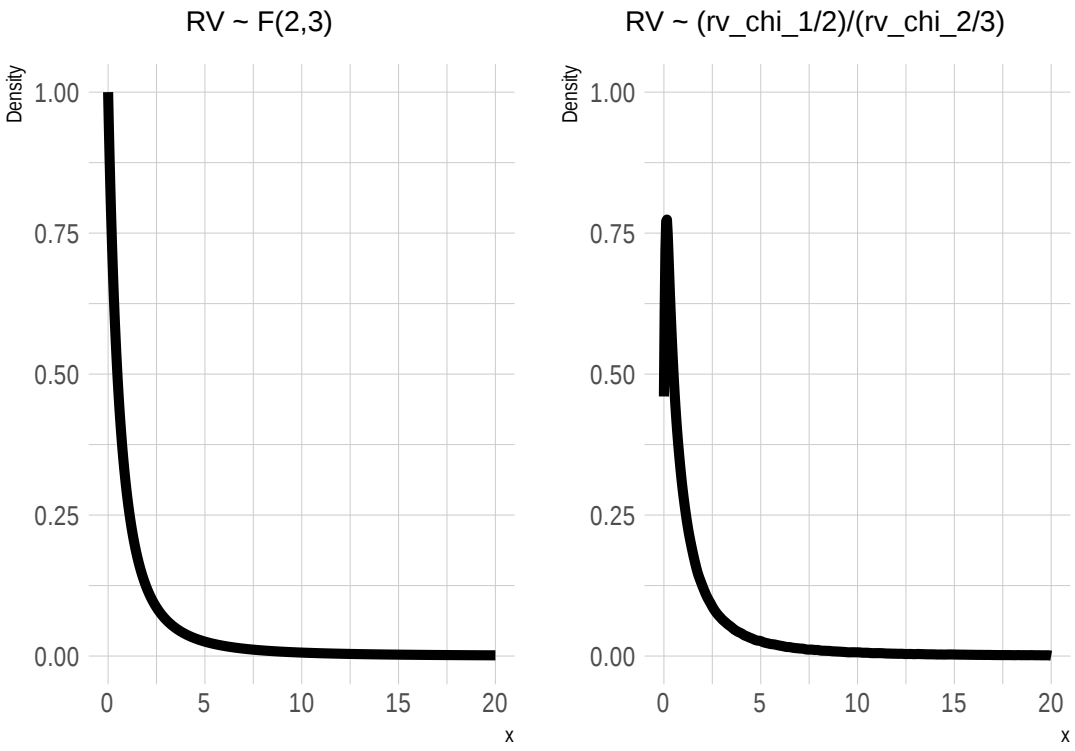


Figure B.25.: Create F-distributed random variable by divison of chi-squared RV by (another independent) chi-squared RV

C. Exponential Family and Maximum Entropy

This chapter deals with the Exponential Family of probability destributions.

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 *conceptional perspective* attractive. Consider for example the following situation:

Consider we want to infer a propability 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.

Consequently, the question arises how *"ignorance" can be measured* and *distributions compared* according to their "information content"? This will be topic of the upcoming exursos, 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 continous 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. Excursos: "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 distrbution 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 perhapse 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 continous 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 $(p_1, ..., p_n)$ 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 $(A_1, ..., A_n)$ (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 $\{p_1, ..., p_m\}$ 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 $n >> m$, 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.

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:

$$p_i = n_i\delta = \frac{n_i}{n}, \text{ with } i = 1, 2, ..., m.$$

The probability that this will happen is the multinomial distribution:

$$m^{-n}\frac{n!}{n_1! \cdot ... \cdot n_m!}.$$

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 $\{p_1, ..., p_m\}$ is accepted.

What is the most likely probability distribution to result from this game? It is the one which maximizes

$$W = \frac{n!}{n_1! \cdot ... \cdot n_m!}$$

subject whatever constraints are imposed by the information I.

We can refine this procedure by using smaller quanta, i.e. large n. By using *Stirlings 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:

$$\frac{1}{n}\log(W) \to -\sum_{i=1}^{m} p_i \log(p_i) = H(p_1, ..., p_m).$$

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 *Langrange multiplier technique*.

C.2.2.1. Lagrangian multiplier technique

Given a mutivariable function $f(x, y, ...)$ and constraints of the form $g(x, y, ...) = c$, 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 λ, 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.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

$$\int_a^b p(x)dx = 1,$$

$$\int_a^b e^{1-\lambda} dx = 1.$$

Solving this equation with respect to λ gives:

$$\lambda = 1 - \log\left(\frac{1}{b-a}\right).$$

Taking both solutions together we get the following probability density function:

$$p(x) = e^{(1-\lambda)} = e^{\left(1-\left(1-\log\left(\frac{1}{b-a}\right)\right)\right)},$$

$$p(x) = \frac{1}{b-a}.$$

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 μ and variance σ^2

Suppose a random variable X with a preassigned standard deviation σ and mean μ. 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 μ and σ are given (whereby only one constrained is needed, as the μ is already included in the definition of σ):

$$\int_{-\infty}^{\infty}(x-\mu)^2 p(x)dx = \sigma^2.$$

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 $p(x)$:

$$\frac{\partial \mathcal{L}}{\partial p(x)} = 0,$$
$$-(1+\log p(x)) + \lambda_0 + \lambda_1(x-\mu)^2 = 0,$$

$$p(x) = e^{\lambda_0+\lambda_1(x-\mu)^2-1}.$$

Further we have to make sure that the result holds for the stated constraints:

$$\int_{-\infty}^{\infty} e^{\lambda_0+\lambda_1(x-\mu)^2-1} - 1dx = 1,$$

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

$$\lambda_1 = \frac{-1}{2\sigma^2}$$

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 λ_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

3. **Discrimination task**

   The participant presses the **F** key on the keybord 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-chrono-data_raw.csv')
glimpse(d_raw)
```

```
## Observations: 3,750
## Variables: 32
## $ submission_id <dbl> 8554, 8554, 8554, 8554, 8554, 8554, 8554, 8554, ...
## $ QUD           <chr> "Press SPACE when you see a shape on the screen"...
## $ RT            <dbl> 376, 311, 329, 270, 284, 311, 269, 317, 325, 240...
## $ age           <dbl> 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, ...
## $ comments      <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ...
## $ correctness   <chr> "correct", "correct", "correct", "correct", "cor...
## $ education     <chr> "high school / college", "high school / college"...
## $ elemSize      <dbl> 100, 100, 100, 100, 100, 100, 100, 100, 100, 100...
## $ endTime       <dbl> 1.570374e+12, 1.570374e+12, 1.570374e+12, 1.5703...
## $ expected      <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ...
## $ experiment_id <dbl> 68, 68, 68, 68, 68, 68, 68, 68, 68, 68, 68, 68, ...
## $ f             <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ...
## $ focalColor    <chr> "blue", "blue", "blue", "blue", "blue", "blue", ...
## $ focalNumber   <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## $ focalShape    <chr> "square", "square", "circle", "square", "circle"...
```

```
## $ gender        <chr> "female", "female", "female", "female", "female"...
## $ j             <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ...
## $ key1          <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ...
## $ key2          <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ...
## $ key_pressed   <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ...
## $ languages     <chr> "Right", "Right", "Right", "Right", "Right", "Ri...
## $ pause         <dbl> 2631, 1700, 1322, 1787, 1295, 2330, 1620, 2460, ...
## $ response      <chr> "space", "space", "space", "space", "space", "sp...
## $ sort          <chr> "grid", "grid", "grid", "grid", "grid", "grid", ...
## $ startDate     <chr> "Sun Oct 06 2019 15:45:19 GMT+0100 (Hora de verã...
## $ startTime     <dbl> 1.570373e+12, 1.570373e+12, 1.570373e+12, 1.5703...
## $ stimulus      <chr> "square", "square", "circle", "square", "circle"...
## $ target        <chr> "square", "square", "circle", "square", "circle"...
## $ timeSpent     <dbl> 7.2514, 7.2514, 7.2514, 7.2514, 7.2514, 7.2514, ...
## $ total         <dbl> 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...
## $ trial_type    <chr> "reaction_practice", "reaction_practice", "react...
```

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, e

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,
    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:

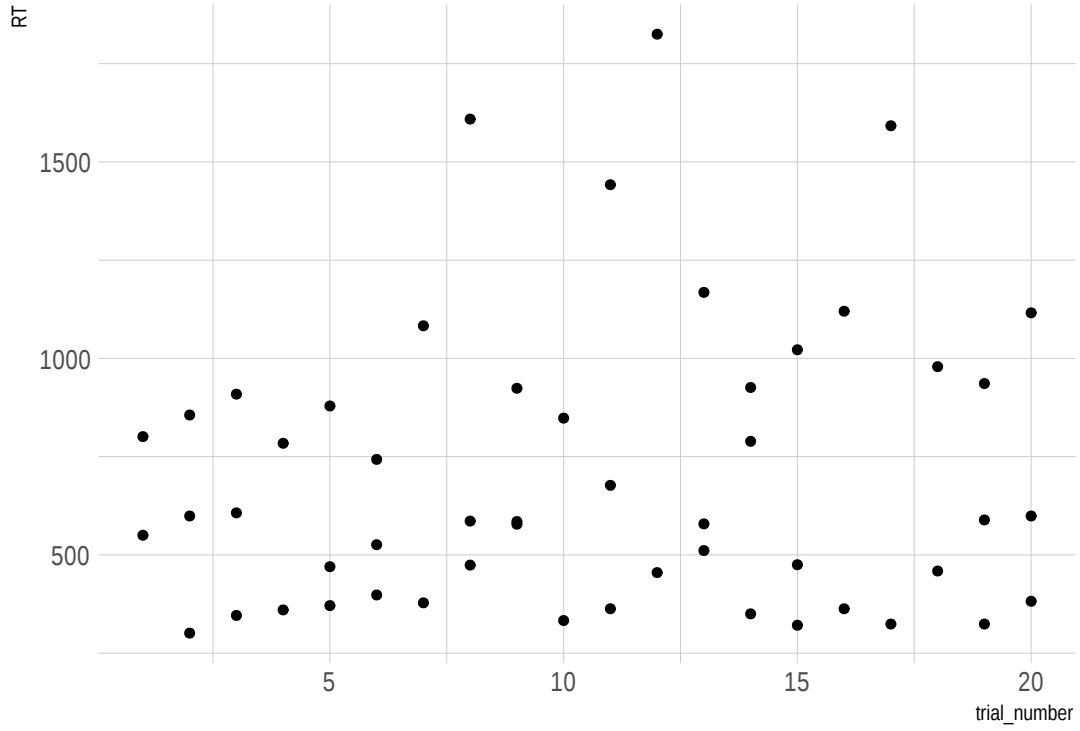
```
# summary stats (means) for participants
d_sum_stats_participants <- d_preprocessed %>%
  group_by(submission_id, block) %>%
```

```
  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")
  ) %>%
  ggplot(aes(x = trial_number, y = RT)) +
  geom_point()
```



We are then going to exclude this participant's entire data from all subsequent analysis:[1]

```
d_cleaned <- d_preprocessed %>%
  filter(submission_id != d_sum_stats_participants$submission_id[1] )
```

D.1.3.2. Cleaning by-trial

Our rule for exclusing data from individual trials is:

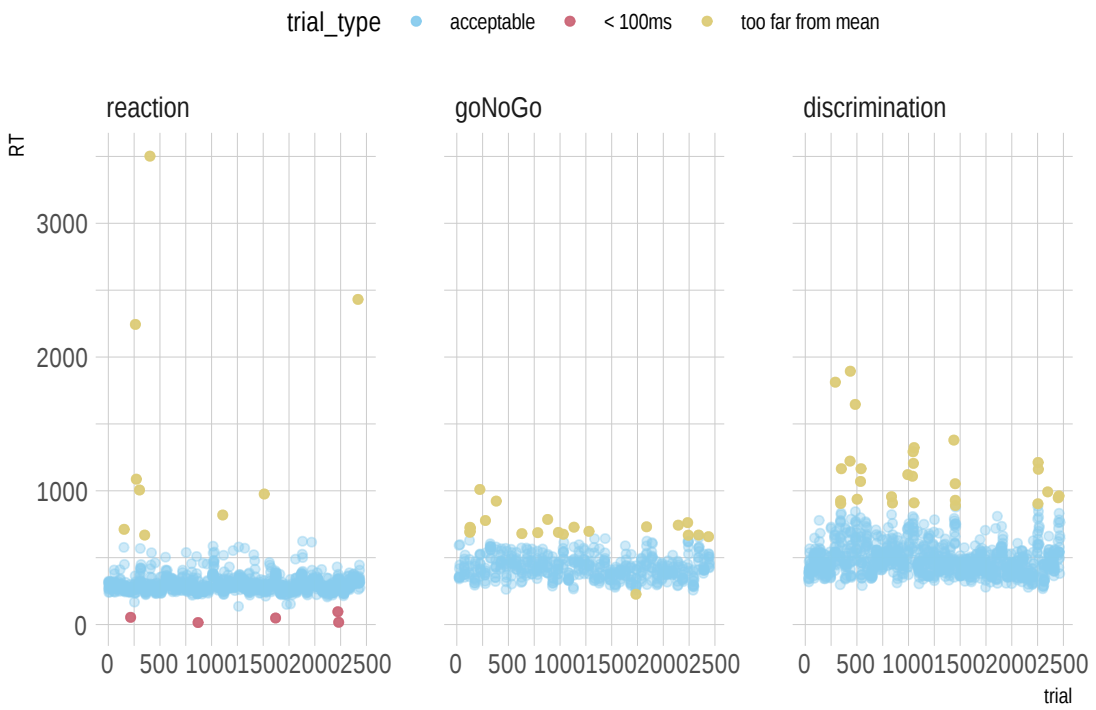
> From the remaining data, we then remove any individual trial Y if the RT of Y is more than 2 standard deviations away from the mean of experimental condition C (where C is the condition of Y, of course). We also remove all trials with reaction times below 100ms.

The following code implements this:

```
# mark individual trials as outliers
d_cleaned <- d_cleaned %>%
  full_join(
    d_sum_stats_conditions,
```

[1] 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.

```
    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"))
```



So, we remove 63 individual trials.

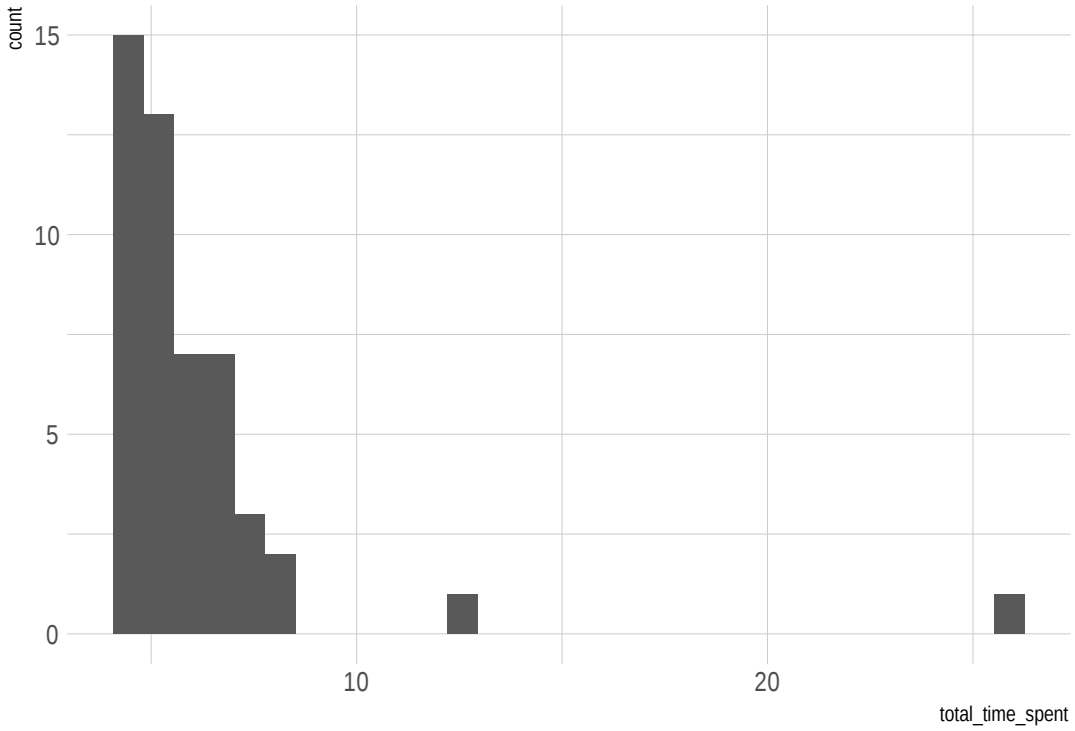
```
d_cleaned <- d_cleaned %>%
  filter(trial_type == "acceptable")
```

```
## 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 noticably 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).

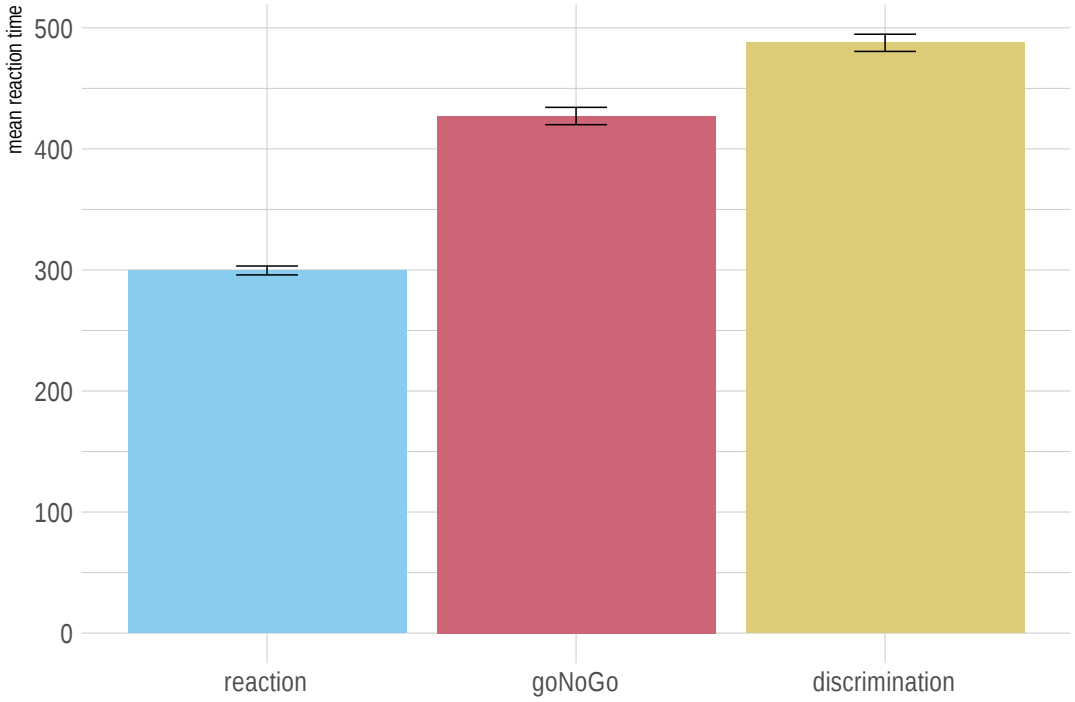
```
d_sum_stats_blocks_cleaned <- d_cleaned %>%
  group_by(block) %>%
```

```
  nest() %>%
  summarise(
    CIs = map(data, function(d) bootstrapped_CI(d$RT))
  ) %>%
  unnest(CIs)

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  481.  488.  495.
```

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 intercets 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

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 outsie 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.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 where 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...
## $ RT              <dbl> 1239, 938, 744, 528, 706, 547, 591, 652, 627, ...
## $ condition       <chr> "incongruent", "incongruent", "incongruent", "...
## $ correctness     <chr> "correct", "correct", "correct", "correct", "c...
## $ class           <chr> "Intro Cogn. Neuro-Psychology", "Intro Cogn. N...
## $ experiment_id   <dbl> 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52, 52...
## $ key_pressed     <chr> "q", "q", "q", "q", "p", "p", "q", "p", "q", "...
## $ p               <chr> "circle", "circle", "circle", "circle", "circl...
## $ pause           <dbl> 1896, 1289, 1705, 2115, 2446, 2289, 2057, 2513...
## $ q               <chr> "square", "square", "square", "square", "squar...
## $ target_object   <chr> "square", "square", "square", "square", "circl...
## $ target_position <chr> "right", "right", "right", "right", "left", "r...
## $ timeSpent       <dbl> 7.565417, 7.565417, 7.565417, 7.565417, 7.5654...
```

```
## $ trial_number    <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,...
## $ trial_type      <chr> "practice", "practice", "practice", "practice"...
```

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.

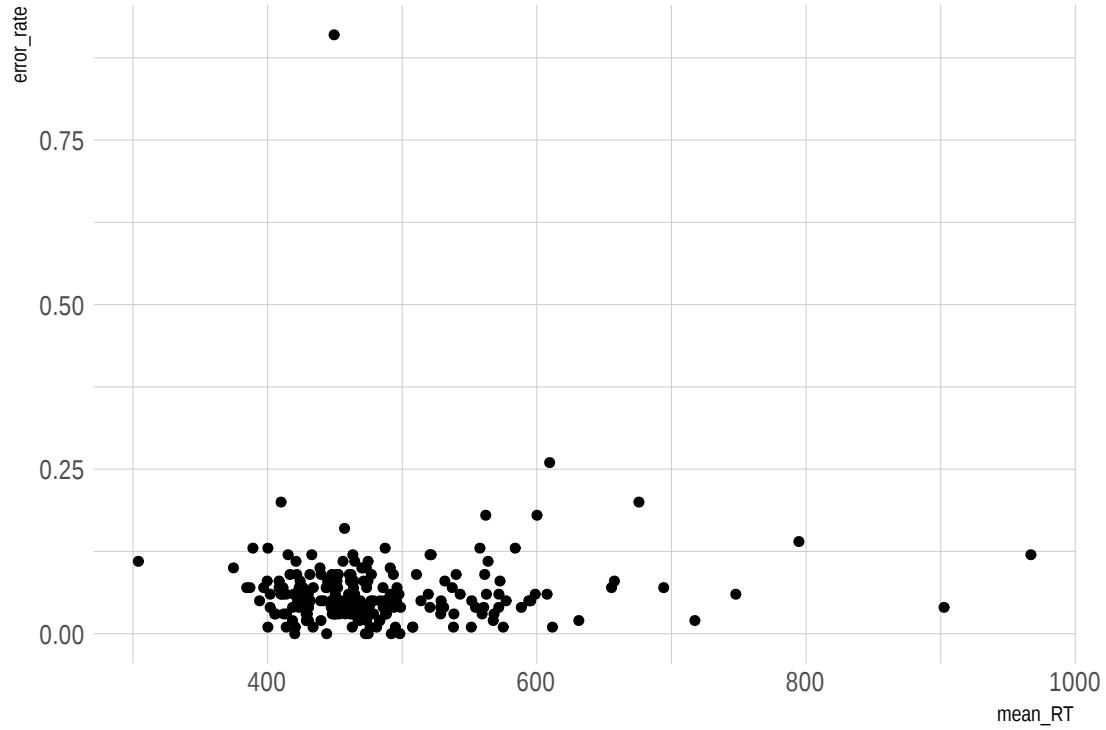
```
d_individual_summary <- d %>%
  filter(trial_type == "main") %>%    # look at only data from main trials
  group_by(submission_id) %>%         # calculate the following for each individual
  summarize(mean_RT = mean(RT),
            error_rate = 1 - mean(ifelse(correctness == "correct", 1, 0)))
head(d_individual_summary)
```

```
## # 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
```

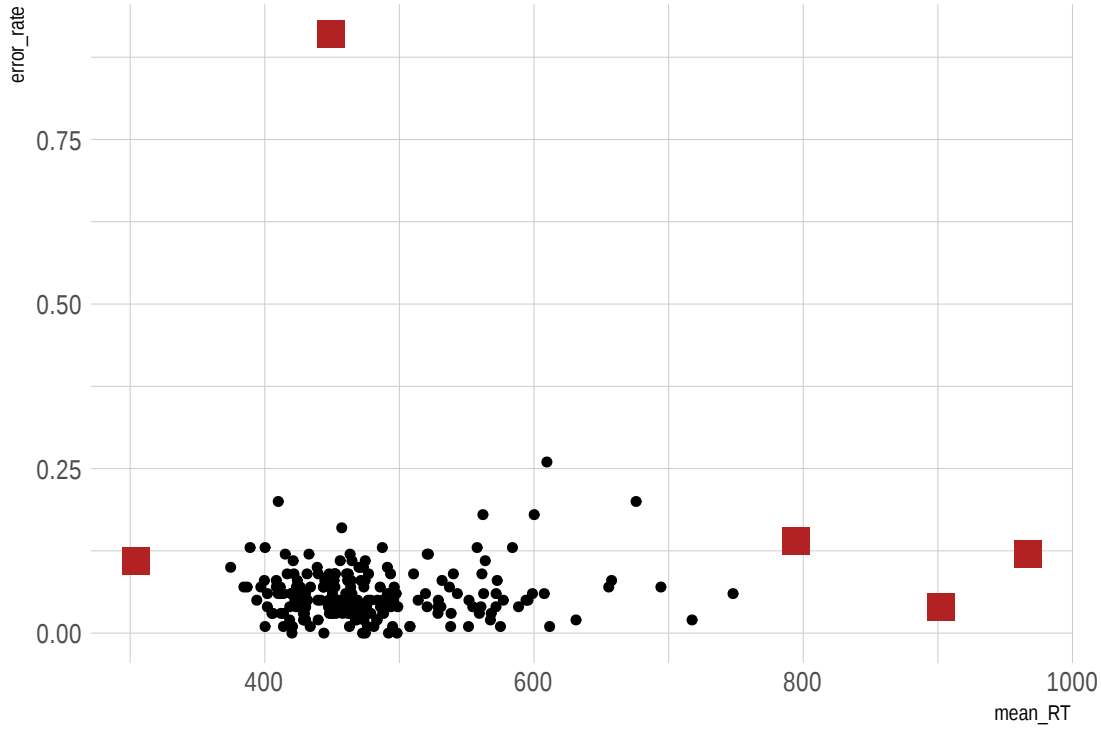
Let's plot this summary information:

```
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)
```



We then clean the data set in a first step by removing all participants identified as outlier-y:

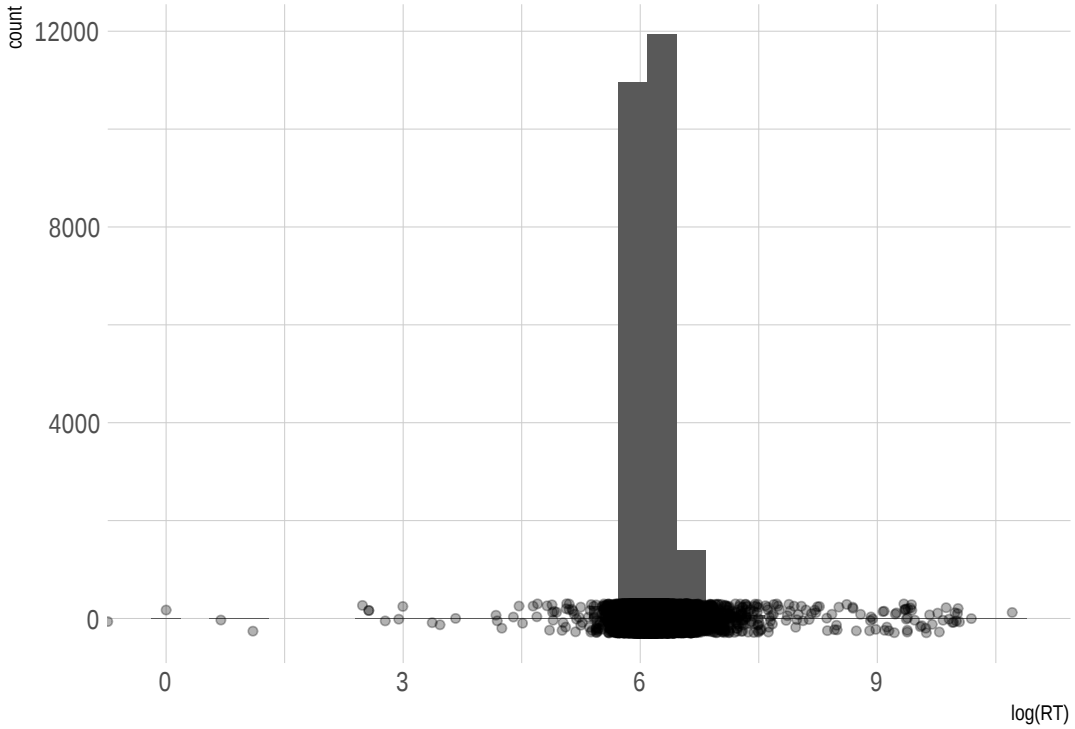
```
d <- full_join(d, d_individual_summary, by = "submission_id") # merge the tibbles
d <- filter(d, outlier == FALSE)
message("We excluded ", sum(d_individual_summary$outlier) , " participants for suspicious mean
```

```
## We excluded 5 participants for suspicious mean RTs and higher error rates.
```

D.2.2.2.2. Trial-level reaction times

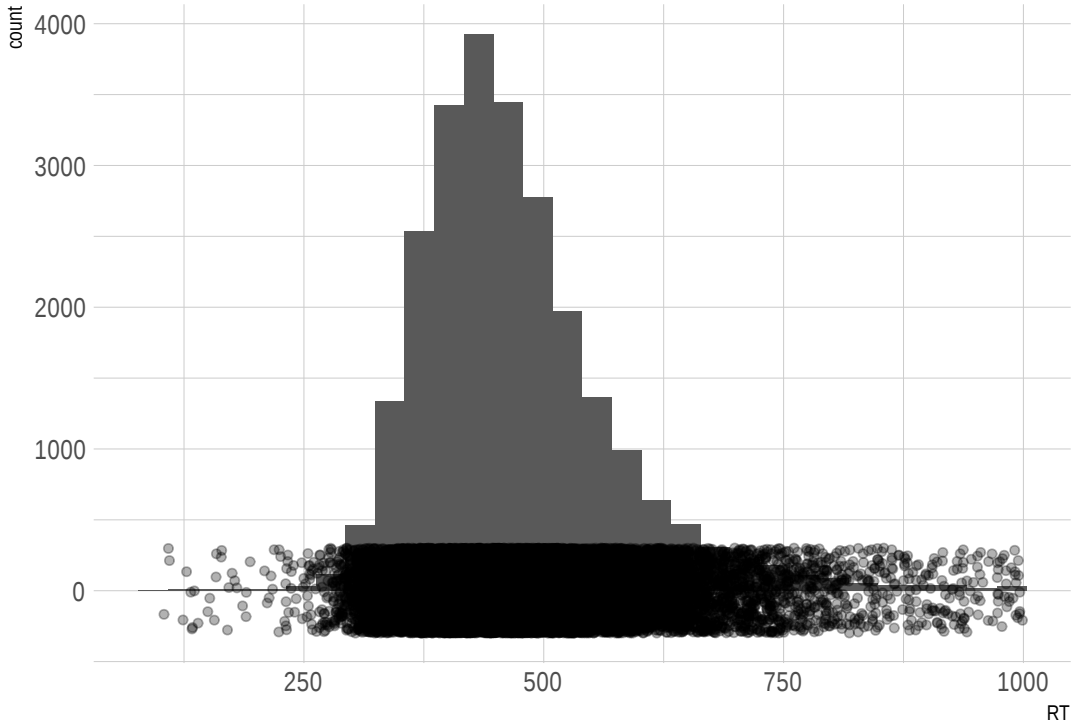
It is also conceivable that inidividual trials resulted in early accidental key presses or were interrupted in some way or another. We therefore look at the overall distribution of RTs and determine (similarly arbitrarily, but once again this should be planned in advance) what to exclude.

```
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.2.2.3. Exploring the (main) data

We are mostly interested in the influence of congruency on the reactions 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.

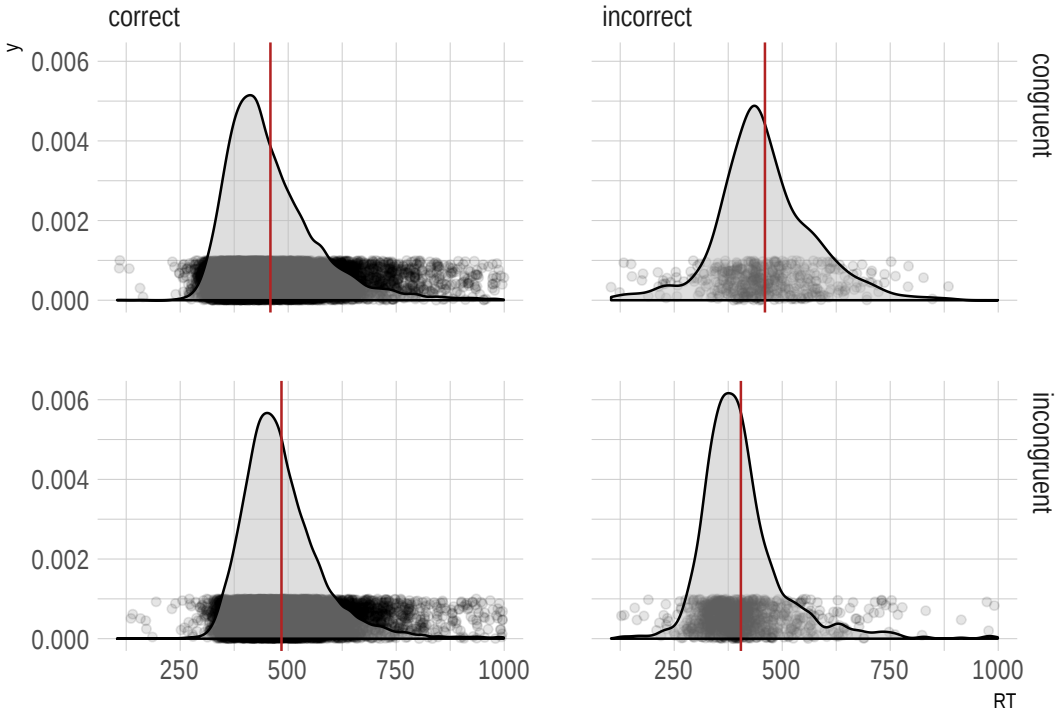
Here is a summary of the means and standard deviations for each condition:

```
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 conguency. In the simplest case we would therefore do this:

```
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 pressupposes, 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 Szendröi (2013) investigating how sentences with false presuppositions are perceived. The main question of 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

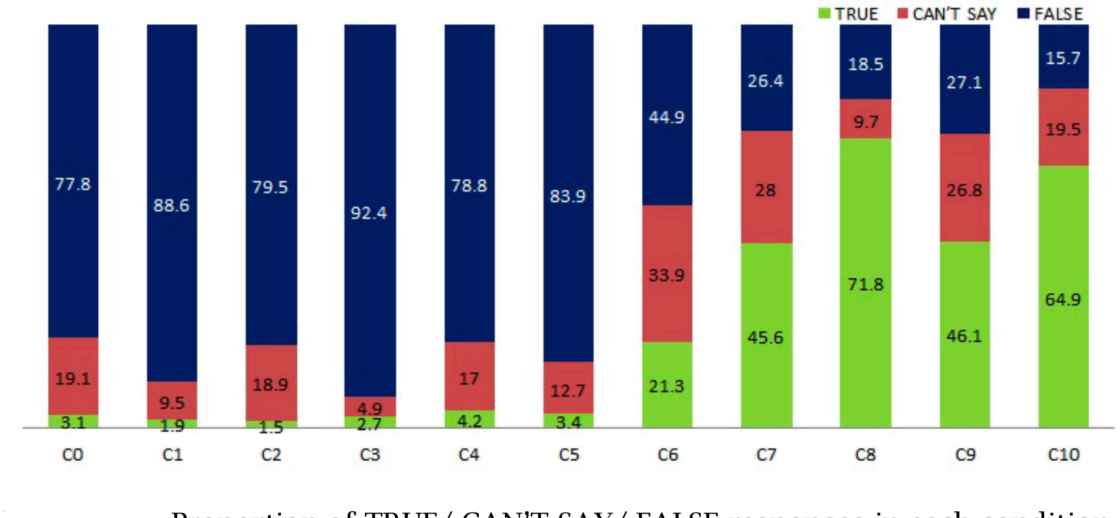


Figure D.1.: Results of @AbrusanSzendroi2013:Experimenting-w .

Abrusán and Szendröi (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 Szendröi (2013) presented their participants with 11 different types of sentences, of which we here only focus on five. Here are examples of the five coniditions we test, using the corresponding condition numbers from the experiment by Abrusán and Szendröi (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 pressupposed (C1) or explicitly asserted (C1). 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 Szendröi (2013).

D.4.1.1. The experiment

D.4.1.1.1. Participants

We obtained data from 97 particpants via the online crowd-sourcing platform Prolific.[2] 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.

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 uncontroversilly 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.

[2] We recruited 100 participants, but the data from three participants was not recorded due to technical problems.

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:[3]

1. Is the overal 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 and glimpse at the data:

```
d_raw <- read_csv('data_sets/king-of-france_data_raw.csv')
glimpse(d_raw)
```

```
## Observations: 2,813
## Variables: 16
## $ submission_id  <dbl> 192, 192, 192, 192, 192, 192, 192, 192, 192, 19...
```

[3] These research questions are a compromise between actual theoretical relevance and practical (= educational) considerations.

```
## $ RT             <dbl> 8110, 35557, 3647, 16037, 11816, 6024, 4986, 13...
## $ age            <dbl> 57, 57, 57, 57, 57, 57, 57, 57, 57, 57, 57, 57,...
## $ comments       <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,...
## $ item_version   <chr> "none", "none", "none", "none", "none", "none",...
## $ correct_answer <lgl> FALSE, TRUE, FALSE, TRUE, TRUE, TRUE, FALSE, FA...
## $ education      <chr> "Graduated College", "Graduated College", "Grad...
## $ gender         <chr> "female", "female", "female", "female", "female...
## $ languages      <chr> "English", "English", "English", "English", "En...
## $ question       <chr> "World War II was a global war that lasted from...
## $ response       <lgl> FALSE, TRUE, FALSE, TRUE, TRUE, TRUE, FALSE, FA...
## $ timeSpent      <dbl> 39.48995, 39.48995, 39.48995, 39.48995, 39.4899...
## $ trial_name     <chr> "practice_trials", "practice_trials", "practice...
## $ trial_number   <dbl> 1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 1...
## $ trial_type     <chr> "practice", "practice", "practice", "practice",...
## $ vignette       <chr> "undefined", "undefined", "undefined", "undefin...
```

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"
- `item_version`: the current item's vignette number (applies only to trials of type `main` and `special`)
- `response`: the answer ("TRUE" or "FALSE") on each trial

As the variable names used in the raw data are not ideal, we will pre-process the raw data a bit for easier analysis.

```
d_processed <-  d_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", "fil
      )
  )
# write_csv(d_processed, "data_sets/king-of-france_data_processed.csv")
```

D.4.3. Cleaning the data

We clean the data in two consecutive steps:

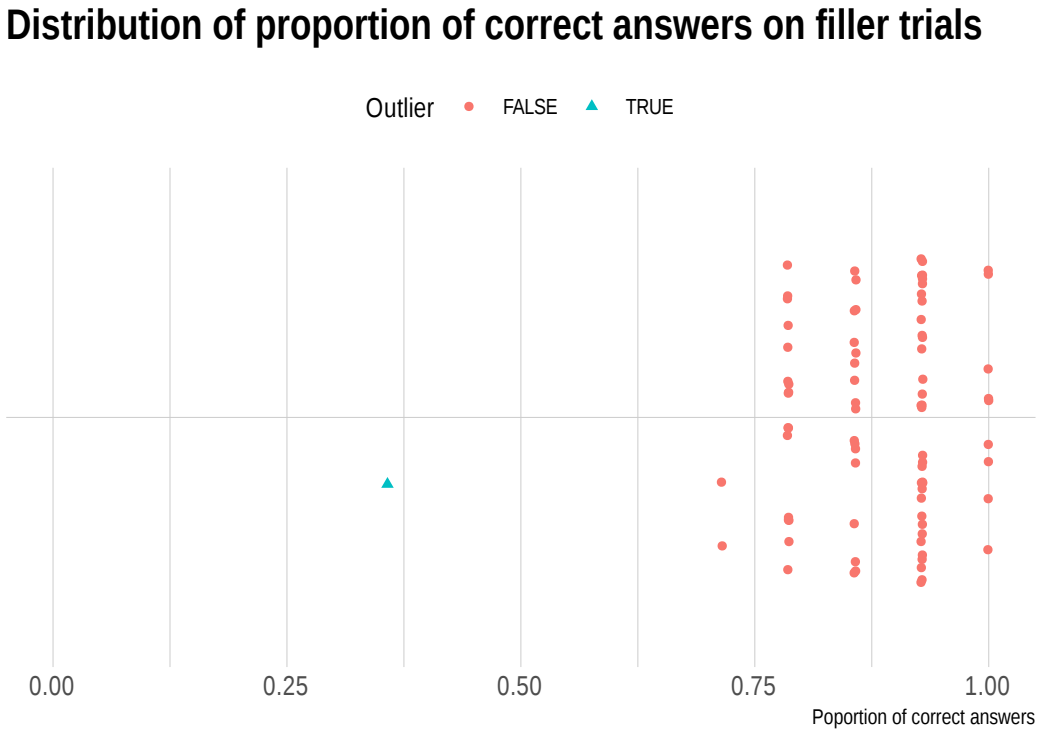
1. Remove all data from any participant who got more than 50% of the answer to filler material wrong.
2. Remove individual main trials if the corresponding "background check" question was answered wrongly.

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 <- d_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")
```



Apply the cleaning step:

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

D.4.3.2. Cleaning by-trial

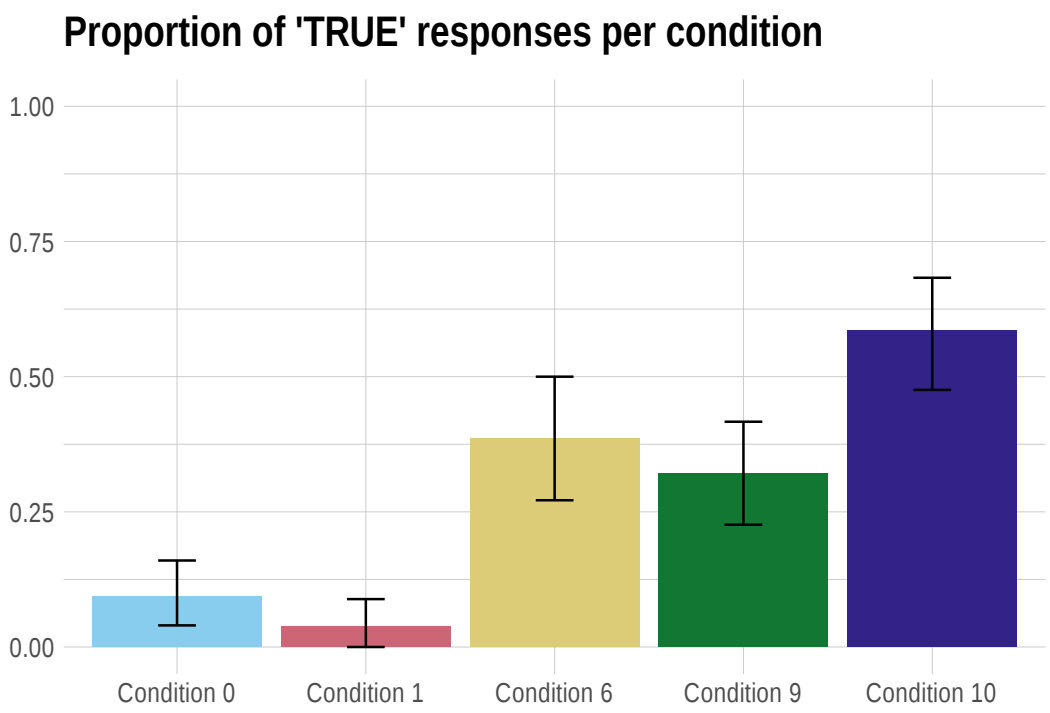
```
# exclude every critical trial whose 'background' test question was answered wrongl

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 check
  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

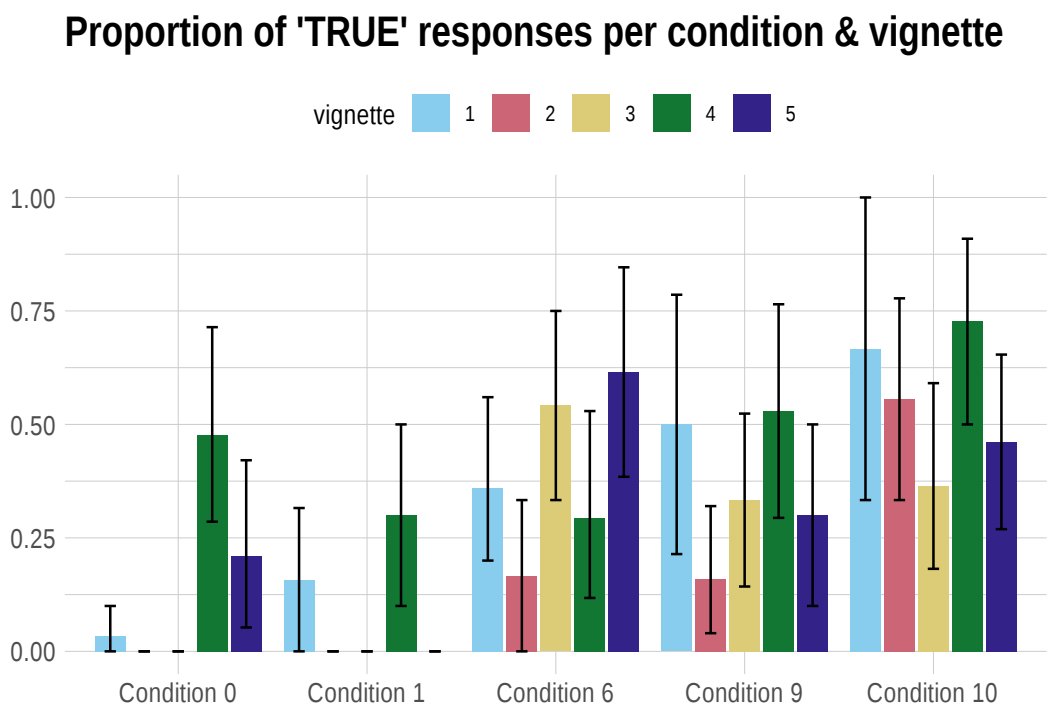
Plot for ratings by condition:

```
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)
```



Plot for each condition & vignette:

```
d_processed %>%
  filter(trial_type == "main") %>%
  droplevels() %>%
  group_by(condition, vignette) %>%
  nest() %>%
  summarise(
    CIs = map(data, function(d) bootstrapped_CI(d$response == "TRUE"))
  ) %>%
  unnest(CIs) %>%
  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 & vig
```



D.4.5. Data analysis

Abrusán, Márta, and Kriszta Szendröi. 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 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.

Vallverdú, Jordi. 2016. *Bayesians Versus Frequentists: A Philosophical Debate on Statistical Reasoning*. Heidelberg: Springer.

Wickham, Hadley. 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.