Statistical Rethinking (2nd ed. 2023)

Peter Baumgartner

2023-07-31 08:16

Table of contents

Preface

Finished about 15%

Content and Goals of this Book

This book collects personal notes during reading of Statistical Rethinking by Richard McElreath. I am using the second edition published 2020 by CRC Press an imprint of Routledge of the Taylor & Francis Group. Additionally I am using Statistical Rethinking 2023, the most recent set of free YouTube video lectures.

You can find links to other material on Richards website about the book. Of special interest for me are the brms+tidyverse and the Stan+tidyverse conversion of his code. As I am not very experienced with R and completely new to Bayesian statistics and their tools this additional material is for me also very challenging. I am planning to read them simultaneously and will dedicate parallel chapters for their approaches. Chapters with the

- suffix `a` refers to the original book,
- suffix `b` to the brms conversion and
- suffix `c` to the Stan conversion.

Caution (2023-17-07)

I am in the process of re-organizing the content I have written so far (chapter 1-4). Instead of separating the original from the brms-version I am going to integrate the content of different code conversions into one chapter. This has the advantage that the chapter numbers of this document conform to the chapter numbers of the second edition of the printed book.
This will take some time. In the meanwhile keep in mind that there is a mixture of old and new structure and that some cross references do not work.

My text consists mostly of quotes from the 2020 book edition or from the text of the video lectures 2023. Sometimes I made minor editing (e.g., shorting the text), but almost all of my text of this Quarto book are not mine, but is coming from the book or video lectures by

Richard McElreath. Therefore I am not going to indicate these quotes and I am also not using exact text snippets for my replications.

I am not only replicating the code to see how it works but also to change the values of parameters to observe their influences. Sometimes I am changing the code myself to tidyverse conventions. Especially when it comes to plotting I try use ggplot2 instead the base plotting system I have no experience at all.

As this is my first book using Quarto instead of bookdown I am using these notes also to learn Quarto. Therefore you will find sometimes remarks to my Quarto experiences.

> **Warning**
>
> I wrote this book as a text for others to read because that forces me to be become explicit and explain all my learning outcomes more carefully. Please keep in mind that this text is not written by an expert but by a learner. In spite of replicating most of the content it may contain many mistakes. All these misapprehensions and errors are my responsibility.

If you find errors please do not hesitate to write issues or PRs on my GitHub site. I really appreciate it to learn from more experienced R users! It shortens the learning paths of self-directed learners.

Package Installation

In contrast to the sparse and partly outdated remarks in the book use the installation section from the `rethinking` package at GitHub.

Step 1

From the three steps I had already successfully installed the first one (`rstan` and the `C++` toolchain), so I had no need to follow the detailed instructions of the `rstan` installation at https://mc-stan.org/users/interfaces/rstan.html.

Step 2

To install the `cmdstanr` package I visited https://mc-stan.org/cmdstanr/. This is an addition to my previous installation with the older version (2nd ed., 2022). As I installed the latest beta version of `cmdstanr` the first time I also needed to compile the libraries with `cmdstanr::install_cmdstan()`.

To check the result of my installation I ran `check_cmdstan_toolchain()`.

~~~
```{r}
#| label: install-cmdstanr
#| eval: false

install.packages("cmdstanr", repos = c("https://mc-stan.org/r-packages/", getOption("repos
cmdstanr::install_cmdstan()
cmdstanr::check_cmdstan_toolchain()

```
~~~

The command for downloaded **cmdstanr** did not install the vignettes, which take a long time to build, but they are always available online at https://mc-stan.org/cmdstanr/articles/.

The vignette Getting started with CmdStanR also recommend to load the `bayesplot` and `posterior` packages, which are used later in the `CmdStanR`-examples. But I believe these two packages are not necessary if you just plan to stick with the book.

Step 3

Once the infrastructure is installed one can install the packages used by the book. With the exception of rethinking — the companion package of the book – they can all be downloaded from CRAN.

I had already devtools installed, therefore I deleted it from the list of installed packages.

~~~
```{r}
#| label: install-packages
#| eval: false

install.packages(c("coda","mvtnorm", "loo","dagitty","shape"))
devtools::install_github("rmcelreath/rethinking")

```
~~~

Get Code Examples

Go to the book website and download the R code examples for the book.

~~~
```{r}
#| label: download-code-examples
~~~

~~~
#| eval: false

dir.create("R")
download.file("http://xcelab.net/rmpubs/sr2/code.txt", "R/code.R")

```
~~~

🔥 Caution

There are big differences between the code snippets of the 2nd edition collected in code.txt and the new version preparing the 3rd book version. These new code snippets can be found in the slides and/or in the videos. I will always refer to the place where they can be found.
Additionally you will find all the scripts supporting the animation in the lectures at the new 2023 github repo.

The style of the code snippets is not the tidyverse style. For instance: The equal sign `=` is not embedded between spaces but a list of variables, separated by comas has in front and before the coma a space.

~~~
```{r}
#| label: code-style-orig
#| eval: false

sample <- c("W","L","W","W","W","L","W","L","W")
W <- sum(sample=="W") # number of W observed
L <- sum(sample=="L") # number of L observed
p <- c(0,0.25,0.5,0.75,1) # proportions W
ways <- sapply( p , function(q) (q*4)^W * ((1-q)*4)^L )
prob <- ways/sum(ways)
cbind( p , ways , prob )

```
~~~

I will always convert the original code style with the RStudio addin styler package to tidyverse style: Assuming that the default value of the style transformer is `styler::tidyverse_style()` I select the code snippet I want to convert and call the addin which runs `styler:::style_selection()`. The transformation of the above code snippet results into the code below:

~~~
```{r}
#| label: code-style-new
#| eval: false

sample <- c("W", "L", "W", "W", "W", "L", "W", "L", "W")
W <- sum(sample == "W") # number of W observed
L <- sum(sample == "L") # number of L observed
p <- c(0, 0.25, 0.5, 0.75, 1) # proportions W
ways <- sapply(p, function(q) (q * 4)^W * ((1 - q) * 4)^L)
prob <- ways / sum(ways)
cbind(p, ways, prob)
```
~~~

As copy & paste from the slides does not work I downloaded the PDF of the Speaker deck slides. But still, it didn't work always. In that case I used TextSniper and formated manually. But these copy & paste problems only arise when using new code, prepared for the 3rd edition. With the book (2nd ed.) I do not have problems to copy the code snippets via calibre with the ePUB eBook version.

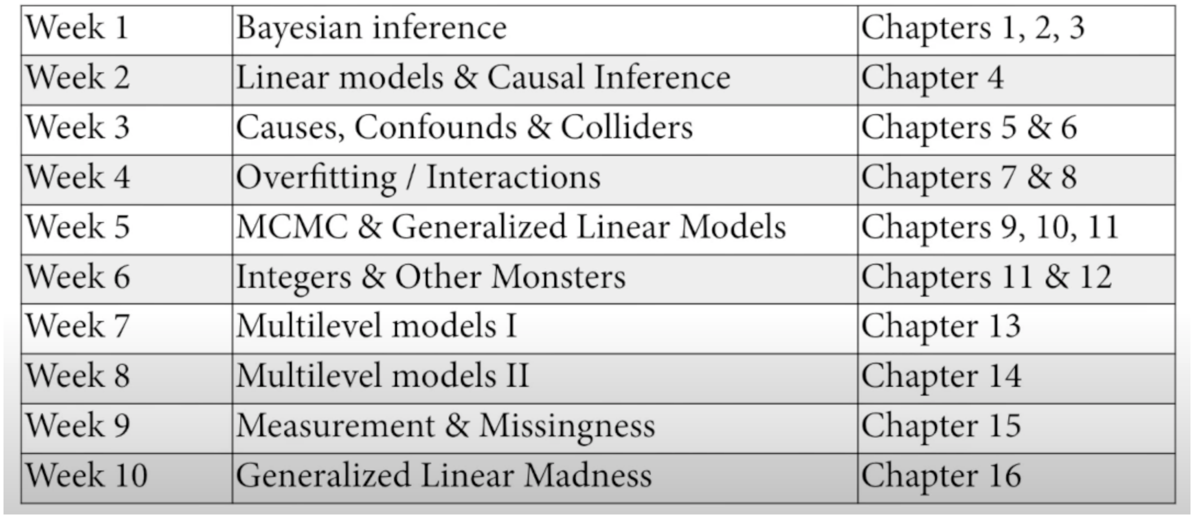
Setup Chunk

In this text you will find very long chapters resulting in long files with many code chunks. To secure that I can run individual chunks for test purposes without rendering the whole file I will collect the `library()` commands of all necessary packages in the setup chunk for each file (See Quarto equivalent to RMarkdown setup chunk).

I am also using the {**conflicted**} package and provide the preferable commands with `conflicts_prefer()` in each setup file.

Course Schedule

The following tables matches the lectures (videos 2023 and slides 2023) with the book chapters of the second edition (2020). It was generated by a screenshot from **Statistical Rethinking 2023 - 01 - The Golem of Prague** (50:09), but can also be found as a slide in **Statistical Rethinking 2023 - Lecture 01**.



Week 1	Bayesian inference	Chapters 1, 2, 3
Week 2	Linear models & Causal Inference	Chapter 4
Week 3	Causes, Confounds & Colliders	Chapters 5 & 6
Week 4	Overfitting / Interactions	Chapters 7 & 8
Week 5	MCMC & Generalized Linear Models	Chapters 9, 10, 11
Week 6	Integers & Other Monsters	Chapters 11 & 12
Week 7	Multilevel models I	Chapter 13
Week 8	Multilevel models II	Chapter 14
Week 9	Measurement & Missingness	Chapter 15
Week 10	Generalized Linear Madness	Chapter 16

A better overview with links to videos and slides provides the following HTML table, taken from the README.md file for the 2023 lectures.

Week ##	Meeting date	Reading	Lectures
Week 01	06 January	Chapters 1, 2 and 3	[1] <Golem of Prague> <Slides> [2] <Garden of Forking Data> <Slides>
Week 02	13 January	Chapter 4	[3] <Geocentric Models> <Slides> [4] <Categories and Curves> <Slides>
Week 03	20 January	Chapters 5 and 6	[5] <Elemental Confounds> <Slides> [6] <Good and Bad Controls> <Slides>
Week 04	27 January	Chapters 7,8,9	[7] <Overfitting> <Slides> [8] <MCMC> <Slides>
Week 05	03 February	Chapters 10 and 11	[9] <Modeling Events> <Slides> [10] <Counts and Confounds> <Slides>
Week 06	10 February	Chapters 11 and 12	[11] <Ordered Categories> <Slides> [12] <Multilevel Models> <Slides>

Week ##	Meeting date	Reading	Lectures
Week 07	17 February	Chapter 13	[13] <Multilevel Adventures> <Slides> [14] <Correlated Features> <Slides>
Week 08	24 February	Chapter 14	[15] <Social Networks> <Slides> [16] <Gaussian Processes> <Slides>
Week 09	03 March	Chapter 15	[17] <Measurement> <Slides> [18] <Missing Data> <Slides>
Week 10	10 March	Chapters 16 and 17	[19] <Generalized Linear Madness> <Slides> [20] <Horoscopes> <Slides>

Important Links

- You can purchase *Statistical Rethinking: A Bayesian Course in R and Stan* from CRC Press.
- **The `rethinking` package**: Statistical Rethinking course and book package: https://github.com/rmcelreath/rethinking. I am using version 2.31.
- **Statistical rethinking 2023**: Course material for January - March 2023. https://github.com/rmcelreath/stat_rethinking_2023. It contains a link to the new Video playlist 2023, and to the slide deck collection. Furthermore it displays a table with the readings per week including the links to the appropriate video and slides. The repo also inlcudes PDFs for the homework and the scripts for the lecture animations. — I could not find the new R scripts associated with the (new) book text. They need to be collected from the slide lectures.
- **Statistical rethinking with brms, ggplot2, and the tidyverse**: brms/tidyverse-Conversion of Statistical Rethinking using bookdown by *A Solomon Kurz* (2023-01-26)
- **Functions for Learning Bayesian Inference**: Maybe I should also check this resource: It is an R package to learn bayesian inference with vignettes as short guides.

Solutions:

There are two different bookdown websites with book solutions:

- Solutions for the 1st edition by *Brynjólfur Gauti Jónsson* and
- Solutions for the 2nd edition by *Jake Thompson.*

I have also found two GitHub repos with solutions:

- GitHub solutions by *Taras Svirskyi* *(jffist)*
- GitHub solutions by William Wolf *(cavaunpeu)*

Caution

These solutions are written by members of the #RStats community and are not authorized by the author.

1 Golem of Prague

A golem (goh-lem) is a clay robot from Jewish folklore. It stands as a metaphor for scientific models. Like in the folklore they are made of dust (silicon chips), are very powerful but also very dumb and therefore dangerous.

There is nothing more in this chapter. I opened this file only to get the same header numbers as in the book.

2 Small and Large Worlds

~~~
```{r}
#| label: setup

library(tidyverse)
```
~~~

```
#> -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
#> v dplyr     1.1.2     v readr     2.1.4
#> v forcats   1.0.0     v stringr   1.5.0
#> v ggplot2   3.4.2     v tibble    3.2.1
#> v lubridate 1.9.2     v tidyr     1.3.0
#> v purrr     1.0.1
#> -- Conflicts ------------------------------------------ tidyverse_conflicts() --
#> x dplyr::filter() masks stats::filter()
#> x dplyr::lag()    masks stats::lag()
#> i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to be
```

~~~
```{r}
#| label: setup

library(patchwork)

```
~~~

Setup chunk results in several output chunks

The single setup chunk produces in the output several separate chunks that are tied with the specifics of the loaded packages. This split-up helps to see the association of messages with their corresponding packages.

Original

- The **small world** is the self-contained logical world of the model.
- The **large world** is the broader context in which one deploys a model.

Meta Remark (Study Guide)

> This chapter focuses on the small world. It explains probability theory in its essential form: counting the ways things can happen. Bayesian inference arises automatically from this perspective. Then the chapter presents the stylized components of a Bayesian statistical model, a model for learning from data. Then it shows you how to animate the model, to produce estimates.
>
> All this work provides a foundation for the next chapter, in which you'll learn to summarize Bayesian estimates, as well as begin to consider large world obligations.

Tidyverse

Meta Remark (My Replication Strategy)

The work by Solomon Kurz has many references to R specifics, so that people new to R can follow the course. Most of these references are not new to me, so I will not include them in my personal notes. There are also very important references to other relevant articles I do not know. But I will put these kind of references for now aside and will me mostly concentrate on the replication and understanding of the code examples.

One challenge for the author (Kurz) was to replicate all the graphics of the original version, even if they were produced just for understanding without underlying code. I will use only code lines that are essential to display Bayesian results. Therefore I will not replicate the very extensive explication how to produce with tidyverse means the graphic of the garden of forking data.

2.1 The Garden of Forking Data

2.1.1 Original

Bayesian inference is counting of possibilities

Bayesian inference is really just counting and comparing of possibilities. … In order to make good inference about what actually happened, it helps to consider everything that could have happened. A Bayesian analysis is a garden of forking data, in which alternative sequences of events are cultivated. As we learn about what did happen, some of these alternative sequences are pruned. In the end, what remains is only what is logically consistent with our knowledge.

2.1.1.1 Counting possibilities

Suppose there's a bag, and it contains four marbles. These marbles come in two colors: blue and white. We know there are four marbles in the bag, but we don't know how many are of each color. We do know that there are five possibilities: (1) [ ], (2) [ ], (3) [ ], (4) [ ], (5) [ ]. These are the only possibilities consistent with what we know about the contents of the bag. **Call these five possibilities the *conjectures*.**

Our goal is to figure out which of these conjectures is most plausible, given some evidence about the contents of the bag. We do have some evidence: A sequence of three marbles is pulled from the bag, one at a time, replacing the marble each time and shaking the bag before drawing another marble. **The sequence that emerges is: , in that order. These are the data.** (bold emphasis pb)

So now let's plant the garden and see how to use the data to infer what's in the bag. Let's begin by considering just the single conjecture, [ ], that the bag contains one blue and three white marbles. …

…

Notice that even though the three white marbles look the same from a data perspective—we just record the color of the marbles, after all—they are really different events. This is important, because it means that there are three more ways to see than to see .

Conjecture	Ways to produce
[ ]	$0 \times 4 \times 0 = 0$
[ ]	$1 \times 3 \times 1 = 3$

[ ]	$2 \times 2 \times 2 = 8$
[ ]	$3 \times 1 \times 3 = 9$
[ ]	$4 \times 0 \times 4 = 0$

I have bypassed the counting procedure related with the step-by-step visualization of the garden of forking data. The multiplication in the above table is still a summarized counting:

! Important

> Multiplication is just a shortcut to enumerating and counting up all of the paths through the garden that could produce all the observations.

> Notice that the number of ways to produce the data, for each conjecture, can be computed by first counting the number of paths in each "ring" of the garden and then by multiplying these counts together. … The fact that numbers are multiplied during calculation doesn't change the fact that this is still just counting of logically possible paths. This point will come up again, when you meet a formal representation of Bayesian inference.

The multiplication in the above table has to be interpreted the following way:

1. The possibility of the conjecture that the bag contains four white marbles is zero because the result shows also black marbles. This is the other way around for the last conjecture of four blue/black marbles.
2. The possibility of the conjecture that the bag contains one black and three white marbles is calculated the following way: The first marble of the result is black and — according to our conjecture — there is only one way (=1) to produce this black marble. The next marble we have drawn is white. This is consistent with three (=3) different ways(marbles) of our conjecture. The last drawn marble is again black which corresponds again with just one way (possibility) following our conjecture. So we get as result of the garden of forking data: `1 x 3 x 1`.
3. The calculation of the other conjectures follows the same pattern.

2.1.1.2 Combining Other Information

> We may have additional information about the relative plausibility of each conjecture. This information could arise from knowledge of how the contents of the bag were generated. It could also arise from previous data. Whatever the source, it would help to have a way to combine different sources of information to update the plausibilities. Luckily there is a natural solution: Just multiply the counts.

To grasp this solution, suppose we're willing to say each conjecture is equally plausible at the start. So we just compare the counts of ways in which each conjecture is compatible with the observed data. This comparison suggests that [ ] is slightly more plausible than [ ], and both are about three times more plausible than [ ]. **Since these are our initial counts, and we are going to update them next, let's label them *prior*.** (bold emphasis pb)

Principle of Indifference

As we will see later: the choice of the prior is not relevant for the final result. This is called the *Principle of Indifference*.
The only difference between a good or bad choice is the time (the number of steps) the updating process needs to produce the final result.
Before seeing any data the most common solution is to assign an equal number of ways that each conjecture could be correct.

Which assumption should we use, when there is no previous information about the conjectures? The most common solution is to assign an equal number of ways that each conjecture could be correct, before seeing any data. This is sometimes known as the **PRINCIPLE OF INDIFFERENCE**: When there is no reason to say that one conjecture is more plausible than another, weigh all of the conjectures equally. …

For the sort of problems we examine in this book, the principle of indifference results in inferences very comparable to mainstream non-Bayesian approaches, most of which contain implicit equal weighting of possibilities. For example a typical non-Bayesian confidence interval weighs equally all of the possible values a parameter could take, regardless of how implausible some of them are. In addition, many non-Bayesian procedures have moved away from equal weighting, through the use of penalized likelihood and other methods.

Bayesian Updating Process

Here's how we do the updating:

1. First we count the numbers of ways each conjecture could produce the new observation, .
2. Then we multiply each of these new counts by the prior numbers of ways for each conjecture.

Conjecture	Ways to produce	Prior counts	New count

[ ]	0	0	$0 \times 0 =$ 0
[ ]	1	3	$3 \times 1 =$ 3
[ ]	2	8	$8 \times 2 =$ 16
[ ]	3	9	$9 \times 3 =$ 27
[ ]	4	0	$0 \times 4 =$ 0

Typo

In the book the table header "Ways to produce" includes instead of — as I think is correct — .

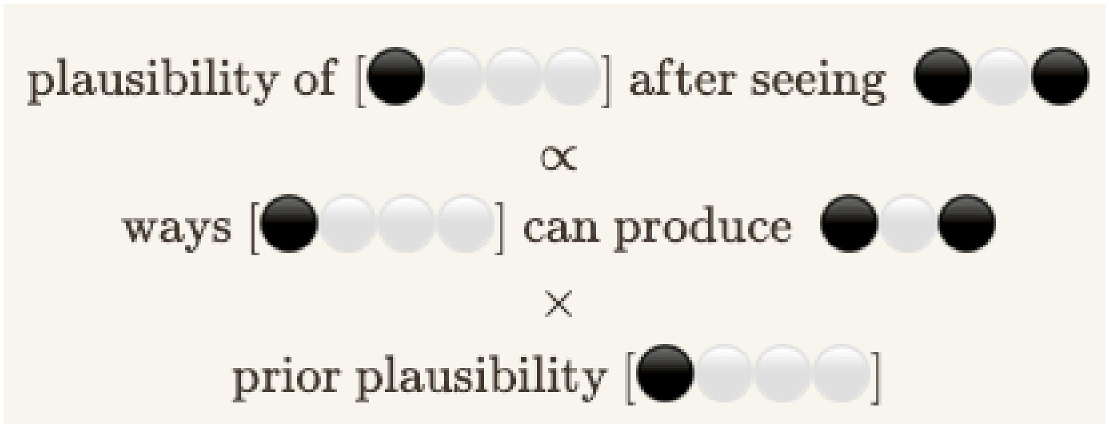
2.1.1.3 From Counts to Probability

Principle of honest ignorance

When we don't know what caused the data, potential causes that may produce the data in more ways are more plausible.

Two reasons for using probabilities instead of counts:

1. Only relative value matters.
2. Counts will very fast grow very large and difficult to manipulate.


plausibility of [●○○○] after seeing ●○●
∝
ways [●○○○] can produce ●○●
×
prior plausibility [●○○○]

Standardizing the plausibility

$$\text{plausibility of } p \text{ after } D_{\text{new}} = \frac{\text{ways } p \text{ can produce } D_{\text{new}} \times \text{prior plausibility } p}{\text{sum of products}}$$

~~~
```{r}
#| label: compute-plausibilities

## R code 2.1
ways <- c(0, 3, 8, 9, 0)
ways / sum(ways)
```
~~~

```
#> [1] 0.00 0.15 0.40 0.45 0.00
```

I understand that in the above code we assume as the very first prior plausibility the ways the results can be produced by the assumed conjecture proportion of blue marbles. It corresponds to the parameter value. This conclusion is somewhat hidden as our very first calculation already takes three drawn marbles ( ) into account. From another perspective this could also be seen as the first draw ( ) followed by two Bayesian updates ( ). This corresponds to the following counting: (In the second and third draw the first factor of the multiplication is always the prior.)

1. First draw : 0,1,2,3,4 ways corresponding to the 4 chosen conjectures.
2. Second draw : 0 x 4 = 0, 1 x 3 = 3, 2 x 2 = 4, 3 x 1 = 3, 4 x 0 = 0
3. Third draw : 0 x 0 = 0, 3 x 1 = 3, 4 x 2 = 8, 3 x 3 = 9, 0 x 4 = 0

This demonstration shows that the book example of already contains three priors: The first is identical with the conjectured proportion of blue marbles (0,1,2,3,4) and is equivalent to the parameter value for each conjecture. The second and third marbles already uses Bayesian updating.

Names of the different parts of the formula

Data = .

- A conjectured proportion of blue marbles, p, is usually called a **PARAMETER** value. It's just a way of indexing possible explanations of the data. For instance one conjectured proportion of one blue marble could be: `(p = 1)`. The others are: `(p = 0)`, `(p = 2` , `(p = 3)`, and `(p = 4` ways).

- The relative number of ways that a value p can produce the data is usually called a **LIKELIHOOD**. It is derived by enumerating all the possible data sequences that could have happened and then eliminating those sequences inconsistent with the data. For instance: `0.00, 0.15, 0.40, 0.45, 0.00`
- The prior plausibility of any specific p is usually called the **PRIOR PROBABILITY**. For instance: `0, 3, 8, 9, 0`
- The new, updated plausibility of any specific p is usually called the **POSTERIOR PROBABILITY**. For instance: `0, 3, 16, 27, 0`

2.1.2 Tidyverse

2.1.2.1 Counting possibilities

If we're willing to code the marbles as 0 = "white" 1 = "blue", we can arrange the possibility data in a tibble as follows.

Changed code slightly

I changed `rep()` to `rep.int()` and added `L` to the value of p1 resp. p5 to get integers (instead of doubles).

~~~
```{r}
#| label: create-marble-data

d <-
  tibble::tibble(p1 = 0L,
         p2 = rep.int(1:0, times = c(1, 3)),
         p3 = rep.int(1:0, times = c(2, 2)),
         p4 = rep.int(1:0, times = c(3, 1)),
         p5 = 1L)

head(d)
```
~~~

```
#> # A tibble: 4 x 5
#>      p1    p2    p3    p4    p5
#>   <int> <int> <int> <int> <int>
#> 1     0     1     1     1     1
#> 2     0     0     1     1     1
#> 3     0     0     0     1     1
#> 4     0     0     0     0     1
```

You might depict the possibility data in a plot.

Listing 2.1 Code listing to show the marble data as graph

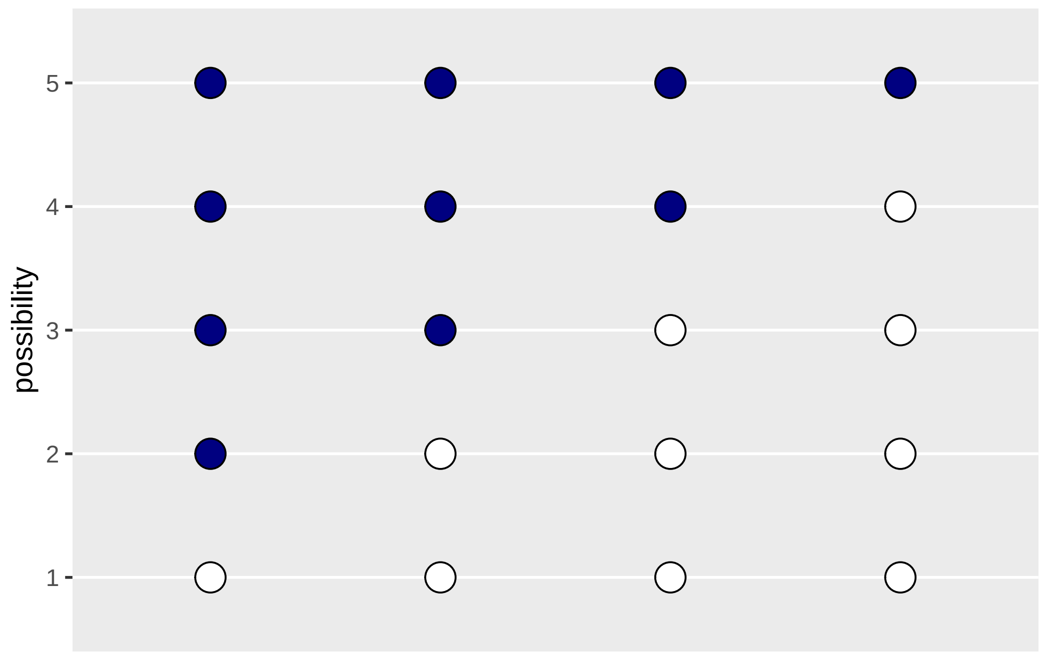
~~~
```{r}
#| label: fig-show-marble-data
#| fig-cap: "Marble Data"
#| attr-source: '#lst-marble-data lst-cap="Code listing to show the marble data as graph"'


d %>%
  set_names(1:5) %>%                                                  ①
  mutate(x = 1:4) %>%                                                 ②
  pivot_longer(-x, names_to = "possibility") %>%                      ③
  mutate(value = value %>% as.character()) %>%                        ④

  ggplot(aes(x = x, y = possibility, fill = value)) +                 ⑤
  geom_point(shape = 21, size = 5) +
  scale_fill_manual(values = c("white", "navy")) +
  scale_x_discrete(NULL, breaks = NULL) +
  theme(legend.position = "none")

```
~~~

① Change the column names from p<number> to <number>.
② Add a new column with the values 1 to 4 for each row.
③ Convert data frame from wide to long, excluding the `x` column.
④ Change the variable type of `value` column from double to character.
⑤ Plot the results.



Picture 2.1: Marble Data

At first I could not understand the code lines. I had to execute line by line to see what happens:

2.1.2.1.1 Annotation (1)

~~~
```{r}
#| label: using-set-names

set_names(d, 1:5)
```
~~~

```
#> # A tibble: 4 x 5
#>     `1`   `2`   `3`   `4`   `5`
#>   <int> <int> <int> <int> <int>
#> 1     0     1     1     1     1
#> 2     0     0     1     1     1
#> 3     0     0     0     1     1
#> 4     0     0     0     0     1
```

`rlang::set_names()` comes from {**rlang**} package which contains function for base types and core tidyverse features. It is exported to {**purrr**}, a package with tools for functional programming. It is equivalent to `stats::setNames()` but has more features and stricter argument checking. I does nothing more as to change the column names from p<number> to <number>. If one had used just numbers for the probability columns this line wouldn't have been necessary as it is shown in the next chunk. (I omitted the `utils::head()` argument of the last line as it is not necessary.)

~~~
```{r}
#| label: create-marble-data-2

df <-
  tibble(`1` = 0,
         `2` = rep(1:0, times = c(1, 3)),
         `3` = rep(1:0, times = c(2, 2)),
         `4` = rep(1:0, times = c(3, 1)),
         `5` = 1)

df
```
~~~

```
#> # A tibble: 4 x 5
#>     `1`   `2`   `3`   `4`   `5`
#>   <dbl> <int> <int> <int> <dbl>
#> 1     0     1     1     1     1
#> 2     0     0     1     1     1
#> 3     0     0     0     1     1
#> 4     0     0     0     0     1
```

It is interesting to see that the first and last column are doubles and not integers. I believe that the reason is that these two columns do not have variations, e.g. do not contain both variable values, so that R assumes the more general class of `numeric` and type of `double`.

- `class(5L)` and `typeof(5L)` both results to *integer* .
- Whereas `class(5)` is *integer* but `typeof(5)` is *double*.

~~~
```{r}
#| label: number-types

class(5L)
typeof(5L)
class(5)
typeof(5)
```
~~~

```
#> [1] "integer"
#> [1] "integer"
#> [1] "numeric"
#> [1] "double"
```

2.1.2.1.2 Annotation (2)

After understanding what `stats::set_names()` does the next line with `dplyr::mutate()` is easy. It adds a new column `x` with the values 1 to 4 for each row.

~~~
```{r}
#| label: add-x-column

df <- df |>
    set_names(1:5) |>
    mutate(x = 1:4)
df
```
~~~

```
#> # A tibble: 4 x 6
#>     `1`   `2`   `3`   `4`   `5`     x
#>   <dbl> <int> <int> <int> <dbl> <int>
#> 1     0     1     1     1     1     1
#> 2     0     0     1     1     1     2
#> 3     0     0     0     1     1     3
#> 4     0     0     0     0     1     4
```

2.1.2.1.3 Annotation (3)

I understood that the data frame is converted from a wide to a long structure. But together with the pipe and not naming the first parameter `-x` I did not catch the essence of the command.

A first understanding comes from the fact, that it is wrong to covert all columns to the long format:

~~~
```{r}
#| label: wrong-long-conversion

pivot_longer(data = df, cols = everything(), names_to = "possibility") |>
    print(n = 24)
```
~~~

```
#> # A tibble: 24 x 2
#>    possibility value
#>    <chr>       <dbl>
#>  1 1               0
#>  2 2               1
#>  3 3               1
#>  4 4               1
#>  5 5               1
#>  6 x               1
#>  7 1               0
#>  8 2               0
#>  9 3               1
#> 10 4               1
#> 11 5               1
#> 12 x               2
#> 13 1               0
#> 14 2               0
#> 15 3               0
#> 16 4               1
#> 17 5               1
#> 18 x               3
#> 19 1               0
#> 20 2               0
#> 21 3               0
#> 22 4               0
#> 23 5               1
#> 24 x               4
```

This (wrong) example shows that it is mandatory to exclude `x` from the conversion. Otherwise it would be included and integrated into the `possibility` column.

~~~
```{r}
#| label: correct-long-conversion

(d <-
    pivot_longer(data = df, cols = -x, names_to = "possibility"))
```
~~~

```
#> # A tibble: 20 x 3
#>        x possibility value
#>    <int> <chr>       <dbl>
#>  1     1 1               0
```

```
#>  2     1 2              1
#>  3     1 3              1
#>  4     1 4              1
#>  5     1 5              1
#>  6     2 1              0
#>  7     2 2              0
#>  8     2 3              1
#>  9     2 4              1
#> 10     2 5              1
#> 11     3 1              0
#> 12     3 2              0
#> 13     3 3              0
#> 14     3 4              1
#> 15     3 5              1
#> 16     4 1              0
#> 17     4 2              0
#> 18     4 3              0
#> 19     4 4              0
#> 20     4 5              1
```

The above code line for the conversion from wide to long is equivalent with naming explicitly all column names to convert:

~~~
```{r}
#| label: test-if-identical

d1 <- pivot_longer(data = df, cols = -x, names_to = "possibility")
d2 <- pivot_longer(data = df,
                   cols = c(`1`, `2`, `3`, `4`, `5`),
                   names_to = "possibility")
identical(d1, d2)
```
~~~

```
#> [1] TRUE
```

The identity demonstrates: The `-x` parameter excludes the `x` column from the wide to long transformation. It is a shorthand for naming all 5 columns that should be transformed.

Instead of the `base::identical()` function you could also use `base::all.equal()`. Comparing data frames is a rather complicated action summarized by a blog post from Sharla Gelfand. But keep in mind that the publication date is 2020-02-17 and that therefore some commands are outdated. Most important: use `base::all.equal()` instead of `dplyr::all_equal()`.

Warning

> `dplyr::all_equal()` allows you to compare data frames, optionally ignoring row and column names. It is deprecated as of dplyr 1.1.0, because it makes it too easy to ignore important differences.

The current version of the {**dplyr**} packages is 1.1.2.

2.1.2.1.4 Annotation (4)

~~~
```{r}
#| label: change-column-type
(d <- d |>
    mutate(value = value %>% as.character()))
```
~~~

```
#> # A tibble: 20 x 3
#>        x possibility value
#>    <int> <chr>       <chr>
#>  1     1 1           0
#>  2     1 2           1
#>  3     1 3           1
#>  4     1 4           1
#>  5     1 5           1
#>  6     2 1           0
#>  7     2 2           0
#>  8     2 3           1
#>  9     2 4           1
#> 10     2 5           1
#> 11     3 1           0
#> 12     3 2           0
#> 13     3 3           0
#> 14     3 4           1
#> 15     3 5           1
#> 16     4 1           0
#> 17     4 2           0
#> 18     4 3           0
#> 19     4 4           0
#> 20     4 5           1
```

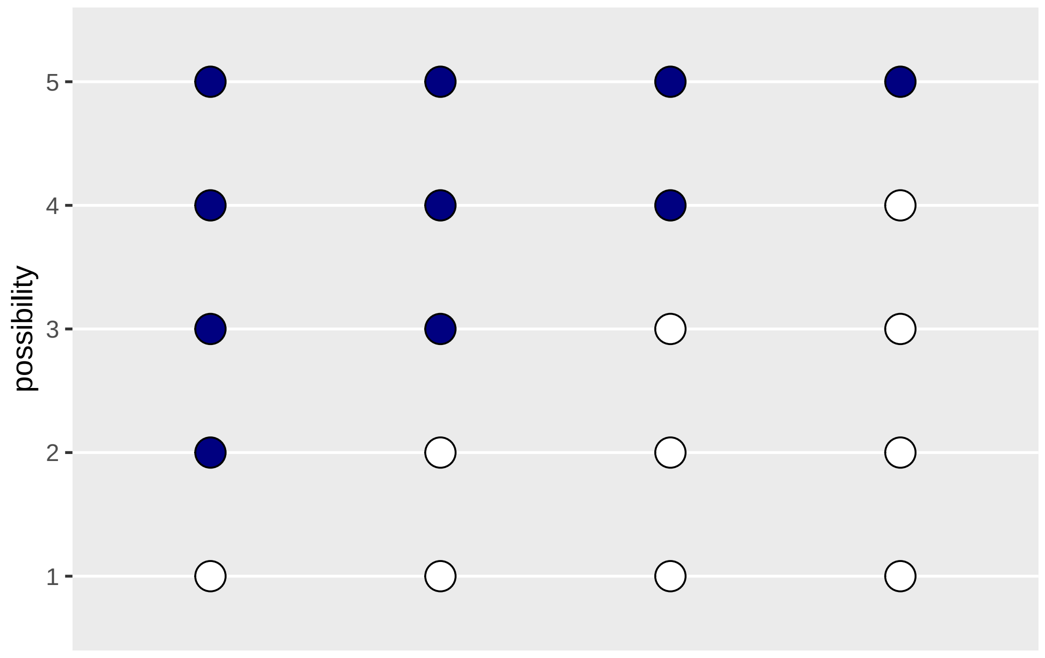
The `value` column is of type `double` as can be seen in the result of Annotation (3). For the plot it has to be changed to the type of `character`. Otherwise it could not be used with the `fill` option.

2.1.2.1.5 Annotation (5)

The plot uses code from the {**ggplot2**} package, which I do understand and will therefore not explain here.

~~~
```{r}
#| label: plot-possibilities-conjectures

d |>
  ggplot(aes(x = x, y = possibility, fill = value)) +
  geom_point(shape = 21, size = 5) +
  scale_fill_manual(values = c("white", "navy")) +
  scale_x_discrete(NULL, breaks = NULL) +
  theme(legend.position = "none")
```
~~~



2.1.2.1.6 Summarize the Possiblity Structure

Here's the basic structure of the possibilities per marble draw.