Martin Schmettow

New statistics for the design researcher

A Bayesian course in tidy R

2020-09-11

Springer

Contents

Part I

Preparations

1

Introduction

1.1 Whom this book is for

1.1.1 The empirical design researcher

If you are not a designer, chances are good that you are a design researcher very well. Are you doing studies with people and is your work about ways to achieve or improve products, services, trainings or business? Welcome to the not-so-special club of people I call design researchers.

Of course, I have no idea who you are, personally, but I figure you as one of the following personas:

You are an industrial design researcher, thirty-three years old, leading a small team in the center for research of a car maker. Your task is to evaluate emerging technologies, like augmented reality, car-to-car communication and smart light. Downstream research relies on your competent assessment of what is feasible. For example, you have just been asked to evaluate the following: Blue light makes people wake up easier, could it also be used to let car drivers not fall asleep? Several engineers and one industrial engineering student are involved in putting a light color stimulator for a premium car of the brand and stuff two workstations in its back, hooked to the recording systems and driving performance and physiological measures. Yes, this is as expensive as it sounds, and this why you have skin in the game, when you do a study.

You are a young academic and just got a cum laude master's degree in computer science. For your thesis you developed and implemented an animated avatar for a digital assistant. A professor from the social psychology department has read your thesis. He just got a funding for research on mass emergency communication using projected virtual characters. He fears that a young psychologist would not be up to the technical part of the project and found you. But, you ask yourself, am I up to the task of running experimental studies and do the statistics?

1.1.2 The experimentalist

If you are doing research with people, but the answers you seek are far from being applicable to anything in sight, don't put this book away. Chances are good that you will be able to read through the ephemeral details of the cases I present and recognize your own research situations, for example that you are comparing two groups. As you can see in the table of contents, you don't even have to read more than half the book to get there.

It may be true that strictly experimental data does not require more than group comparison. Still, I please you: also read the chapter on multilevel models. So much contemporary experimental research mistakens *average* for *universal*. It makes a difference to ask:

> "Are responses in the Stroop task responses delayed in the incongruent condition?"

or to ask:

> "Every person responds delayed in the incongruent condition?"

If your work is about revealing universal laws of behaviour, you have to search for the answer on an individual level. Technically, that means a rather simple multi-level model and a spaghetti plot will do the rest. But note that such a model is composed of many little participant-level models and all of them need their data. For multi-level models you need a within-subject design and repeated measures. On the second thought that makes full sense, as what really ask is:

> "Do all people shift into a slower mode in the incongruent condition?"

This is not about groups of people, but mode transitions in individual brains. These transitions can only be observed by putting one-and-the-same person into all the conditions. If you fear that the order of modes makes a difference, why not put order under statistical control? Record what you cannot control and check the interaction effects. Maybe it is not so bad.

Just another candy for you: Response times! Have you been struggling with them forever, because they are not Normal distributed? Have you resorted to non-parametric tests so many times that what you say became "response times are non-parametric"? You shouldn't say that. Furthermore, your non-parametric sacrifices can be history with Generalized Linear Models.

Then again, you may soon notice a disturbing lack of stars. Is this book like a very dark night? Consider the opposite possibility: most times when you

don't see the stars is at daylight. But let me hand you a torch with a chapter on model selection. The following paragraph may help you pushing your work through the review process:

> In order to confirm that the Stroop effect exists, we compared the predictive power of two models by information crieria. M_0 is an intercept-only model that denies the Stroop effect, M_1 allows for it. In order to evaluate universality of the Stroop effect, the models got participant-level random effects. As response times tend to have an offset and are left-skewed we chose exponential-Gaussian error distributions.

1.1.3 The applied researcher

Tongue-in-cheek, applied researchers take real problems to get their questions, but rarely solve them. Why not? It is legitimate, almost natural, to ask what causes the Uncanny Valley effect, for instance. You do a series of experimental study and also throw personality scales into the game. Maybe, the effect is not unversal. Why? Just that.

1.2 What is New Statistics?

This book makes the following assumptions:

1. Design research is for decision making, where one accounts for expected utility of design options. This requires

a. quantitative statements
b. statements of (un)certainty

2. Bayesian statistics is intuitive. Everybody has a good intuition about probability, value, decision making and gambling. It requires little effort to get to a more formal level of understanding.
3. Data arrives through data generating processes. Premise of statistical modelling is to neatly align to the generating process' anatomy.
4. Applied research data is multivariate and correlated. There is nothing such as a nuisance variable. Everything that helps understanding advantages and drawbacks of a design matters.
5. Average? Neverage! People differ, and diversity matters in design.
6. The universe is endless, but everything in it is finite, which strictly is at odds with linearity assumptions.

7. The human eye is a highly capable pattern evaluator. It would be a waste to not use visuals for exploring data and communicating results.
8. The best way to anticipate and plan data analysis is to simulate data upfront.
9. R is the statisticians preferred toolbox. 80% of statistical analysis can be done with 20% of R's full capabilities.

1.3 How to read this book

Chapter 1.4 introduces a framework for quantitative design research. It carves out the basic elements of empirical design research, such as users, designs and performance and links them to typical research problems. Then the idea of design as decision making under uncertainty is developed at the example of two case studies.

Chapter 3 introduces the basic idea of Bayesian statistics, which boils down to three remarkably intuitive conjectures:

1. uncertainty is the rule
2. you gain more certainty by observations
3. your present knowledge about the world is composed of what you learned from data and what you knew before.

The chapter goes on with introducing basic terms of statistical thinking. Finally, an overview on common statistical distributions serve as a vehicle to make the reader familiar with data generazing processes. I am confident that this chapter serves newbies and classically trained researchers with all tools they need to understand Bayesian statistics. The more formally trained reader may want to take this as a bedtime reading.

Chapter 2 is a minimal introduction to this marvelous programming language. Readers with some programming experience can work through this in just one day and they will get everything they need to get started with the book. Readers with prior R experience may get a lot out of this chapter, too, as it introduces the *tidy* paradigm of R programming. Tidy R is best be thought of as a set standard libraries 2.0. New tidy packages are arriving at an accelerating pace in the R world, and coding tidy is usually much briefer, easier to read and less error prone. It has at least the same significance as turning from procedural programming to object orientation. While this chapter serves as a stepping stone, the reader will encounter countless code examples throughout the book, working through these examples is a

The second part of the book consists of three chapters that strictly built on each other. The first chapter 4 introduces basic elements of linear models,

which are factors, covariates and interaction effects. A number of additional sections cover under-the-hood concepts, such as dummy variables or contrasts, as well as basic model criticism (residual analysis). Working through this chapter fully is essential as develops the jargon to a good deal.

However, for most readers, the first chapter is not sufficient to get to work, as it does not cover models for repeated measures. This is added extensively in chapter 5. It proceeds from simple repeated measures designs to complex designs, where human and multiple non-human populations encounter each other. After working through this chapter, the reader is prepared to design highly effective studies. As long as the patterns of randomness are not off by to much from the linear model assumptions, it also suffices to get to work, seriously.

The third chapter 6 opens up a plethora of possibilities to analyze all kinds of performance variables. Standard models are introduces to deal with counts, chances, temporal variables and rating scales. It is also meant as an eye opener for reseachers who routinely resorted to non-parametric procedures. After working through this chapter, I would consider anyone a sincere data analyst.

1.4 Quantitative design research

1.4.0.1 BEGIN

1.4.0.2 REDUCE

1.4.0.3 STRUCTURE

1.4.0.4 COMPLETE

A *design* is the structure of an artifact or process that people make for a purpose. This definition appears extremely broad as it covers everything, tea kettles, software, cockpits, training programs and complex human-machine systems. In fact, this definition is even redundant in two points: first, artifacts are usually made for a purpose and some call them just “tools”. Second, who else than people have purposes? Well, many wild animals reportedly create artificial structures for purposes such as shelter, mating and hunting. And when I have chosen such a broad definition for design, it is that I have absolutely no objections against using this book to compare the stability of birds nestings, for example. (In such a case, everyone please, read “design” as “part of the extended phenotype” and “purpose” as “emerged by natural selection”.

This book may apply to everything anyone likes to call a design; but only so for a specific set of research questions that hinge on the word “purpose”.

We all have purposes and much of the time it is quite a job to get there. Getting-there typically requires resources and often enough we find ourselves right there only to some extent. So, the two basic questions in *quantitative design research* are:

1. to what amount is a purpose fulfilled?
2. how easy is it to get there?

To the apt reader this may sound as i would next reduce all quantitative design research to the concept of usability and in a way i will be doing just that below (even discarding some subconcepts). The point here is that statistician actually don't care much about definitions from a domain, but think about research problems in a more abstract manner. Here is a number of things, a statistician would inquire:

- On what a scale is the measurement of purpose fulfillment?
- What is the expected precision of measures? Is a lot of noise to be expected?
- How many observations are available?
- Is the research concerned with how purpose fulfillment compares under various conditions?
- How are observations grouped?

In this book, quantitative design research is rather defined by a set of typical research problems, which includes the structure of the research question, as well as the empirical circumstances. In the following I will break this down one by one, and will also point out why the statistical framework of this book *Bayesian regression models* will do the job.

1.4.0.4.1 Introduce magnitude, as this is used when explaining uncertainty.

One obvious element is *quantification*, but that carries some ambiguity, as we can speak of measures or research questions. In much research, quantification happens during the process of measuring. Recording reaction times or counting errors certainly is quantitative. But, what matters is to really ask *quantitative research questions* and try to answer them. In the real world, decisions are (or should be) based on benefits and rational allocation of resources. Changing the background color of a website might just be a switch (and can have undesired effects as users hate change), but restructuring an intranet site can cost a million. In industrial design research, there usually is someone who wants to know whether this or that redesign is worth it, and that requires to ask research questions like the following:

- By how much does reaction ability degrade with age and it is safe that people beyond 80 still drive?

- Does design B reduce the required number of steps by a third, at least?
- What proportion of users prefers red over yellow? All websites better go red?

Sadly, in much existing research, precise quantification is frequently lost along the way and conclusions read more like:

- Older people have longer reaction times in traffic situations ($p \leq .05$).
- People find information more easily with design A, compared to B ($p \leq .05$).
- Chinese people on average prefer red color schemes over yellow ($p \leq .001$).

There simply is no numbers in these statements (except for the notorious p-values, which I will briefly discuss in chapter 3).

Regression models are just right for measures and they are terrific at drawing quantitative conclusions. Every regression model features a so called *outcome*, which must be a measure (rather than a category). At the same time, regression models can deal with a broad class of measures and their peculiarities.

Modern designs tend to be very complex and so are research questions, potentially. The options for designing just your personal homepage are myriad and there is considerable uncertainty about which options, or rather which configuration works best. Consider every option, say font type, font size, color, background color, position of menu, a potential impact factor on how pleasant the page is to read. Perhaps, someone should once and for all figure out the optimal configuration. It is recommended in such a case, that as many as possible impact factors are represented in a single study and evaluated by a single comprehensive statistical model. The primary reason for this recommendation is given in chapter **??**: impact factors have the nasty tendency to not act out independent of each other. Regression models handle such complexity with grace. There is theoretically no limit for the number of impact factors or *predicors* and interaction effects.

The central peculiarity in all behavioural research is that measures are extremely *noisy*. In the next chapter, the concept of measurement errors will be elaborated upon, but for now it suffices to understand that all measures approximate the measured property, only and that leaves room for *uncertainty*. In a simple experiment where participants respond to a signal and time is recorded, the first three trials will more likely have values that are widely scattered, like 900ms, 1080ms, 1110ms. Imagine this were an ability test in a training for, say, astronauts. To be admissioned to the space program the applicant needs a score of less than 1000ms. Would you dare to decide on the career of a young person based on these three observations? Hardly so.

On the other side, consider measuring a persons waste length for the purpose of tayloring a suit. By using a meter the taylor measures 990mm, and would

be perfectly fine with that. Why did the taylor not take a second and a third measure? Well, experience tells that meters are pretty precise measures and waste length shows relatively little variation (under constant habits). Say the two measures were 995mm and 989mm. Such small deviations have practically no influence on cutting the linen.

> "Our minds are not run as top - down dictatorships ; they are rambunctious parliaments, populated by squabbling factions and caucuses, with much more going on beneath the surface than our conscious awareness ever accesses."
>
> Carroll, Sean. The Big Picture (Kindle Locations 5029-5031). Oneworld Publications. Kindle Edition.

Vast fluctuations of measures are common in design research, simply for the fact that human behaviour is involved. Every magnitude we derive from a study is uncertain to some degree. Uncertainty makes that at any moment, we can rely on a quantitative result only to some extent, which influences how we take risks. Statistics is called inferential, when every derived magnitude comes with a degree of certainty attached. Section 3.1 gives some reasoning and examples, how to operate rationally under uncertainty, and drives at right into the arms of Bayesian statistics.

In an admission test for astronaut training, a decision is raised on very few individuals. Down on earth, everyday designs affect many people at once. Consider your personal homepage. If you decide, for aesthetic reasons, to shrink the font size, I promise you, that you just start loosing all visitors from the e-senior generation. Or, if your content is really good, they start using looking glasses on their o-pads. As a researcher, you can approach this in two ways: do a user study to compare the new design to the current design, or be the one who finds out, once and for all, what the optimal trade-off is between readability and aesthetic pleasure. Your method of choice would be an experiment.

When you have no greater goal in mind than proving your design is of quality, *user studies* are effective and quick. In the easiest case, you want to put your design against a fixed benchmark. For example, in the design of automotives, media devices in the cockpit may not distract the driver for more than 1.5 seconds at times [%REF]. To prove that, you will have to plug some advanced eye tracking gear into a prototype car and send people on the test drive. But once the precise data is in, things get really simple. The saccade measures directly represent what you were out for: the length of episodes of visual attention on the media display. You can take the average value. IN web design, it is common to compare two or more designs in order to make the best choice. An e-commerce company can put a potential future design on the test drive,

delivering it to a customer sample. Performance is measured as hard currency, which is as close to the purpose as it can get.

A user studies solves the momentary problem of comparing a local design to a benchmark (which can be another design). In the long run, design configurations are too manyfold to be compared in a one-by-one manner. It is inevitable that we try to trace some general patterns and apply our knowledge to a whole class of designs at once. Our research design just got one step more complex. Instead of just checking whether a smaller font size creates problems on a single website, the reseacher reaches out to comparing the combined effect of aging and font size on reading time, in general. This is what I call a *design experiment*.

Design experiments allow for much broader conclusions, if done right, and there are a some issues:

1. The design features under scrunity must be under control of the researcher. It does not suffice to collect some websites with varying font sizes, but every website needs to undergo the test at various font sizes.
2. The design feature must undergo a full range of manipulations. You will not find the laws of readability by just comparing 10pt versus 12pt.
3. Design features usually do not stand on their own. Readability is influenced by other factors, such as contrast and comprehensibility. Deriving a model from just one design will only generalize to this one design. Hence, the researcher must run the experiment on a sample of designs, with one of two strategies (or a mix of both): the *randomization strategy* takes a representative sample of designs, hoping that other impact factors average out. As we will see in **??**, this is a daring assumption. Therefore, the preferred way is *statistical control*, where potential impact factors are recorded and added as control variables to the regression model.

1.4.0.4.2 TODO

- Designing is wicked
- Evaluative design research
- Decision problems in design research
- Design research as exploration
- Mapping multidimensional impact factors
- Quantification for decision making
- Minimax decision making on designs
- Measures and psychometrics
- Emergent design theories

1.5 Studies in Design Research

- user studies
- experimental/fundamental studies
- qualitative vs quantitative

1.6 Observations and measures

1.6.1 Interaction sequences

Behavioural records not necessarily require one-way mirrors and the nights of video coding. Log files of web servers provide sequences of how users navigate a web site. Plugin software is available that records keystrokes and mouse actions on computers. The difficult part is the following: When observing 50 users while doing a non-trivial task, no two interaction sequences are exactly the same (if i had to bet on it). By itself, there is little value without further means of interpretation and this can go two ways up:

The diligent and skilled *qualitative* researchers are able to extract meaningful patterns from such sequences. For example, in usability tests, the following patterns are frequently observed and interpreted.

- a user jumping back to the main screen possibly went into a wrong direction
- users randomly probing around either have no clue or no motivation
- users in a longitudinal study who only interact at the begin and the end of the period, have no use for the system, really.

The *quantitative* researcher will aim at deriving measures from the interaction sequence. Formally, *to measure* means assigning numbers to observed sequences, so that these can be brought into an *order*, at least. Obviously, we need some sort of transformation that gives us a single *performance score*. This we call here the *performance function* and we have many choices, for example:

- the number of all exploratory events
- their relative frequency
- their longest uninterrupted sequence
- total time spent in exploration

Sometimes, you have to go a long way up the qualitative route, before you can derive useful measures. In [%Schnittker, Schmettow & Schraagen, 2016],

we coded sequences from nurses using an infusion pump. Individual sequences were compared to a optimal reference path, the similar the better. But how to measure similarity? For eample, *Levensthein distance* counts the number of edits to transform one sequence into the other and we used it as the performance score: *deviation from optimal path*. Even more interpreting was another study we did on the active user paradox. We recorded interaction sequences of users doing some repetitive tasks with a graphics software. The sessions were taped and analysed using a behavioural coding scheme. First, events were classified on a low level (e.g. "reading the handbook", "first time trying a function") and later aggregated to broader classes (e.g. "exploratory behavior"). The number of events in a given time frame that were classified as such were finally taken as a measure, representing the exploratory tendencies.

1.6.2 performance measures

A good point to start with is the classic concept of "usability", which the ISO 9142-11 defines by the following three high-level criteria:

- effectiveness: can users accomplish their tasks?
- efficiency: what resources have to be spend for a task, e.g. time.
- satisfaction: how did the user like it?

While these criteria originated in the field of Human-Computer Interaction, they can easily be adapted to compare everything that people do their work with. Even within the field, it has been adapted to hundreds of studies and a hundred ways are reported of assessing these criteria.

Effectiveness is often measured by *completion rate (CR)*. A classic waterfall approach would be to consult the user requirements documents and identify the, let's say eight, crucial tasks the system must support. User test might then show that most users fail at the two tasks, and a completion rate of 75% is recorded for the system. Completion rate is only a valid effectiveness measure with *distinct tasks*. Strictly, the set of tasks also had to be *complete*, covering the whole system. When completion rate is taken from in series of *repetitive tasks*, it depends on whether it is effectiveness or efficiency. It is effectiveness, when a failed operation is unrepairable, such as a traffic accident, data loss on a computer or medical errors. But, who cares for a single number between 0 and 1, when the user test provides such a wealth of information on *why* users failed? Effectiveness, in the sense of task completion, is primarily a qualitative issue and we shall rarely encounter it in this book.

A more subtle notion of effectiveness is the *quality of outcome*, and despite the very term, is a measure. Perhaps, it should better be called *task perfection*. Reconsider the AUP study, where participants had to modify a given graphic, e.g. change some colors and erase parts of it. Of course, some participants

worked neatly, whereas others used broader strokes (literally). There certainly is a reasonable objective way to rank all results by quality.

Efficiency is where it really gets quantitative as with efficiency, we ask about resources: time, attention, strain, distraction and Euros. Other than that, efficiency can be measured in a vast variety of ways: ToT, clicks, mental workload or time spent watching the traffic while driving.

Counting the *number of clicks* before someone has found the desired piece of informnation is a coarse, but easy to acquire and intuitive measure of efficiency, and so is *time-on-task (ToT)*, which is just the sum . Still, these It differs from ToT in that it only counts real action, not the time a participant read the manual or watched the sky. In the downside, this is a very limited definition of action. While errors are usually best analyzed qualitatively, *error rate* can be efficiency, too, when failures are non-critical, but can be repaired in some time, such as correcting a typing error.

ToT and other performance measures can be very noisy, and RT even more so. In order to arrive at sufficiently certain estimates, it is recommended to always plan for *repetition*. In 5 we will make heavy use of repeated measures on users, tasks and designs.

Above, I did not mean to say that design research always requires the sometimes painful recording of interaction sequences. The majority of studies jumps over them and proceed to performance measures, directly, by counting attempts or pressing stop watches.

In many situations, such *direct measures* are right spot-on: When controlling a car in dense city traffic, a few hundred milliseconds is what makes huge a difference and therefore *reaction time* is a valid performance measure. In experimental cognitive research reaction times are a dominant tool to reveal the structure of the mind. The Stroop tasks is a golden evergreen cognitive psychology and serves as an example. Participants are shown a words drawn in one of three ink colors and are asked to name the ink as quick as possible. The Stroop effect occurs: in the incongruent condition, participants respond much slower than in the congruent and neutral (a few hundred ms). There is ample sweet replication of this effect and an ecosystem of theories surround it. In [%Schmettow, Noordzij, Mundt] we employ the Stroop task to read the minds of participants. We showed them pictures with computers on them, followed by a printed word (e.g., “explore”) on which to perform the Stroop task. It was conceived that reaction time is increased when a participant experiences a strong association, like:

black tower PC + “explore” –> “can someone hand me a screwdriver”

CONDITION WORD : Ink congruent [GREEN]: green incongr [YELLOW]: green neutral [CHAIR]: green

1.6.2.1 COMPLETE ME

1.6.2.2 –>*Satisfaction*

1.6.3 experience [TBD]

1.6.4 design features [TBD]

1.6.5 the human factor [TBD]

1.6.6 situations [TBD]

2

Getting started with R

In this book, we will be using the statistical computing environment R. R at its core is a programming language that specializes on statistics and data analysis. Like all modern programming languages, R is more than just a compiler or interpreter that translates human-writeable formal statements into something a computer understands and can execute. R comes with a complete set of standard libraries that cover the usual basic stuff, as well as a collection of common statistical routines, for example the t-test or functions to compute mean and variance. Even regression analysis and plotting is included in R. Finally, packaged with R comes a rudimentary programming environment, including a console, a simple editor and a help system.

Most of what comes packaged with R, we will set aside. R is basically a brilliantly designed programming language for the task, but many of the standard libraries are an inconsistent and outdated mess. For example, R comes with a set of commands to import data from various formats (SPSS, CSV, etc.). Some of these commands produce objects called data.frames, whereas others return lists of lists. Although one can convert lists into dataframes, it is easier and prevents confusion if all import functions simply create dataframes. As another example, to sort a data frame by participant number, one has to write the following syntax soup:

```
D[order(D$Part),]
```

In the past few years, a single person, Hadley Wickham, has almost single-handedly started an initiative known as the *tidyverse*. The tidyverse is a growing collection of libraries that ground on a coherent and powerful set of principles for data management. One of the core packages is *dplyr* and it introduces a rich, yet rather generic, set of commands for data manipulation. The sorting of a data frame mentioned above would be written as

```
D %>% arrange(Part)
```

One of Wickham's first and well-known contributions is the *ggplot* system for graphics. Think of R's legacy graphics system as a zoo of individual routines, one for boxplots, another one for scatterplots asf. Like animals in a zoo, they live in different habitats with practically no interaction. ggplot implements a rather abstract framework for data plots, where all pieces can be combined in a myriad of ways, using a simple and consistent syntax.

Where to get these gems? R is an open source system and has spawned an open ecosystem for statistical computing. Thousands of extensions for R have been made available by data scientists for data scientists. The majority of these packages is available through the *comprehensive R archive network (CRAN)*. For the common user it suffices to think of CRAN as an internet catalogue of packages, that can be searched and where desired packages can be downloaded and installed in an instance.

Finally, R comes with a very rudimentary programming environment that carries the questionable charm of early 1990s. Whereas several alternatives exist, most R users will feel most comfortable with the R programming environment Rstudio. At the time of writing, it is the most user-friendly and feature rich software to program in R. The next sections describe how you can set up a fully functional environment and verify that it works. Subsequently, we will get to know the basics of programming in R.

2.1 Setting up the R environment

First, we have to make sure that you have the two essential applications R and Rstudio downloaded and installed on your computer. The two programs can be retrieved from the addresses below. Make sure to select the version fit for your operating system.

- R
- Rstudio

If you fully own the computer you are working with, meaning that you have administrator rights, just do the usual downloading and running of the setup. If everything is fine, you'll find R and Rstudio installed under `c\Programs\` and both are in your computers start menu. You can directly proceed to the installation of packages.

In corporate environments, two issues can arise with the installation: first a user may not have administrator rights to install programs to the common path `c:\programs\`. Second, the home directory may reside on a network drive, which is likely to cause trouble when installing packages.

If you have no administrator rights, you must choose your home directory during the setup. If that is a local directory, (`c:/Users/YourName/`), this should work fine and you can proceed with the installation of packages.

If your home directory (i.e. `My Documents`) is located on a network drive, this is likely to cause trouble. In such a case, you must install R and Rstudio to a local directory (on your computers hard drive), where you have full read/write access. In the following, it is assumed that this directory is `D:/Users/YourName/`:

1. create a directory `D:/Users/YourName/R/`. This is where both programs, as well as packages will reside.
2. create a sub directory `Rlibrary` where all additional packages reside (R comes with a pack of standard packages, which are in a read-only system directory).
3. start Rstudio
4. create a regular text file `File -> New File -> Text file`
5. copy and paste code from the box below
6. save the file as `.Rprofile` in `D:/Users/YourName/R/`
7. open the menu and go to `Tools -> Global options -> General -> Default working directory`. Select `D:/Users/YourName/R/`.

```
## .Rprofile

options(stringsAsFactors = FALSE)

.First <- function(){
  RHOME <<- getwd()
    cat("\nLoading .Rprofile in", getwd(), "\n")
  .libPaths(c(paste0(RHOME,"Rlibrary"), .libPaths()))
}

.Last <- function(){
  cat("\nGoodbye at ", date(), "\n")
}
```

With the above steps you have created a customized start-up profile for R. The profile primarily sets the library path to point to a directory on the computers drive. As you are owning this directory, R can install the packages without admin rights. In the second part, you configure Rstudio's default path, which is where R, invoked from Rstudio, searches for the .Rprofile.

After closing and reopening Rstudio, you should see a message in the console window saying:

> Loading .Rprofile in D:/Users/YourName/R/

That means that R has found the `.Rprofile` file and loaded it at start-up. The `.Rprofile` file primarily sets the path of the *library*, which is the collection of packages you install yourself. Whether this was successful can be checked by entering the `console` window in Rstudio, type the command below and hit Enter.

```
.libPaths()
```

If your installation went fine, you should see an output like the following. If the output lacks the first entry, your installation was not successful and you need to check all the above steps.

> [1] "D:/Users/YourName/R/Rlibrary" "C:/Program Files/R/R-3.3.0/library"

2.1.1 Installing CRAN packages

While R comes with a set of standard packages, thousands of packages are available to enhance functionality for every purpose you can think of. Most packages are available from the *Comprehensive R Network Archive (CRAN)*.

For example, the package *foreign* is delivered with R and provides functions to read data files in various formats, e.g. SPSS files. The package *haven* is a rather new package, with enhanced functionality and usability. It is not delivered with R, hence, so we have to fetch it.

Generally, packages need to be *installed once* on your system and to be *loaded everytime* you need them. Installation is fairly straight-forward once your R environment has been setup correctly and you have an internet connection.

In this book we will use a number of additional packages from CRAN. The listed packages below are all required packages, all can be loaded using the `library(package)` command. The package *tidyverse* is a metapackage that installs and loads a number of modern packages. The ones being used in this book are:

- *dplyr* and *tidyr* for data manipulation
- *ggplot* for graphics
- *haven* for reading and writing files from other statistical packages
- *readr* for reading and writing text-based data files (e.g., CSV)
- *readxl* for reading Excel files
- *stringr* for string matching and manipulation

```
## tidyverse
library(tidyverse)

## data manipulation
library(openxlsx)

## plotting
library(gridExtra)

## regression models
library(rstanarm)

## other
library(devtools) ## only needed for installing from Github
library(knitr)

## non-CRAN packages
library(mascutils)
library(bayr)
```

We start by *checking the packages*:

1. create a new R file by `File --> New file --> R script`
2. copy and paste the above code to that file and run it. By repeatedly pressing `Ctrl-Return` you run every line one-by-one. As a first time user with a fresh installation, you will now see error messages like:

   Error in library(tidyverse) : there is no package called 'tidyverse'

This means that the respective package is not yet present in your R library. Before you can use the package *tidyverse* you have to install it from CRAN. For doing so, use the built-in package management in RStudio, which fetches the package from the web and is to be found in the tab *Packages*. At the first time, you may have to select a repository and refresh the package list, before you can find and install packages. Then click `Install`, enter the names of the missing package(s) and install. On the R console the following command downloads and installs the package *tidyverse*.

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

CRAN is like a giant stream of software pebbles, shaped over time in a growing tide. Typically, a package gets better with every version, be it in reliability or versatility, so you want to be up-to-date. Rstudio has a nice dialogue to update packages and there is the R command:

```
update.packages("tidyverse")
```

Finally, run the complete code block at once by selecting it and `Ctrl-Enter`. You will see some output to the console, which you should check once again. Unless the output contains any error messages (like above), you have successfully installed and loaded all packages.

Note that R has a somewhat idiosyncratic jargon: many languages, such as Java or Python, call "libraries" what R calls "packages". *The* library in R is strictly the set of packages installed on your computer and the `library` command loads a package from the library.

2.1.2 Installing packages from Github

Two packages, *mascutils* and *bayr* are written by the author of this book. They have not yet been committed to CRAN, but they are available on *Github* which is a general purpose versioning system for software developers and few authors. Fortunately, with the help of the *devtools* package it is rather easy to install these packages, too. Just enter the Rstudio console and type:

```
library(devtools)
install_github("schmettow/mascutils")
install_github("schmettow/bayr")
```

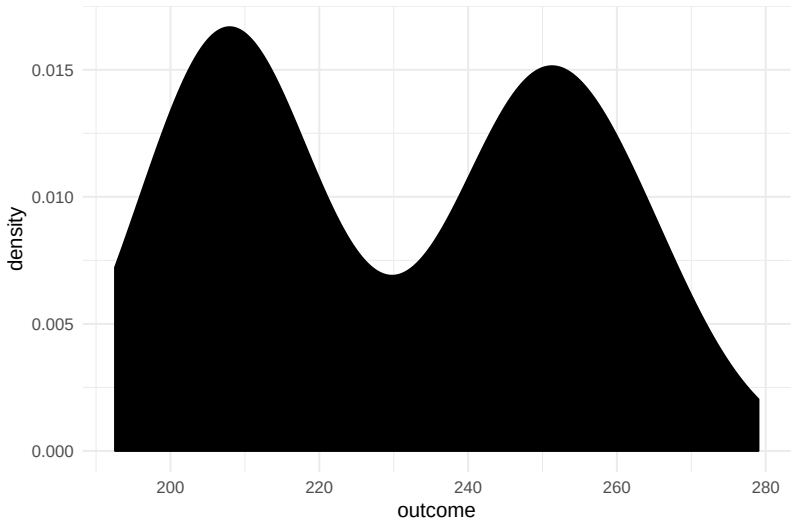
Again, you have to do that only once after installing R and you can afterwards load the packages with the `library` command. Only if the package gets an update to add functionality or remove bugs, you need to run these commands again.

2.1.3 A first statistical program

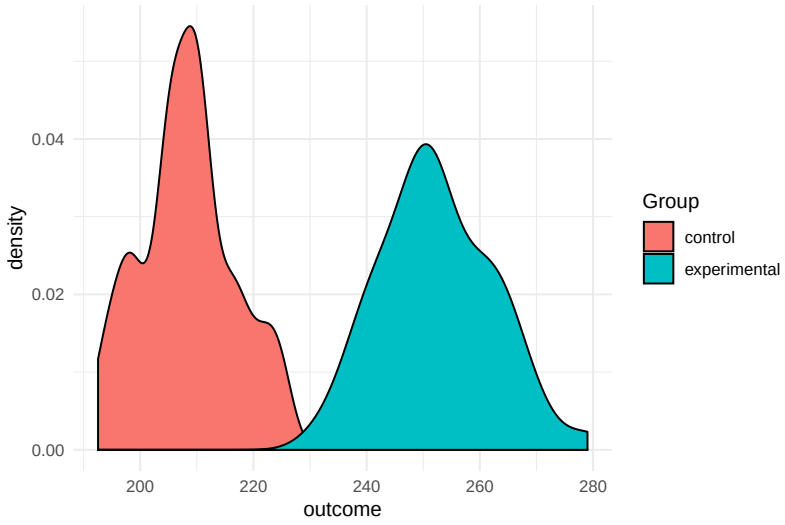
After you have set up your R environment, you are ready to run your first R program (you will not yet understand all the code, but as you proceed with this book, all will become clear):

1. Stay in the file where you have inserted and run the above code for loading the packages.
2. Find the *environment tab* in Rstudio. It should be empty.
3. Copy-and-paste the code below into your first file, right after library commands.
4. Run the code lines one-by-one and observe what happens (in RStudio: Ctrl-Enter)

```
## Simulation of a data set with 100 participants
## in two between-subject conditions
N <- 100
levels <- c("control","experimental")
Group <- rep(levels, N/2)
age <- round(runif(N, 18, 35),0)
outcome <- rnorm(N, 42 * 5, 10) + (Group == "experimental") * 42
Experiment <- tibble(Group, age, outcome) %>% mascutils::as_tbl_obs()

Experiment %>% sample_n(8)

## Plotting the distribution of outcome
Experiment %>%
  ggplot( aes(x = outcome) ) +
  geom_density(fill = 1)
```




```
## ... outcome by group
Experiment %>%
  ggplot( aes(fill = Group, x = outcome) ) +
  geom_density()
```




```
## ... statistical model comparing the groups

model <- stan_glm(formula = outcome ~ Group,
            data = Experiment)
```

```
fixef(model)
```

fixef	center	lower	upper
Intercept	209.9	207	213.0
Groupexperimental	41.4	37	45.6

Observe Console, Environment and Plots. Did you see

- how the *Environment* window is populated with new variables (Values and Data)?
- a table appears in the *Console*, when executing the `summary(Experiment)` command?
- how the "camel-against-the-light" in *Plots* tab morphed into "two-piles-of-colored-sugar"?

Congratulations! You have just

- simulated data of a virtual experiment with two groups
- summarized the data

- plotted the data and
- estimated a Bayesian regression model that compares the two groups.

Isn't it amazing, that in less than 20 simple statements we have just reached the level of a second-year bachelor student? Still, you may find the R output a little inconvenient, as you may want to save the output of your data analysis. Not long ago, that really was an issue, but in the past few years R has become a great tool for *reproducable research*. The most simple procedure of saving your analysis for print or sharing is to:

1. save the R file your have created by hitting `CTRL-S` and selecting a directory and name.
2. in RStudio open `File --> Compile notebook` and select Word as a format.
3. Hit `Compile`

A new Word document should appear that shows all code and the output. Now, you can copy-and-paste the graphics into another document or a presentation.

2.1.4 Bibliographic notes

Getting started with Rstudio (presentation)

Getting started with Rstudio (ebook)

rstudio cheat sheets is a collection of beautifully crafted cheat sheets for ggplot, dplyr and more. I suggest you print the data mangling and ggplot cheat sheets and always keep them on your desk.

The tidyverse is a meta package that loads all core tidyverse packages by Hadley Wickham.

2.2 Learning R: a primer

cRazy as in 'idiosyncrasy'

This book is for applied researchers in design sciences, whose frequent task is to analyze data and report it to stakeholders. Consequently, the way I use R in this book capitalizes on interactive data analysis and reporting. As it turns out, a small fraction of R, mostly from the tidyverse, is sufficient to write R code that is effective and fully transparent. In most cases, a short chain of simple data transformations tidies the raw data which can then be pushed into

a modelling or graphics engine that will do the hard work. We will not bother (ourselves and others) with usual programming concepts such as conditionals, loops or the somewhat eccentric approaches to functional programming. At the same time, we can almost ignore all the clever and advanced routines that underlay statistical inference and production of graphics, as others have done the hard work for us.

R mainly serves three purposes, from easy to advanced: 1. interactive data analysis 2. creating data analysis reports 3. developing new statistical routines.

It turns out that creating multimedia reports in R has become very easy by the knitr/markdown framework that is neatly integrated into the Rstudio environment. I

With R one typically *works interactively through a data analysis*. The analysis often is a rather routine series of steps, like:

1. load the data
2. make a scatter plot
3. run a regression
4. create a coefficient table

A program in R is usually developed iteratively: once you've loaded and checked your data, you progress to the next step step of your analysis, test it and proceed. At every step, one or more new objects are created in the environment, capturing intermediate and final results of the analysis:

1. a data frame holding the data
2. a graphics object holding a scatterplot
3. a model object holding the results of a regression analysis
4. a data frame for the coefficient table

As R is an interpreter language, meaning there is no tedious compile-and-run cycles in everyday R programming. You develop the analysis as it happens. It is even normal to jump back and forth in an R program, while building it.

R is a way to *report and archive* what precisely you have been doing with your data. In statistics, mathematical formulas are the common form of unambiguously describing a statistical model. For example, the following equation defines a linear regression model between the observed outcome y and the predictor x:

$$\mu_i = \beta_0 + \beta_1 x_i$$
$$y_i \sim N(\mu_i, \sigma)$$

As we will later see 4.3.1, in Rs formula language the same model is unambiguously specified as:

```
y ~ x
```

R is currently the *lingua franca* of statistical computing. As a programming language, R has the same precision as math, but is more expressive. You can specify complex models, but also graphics and the steps of data checking and preparation. As an example, consider an outlier removal rule of:

> An observation is valid if it does not exceed the tenfold of the observation mean.

We just applied our own rule of outlier removal to the data. Others may consider this rule invalid or arbitrary. Disagreement is virtue in science and one can only disagree with what one actually sees. In R, the researcher *formally reports* what precisely has been done with the data. For example, the same outlier removal rule is unambiguously specified by the following code (the first line just simulates some data).

```
D <- tibble(score = c(1, 2, 4, 3, 5, 6, 50, 800))

D %>% filter(score < mean(score))
```

score
1
2
4
3
5
6
50

Finally, R is a way to *develop and share statistical programs*. Thousands of packages in the R ecosystem cover almost all statistical problems you can imagine. As a programming language, R has been designed for that particular purpose. Under the hood of R, a bunch of generic, yet powerful, principles purr to make it a convenient language for typical problems in statistical computation. Readers with programming experience can fly over the R basics that follow. But, as a specific purpose language R has a few idiosyncracies you should know about:

Almost all programming languages the first element of a list has the index zero. We got used to it, but for beginners it is a another hurdle that is unnecessary. Mathematicians, catholiques, software developers in bars and everyone, young or old, counts

> “one”, “two” ,“three”.

And so does R:

```
c(1:3)[1:3]
```

```
## [1] 1 2 3
```

Counting from one is perhaps the most lovable idiosyncracy of R. But, lets also welcome people who have experience with other programming languages:

The first thing one has to know about R is that it is a *functional programming* language. A function simply is a programmed procedure that takes data as input, applies some transformation and returns data as output. That sounds trivial, but there is an important difference to most other languages: Different to procedures in Pascal or object oriented methods (in Java or Python), functions are forbidden to modify any external object. A certain function is a black box, but one can be sure that the only thing it does is return a new object.

At the same time, functions are first-class citizens in R and can be called everywhere, even as an *argument to another function*. The *plyr* package is famous for functions that call functions, also called high-level functions. The following three liner makes heavy use of high level functions. First, a list of binary matrices is generated by repeatedly calling a random number generator, then the row sum of all matrices is computed and returned as a (new) list.

```
make_binary_matrix <-
  function(size) as.matrix(rbernoulli(size^2, p = 1/4))

LoM <- llply(c(5:7), make_binary_matrix)

llply(LoM, rowSums)
```

```
## [[1]]
##  [1] 0 0 0 0 0 0 1 0 0 0 1 1 0 0 0 1 1 0 0 1 1 0 0 1 1
##
## [[2]]
##  [1] 0 0 1 1 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 0 1 0 1 0 0 0 1 0 0 0 0 0 0 0
##
## [[3]]
##  [1] 0 1 0 0 0 1 1 0 0 1 0 0 1 0 0 1 0 0 0 0 1 0 0 0 0 1 1 0 0 0 0 0 0 0 1 0 0 1
## [39] 1 1 1 0 0 0 0 0 0 1 0
```

What we have seen is a routine application of higher level functions: apply a transformation to a sequence of data sets. In a majority of programming languages you had to write a loop instead and this is why experienced programmers can easily fall for the active user paradox when they learn R, by

sticking to loops. Believe me one thing: Once you have wrapped your head around functional programming, you will program the same procedure in a quarter of time with half the code and your program will run significantly faster. My general advice is:

> Whenever you think you need a loop, you don't.

For me it helped to imagine the following: loops carry the notion of chain of data that moves over a fixed transformation device. In functional programing functions the can work in a hierarchy, a high-level device moves along a string of data. It carries an exchangeable low-level device that applies the desired transformation to every position.

If you come from relational data bases you you have something in common with the statistician: you both think in transformation of tables. Not coincidently, the features in *dplyr*, the tidy data transformation engine, are clearly borrowed from SQL. You will also feel at home with the idea of reports powered by functional chunks embedded in a templating system.

For object orientation folks, R is a good choice, but you have to get used to it. First, it gives you the choice of several object orientation systems, which sometimes requires to install a package. The so-called *S3 system* is the original. It is rather limited and some even call it informal. The approach is as simple as it is unusal. S3 mainly is a raw method dispatcher that can handles overloading of functions. S3 puts methods first, then objects, whereas in traditional object orientation, the method belongs to the class, making the object "knows" its methods. In S3, the object and the method both know their class. When calling an S3 method, you actually call a generic method that finds the matching method and applies it.

Beginners are at peace with all of this. You can count like you do. Functional programming is intuitive for working on research data. And because of S3 the function `summary` always does something useful.

2.2.1 Assigning and calling Objects

Any statistical analysis can be thought of as a production chain. You take the raw data and process it into a neat data table, which you feed into graphics and regression engines or summarize by other means. At almost every step there is an input and an output object.

Objects are a basic feature of R. They are temporary storage places in the computer's memory. Objects always have a name chosen by the programmer. By its name, a stored object can be found back at any time. Two basic operations apply for all objects: an object is stored by *assigning* it a name and it is

retrieved by *calling* its name. If you wanted to store the number of observations in your data set under the name `N_obs`, you use the assignment operator `<-`. The name of the variable is left of the operator, the assigned value is right of it.

```
N_obs <- 100
```

Now, that the value is stored, you can call it any time by simply calling its name:

```
N_obs
```

```
## [1] 100
```

Just calling the name prints the value to Console. In typical interactive programming sessions with R, this is already quite useful. But, you can do much more with this mechanism.

Often, what you want is to do calculations with the value. For example, you have a repeated measures study and want to calculate the average number of observations per participant. For this you need the number of observations, and the number of participants. The below code creates both objects, does the calculation (right of `<-`) and stores it in another object `avg_N_Obs`

```
N_Obs <- 100
N_Part <- 25
avg_N_Obs <- N_Obs/N_Part
avg_N_Obs
```

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

Objects can exist without a name, but are volatile, then. They cannot be used any further. The following arithmetic operation does create an object, a single number. For a moment or so this number exists somewhere in your computers memory, but once it is printed to the screen, it is gone. Of course, the same expression can be called again, resulting in the same number. But, strictly, it is a different object.

```
N_Obs/N_Part ## gone after printing
```

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

```
N_Obs/N_Part ## same value, different object
```

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

There even is a formal way to show that the two numbers, although having the same value assigned, are located at different addresses. This is just for the purpose of demonstration and you will rarely use it in everyday programming tasks:

```
tracemem(N_Obs/N_Part)
```

```
## [1] "<000000003E499478>"
```

```
tracemem(N_Obs/N_Part)
```

```
## [1] "<000000003E500920>"
```

2.2.2 Vectors

Notice the `[1]` that R put in front of the single value when printing it? This is an *index*. Different to other programming languages, all basic data types are *vectors* in R. Vectors are containers for storing many values *of the same type*. The values are addressed by the index, `N_Obs[1]` calls the first element. In R, *indices start counting with 1*, which is different to most other languages that start at zero. And if you have a single value only, this is just a vector of length one.

For statistics programming having vectors as basic data types makes perfect sense. Any statistical data is a collection of values. What holds for data is also true for functions applied to data. Practically all frequently used mathematical functions work on vectors, take the following example:

```
X <- rnorm(100, 2, 1)
mean(X)
```

```
## [1] 2.04
```

The `rnorm` command *produces* a vector of length 100 from three values. More precisely, it does 100 random draws from a normal distribution with mean 2 and an SD of 1. The `mean` command takes the collection and *reduces* it to a single value. By the way, this is precisely, what we call *a statistic: a single quantity that characterizes a collection of quantities*.

2.2.3 Basic object types

Objects can be of various *classes*. In R the common basic classes are: logical, factor, character, integer and numeric. Besides that, programmers can define their own complex classes, for example, to store the results of regression models.

Objects of type *logical* store the two levels `TRUE`, `FALSE`, like presence or absence of a treatment, or passed and failed test items. With Boolean operators one can compute new logical values, for example

```
Apple <- TRUE
Pear  <- FALSE

Apple & Pear ## and
```

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

```
Apple | Pear ## or
```

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

More generally, logical values can be used for categorization, when their are only two categories, which is called a *dichotomous variable*. For example, gender is usually coded as a vector of characters (`"m", "f", "f"`), but one can always do:

```
is_female <- c(FALSE, TRUE, TRUE)
is_male   <- c(T, F, F)
```

Programmers are lazy folks when it comes to typing, therefore R allows you to abbreviate `TRUE` and `FALSE` as shown above. As a consequence, one should never assign objects the name reserved for logical values, so don't do one of the following:

```
## never do this
TRUE <- "All philosophers have beards" ## a character object, see below
F    <- "All gods existed before the big bang"
42 * F
```

The class `numeric` stores real numbers and is therefore abundant in statistical programming. All the usual arithmetic operations apply:

```
a <- 1.0
b <- a + 1
sqrt(b)
```

```
## [1] 1.41
```

Objects of class `integer` are more specific as they store natural numbers, only. This often occurs as counts, ranks or indices.

```
friends <- c(anton = 1,
             berta = 3,
             carlo = 2)
order(friends)
```

```
## [1] 1 3 2
```

The usual arithmetic operations apply, although the result of operation may no longer be `integer`, but `numeric`

```
N <- 3
sum_of_scores <- 32
mean_score <- 32/3
mean_score
```

```
## [1] 10.7
```

```
class(mean_score)
```

```
## [1] "numeric"
```

Surprisingly, logical values can be used in arithmetic expressions, too. When R encounters value `TRUE` in an arithmetic context, it replaces it with 1, zero otherwise. Used with multiplication, this acts like an on/off switch, which we will put to use for building factorial models.

```
TRUE + TRUE
```

```
## [1] 2
```

```
sqrt(3 + TRUE)
```

```
## [1] 2
```

```
is_car <- c(TRUE, TRUE, FALSE) ## bicycle otherwise
wheels <- 2 + is_car * 2
wheels
```

```
## [1] 4 4 2
```

Data sets usually contain variables that are not numeric, but partition the data into groups. For example, we frequently group observations by the following

- `Part`: participant
- `Condition`: experimental condition
- `Design`: one of several designs
- `Education`: level of education (e.g., low, middle or high)

Two object types apply for grouping observations: *factor* and *character*. While factors specialize on grouping observations, character objects can also be used to store longer text, say the description of a usability problem. The following identifies two conditions in a study, say a comparison of designs A and B. Note how the factor identifies its *levels* when called to the screen:

```
design_char <- c("A", "B", "B", "A")
design_char
```

```
## [1] "A" "B" "B" "A"
```

```
design_fact <- factor(c("A", "B", "B", "A"))
design_fact
```

```
## [1] A B B A
## Levels: A B
```

Statistical analyses deal with real world data which ever so often is messy. Frequently, a planned observation could not be recorded, because the participant decided to quit or the equipment did not work properly or the internet collapsed. Users of certain legacy statistics packages got used to coding missing observations as `-999` and then declared this a missing value. In R *missing values are first-class citizens*. Every vector of a certain class can contain missing values, which are identified as `NA`.

Most basic statistics functions, like `mean()`, `sd()` or `median()` are act conservatively when the data contains missing values. If there is a single `NA` in the variable to be summarized, the result is `NA`. While this is good in the sense of transparency, much of the time what the researcher wants is to have the summary statistic with `NA` values being removed, first.

```
clicks <- c(3, NA, 2)
mean(clicks)
```

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

```
mean(clicks, na.rm = T)
```

```
## [1] 2.5
```

This book is about programming and statistics at the same time. Unfortunately, there are a few terms that have a particular meaning in both domains. One of those is a "variable". In statistics, a variable usually is a property we have recorded, say the body length of persons, or their gender. In general programming, a variable is a space in the computers memories, where results can be stored and recalled. Fortunately, R avoids any confusion and calls *objects* what is usually called a programming variable.

2.2.4 Operators and functions

R comes with a full bunch of functions for creating and summarizing data. Let me first introduce you to functions that produce exactly one number to characterize a vector. The following functions do that with to the vector `X`, every in their own way:

```
length(X)
sum(X)
mean(X)
var(X)
sd(X)
min(X)
max(X)
median(X)
```

These functions are a transformation of data. The input to these transformations is `X` and is given as an *argument* to the function. Other functions require more than one argument. The `quantile` function is routinely used to summarize a variable. Recall that `X` has been drawn from a Normal distribution of $\mu = 2$ and standard deviation $\sigma = 1$. All Normal distributions have the property that about 66% of the mass is within the range of $\mu - \sigma$ and $\mu + \sigma$. That means in turn: 17% are below $\mu - \sigma$ and $66 + 17 = 83\%$ are *below* $\mu + \sigma$. The number of observations in a certain range of values is called a *quantile*. The `quantile` function operates on X, but takes an (optional) vector of quantiles as second argument:

```
quantile(X, c(.17, .83))
```

```
##  17%  83%
## 1.07 3.02
```

Most functions in R have *optional arguments* that let you change how the function performs. The basic mathematical functions all have the optional argument `na.rm`. This is a switch that determines how the function deals with missing values `NA`. Many optional arguments have *defaults*. The default of `na.rm` is `FALSE` ("return NA in case of NAs in the vector"). By setting it to `TRUE`, they are removed before operation.

```
B <- c(1, 2, NA, 3)
mean(B)
```

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

```
mean(B, na.rm = TRUE)
```

```
## [1] 2
```

Most more complex routines in R have an abundance of parameters, most of which have reasonable defaults, fortunately. To give a more complex example, the first call of `stan_glm` performs a Bayesian estimation of the grand mean model. The second does a Poisson grand mean model with 5000 iterations per MCMC chain. As `seed` has been fixed, every subsequent run will produce the exact same chains. My apologies for the jargon!

```
D_1 = tibble(X = rnorm(20, 2, 1))
M_1 = stan_glm(X ~ 1,
               data = D_1)

D_2 = tibble(X = rpois(20, 2))
M_2 = stan_glm(X ~ 1,
               family = poisson,
               seed = 42,
               iter = 5000,
               data = D_1)
```

R brings the usual set of arithmetic operators, like `+`, `-`, `*`, `/` and more. In fact, an operator is just a function. The sum of two numbers can, indeed, be written in these two ways:

```
1 + 2
```

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

```
`+`(1,2)
```

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

The second term is a function that takes two numbers as input and returns a third. It is just a different syntax, and this one is called the *polish notation*. I will never use it throughout the rest of this book.

Another set of commonly used operators are logical, they implement *Boolean algebra*. Some common Boolean operators are shown in the truth table:

A	B	!A	A & B	A | B	A == B
		not	and	or	equals
T	T	F	T	T	T
T	F	F	F	T	F
F	T	T	F	T	F
F	F	T	F	F	T

Be careful not to confuse Boolean "and" and "or" with their common natural language use. If you ask: "Can you buy apples *or* pears on the market?", the natural answer would be: "both". The Boolean answer is: TRUE. In a requirements document you could state "This app is for children and adults". In Boolean the answer would be FALSE, because no one can be a child and an adult at the same time, strictly. A correct Boolean statement would be: "The envisioned users can be adult or child".

Further Boolean operators exist, but can be derived from the three above. For example, the exclusive OR, "either A or B" can be written as: `(A | B) & !(A & B)`. This term only gets TRUE when A or B is TRUE, but not both. In the data analysis workflow, Boolean logic is frequently used for filtering data and we re-encounter them in data transformation.

Finally, it sometimes is convenient or necessary to program own functions. A full coverage of developing functions is beyond the scope of this introduction, so I show just one simple example. If one desires a more convenient function to compute the mean that ignore missing values by default, this can be constructed as follows:

```r
mean_conv <- function(x) {
  mean(x = B, na.rm = TRUE)
}

mean_conv(B)
```

```
## [1] 2
```

Notice that:

- the `function()` function creates new functions
- the arguments given to `function(x)` will be the arguments expected by the function `mean_conv(x)`.
- code enclosed by

More examples of creating basic functions can be found in section @ref(descriptive_stats). As R is a functional programming language, it offers very elaborate ways of programming functions, way beyond what is found in common languages, such as Python or Java. An advanced example is given in section @ref(forecasting_LOO).

2.2.5 Storing data in data frames

Most behavioral research collects *real data* to work with. As behavior researchers are obsessed about finding associations between variables, real data usually contains several. If you have a sample of observations (e.g. participants) and every case has the same variables (measures or groups), data is stored in a table structure, where columns are variables and rows are observations.

R knows the `data.frame` objects to store variable-by-observation tables. Data frames are tables, where columns represent statistical variables. Variables have names and can be of different data types, as they usually appear in empirical research. In many cases data frames are imported to R, as they represent real data. Here, we first see how to create data frames by simulation. First, we usually want some initial inspection of a freshly harvested data frame.

Several commands are available to look into a data frame from different perspectives. Another command that is implemented for a variety of classes, ibluding dara frames, is `summary`. For all data frames, it produces an overview with descriptive statistics for all variables (i.e. columns), matching their object type. Particularly useful for data initial screening, is that missing values are listed per variable.

```
print(summary(Experiment))
```

```
##       Obs           Group                age          outcome
##  Min.   :  1.0   Length:100         Min.   :18.0   Min.   :189
##  1st Qu.: 25.8   Class :character   1st Qu.:22.0   1st Qu.:213
##  Median : 50.5   Mode  :character   Median :26.0   Median :235
##  Mean   : 50.5                      Mean   :26.2   Mean   :232
##  3rd Qu.: 75.2                      3rd Qu.:31.0   3rd Qu.:253
##  Max.   :100.0                      Max.   :35.0   Max.   :275
```

The `str` (structure) command works on any R object and displays the hierarchical structure (if there is one):

```
str(Experiment)
```

```
## Classes 'tbl_obs', 'tbl_df', 'tbl' and 'data.frame': 100 obs. of  4 variables:
##  $ Obs    : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Group  : chr  "control" "experimental" "control" "experimental" ...
##  $ age    : num  23 34 20 22 28 22 26 24 30 23 ...
##  $ outcome: num  195 260 214 254 203 ...
```

Data frames store variables, but statistical procedures operate on variables. We need ways of accessing and manipulating statistical variables and we will have plenty. First, recall that in R the basic object types are all vectors. You can store as many elements as you want in an object, as long as they are of the same class.

Internally, data frames are a collection of "vertical" vectors that are equally long. Being a collection of vectors, the variables of a data frame can be of different classes, like `character`, `factor` or `numeric`. In the most basic case, you want to calculate a statistic for a single variable out of a data frame. The `$` operator pulls the variable out as a vector:

```
mean(Experiment$outcome)
```

```
## [1] 232
```

As data frames are rectangular structures, you can also access individual values by their addresses. The following commands call

- the first *outcome* measure
- the first to third elements of *Group*
- the complete first row

```
Experiment[  1, 3]
Experiment[1:3, 2]
Experiment[  1,  ]
```

Addressing one or more elements in square brackets, always requires two elements, first the row, second the column. As odd as it looks, one or both elements can be empty, which just means: get all rows (or all columns). Even the expression `Experiment[,]` is fully valid and will just the return the whole data frame.

There is an important difference, however, when using R's classic `data.frame` as compared to dplyr's `tibble`implementation: When using single square brackets on dplyr data frames one always gets a data frame back. That is a very predictable behavior, and very much unlike the classic: with `data.frame`, when the addressed elements expand over multiple columns, like `Experiment[, 1:2]`, the result will be a `data.frame` object, too. However, when slicing a single column, the result is a vector:

```
Exp_classic <- as.data.frame(Experiment)
Exp_classic[1:2,1:2]  ## data.frame
```

Obs	Group
1	control
2	experimental

```
Exp_classic[1,]       ## data.frame
```

Obs	Group	age	outcome
1	control	23	195

```
Exp_classic[,1]       ## vector
```

```
##   [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
##  [19]  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36
##  [37]  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54
##  [55]  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72
##  [73]  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90
##  [91]  91  92  93  94  95  96  97  98  99 100
```

Predictability and a few other useful tweaks made me prefer `tibble` over `data.frame`. But, many third-party packages continue to produce classic `data.frame` objects. For example, there is an alternative to package '`readxl`, `openxlsx`, which reads (and writes) Excel files. It returns classic data.frames, which can easily be converted as follows:

```
D_foo <-
  read.xlsx("foo.xlsx") %>%
  as_tibble()
```

While `tibble[]` behaves perfectly predictable, in that it always returns a data frame, even when this is just a single column or cell. Sometimes, one wants to truly extract a vector. With a `tibble` a single column can be extracted as a vector, using double square brackets, or using the `$` operator.

```
Experiment[[1]]       ## vector
Experiment$Group      ## the same
```

Sometimes, it may be necessary to change values in a data frame. For example, a few outliers have been discovered during data screening, and the researcher decides to mark them as missing values. The syntax for indexing elements in a data frame can be used in conjunction with the assignment operator `<-`. In the example below, we make the simulated experiment more realistic by injecting a few outliers. Then we discard these outliers by setting them all to `NA`.

```
## injecting
Experiment[2,      "outcome"] <- 660
Experiment[6,      "outcome"] <- 987
Experiment[c(1,3), "age"]     <- -99

## printing first few observations
head(Experiment)

## setting to NA
Experiment[c(2, 6),"outcome"] <- NA
Experiment[c(1, 3),"age"]     <- NA
```

Besides the injection, note two more features of addressing data frame elements. The first is, that vectors can be used to address multiple rows, e.g. 2 and 6. In fact, the range operator `1:3` we used above is just a convenient way of creating a vector `c(1,2,3)`. Although not shown in the example, this works for columns alike.

The careful reader may also have noted another oddity in the above example. With `Experiment[c(2, 6),"outcome"]` we addressed two elements, but right-hand side of `<-` is only one value. That is a basic mechanism of R, called *reuse*. When the left-hand side is longer than the right-hand side, the right-hand side is reused as many times as needed. Many basic functions in R work like this, and it can be quite useful. For example, you may want to create

a vector of 20 random numbers, where one half has a different mean as the second half of observations.

```
rnorm(20, mean = c(1,2), sd = 1)
```

The above example reuses the two mean values 50 times, creating an alternating pattern. Strictly speaking, the `sd = 1` parameter is reused, too, a 100 times. While reuse often comes in convenient, it can also lead to difficult programming errors. So, it is a good advice to be aware of this mechanism and always carefully check the input to vectorized functions.

2.2.6 Import, export and archiving

R lets you import data from almost every conceivable source, given that you have installed and loaded the appropriate packages (foreign, haven or openxlsx for Excel files). Besides that R has its own file format for storing data, which is *.Rda* files. With these files you can save data frames (and any other object in R), using the `save(data, file = "some_file.Rda")` and `load(file = "some_file.Rda")` commands.

Few people create their data tables directly in R, but have legacy data sets in Excel (*.xslx*) and SPSS files (*.sav*). Moreover, the data can be produced by electronic measurement devices (e.g. electrodermal response measures) or programmed experiments can provide data in different forms, for example as *.csv* (comma-separated-values) files. All these files can be opened by the following commands:

```
## Text files
Experiment <-
  read_csv("Data/Experiment.csv")

## Excel
Experiment <-
  read_excel("Data/Experiment.xlsx", sheet = 1)

## SPSS (haven)
Experiment <-
  read_sav("Data/Experiment.sav")

## SPSS (foreign)
Experiment <-
  read.spss("Data/Experiment.sav", to.data.frame = TRUE)
```

Note that I gave two options for reading SPSS files. The first (with an underscore) is from the newer haven package (part of tidyverse). With some SPSS

files, I experienced problems with this command, as it does not convert SPSS's data type *labelled* (which is almost the same as an R factor). The alternative is the classic `read.spss` command which works almost always, but as a default it creates a list of lists, which is not what you typically want. With the extra argument, as shown, it behaves as expected.

Remember, data frames are objects and volatile as such. If you leave your session, they are gone. Once you have you data frame imported and cleaned, you may want to store it to a file. Like for reading, many commands are available for writing all kinds of data formats. If you are lucky to have a complete R-based workflow, you can conveniently use R's own format for storing data, `Rdata` files. For storing a data frame and then reading it back in (in your next session), simply do:

```
save(Experiment, file = "Data/Experiment.Rda")
load(file = "Data/Experiment.Rda")
```

Note that with `save` and `load` all objects are restored by their original names, without using any assignments. Take care, as this will not overwrite any object with the same name. Another issue is that for the `save` command you have to explicitly refer to the `file` argument and provide the file path as a character object. In Rstudio, begin typing `file=""`, put the cursor between the quotation marks and hit `Tab`, which opens a small dialogue for navigation to the desired directory.

Once you have loaded, prepared and started to analyze a data frame in R, there is little reason to go back to any legacy program. Still, the `haven` and `foreign` packages contain commands to write to various file formats. I'll keep that as an exercise to the reader.

2.2.7 Case environments

This book features more than a dozen case studies. Every case will be encountered several times and multiple objects are created along the way: data sets, regressions, graphics, tables, you name it. That posed the problem of naming the objects, so that they are unique. I could have chosen object names, like: `BrowsingAB_M_1`, `AUP_M_1`, etc. But, this is not what you normally would do, when working on one study at a time. Moreover, every letter you add to a line of code makes it more prone to errors and less likely that you, dear reader, are typing it in and trying it out yourself.

For these reasons, all cases are enclosed in *case environments* and provided with this book. For getting a case environment to work in your session, it has to be loaded from the respective R data file first:

```
load("BrowsingAB.Rda")
```

In R, *environments* are containers for collections of objects. If an object `BAB1` is placed in an environment `BrowsingAB`, it can be called as `BrowsingAB$BAB1`. This way, no brevity is gained. Another way to assess objects in an environment is to *attach the environment first* as:

```
attach(BrowsingAB)
BAB1 %>% as_tbl_obs()
detach(BrowsingAB)

attach(Reading)
D_1 %>% as_tbl_obs()
```

Calling `attach` gets you into the *namespace* of the environment (formally correct: the namespace gets imported to your working environment). All objects in that namespace become immediately visible by their mere name. The `detach` command leaves the environment, again. When working with the case environments, I strongly recommend to detach before attaching another environment.

All case environments provided with this book contain one or more data sets. Many of the cases are synthetic data which has been generated by a simulation function. This function, routinely called `simulate`, is provided with the case environment, too. That gives you the freedom to produce your own data sets with the same structure, but different effects. Generally, calling the simulation function without any further arguments, exactly reproduces the synthetic data set provided with the case environment.

```
simulate()  %>% as_tbl_obs() ## reproduces the data frame D_1
simulate(N = 6)  %>% as_tbl_obs() ## simulates first 6 participants only
```

Furthermore, once you delve deeper into R, you can critically inspect the simulation function's code for its behavioral and psychological assumptions (working through the later chapters on data management and simulation will help).

```
simulate ## calling a function without parantheses prints its code
```

```
## function(N = 40,
##             beta = c(Intercpt = 60,
##                      fnt_size_12 = -12,
##                      fnt_color_blk = -10,
##                      ia_blk_12 = 8),
```

```
##             sigma = 5,
##             seed = 42)
##     {
##     set.seed(seed)
##     out <-
##       tibble(Part = 1:N,
##                  font_size = factor(rep(c(1, 2), N/2),
##                                     levels = c(1,2),
##                                     labels = c("10pt", "12pt")),
##                  font_color = factor(c(rep(1, N/2), rep(2, N/2)),
##                                      levels = c(1,2),
##                                      labels = c("gray", "black"))) %>%
##        mutate( mu = beta[1] +
##                  beta[2] * (font_size == "12pt") +
##                  beta[3] * (font_color == "black") +
##                  beta[4] * (font_color == "black") * (font_size == "12pt"),
##                ToT = rnorm(N,  mu, sigma)) %>%
##     as_tbl_obs()
##
##     #class(out) <- append(class(out), "sim_tbl")
##     attr(out, "coef") <- list(beta = beta,
##                               sigma = sigma)
##     attr(out, "seed") <- seed
##
##     out %>% as_tbl_obs()
##     }
## <bytecode: 0x000000002dccba00>
```

```
detach(Reading)
```

Finally, the case environments contain all objects that have been created throughout this book. This is especially useful for the regression models, as fitting these can take from a few seconds to hours.

Note that working with environments is a tricky business. Creating these case environments in an efficient way was more difficult than you may think. Therefore, I do *not* recommend using environments in everday data analysis, with one exception: at any moment the current working environment contains all objects that have been created, so far. That is precisely the set of objects shown in the Environment pane of Rstudio (or call `ls()` for a listing). Saving all objects and retrieving them when returning from a break is as easy as:

```
save(file = "my_data_analysis.Rda")
## have a break
load("my_data_analysis.Rda")
```

Next to that, Rstudio can be configured to save the current workspace on exit and reload it on the next start. When working on just one data analysis for a longer period of time, this can be a good choice.

2.2.8 Structuring data

In the whole book, data sets are structured according to the rule *one-row-per-observation of the dependent variable* (the ORPO rule). Many researchers still organize their data tables as one-row-per-participant, as is requested by some legacy statistics programs. This is fine in research with non-repeated measures, but will not function properly with modern regression models, like linear mixed-effects models. Consider a study where two designs were evaluated by three participants using a self-report item, like "how easy to use is the interface?" Then, the *wrong way* of structuring the data would be:

```
ORPO %>%
  filter(Task == 1, Item == 1) %>%
  mascutils::discard_redundant() %>%
  spread(Design, response) %>%
  as_tbl_obs()
```

The correct way of setting up the data frame is:

```
ORPO %>%
  filter(Task == 1, Item == 1) %>%
  mascutils::discard_redundant()
```

Part	Design	response
1	A	4
2	A	4
3	A	6
1	B	1
2	B	4
3	B	3

The ORPO rule dictates another principle: every row should have a unique identifier, which can be a combination of values. In the example above, every observation is uniquely identified by the participant identifier and the design condition. If we extend the example slightly, it is immediatly apparent, why the ORPO rule is justified. Imagine, the study actually asked three partcipants to rate two different tasks on two different designs by three self-report items. By the ORPO rule, we can easily extend the data frame as below (showing a random selection of rows). I leave it up to the reader to figure out how to press such a data set in the wide legacy format.

Using identifiers is good practice for several reasons. First, it reduces problems during manual data entry. Second, it allows to efficiently record data in multi-method experiments and join them automatically. An exammple is given in Appendix A.1.8. Lastly, the identifiers will become statistically interesting by themselves when we turn to linear mixed-effects models and the notion of *members of a population* 5.5. Throughout the book I will use standard names for recurring identifier variables in design research:

- Part
- Design
- Item
- Task

Note that usually these entities get numerical identifiers, but these numbers are just labels. Throughout, variables are written Uppercase when they are entities, but not real numbers. An exception is the trial order in experiments with massive repeated measures. These get a numerical type to allow exploring effects over time such as learning, training and fatigue.

```
ORPO %>%
  sample_n(6) %>%
  arrange(Part, Task, Design, Item)
```

Part	Task	Design	Item	response
1	1	B	1	1
1	2	B	1	6
2	1	B	3	5
2	2	B	2	3
3	2	A	2	6
3	2	B	2	4

2.2.9 Data transformation

Do you wonder about the strange use of `%>%` in my code above? This is the tidy way of programming data transformations in R.

The so-called magritte operator `%>%` is part of the *dplyr/tidyr* framework for data manipulation. It chains steps of data manipulation by connecting transformation functions, also called piping. In the following, we will first see a few basic examples. Later, we will proceed to longer transformation chains and see how graceful dplyr piping is, compared to the classic data transformation syntax in R.

Importing data from any of the possible resources, will typically give a data frame. However, often the researcher wants to *select* or *rename* variables in the

data frame. Say, you want the variable *Group* to be called *Condition*, omit the variable *age* and store the new data frame as *Exp*. The `select` command does all this. In the following code the data frame `Experiment` is piped into `select`. The variable *Condition* is renamed to *Group*, and the variable *outcome* is taken as-is. All other variables are discarded.

```
Exp <-  Experiment %>%
  select(Condition = Group, outcome) %>% as_tbl_obs()
```

Another frequent step in data analysis is cleaning the data from missing values and outliers. In the following code example, we first "inject" a few missing values for age (which were coded as -99) and outliers (>500) in the outcome variable. Note that I am using some R commands that you don't need to understand by now. Then we reuse the above code for renaming (this time keeping *age* on board) and add some filtering steps:

```
## rename, then filtering
Exp <-
  Experiment %>%
  select(Condition = Group, age, outcome) %>%
  filter(outcome < 500) %>%
  filter(age != -99)

Exp %>% as_tbl_obs()
```

During data preparation and analysis, new variables are created routinely. For example, the covariate is often shifted to the center before using linear regression:

```
mean_age = mean(Exp$age)
Exp <- Exp %>%
  mutate(age_cntrd = age - mean_age)
Exp %>% as_tbl_obs()
```

Finally, for the descriptive statistics part of your report, you probably want to summarize the outcome variable per experimental condition. The following chain of commands first groups the data frame, then computes means and standard deviations. At every step, a data frame is piped into another command, which processes the data frame and outputs a data frame.

```
Exp %>%
  group_by(Condition) %>%
  summarize(mean = mean(outcome),
            sd = sd(outcome) )
```

Condition	mean	sd
control	211	9.25
experimental	253	8.66

2.2.10 Plotting data

Good statistical graphics can vastly improve your and your readers understanding of data and results. This book exclusively introduces the modern ggplot2 graphics system of R, which is based on the grammar of graphics.

Every plot starts with piping a data frame into the `ggplot(aes(...))` command. The `aes(...)` argument of ggplot creates the *aesthetics*, which is a *mapping between variables and features of the plot* (and only remotely has something to do with beauty). Review the code once again that produces the piles-of-sugar: the aesthetics map the variable *Group* on the fill color, whereas *outcome* is mapped to the x axis. For a plot to be valid, there must at least one layer with a *geometry*. The above example uses the density geometry, which calculates the density and maps it to the y axis.

The ggplot2 plotting system knows a full set of geometries, like:

- scatter plots with `geom_point()`
- smooth line plots with `geom_smooth()`
- histograms with `geom_histogram()`
- box plots with `geom_boxplot()` and
- my personal favorite: horizontal density diagrams with `geom_violin()`

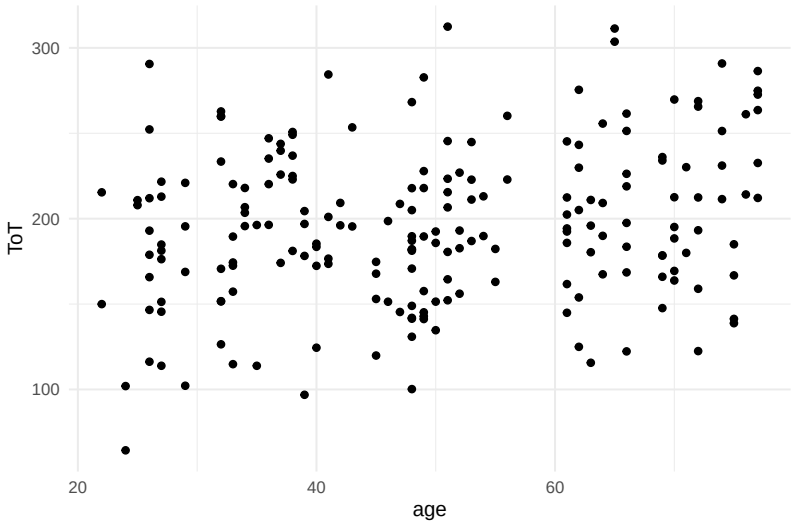
For a brief demonstration of ggplots basic functionality, we use the `BAB1` data set of the BrowsingAB case. We attach the case environment and use the `str` command to take a first look at the data:

```
attach(BrowsingAB)
BAB1 %>% as_tbl_obs()
```

The BrowsingAB case is a virtual series of studies, where two websites were compared by how long it takes users to complete a given task, time-on-task (ToT). Besides the design factor, a number of additional variables exist, that could possibly play a role for ToT, too. We explore the data set with ggplot:

We begin with a plot that shows the association between age of the participant and ToT. Both variables are metric and suggest themselves to be put on a 2D plane, with coordinates x and y, a *scatter plot*.

```
BAB1 %>%
  ggplot(aes(x = age,  y = ToT)) +
  geom_point()
```



Let's take a look at the elements of the command chain: The first two lines pipe the data frame into the ggplot engine.

```
BAB1 %>%
  ggplot(...)
```

At that moment, the ggplot engine "knows" which variables the data frame contains and hence are available for the plot. It does not yet know, which variables are being used, and how. The next step is, usually, to consider a basic (there exist more than 30) *geometry* and put it on a *layer*. The scatter plot geometry of ggplot is `geom_point`:

```
BAB1 %>%
  ggplot(...) +
  geom_point()
```

The last step is the *aesthetic mapping*, which tells ggplot the variables to use and how to map them to *aesthetic* properties of the geometry. The basic properties of points in a coordinate system are the x and y-positions:

```
BAB1 %>%
  ggplot(aes(x = age, y = ToT)) +
  geom_point()
```

The function `aes` creates a mapping where the aesthetics per variable are given. When call `aes` directly, we see that it is just a table.

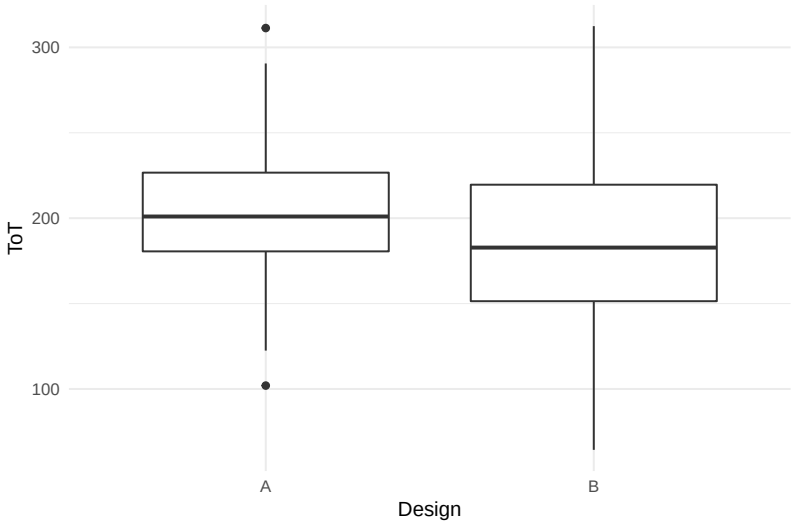
```
aes(x = age, y = ToT)
```

```
## Aesthetic mapping:
## * `x` -> `age`
## * `y` -> `ToT`
```

One tiny detail in the above chain has not yet been explained: the `+`. When choosing the geometry, you actually *add a layer* to the plot. This is, of course, not the literal mathematical sum. Technically, what the author of the ggplot2 package did, was to *overload* the `+` operator. A large set of ggplot functions can be combined in a myriad of ways, just using `+`. The overloaded `+` in ggplot is a brilliant analogy: you can infinitely chain ggplot functions, like you can create long sums. You can store ggplot object and later modify it by adding functions. The analogy has its limits, though: other than sums, order matters in ggplot combinations: the first in the chain is always `ggplot` and layers are drawn upon each other.

Let's move on with a slightly different situation that will result in a different geometry. Say, we are interested in the distribution of the time-on-task measures under the two designs. We need a geometry, that visualizes the distribution of quantitative variables split by a grouping variable, factor. The box plot does the job:

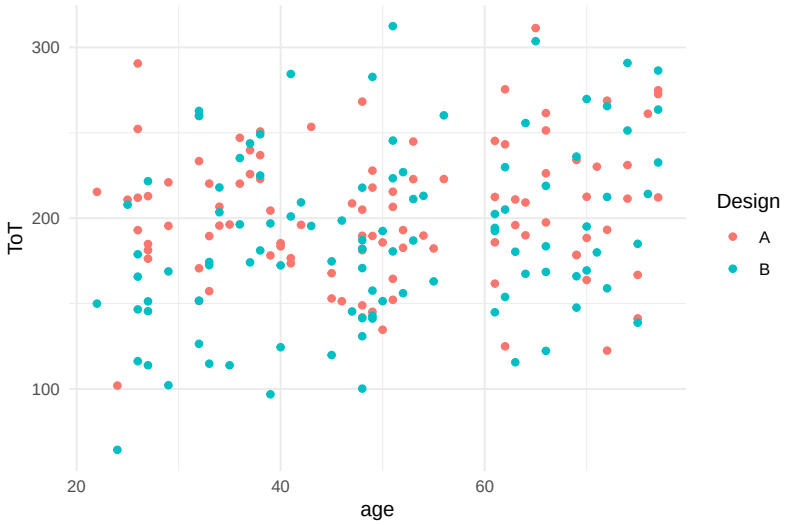
```
BAB1 %>%
  ggplot(aes(x = Design,  y = ToT)) +
  geom_boxplot()
```



The box plot maps ToT to y (again). The factor Design is represented as a split on the x-axis. Interestingly, the box plot does not represent the data as raw as in the scatter plot example. The geometry actually performs an analysis on ToT, which produces five statistics: min, first quartile, median, third quartile and max. These statistics define the vertical positions of bars and end points.

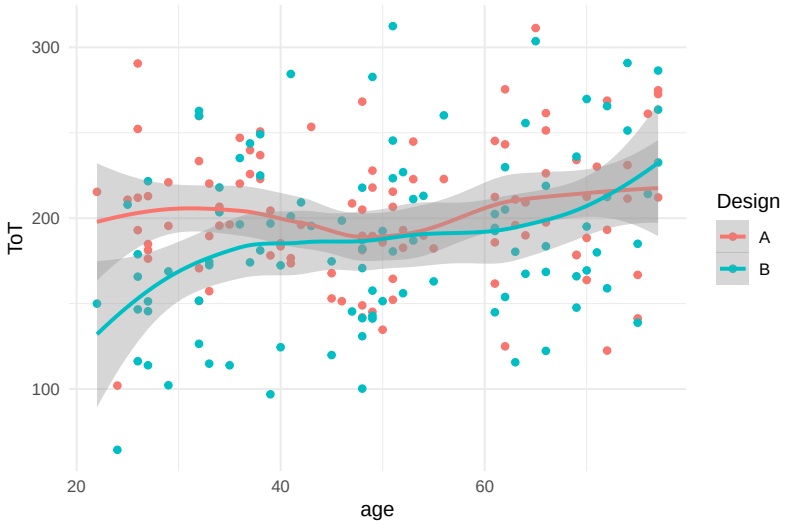
Now, we combine all three variables in one plot: how does the association between ToT and age differ by design? As we have two quantitative variables, we stay with the scatter plot for now. As we intend to separate the groups, we need a property of points to distinguish them. Points offer several additional aesthetics, such as color, size and shape. We choose color, and add it to the aesthetic mapping by `aes`. Note, that it does not matter whether you use the British or American way of writing (colour vs. color).

```
BAB1 %>%
  ggplot(aes(x = age,  y = ToT, color = Design)) +
  geom_point()
```



Now, we can distinguish the groups visually, but there is too much clutter to discover any relation. With the box plot we saw that some geometries do not represent the raw data, but summaries (statistics) of data. For scatter plots, a geometry that does the job of summarizing the trend is `geom_smooth`. This geometry summarizes a cloud of points by drawing a LOESS-smooth line through it. Note how the color mapping is applied to all geometry layers.

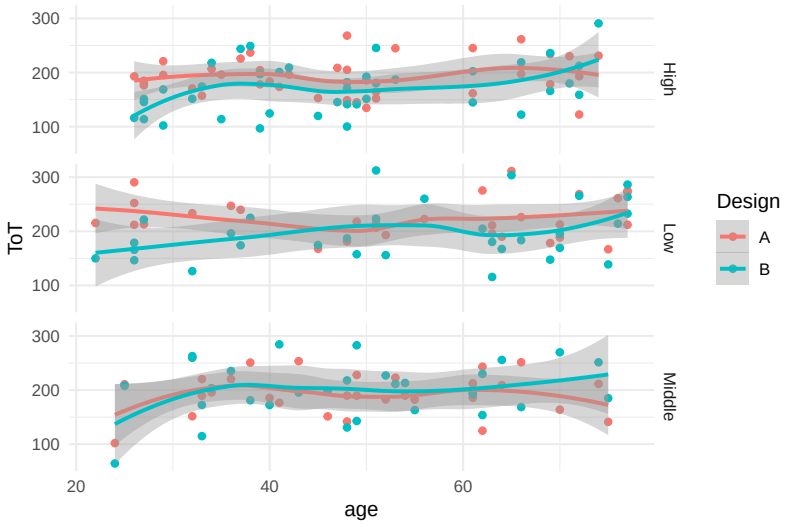
```
BAB1 %>%
  ggplot(aes(x = age,  y = ToT, color = Design)) +
  geom_point() +
  geom_smooth()
```



We see a highly interesting pattern: the association between age and ToT follows two slightly different mirrored sigmoid curves.

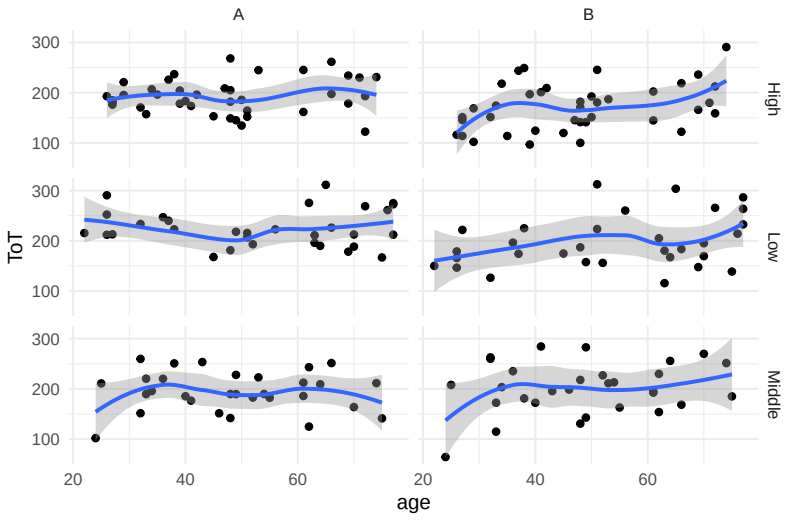
Now that we have represented three variables with properties of geometries, what if we wanted to add a fourth one, say education level? Formally, we could use another aesthetic, say shape of points, to represent it. You can easily imagine that this would no longer result in a clear visual figure. For situations, where there are many factors, or factors with many levels, it is impossible to reasonably represent them in one plot. The alternative is to use *facetting*. A facet splits the data by a grouping variable and creates one single plot for every group:

```
BAB1 %>%
  ggplot(aes(x = age,  y = ToT, color = Design)) +
  geom_point() +
  geom_smooth() +
  facet_grid(Education ~ .)
```



See, how the `facet_grid` command takes a formula, instead of just a variable name. This makes faceting the primary choice for highly-dimensional situations. For example, we may also choose to represent both factors, Design and education by facets:

```
BAB1 %>%
  ggplot(aes(x = age,  y = ToT)) +
  geom_point() +
  geom_smooth() +
  facet_grid(Education ~ Design)
```



Note how the color aesthetic, although unnecessary, is kept. It is possible to map several aesthetics (or facets) to one variable, but not vice versa.

2.2.11 Fitting regression models

Above we have seen examples of functions that boil down a vector to a single statistic, like the mean. R has several functions that summarize data in a more complex way. One function with a wide range of applications is the `lm` command, that applies regression models to data (provided as data frames).

In the following, we will use another simulated data frame `Exp` to demonstrate linear models. To make this more interesting, we simulate `Exp` in a slightly advanced way, with quantitative associations between variables. Note how the *expected value* μ is created by drawing on the variables `Condition` and `age`. The last step adds (somewhat) realistic noise to the measures, by drawing from the normal distribution with a mean of *mu*.

```
N_Obs <- 20
set.seed(42)
Exp <-
  tibble(Obs = 1:N_Obs,
              Condition = rep(c("Experimental", "Control"),
                              N_Obs/2),
              age = runif(N_Obs, 18, 35),
              mu = 200 + (Condition == "Control") * 50 + age * 1,
              outcome = rnorm(N_Obs, mu, 10))
```

The experiment involves two groups, which in classic statistics would clearly point to what is commonly referred to as *ANOVA*. As it will turn out in 4.3.1, old-fashioned ANOVA can be replaced by a rather simple regression model, that I call comparison of groups model (CGM). The estimation of regression models is done by a *regression engine*, which basically is a (very powerful) R command. The specification for any regression model is given in R's formula language. Learning this formula language is key to unleashing the power of regression models in R. We can perform a CGM on the data frame `Exp` using the regression engine `stan_glm`. The desired model estimates the effect of `Condition` on `outcome`. This produces a regression object that contains an abundance of information, much of it is of little interest for now. (A piece of information, that it does *not* contain is F-statistics and p-values; and that is why it is not an ANOVA, strictly speaking!) The foremost question is how strong the difference between the groups is. The `fixef` command extracts the parameter estimates from the model to answer the question.

```
M_1 <-
  stan_glm(outcome ~ Condition,
     data = Exp)
```

```
fixef(M_1)
```

fixef	center	lower	upper
Intercept	275.4	264.0	286.2
ConditionExperimental	-47.5	-62.6	-31.9

Another classic model is *linear regression*, where outcome is predicted by a metric variable, say `age`. The `stan_glm` regression engine is truly multi-purpose and does the job with grace:

```
M_2 <-
  stan_glm(outcome ~ age,
     data = Exp)
```

```
fixef(M_2)
```

fixef	center	lower	upper
Intercept	219.02	133.51	308.60
age	1.16	-1.98	4.18

If you are interested in both at the same time, you can combine that in one model by the following formula:

```
M_3 <-
  stan_glm(outcome ~ Condition + age,
     data = Exp)
```

```
fixef(M_3)
```

fixef	center	lower	upper
Intercept	219.72	177.280	261.59
ConditionExperimental	-50.35	-63.763	-36.38
age	2.01	0.557	3.52

A statistical model has several components, for example the coefficients and residuals. Models are complex objects, from which a variety of inferences can be made. For example, the coefficient estimates can be extracted and used for prediction. This is what `fixef()` does in the above code.

A number of functions can be used to extract certain aspects of the model. For example:

- `fixef(model)` extracts the linear effects
- `residuals(model)` extracts the measurement errors
- `predict(model)` extracts the expected values

These will all be covered in later chapters.

2.2.12 Knitting statistical reports

As you have seen throughout this chapter, with R you can effectively manage data, create impressively expressive graphics and conveniently estimate statistical models. Then usually comes the painful moment where all this needs to be assembled into a neat report. With R and Rstudio it has never been easier than that. In fact, complete books have been written in R, like the one you are reading.

A *minimal statistical report* contains four elements:

1. a recap of the research question
2. description of how the statistical model relates to the research question
3. a few figures or tables that answers the research question
4. an explanation of the results

Of these four elements, three are pure text. For a minimal report it is a fairly convenient to use a word processor software for the text, craft the figure in R and copy it. One problem with this approach is that a *scrunitable statistical report* contains at least the following *additional* elements:

1. procedures of data preparation (sources, transformations, variable names, outlier removal)
2. data exploration (ranges of variables, outlier discovery, visualizing associations, etc.)
3. model estimation (formula specification, convergence checks)
4. model criticism (normality of residuals, etc.)

In advanced statistical workflows this is then multiplied by the number of models, an iterative selection process. Because it is easy to lie with statistics,

these elements are needed as to build a fundament of credibility. Full transparency is achieved, when another researcher can exactly reproduce all steps of the original analysis. It is obvious that the easiest way to achieve this, is to hand over the full R script.

The most user-friendly way to achieve both, a good looking report and full transparency, is to write a document that contains all before mentioned: text, graphics, tables and R code. In the R environment such mixed documents can be written in the *markdown/knitr* framework.

Markdown implements a simple markup language, with that you can typset simple, structured texts in a plain ASCII editor. Later in the workflow, such a markup document is transformed into one of various output formats, that are rendered by the respective programs, such as Microsoft Word or an HTML browser.

Fig. 2.1. A minimal statistical report in markdown

The above text is an alternation of markup text and *chunks*, those weirdly enclosed pieces of R code. While the text is static, the chunks are processed by the knitr engine, evaluating the enclosed R code and knitting the output into a document. Very conveniently, when the output is a figure, it will be inserted into the document right away. The `kable` command from the knitr package, in turn, produces neatly rendered tables from data frame objects. By default, the R code is shown, too, but that can be customized.

The minimal workflow for statistical reporting with knitr is as follows:

1. Use markdown right away, covering all steps of your data analysis, i.e. a scrutable report. You may even start writing when only one part of your data gathering is completed, because due to the dynamic chunks, updating the report when new data arrives is just a button click away.
2. When the data analysis is complete, compile the scrutable report to Word format
3. Extract the passages, figures and tables for a minimal statistical report. This is your results section.
4. provide the scrutable report as appendix or supplementary material

In the notion of this chapter, this is just to get you started and knitr is so tightly integrated with the Rstudio environment that I don't even bother to explain the commands for knitting a document. Once acquainted with the basics, markdown provides a few additional markup tokens, like footnotes, hyperlinks or including images. The customization options and addons for knitr are almost endless and various interesting addons are available, just to mention two:

1. The bookdown package provides an infrastructure for writing and publishing longer reports and books.
2. With the shiny package one can add dynamic widgets to HTML reports. Think of a case, where your statistical model is more complicated than a linear regression line or a few group means, say you are estimating a polynomial model or a learning curve. Then, with a simple shiny app, you can enable your readers to understand the model by playful exploration.

2.2.13 Exercises

1. In the book package (directory `/Data`) you will find the data set of the (virtual) study BAB1, which we will be using in coming chapters. This data comes as comma-separated value file with the file ending `.csv`. Load this file into R using the `read_csv` command and check the dataframe.
2. Find the documentation of the packages `haven` and `readr`. Find two import functions. The one that is most useful for you and the one that you consider most exotic.
3. We have seen how to extract a data frame column as a vector using the double square brackets. There seems to be no such option to extract an individual row as a vector. Why? (Think about object types). <–! #14 –>
4. Use the world wide web to find geometries that are useful for plotting associations between grouping variables (factors) and a metric variables. Try them all on the BAB1 data frame. Compare the geometries on what properties of the data they convey.
5. Like data frames and regression results, plots produced by ggplot are complex objects, too. Create an arbitrary plot, store it in a variable and inspect it using `class`, `summary` and `str`. In addition, what happens when you assign the plot to a variable and what happens when you call the variable?
6. Revisit the python-swallowed-camel plot and check out how the aesthetic mapping is created. The plot uses a density geometry. Change it into a histogram. Then produce a box plot that shows the two conditions (think carefully about the mappings of x and y).

7. Use the data set BAB5 in BrowsingAB. It contains a follow-up experiment, where participants had to do five different tasks on the website. Plot the association between age and ToT by task, using color. Then put Task on a facet grid, and use color to represent Design again.
8. Use the data set BAB5 in BrowsingAB. Using a transformation chain, take the sum or average of participants' ToT. Then run a few simple regression models.
9. Use your own data. Drop an Excel and/or an SPSS file into the same directory as your current R file. Read the data into a data frame and summarize what is in the data frame. Use ggplot to create one or more exploratory graphs. Then use dplyr `summarize` to create summary statistics.
10. Do a full exploratory analysis of the dataframe D_agg in case environment IPump. It has the predictors Design, Group, Education and experience. It has the outcome variables ToT, deviations and workload.
    1. Get the data frame into R and produce a summary.
    2. Plot a histogram for all dependent variables.
    3. Produce a table that counts the number of observations per Education(al level).
    4. Produce a table that displays minimum, maximum, median, mean and standard deviation of experience.
    5. Exclude participants with less than four years of experience.
    6. Produce a table with group means per Design and Session.
    7. For every outcome variable, produce a plot for Design by session.
    8. Explore graphically, what other relations may exist between outcome variables and Group, Education and experience.
    9. Run a regression model for ToT with Design and Session as predictors.
    10. Produce a coefficient table.
    11. Plot the residuals.

2.2.14 Bibliographic notes

R for Data Science is a book co-authored by Hadley "Tidy" Wickham.

ggplot2 Version of Figures in "25 Recipes for Getting Started with R" for readers who are familiar with the legacy plotting commands in R.

Introduction to dplyr for Faster Data Manipulation in R introduces dplyr, the next generation R interface for data manipulation, which is used extensively in this book.

Quick-R is a comprehensive introduction to many common statistical techniques with R.

Code as manuscript features a small set of lessons with code examples, assignments and further resources. For if you are in a haste.

bookdown: Authoring Books and Technical Documents with R Markdown fully unleashes the power of knitr for writing and publishing longer reports and books.

3

Elements of Bayesian statistics

As human beings we make our decisions on what has happened to us in the past. For example, we trust a person or a company more, when we can look back at a series of successful transactions. And we have remarkable capability to recall what has just happened, but also what happened yesterday or years ago. By integrating over all the evidence, we form a view of the world we forage. When evidence is abundant, we vigorously experience a feeling of certainty, or lack of doubt. That is not to deny, that in a variety of situations, the boundedness of the human mind kicks in and we become terrible decision makers. This is for a variety of psychological reasons, to name just a few:

- forgetting evidence
- the primacy effect: recent events get more weight
- confirmation bias: evidence that supports a belief is actively sought for, counter-evidence gets ignored.
- the hindsight bias: once a situation has taken a certain outcome, we believe that it had to happen that way.

The very aim of scientific research is to avoid the pitfalls of our minds and act as rational as possible by translating our theory into a formal model, gathering evidence in an unbiased way and weigh the evidence by formal procedures. This weighing of evidence using data essentially is *statistical modeling* and statistical models in this book all produces two sorts of numbers: *magnitude of effects* and *level of certainty*. When a statistic is reported together with the strength of evidence, this is conventionally called an *inferential statistic*. In applied research, real world decisions depend on the evidence, which has two aspects: first, the strength of effects and the level of certainty we have reached.

Bayesian inferential statistics grounds on the idea of accumulating evidence along research. *Certainty* (or *strength of belief* or *credibility* or *credence*) in Bayesian statistics is formalized as a *probability scale (0 = impossible, 1 =*

certain). Certainty is reached by looking at evidence and such evidence comes from two sources: everything that we already know about the subject and the data we just gathered. These sources are only seemingly different, because when new data is analyzed, a transition occurs from what you new before, *prior belief*, to what you know after seeing the data, *posterior belief*. In other words: by data the current belief gets an update.

Updating our beliefs is essential for acting in rational ways. The first section of this chapter is intended to tell the Big Picture. It puts statistics into the context of decision-making in design research. For those readers with a background in statistics, this section may be a sufficient introduction all by itself.

In the remainder of this chapter the essential concepts of statistics and Bayesian analysis will be introduced from ground up. First we will look at descriptive statistics, introducing analysis tools, such as summary statistics and statistical graphs. Descriptive statistics can be used effectively to explore data and prepare the statistical modelling, but they lack one important ingredient: information about uncertainty. For that, a sound understanding is required of what certainty is. In Bayesian thinking, certainty is a sort of probability and section 3.3 first derives probability from set theory and relative frequencies. Then it goes on to explain basic concepts of statistical modeling, such as the likelihood and finishes with Bayes theorem, which does the calculation of the posterior certainty from prior knowledge and data. Despite the matter, I will make minimal use of mathematical formalism. Instead, I use R code as much as possible to illustrate the concepts. If you are not yet familiar with R, you may want to read chapter 2.2 first, or alongside.

Section 3.4 goes into the practical details of modelling. A statistical model is introduced by its two components: the structural part, which typically carries the research question or theory, is only briefly introduced, followed by a rather deep account of the second component of statistical models: the random part.

3.1 Rational decision making in design research

- I see clouds. Should I take my umbrella?
- Should I do this bungee jump? How many people came to death by jumping? (More or less than alpine skiing?) And how much fun is it really?
- Overhauling our company website will cost us EUR 100.000. Is it worth it?

All the above cases are examples of decision making under uncertainty. The actors aim for maximizing their outcome, be it well being, fun or money. But, they are uncertain about what will really happen. And their uncertainty occurs on two levels:

1. One cannot precisely foresee the exact outcome of one's chosen action:

- Taking the umbrella with you can have two consequences: if it rains, you have the benefit of staying dry. If it does not rain, you have the inconvenience of carrying it with you.
- You don't know if you will be the rare unlucky one, who's bungee rope breaks.
- You don't know by how much the new design will attract more visitors and how much the income will raise.

2. It can be difficult to precisely determine the benefits or losses of potential outcomes:

- How much worse is your day when carrying a useless object with you? How much do you hate moisture? In order to compare the two, they must be assessed on the same scale.
- How much fun (or other sources of reward, like social acknowledgments) is it to jump a 120 meter canyon? And how much worth is your own life to you?
- What is the average revenue generated per visit? What is an increase of recurrence rate of, say, 50% worth?

Once you know the probabilities of all outcomes and the respective losses, *decision theory* provides an intuitive framework to estimate these values. *Expected utility* U is the sum product of *outcome probabilities* P and the involved *losses*. In the case of the umbrella, the decision is between two options: taking an umbrella versus taking no umbrella, when it is cloudy. We calculate and compare the expected utilities `U` as follows (P(outcome) is the probability of an outcome):

$$
\begin{array}{lll}
P(\text{rain}) & & =0.6 \\
P(\text{no rain}) & =1-P(\text{rain}) & =0.4 \\
L(\text{carry}) & & =2 \\
L(\text{wet}) & & =4 \\
U(\text{umbrella}) & =P(\text{rain})L(\text{carry})+P(\text{no rain})L(\text{carry})=L(\text{carry}) & =2 \\
U(\text{no umbrella}) & =P(\text{rain})L(\text{wet}) & =2.4
\end{array}
$$

```
attach(Rainfall)

Actions <-
  tibble(action = c("umbrella", "no umbrella"))
Actions
```

action
umbrella
no umbrella

```
Outcomes  <-
  tibble(outcome  = c("rain", "no rain"),
                    prob   = c(0.6, 0.4))
Outcomes
```

outcome	prob
rain	0.6
no rain	0.4

```
Losses  <-
  expand.grid(action = Actions$action, outcome = Outcomes$outcome) %>%
  join(Outcomes) %>%
  mutate(loss = c(2, 4, 2, 0))
Losses
```

action	outcome	prob	loss
umbrella	rain	0.6	2
no umbrella	rain	0.6	4
umbrella	no rain	0.4	2
no umbrella	no rain	0.4	0

```
Utility <-
  Losses %>%
  mutate(conditional_loss = prob * loss) %>%
  group_by(action) %>%
  summarise(expected_loss = sum(conditional_loss))
Utility
```

action	expected_loss
umbrella	2.0
no umbrella	2.4

We conclude that, given the high chance for rain, and the conditional losses, the expected loss is larger for not taking an umbrella with you. It is rational to take an umbrella when it is cloudy.

3.1.1 Measuring uncertainty

As we have seen above, a decision requires two investigations: outcomes and their probabilities, and the assigned loss. Assigning loss to decisions is highly context dependent and often requires domain-specific expertise. The issues of

probabilistic processes and the uncertainty that arises from them is basically what the idea of *New Statistics* represents. We encounter uncertainty in two forms: first, we usually have just a limited set of observations to draw inference from, this is *uncertainty of parameter estimates*. From just 20 days of observation, we cannot be absolutely certain about the true chance of rain. It can be 60%, but also 62% or 56%. Second, even if we precisely knew the chance of rain, it does not mean we could make a certain statement of the future weather conditions, which is *predictive uncertainty*. For a perfect forecast, we had to have a complete and exact figure of the physical properties of the atmosphere, and a fully valid model to predict future states from it. For all non-trivial systems (which excludes living organisms and weather), this is impossible.

Review the rainfall example: the strategy of taking an umbrella with you has proven to be superior under the very assumption of *predictive uncertainty*. As long as you are interested in long-term benefit (i.e. optimizing the average loss on a long series of days), this is the best strategy. This may sound obvious, but it is not. In many cases, where we make decisions under uncertainty, the decision is not part of a homogeneous series. If you are member of a startup team, you only have this one chance to make a fortune. There is not much opportunity to average out a single failure at future occasions. In contrast, the investor, who lends you the money for your endeavor, probably has a series. You and the investor are playing to very different rules. For the investor it is rational to optimize his strategy towards a minimum average loss. The entrepreneur is best advised to keep the maximum possible loss at a minimum.

As we have seen, predictive uncertainty is already embedded in the framework of rational decision making. Some concepts in statistics can be of help here: the uncertainty regarding future events can be quantified (for example, with posterior predictive distributions 4.1.3 and the process of model selection can assist in finding the model that provides the best predictive power.

Still, in our formalization of the Rainfall case, what magically appears are the estimates for the chance of rain. Having these estimates is crucial for finding an optimal decision, but they are created outside of the framework. Furthermore, we pretended to know the chance of rain exactly, which is unrealistic. Estimating parameters from observations is the reign of statistics. From naive calculations, statistical reasoning differs by also regarding uncertainty of estimates. Generally, we aim for making statements of the following form:

"With probability p, the attribute A is of magnitude X."

In the umbrella example above, the magnitude of interest is the chance of rain. It was assumed to be 60%. This appears extremely high for an average day. A more realistic assumption would be that the probability of rainfall is 60% *given the observation of a cloudy sky*. How could we have come to the belief that with 60% chance, it will rain when the sky is cloudy? We have several options, here:

1. Supposed, you know that, on average, it rains 60% of all days, it is a matter of common sense, that the probability of rain must be equal or larger than that, when it's cloudy.
2. You could go and ask a number of experts about the association of clouds and rain.
3. You could do some systematic observations yourself.

Imagine, you have recorded the coincidences of clouds and rainfall over a period, of, let's say, 20 days, with the following observations:

```
Rain %>% as_tbl_obs()
```

Intuitively, you would use the average to estimate the probability of rain under every condition.

```
T_desc <-
  Rain %>%
  group_by(cloudy) %>%
  summarize(chance_of_rain = mean(rain))
```

These probabilities we can feed into the decision framework as outlined above. The problem is, that we obtained just a few observations to infer the magnitude of the parameter $P(rain|cloudy) = 60\%$. Imagine, you would repeat the observation series on another 20 days. Due to random fluctuations, you would get a more or less different series and different estimates for the probability of rain. More generally, the *true* parameter is only imperfectly represented by any sample, it is not unlikely, that it is close to the estimate, but it could be somewhere else, for example, $P(rain|cloudy) = 54.098\%$.

The trust you put in your estimation is called *level of certainty* or *belief* or *confidence*. It is the primary aim of statistics to rationally deal with uncertainty, which involves to *measure the level of certainty* associated with any statement derived from teh data. So, what would be a good way to determine certainty? Think for a moment. If you were asking an expert, how would you do that to learn about magnitude and uncertainty regarding $P(rain|cloudy)$?

Maybe, the conversation would be as follows:

> YOU: What is the chance of rain, when it's cloudy.
>
> EXPERT: Wow, difficult question. I don't have a definite answer.
>
> YOU: Oh, c'mon. I just need a rough answer. Is it more like 50%-ish, or rather 70%-ish.
>
> EXPERT: Hmm, maybe somewhere between 50 and 70%.

YOU: Then, I guess, taking an umbrella with me is the rational choice of action.

Note how the expert gave two endpoints for the parameter in question, to indicate the location and the level of uncertainty. If she had been more certain, she had said "between 55 and 65%. While this is better than nothing, it remains unclear, which level of uncertainty is enclosed. Is the expert 100%, 90% or just 50% sure the true chance is in the interval? Next time, you could ask as follows:

...

EXPERT: Hmm, maybe somewhere between 70-90%

YOU: What do you bet? I'm betting 5 EUR that the true parameter is outside the range you just gave.

EXPERT: I dare you! 95 EUR it's inside!

The expert feels 95% certain, that the parameter in question is in the interval. However, for many questions of interest, we have no expert at hand (or we may not even trust them altogether). Then we proceed with option 3: making our own observations.

```
detach(Rainfall)
```

3.1.2 Benchmarking designs

The most basic decision in practical design research is whether a design fulfills an external criterion. External criteria for human performace in a human-machine system are most common, albeit not abundant, in safety-critical domains.

Consider Jane: she is user experience researcher at the mega-large rent-a-car company smartr.car. Jane was responsible for a overhaul of the customer interface for mobile users. Goal of the redesign was to streamline the user interface, which had grown wild over the years. Early customer studies indicated that the app needed a serious visual de-cluttering and stronger funneling of tasks. 300 person months went into the re-development and the team did well: a recent A/B study had shown that users learned the smartr.car v2.0 fast and could use its functionality very efficiently. Jane's team is prepared for the roll-out, when Marketing comes up with the following request:

Marketing: We want to support market introduction with the following slogan: “rent a car in 99 seconds”.

Jane: Not all users manage a full transaction in that short time. That could be a lie.

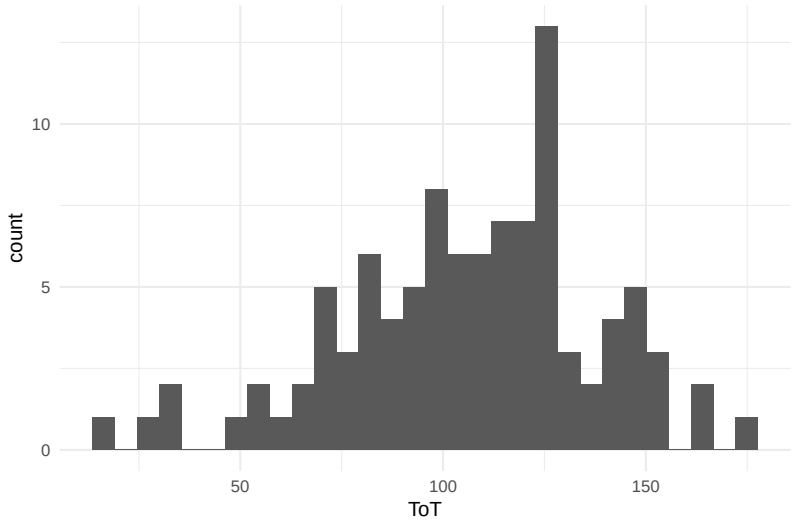
Marketing: Legally, the claim is fine if it holds on average.

Jane: That I can find out for you.

Jane takes another look at the performance of users in the smartr car v2.0 condition. As she understands it, she has to find out whether the average of all recorded time-on-tasks with smartr.car 2.0 is 99 seconds, or better. Here is how the data looks like:

```
attach(Sec99)
```

```
Ver20 %>%
  ggplot(aes(x = ToT)) +
  geom_histogram()
```



The performance is not completely off the 99 seconds, many users are even faster. Jane figures out that she has to ask a more precise question, first, as teh slogan can mean different things, like:

- all users can do it within 99 seconds
- at least one user can do it

- half of the users can do it

Jane decides to go the middle way and chooses the population average, hence the average ToT must not be more than 99 seconds. Unfortunately, she had only tested a small minority of users and therefore cannot be certain about the true average:

```
mean(Ver20$ToT)
```

```
## [1] 106
```

Because the sample average is an uncertain, Jane is afraid, Marketing could use this as an argument to ignore the data and go with the claim. Jane sees no better way as quantifying the chance of being wrong using a statistical model, which will later become known as the *Grand Mean Model*)

```
M_1 <-
  Ver20 %>%
  stan_glm(ToT ~ 1, data = .)

P_1 <-
  posterior(M_1)
```

```
coef(P_1)
```

model	parameter	type	fixef	center	lower	upper
M_1	Intercept	fixef	Intercept	106	100	112

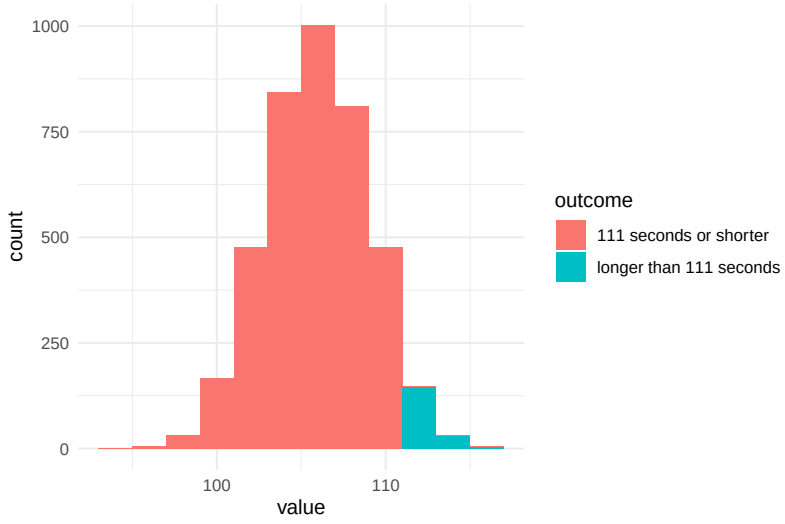
Let's see what the GMM reports about the population average and its uncertainty: The table above is called a *CLU table*, because it reports three estimates per coefficient:

- Center, which (approximately) is the most likely position of the true value
- Lower, which is the lower 95% credibility limit. There is a 2.5% chance that the true value is lower
- Upper, the 95% upper limit. The true value is larger than this with a chance of 2.5%.

This tell Jane that most likely the average time-on-task is *Intercept*. That is not very promising, and it is worse: 99 is even below the lower 95% credibility limit. So, Jane can send a strong message: The probability that this claim is justified, is smaller than 2.5%.

Luckily, Jane had the idea that the slogan could be changed to *"rent a card in 1-1-1 seconds"*. The 95% credibility limits are in her favor, since 111 is at the upper end of the credibility limit. It would be allowed to say that the probability to err is not much smaller than 2.5%. But Jane desires to make an accurate statement. But what precisely is the chance that the true population average is 111 or lower? In Bayesian analysis there is a solution to that. When estimating such a model, we get the complete distribution of certainty, called the posterior distribution. In fact, a CLU table with 95% credibility limits is just a summary on the posterior distribution. This distribution is not given as a function, but has been generated by a (finite) random walk algorith, known as Markov-Chain Monte Carlo. At every step (or most, to be precise), this algorithm jumps to another set of coordinates in parameter space and a frequency distribution arises that can be used to approximate levels of certainty. The following illustration shows the posterior frequency distribution of the coefficient, divided into the two possible outcomes.

```
P_1 %>%
  filter(parameter == "Intercept") %>%
  mutate(outcome = ifelse(value <= 111, "111 seconds or shorter", "longer than 111 seconds"
  ggplot(aes(x = value, fill = outcome)) +
  geom_histogram(binwidth = 2)
```



In the present case, we can derive the chance that the true average of the customer population is lower than the target of 111:

In a similar manner to how the graph above was produced, a precise certainty level can be estimated from the MCMC frequency distribution contained in

the posterior object. The certainty that the 111 seconds slogan holds is much better:

```
P_1 %>%
  filter(parameter == "Intercept") %>%
  summarise(certainty = mean(value <= 111))
```

certainty
0.954

```
detach(Sec99)
```

The story of Jane is about decision making under risk and under uncertainty. We have seen how easily precise statements on uncertainty can be derived from a statistical model. But regarding rational decision making, this is not an ideal story: What is missing is a systematic analysis of losses (and wins). The benefit of going with the slogan has never been quantified. How many new customers it will really attract and how much they will spend cannot really be known upfront. Let alone, predicting the chance to loose in court and what this costs are almost unintelligeable. The question must be allowed, what good is the formula for utility, when it is practically impossible to determine the losses. And if we cannot estimate utilities, what are the certainties good for?

Sometimes, one possible outcome is just so bad, that the only thing that practically matters, is to avoid it at any costs. Loosing a legal battle often falls into this category and the strategy of Marketing/Jane effectively reduced this risk: they dismissed a risky action, the 99 seconds statement, and replaced it with a slogan that they can prove is true with good certainty.

In general, we can be sure that there is at least some implicit calculation of utilities going on in the minds of Marketing. Perhaps, that is a truly intuitive process, which is felt as an emotional struggle between the fear of telling the untruth and love for the slogan. This utility analysis probably is inaccurate, but that does not mean it is completely misleading. A rough guess always beats complete ignorance, especially when you know about the attached uncertainty. Decision-makers tend to be pre-judiced, but even then probabilities can help find out to what extent this is the case: Just tell the probabilities and see who listens.

3.1.3 Comparison of designs

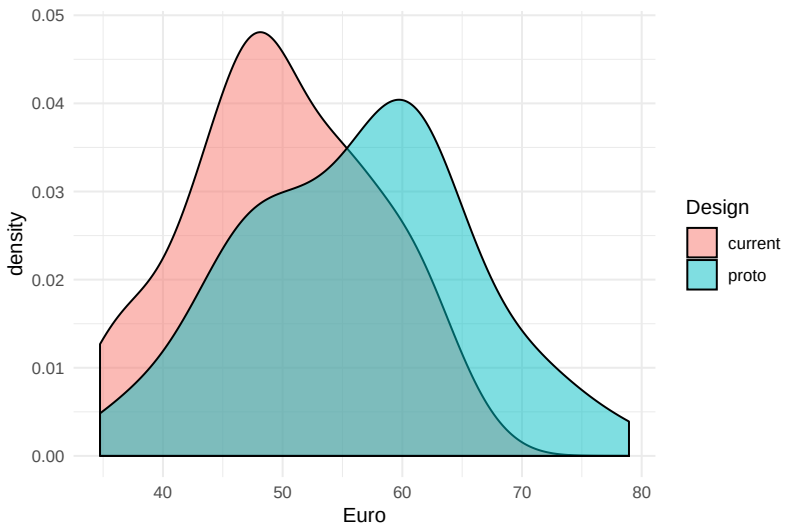
The design of systems can be conceived as a choice between design options. For example, when designing an informational website, you have the choice of making the navigation structure flat or deep. Your choice will change usability of the website, hence your customer satisfaction, rate of recurrence, revenue

etc. Much practical design research aims at making good choices and from a statistical perspective that means to compare the outcomes of two (or more) design option. A typical situation is to compare a redesign with its predecessor, which will now be illustrated by a hypothetical case:

Violet is a manager of an e-commerce website and at present, a major overhaul of the website is under debate. The management team agrees that this overhaul is about time and will most likely increase the revenue per existing customer. Still, there are considerable development costs involved and the question arises whether the update will pay itself in a reasonable time frame. To answer this question, it is not enough to know that revenues increase, but an more accurate prediction of *how much precisely* is gained. In order to return the investment, the increase must be in the ballpark of 10% increase in revenue. For this purpose, Violet carries out a user study to compare the two designs. Essentially, she observes the transactions of 50 random customers using the current system with 50 transactions with a prototype of the new design. The measured variable is the money every user spends during the visit. The research question is: *By how much do revenues increase in the prototype condition?*. The figure below shows the distribution of measured revenue in the experiment.

```
attach(Rational)
```

```
RD  %>%
  ggplot(aes(x = Euro, fill = Design)) +
  geom_density(alpha = .5)
```



There seems to be a slight benefit for the prototype condition. But, is it a 10% increase? The following calculation shows Violet that it could be the case:

```
RD %>%
  group_by(Design) %>%
  summarize(mean_revenue = mean(Euro))
```

Design	mean_revenue
current	49.9
proto	56.3

Like in the previous case, testing only a sample of 100 users out of the whole population leaves room for uncertainty. So, how certain can Violet be? A statistical model can give a more complete answer, covering the magnitude of the improvement, as well as a level of certianty. Violet estimates a model that compares the means of the two conditions 4.3.1, assuming that the randomness in follows a Gamma distribution 6.3.1.

```
library(rstanarm)
library(tidyverse)
library(bayr)

M_1 <-
  RD %>%
  stan_glm(Euro ~ Design,
           family = Gamma(link="log"),
       data = .)

P_1 <-  posterior(M_1)
```

The coefficients of the Gamma model are on a logartithmic scale, but when exponentiated, they can directly be interpreted as multiplicators 6.2.1.1. That precisely matches the reseach question, which is stated as percentage increase, rather than a difference.

```
coef(P_1, mean.func = exp)
```

parameter	fixef	center	lower	upper
Intercept	Intercept	49.91	47.04	53.00
Designproto	Designproto	1.13	1.03	1.23

```
T_fixef <- fixef(P_1, mean.func = exp)
T_fixef
```

fixef	center	lower	upper
Intercept	49.91	47.04	53.00
Designproto	1.13	1.03	1.23

The results tell Violet that, most likely, the average user spends $

type
fixef

$ Euro with the current design. The prototype seems to increase the revenue per transaction by a factor of $

type
fixef

$. That would be a sufficient increase, however this estimate comes from a small sample of users and there remains a considerable risk, that the true improvement factor is much weaker (or stronger). The above CLU table tells that with a certainty of 95% the true value lies between $

|| || || || ||

$ and $

fixef
Designproto

$. But, what precisely is the risk of the true value being lower than 1.1. This information can be extracted from the model (or the posterior distribution):

```
N_risk_of_failure <-
  P_1 %>%
  filter(parameter == "Designproto") %>%
  summarize(risk_of_failure = mean(exp(value) < 1.1))
N_risk_of_failure
```

risk_of_failure
0.288

The risk of failure is just below 30%. With this information in mind Violet now has several options:

1. deciding that 28.85% is a risk she *dares* to take and recommend going forward with the development
2. *continue testing* more users to reach a higher level of certainty
3. take into account sources of evidence from the past, or *prior knowledge*

```
detach(Rational)
```

3.1.4 Prior knowledge

It is rarely the case that we encounter a situation as a *blank slate*. Whether we are correct or not, when we look at the sky in the morning, we have some expectations on how likely there will be rain. We also take into account the season and the region and even the very planet is sometimes taken into account: the Pathfinder probe carried a bag of high-tech gadgets, including an umbrella, but this was for safe landing only, not to cover from precipation, as Mars is a dry planet.

Only in behavioral research it has become standard that every experiment had to be judged on the produced data alone. For the sake of objectivity, researchers were not allowed to take into account previous results, let alone their personal opinion. In Bayesian statistics, you have the choice. You can make use of external knowledge, but you don't have to.

Violet, the rational design researcher has been designing and testing e-commerce systems for many years and has supervised several large scale roll-outs. So the current project is not a completely new situation at all. From the top of her head, Violet produces the following table to capture her past twenty projects and the increase in revenue that had been recorded afterwards (on the whole population).

```
attach(Rational)
```

```
D_prior
```

project	revenue_increase
1	0.983
2	1.016
3	1.059
4	0.976
5	1.008
6	1.014
7	1.032
8	1.003
9	1.072
10	1.006
11	1.023
12	1.040
13	0.998
14	0.979
15	1.065
16	0.942
17	1.037
18	1.011
19	1.041
20	1.023

On this small data set, Violet estimates another grand mean model that essentially captures prior knowledge about revenue increases after redesign:

```
M_prior <-
  D_prior %>%
  stan_glm(revenue_increase ~ 1,
           family = Gamma(link = "log"),
           data = .,
           iter = 5000)

P_prior <- posterior(M_prior)
```

Note that the above model is a so called Generalized Linear Model with a Gamma shape of randomness, which will be explained more deeply in chapter 6.3.

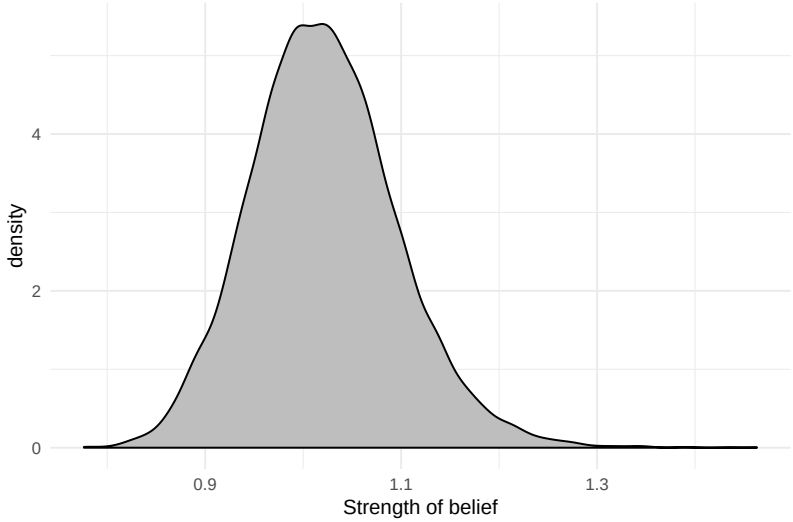
The following CLU table shows the results. The mean increase was Intercept and without any further information, this is the best guess for revenue increase in any future projects (of Violet). A statistical model that is based on such a small number of observations usually produces very uncertain estimates. In this case, we have a very low level of certainty, as the 95% credibility limits are wide. There even remains a considerable risk that a project results in a decrease of revenue, although that has never been recorded. (In Gamma models coefficients are usually multiplicative, so an Intercept < 1 is a decline).

```
coef(P_prior, mean.func = exp)
```

model	parameter type	fixef	center	lower	upper
M_prior	Intercept	fixef Intercept	1.02	0.886	1.18

Or graphically, we can depict the belief as follows:

```
P_prior %>%
  filter(parameter == "Intercept") %>%
  mutate(value = exp(value)) %>%
  ggplot(aes(x = value)) +
  geom_density(fill = "grey") +
  xlab("Strength of belief")
```



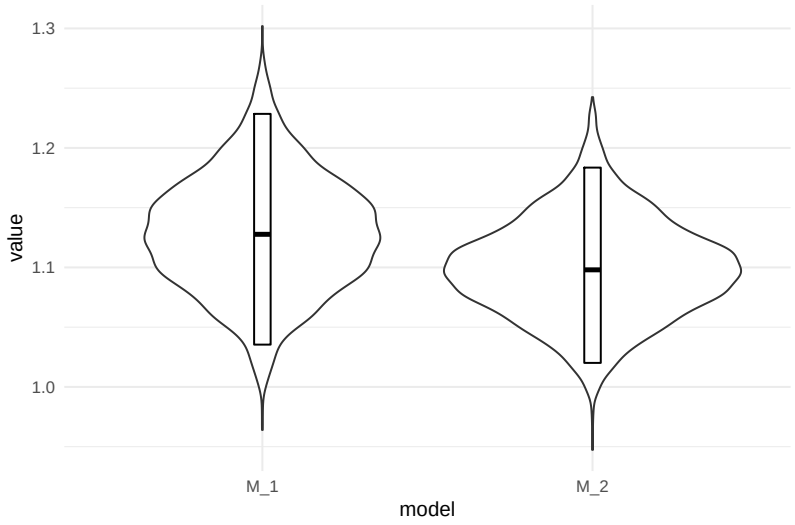
The population average (of projects) is less favorable than what Violet saw in her present experiment. If the estimated revenue on the experimental data is correct, it would be a rather extreme outcome. And that is a potential problem, because extreme outcomes are rare. Possibly, the present results are overly optimistic (which can happen by chance) and do not represent the true change revenue, i.e. on the whole population of users. In Bayesian Statistics, mixing present results with prior knowledge is a standard procedure to correct this problem. In the following step, she uses the (posterior) certainty from M_prior and employs it as prior information (by means of a Gaussian distribution). Model M_2 has the same formula as M_1 before, but combines the information of both sources, data and prior certainty.

```
T_prior <-
  P_prior %>%
  filter(parameter == "Intercept") %>%
  summarize(mean = mean(value), sd = sd(value))

M_2 <-
  stan_glm(formula = Euro ~ Design,
      prior_intercept = normal(0, 100),
      prior = normal(T_prior[[1,1]], T_prior[[1,2]]),
      family = Gamma(link = "log"),
      data = RD)

P_2 <- posterior(M_2)
```

Note that the standard deviation here is saying how the strength of belief is distributed for the average revenue. It is not the standard deviation in the a population of projects.

model	parameter	center	lower	upper
M_1	Designproto	1.13	1.03	1.23
M_2	Designproto	1.10	1.02	1.18

```
P_comb %>%
  ggplot(aes(x = model, y = value)) +
  geom_violin() +
  geom_crossbar(data = coef(P_comb),
                aes(y = center, ymin = lower, ymax = upper),
                width = .05)
```



Model

`M_2` reduces the estimated expected revenue by a small amount. But, remember that Violet has to meet the criterion of a 110% increase in revenue. In the following she extracts the risk of failure (revenue smaller than 110%) from the posterior distribution.

```
P_comb %>%
  mutate(outcome = value < 1.1) %>%
  group_by(model) %>%
  summarize(risk_to_fail = mean(outcome))
```

model	risk_to_fail
M_1	0.288
M_2	0.522

So what should Violet report to decision makers. Something like this: "The data from the study us that our chance of success is around two-thirds. *However*, in my long career I have never actually reached such an increase in revenue, so I'd rather say, chance of success is more arounf 50%.

```
detach(Rational)
```

3.2 Descriptive statistics

In empirical research we systematically gather observations. Observations of the same kind are usually subsumed as variables. A set of variables that have been gathered on the same sample are called a data set, which typically is a

table with variables in columns. In the most general meaning, *a statistic* is a single number that somehow represents relevant features of a data set, such as:

- frequency: how many measures of a certain kind can be found in the data set?
- central tendency: do measures tend to be located left (weak) or right (strong) on a scale?
- dispersion: are measures close together or widely distributed along the scale?
- association: does one variable X tend to change when another variable Y changes?

3.2.1 Frequencies

```
attach(Sec99)
```

The most basic statistics of all probably is the number of observations on a variable x, usually denoted by n_x. The number of observations is a rough indicator for the amount of data that has been gathered. In turn, more data usually results in better accuracy of statistics and higher levels of certainty can be reached.

```
length(Ver20$ToT)
```

```
## [1] 100
```

The number of observations is not as trivial as it may appear at first. In particular, it is usually not the same as the sample size, for two reasons: First, most studies employ repeated measures to some extent. You may have invited n_{Part} participants to your lab, but on each participant you have obtained several measures of the same kind. When every participant is tested on, let's say, five tasks, the number of observations is n_{Part} times five. Second, taking a valid measure can always fail for a variety of reasons, resulting in *missing values*. For example, in the 99 seconds study, it has happened, that a few participants missed to fill in their age on the intake form. The researcher is left fewer measures of age n_{age} than there were participants.

```
N <- function(x) sum(!is.na(x))
N(Ver20$age)
```

```
## [1] 100
```

Another important issue is the distribution of observations across groups. Again, the number of observations in a group is linked to the certainty we can gain on statistics of that group. Furthermore, it is sometimes important to have the distribution match the proportions in the population, as otherwise biases may occur.

```
Ver20 %>%
  group_by(Gender) %>%
  summarize(n())
```

Gender	n()
female	59
male	41

The table above shows so called absolute frequencies. When comparing frequencies by groups, it often is more appropriate to report *relative frequencies* or *proportions*:

```
n_Gender <- N(Ver20$Gender)

Ver20 %>%
  group_by(Gender) %>%
  summarize(rel_freq = n()/n_Gender)
```

Gender	rel_freq
female	0.59
male	0.41

Summarizing frequencies of metric measures, such as time-on-task (ToT) or number of errors is useful, too. However, a complication arises by the fact that continuous measures do not naturally fall into groups. Especially in duration measures no two measures are exactly the same.

```
length(unique(Ver20$Gender))
```

```
## [1] 2
```

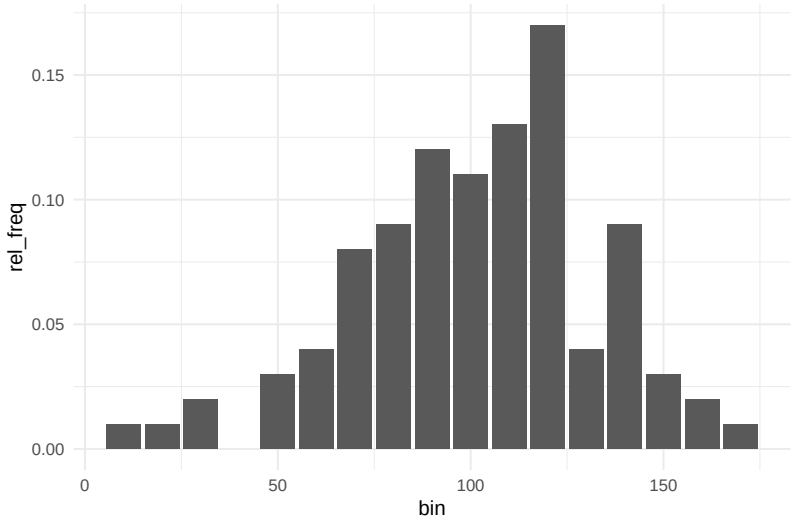
```
length(unique(Ver20$ToT))
```

```
## [1] 100
```

The answer to this problem is *binning*: the scale of measurement is divided into a number of adjacent sections, called bins, and all measures that fall into one bin are counted. For example, we could use bins of 10 seconds and assess whether the bin with values larger than 90 and smaller or equal to 100 is

representative in that it contains a large proportion of values. If we put such a binned summary of frequencies into a graph, that is called a *histogram*.

```
bin <- function(x, bin_width = 10) floor(x/bin_width) * bin_width
n_ToT <- N(Ver20$ToT)

Ver20 %>%
  mutate(bin = bin(ToT)) %>%
  group_by(bin) %>%
  summarize(rel_freq = n()/n_ToT) %>%
  ggplot(aes(x = bin, y = rel_freq)) +
  geom_col()
```




```
# Ver20 %>%
#   ggplot(aes(x = ToT)) +
#   geom_histogram(binwidth = 10)
```

Strictly spoken, grouped and binned frequencies are not one statistic, but a vector of statistics. It approximates what we will later get to know more closely as a *distribution*.

3.2.2 Central tendency

Reconsider Jane A.2.2. When asked about whether users can complete a transaction within 99, she looked at the population average of her measures. The

population average is what we call the *(arithmetic) mean*. The mean is computed by summing over all measures and divide by the number of observations. The mean is probably the most often used measure of central tendency, but two more are being used and have their own advantages: *median* and *mode*.

```
mean <- function(x) sum(x)/length(x)
mean(Ver20$ToT)
```

```
## [1] 106
```

Imagine a competitor of the car rental company goes to court to fight the 99-seconds claim. Not an expert in juridical matters, my humble opinion is that one of the first questions to be regarded probably is: what does "rent a car in 99 seconds" actually promise? One way would be the mean ("on average users can rent a car in 99 seconds"), but here are some other ways to interpret the same slogan:

"50% (or more) of users can ...". This is called the *median*. The median is computed by ordering all measures and identify the the element right in the center. If the number of observations is even, there is no one center value, and the mean of the center pair is used, instead.

```
median <- function(x){
  n <- length(x)
  center <- (n + 1)%/%2
    if (n%%2 == 1)
        sort(x, partial = center)[center]
    else mean(sort(x, partial = center + 0:1)[center + 0:1])
}

median(Ver20$ToT)
```

Actually, the median is a special case of so called *quantiles*. Generally, an quantiles are based on the order of measures and an X% quantile is that value where X% of measures are equal to or smaller. The court could decide that 50% of users is too lenient as a criterion and could demand that 75% percent of users must complete the task within 99 seconds for the slogan to be considered valid.

```
quantile(Ver20$ToT, c(.50, .75))
```

```
## 50% 75%
## 108 125
```

A common pattern to be found in distributions of measures is that a majority of observations accumulate in the center region. The point of highest density of a distribution is called the *mode*. In other words: the mode is the region (or point) that is most likely to occur. For continuous measures this once again poses the problem that every value is unique. Sophisticated procedures exist to smooth over this inconvenience, but by binning we can construct an approximation of the mode: just choose the center of the bin with highest frequency.

```
mode <- function(x, bin_width = 10) {
  bins <- bin(x, bin_width)
  bins[which.max(tabulate(match(x, bins)))] + bin_width/2
}
```

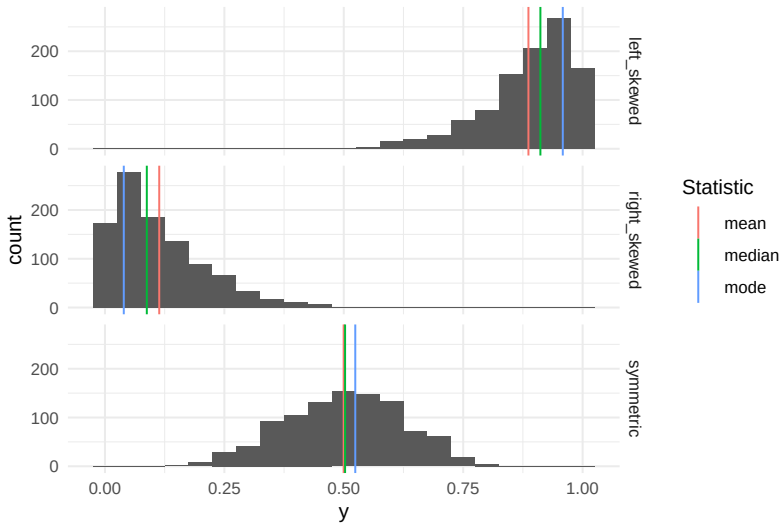
```
mode(Ver20$ToT)
```

```
## [1] 145
```

```
Ver20 %>%
  group_by() %>%
  summarize(
    mean_ToT   = mean(ToT),
    median_ToT = median(ToT),
    mode_ToT = mode(ToT))
```

mean_ToT	median_ToT	mode_ToT
106	108	145

The table above shows the three statistics for central tendency side-by-side. Mean and median are close together. This is frequently the case, but not always. When the distribution of measures is completely symmetric mean and median perfectly coincide. In section @??distributions) we will encounter distributions that are not symmetric. The more a distribution is skewed, the stronger the difference between mean and median increases.



To be more precise: for left skewed distributions the mean is strongly influenced by few, but extreme, values in the left tail of the distribution. The median only counts the number of observations to both sides and is not influenced by how extreme these values are. Therefore, it is located more to the right. The mode does not regard any values other than those in the densest region and just marks that peak. The same principles hold for right-skewed distributions.

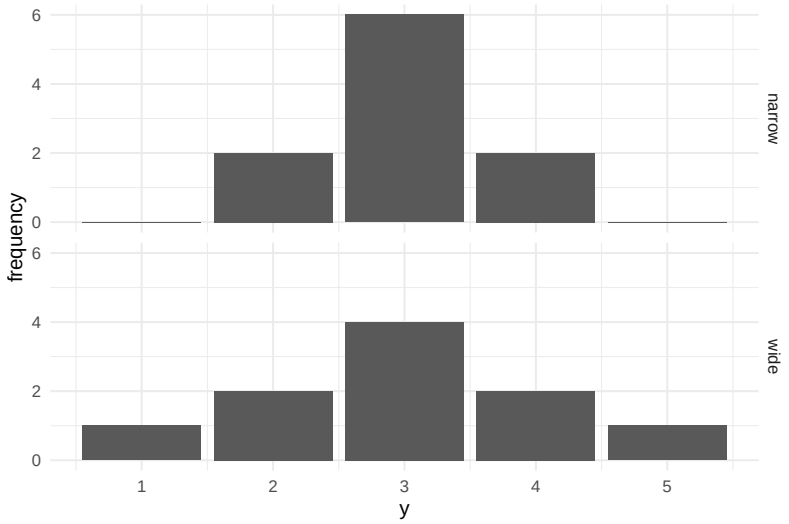
To summarize, the mean is the most frequently used measure of central tendency, one reason being that it is a so called *sufficient statistic*, meaning that it exploits the full information present in the data. The median is frequently used when extreme measures are a concern. The mode is the point in the distribution that is most typical.

3.2.3 Dispersion

In a symmetric distribution with exactly one peak, mean and mode coincide and the mean represents the most typical value. For a value being more typical does not mean it is very typical. That depends on how the measures are dispersed over the whole range. In the figure below, the center value of the narrow distribution contains 60% of all measures, as compared to 40% in the wide distribution, and is therefore more representative.

```
D_disp <-
  tribble(~y, ~narrow, ~wide,
          1, 0, 1,
          2, 2, 2,
```

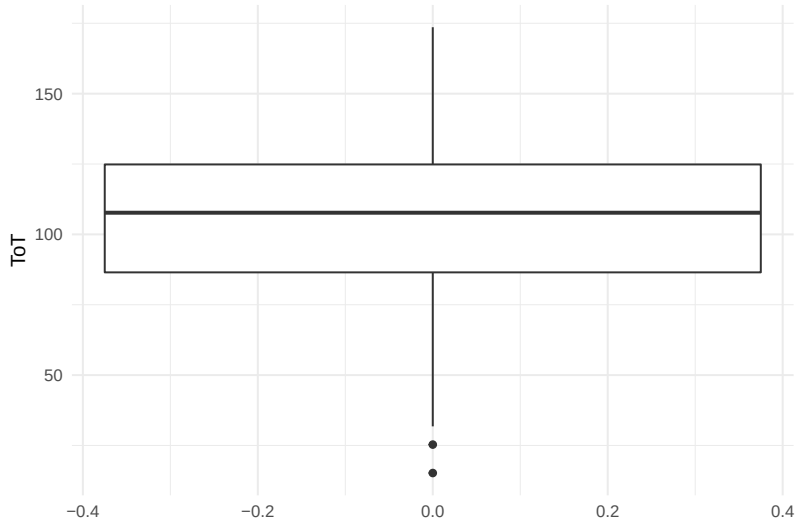
```
        3, 6, 4,
        4, 2, 2,
        5, 0, 1) %>%
  gather(Distribution, frequency, -y)


D_disp %>%
  ggplot(aes(x = y,
             y = frequency)) +
  facet_grid(Distribution ~ .) +
  geom_col()
```



A very basic way to describe dispersion of a distribution is to report the *range* between the two extreme values, *minimum* and *maximum*. These are easily computed by sorting all values and selecting the first and the last element. Coincidentally, they are also special cases of quantiles, namely the 0% and 100% quantiles.

A *boxplot* is a commonly used geometry to examine the shape of dispersion. Like histograms, boxplots use a binning mechanisms and are therefore useful for continuous measures. Whereas histograms use equidistant bins on the scale of measurement, boxplots create four bins based on 25% quantile steps. These are also called *quartiles*.

```
Ver20 %>%
  ggplot(aes(y = ToT)) +
  geom_boxplot()
```



The min/max statistics only uses just these two values and therefore does not fully represent the amount of dispersion. A statistic for dispersion that does so is the *variance*, which is the mean of squared deviations from the mean. Squaring the deviations always produces a positive value, but makes variance difficult to interpret. The *standard deviation* is the square root of variance. By reversing the square the standard deviation is on the same scale as the original measures and their mean.

```
min <- function(x) sort(x)[1]
max <- function(x) quantile(x, 1)
range <- function(x) max(x) - min(x)
var <- function(x) mean((mean(x) - x)^2)
sd  <- function(x) sqrt(var(x))

Ver20 %>%
  summarize(min(ToT),
            max(ToT),
            range(ToT),
            var(ToT),
            sd(ToT))
```

min(ToT)	max(ToT)	range(ToT)	var(ToT)	sd(ToT)
15.2	174	158	966	31.1

```
detach(Sec99)
```

3.2.4 Associations

- Are elderly users slower at navigating websites?
- How does reading speed depend on font size?
- Is the result of an intelligence test independent from gender?

In the previous section we have seen how all individual variables can be described by location and dispersion. A majority of research deals with associations between measures and the present section introduces some statistics to describe them. Variables represent properties of the objects of research and fall into two categories: *Metric variables* represent a measured property, such as speed, height, money or perceived satisfaction. *Categorical variables* put observations (or objects of research) into non-overlapping groups, such as experimental conditions, persons who can program or cannot, type of education etc. Consequently, associations between any two variables fall into precisely one of three cases, as shown in the table. In the following I will explain these types of associations.

	categorical	metric
categorical	frequency cross tables	differences in mean
-	-	-
metric	classification	covariance, correlation
-	-	-

3.2.4.1 Categorical associations

Categorical variables group observations, and when they are *both categorical*, the result is just another categorical case and the only way to compare them is by size, which is done by *frequency tables*. To illustrate the categorical-categorical case, consider a study to assess the safety of two syringe infusion pump designs, called Legacy and Novel. All participants of the study are asked to perform a typical sequence of operation on both devices (categorical variable Design) and it is recorded whether the sequence was completed correctly or not (categorical variable Correctness).

```
attach(IPump)

D_agg %>%
  filter(Session == 3) %>%
  group_by(Design, completion) %>%
  summarize(frequency = n()) %>%
  ungroup() %>%
  spread(completion, frequency)
```

Design	FALSE	TRUE
Legacy	21	4
Novel	22	3

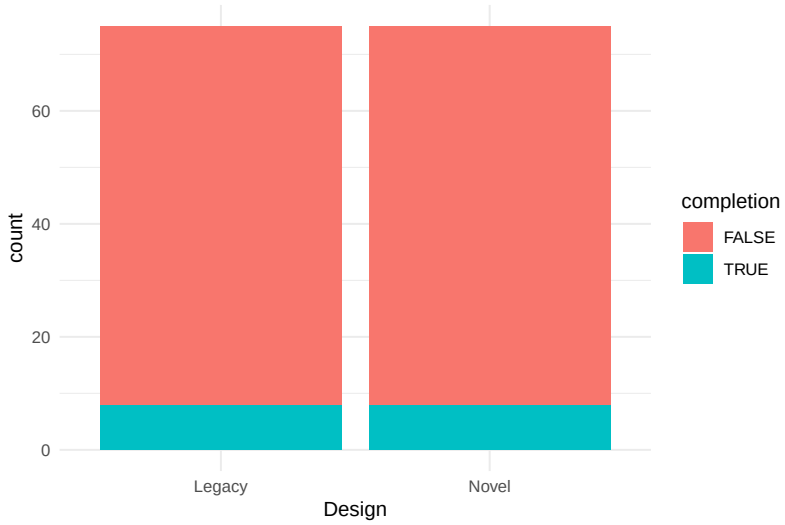
Besides the troubling result that incorrect completion is the rule, not the exception, there is almost no difference between the two designs. Note that in this study, both professional groups were even in number. If that is not the case, absolute frequencies are difficult to compare and we better report *relative frequencies*. Note how every row sums up to 1.

```
D_agg %>%
  filter(Session == 3) %>%
  group_by(Design, completion) %>%
  summarize(frequency = n()) %>%
  group_by(Design) %>%
  mutate(frequency = frequency/sum(frequency)) %>%
  ungroup() %>%
  spread(completion, frequency)
```

Design	FALSE	TRUE
Legacy	0.84	0.16
Novel	0.88	0.12

The absolute or relative frequencies can be shown in a stacked *bar plot*:

```
D_agg %>%
  ggplot(aes(x = Design, fill = completion)) +
  geom_bar()
```



3.2.4.2 Categorical-metric associations

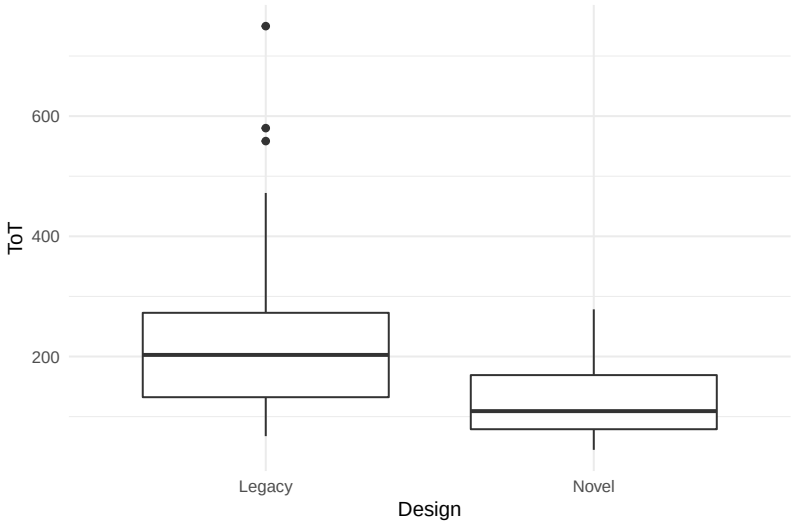
Associations between *categorical and metric* variables are reported by *grouped location statistics*. In the case of the two infusion pump designs, the time spent to complete the sequence is compared by the following table. And as you can see, adding a comparison of variance (or any other statistic) is not a hassle.

```
D_agg %>%
  filter(Session == 3) %>%
  group_by(Design) %>%
  summarize(mean_ToT = mean(ToT),
            sd_ToT = sd(ToT))
```

Design	mean_ToT	sd_ToT
Legacy	151.0	62.2
Novel	87.7	33.8

For the illustration of categorical-metric associations case, *boxplots* haven proven useful. Boxplots show differences in central tendency (median) and dispersion (other quartiles) simultaneously. Here we can observe that Novel design produces shorter ToT and seems to have less dispersion.

```
D_agg %>%
  ggplot(aes(x = Design, y = ToT)) +
 geom_boxplot()
```



3.2.4.3 Covariance and correlation

For associations between *both metric* variables, *covariance* and *correlations* are commonly employed statistics. As correlations derive from covariances, I describe them first:

1. A covariance is a real number that is *zero* when there really is *no association* between two variables.
2. When two variables move into the *same direction*, covariance gets the *positive*.
3. When they move in *opposite directions*, covariance is *negative*

For an illustration, consider the following hypothetical example of study on the relationship between mental ability test scores and performance in real tasks. The real task could be a buzz wire task or any other complex motor task that has a strong spatial component. As predictors for ToT on the buzz-wire task, mental rotation speed was taken. It was even taken twice (MRS_1, MRS_2), because the researchers are also interested in how stable over time the scores are. The second test is for measuring visual-spatial working memory, using the Corsi task.

```
D_psychomet %>% sample_n(5)
```

MRS_1	MRS_2	ToT	Corsi
2.02	2.04	188	6.59
1.78	1.91	215	6.74
1.68	1.83	166	6.57
2.35	2.29	201	7.56
1.86	1.81	230	5.52

The following function computes the covariance of two variables. The covariance between the two MRS scores is positive, indicating that they move into the same direction.

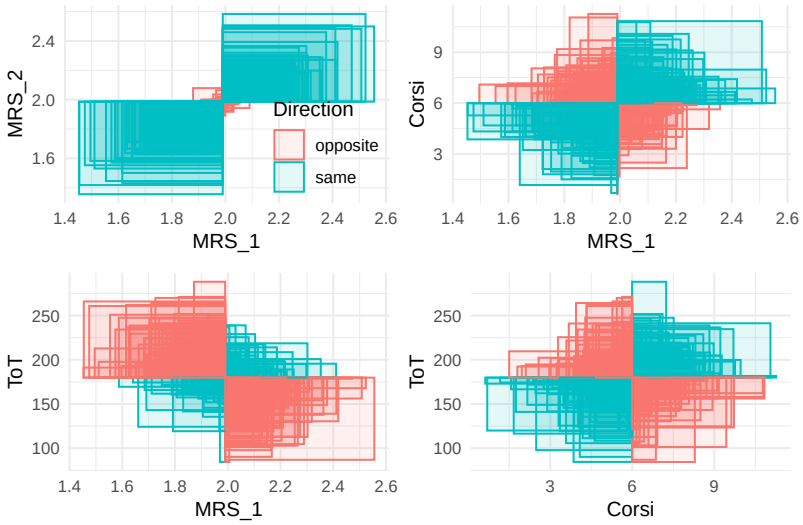
```
cov <- function(x, y)
  mean((x - mean(x)) * (y - mean(y)))

cov(D_psychomet$MRS_1, D_psychomet$MRS_2)
```

```
## [1] 0.0407
```

The problem with covariance is that it usually is hard to interpret. This is why later, we will transformk covariances into correlations, but for understanding the stapes to go there, we have to understand the link between variance and covariance. The formula for covariance is (with $E(X)$ the mean of X):

$$\text{cov}_{XY} = \frac{1}{n}\sum_{i=1}^{n}(x_i - E(X))(y_i - E(Y))$$

Covariance essentially arises by the multiplication of deviations from the mean, $(x_i - E(X))(y_i - E(Y)$. When for one observation both factors go into the same direction, be it positive or negative, this term gets positive. If that happens a lot, the whole sum gets largely positive. When the deviations systematically move in opposite direction, such that one factor is always positive and the other negative, we get a large negative cvariance. When the picture is mixed, i.e. no clear tendenvcy, covariance will stay close to zero. The following illustration uses a geometric interpretation of the multiplication as the area of rectangles. Rectangles with equal directions (blue) are in the upper-right and lower-left quadrant. They overwhelm the opposite direction rectangles (red), which speaks for a strong positive association. The associations between MRT_1 and Corsi, as well as between Corsi and ToT seem to have a slight overhead in same direction, so the covariance is positive, but less strong. A clear negative association exists between MRS_1 and Corsi. It seems these two tests have some common ground.



If we now look at the defintion of variance, it is apparent that variance is just covariance of a variable with itself (, because $(x_i - E(X))^2 = (x_i - E(X))(x_i - E(X))$:

$$\text{var}_X = \frac{1}{n}\sum_{i=1}^{n}(x_i - E(X))^2)$$

That gives rise to a compact form to show all covariances and variances between a bunch of variables at once. The following table is a *variance-covariance matrix*. It shows the variance of every variable in the diagonale and the mutual covariances in the off-diagonal cells.

```
D_psychomet %>% cov()
```

	MRS_1	MRS_2	ToT	Corsi
MRS_1	0.043	0.041	-4.25	0.106
MRS_2	0.041	0.043	-4.13	0.104
ToT	-4.246	-4.127	1515.49	7.615
Corsi	0.106	0.104	7.62	4.185

As intuitive the idea of covariance is, as unintelligible is the statistic itself for reporting results. Th problem is that covariance is not a pure measure of association, but is contaminated by the dispersion of X and Y. For that reason, two covariances can only be compared if the variables have the same variance. The *Pearson correlation coefficient* r solves the problem by rescaling covariances by the product of the two standard deviations:

$$r_X Y = \frac{\text{cov}_{XY}}{\text{sd}_X \text{sd}_Y}$$

```
cor <- function(x, y)
  cov(x, y)/(sd(x, na.rm = T) * sd(y, na.rm = T))

cor(D_psychomet$MRS_1, D_psychomet$MRS_2)
```

```
## [1] 0.952
```

Due to the standardization of dispersion, Pearson correlation r can be interpreted as strength of association independent of scale of measurement. More precisely, r will always be in the interval $[-1, 1]$. That makes it the perfect choice when associations are being compared to each other or to an external standard. In the field of psychometrics, correlations are ubiquitously employed to represent *reliability* and *validity* of psychological tests. *Test-retest stability* is one form to measure reliability and it is just the correlation of the same test taken on different days. For example, we could ask whether mental rotation speed as measured by the mental rotation task (MRT) is stable over time, such that we can use it for long-term predictions, such as how likely someone will become a good surgeon. Validity of a test means that it represents what it was intended for, and that requires an external criterion that is known to be valid. For example, we could ask how well the ability of a person to become a minimally invasive surgeon depends on spatial cognitive abilities, like mental rotation speed. Validity could be assessed by taking performance scores from exercises in a surgery simulator and do the correlation with mental rotation speed. A correlation of $r = .5$ would indicate that mental rotation speed as measured by the task has rather limited validity. Another form is called *discriminant validity* and is about how specific a measure is. Imagine another test as part of the surgery assessment suite. This test aims to measure another aspect of spatial cognition, namely the capacity of the visual-spatial working memory (e.g., the Corsi block tapping task). If both tests are as specific as they claim to be, we would expect a particularly low correlation.

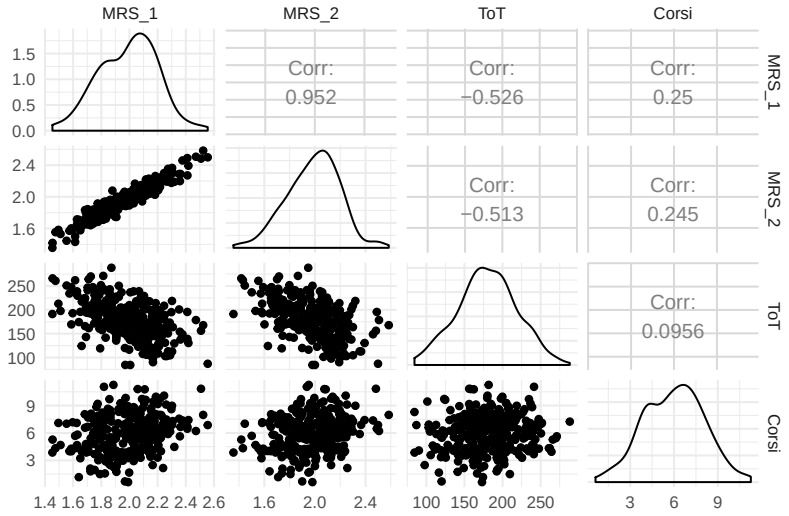
And similar to covariances, correlations between a set of variables can be put into a correlation table. This time, the diagonal is the correlation of a variable with itself, which is perfect correlation and therefore equals 1.

```
D_psychomet %>% cor()
```

	MRS_1	MRS_2	ToT	Corsi
MRS_1	1.000	0.952	-0.526	0.250
MRS_2	0.952	1.000	-0.513	0.245
ToT	-0.526	-0.513	1.000	0.096
Corsi	0.250	0.245	0.096	1.000

Another way to illustrate a bunch of correlations is produced by the following command, combining scatterplots, density plots and correlation coefficients

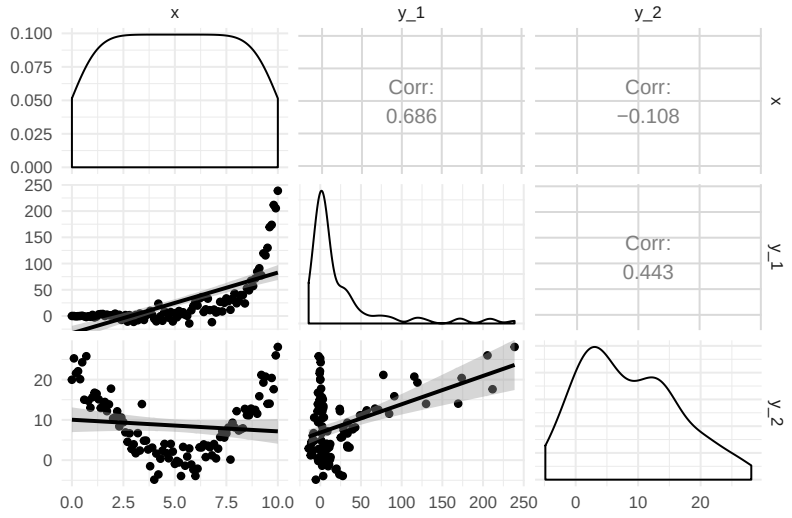
```
D_psychomet %>%
  GGally::ggpairs()
```



Correlations give psychometricians a comparable standard for the quality of measures, irrespectively on what scale they are. In exploratory analysis, one often seeks to get a broad overview of how a bunch of variables is associated. Creating a correlation table of all variables is no hassle and allows to get a broad picture of the situation. Correlations are ubiquitous in data analysis, but have limitations: First, a correlation only uncovers linear trends, whereas the association between two variables can take any conceivable form. The validity of correlations depends on how salient the feature of linear trend is. In the example below, Y_1 reveals a strong parabolic form, which results in zero correlation. The curvature of an exponentially rising function is only captured insufficiently. For that reason, I recommend that correlations are always cross-checked by a scatterplot.

Another situation where covariances and correlations fail is when there simply is no variance. It is almost trivial, but for observing how a variable Y changes when X moves is that both variables *vary*, there is no co-variance without variance.

```
tibble(x =  (0:100)/10,
           y_1 = rnorm(101, exp(x)/100, x * 2),
           y_2 = rnorm(101, (x - 5)^2, 3)) %>%
  ggpairs(lower=list(continuous="smooth"))
```



As we have seen, for every combination of two categorical and metric variables, we can produce summary statistics for the association, as well as graphs. The second part of this book really is about statistical models that extend association statistics in two ways: first, we will see that linear models 4.4 can capture the associations between more than just two variables. Second, by Bayesian estimation, we can derive statements of uncertainty.

While it could be tempting to primarily use summary statistics and rather omit statistical graphs, the last example makes clear that some statistics like correlation are making assumptions on the shape the association. The different graphs we have seen are much less presupposing and can therefore be used to check the assumptions of statistics and models.

3.3 Bayesian probability theory

Mathematics is emptiness. In its purest form, it does not require or have any link to the real world. That makes it so difficult to comprehend. Sometimes a mathematical theory describes real world phenomena, but we have no intuition about it. A classic example is Einstein's General Relativity Theory, which assumes a curved space, rather than the straight space our senses are tuned to. Our minds are Newtonian and the closest to intuitive understanding we can get is the imagination of the universe as a four-dimensional mollusk, thanks to Einstein.

Math can also be easy, even trivial, if mathematical expressions directly translate into ideas and sensations as we have them. I recall how my primary school

teacher introduced the sum of two numbers as removing elements from one stack and place it on second (with an obvious stop rule). Later, as a student, I was taught how the sum of two numbers is defined within the Peano axiomatic theory of Natural Numbers. As it turned out, I knew this already, because they just formalized the procedure I was taught as a kid. The formal proof for $1 + 1 = 2$ is using just the same elements as me shifting blocks between towers.

In this section I will introduce *probability theory*, which is largely based on another mathematical theory that most people find intuitive, *set theory*. The formal theory of probability, the Kolmogorov axioms may be somewhat disappointing from an ontological perspective, as it just defines rules for when a set of numbers can be regarded probabilities. But calculating actual probabilities is rather easy and a few R commands will suffice to start playing with set theory and probability. The most tangible interpretation of probabilities is that the probability of an event to happen, say getting a Six when rolling a dice, coincides with the relative frequency of Six in a (very long) sequence of throws. This is called the *frequentist interpretation* of probability and this is how probability will be introduced in the following. While thinking in terms of relative frequency in long running sequences is rather intuitive, it has limitations. Not all events we want to assign a probability can readily be imagined as a long running sequence, for example:

- the probability that your house burns down (you only have this one)
- the probability that a space ship will safely reach Mars (there's only this one attempt)
- the probability that a theory is more true than another (there's only this pair)

The *Bayesian interpretation* of probability is essentially the same as the frequentist, but is more relaxed as it does *not require that all* probabilities are measured through relative frequencies in long running sequences. Bayesian thinking includes the idea that probabilities can also be a *degree of belief*, which can, but doesn't have to be grounded in long-running series. In the following I will present in broad strokes how the theory of probability emerges from set theory and can be set into motion by computing relative frequencies of sets and subsets. Then I will introduce the *likelihood*, which is a concept equally used in classic and Bayesian statistics. After clarifying the differences between frequentist and Bayesian ideas of measuring probability, *Bayes theorem* is introduced as the formal underpinning of all Bayesian statistics. We will see how the likelihood and idea of *certainty as probability* combines to a scientific framework that emphasizes the incremental updating of our knowledge about the world we measure.

3.3.1 Some set theory

The mathematical concept of probability can most intuitively be approached by thinking in terms of relative frequency in long-running sequences. Actually, it is not even required to think of a sequence (where events have an order). It suffices to assume a set of events that emerge from one experiment.

A mathematical *set* is a collection of elements taken from a domain (or universe, more dramatically). These can either be defined by stating all the elements, like $S = \{\text{red}, \text{yellow}, \text{green}, \text{off}\}$ or by a characterizing statement, like:

$S :=$ possible states of a Dutch traffic light

The elements should be clearly identified, but need not have a particular order. (If they do, this is called an *ordered set*, the set of natural numbers is an example). Sets can have all possible sizes, which is called the *cardinality* of a set:

- finite (and countable) like the states of a traffic light
- empty like "all opponents who can defeat Chuck Norris", {} or $\oslash$
- infinite, but countable, like the natural numbers N
- infinite, uncountable, like the real numbers R

You may wonder now, whether you would ever need such a strange concept as uncountable infinite sets in your down-to-earth design research. Well, the set of primary interest in every design study is the possible outcomes. Sometimes, these are finite, like $\{\text{success}, \text{failure}\}$, but when you measure durations or distances, you enter the realm of real numbers. We will set this issue aside for the moment and return to it later in the context of continuous distributions of randomness.

In order to introduce the mathematical concept of probability, we first have to understand some basic operations on sets. For an illustration, imagine a validation study for a medical infusion pump, where participants were given a task and the outcome was classified by the following three criteria:

- was the task goal achieved successfully?
- was the task completed timely (e.g., one minute or below)?
- were there any operation errors along the way with potentially harmful consequences?

Note how the data table makes use of logical values to assign each observation a membership (or not) to each of the three sets. We can use the filter command to create all kinds of subsets and actually that would carry us pretty far into set theory. In the following I will introduce set theory thze Programming way,

but use the package Sets, as it most closely resembles the mathematical formalism, it replaces. We begin with loading the package, which unfortunately uses the %>% operator for its own purpose.

```
library(sets)
## masks %>% operator. don't use in conjunction with dplyr
```

```
All      <- as.set(D_sets$Obs)
Success  <- as.set(filter(D_sets, Success)$Obs)
Harmful  <- as.set(filter(D_sets, Harmful)$Obs)
Timely   <- as.set(filter(D_sets, Timely)$Obs)
```

Once there is more than one set in the game, set operators can be used to create all kinds of new sets. We begin with the *set difference*, which removes elements of one set from another (if they exist), for example the set of all successful tasks that were not completed in time. Note how the Sets package uses the minus operator to remove elements of one set (Timely) from another (Success).

```
Success - Timely
```

```
## {24L, 25L, 26L, 27L, 28L, 29L}
```

Using the set difference, we can produce *complementary set* which include all elements that are not included in a set.

```
Failure  <- All - Success
Harmless <- All - Harmful
Delayed  <- All - Timely
```

In probability theory this corresponds with the probability of an event (Success) and its *counter-event* (Failure). A set and its complementary set taken together produce the *universal set*, which in probability theory is the *sure event* with a probability of One. To show that we can use *set union*, which collects the elements of two separate sets into one new set, for example reuniting a set with its complementary,

```
Success | Failure == All
```

```
## {7L, 8L, 9L, 10L, 11L, 12L, 13L, 14L, 15L, 16L, 17L, 18L, 24L, 25L,
##  26L, 27L, 28L, 29L, FALSE}
```

or creating the set of all observations that were failure or delayed (or both):

```
Failure | Delayed
```

```
## {1L, 2L, 3L, 4L, 5L, 6L, 19L, 20L, 21L, 22L, 23L, 24L, 25L, 26L, 27L,
##  28L, 29L, 30L}
```

Another commonly used set operator is the *intersect*, which produces a set that contains only those elements present in both original sets, like the set of timely and successful task completions.

```
Success & Timely
```

```
## {7L, 8L, 9L, 10L, 11L, 12L, 13L, 14L, 15L, 16L, 17L, 18L}
```

Turns out all successful observations are also harmless. But not all harmless observations were successful. In set theory Success is therefore a *subset* of Harmless. The subset operator differs from those discussed so far, in that it does not produce a new set, but a truth value (also called logical or Boolean). Per definition, two *equal* sets are also subsets of each other. The `<` operator is more strict and it means a *proper subsets*, where being a subset has just one direction.

```
# subset
Success <= Harmless
```

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

```
# proper subset
Success < Harmless
```

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

```
# set equality
Success == Harmless
```

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

```
Success == (All - Failure)
```

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

The example above demonstrates the, figuratively, smallest concept of set theory. The *empty set* has the special property of being a subset of all other set:

```
set() < Success
```

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

The empty set is important for the intersect operator to work properly. It may happen that two sets do not share any elements at all. It would be problematic, if the intersect operator only worked if common elements truly existed. In such a case, the intersection of two sets is the empty set. Sets that have an empty intersection are called *disjunct sets* (with complementary sets as a special case). The package Sets, which defines all operators on sets so far is lacking a dedicated function for disjunctness, but this is easily defined using the intersect function:

```
is_disjunct <- function(x, y) set_is_empty(x & y)
is_disjunct(Success, Harmful)
```

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

So far, we have only seen sets of atomic elements, where all elements are atomic, i.e. they are not sets themselves.

With a little more abstraction, we can also conceive a set that has other sets as its elements. The set of sets that are defined by the three performance criteria and their complementary sets is an obvious example:

```
set(Success, Failure, Harmful, Harmless, Timely, Delayed)
```

```
## {<<set(9)>>, <<set(10)>>, <<set(12)>>, <<set(18)>>, <<set(20)>>,
##  <<set(21)>>}
```

For the formal introduction of probability, we need two concepts related to sets of sets: First, a *partition of a set* is a set of non-empty subsets such that every element is assigned to exactly one subset. The subsets of successes and its complementary set, all failures, is such a partition. Second, the *power set* is the set of all possible subsets in a set. Even with a rather small set of 20 elements, this is getting incredibly large, so let's see it on a smaller example:

```
S <- set(1, 2, 3)
P <- set_power(S)
P
```

```
## {{}, {1}, {2}, {3}, {1, 2}, {1, 3}, {2, 3}, {1, 2, 3}}
```

The power set is tantamount for the definition of probability that follows, because it has two properties: first, for every subset of `S` it also contains the complementary set. That is called *closed under complementarity*. Second, for every pair of subsets of `S`, `P` it also contains the union, it is *closed under union*. In the same way, power sets are also *closed under intersection*. Generally, all sets of subsets that fulfill these three requirements are called Σ *algebras*. The mathematical theory of Σ algebras is central for the mathematical definition of all measures.

Without going into to much depth on measurement theory, a measure is a mapping from the domain of empirical observations to the domain of numbers, such that certain operations in the domain of measurement work conistently with numerical operations. One example is the following: if you have to towers of blocks, L and R, next to each other and you look at them from one side, then the following rule applies for translating between the world of sensations and the world of sets:

`L L R <-- Observer L R L R` If you can see the top of tower L, when looking from the right side, then tower L is larger than tower R is build with *more* blocks than tower R. Probabilities are measures and in the next section we will see how numerical operations on probabilities relate to set operations in a Σ algebra. We will also see that relative frequencies are measures of probability.

3.3.2 Probability

In the following I will outline the formal theory of probability and use the same fictional validation study to illustrate the relevant concepts. introduced in the previous section. Performance of participants was classified by the three two-level criteria, success, harm and timeliness. Every recorded outcome therefore falls into one of eight possible sets and a purposeful way to summarize the results of the study would be relative frequencies (π, `pi`):

```
N_sets <- nrow(D_sets)

D_freq <-
  D_sets %>%
  group_by(Success, Harmful, Timely) %>%
  summarize(n = n()) %>%
  ungroup() %>%
  complete(Success, Harmful, Timely, fill = list(n = 0)) %>% # adds empty events
  mutate(pi = n/sum(n))

D_freq
```

Success	Harmful	Timely	n	pi
FALSE	FALSE	FALSE	1	0.033
FALSE	FALSE	TRUE	2	0.067
FALSE	TRUE	FALSE	3	0.100
FALSE	TRUE	TRUE	6	0.200
TRUE	FALSE	FALSE	6	0.200
TRUE	FALSE	TRUE	12	0.400
TRUE	TRUE	FALSE	0	0.000
TRUE	TRUE	TRUE	0	0.000

Let's examine on an abstract level, what has happened here:

1. The set of events has been partitioned into eight non-overlapping categories made by three-way intersections. The first row, for example, is the intersect of the three sets Failure, Harmless and Delayed (see previous section).
2. All subsets got a real number assigned, by the operation of relative frequencies which produces numbers between (and including) Zero and One.
3. A hidden property is that, if we unite all sets, we get the universal set and, not coincidentally, if we sum over the frequencies the result is One:

```
sum(D_freq$pi)
```

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

Back to formal: The mathematical theory of probability departs from a set of *outcomes* Ω and a Σ algebra F defined on Ω. An element E of F therefore is a set of outcomes, which is called an *event*.

The eight threeway interaction sets above are a partition of Ω, but not a Σ-algebra. As disjunct sets they are closed under intersection for trivial reasons, but they are not closed under union. For that we had to add a lot of possible outcomes, all counter-sets to start with. The point is that we can construct all these subsets using filter commands and produce relative frequencies, like above.

Probability as an axiomatic theory is defined by the three *Kolmogorov axioms*:

The *first Kolmogorov axiom* states that a probability is a non-negative real number assigned to every event. The computation of relative frequencies satisfies this condition hands down.

The first axiom defines a lower border of Zero for a probability measure, the *second Kolmogorov axiom* is taking care of an upper limit of One. This happens indirectly by stating that the set of all observations Ω (which is an element of F) is assigned a probability of One. In the table of relative frequencies that is not yet covered, but we can easily do so:

```
D_sets %>%
  # no group_by
  summarize(pi = n()/N_sets) %>%
  c()
```

```
## $pi
## [1] 1
```

So far, the theory only cared for assigning numbers to events (subsets), but provides no means to operate on probabilities. The *third Kolmogorov axiom* establishes a relation between the union operator on sets and the sum operator on probabilities by stating that the probability of a union of disjunct events is the sum of the individual probabilities. We can approve this to be true for the relative frequencies. For example, the question could be: Is the set of all successful observations the union of successful timely observations. Indeed, the relative frequency of all successful events is the sum of the two and satisfies the third axiom:

```
D_sets %>%
  group_by(Success) %>%
  summarize(n = n()) %>%
  mutate(pi = n/sum(n))
```

Success	n	pi
FALSE	12	0.4
TRUE	18	0.6

The Kolmogorov axioms establish a probability measure and lets us do calculations on disjunct subsets. That would be a meager toolbox to do calculations with probabilities. What about all the other set operators and their possible counterparts in the realm of numbers? It is one of greatest wonders of the human mind that the rich field of reasoning about probabilities spawns from just these three axioms and a few set theoretic underpinnings. To just give one very simple example, we can derive that the probability of the complement of a set A is $P(\Omega/A) = 1 - P(A)$:

1. From set theory follows that a set A and its complement Ω/A are disjunct, hence axiom 3 is applicable: $P(A \cup \Omega/A) = P(A) + P(\Omega/A)$
2. From set theory follows that a set A and its complement Ω/A form a partition on Ω. Using axiom 2, we can infer:

$$\begin{aligned} A \cup \Omega/A &= \Omega \\ \Rightarrow \quad P(A) + P(\Omega/A) = P(\Omega) &= 1 \\ \Rightarrow \quad P(\Omega/A) &= 1 - P(A) \end{aligned}$$

The third axiom tells us how to deal with probabilities, when events are disjunct. As we have seen, it applies for defining more general events. How about the opposite direction, calculating probabilities of more special events? In our example, two rather general events are Success and Timely, whereas the intersection event Success *and* Timely is more special. The probability of two events occurring together is called *joint probability* $P(\text{Timely} \cap \text{Success})$. The four joint probabilities on the two sets and their complements are shown in the following table.

```
D_sets %>%
  group_by(Success, Timely) %>%
  summarize(pi = n()/N_sets) %>%
  ungroup()
```

Success	Timely	pi
FALSE	FALSE	0.133
FALSE	TRUE	0.267
TRUE	FALSE	0.200
TRUE	TRUE	0.400

As joint probability asks for simultaneous occurrence it treats both involved sets symmetrically: $P(\text{Timely} \cap \text{Success}) = P(\text{Successes} \cap \text{Timely})$. What if you are given one piece of information first, such as "this was a successful outcome" and you have to guess the other "Was it harmful?". That is called *conditional probability* and in this case, it is Zero. But what is the conditional probability that, when you know an event was a failure, it really caused harm?

$P(\text{Harmful}|\text{Success})$

```
D_sets %>%
  filter(!Success) %>%
  group_by(Harmful) %>%
  summarize(n = n()) %>%
  mutate(pi = n/sum(n)) %>%
  ungroup()
```

Harmful	n	pi
FALSE	3	0.25
TRUE	9	0.75

In the manner of a speed-accuracy trade-off, there could be a relationship between Timely and Harm. Participants who rush through the task are likely

to make more harmful errors. We would then expect a different distribution of probability of harm by whether or not task completion was timely.

```
D_sets %>%
  group_by(Timely, Harmful) %>%
  summarize(n = n()) %>%
  mutate(pi = n/sum(n)) %>%
  ungroup()
```

Timely	Harmful	n	pi
FALSE	FALSE	7	0.7
FALSE	TRUE	3	0.3
TRUE	FALSE	14	0.7
TRUE	TRUE	6	0.3

See how conditional probabilities sum up to one *within* their condition. In this case, the conditional probabilities for harm are the same for successes and failures. As a consequence, it is also the same as the overall probability, hence:

$$P(\text{Harm}|\text{Timely}) = P(\text{No harm}|\text{Timely}) = P(\text{Timely})$$

This situation is called *independence of events* and it means that knowing about one variable does not help in guessing the other. In Statistics, *conditional probability* and *independence of events* are tightly linked to *likelihoods* and *Bayes theorem* 3.3.5.

3.3.3 Likelihood

In the previous section we have seen how to create probabilities from relative frequencies in data how to do basic calculations with those probabilities. Using these concept, we will see in teh following how data can be used for inference. Inference means to draw conclusions about theoretical models. In the following this will be illustrated at the example of rolling dices, where the default theory is that the dice is fair, hence the probability of Six is assumed to be `\pi_{Six}`. For the matter here, we take the theory of the dice being fair into the equantion, as a condittional probability, in words: "Conditional on this dice being fair, the chance of rolling Six is 1/6".

$$P(y = \text{Six}|\pi_{\text{Six}} = 1/6)$$

When such a standard standard dice is rolled twice, how likely is an outcome of two times Six? We use probability theory: It is one dice but the results of

the rolls are otherwise independent; for example, the probability to roll a Six does not wear off over time or anything like that.

Because of independence, the joint probability of rolling two times Six is just the product:

$$\begin{aligned} & P(y_1 = \text{Six and } y_2 = \text{Six} | \pi = 1/6) \\ =& P(y_1 = \text{Six} | \pi = 1/6) \times P(y_2 = \text{Six} | \pi = 1/6) \\ =& 1/36 \end{aligned}$$

The joint probability of all data points is called the *Likelihood* and generalizes to situations where the probabilities of events are not the same, for example: What is the probability of the one dice being a Four, the second a five and the third a Three or a Six?

$$P(y_1 = \text{Four and } y_2 = \text{Five and } y_3 = \text{Three or Six}) = \frac{1}{6} \times \frac{1}{6} \times \frac{1}{3} = \frac{1}{108}$$

Notice how the likelihood gets smaller in the second example. In fact, likelihoods are products of numbers between zero and one and therefore become smaller with every observation that is added. In most empirical studies, the number of observations is much larger than two or three and the likelihood becomes inexpressibly small. Consider the following results from 16 rolls.

```
set.seed(42)
Events <- c("One", "Two", "Three", "Four", "Five", "Six")
Result <- sample(Events, 16, replace = T)
pi = 1/6
Likelihood <- pi^length(Result)
```

The likelihood of this result, given that it is a fair dice, is $\frac{1}{6}^{16} = 3.545 \times 10^{-13}$. Therefore, one usually reports the *logarithm of the likelihood (log-likelihood)*. This results in "reasonable" negative numbers. Why negative? Because all Likelihoods are fractions of One (the identity element of multiplication), which results in a negative logarithm.

```
(logLik <- log(Likelihood))
```

```
## [1] -28.7
```

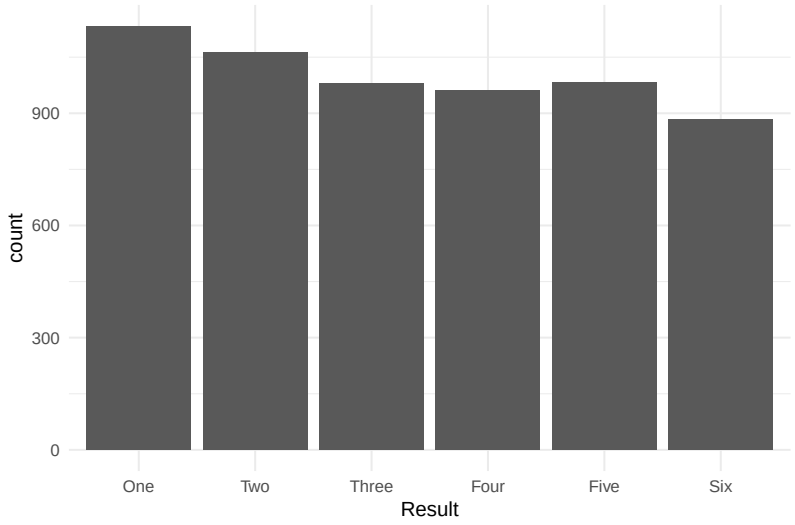
The dice rolling example above has a twist. assumed that we may enter $\pi = 1/6$, because we *believe* that it is a fair dice, without further notice. In other words, we needed no data, because of overwhelming prior knowledge

(or theory, if you will). And now we will come to see, why I took the effort to write the probabilities above as conditional probabilities. A likelihood is the probability of the data, given a parameter value. The basic idea of likelihoods is to consider data constant, and vary the parameter. In such a way, we can see how the likelihood changes when we assume different values for π_{Six}.

Imagine we have been called in to uncover fraud with biased dices in a casino. There is suspicion, that the chance of rolling a Six is lower than 1/6. So, what is the most likely chance of rolling a Six? In the following simulation, 6000 rolls have been recorded:

```
n_Rolls <- 6000

Biased_dice <-
  tibble(Side = as_factor(Events),
         pi = c(1/6 + .02, rep(1/6, 4), 1/6 - .02))



set.seed(41)
Rolls <- tibble(Roll = 1:n_Rolls,
                    Result = sample(Biased_dice$Side,
                                    prob    = Biased_dice$pi,
                                    size    = n_Rolls,
                                    replace = T)) %>%
  as_tbl_obs()

Rolls
```

```
Rolls %>%
  ggplot(aes(x = Result)) +
  geom_bar()
```



The result is shown in the figure above and just for simplicity we just focus on the events of rolling a Six. If we have no prior suspicion about the dice, the estimated probability is simply the relative frequency of Six.

```
Rolls %>%
  group_by(Result) %>%
  summarize(pi = n()/n_Rolls)
```

Result	pi
One	0.189
Two	0.177
Three	0.163
Four	0.160
Five	0.164
Six	0.147

In this case, we can simply note down that the *most likely value* is $\pi_{Six} = .147$, which is lower than the fair 0.167. But, note the slight ruggedness of the bar chart. Not a single bar is read as exactly 1/6, so the deviation of Six could have happened by chance. One way to approach this question is comparing the likelihoods $P(\text{Result} = \text{Six}|\pi = 1/6)$ and $P(\text{Result} = \text{Six}|\pi = .147)$. For that purpose, we create a new event variable `Six`, that indicates whether a roll is a Six (`TRUE`) or not (`FALSE`). Further, a *distribution function* is required that assigns these events their probabilities. Distribution functions can take very complicated forms 3.4.2, but in the case here it is the rather simple *Bernoulli distribution*.

The log-likelihood function of the Bernoulli distribution is just the sum of log-probabilities of any roll y_i.

$$LL_{\text{Bern}}(Y|\pi) = \sum_i \log(d_{\text{Bern}}(y_i, \pi))$$

Now, we can determine the *ratio of likelihoods LR* with different values for π. Note that on the logarithmic scale, what was a ratio, becomes a difference.

$$LR = \exp(LL_{\text{Bern}}(\text{Rolls}, .147) - LL_{\text{Bern}}(\text{Rolls}, \frac{1}{6})$$

```
Rolls <-  Rolls %>%
  mutate(Six = (Result == "Six"))

dbern <- function(y, pi) if_else(y, pi, 1 - pi)
LL_bern <- function(pi) sum(log(dbern(Rolls$Six, pi)))

pi_fair = 1/6
pi_est  = .147

exp(LL_bern(pi_est) - LL_bern(pi_fair))
```

```
## [1] 4846
```

Now, recall what a likelihood is: the probability of the observed data, under a certain model. Here, the data is almost 5000 times more likely with $p = .147$. In classic statistics, such likelihood ratios are routinely been used for comparison of models.

Previously, I have indicated that the relative frequency gives us the most likely value for parameter π (the case of a Bernoulli distributed variable), the *maximum likelihood estimate (MLE)*. The MLE is that point in the parameter range (here $[0; 1]$), which maximizes the likelihood. It is the point, where the data is most likely. In a similar way, the mean of Gaussian distributed measures is the maximum likelihood estimate for the distribution parameter μ. But, more advanced models do not have such a closed form, i.e., a formula that you can solve. Parameter estimation in classic statistics heavily grounds on numerical procedures to find *maximum likelihood estimates*, which I will outline now:

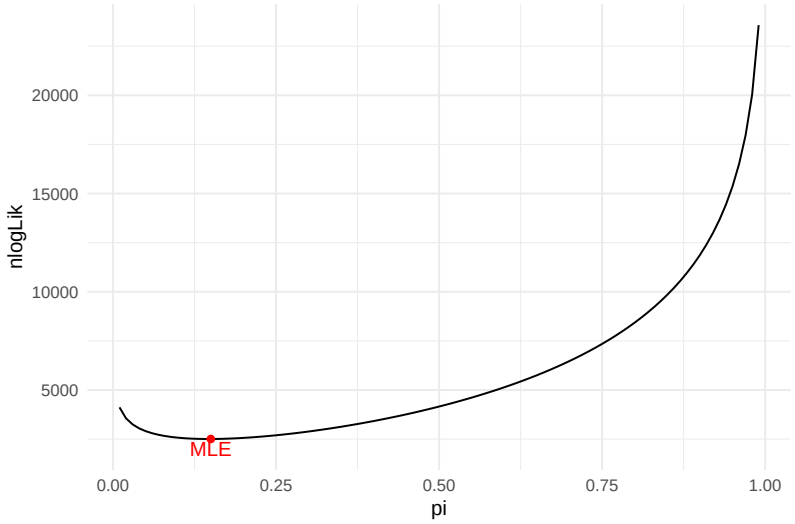
The probability function $d_{\text{Bern}}(y_i, \pi)$ has two parameters that vary, the result of a roll y_i and the assumed chance *pi*. The likelihood function, as we use it here, in contrast, takes the data as fixed and only varies on parameter π. By varying the parameter and reading the resulting likelihood of data, we can numerically interpolate the MLE. The most basic numerical interpolation method is a grid search, which starts at the left boundary of parameter range, zero in this case, and walks in small steps along a grid to the right

boundary (one). By convention, maximum likelihood estimation is performed by *minimizing the negative log-likelihood*. For the convenience, the following likelihood function has been vectorized, to make it work smoothly in a tidy processing chain.

```
LL_bern <- function(pi) map_dbl(pi, function(x) sum(log(dbern(Rolls$Six, x))))

LL_bern(c(.1, .2))
```

```
## [1] -2572 -2563
```

```
LL_grid <-
  tibble(pi = seq(.01, .99, by = .01)) %>%
  mutate(nlogLik = -LL_bern(pi),
         rank = min_rank(nlogLik),
         MLE = (rank == 1))

LL_grid  %>%
  ggplot(aes(x = pi, y = nlogLik)) +
  geom_line() +
  geom_point(data = filter(LL_grid, MLE), color = "Red") +
  geom_text(data = filter(LL_grid, MLE), label = "MLE", color = "Red", nudge_y = -500)
```



Because we use the *negative* log-likelihood, the value for π with maximum likelihood is the minimum of the likelihood curve. Here, $\pi_{\text{MLE}} = 0.15$, which is very close to the relative frequency we obtained above. The slight deviation is due to the limited resolution of the grid, but it is always possible to be more accurate by using a finer grid.

```
(MLE <- filter(LL_grid, MLE))
```

pi	nlogLik	rank	MLE
0.15	2507	1	TRUE

In classic statistics MLE is one of the most common methods for estimating parameters from models. However, most of the time, data is more abundant and there is more than one parameter. It is possible to extend the grid method to as many parameters as the model contains by just creating multi-dimensional grids and walk through them by the likelihood function. However, already a two-dimensional grid of rather coarse 100×100 would require the computation of 10.000 likelihoods. Classic statisticians have therefore developed optimization methods to identify the MLE more efficiently. Soon, we will turn our attention to Bayesian estimation, where the Likelihood plays a central role in estimation, too 3.5.

3.3.4 Bayesian and frequentist probability

All statisticians believe that relative frequencies satisfy Kolmogorov's axioms, but not everyone thinks that this is the only way. And that is where frequentist and Bayesian statistics diverge: Frequentest believe that *only relative frequencies* are valid measures for probability, whereas Bayesians believe that *certainty* is a measure of probability, too. As subtle this may sound, it has remarkable consequences in context with Bayes theorem.

> QUESTION: What is the chance that this dice will fall on Six? FREQUENTIST: I can only answer this question after a long-running series of dice rolls. BAYESIAN: I am 50% certain that this is a trick question, so I'd say 50% it never falls on Six. If this is not a trick question, this seems to be a dice from a well-known manufacturer, so I'd say 1/6. FREQUENTIST (frowning): And how does the manufacturer know? BAYESIAN (rolls eyes): ... by a long-running series of experiments.

The frequentist in the caricature is someone sticking to the principles: every probability must be produced by relative frequencies, which must be produced in repetitions of the same experiment. The Bayesian has no problem with long running sequences, but feels confident to use other sources of information. In a strictly frequentist view, this is like pulling numbers, just in the unit interval, out of the thin air. But, as we will see, Bayes theorem is amazingly useful, when we allow for probabilities that don't derive from long-running sequences. These are called *prior probabilities* and they can be used to factor in prior knowledge about the world.

Consequently, a Bayesian also accepts that levels of certainty after the experiment can be reported as probabilities. In contrast, a frequentist must insist that certainty can be interpreted as long-running series. In frequentist statistics, a common way to express ones level of certainty is the infamous p-value. And this is its definition: *A result is called statistically significant on level $\alpha = .05$, if drawing from the null distribution (an infinite number of times) will produce the observed result or a larger result in no more than 5% of cases.*

Another, and more preferable way of expressing ones certainty about a parameter (say, the population mean) is the *95% confidence interval*, which is expressed as two endpoints:

```
attach(Sec99)
```

```
Ver20 %>%
  lm(ToT ~ 1, data = .) %>%
  confint()
```

	2.5 %	97.5 %
(Intercept)	99.8	112

It is common to say: "we can be 95% certain that the true values is between these bounds", but the 95% confidence interval really is defined by assuming an (infinite) set of replications of the very same experiment and using relative frequencies: *The 95% confidence interval is constructed in such a way that, if the same experiment were repeated an infinite number of times, in 95% of these repetitions the true value is contained in the interval.*

You are not alone when you lack intuition of what the definition says and when you feel at unease about where all these experiments are supposed to come from. It seems as if a a true frequentist cannot imagine the future other than by a series of long-running experiments. In Bayesian statistics, the level of certainty is expressed as a *proposition about one's state of knowledge*, like : *Based on my data I am 95% sure that there is a difference.* Equating level of certainty with probability directly, without taking the detour via relative frequencies, may be a little lax, but it leads to remarkably intuitive statements on uncertainty. In Bayesian statistics the *credibility interval* is defined as: *With a probability of 95%, the true value is contained.* Since there seems to be no external criterion (such as a series of experiments, imagined or not), Bayesian statistics often faced the criticism of being subjective. In fact, if we imagine a certainty of 95% as some number in the researchers mind, that might be true. But, it is quite easy to grasp certainty as an objective quantity, when we assume that there is something at stake for the researcher and that she aims for a rational decision. In the previous chapter I have illustrated this idea by the example of carrying an umbrella with you (or not) and the 99 seconds claim. Generally, it helps to imagine any such situation as a gamble:

if you bet 1 EUR that the true population mean is outside the 95% credibility interval, as a rational person I would put 19 EUR against.

In effect, the Bayesian certainty is a probability in mathematical terms, without the necessity to implement it as a relative frequency. That liberates our reasoning from the requirement to think of long-running series. In particular, it allows us to enter non-frequentist probabilities into Bayes theorem, resulting a statistical framework that captures the dynamics of belief in science 3.3.6. Before we come to that, I will explain Bayes theorem in another context of updating knowledge, medical diagnostics.

3.3.5 Bayes theorem

Bayes' theorem emerges from formal probability theory and therefore is neither Bayesian nor frequentist. Essentially, the theorem shows how to calculate a conditional probability of interest $P(A|B)$ from a known conditional probability $P(B|A)$ and two marginal probabilities $P(A)$ and $P(B)$:

$$P(A|B) = \frac{P(A)P(B|A)}{P(B)}$$

The proof of the theorem is rather simple and does not require repetitiuon here. But, Bayes' theorem can lead to rather surprising conclusions, which can be illustrated by example of a medical screening test, as follows:

In the 1980, the human immunodeficiency virus (HIV) was discovered and since then has become a scourge for humanity. Given that the first outbursts of the disease raged among homosexual men, it is not really surprising that some conservative politicians quickly called for action an proposed a mandatory test for every one, with the results to be registered in a central data base. Without much of a stretch it may seem justifiable to store (and use) the information that someone is carrying such a dangerous disease. The problem is with those people who do not carry it, but could be mis-diagnosed. These are called *false-positives*. Let#s assume the power of a screening test has been assessed by examining samples of participants where it is fully known whether someone is a carrier of the virus $C+$ or not $C-$. The result is a *specificity* of 95%, meaning that 95% of $C-$ are diagnosed correctly ($P(T-|C-)$), and a *sensitivity* of 99%, meaning that 99% with $C+$ are diagnosed correctly ($P(T+|C+)$). The question that Bayes' theorem can answer in such a situation is *How many citizens would be registered as HIV carrying, although they are not?*. For this to work, we must also know the probability that someone randomly chosen from the population is a carrier ($P(C+)$) and the proportion of positive test results $P(T+)$.

$$
\begin{array}{lll}
P(C+) & & =.0001 \\
P(C-) & =1-P(C+) & =.9999 \\
P(T+|C+) & & =.99 \\
P(T-|C+) & =1-P(T-|C+) & =.01 \\
P(T-|C-) & & =.95 \\
P(T+|C-) & =1-P(T-|C-) & =.05 \\
P(T+) & =P(C+)P(T+|C+)+P(C-)P(T+|C-) & \approx.05 \\
P(T-) & =1-P(T+) & \approx.95
\end{array}
$$

How do these numbers arise? The first, $P(C+)$) is the proportions of HIV carriers in the whole population. If you have no test at all, that is your best guess for whether a random person has the virus, your *prior knowledge*. Then, the validation study of the test provides us with more *data*. The study examined the outcome of the test ($T+$ or $T-$) in two groups of participants, those that were knowingly carriers $C+$ and those that were not $C-$. This is where the four conditional probabilities come from. Finally, we need the expected proportion of positive test results $P(T+)$, which we compute as a marginal probability over the two conditions. Because non-carriers $C-$ dominate the population by so much, the marginal probability for a positive test is almost the same as the probability for a positive test amomg non-carriers $P(T+|C-)$.

All conditional probabilities here emerge from a validation study, where it is known upfront whether someone is a carrier or not. What matters for the application of the screening test is the reverse: what is learned by test about carrier status? In the following the test is being characterized by two types of errors that can occur: false alarms and misses.

- False alarms: What is the probability that someone will be registered as carrier, although being a non-carrier? That is: $P(C-|T+)$. When the false alarm rate is low, the test is called to have good *specificity*.
- Misses: Which proportion of the tested population are carriers, but have a negative test result (and act accordingly)? That is: $P(C+|T-)$. When the probability of misses is low, the test is called to have good *sensitivity*.

Using Bayes' theorem, we obtain the following probability for false alarms:

$$
\begin{aligned}
P(C-|T+) &= \frac{P(C-)P(T+|C-)}{P(T+)} \\
&\approx \frac{.9999 \times .05}{05} \\
&\approx .9999
\end{aligned}
$$

Obviously, testing the whole population with the screening test would result in disaster. Practically everyone who is registered as carrier in the data base

is really a non-carrier. In turn, the probability that a person has the virus despite a negative test result is:

$$\begin{aligned} P(C+|T-) &= \frac{P(C+)P(T-|C+)}{P(T-)} \\ &\approx \frac{.0001 \times .01}{.95} \\ &\approx .000001 \end{aligned}$$

We see a strong asymmetry in how useful the test is in the two situations. Specificity of the test is rather low and stands no chance against the overwhelming prevalence of non-carriers. In the second use case, prevalence and high test sensitivity work in the same direction, which results in a fantastically low risk to err. That is what Bayes' theorem essentially does: it combines prior knowledge (prevalence, $P(C+)$) against obtained evidence and produces *posterior* knowledge. In the following section we will see how this makes Bayes' theorem a very useful tool in a context that is all about updating knowledge, like in science. In the case of medical diagnostics, the probabilty $P(C+|T+)$ can be called a posterior probability, but it becomes prior knowledge in teh very moment that new data arrives and we can update our knowledge once again. A reasonable strategy arises from the asymmetry of certainties: a negative test almost certainly coincides with being a non-Carrier, whereas identifying true carriers remains problematic. Follow-up research should therefore capitalize on reducing false alarms and use a test with extreme specificity (= absence of false alarms) at the expense of sensitivity.

3.3.6 Bayesian dynamics of belief

Bayes' formula is just a law that follows directly from probability theory. It can easily be illustrated in terms of frequencies: If the number of false alarms is very small in a small group, it gets large in a very large group. What makes such an innocent formula the patron of Bayesian Statistics?

Bayes' formula can be seen as a sort of knowledge processing device. At the example of HIV test, what we want to know is how likely it is that someone is infected. That is something we can already do without any test, if we have an idea about the prevalence of infections. It already helps to know that only 1 in 10.000 has it. It is a rare desease, so usually no reason to get paranoid! In contrast, if you are running blood bank, knowledge of prevalence is just not good enough and a test is required. This test is used not to make up our minds, but to update our minds. Bayes' formula is just a rational way of updating knowledge by combing what one knew before and what one learns from the present data. A Bayesian estimation takes *prior knowledge* $P(\theta)$ and

the *likelihood* $P(\text{Data}|\theta)$, representing the data as input and the output is *posterior knowledge* $P(\theta|\text{Data})$:

$$P(\theta|\text{Data}) = \frac{P(\theta)P(\text{Data}|\theta)}{P(\text{Data})}$$

As it turns out the marginal likelihood $P(Data)$ cannot be known or easily estimated. The reason why we can still use Bayes rule for estimation is that this term does not depend on the parameter vector θ, which is our search space, meaning that it is practically a constant. For parameter estimation, Bayes rule can be reduced to saying that the posterior certainty is *proportional to* the product of prior certainty and evidence:

$$\text{posterior} \propto \text{prior} \times \text{likelihood}$$

Recall that the likelihood is the probability of the data at hand, given parameter θ. This is typical for frequentist statistics, which assumes that there is one true value and all randomness is caused by fluctuations in the set of possible samples. Your data is just one fish from the pond. Bayesians typically feel that data is the real thing and that parameter θ is never really true, but represents teh current state of knowledge.

Brains process data, too, and some argue that the behaviour of brains can be described as Bayesian engines. It is true that prior knowledge is something we use everyday, and many things we do know because of our genes. For example, parents sometimes report that their toddlers abruptly start to reject food that is green. This is often perceived as stubborn behaviour, because of vitamines, fiber, lack of sugar and the greenhouse effect. If it is true, that our ancestors headed their ways through a savannah, the child may actually be right. Savannah flora's worst enemy are vegetarian savannah animals, and in an arms race, plants have developed an abundance of weapons, such as a tough structure, scarcity of nutriants, thorns, blades and poisons. The child knows that green stuff usually is not edible or even dangerous. This prior knowledge is so over-whelming, that it costs modern parents a lot of patience to update their childrens knowledge of food, and reportedly many fail. A Bayesian statistician would call this a *strong prior*.

Science is what only some central nervous systems do, but the principle is the same: we have a theory which we are unsure about; data is collected and after that we may find this theory more likely or reject it. Testing theories is not a one-shot. Some theories have been tested many times For example, in the famous Stroop effect, participants have to name the ink color of a word. When the word is itself a color word and refers to a different color, response times typically increase. This effect has been replicated in many dozens of published studies and, probably, thousands of student experiments.

Cumulative evidence is so strong that, would someone repeat the experiment another time and find the reverse effect, no one would seriously take this as a full debunk of decades of research. In Bayes' formula, prior and posterior are both probabilities, which means we can use the posterior from experiment n as prior for experiment $n + 1$, or:

> Today's posterior is tomorrow's prior.

There is no principled difference between prior and posterior knowledge. They are just degrees of belief at different points in time, expressed as probabilities. Both can differ in strength: prior knowledge can be firm when it rests on an abundance of past evidence. But, prior knowledge can also be over-ruled when a lot of data disproves it. Evidence in data varies in strength: for example, the more observations, the larger the (negative) likelihood becomes, which means stronger evidence. If measures are more accurate (and less disturbed by randomness), smaller sample sizes suffice to reach conclusions of similar strength. This is why larger sample sizes are preferred and why researchers try to improve their methods.

The problem with frequentist statistics is that it has not developed general methods to incorprorate prior knowledge in the analysis. It is even worse: When practicing null hypothesis significance testing, which is very common among frequentists, you strictly *must not update* your degree of belief during the study and act accordingly. Neither is there a way to express one's prior belief when doing a t-test, nor may you adjust sample size ad hoc until satisfactory certainty is reached. Classic data analysis pretends as if no one has ever done any such an experiment before. Also, it is strictly forbidden to invite further participants to the lab, when the evidence is still to weak. If you planned the study with, say, $N = 20$, this is what you must do, no less no more. If you reach your goal with less participants, you must continue testing. If you are unsatisfied with the level of certainty, the only permissible action is dump your data and start over from zero. The frequentist denial of incremental knowledge is not just counter-intuitive in all science, it is a millstone around the neck of every researcher.

3.4 Statistical models

It is a scientific, maybe even naturalistic, principle that every event to happen has its causes (from the same universe). The better these causes are understood, the better will be all predictions of what is going to happen the next moment, given that one knows the complete state of the universe and the laws of physics. *Laplace demon* is a classic experiment of thought on the issue: the demon is said to have perfect knowledge of laws of physics and about the universe's current state. Within naturalistic thinking, the demon should be able

to perfectly predict what is going to happen in the next moment. Of course, such an entity could never exist, because it would actually be a computer that matches the universe in size (and energy consumption). In addition, there are limits to how precisely we can measure the current state, although physicist and engineers have pushed the limits very far. Behavioural science frequently lacks this precision, which results in rather imperfect knowledge about the state. We are often left with a good amount of uncertainty.

When Violet did her experiment to prove the superiority of design B, the only two things she knew about the state of affairs was that the participant sitting in front of her is member of a very loose group of people called the "typical user" and the design her or she was exposed to. That is painstakingly little information on what's currently going on in the participant's central nervous system. Her lack of knowledge is profound but still not a problem as the research question was on a gross scale itself, too. Not what happens to individual users needed to be described, but just the difference in *average* duration.

Imagine Violet and a colleague had invented a small game where they both guess the time-on-task of individual participants as they enter the lab. Who comes closest wins. As both players are smart people, they do not just randomly announce numbers, but let themselves guide by data of previous sessions. A very simple but reasonable approach would be to always guess what the average ToT in all previous sessions has been. In **??**, we will call this a grand mean model.

Of course, Violet would never expect her grand mean model to predict the accurate outcome of a session. Still, imagine a device that has perfect knowledge of the website, the complete current neural state of a the participant and the physical environment both are in. As an all-knowing device it would also have complete knowledge on how neural states change. With this device, Jane could always make a perfect prediction of the outcome. Unfortunately, real design researchers are far from Laplace demonism. Routinely borrowing instruments from social sciences, precision of measurement is humble and the understanding of neural processes during web navigation is highly incomplete. Participants vary in many complex ways in their neural state and this makes a myriad of *small unmeasured forces (SMURF)* that can steer an individual users towards or away from the average. Laplace demon would have perfect knowledge of all forces and would produce a perfect prediction. Violet, in contrast, is completely ignorant of any SMURFs and her predictions will be way off many times.

A common way to conceive this situation is that all measures are composed of a *structural part*, that captures the relation between known and measured forces and a *random part*, that serves as a bucket for unmeasured forces.

$$\text{Measure}_i = \text{structural part} + \text{random part}_i$$

Note that Measure and Random part both have an observation-level index i, which means they are *unique* per observation i. The structural part does not have such an index, because it is usually designed as a *universal* statement, a proposition that holds for all the observations. If this were different, there would be no way to make predictions from the model. After a minimally sufficient introduction to the matter, the remainder of this section will focus on the random part and we will discover that randomness is *not* completely arbitrary, but falls into certain *shapes of randomness*. And for the scope of this book, a *statistical model* is defined as a *mechanism that separates the structural from the random part* to some extent.

3.4.1 The structural part

Generally, statistical models consist of these two parts: a structural part that describes the association between measures and the random part, which is the sum of all small unmeasured forces (SMURFs). The structural part of statistical models is called so, because it describes what all observations supposedly have in common. The structural part is what encodes the structure of a study and the dependencies between measures.

The most simple structure is just that an observation was taken on a member of a population. If we speak of a population (rather than a set), we typically imply that members of the population have something in common. The most simple structural part is therefore taking the mean of the population, which is also called a Grand Mean Model **??**. From here on, μ_i is the result of the structural part for one observation, which for a GMM, is just a constant β_0.

$$\mu_i = \beta_0$$

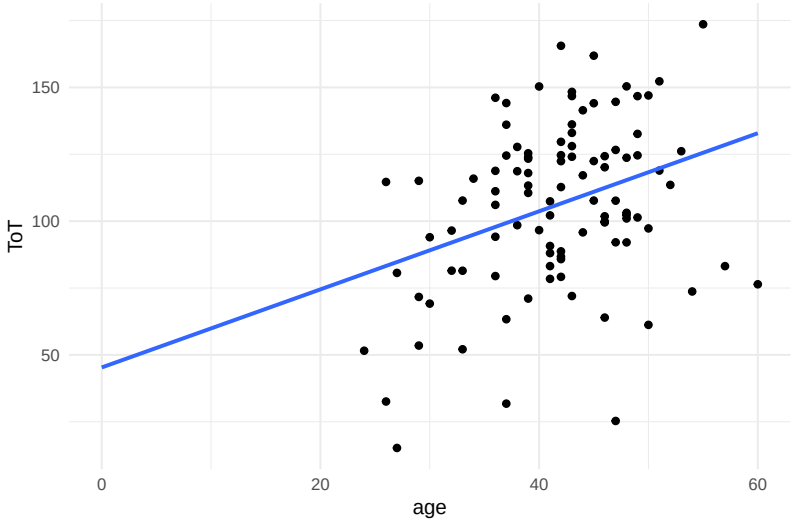
We will see throughout chapters 4 and 5, how the structural part encodes complex dependencies between measures.

The structural part makes a universal statement about the relation between measures. This can basically be anything. However, in many (if not most) statistical models, it is common to separate the set of measure into exactly one *response* (or outcome, or dependent variable) and a set of predictor variables. In the GMM above, there are no predictors, just a constant value.

In the guessing game, Violet could try to improve her guesses, by also taking age of participants into account. Older people tend to be slower. Violet could create a plot from past records. The ellipsoid form of the point cloud indicates that ToT is somehow depending on age. Violet draws a straight line with an upward slope to approximate the relationship. It seems that zero year old persons has an average ToT of around 45 seconds, which increases to around 120 seconds for 50 year old. Arithmetically, this is an increase of around 1.5 seconds per year of age.

```
##
## Call:
## lm(formula = ToT ~ age, data = Sec99$Ver20)
##
## Coefficients:
## (Intercept)          age
##       45.31         1.46
```

```
Sec99$Ver20 %>%
  ggplot(aes(x = age, y = ToT)) +
  geom_point() +
  geom_smooth(method = "lm", se = F, fullrange = T) +
  xlim(0,60)
```



Violet can use this information to improve her chance of winning. Instead of stoically calling the population mean, she uses a linear function as predictor: $45 + (age) 1.5 $. This is called a *linear association* and the general notation is

$$\mu_i = \beta_0 + \beta_1 x_{1i}$$

where β_0 is the intercept value, X_1 is a measure (age) and β_1 is a *coefficient*. The primary purpose of using a model on data is to estimate the coefficients.

Linear models are very common in statistical modeling, but the structural part of a model can basically take all mathematical forms. For example:

- two predictors with a linear relationship: $\mu_i = \beta_0 + \beta_1 x_{1i} + \beta_1 x_{2i}$

- a parabolic relationship: $\mu_i = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i}^2$
- a nonlinear learning curve: $\mu_i = \beta_{\text{asym}}(1 + \exp(-\beta_{\text{rate}}(x_{\text{training}} + \beta_{\text{pexp}})))$
- the difference between groups A and B, where x_1 is a membership (dummy) variable coded as $A \to x_1 := 0, B \to x_1 := 1$: $\mu_i = \beta_0 + \beta_1 x_{1i}$

In the vast majority of cases, the structural part is the one of interest, as this is where researchers transform their theoretical considerations or practical questions into a mathematical form. The parameters of the likelihood function are being estimated and answer the urging questions, such as:

- Is the design efficient enough? (β_0)
- By how much does performance depend on age? (β_1)
- Under which level of arousal does performance peak? (determining the stationary point of the parabola)
- How fast people learn by training (β_{rate})
- By how much design B is better than A (β_1)

A subtle, but noteworthy feature of structural part is that μ_i and x_i have indicators i. Every observation i has their own measures and therefore the realization of the structural part gets a unique expected value. In contrast, the coefficients β_0, β_1 asf. are universal values that apply for all observations at once.

When all all measures y, x are known and all coefficients $\beta_\cdot$ have been estimated, we can solve the structural part and obtain a unique value μ_i for all observations. Considering that $y_i = \mu_i +$ random part, that means we can also view model estimation of a model as a separation of structural part and random part. Unfortunately, it is not sufficient to just have measures and a structural part. We also need to know the shape of randomness and put this into the model.

3.4.2 Distributions: shapes of randomness

Review the formula $\text{Measure}_i = \text{structural part} + \text{random part}_i$. What do we actually have? It turns out that the only part we know for sure are measures y_i and $x_{\cdot i}$. The structural part contains the answer to our research question, so this is what we want to estimate. Unfortunately, there is another unknown, the *randompart*, which we literally need to subtract to get to our answers. Obviously, this would never work if the random part were random in the sense of being completely arbitrary. Fortunately, randomness is not arbitrary, but has a shape, which is described as a probability (or density) distribution. These distributions typically belong to a certain family of distributions, e.g. Poisson disributions or Gaussian distributions. By assuming a certain shape of randomness, we know a lot more about the random component, which will

make it possible to flesh out the structure. Choosing the most appropriate distribution therefore is a crucial step in creating a valid statistical model. In this section I will introduce in more depth how probability and density distributions describe the shape of randomness.

3.4.2.1 Probability and density distributions

The random part of a statistical model contains all (small) unmeasured forces on the response y_i. When using the grand mean model, the only information we are using is that the person is from the target population. Everything else is left to the unobserved SMURFs and that goes into the random part of the model. Fortunately, SMURFs don't work completely arbitrary and in practice there is just a small number of recognizable shapes randomness can take. These patterns can be formulated mathematically as probability and density distributions. A probability distribution is given by a probability mass function (PMF) that assigns *probabilities to outcomes*, such as:

- probability of *task success* is .81
- probability of *99 seconds or better* is .22
- probability of all SMURFs together pushing a persons *IQ beyond 115* is around .33

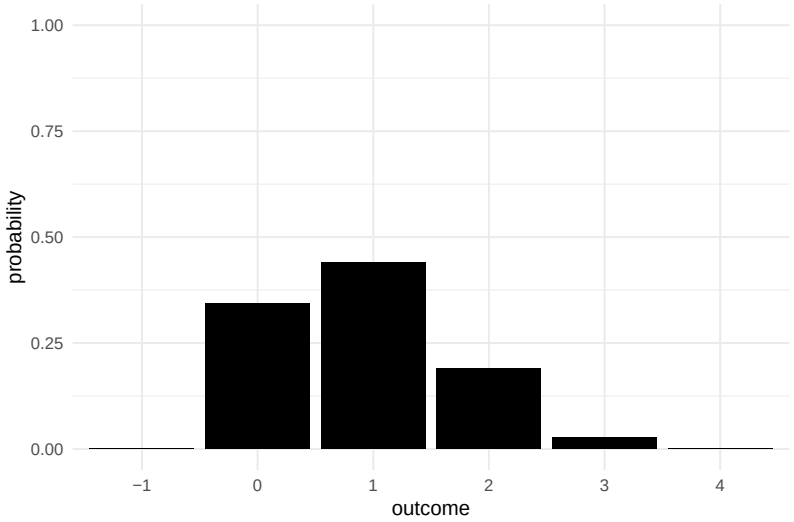
Let's see this at a small example: Consider a participant who is asked to complete three tasks of constant difficulty, with a chance of .3 for each one to be solved. The outcome variable of interest is the number of correct results, which can take the values 0, 1, 2 or 3. Under idealized conditions (but not removing randomness), the Binomial distribution assigns every possible outcome a probability to occur, it is given as:

$$P(y|p,k) = \binom{k}{y} p^y (1-p)^{k-y}$$

A Binomial distribution with probability of success $p = .3$ and $k = 3$ tasks looks like the following:

```
D_three_tasks <-
  tibble(y = -1:4,
             outcome = as.character(y),
             probability = dbinom(y, size = 3, prob = 0.3),
             cumul_prob     = pbinom(y, size = 3, prob = 0.3))


D_three_tasks %>%
```

```
ggplot(aes(x = outcome, y = probability)) +
geom_col(fill = 1) +
ylim(0,1) +
theme(legend.position="none")
```



As we can see, there exist four possible outcomes for a sequence of three tasks: zero, one, two and three tasks correct. We also see that the most likely outcome is one correct task, which occurs with a probability of $P(y = 1) = 0.441$. At the same time, it is surprisingly likely to fail at all tasks, $P(y = 0) = 0.343$. We may also look at *combined events*, say the probability for less than two correct. That is precisely the sum $P(y \leq 1) = P(y = 0) + P(y = 1) = 0.784$. We can bundle basic events by adding up the probabilities. An extreme case of that is the universal event that includes all possible outcomes. You can say with absolute certainty that the outcome is, indeed, within zero and three and certainty means the probability is 1, or: $P(0 \leq y \leq 3) = 1$. Simply in order to comply with the third Kolmogorov axiom, all probability (and density) distributions have *probability mass of One*. More precisely, the area covered by the function must be exactly One. Another extreme outcome is that four out of three tasks are correct, which is impossible, or $P(y > 3) = 0$.

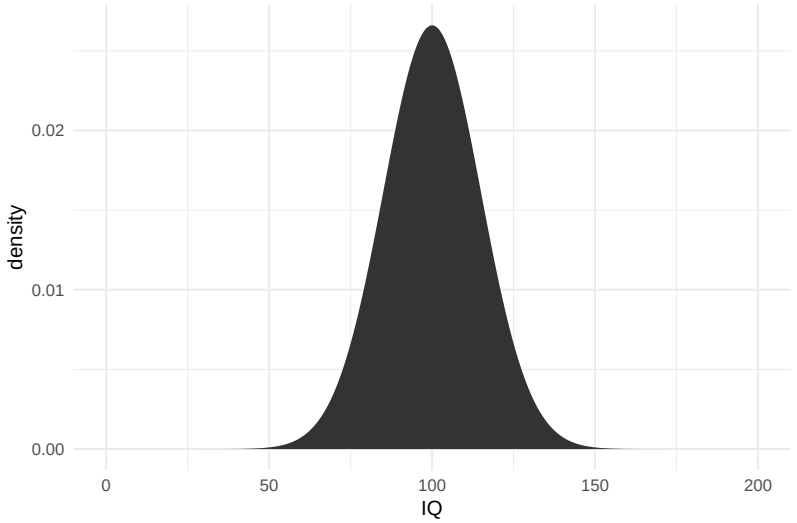
Number of successes is a countable measure, and are therefore called *discrete*. If we want to obtain the probability of combined events, we just have to sum probabilities over all the included events, which is like stacking the blocks in the figure above. This approach fails, if the outcome is not discrete, but *continuous*. On a continuous scale, every outcome is a line, in the strictly geometrical sense. That makes the probability masses of any such point extensionless, too,

i.e. exactly zero. The probability that the outcome of a continuous measure is *exactly*, say, 42.18 is zero.

when the measure is continuous, rather than discrete, we cannot assign non-zero probabilities to exact outcomes. But, we can give non-zero probabilities to ranges of outcomes. Consider the distribution of intelligence quotients (IQ) scores. Strictly spoken, the IQ is not continuous, but for instructional reasons, let us assume it were and we can give the IQ with arbitrary precision, for example 115.0, 100.00010... or $\pi * 20$..

IQ scores are *designed to follow the Gaussian* distribution with a mean of 100 and a standard distribution of 15, so we can easily simulate some data for 200 participants undergoing an IQ test.

```
D_IQ <- tibble(IQ = 0:2000/10,
               density = dnorm(IQ, 100, 15),
               cdf = pnorm(IQ, 100, 15),
               SE = (IQ > 85 ) * (IQ <= 115) * density,
               around_100 = (IQ >= 99 ) * (IQ <= 101) * density,
               PDF_085 = (IQ < 85) * density,
               PDF_115 = (IQ < 115) * density,
               label = str_c("P(IQ < ", IQ, ")"))

D_IQ %>%
  ggplot(aes(x = IQ, y = density)) +
  geom_area()
```

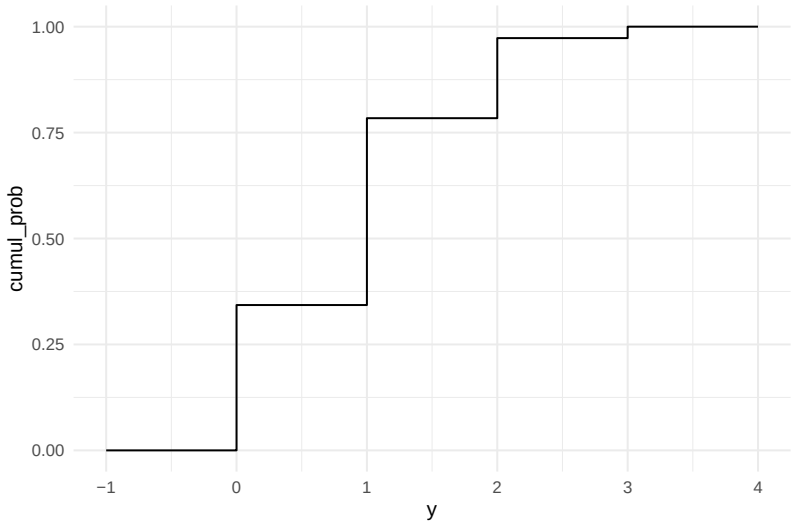


We observe that the most likely IQ is 100 and that almost nobody reaches scores higher than 150 or lower than 50. But, how likely is it to have an IQ of

exactly 100? The answer is Zero. With continuous measures, we can no longer think in blocks that have a certain area. In fact, the probability of having an IQ of *exactly* 100.00...0 is exactly zero. The block of IQ = 100 is infinitely narrow and therefore has an area of zero. That is why for continuous outcome variables, we cannot have a PDF, but a *density distribution function (DDF).* At the example of the above Gaussian distribution, we can still identify the point of highest density (IQ = 100), but the density scale is *not probabilities.* What DDFs share with PDFs is that the area under the curve is always exactly One. In addition, probabilities still arise from density when taking intervals of values, which is nothing but combined events. Before we return to that, another way of specifying statistical distributions must be introduced.

A cumulative distribution function is the *integral* of a PDF or DDF. *Cumulative distribution functions (CDF)* render how *probability mass* increases when moving from left to right over the outcome scale. Because probability and density are non-negative, a CDF always is *monotonously increasing.* Below we see the CDF of the three-tasks situation. The function starts at Zero, then moves upwards in increments and reaches One at the right end of the scale.

```
D_three_tasks %>%
  ggplot(aes(x = y, y = cumul_prob)) +
  geom_step() +
  ylim(0,1)
```



Using the CDF, the probability for any outcome to be in a certain range can be written as the difference between the upper and the lower limit of cumulative probability. For example, the probability of having zero or one tasks correct is

$P(\text{correct} \leq 1) = \text{CDF}_{Binom}(x = 2, k = 3, p = .3)$

or

```
pbinom(1, 3, .3)
```

```
## [1] 0.784
```

On the PDF, we calculated the probability of combined events by summing. With the PDF, we can calculate the probability for any range by subtraction. The probability of more than one tasks correct is:

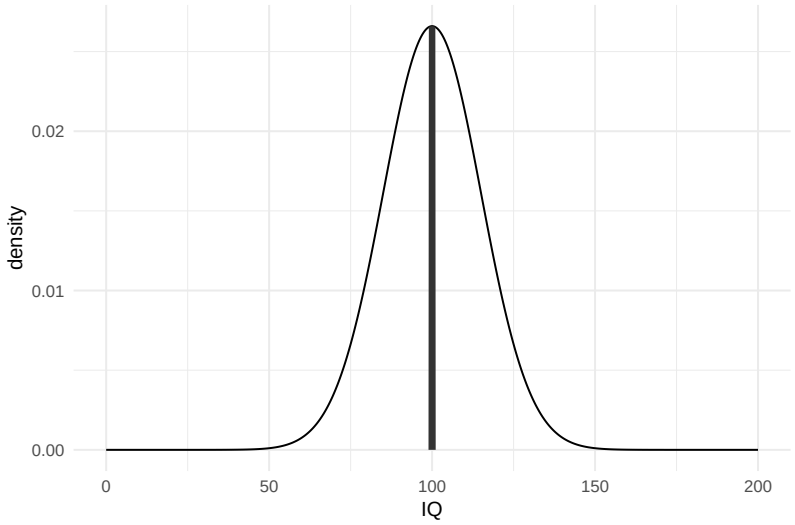
$P(\text{correct} \geq 2) = \text{CDF}_{Binom}(x = 3, k = 3, p = .3) - \text{CDF}_{Binom}(x = 1, k = 3, p = .3)$

```
pbinom(3, 3, .3) - pbinom(1, 3, .3)
```

```
## [1] 0.216
```

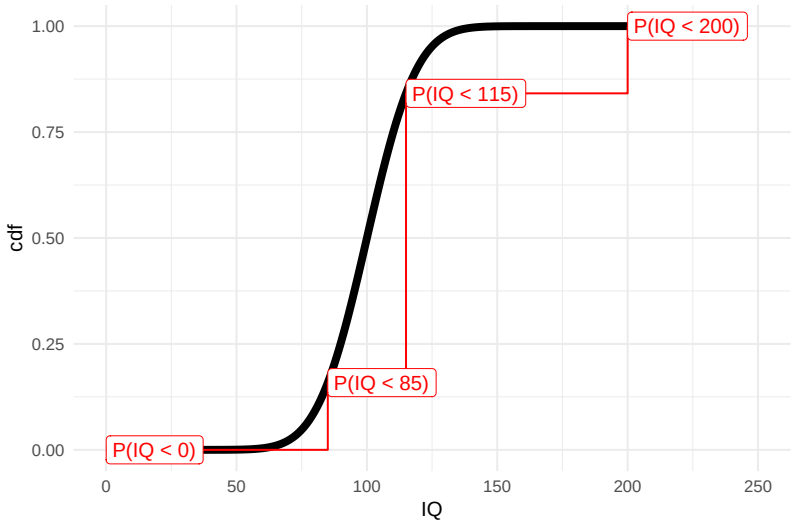
With this method, we can also treat CDFs and obtain probabilities for intervals of values. Practically, nobody is really interested in infinite precision, anyhow. When asking *"what is the probability of IQ = 100?"*, the answer is *"zero"*, but what was really meant was: *"what is the probability of an IQ in a close interval around 100?"*. Once we speak of intervals, we clearly have areas larger than zero. The graph below shows the area in the range of 99 to 101.

```
D_IQ %>%
  ggplot(aes(x = IQ, y = density)) +
  geom_line() +
  geom_area(aes(y = around_100))
```



The size of the area is precisely the probability of such an event (IQ between 85 and 115). But, how large is this area exactly? As the distribution is curved, we can no longer simply stack virtual blocks. It is the CDF giving the us probability mass up to a certain value, i.e. the area under the curve up to a point. Continuous distributions have no PDFs CDFs, too, and the the graph below shows the CDF for the IQs. We observe how the curve starts to rise from zero at around 50, has its steepest point at 100, just to slow down and run against 1.

```
D_IQ %>%
  ggplot(aes(x = IQ, y = cdf)) +
  geom_line(size = 2) +
  geom_step(data = filter(D_IQ, IQ %in% c(0, 85, 115, 200)), color = "red") +
  geom_label(data = filter(D_IQ, IQ %in% c(0, 85, 115, 200)), aes(label = label), color = "
  xlim(0, 250)
```



Underneath the smooth CDF, a discrete probability function reduces the continuum into three horizontal intervals, that have a certain total height associated, as well as a stepsize. The height is the total probability up to this point, whereas stepsize is the probability for an IQ measure to fall within an interval. No that the functionb is block-ified, we can assign probabilities to intervals of values, by simple subtractions on the CDF:

$$\begin{aligned} CDF(IQ \geq 99) &= 0.473 \\ CDF(IQ \leq 101) &= 0.527 \\ CDF(99 \leq IQ \leq 101) &= P(IQ \leq 101) - P(IQ \leq 99) \\ &= 0.683 \end{aligned}$$

To summarize, a density distribution function (DDF) is a function that covers an area of exactly One. A probability distribution function (PDF) is a discrete density function, where the area is composed of discrete blocks. A PDF has the special property to return plain probabilities. With continuous density this does not work, because it is composed of infintely many and therefore infinetely small blocks. By using the cumuluative distribution function, we can extract probabilities of intervals of values.

3.4.2.2 Features of shape

Density distributions can take a variety of shapes. Most distributions are shaped as one hill, like the two distributions covered in the previous section. This means that there is just one maximum point of density and is called

a *unimodal* function. Besides discreteness or continuity, as covered in the previous section, unimodal distributions can have several other features:

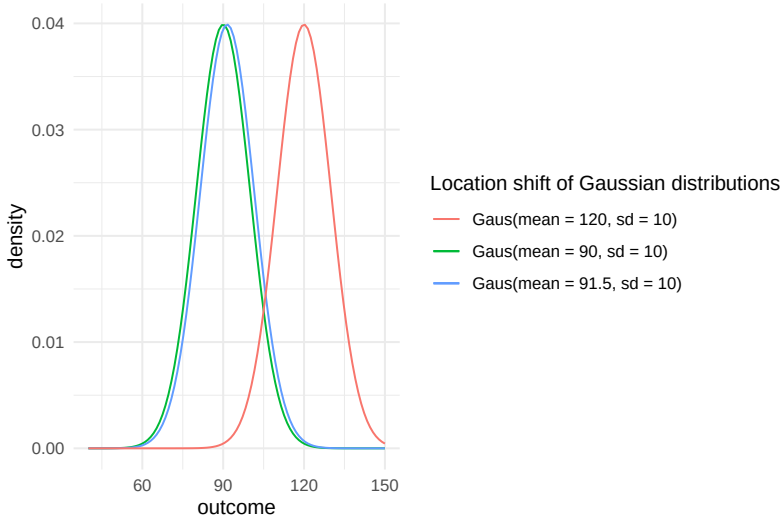
1. range of support
2. location on the x axis
3. dispersion
4. skew

In this book I advocate the thoughtful choice of density distributions rather than doing batteries of goodness-of-fit to confirm that one of them, often the Gaussian, is an adequate approximation. It usual is straight-forward to determine whether a measure is discrete (like everything that is counted) or (quasi)continuous and that is the most salient feature of distributions. A second, nearly as obvious, feature of any measure is its range. Practically all physical measures, such as duration, size or temperature have natural lower bounds, which typically results in scales of measurement which are non-negative. Counts have a lower boundary, too (zero), but there can be a known upper bound, such as the number of trials. *Range of support* will be an important feature when we approach Generalized Linear Models 6 and is covered there in more detail. For now, it suffices to say that all standard density distributions fall into one of three classes:

1. *unbound* distributions range from minus to plus infinity and are suitable for measures that are practically unbound. One example is using the Gaussian distributions for IQ scores.
2. distributions *bound at zero* cover the range from Zero up to plus infinity. It is suitable for measures that have a natural lower boundary (of Zero) and no upper boundary for practical reasons. An example is the number of errors during the execution of a complex task.
3. *double-bound* distributions have a lower and an upper limit. Typically the lower limit is Zero. And example is the distribution of task success that we saw in the previous section.

Location of a distribution is the most important feature for almost all statistical models covered later in this book. When we say: "Time-on-task increases by 1.5 seconds per year of age", this translates into a shift on the x-axis. The mean of the ToT distribution is 90 for a 30-year old, 91.5 for a 31-year old and 120 for a 50 year old person. When an experimenter asks for the difference of two designs in ToT, this is purely about location. In the class of models covered in this book, location is always indicated by the arithmetic mean. Still, it can theoretically also be represented by the median or the mode (in case of uni-modality).

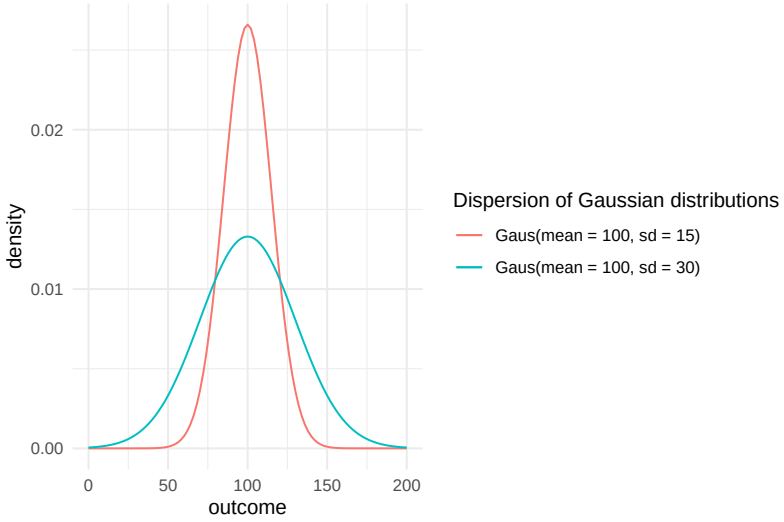
```
tibble(location = c(90, 91.5, 120), sd = 10) %>%
  ggplot(aes(x = location)) +
  stat_function(fun = dnorm, args = list(mean = 90, sd = 10),
                mapping = aes(colour = "Gaus(mean = 90, sd = 10)")) +
  stat_function(fun = dnorm, args = list(mean = 91.5, sd = 10),
                mapping = aes(colour = "Gaus(mean = 91.5, sd = 10)")) +
  stat_function(fun = dnorm, args = list(mean = 120, sd = 10),
                mapping = aes(colour = "Gaus(mean = 120, sd = 10)")) +
  xlim(40, 150) +
  labs(colour="Location shift of Gaussian distributions", x = "outcome", y = "density")
```



Dispersion represents the level of *variation* around the center of the distribution. In Gaussian distributions, this is given by the standard deviation (or variance). The most significant impact of dispersion is that spreading out a distribution reduces its overall density at any point. In uni-modal distributions, we can think of the mode as the most typical value, the one you should place your bet on. With more variance, the density at this point is reduced, and you should not bet to hard on it any more. IQ measures usually have a mode at a value of 100, with a standard deviation of 15. The density of a usual IQ distribution at an IQ of 100 is 0.027. If IQs had a standard deviation of 30, the density at 100 would be reduced to 0.013. If you were in game to guess an unfamiliar persons IQ, in both cases 100 would be the best guess, but you had a considerable higher chance of being right, when variance is low.

```
ggplot(tibble(location = 100), aes(x = location)) +
  stat_function(fun = dnorm, args = list(mean = 100, sd = 15),
                mapping = aes(colour = "Gaus(mean = 100, sd = 15)")) +
```

```
  stat_function(fun = dnorm, args = list(mean = 100, sd = 30),
                mapping = aes(colour = "Gaus(mean = 100, sd = 30)")) +
  xlim(0, 200) +
  labs(colour="Dispersion of Gaussian distributions", x = "outcome", y = "density")
```



Large location shifts (between experimental conditions) are usually a delight for researchers, as most often research questions are about location changes. In contrast, large dispersion is often associated with uncertainty (or imperfect measures). In Gaussian models this uncertainty or error is directly represented as the standard deviation of residuals, also called the *standard error*. We will re-encounter this idea when turning to model criticism 7.1.

Dispersion also plays a role as a measure for *variation within a population*, such as the differences in IQ. This is most prominent in psychometrioc situations, where we want to discern between individuals. A psychometric tool, such as the IQ test, must produce dispersion, otherwise it would not discern between individuals We will re-encounter this idea when turning to multi-level models 5.

Most distributions routinely used in statistics have one or two parameters. Generally, if there is one parameter this determines both, location and dispersion, with Poisson distributions being an example (see below). Two-parameter distributions can vary location and dispersion, to some extent. However, for most distributions, both parameters are tied to location and dispersion, simultaneously, and in sometimes convoluted ways. The Gaussian distribution is a special case as parameter μ purely does location, whereas σ is just dispersion.

As was said above, most statistical models use the mean of a distribution to render differences between situations. The mean is but one measure of central

tendency, with the median and the model being alternatives. If a distribution is symmetric, such as Gaussian distributions, these three parameters are the same. The mean divides the distribution into two areas of same size, and it is the point of highest density. The Gaussian distribution owes its symmetry to the fact, that it has the same range of support in both directions, looking from the mean of the distribution. That is different for zero-bound distributions, which run to plus infinity to the right, but hit a hard limit to the left. These distributions are asymmetric, which typically means that the left side runs steeper than the right side. Such distributions are called to be left-skewed. An example is the Poisson distribution, which will be covered in more detail below. Double-bound distributions, such as Binomial distributions, can also be right skewed, when the mean of the distribution is approaching the right boundary. When moving the mean from left to right, the distribution starts left-skewed, takes a symmetric form in the center of the range and then continues with increasing right skew. The following illustration shows that for the Beta distribution, which has range of support between (not including) Zero and One. At the same time, this illustration conveys another important relationship between features of shape: a distribution that approaches a boundary also is less dispersed.

```
ggplot(tibble(location = c(.25, .5, .75)), aes(x = location)) +
  stat_function(fun = dbeta, args = list(shape1 = 2, shape2 = 8),
                mapping = aes(colour = "Beta(mean = .25)")) +
    stat_function(fun = dbeta, args = list(shape1 = 5, shape2 = 5),
                mapping = aes(colour = "Beta(mean = .5)")) +
    stat_function(fun = dbeta, args = list(shape1 = 8, shape2 = 2),
                mapping = aes(colour = "Beta(mean = .75)")) +
  xlim(0, 1) +
  labs(colour="Location shift of Gaussian distributions", x = "outcome", y = "density")
```

A frequent problem in statistical modelling is to choose a symmetric distribution, in particular the Gaussian, when the distribution is highly skewed. This will be treated in more depth in chapter 6.

Many dozens distribution functions are known in statistical science and are candidates to choose from. In first orientation, our choice can ground on obvious characteristics of measures, such as discrete/continuous or range, but that is sometimes not sufficient. For example, the pattern of randomness in three-tasks falls into a binomial distribution only, when all trials have the same chance of success. If the tasks are very similar in content and structure, learning is likely to happen and the chance of success differs between trials. Using the binomial distribution when chances are not constant leads to mistaken statistical models.

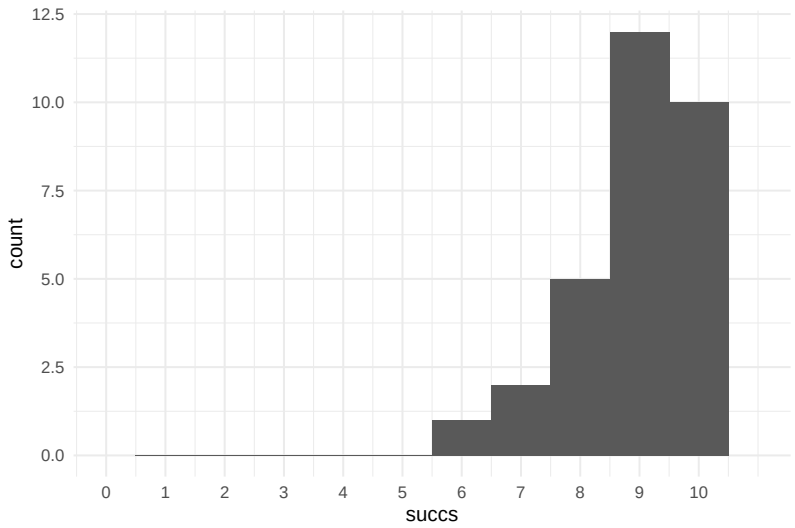
For most distributions, strict mathematical descriptions exist for under which circumstances randomness takes this particular shape. Frequently, there is one or more natural phenomena that accurately fall into this pattern, such as the number of radioactive isotope cores decaying in a certain interval (Poisson distributed) or the travel of a particle under Brownian motion (Gaussian). This is particularly the case for the canonical four random distributions that follow: Poisson, Binomial, Exponential and Gaussian. My goal here is not to bother anyone with formalism, but to introduce my readers to think about distributions as arising from data generating processes.

3.4.2.3 Binomial distributions

A very basic performance variable in design research is task success. Think of devices in high risk situations such as medical infusion pumps in surgery.

These devices are remarkably simple, giving a medication at a certain rate into the bloodstream for a given time. Yet, they are operated by humans under high pressure and must therefore be extremely error proof in handling. Imagine, the European government would set up a law that manufacturers of medical infusion pump must prove a 90% error-free operation in routine tasks. A possible validation study could be as follows: a sample of $N = 30$ experienced nurses are invited to a testing lab and asked to complete ten standard tasks with the device. The number of error-free task completions per nurse is the recorded performance variable to validate the 90% claim. Under somewhat idealized conditions, namely that all nurses have the same proficiency with the device and all tasks have the success chance of 90%, the outcome follows a *Binomial distribution* and the results could look like the following:

```
set.seed(1)

tibble(succs = rbinom(30, 10, .9)) %>%
  ggplot(aes(x = succs)) +
  geom_histogram(binwidth = 1) +
  scale_x_continuous(breaks = 0:10, limits = c(0,11))
```

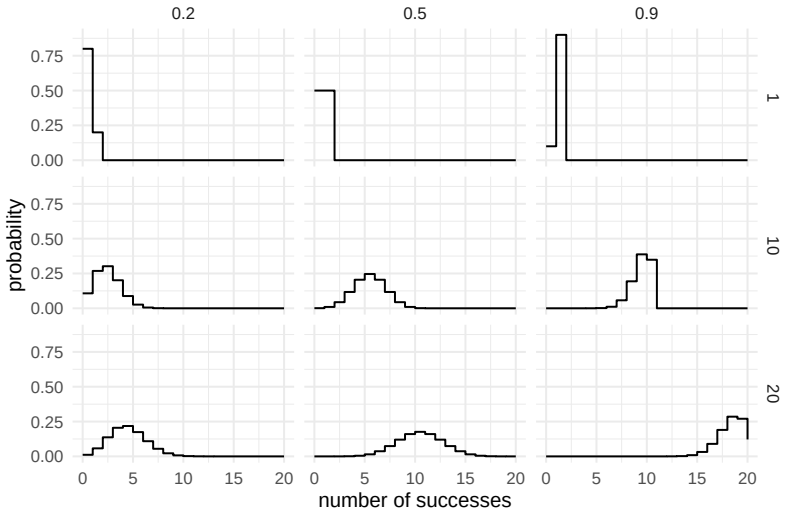


Speaking about the Binomial distribution in terms of *successes in a number of attempts* is common. As a matter of fact, *any* binary classification of outcomes is amenable for Binomial modeling, like on/off, red/blue, male/female.

An example in web design research is the rate of visitor return, resulting in a variable, that has two possible outcomes: : 0 for one-timers and 1 for returners.

This is in fact, a special case of the Binomial distribution with $k = 1$ attempts (the first visit is not counted) and is also called *Bernoulli distribution*.

```
mascutils::expand_grid(k = c(1, 10, 20),
                       p = c(0.2, 0.5, 0.9),
                       succs = 0:20) %>%
  mutate(probability = dbinom(succs, k, p)) %>%
  ggplot(aes(x = succs, y = probability)) +
  geom_step() +
  facet_grid(k ~ p) +
  labs(x = "number of successes")
```

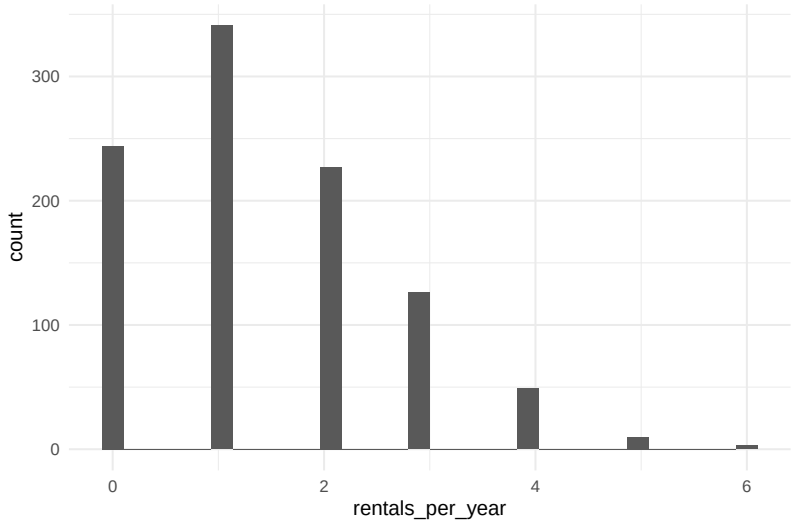


Binomial distributions have two parameters: p is the chance of success and k is the number of attempts. p is a probability and therefore can take values in the range from zero to one. Probability of success determines the location of a distribution in that With larger p the distribution moves to the right. The mean of Binomial distributions is the probability scaled by number of attempts, Mean $= kp$. Logically, there cannot be more successes then k, but with larger k the distribution gets wider, where the variance is the odds scaled by number of attempts, Var $= kp(1-p)$. As mean and variance depend on the exact same parameters, they cannot be set independently. In fact, the relation is parabolic, so that variance is largest at $p = .5$, but decreases towards both boundaries. A Binomial distribution with, say $k = 10$ and $p = .4$ always has mean 4 and variance 2.4. This means, in turn, that an outcome with a mean of 4 and a variance of 3 is not Binomially distributed. This occurs frequently, when the success rate is not identical across trials. A common solution is to use plugin distributions, where the parameter p itself is distributed, rather

than fixed. A common distribution for p is the *beta* distribution and the *logit-normal* distribution is an alternative.

The Binomial distribution has two boundaries, zero below and number of attempts k above. Hence, Binomial distributions also require that there is a clearly defined numner of attempts. For example, the number of times a customer returns to the a website does not have a natural interpretation of number of attempts, because attempts can happen at any moment. Literally, every single moment is an opportunity to go online and hire a car, even many times a day, if you are a holiday planner. At the same time, for a typical customer the per-day chance to hire a car is miniscule. Under these conditions, we get an infinite number of attempts, or painstakingly large, at least, say $k = 365$ days. At the same time the rate is very small, e.g. $p = 1.5/365$.

```
set.seed(42)
tibble(rentals_per_year = rbinom(1000, size = 365, prob = 1.5/365)) %>%
  ggplot(aes(x = rentals_per_year)) +
  geom_histogram()
```



This is a workable solution to display the number of events per period, but it remains a little odd, because the question really is about events per timespan, not success per attempt. When events per timespan is the question, the process is better covered by *Poisson distributions*.

the variable is more neatly summarized by *Poisson distributions*.

3.4.2.4 Poisson distributions

Poisson distributions can mathematically be derived from Binomial distributions with very small probability of success with infinite attempts. This implies that Poisson distributions have no upper limit. In design research, a number of measures have no clearly defined upper limit:

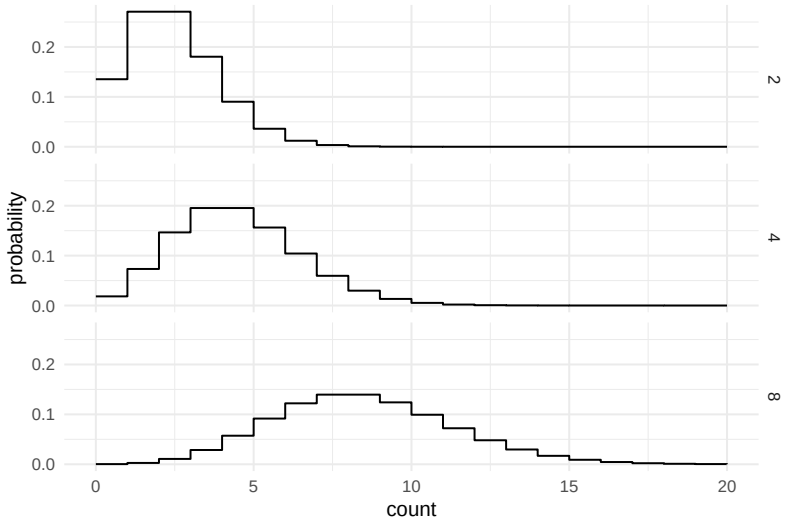
- number of erroneous actions
- frequency of returns
- behavioral events, e.g. showing exploratory behavior
- physiological events, such as number of peaks in galvanic skin response

These measures can often be modeled as *Poisson distributed*. A useful way to think of unbound counts, is that they can happen at every moment, but with a very small chance. Think of a longer interaction sequence of a user with a system, where errors are recorded. It can be conceived as an almost infinite number of opportunities to err, with a very small chance of something to happen.

Poisson distributions possess only one parameter λ (lambda), that is strictly positive and determines mean and variance of the distribution alike: $\lambda = \text{Mean} = \text{Var}$. As a result, distributions farther from Zero are alwwayas more dispersed. Because Poisson distributions have a lower boundary, they are *asymmetric*, with the left tail always being steeper. Higher λs push the distribution away from the boundary and the skew diminishes. Poisson distributions with a large mean can therefore be approximated by Gaussian distributions:

$$Y \sim \text{Pois}(\lambda) \Rightarrow Y \dot{\sim} \text{Gaus}(\mu = \lambda, \sigma = \sqrt{\lambda})$$

```
mascutils::expand_grid(lambda = c(2, 4, 8),
                       count = 0:20) %>%
  mutate(probability = dpois(count, lambda)) %>%
  ggplot(aes(x = count, y = probability)) +
  geom_step() +
  facet_grid(lambda~.)
```

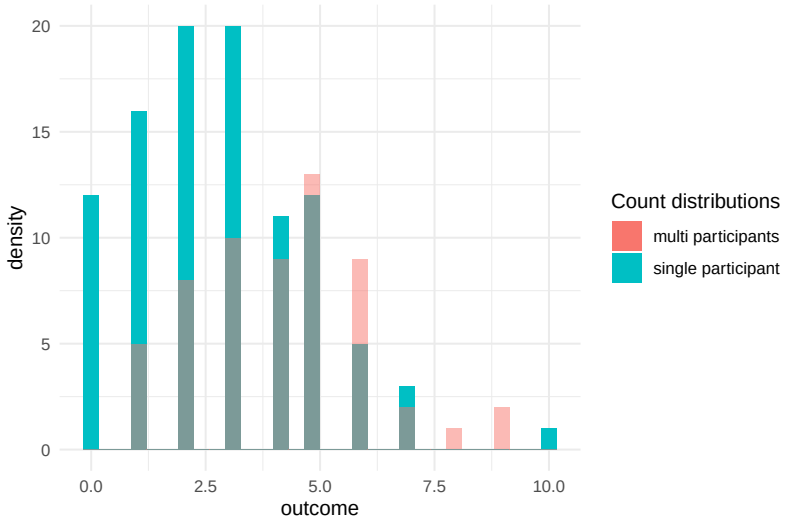


For an illustration, consider a video game, *subway smurfer*, where the player jumps and runs a little blue avatar on the roof of a train and catches items passing by. Many items have been placed into the game, but catching a single one is very difficult. The developers are aware that a too low success rate would demotivate players as much as when the game is made to easy. In this experiment, only one player is recorded, and in wonderful ways this player never suffers from fatigue, or is getting better with training. The player plays a 100 times and records the catches after every run. In this idealized situation, the distribution of catches would, approximately, follow a Poisson distribution.

Because mean and variance are tightly linked, only a certain amount of randomness can be contained at a location. If there is more randomness, and that is almost certainly so in real data, Poisson distributions are not appropriate. One speaks of *over-dispersion* in such a case. For understanding overdispersion, consider a variation of the experiment with 100 players doing one game and less restrictive rules. Players come differently equipped to perform visual search tasks and coordinate actions at high speeds. They are tested at different times of the day and by chance feel a bit groggy or energized. The chance of catching varies between players, which violates the assumption that was borrowed from the Binomial, a constant chance of success. The extra variation is seen in the wider of the two distributions.

```
## FIXME



tibble(lambda = 20,
```

```
          ci = rpois(100, log(lambda)),
          ci_ovdsp = rpois(100, log(lambda + rnorm(100, 0, 100)))
          ) %>%
  ggplot(aes(x = ci)) +
  geom_histogram(aes(fill = "single participant")) +
  geom_histogram(aes(x = ci_ovdsp, fill = "multi participants"), alpha = .5) +
  labs(fill="Count distributions", x = "outcome", y = "density")
```



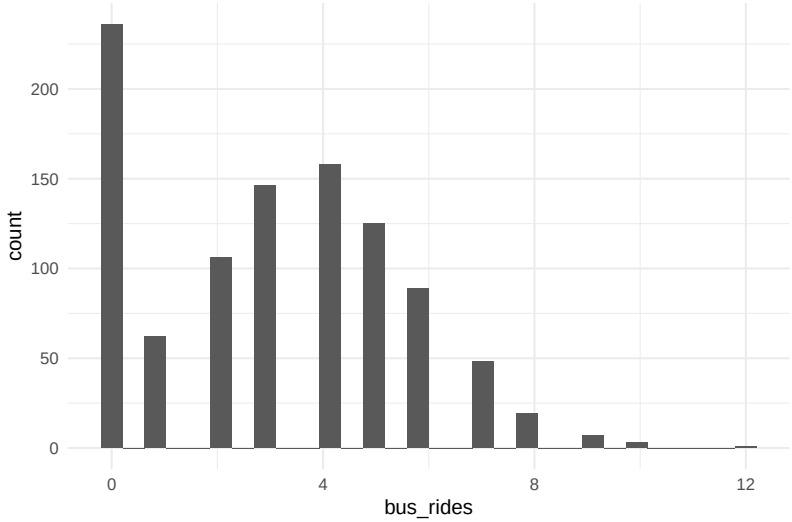
Another area for trouble can be Poisson distributions' lower boundary of exactly Zero. Often this works well, a person can perform a sequence with no errors, catch zero items or have no friends on social media. But, you cannot complete an interaction sequence in zero steps, have a conversation with less than two statements or be a customer with zero customer contacts. If the lower bound is obvious, such as the minimum necessary steps to complete a task. In such a case, a count that starts at zero can be created easily:

$$\text{additional steps} = \text{steps} - \text{neccessary steps}$$

Another frequent problem at the lower bound bound is zero-inflation. In traffic research, the frequency of going by bus can be an interesting variable to measure acceptance of public transport. If we would make such a survey, e.g. "How many times have you taken the bus the last seven days?", it is not unlikely, that we get a distribution like the following:

```
tibble(uses_public_transport = rbernoulli(1000, .8),
       bus_rides = rpois(1000, uses_public_transport * 4)) %>%
```

```
ggplot(aes(x = bus_rides)) +
geom_histogram()
```



In this scenario, the population is not homogenous, but falls into two classes, those who use public transport and those who just never do. The result is a *zero-inflated* distribution. A way to deal with this are hurdle models, which are called as such, because you first have to jump over the hurdle to use public transport, before you are in the game.

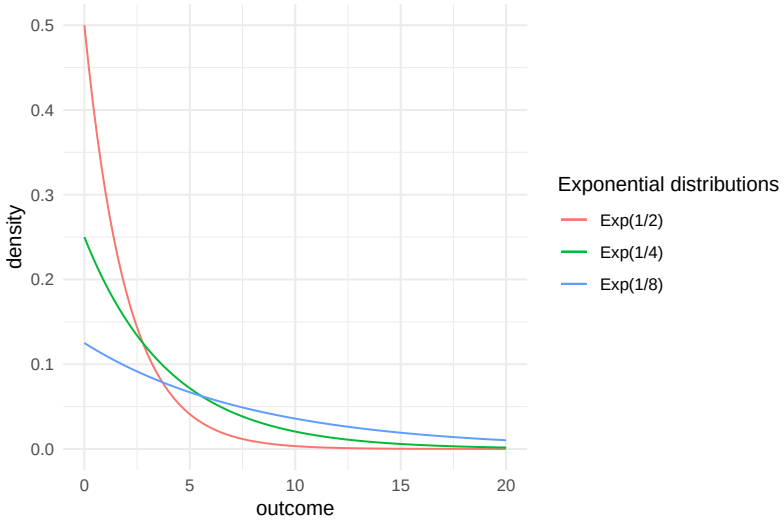
3.4.2.5 Exponential distribution

Exponential distributions apply for measures of duration. Exponential distributions have the same generating process as Poisson distributions, except, that the *duration between events to happen* is the variable of interest, rather than number of events in a given time. Under the idealized conditions of a Subway Smurfer player with constant ability to catch items, the duration between any two catches is exponentially distributed.

Just like Poisson distributions, Exponential distributions have one parameter, called rate and frequently written as λ. The following figure illustrates Exponential distributions with varying rates.

```
ggplot(tibble(x = c(0, 20)), aes(x = x)) +
  stat_function(fun = dexp, args = list(rate = 1/2),
                mapping = aes(colour = "Exp(1/2)")) +
  stat_function(fun = dexp, args = list(rate = 1/4),
```

```
                mapping = aes(colour = "Exp(1/4)")) +
  stat_function(fun = dexp, args = list(rate = 1/8),
                mapping = aes(colour = "Exp(1/8)")) +
  labs(colour="Exponential distributions", x = "outcome", y = "density")
```



Durations are common measures in design research, most importantly, time-on-task and reaction time. Unfortunately, the exponential distribution is a poor approximation of the random pattern found in duration measures. That is for three reasons: first, Exponential distributions have just one parameter, which means that mean and variance are tightly linked as Mean $= 1/\lambda$ and Var $= 1/\lambda^2$. Like with Poisson distributions, this it does not allow extra variance, e.g. between participants. Second, the distribution always starts at zero, whereas human reactions always require some basic processing, and be this just the velocity of signals travelling nerve cells, which is far below speed of sound (in air). And third, the exponential process has one special property, which is called *memoryless-ness*. That basically means, that the probability for an event to happen is completely independent of how long one has been waiting for it. In basic physical systems, like nucleus decay, this is the truly the case. But, if a person works on a task, the chance of task completion usually increases when time progresses.

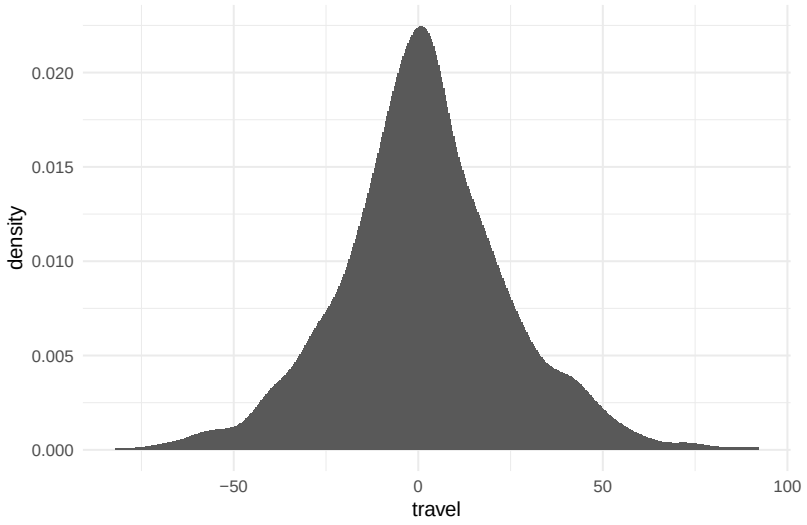
The one thing that exponential distribtions are getting right about durations (in contrast to the frequently employed Gaussian distribution), is the asymmetry. Durations have a lower boundary and that will always produce some skew. For the stated reasons, pure exponential distributions themselves are rarely used in statistical modelling, but two- and three parameter distributions, like Gamma or Exponential-Gaussian are extensions of Exponential distributions.

Gamma distributions provide a scale parameter for incorporating extra variance, whereas Exponential-Gauss distributions can be used, when the lower boundary is not Zero, but positive 6.3.

3.4.2.6 Gaussian distributions

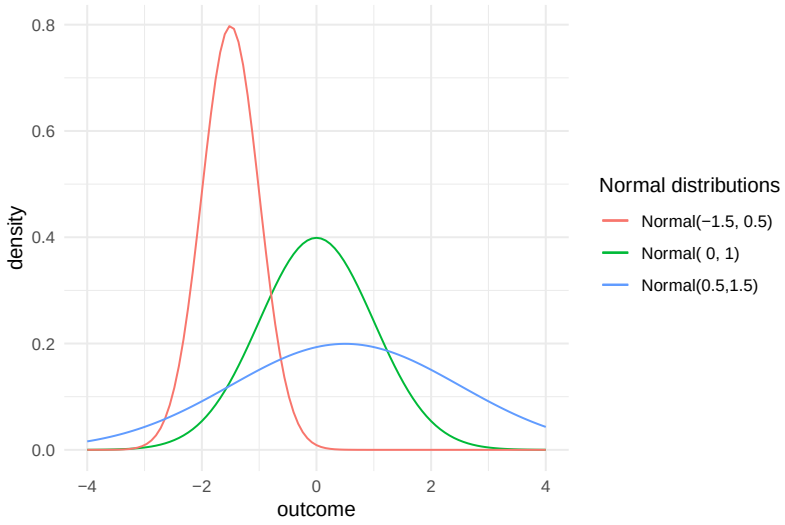
The best known distributions are *Gaussian distributions* or *Normal distributions*. These distributions arise mathematically under the assumption of a myriad of small unmeasured forces (SMURF) pushing performance (or any other outcome) up or down. As SMURFs work in all directions independently, their effects often average out and the majority of observations stays clumped together in the center, more or less. The physical process associated with Gaussian distribution is Brownian motion, the movement of a small particles caused by bumping into molecules of a warm fluid. Imagine, you are standing blind-folded in the middle of a tunnel, which is frequented by blind-folded passengers moving uncoordinated in both directions of the tunnel. Passengers will randomly bump into you from both directions, and this pushes you sometimes forward, sometimes backwards. The total travel caused by many of these small pushes is approximately Gaussian distributed. By tendency, the small pushes cancel each other out, such that the most likely position after, say 100 bumps will be close to where you started (Zero). When this experiment is repeated very often, a Gaussian distribution arises:

```
set.seed(2)
rtravel <- function(n, bumps = 100)
  map_int(1:n, function(x) as.integer(sum(sample(c(-1, 1), x, replace = T))))

D_Tunnel <-
  tibble(Part = 1:1000,
         travel = rtravel(n = 1000))

D_Tunnel %>%
  ggplot(aes(x  = travel)) +
  geom_histogram(stat = "density")
```



Gaussian distributions take two parameters: μ marks the location of the mean of the distribution. Because distributions are symmetric, the mean coincides with median and mode of the distribution. The second parameter σ represents the dispersion of the random pattern. When randomness is pronounced, the center of the distribution gets less mass assigned, as the tails get thicker. Different to Poisson and Binomial distributions, mean and variance of the distribution can be set independently and therefore over-dispersion is never an issue.

```
ggplot(tibble(x = c(-4, 4)), aes(x = x)) +
  stat_function(fun = dnorm, args = list(mean = 0, sd = 1),
                mapping = aes(colour = "Normal( 0, 1)")) +
  stat_function(fun = dnorm, args = list(mean = -1.5, sd = 0.5), mapping = aes(colour = "No
  stat_function(fun = dnorm, args = list(mean = 0.5, sd = 2), mapping = aes(colour = "Norma
  labs(colour="Normal distributions", x = "outcome", y = "density")
```



As illustrated by the Tunnel example, Gaussian distributions have the most compelling interpretation of summarizing the effect of SMURFs. Therefore, they are useful to capture randomness in a broad class of regression models and other statistical approaches.

The Gaussian distribution is called "normal", because people normally use it. For many decades it was the standard distribution to tackle almost every statistical problem. That is so, because statistical models with Gaussian shape of randomness can be computed very efficiently. Indeed, Gaussian distributions are often good approximations, which goes back to the *central limit theorem*. Basically, this theorem proves what we have passingly observed at binomial and Poisson distributions: the more they move to the right, the more symmetric they get. The central limit theorem proves that, in the long run, a wide range of distributions are indistinguishable from the Gaussian distribution. But, the key word is *in the long run*. As a general rule, Gaussian distributions approximates other distributions well, when the majority of measures stay far from the natural boundaries. That is the case in experiments with very many attempts and moderate chances (e.g. signal detection experiments), when counts are in the high numbers (number of clicks in a complex task) or with long durations and little variance in duration. However, in the remainder of this book, especially in chapter 6, we will encounter numerous real data sets where the Gaussian distribution is not a good approximation. Gaussian models may be convenient to set up and interpret, but strictly speaking, a Gaussian model is always wrong. And that is for one particular reason: Gaussian distributions are unbound, but there is not a single measure in any scientific discipline that really has no limits.

these rules are no guarantee and careful model criticism is essential. We will return to this issue in 7.1

Nowadays, computing power has evolved to a point, where efficient approximations are less needed. Especially with the advent of Bayesian modelling, statistical users have a plethory of distributions to choose from, and we will see more of them when Generalized Linear Models are introduced 6. Most of the time, choosing an appropriate shape for randomness is straight-forward, and doing so makes a model less prone to criticism 7.1.

3.5 Towards Bayesian estimation

Frequentist statistics falls short on recognizing that research is incremental. Bayesian statistics embraces the idea of gradual increase in certainty when new data arrives. Why has Bayesian statistics not been broadly adopted earlier? The reason is that Bayesian estimation was computationally unfeasible for many decades. In practice, the innocent multiplication of prior probability and likelihood of Bayes' becomes a multiplication of two density distributions, which results in complex integrals, which in most cases have no analytic solution. If you have enjoyed a classic statistics education, you may recall how the computation of an ANOVA could be carried out on paper and pencil in reasonable time (e.g. during an exam). And that is precisely how statistical computations have been performed before the advent of electronic computing machinery. In the frequentist statistical framework, ingenious mathematicians have developed procedures that were rather efficient to compute. That made statistical data analysis possible in those times.

Expensive computation is in the past. Modern computers can simulate realistic worlds in real time and the complex integrals in Bayesian statistics they solve hands down. When analytic solutions do not exist, the integrals can still be solved using numerical procedures. Numerical procedures have been used in frequentist statistics, too, for example the iterative least squares algorithm applies for Generalized Linear Models, Newton-Rapson optimizer can be used to find the maximum likelihood estimate and boot-strapping produces accurate confidence limits. However, these procedures are too limited as they fail for highly multidimensional problems as they are common in advanced regression models.

Today, more and more researchers leave frequentist statistics alone and enter the world of Bayesian Statistics. Partly, this may be due to the replicability crisis of 2015, which has shown that, in general, behavioural researchers are not capabable of carrying out frequentist statistics, correctly. In result, the incorrect use of null hypothesis testing and p-values has fully under-mined the discipline of Psychology and that may just be the tip of the iceberg.

I assume that the majority of researchers, who started using Bayesian Statistics, are just thrilled by the versatility of Bayesian estimation, and its natural interpretation. To name just one example for the latter, Frequentist statisticians emphasize the randomness of sampling. It all goes by asking: how would an estimate change, when we draw another sample? This is best seen by the definition of the frequentist *confidence interval*: "the confidence level represents the frequency (i.e. the proportion) of possible confidence intervals that contain the true value of the unknown population parameter." [%Wikipedia] Interestingly, when asked about the definition, most researchers answer: "the range, in which the true value falls with a certainty of 95%." Ironically, this is precisely the definition of the Bayesian *credibility interval*. It seems that thinking of certainty by itself, not through frequencies, is more natural than a long series of imagined replications of the same experiment.

The other reason for Bayesian Statistics gaining ground is that today's implementations offer endless possibilities. In frequentist statistics, you would find an implementation for, say, linear models with a Beta shape of randomness 6.4.2. But, if your research design is multi-level, say for psychometric evaluation, you would be less lucky. You would find plenty implementations for psychometrics, but none supports Beta distribution. With Bayesian implementations, that is the past. In chapter 5.8 I will show how multi-level models can replace classic tools for psychometric data, and in section 6.4.2 we use Beta distribution on that model and I will explain why this is the better choice. The greatest advantage is that all these possibilities are governed by just one implementation, the Brms engine, which even allows us to add custom distributions, like Beta-binomial distribution 6.2.3.2. The most astonishing fact is accomplished by a relatively simple algorithm, called Markov-Chain Monte Carlo sampling.

Most Bayesian estimation engines these days ground on a numerical procedure called *Markov-Chain Monte-Carlo (MCMC)* sampling, which we will closer examine in chapter 4.1.1. This method differs from the earlier mentioned in that it grounds on random walk. The closest frequentist counterpart to MCMC is the bootstrapping algorithm, which draws many samples from data (how could it be different) and computes the estimates many times. Bayesian estimation with MCMC turns this upside down, by randomly drawing possible parameter values and computing the posterior probability many times. Similar to boot strapping, the basic MCMC algorithm is so simple, it can be explained on half a page and implemented with 25 lines of code. Despite its simplicity, the MCMC algorithm is applicable to practically all statistical problems one can imagine. Being so simple and generic at the same time must come at some costs. The downside of MCMC sampling still is computing time. Models with little data and few variables, like the examples in 3.1, are estimated within a few seconds. Multi-level models, which we will encounter later in this book, can take hours and large psychometric models can take up to a few days of processing time.

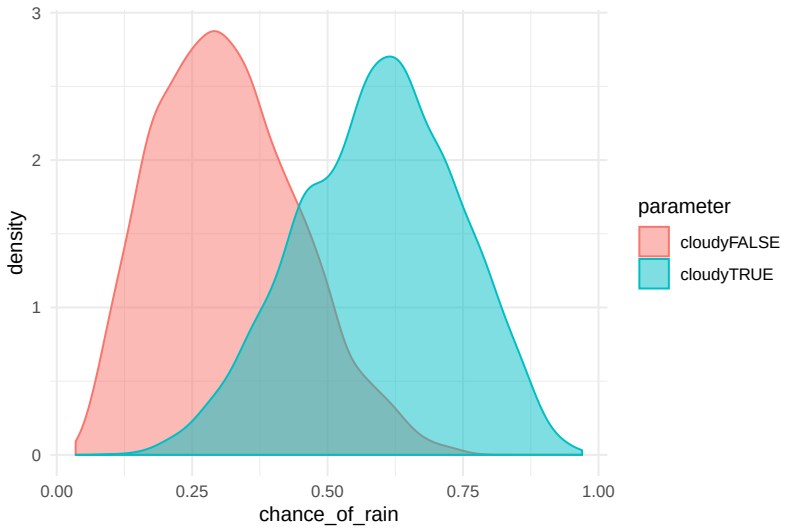
The particular merit of the MCMC algorithm is that it not only delivers accurate point estimates in almost any situation, it produces the full *posterior probability distribution*. This lets us characterize a parameters magnitude and degree of (un-)certainty. Let's run an analysis on the 20 rainfall observations to see how this happens.

```
attach(Rainfall)
```

```
M_1 <-
  Rain %>%
  stan_glm(rain ~ cloudy -1,
           family = binomial,
           data = .)
```

What the estimation does, is to calculate the *posterior distribution* from the observations. The *posterior distribution* contains the probability (more precisely: the *density*) for all possible values of the parameter in question. The following density plot represents our belief about the parameter $P(rain|cloudy)$ after we have observed twenty days:

```
posterior(M_1) %>%
  filter(type == "fixef") %>%
  mutate(chance_of_rain = plogis(value)) %>%
  ggplot(aes(x = chance_of_rain, fill = parameter, col = parameter)) +
  geom_density(alpha = .5)
```



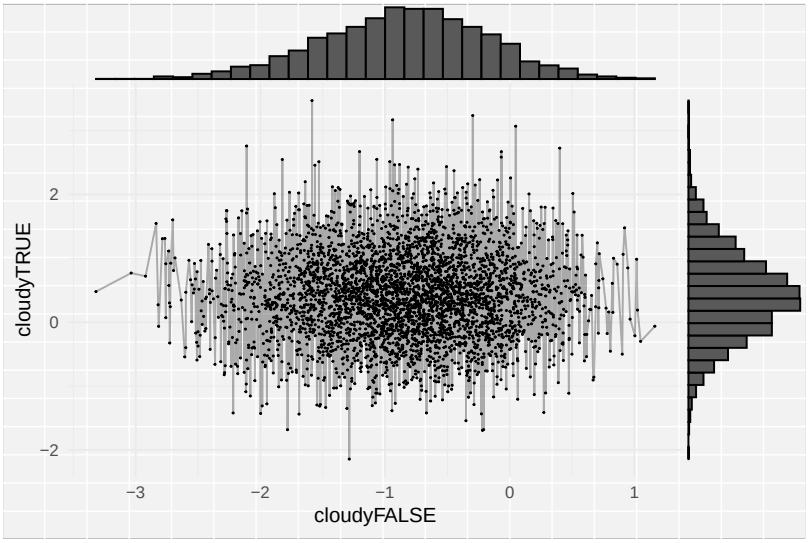
From the posterior distribution, we can deduct all kinds of summary statistics, such as:

1. the most likely value for a parameter in question is called the *posterior mode* and, when prior knowledge is absent, is the same as the frequentist *maximum likelihood estimate*.
2. the average of parameter values, weighted by their probability is called the *posterior mean*
3. a defined range to express 95% (or any other level of) certainty is the *95% credibility interval*

We can also pull non-standard summary statistics from the posterior distribution. For example, in 4.6.1 we will ask how valid a third degree polynomial is for rendering the Uncanny Valley effect.

Coming back to MCMC random walk: how is the posterior distribution actually produced. The MCMC makes a particular type of random walk through parameter space, where more likely regions are visited more often. Basically, the posterior distribution is approximated by a frequency distribution of the visited values. See the random walk in action in the following figure! Note how the gray connecting lines show the jumps in the MCMC random walk and how the marginal frequency distributions emerge.

```
G_mcmc <-
  posterior(M_1) %>%
  select(iter, chain, fixef, value) %>%
  spread(fixef, value) %>%
  ggplot(aes(x = cloudyFALSE, y = cloudyTRUE)) +
  geom_point(size = .1) +
  geom_line(alpha = .3)

plot(ggExtra::ggMarginal(G_mcmc, type = "histogram"))
```




```
detach(Rainfall)
```

3.5.0.0.1 WRITE END

Part II

Models

4

Linear models

Linear models answer the question of how one quantitative outcome, say ToT, decreases or increases, when a condition changes.

First, I will introduce the most basic LM. The grand mean model (GMM does not have a single predictors and produces just a single estimate: the grand mean in the population. That can be useful, when there exists an exernal standard to which the design must adhere to. In R, the GMM has a formaula, like this: `ToT ~ 1`. At the example of the GMM, I describe some basic concepts of Bayesian linear models. On the practical side of things, CLU tables are introduced, which is one major work horse to report our results.

The most obvious application of LM comes next: in linear regression models (LRM), a metric predictor (e.g. age) is linked to the outcome by a linear function, such as: $f(x) = \beta_0 + \beta_1 x$. In R, this is: `ToT ~ 1 + age`. Underneath is a very practical section that explains how simple transformations can make the results of an estimation more clear. Correlations work in similar situations as LRMs, and it is good to know what the differences and how correlations are linked to linear slope parameters. I end this section with a warning: the assumption of linearity is limited to a straight line and this is violated not by some, but all possible data.

A very common type of research question is how an outcome changes under different conditions. In design research this is always the case, when designs are compared. In R, a *factorial model* looks just like an LRM: `ToT ~ 1 + Design`. The rest of the section is dedicated to the techniques that make it possible to put a qualitative variable into a linear equation. Understanding these techniques opens the door to making your own variants that exactly fit your purpose, such as when a factor is ordered and you think of it of as a stairway, rather than treatments. Finally, we will see how factorial models can resolve problems of linearity, like they appear with learning curves.

Once linear regression and factorial models are united, nothing can stop us to combine them into *multi-preditor models* (MPM). I will introduce all possible

combinations: multi-linear, multi-factorial and mixed. However, this section is more intended as a smooth transition models with conditional effects, as these are often more realistic (and interesting).

Conditional models go one step further in that it allows predictors to change there efefct, depending on the other predictors. First I will introduce the practical issues of interpreting conditional effects models. Then we come to some explanations: We will see that conditional effects can adjust for saturation effects, which can severely compromise the linearity assumption. Then we will move on to conditional models which are theoretically more interesting.

And if it weren't enough about *non-linearity* in a chapter on linear models. Turns out you can estimate wobbly lines, too, by using polynomials.

The linear model gives you endless possibilities to recombine any number of predictors. But, how good are these models, actually? The last part of this chapter introduces basic techniques of model criticism and model comparison. In model criticism, a single estimated model is under scrunity. The structural part of the model we examine by looking at model fit, where fitted responses are compared to measures. With residual analysis, the pattern of randomness is checked. Model selection deals with comparing models. We will learn about a surprsingly simple technique to compare a set of models by their forecasting accuracy. And finally, I will introduce *another* p-value. It is not what you think, but it is something you want.

4.1 Quantification at work: grand mean models

Reconsider Jane from 3.1. She was faced with the problem that potential competitors could challenge the claim "rent a car in 99 seconds" and in consequence drag them to court. More precisely, the question was: "will users on average be able ...", which is nothing but the *population mean*. A statistical model estimating just that, we call a *grand mean model* (GMM). The GMM is the most simple of all models, so in a way, we can also think of it as the "grandmother of all models". Although its is the simplest of all, it is of useful application in design research. For many high risk situations, there often exist minimum standards for performance to which one can compare the population mean, here are a few examples:

- with medical infusion pump the frequency of decimal input error (giving the tenfold or the tenth of the prescribed dose) must be below a bearable level
- the checkout process of an e-commerce website must have a a cancel rate not higher than ...
- the timing of a traffic light must be designed to give drivers enough time to hit the brakes.

A GMM predicts the *average* expected level of performance in the population (β_0). Let's atsrt with a toy example: When you want to predict the IQ score of a totally random and anonymous individual (from this population), the population average (which is standardized to be 100) is your best guess. However, this best guess is imperfect due to the individual differences. and chances are rather low that 100 is the perfect guess.

```
# random IQ sample, rounded to whole numbers
set.seed(42)
N <- 1000
D_IQ <-   tibble(score = rnorm(N, mean = 100, sd = 15),
                 IQ = round(score, 0))

# proportion of correct guesses
pi_100 <- sum(D_IQ$IQ == 100)/N
str_c("Proportion of correct guesses (IQ = 100): ", pi_100)
```

```
## [1] "Proportion of correct guesses (IQ = 100): 0.031"
```

This best guess is imperfect, for a variety reasons:

1. People differ a lot in intelligence.
2. The IQ measure itself is uncertain. A person could have had a bad day, when doing the test, whereas another person just had more experience with being tested.
3. If test items are sampled from a larger set, tests may still differ a tiny bit.
4. The person is like the Slum Dog Millionaire, who by pure coincidence encountered precisely those questions, he could answer.

In a later chapters we will investigate on the sources of randomness 5. But, like all other models in this chapter, the GMM is a *single level linear model*. This single level is the *population-level* and all unexplained effects that make variation are collected in ϵ_i, the *residuals* or *errors*, which are assumed to follow a Gaussian distribution with center zero and *standard error* σ_ϵ.

Formally, a GMM is written as follows, where μ_i is the *predicted value* of person i and β_0 is the population mean β_0, which is referred to as *Intercept* (see 4.3.1).

$$
\begin{aligned}
\mu_i &= \beta_0 \\
y_i &= \mu_i, + \epsilon_i \\
\epsilon_i &\sim \text{Gaus}(0, \sigma_\epsilon)
\end{aligned}
$$

This way of writing a linear model only works for Gaussian linear models, as only here, the residuals are symmetric and are adding up to Zero. In chapter

6, we will introduce linear models with different error distributions. For that reason, I will use a slightly different notation throughout:

$$\mu_i = \beta_0$$
$$y_i \sim \text{Gaus}(\mu_i, \sigma_\epsilon)$$

The notable difference between the two notations is that in the first we have just one error distribution. In the second model, every observation actually is taken from its own distribution, located at μ_i, albeit with a *constant variance*.

Enough about mathematic formulas for now. In R regression models are specified by a dedicated formula language, which I will develop step-by-step in chapter. This formula language is not very complex, at the same time provides a surprisingly high flexibility for specification of models. The only really odd feature of this formula language is that it represents the intercept β_0 with `1`. To add to the confusion, the intercept means something different, depending on what type of model is estimated. In GMMs, it is the grand mean, whereas in group-mean comparisons, it is the mean of one reference group 4.3.1 and in linear regression, it is has the usual meaning as in linear equations.

Below we estimate a GMM on (simulated) IQ scores using the `stan_glm` regression engine. The `clu()` command extracts all the parameters of a models and reports them with 95% certainty limits:

```
M_IQ <- stan_glm(IQ ~ 1,
                 data = D_IQ)
```

```
bayr::clu(M_IQ)
```

parameter	type	fixef	center	lower	upper
Intercept	fixef	Intercept	99.6	98.7	100.5
sigma_resid	disp	NA	15.1	14.4	15.8

So, when estimating the grand mean model, we estimate the intercept β_0 and the standard error σ. In R, the analysis of the 99 seconds problem 3.1 unfolds as follows: completion times (ToT) are stored in a data frame, with one observation per row. This data frame is send to the R command `stan_glm` for estimation, using `data = D_1`. The formula of the grand mean model is `ToT ~ 1`. Left of the `~` (*tilde*) operator is the outcome variable. In design research, this often is a performance measure, such as time-on-task, number-of-errors or self-reported cognitive workload. The right hand side specifies the *deterministic part*, containing all variables that are used to predict performance.

```
attach(Sec99)
```

```
M_1 <- stan_glm(ToT ~ 1, data = D_1)
```

```
summary(M_1)
```

```
##
## Model Info:
##  function:     stan_glm
##  family:       gaussian [identity]
##  formula:      ToT ~ 1
##  algorithm:    sampling
##  sample:       4000 (posterior sample size)
##  priors:       see help('prior_summary')
##  observations: 100
##  predictors:   1
##
## Estimates:
##               mean   sd    10%   50%   90%
## (Intercept) 105.9    3.0 102.0 106.0 109.8
## sigma        31.4    2.1  28.8  31.3  34.2
##
## Fit Diagnostics:
##            mean   sd    10%   50%   90%
## mean_PPD 105.9    4.3 100.3 105.9 111.2
##
## The mean_ppd is the sample average posterior predictive distribution of the outcome va
##
## MCMC diagnostics
##               mcse Rhat n_eff
## (Intercept)   0.1  1.0  2575
## sigma         0.0  1.0  3070
## mean_PPD      0.1  1.0  3445
## log-posterior 0.0  1.0  1840
##
## For each parameter, mcse is Monte Carlo standard error, n_eff is a crude measure of ef
```

Most of the time a researcher does not want to deal with the posterior directly, but desires a brief summary of where the effects lie and what the level of certainty is. *Tables of estimates*, like the one shown below, serve exactly this purpose. Estimates tables report the central tendency of every estimate, which is the best guess for the true magnitude of an effect. Next to that, the spread of the posterior distribution is summarized as 95% credibility intervals and represent the degree of uncertainty: the less certain an estimate is, the wider

is the interval. A 95% credibility interval gives a range of possible values where you can be 95% certain that it contains the true value. A complete center-lower-upper table of estimates is produced by the `clu` command (bayr package):

```
clu(M_1)
```

parameter	type	fixef	center	lower	upper
Intercept	fixef	Intercept	106.0	100.1	112
sigma_resid	disp	NA	31.3	27.6	36

The `clu` command being used in this book is from the accompanying R package `bayr` and produces tables of estimates, showing *all parameters* in a model, that covers the effects, i.e. coefficients the dispersion or shape of the error distribution, here the standard error. Often, the distribution parameters are of lesser interest and `clu` comes with sibling commands to only show the (population-level) coefficients:

```
coef(M_1)
```

parameter	type	fixef	center	lower	upper
Intercept	fixef	Intercept	106	100	112

Note that regression engines, such as rstanarm, bring their own commands to extract estimates, especially `fixef`, but these often report the center estimates, only.

```
rstanarm:::coef.stanreg(M_1)
```

```
## (Intercept)
##         106
```

In order to always use the convenient commands from package bayr, it is necessary to load Bayr after package Rstanarm.

```
library(rstanarm)
library(bayr)
```

Then, Bayr overwrites the fixef (coef and ranef) commands to produce coefficient tables.

```
coef(M_1)
```

parameter	type	fixef	center	lower	upper
Intercept	fixef	Intercept	106	100	112

```
detach(Sec99)
```

A GMM is the simplest linear model and as such makes absolute minimal use of knowledge when doing its predictions. The only thing one knows is that test persons come from one and the same population (humans, users, psychology students). Accordingly, individual predictions are very inaccurate. From the GMM we will depart in two directions. First, in the remainder of this chapter, we will add predictors to the model, for example age of participants or a experimental conditions. These models will improve our predictive accuracy by using additional knowledge about participants and conditions of testing.

Reporting a model estimate together with its level of certainty is what makes a statistic *inferential* (rather than merely descriptive). In Bayesian statistics, the posterior distribution is estimated (usually by means of MCMC sampling) and this distribution carries the full information on certainty. If the posterior is widely spread, an estimate is rather uncertain. You may still bet on values close to the center estimate, but you should keep your bid low. Some authors (or regression engines) express the level of certainty by means of the standard error. However, the standard deviation is a single value and has the disadvantage that a single value does not represent non-symmetric distributions well. A better way is to express certainty as limits, a lower and an upper. The most simple method resembles that of the median by using quantiles.

It is common practice to explain and interpret coefficient tables for the audience. My suggestion of how to *report regression results* is to simply walk through the table row-by-row and for every parameter make *three statements*:

1. What the parameter says
2. a quantitative statement based on the central tendency
3. an uncertainty statement based on the CIs

In the present case Sec99 that would be:

The *intercept* (or β_0) is the population average and is in the region of 106 seconds, which is pretty far from the target of 99 seconds. The certainty is pretty good. At least we can say that the chance of the true mean being 99 seconds or smaller is pretty marginal, as it is not even contained in the 95% CI.

And for σ:

The population mean is rather not representative for the observations as the standard error is almost one third of it. There is much deviation from the population mean in the measures.

From here on, we will build up a whole family of models that go beyond the population mean, but have effects. A *linear regression model* can tell us what effect *metric predictors*, like age or experience have on user performance. 4.2 *Factorial models* we can use for experimental conditions, or when comparing designs.

4.1.1 Do the random walk: Markov Chain Monte Carlo sampling

So far, we have seen how linear models are specified and how parameters are interpreted from standard coefficient tables. While it is convenient to have a standard procedure it may be useful to understand how these estimates came into being. In Bayesian estimation, an approximation of the *posterior distribution (PD)* is the result of running the engine and is the central point of departure for creating output, such as coefficient tables. PD assigns a degree of certainty for every possible combination of parameter values. In the current case, you can ask the PD, where and how certain the population mean and the residual standard error are, but you can also ask: How certain are we that the population mean is smaller than 99 seconds and σ is smaller than 10?

In a perfect world, we would know the analytic formula of the posterior and derive statements from it. In most non-trivial models, though, there is no such formula one can work with. Instead, what the regression engine does is to approximate the PD by a random-walk algorithm called Markov-Chain Monte Carlo sampling (MCMC).

The `stan_glm` command returns a large object that stores, among others, the full random walk. This random walk represents the posterior distribution almost directly. The following code extracts the posterior distribution from the regression object and prints it. When calling the new object (class: tbl_post) directly, it provides a compact summary of all variables in the model, in this case the intercept and the residual standard error.

```
attach(Sec99)
```

```
P_1 <-  posterior(M_1)
P_1
```

** tbl_post: 4000 samples in 1 chains

model	parameter	type	fixef	entities
M_1	Intercept	fixef	Intercept	1

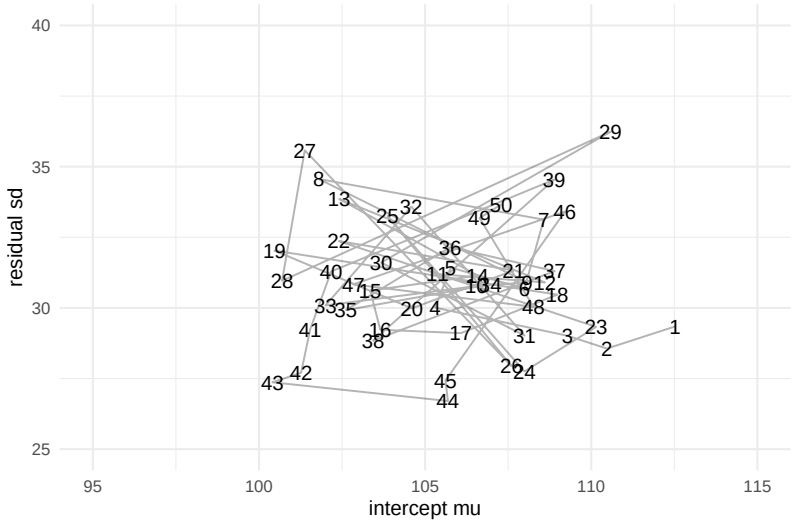
The 99 second GMM has two parameters and therefore the posterior distribution has three dimensions: the parameter dimensions β_0, σ and the probability density. Three dimensional plots are difficult to put on a surface, but for somewhat regular patterns, a density plot with contour lines does a sufficient job:

```
P_1 %>%
  select(chain, iter, parameter, value) %>%
  spread(parameter, value) %>%
  ggplot(aes(x = Intercept, y = sigma_resid, fill = ..level..)) +
  stat_density_2d(geom = "polygon") +
  xlim(95, 115) + ylim(25, 40) +
  scale_fill_continuous(name="relative frequency")
```

Let's see how this PD "landscape" actually emerged from the random walk. In the current case, the *parameter space* is two-dimensional, as we have μ and σ. The MCMC procedure starts at a deliberate point in parameter space. At every iteration, the MCMC algorithm attempts a probabilistic jump to another location in parameter space and stores the coordinates. This jump is called probabilistic for two reasons: first, the new coordinates are selected by a random number generator and second, it is either carried out, or not, and that is probabilistic, too. If the new target is in a highly likely region, it is carried out with a higher chance. This sounds circular, but it provenly works. More specifically, the MCMC sampling approach rests on a general proof, that the emerging frequency distribution converges towards the true posterior distribution. That is called *ergodicity* and it means we can take the *relative frequencies* of jumps into a certain area of parameter space as an approximation for our degree of belief that the true parameter value is within this region.

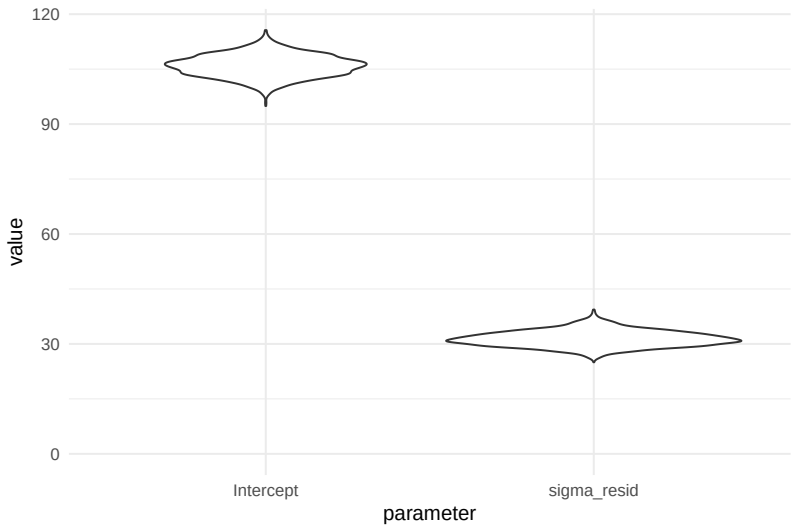
The regression object stores the MCMC results as a long series of positions in parameter space. For any range of interest, it is the relative frequency of visits that represents its certainty. The first 50 jumps of the MCMC random walk are shown in @ref(99_seconds_random_walk)‘. Apparently, the random walk is not fully random, as the point cloud is more dense in the center area. This is where the more probable parameter values lie. One can clearly see how the MCMC algorithm jumps to more likely areas more frequently. These areas become more dense and, finally, the cloud of visits will approach the contour density plot above.

```
G_random_walk <-
  P_1 %>%
  filter(iter <= 50) %>%
  select(iter, parameter, value) %>%
  spread(parameter, value) %>%
  ggplot(aes(x = Intercept, y = sigma_resid, label = iter)) +
  geom_text() +
  geom_path(alpha = .3) +
  ylab("residual sd") +
  xlab("intercept mu") +
  xlim(95, 115) + ylim(25, 40)

G_random_walk
```



The more complex regression models grow, the more dimensions the PD gets. The linear regression model in the next chapter has three parameter dimensions, which is difficult to visualize. Multi-level models 5 have hundreds of parameters, which is impossible to intellectually grasp at once. Therefore, it is common to use the *marginal posterior distributions* (MPD), which give the density of one coefficient at time. My preferred geometry for plotting many MPDs is the violin plot, which packs a bunch of densities and therefore can be used when models of many more dimensions.

```
P_1 %>%
  ggplot(aes(x = parameter, y = value)) +
  geom_violin() +
  ylim(0, NA)
```



In our example, in @ref(99_seconds_post) we can spot that the most likely value for average time-on-task is 105.97. Both distributions have a certain spread. With a wider PD, far-off values have been visited by the MCMC chain more frequently. The probability mass is more evenly distributed and there is less certainty for the parameter to fall in the central region. In the current case, a risk averse decision maker would maybe take the credibility interval as "reasonably certain".

Andrew and Jane expect some scepticism from the marketing people, and some lack in statistical skills, too. What would be the most comprehensible single number to report? As critical decisions are involved, it seems plausible to report the risk to err: how certain are they that the true value is more than 99 seconds. We inspect the histograms. The MPD of the intercept indicates that the average time-on-task is rather unlikely in the range of 99 seconds or better. But what is the precise probability to err for the 99 seconds statement? The above summary with `coef()` does not accurately answer the question. The CI gives lower and upper limits for a range of 95% certainty in total. What is needed is the certainty of $\mu \geq 99$. Specific questions deserve precise answers. And once we have understood the MCMC chain as a frequency distribution, the answer is easy: we simply count how many visited values are larger than 99. In R, the `quantile` function handles the job:

```
T_certainty <-
  P_1 %>%
  filter(parameter == "Intercept") %>%
  summarize(certainty_99s = mean(value >= 99),
            certainty_111s = mean(value >= 111))
```

```
kable(T_certainty)
```

certainty_99s	certainty_111s
0.991	0.046

It turns out that the certainty for average time-on-task above the 99 is an overwhelming 0.991. The alternative claim, that average completion time is better than 111 seconds, has a rather moderate risk to err (0.046).

```
detach(Sec99)
```

4.1.2 Likelihood and random term

In formal language, regression models are usually specified by *likelihood functions* and one or more *random terms* (exactly one in linear models). The likelihood represents the common, predictable pattern in the data. Formally, the likelihood establishes a link between *predicted values* μ_i and predictors. It is common to call predictors with the Greek letter β (beta). If there is more than one predictor, these are marked with subscripts, starting at zero. The "best guess" is called the *expected value* and is denoted with μ_i ("mju i"). If you just know that the average ToT is 106 seconds and you are asked to guess the performance of the next user arriving in the lab, the reasonable guess is just that, 106 seconds.

$$\mu_i = \beta_0$$

Of course, we would never expect this person to use 106 second, exactly. All observed and imagined observations are more or less clumped around the expected value. The *random term* specifies our assumptions on the pattern of randomness. It is given as distributions (note the plural), denoted by the $\sim$ (tilde) operator, which reads as: "is distributed". In the case of linear models, the assumed distribution is always the Normal or *Gaussian distribution*. Gaussian distributions have a characteristic bell curve and depend on two parameters: the mean μ as the central measure and the standard deviation σ giving the spread.

$$y_i \sim \text{Gaus}(\mu_i, \sigma_\epsilon)$$

The random term specifies how all unknown sources of variation take effect on the measures, and these are manifold. Randomness can arise due to all kinds of individual differences, situational conditions, and, last but not least, measurement errors. The Gaussian distribution sometimes is a good approximation for randomness and linear models are routinely used in research. In

several classic statistics books, the following formula is used to describe the GMM (and likewise more complex linear models):

$$
\begin{aligned}
y_i &= \mu_i + \epsilon_i \\
\mu_i &= \beta_0 \\
\epsilon_i &\sim \text{Gaus}(0, \sigma_\epsilon)
\end{aligned}
$$

First, it is to say, that these two formulas are mathematically equivalent. The primary difference to our formula is that the *residuals* ϵ_i, are given separately. The pattern of residuals is then specified as a single Gaussian distribution. Residual distributions are a highly useful concept in modelling, as they can be used to check a given model. Then the the classic formula is more intuitive. The reason for separating the model into likelihood and random term is that it works in more cases. When turning to Generalized Linear Models (GLM) in chapter **??**, we will use other patterns of randomness, that are no longer additive, like in $\mu_i + \epsilon_i$. As I consider the use of GLMs an element of professional statistical practice, I use the general formula throughout.

4.1.3 Working with the posterior distribution

Coefficient tables are the standard way to report regression models. They contain all effects (or a selection of interest) in rows. For every parameter, the central tendency (center, magnitude, location) is given, and a statement of uncertainty, by convention 95% credibility intervals (CI).

```
attach(Sec99)
```

The object `M_1` is the model object created by `stan_glm`. When you call `summary` you get complex listings that represent different aspects of the estimated model. These aspects and more are saved inside the object in a hierarchy of lists. The central result of the estimation is the *posterior distribution (HPD)*. With package Rstanarm, the posterior distribution is extracted as follows:

```
P_1_wide <-
  as_tibble(M_1) %>%
  rename(Intercept = `(Intercept)`)

str(P_1_wide)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    4000 obs. of  2 variables:
##  $ Intercept: num  113 110 109 105 106 ...
##  $ sigma    : num  29.3 28.6 29 30 31.4 ...
```

The resulting data frame is a matrix, where each of the 4000 rows is one coordinate the MCMC walk has visited in a two-dimensional parameter space 4.1.1. For the purpose of reporting parameter estimates, we could create a *coefficient table* like follows:

```
P_1_wide %>%
  summarize(center_Intercept = median(Intercept),
            center_sigma = median(sigma),
            lower_Intercept = quantile(Intercept, .025),
            lower_sigma = quantile(sigma, .025),
            upper_Intercept = quantile(Intercept, .975),
            upper_sigma = quantile(sigma, .975))
```

center_Intercept	center_sigma	lower_Intercept	lower_sigma	upper_Intercept	upper_sigma
106	31.3	100	27.6	112	36

As can be seen, creating coefficient tables from wide posterior objects is awful and repetitive, even when there are just two parameters (some models contain hundreds of parameters). Additional effort would be needed to get a well structured table. The package Bayr can extract posterior distributions, too, but produces a *long format*. This works approximately like can be seen in the following code, which employs `tidyr::gather` to make the wide Rstanarm posterior long.

```
P_1_long <- P_1_wide %>%
  tidyr::gather(key = parameter)


P_1_long %>%
  sample_n(10) %>%
  arrange(parameter)
```

parameter	value
Intercept	105.7
Intercept	100.1
Intercept	103.3
sigma	30.0
sigma	31.0
sigma	34.5
sigma	30.6
sigma	30.6
sigma	32.5
sigma	30.1

With long posterior objects, summarizing over the parameters is efficient and straight-forward, in other words: it is tidy.

```
P_1_long %>%
  group_by(parameter) %>%
  summarize(center = median(value),
            lower = quantile(value, .025),
            upper = quantile(value, .975))
```

parameter	center	lower	upper
Intercept	106.0	100.1	112
sigma	31.3	27.6	36

With the Bayr package, the `posterior` command produces such a long posterior object:

```
P_1 <- bayr::posterior(M_1)
P_1
```

** tbl_post: 4000 samples in 1 chains

model	parameter	type	fixef	entities
M_1	Intercept	fixef	Intercept	1

When called, the posterior object identifies itself by telling the number of MCMC samples, and the estimates contained in the model, grouped by *type of parameter*. In the case here, there is just one coefficient, the intercept and one dispersion parameter, the standard deviation of residuals The following gives a glance on the real structure of the long posterior object. Most essential are the identification of the iteration (and chain), the parameter name and the value. However, there is a lot more information stored along side, much of which we will only use in later chapters.

```
P_1 %>%
  as_tibble() %>%
  filter(!as.logical(iter %% 500)) ## <-- modulo division selects every 500th iteration
```

model	chain	iter	order	parameter	type	nonlin	fixef	re_factor	re_entity	value
M_1	NA	500	2	sigma_resid	disp	NA	NA	NA	NA	33.1
M_1	NA	500	1	Intercept	fixef	NA	Intercept	NA	NA	106.7
M_1	NA	1000	2	sigma_resid	disp	NA	NA	NA	NA	28.9
M_1	NA	1000	1	Intercept	fixef	NA	Intercept	NA	NA	108.7
M_1	NA	1500	2	sigma_resid	disp	NA	NA	NA	NA	29.8
M_1	NA	1500	1	Intercept	fixef	NA	Intercept	NA	NA	109.1
M_1	NA	2000	2	sigma_resid	disp	NA	NA	NA	NA	30.2
M_1	NA	2000	1	Intercept	fixef	NA	Intercept	NA	NA	101.4
M_1	NA	2500	2	sigma_resid	disp	NA	NA	NA	NA	32.0
M_1	NA	2500	1	Intercept	fixef	NA	Intercept	NA	NA	109.8
M_1	NA	3000	2	sigma_resid	disp	NA	NA	NA	NA	30.3
M_1	NA	3000	1	Intercept	fixef	NA	Intercept	NA	NA	110.5
M_1	NA	3500	2	sigma_resid	disp	NA	NA	NA	NA	32.4
M_1	NA	3500	1	Intercept	fixef	NA	Intercept	NA	NA	103.5
M_1	NA	4000	2	sigma_resid	disp	NA	NA	NA	NA	28.7
M_1	NA	4000	1	Intercept	fixef	NA	Intercept	NA	NA	106.6

Note how the two parameters `Intercept` and `sigma` are assigned different parameter types: Effects and Dispersion. That is a generally useful classification made by the command `bayr::posterior`. It allows us to filter by type of parameter and produce CLUs:

```
P_1 %>%
  filter(type == "fixef") %>%
  clu()
```

model	parameter	type	fixef	center	lower	upper
M_1	Intercept	fixef	Intercept	106	100	112

That is almost precisely how the `bayr::fixef` command is implemented. Note that `coef` and `fixef` can be called on the rstanarm model object, directly, which produces the long posterior in the background.

```
coef(M_1)
```

parameter	type	fixef	center	lower	upper
Intercept	fixef	Intercept	106	100	112

```
detach(Sec99)
```

4.1.4 Center and interval estimates

The authors of Bayesian books and the various regression engines have different opinions on what to use as center statistic and credibility limits in a coefficient table. The best known option are: the mean, the median and the mode.

```
T_1 <-
  P_1 %>%
  group_by(parameter) %>%
  summarize(mean   = mean(value),
            median = median(value),
            mode   = mascutils::mode(value),
            lower = quantile(value, .025),
            upper = quantile(value, .975))
T_1
```

parameter	mean	median	mode	lower	upper
Intercept	105.9	106.0	105.7	100.1	112
sigma_resid	31.4	31.3	31.1	27.6	36

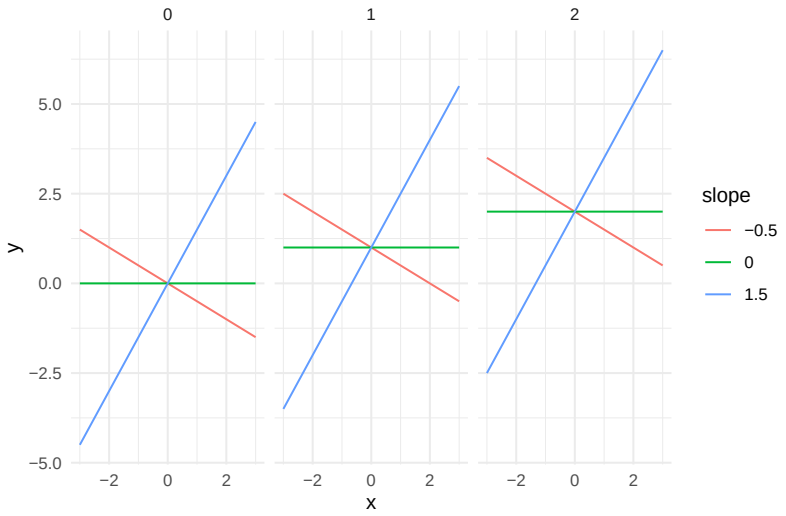
We observe that for the Intercept it barely matters which center statistic we use, but there are differences for the standard error. We investigate this further by producing a plot with the marginal posterior distributions of μ and σ with mean, median and mode.

```
T_1_long <-
  T_1 %>%
  gather(key = center, value = value, -parameter)
T_1_long
```

parameter	center	value
Intercept	mean	105.9
sigma_resid	mean	31.4
Intercept	median	106.0
sigma_resid	median	31.3
Intercept	mode	105.7
sigma_resid	mode	31.1
Intercept	lower	100.1
sigma_resid	lower	27.6
Intercept	upper	111.8
sigma_resid	upper	36.0

```
G_1 <- P_1 %>%
  ggplot(aes(x = value)) +
```

```
  facet_wrap(~parameter, scales = "free_x") +
  geom_density(fill = 1)  +
  geom_vline(aes(xintercept = value, col = center), data  = T_1_long)

G_1
```



This example demonstrates how the long format posterior works together with the GGplot graphics engine. A density plot very accurately renders how certainty is distributed over the range of a parameter. In order to produce vertical lines for point estimate and limits, we first make the summary table long, with one value per row. This is not how we would usually like to read it, but it is very efficient for adding to the plot.

When inspecting the two distributions, it appears that the distribution of Intercept is completely symmetric. For the standard error, in contrast, we note a slight left skewness. This is rather typical for dispersion parameters, as these have a lower boundary. The closer the distribution sits to the boundary, the steeper becomes the left tail.

A disadvantage of the *mean* is that it may change under monotonic transformations. A monotonic transformations is a recoding of a variable x_1 into a new variable x_2 by a transformation function ϕ (*phi*) such that the order of values stays untouched. Examples of monotonic functions are the logarithm ($x_2 = \log(x_1)$), the exponential function ($x_2 = \exp(x_1)$), or simply $x_2 = x_1 + 1$. A counter example is the quadratic function $x_2 = x_1^2$. In data analysis monotonous transformations are used a lot. Especially Generalized Linear Models make use of monotonous link functions to establish linearity 6.1.1. Furthermore, the mean can also be highly influenced by outliers.

The *mode* of a distribution is its point of highest density. It is invariant under monotonic transformations. It also has a rather intuitive meaning as the most likely value for the true parameter. Next to that, the mode is compatible with classic maximum likelihood estimation. When a Bayesian takes a pass on any prior information, the posterior mode should precisely match the results of a classic regression engine (e.g. `glm`). The main disadvantage of the mode is that it has to be estimated by one of several heuristic algorithms. These add some computing time and may fail when the posterior distribution is bimodal. However, when that happens, you probably have a more deeply rooted problem, than just deciding on a suitable summary statistic.

The *median* of a distribution marks the point where half the values are below and the other half are equal or above. Technically, the median is just the 50% quantile of the distribution. The median is extremely easy and reliable to compute, and it shares the invariance of monotonous transformations. This is easy to conceive: The median is computed by ordering all values in a row and then picking the value that is exactly in the middle. Obviously, this value only changes if the order changes, i.e. a non-monotonous function was applied. For these advantages, I prefer using the median as center estimates. Researchers who desire a different center estimate can easily write their own `clu`.

In this book, *2.5% and 97.5% certainty quantiles* are routinely used to form *95% credibility intervals (CI)*. There is nothing special about these intervals, they are just conventions, Again, another method exists to obtain CIs. Some authors prefer to report the *highest posterior interval (HPD)*, which is the narrowest interval that contains 95% of the probability mass. While this is intriguing to some extent, HPDs are not invariant to monotonic transformations, either.

So, the parameter extraction commands used here give the median and the 2.5% and 97.5% limits. The three parameters have in common that they are quantiles, which are handled by Rs `quantile` command. To demystify the `clu`, here is how you can make a basic coefficient table yourself:

```
P_1 %>%
  group_by(parameter) %>%
  summarize(center = quantile(value, 0.5),
            lower  = quantile(value, 0.025),
            upper  = quantile(value, 0.975)) %>%
  kable()
```

parameter	center	lower	upper
Intercept	106.0	100.1	112
sigma_resid	31.3	27.6	36

Note that we get CIs for the dispersion parameter σ, too. Many classic analyses call σ are nuisance parameter and ignore it, or they blame high variation

between observations for not reaching "statistical significance" for the parameter of interest. Furthermore, classic regression engines don't yield any measures of certainty on dispersion parameters. I believe that understanding the amount of variation is often crucial for design research and several of the examples that follow try to build this case. This is why we should be glad that Bayesian engines report uncertainty on all parameters involved.

4.2 Walk the line: linear regression

In the previous section we have introduced the most basic of all regression models: the grand mean model. It assigns rather coarse predictions, without any real predictors. Routinely, design researchers desire to predict performance based on *metric variables*, such as:

- previous experience
- age
- font size
- intelligence level and other innate abilities
- level of self efficiacy, neuroticism or other traits
- number of social media contacts

To carry out such a research question, the variable of interest needs to be measured next to the outcome variable. And, the variable must vary. You cannot examine the effects of age or font size on reading performance, when all participants are of same age and you test only one size. Then, for specifying the model, the researcher has to come up with an expectation of how the two are related. Theoretically, that can be any mathematical function, but practically, a *linear function* is often presumed. The following plot shows a variety of linear relations between two variables x and y.

```
mascutils::expand_grid(intercept = c(0, 1, 2),
                       slope = c(-.5, 0, 1.5),
                       x = -3:3) %>%
  arrange(x) %>%
  mutate(y = intercept + x * slope,
         slope = as.factor(slope)) %>%
  ggplot(aes(x = x, y = y, color = slope)) +
  geom_line() +
  facet_grid(~intercept)
```

A linear function is a straight line, which is specified by two parameters: *intercept* β_0 and *slope* β_1:

$$f(x_1) = \beta_0 + \beta_1 x_{1i}$$

The intercept is *"the point where a function graph crosses the x-axis"*, or more formally:

$$f(x_1 = 0) = \beta_0$$

The second parameter, β_1 is called the *slope*. The slope determines the steepness of the line. When the slope is .5, the line will rise up by .5 on Y, when moving one step to the right on X.

$$f(x_1 + 1) = \beta_0 + \beta_1 x_{1i} + \beta_1$$

There is also the possibility that the slope is zero. In such a case, the predictor has no effect and can be left out. Setting $\beta_1 = 0$ produces a horizontal line, with y_i being constant over the whole range. This shows that the GMM is a special case of LRMs, where the slope is fixed to zero, hence $\mu_i = \beta_0$.

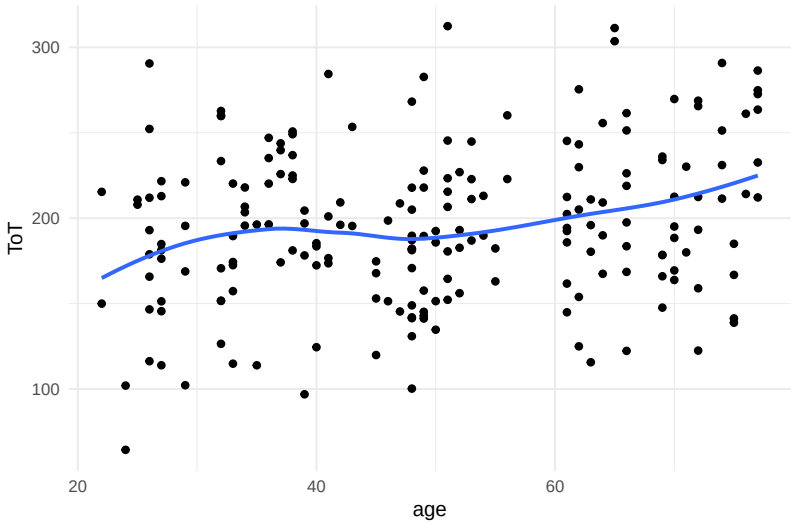
Linear regression gives us the opportunity to discover how ToT can be predicted by age (x_1) in the BrowsingAB case. In this hypothetical experiment, two designs A and B are compared, but we ignore this for now. Instead we ask: are older people slower when using the internet? Or: is there a linear relationship between age and ToT? The structural term is:

$$\mu_i = \beta_0 + \beta_1 \text{age}_i$$

This literally means: with every year of age, ToT increases by β_1 seconds. Before we run a linear regression with `stan_glm`, we visually explore the association between age and ToT using a scatter plot. The blue line in the graph is a so called a *smoother*, more specifically a LOESS. A smoother is an estimated line, just as linear function. But, it is way more flexible. Where the linear function is a straight stick fixed at a pivotal point, LOESS is more like a pipe cleaner. here, LOESS shows a more detailed picture of the relation between age and ToT. There is a rise between 20 and 40, followed by a stable plateau, and another rise starting at 60. Actually, that does not look like a straight line, but at least there is steady upwards trend.

```
attach(BrowsingAB)
```

```
BAB1 %>%
  ggplot(aes(x = age, y = ToT)) +
  geom_point()+
  geom_smooth(se = F, fullrange = F)
```



In fact, the BrowsingAB simulation contains what one could call a psychological model. The effect of age is partly due to farsightedness of participants (making them slower at reading), which more or less suddenly kicks in at a certain range of age. Still, we make do with a rough linear approximation. To estimate the model, we use the `stan_glm` command in much the same way as before, but add the predictor age. The command will internally check the data type of your variable, which is metric in this case. Therefore, it is treated as a *metric predictor* (sometimes also called covariate) .

```
M_age <-
  BAB1 %>%
  stan_glm(ToT ~ 1 + age,
           data = .)
```

```
T_age <- coef(M_age)
T_age
```

parameter	fixef	center	lower	upper
Intercept	Intercept	164.217	144.057	184.13
age	age	0.637	0.258	1.03

Is age associated with ToT? The coefficient table tells us that with every year of age, users get 0.64 seconds slower, which is considerable. It also tells us that the predicted performance at age = 0 is 164.22.

4.2.1 Transforming measures

In the above model, the intercept represents the predicted ToT at `age == 0`, of a newborn. We would never seriously put that forward in a stakeholder presentation, trying to prove that babies benefit from the redesign of a public website, would we? The prediction is bizarre because we intuitively understand that there is a discontinuity up the road, which is the moment where a teenager starts using public websites. We also realize that over the whole life span of a typical web user, say 12 years to 90 years, age actually is a proxy variable for two distinct processes: the rapid build-up of intellectual skills from childhood to young adulthood and the slow decline of cognitive performance, which starts approximately, when the first of us get age-related far-sightedness. Generally, with linear models, one should avoid making statements about a range that has not been observed. Linearity, as we will see in 6.1.1, always is just an approximation for a process that truly is non-linear.

Placing the intercept where there is no data has another consequence: the estimate is rather uncertain, with a wide 95% CI, $164.22[144.06, 184.13]_{CI95}$. As a metaphor, think of the data as a hand that holds the a stick, the regression line and tries to push a light switch. The longer the stick, the more difficult is becomes to hit the target.

4.2.1.1 Shifting an centering

Shifting the predictor is a pragmatic solution to the problem: "Shifting" means that the age predictor is moved to the right or the left, such that point zero is in a region populated with observations. In this case, two options seem to make sense: either, the intercept is in the region of youngest participants, or it is the sample average, which is then called *centering*. To shift a variable, just subtract the amount of units (years) where you want the intercept to be. The following code produces a shift of -20 and a centering on the original variable age:

```
BAB1 <-
  BAB1 %>%
  mutate(age_shft = age - 20,
         age_cntr = age - mean(age))

BAB1 %>%
  tidyr::gather("predictor", "age", starts_with("age")) %>%
  ggplot(aes(x = age, y = ToT)) +
  facet_grid(predictor~.) +
  geom_point() +
  geom_smooth(se = F, method = "lm", fullrange = T)
```

By shifting the age variable, the whole data cloud is moved to the left. To see what happens on the inferential level, we repeat the LRM estimation with the two shifted variables:

```
M_age_shft <-
  stan_glm(ToT ~ 1 + age_shft, data = BAB1)

M_age_cntr <-
  stan_glm(ToT ~ 1 + age_cntr, data = BAB1)
```

We combine the posterior distributions into one multi-model posterior and read the *multi-model coefficient table*:

```
P_age <-
  bind_rows(posterior(M_age),
            posterior(M_age_shft),
            posterior(M_age_cntr))


T_age <- coef(P_age)
T_age
```

model	parameter	fixef	center	lower	upper
M_age	Intercept	Intercept	164.217	144.057	184.13
M_age	age	age	0.637	0.258	1.03
M_age_cntr	Intercept	Intercept	195.825	189.651	201.91
M_age_cntr	age_cntr	age_cntr	0.649	0.246	1.05
M_age_shft	Intercept	Intercept	176.883	163.803	190.13
M_age_shft	age_shft	age_shft	0.641	0.250	1.02

```
detach(BrowsingAB)
```

```
## [1] "BAB1"
```

When comparing the regression results the shifted intercepts have moved to higher values, as expected. Surprisingly, the simple shift is not exactly 20 years. This is due to the high uncertainty of the first model, as well as the relation not being exactly linear (see Figure XY). The shifted age predictor has a slightly better uncertainty, but not by much. This is, because the region around the lowest age is only scarcely populated with data. Centering, on the other hand, results in a highly certain estimate, due to the dence data. The

slope parameter, however, practically does not change, neither in magnitude nor in certainty.

Shift (and centering) move the scale of measurement and make sure that the intercept falls close (or within) the cluster of observations. Shifting does not change the unit size, which is still years. For most metric predictors that would also not be desireable, as the unit of measurement is natural and intuitive.

4.2.1.2 Rescaling

Most rating scales are not natural units of measure. Most of the time it is not meaningful to say: "the user experience rating improved by one". The problem has two roots, as I will illustrate by the following four rating scale items:

This product is ...

1. difficult to use `|1 ... X ... 3 ... 4 ... 5 ... 6 ... 7|` easy to use
2. from hell `|-----X----------------------| (10cm)` heavenly
3. neutral `|1 ... X ... 3 ... 4|` uncanny

If you would employ these three scales to assess one and the same product, the data could look like this:

```
set.seed(42)
Raw_ratings <-
  tibble(Part = 1:100,
         difficult_easy  = mascutils::rrating_scale(100, 0, .5,
                                                    ends = c(1,7)),
         heavenly_hell   = mascutils::rrating_scale(100, 0, .2,
                                                    ends = c(0,10), bin = F),
         neutral_uncanny = mascutils::rrating_scale(100, -.5, .5,
                                                    ends = c(1,5)))

head(Raw_ratings)
```

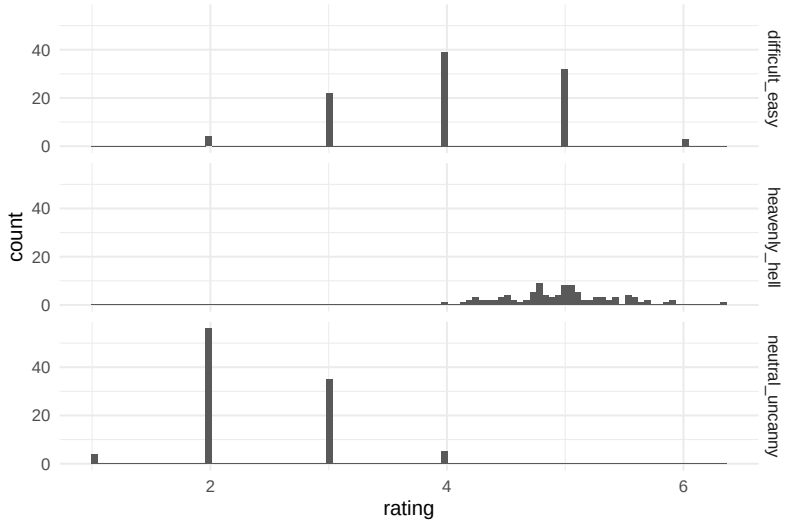
Part	difficult_easy	heavenly_hell	neutral_uncanny
1	5	5.60	1
2	4	5.52	3
3	4	4.50	3
4	5	5.91	4
5	4	4.67	2
6	4	5.05	2

In the following, we are comparing the results of these three items. However, they came in the wide format, as you would use to create a correlation table.

For a tidy analysis, we first make the data set long. Ratings are now classified by the item they came from. We can produce a grid histogram.

```
D_ratings <-
  Raw_ratings %>%
  gather(key = Item, value = rating, -Part) %>%
  mascutils:::as_tbl_obs()

D_ratings
```

```
D_ratings %>%
  ggplot(aes(x = rating)) +
  facet_grid(Item ~ .) +
  geom_histogram() + xlim(0, 10)
```



The first problem is that rating scales have been designed with different end points. The first step when using different rating scales is shifting the left-end point to zero and dividing by the range of the measure (`upper - lower` boundary). That brings all items down to the range between zero and one. Note how the following tidy code joins in a table that holds the properties of our items.

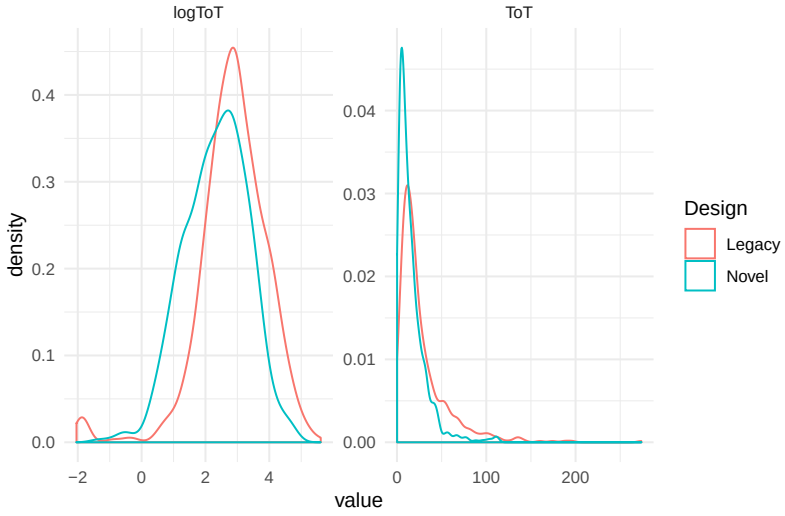
```
D_Items <- tribble(~Item,             ~lower, ~upper,
                   "difficult_easy",  1,      7,
                   "heavenly_hell",   0,      10,
                   "neutral_uncanny", 1,      5)
```

```
D_ratings <-
  D_ratings %>%
  left_join(D_Items, by = "Item") %>%
  mutate(scaled = (rating - lower)/(upper - lower))

D_ratings %>%
  ggplot(aes(x = scaled)) +
  facet_grid(Item ~ .) +
  geom_histogram(bins = 100) +
  xlim(0,1)
```

This partly corrects the horizontal shift between scales. However, the ratings on the third item still are shifted relative to the other two. The reason is that the first two items have the neutral zone right in the center, whereas the third item is neutraul at its left-end point. The second inconsistency is that the second item uses rather extreme anchors (end point labels), which produces a tight accumulation in the center of the range (with a lot of polite people in the sample, at least). The three scales have been rescaled by their *nominal range*, but they differ in their observed variance.

z-transformation rescales a measure by its *observed variance*. A set of measures is z-transformed by centering it and scaling it by its own standard deviation.

```
D_ratings %>%
  group_by(Item) %>%
  mutate(zrating = (rating - mean(rating))/sd(rating)) %>%
#   mascutils::z_score(rating) %>%

  ggplot(aes(x = rating)) +
   facet_grid(Item ~ .) +
   geom_histogram(bins = 100)
```



By z-transformation, the three scales now exhibit the same mean location and the same dispersion. This could be used to combine them into one general score. Note however, that information is lost by this process, namely the differences in location or dispersion. If the research question is highly detailed, such as "Is the design consistently rated low on uncanniness?", this can no longer be answered from the z-transformed variable.

Finally, sometimes researchers use *logarithmic transformation* of outcome measures to reduce what they perceive as pathologies of tha data. In particular, many outcome variables do not follow a Normal distribution, as the random term of linear models assumes, but are left-skewed. Log-transformation often mitigates such problems. However, as we will see in chapter 6, linear models can be estimated gracefully with a random component that precisely matches the data as it comes. The following time-on-task data is from the IPump study, where nurses have tested two infusion pump interfaces:

```
attach(IPump)
```

```
D_pumps %>%
  mutate(logToT = log(ToT)) %>%
  select(Design, ToT, logToT) %>%
  gather(key = Measure, value = value, -Design) %>%
  ggplot(aes(x = value, color = Design)) +
  facet_wrap(Measure~., scale = "free") +
  geom_density()
```

```
detach(IPump)
```

Frequently, it is count measures and temporal measures to exhibit non-symmetric error distributions. By log transformation one often arrives at a reasonably Gaussian distributed error. However, the natural unit of te measure (seconds) gets lost by the transformation, making it very difficult to report the results in a quantitative manner.

4.2.2 Correlations

LRM render the quantitative relationship between two metric variables. Another commonly known statistic that seems to do something similar is Pearson's correlation statistic r (@??#associations)). In the following, we will see that a tight connection between correlation and linear coefficients exists, albeit both having their own advantages. For a demonstration, we reproduce the steps on a simulated data set where X and Y are linearly linked:

```
D_cor <-
  tibble(x = runif(100, 0, 50),
              y = rnorm(100, x *.2, 3))
```

```
D_cor %>%
  ggplot(aes(x = x, y = y)) +
  geom_point() +
  geom_smooth(method = "lm", se = F)
```

Recall, that r is covariance standardized for dispersion, not unsimilar to z-transformation 4.2.1 and that a covariance is the mean squared deviance from the population mean. This is how the correlation is decontaminated from the idiosyncracies of the involved measures, their location and dispersion. Similarly, the slope parameter in a LRM is a measure of association, too. It is agnostic of the overall location of measures since this is captured by the intercept. However, dispersion remains intact. This ensures that the slope and the intercept together retain information about location, dispersion and association of data, and we can ultimately make predictions. Still, there is a tight relationship between Pearson's r and a slope coefficient β_1, namely:

$$r = \beta_1 \frac{\sigma_X}{\sigma_Y}$$

For the sole purpose of demonstration, we here resort to the built-in non-Bayesian command `lm` for doing the regression.

```
M_cor <- lm(y ~ x, D_cor)
beta_1 <- stats::coef(M_cor)[2]

r <- beta_1 * sd(D_cor$x) / sd(D_cor$y)
r
```

```
##    x
## 0.66
```

The clue with Pearson's r is that it normalized the slope coefficient by the variation found in the sample. This resembles z-transformation as was introduced in 4.2.1. In fact, when both, predictor and outcome, are z-transformed before estimation, the coefficient equals Pearson's r exactly:

```
M_z <-
  D_cor %>%
  mutate(x_z = (x - mean(x))/sd(x),
         y_z = (y - mean(y))/sd(y)) %>%
  lm(y_z ~ x_z, .)

stats::coef(M_z)[2]
```

```
##  x_z
## 0.66
```

In regression modelling the use of coefficients allows for predictions made in the original units of measurement. Correlations, in contrast, are unit-less. Still, correlation coefficients play an imortant role in exploratory data analysis (but also in multilevel models, see @??re_correlations) for the following reasons:

1. Correlations between predictors and responses are a quick and dirty assessment of the expected associations.
2. Correlations between multiple response modalities (e.g., ToT and number of errors) indicate to what extent these responses can be considered exchangeable.
3. Correlations between predictors should be checked upfront to avoid problems arising from so-called collinearity.

The following table shows the correlations between measures in the MMN study, where we tested the association between verbal and spatial working memory capacity (Ospan and Corsi tests and performance in a search task on a website (clicks and time).

```
attach(MMN)

MMN_2 %>%
  select(Corsi, Ospan.A, Ospan.B, time, clicks) %>%
  corrr::correlate()
```

rowname	Corsi	Ospan.A	Ospan.B	time	clicks
Corsi	NA	0.177	0.116	0.068	0.137
Ospan.A	0.177	NA	0.876	0.053	0.111
Ospan.B	0.116	0.876	NA	0.060	0.119
time	0.068	0.053	0.060	NA	0.838
clicks	0.137	0.111	0.119	0.838	NA

```
detach(MMN)
```

These correlations give an approximate picture of associations in the data:

1. Working memory capacity is barely related to performance.
2. There is a strong correlations between the performance measures.
3. There is a strong correlation between the two predictors Ospan.A and Ospan.B.

Linear coefficients and correlations both represent associations between measures. Coefficients preserve units of measuremnent, allowing us to make meaningful quantitative statements. Correlations are re-scaled by the observed dispersion of measures in the sample, making them unit-less. The advantage is that larger sets of associations can be screened at once and compared easily.

4.2.3 Endlessly linear

On a deeper level the bizarre age = 0 prediction is an example of a principle , that will re-occur several times throughout this book.

In our endless universe everything is finite.

A well understood fact about LRM is that they allow us to fit a straight line to data. A lesser regarded consequence from the mathematical underpinnings of such models is that this line extends infinitely in both directions. To fulfill this assumption, the outcome variable needs to have an infinite range, too, $y_i \in [-\infty; \infty]$ (unless the slope is zero). Every scientifically trained person and many lay people know, that even elementary magnitudes in physics are finite: all speeds are limited to $\approx 300.000 km/s$, the speed of light, and temperature has a lower limit of $-276 \check{r} C$ (or $0 \check{r} K$). If there can neither be endless

acceleration nor cold, it would be daring to assume any psychological effect to be infinite in both directions.

The endlessly linear assumption (ELA) is a central piece of all LRMs, that is always violated in a universe like ours. So, should we never ever use a linear model and move on to non-linear models right away? Pragmatically, the LRM often is a reasonably effective approximation. At the beginning of section 4.2, we have seen that the increase of time-on-task by age is not strictly linear, but follows a more complex curved pattern. This pattern might be of interest to someone studying the psychological causes of the decline in performance. For the applied design researcher it probably suffices to see that the increase is monotonous and model it approximately by one slope coefficient. In 4.4.1 we will estimate the age effects for designs A and B separately, which lets us compare fairness towards older people.

As has been said, theorists may desire a more detailed picture and see a disruption of linearity as indicators for interesting psychological processes. A literally uncanny example of such theoretical work will be given when introducing polynomial regression 4.6. For now, linear regression is a pragmatic choice, as long as:

1. the pattern is monotonically increasing
2. any predictions stay in the observed range and avoid the boundary regions, or beyond.

4.3 Factorial Models

In the previous section we have seen how linear models are fitting the association between a metric predictor X and an outcome variable Y to a straight line with a slope and a point of intercept. Such a model creates a prediction of Y, given you know the value of measure X.

However, in many research situations, the predictor variable carries not a measure, but a *group label*. *Factor variables* assign observations to one of a set of predefined groups, such as the following variables do in the BrowsingAB case:

```
attach(BrowsingAB)

BAB5 %>%
  select(Part, Task, Design, Gender, Education, Far_sighted) %>%
  sample_n(8)
```

```
detach(BrowsingAB)
```

Two of the variables, Gender and Education clearly carry a group membership of the participants. That is a natural way to think of people groups, such as male of female, or which school type they went to. But, models are inert to anthropocentrism and can divide everything into groups. Most generally, it is always the observations, i.e. the rows in a (tidy) data table, which are divided into groups. Half of the observations habe been made with design A, the rest with B.

The data also identifies the participant and the task for every observation. Although do we see numbers on participants, these are factors, not metric variables. If one had used initials of participants, that would not make the slightest difference of what this variable tells. It also does not matter, whether the researcher has actually created the levels, for example by assigning participants to one of two deisgn conditions, or has just observed it, such as demographic variables.

Factorial models are frequently used in experiments, where the effect a certain condition on performance is measured. In design research, that is the case when comparing two (or more) designs and the basic model for that, the comparison of group means model, will be introduced, first 4.3.1, with more details on the inner workings and variations in the two sections that follow: dummy variables 4.3.2 and 4.3.3. A CGM requires that one can think of one of the groups as some kind of default to which all the other conditions are compared to. That is not always given. When groups are truly equal among sisters, the absolute means model (AMM) 4.3.4 does just estimates the absolute group mans, being like multi-facetted GMMs.

Factors are not metric, but sometimes they have a natural ordered, for example levels of education, or position in a sequence. In section 4.3.5 we will apply an ordered factorial model to a learning sequence.

4.3.1 A versus B: Comparison of groups

The most common linear models on factors is the *comparison of groups* (CGM), which replaces the commonly known analysis of variance (ANOVA). In design research group comparisons are all over the place, for example:

- comparing designs: as we have seen in the A/B testing scenario
- comparing groups of people, based on e.g. gender or whether they have a high school degree
- comparing situations, like whether an app was used on the go or standing still

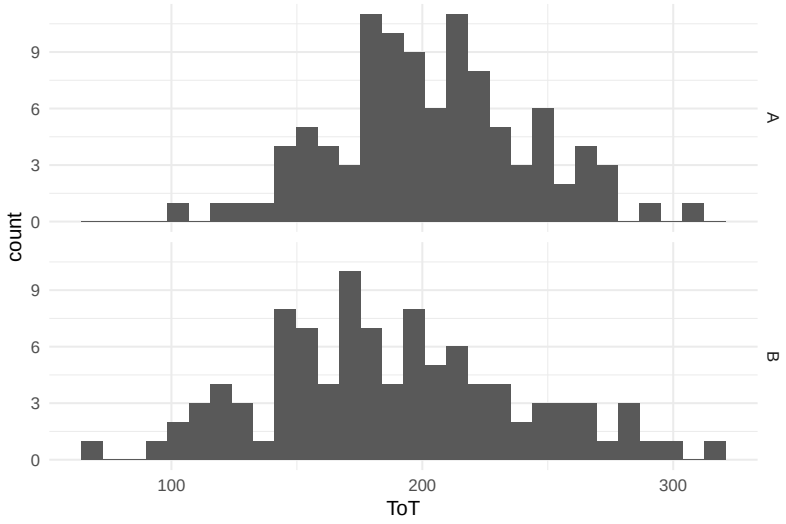
In order to perform a CGM, a variable is needed that establishes the groups. This is commonly called a *factor*. A factor is a variable that identifies members of groups, like "A" and "B" or "male" and "female". The groups are called *factor levels*. In the BrowsingAB case, the most interesting factor is Design with its levels A and B.

Asking for differences between two (or more) designs is routine in design research. For example, it could occur during an overhaul of a municipal website. With the emerge of e-government, many municipal websites have grown wildly over a decade. What once was a lean (but not pretty) 1990 website has grown into a jungle over time, to the disadvantage for users. The BrowsingAB case could represent the prototype of a novel web design, which is developed and tested via A/B testing at 200 users. Every user is given the same task, but sees only one of the two designs. The design team is interested in: *Do the two web designs A and B differ in user performance?*

Again, we first take a look at the raw data:

```
attach(BrowsingAB)
```

```
BAB1 %>%
  ggplot(aes(x = ToT)) +
  geom_histogram() +
    facet_grid(Design~.)
```



This doesn't look too striking. We might consider a slight advantage for design B, but the overlap is immense. We perform the CGM. Again, this is a two-step procedure:

1. The `stan_glm` command lets you specify a simple formula to express the dependency between one or more predictors (education) and an outcome variable (ToT). It performs the parameter estimation using the method of *Markov-Chain Monte-Carlo Sampling*. The results are stored in a new object `M_CGM`.
2. With the `fixef` command the estimates are extracted and can be interpreted.

```
M_CGM <-
  BAB1 %>%
  stan_glm(ToT ~ 1 + Design,
           data = .)
```

```
T_Design <- coef(M_CGM)
T_Design
```

parameter	fixef	center	lower	upper
Intercept	Intercept	203	194.6	212.19
DesignB	DesignB	-15	-27.6	-2.61

The model contains two parameters, one Intercept and one slope. Wait a second? How can you have a slope and a "crossing point zero", when there is no line, but just two groups? This will be explained further in 4.3.2 and 4.3.3. Fact is, in the model at hand, the Intercept is the *mean of a reference group*. Per default, stan_glm chooses the alphabetically first group label as the reference group, in this case design A. We can therefore say that design A has an average performance of $203.36[194.57, 212.19]_{CI95}$.

The second parameter is the effect of "moving to design B". It is given as the *difference to the reference group*. With design B it took users $14.98[27.58, 2.61]_{CI95}$ seconds less to complete the task. However, this effect appears rather small and there is huge uncertainty about it. It barely justifies the effort to replace design A with B. If the BrowsingAB data set has some exciting stories to tell, the design difference is not it.

A frequent user problem with CGMs is that the regression engine selects the alphabetically first level as the reference level, which often is not correct. Supposed, the two designs had been called Old (A) and New (B), then regression engine would pick New as the reference group. Or think of non-discriminating language in statistical reports. In BrowsingAB, gender is coded as f/m and female participants conquer the Intercept. But, sometimes my students code gender as v/m or w/m. Oh, my dear! The best solution is, indeed, to think upfront and try to find level names that make sense. If that is not possible, then the factor variable, which is often of type `character` must be made a

factor, which is a data type in its own right in R. When the regression engine sees a factor variable, it takes the first factor level as reference group. That would be nice, but when a factor is created using the `as.factor`, it again takes an alphabethical order of levels. This is over-run by giving a vector of levels in the desired order. The tidy Foracts package provides further commands to set the order of a factor levels.

```
Gender <- sample(c("v","m"), 4, replace = T)

factor(Gender)
```

```
## [1] m m v v
## Levels: m v
```

```
factor(Gender, c("v", "m"))
```

```
## [1] m m v v
## Levels: v m
```

```
detach(BrowsingAB)
```

4.3.2 Not stupid: dummy variables

Are we missing anything so far? Indeed, I avoided to show any mathematics on factorial models. The CGM really is a linear model, although it may not appear so, at first. So, how can a variable enter a linear model equation, that is not a number? Linear model terms are a sum of products $\beta_i x_i$, but factors cannot just enter such a term. What would be the result of DesignB $\times \beta_1$?

Factors basically answer the question: *What group does the observation belong to?*. This is a label, not a number, and cannot enter the regression formula. *Dummy variables* solve the dilemma by converting factor levels to numbers. This is done by giving *every level l of factor K* its own dummy variable *K_l*. Now every dummy represents the simple question: *Does this observation belong to group DesignB?*. The answer is coded as 0 for "Yes" and 1 for "No".

```
attach(BrowsingAB)
```

```
BAB1 <-  BAB1 %>%
  mutate(Design_A = if_else(Design == "A", 1, 0),
         Design_B = if_else(Design == "B", 1, 0))
BAB1 %>%
  select(Obs, Design, Design_A, Design_B, ToT) %>%
```

```
  sample_n(8) %>%
  kable()
```

Obs	Design	Design_A	Design_B	ToT
32	A	1	0	196
90	A	1	0	193
97	A	1	0	183
188	B	0	1	146
178	B	0	1	235
184	B	0	1	286
151	B	0	1	152
18	A	1	0	260

The new dummy variables are numerical and can very well enter a linear formula, every one getting its own coefficient. For a factor K with levels A, B and C the linear formula can include the dummy variables K_{Ai} and K_{Bi}:

$$\mu_i = K_{Ai}\beta_A + K_{Bi}\beta_B$$

The zero/one coding acts like a switches. When $K_{Ai} = 1$, the parameter β_A is switched on and enters the sum, and β_B is switched off. An observation of group A gets the predicted value: $\mu_i = \beta_A$, vice versa for members of group B.

```
M_dummy_1 <-
  stan_glm(ToT ~ 0 + Design_A + Design_B,
     data = BAB1)
```

```
coef(M_dummy_1)
```

parameter	fixef	center	lower	upper
Design_A	Design_A	203	194	212
Design_B	Design_B	188	180	197

In its predictions, the model `M_dummy` should be equivalent to the CGM model `M_CGM`, but the coefficients mean something different: they are exactly the group means. This model we call an *absolute means model (AMM)* and will discuss it in section 4.3.4. First, we have to come back to the question, how the regression engine produces its dummy variables, such that the coefficients are differences towards one Intercept. This is called *treatment contrasts* 4.3.3.

4.3.3 Treatment contrast

The default behaviour of regression engines, when encountering a factor, is to select the first level as reference group and estimate all other levels relative to that. Coefficients express differences. This fully makes sense if the effect of a treatment is what you are after, and is therefore called *treatment contrasts*. Treatment contrasts do not have anything special or natural to them, but is a very particular way of thinking about levels of a factor, namely that *one level is special*. In controlled experiments, this special level often is the *control condition*, whereas the coefficients are the effects of well-defined manipulations. This most prominently is the case for clinical trials, where the *placebo group* is untreated. This works well in all situations where a default situation exists and the other factor levels can be thought of manipulations of the default:

- A redesign as an improvement over the *current design*.
- A quiet, comfortable environment is the *optimal situation* for cognitive performance.
- There is a *minimum level* of education requird for most jobs

We have seen how to create dummy variables ourselves by means of mututally exclusive on-off switches, which results in absolute means coefficients. Regression engines quietly assume that treatment effects is what the user wants and expand dummy variables in a different way: For a factor with levels A and B, the dummy for B is an on-off switch, whereas the reference level A is set *always on*. This is called *treatment contrast coding*:

```
BAB1 <-  BAB1 %>%
  mutate(Intercept = 1,
         Design_B = if_else(Design == "B", 1, 0))
BAB1
```

A frequent user problem with treatment coding is that the regression engine selects the alphabetically first level as the reference level. Supposed, the two designs had been called Old (A) and New (B), then regression engine would pick New as the reference group. By the following you can define dummy variables to have Old be the reference. (But recall the more convenvient ways that were outlined earlier 4.3.1.)

```
BAB1 %>%
  mutate(Design = if_else(Design == "A", "Old", "New")) %>%
  mutate(Intercept = 1,
         Design_B = if_else(Design == "New", 1, 0))
```

The following chapters deal with more variations of factorial models. Next, we will take a closer look at the absolute means model 4.3.4, which is useful,

when a reference group does not come natural. In section 4.3.5, we deal with factorial models, where levels are ordered and introduce *contrast codings*.

```
detach(BrowsingAB)
```

4.3.4 Absolute Means Model

Not all factor variables are experimental and identifying a default can be difficult or unnatural. This often happens when the levels are just a set of conditions that you have *found as given*, such as the individuals in the human population, or all words in a language. Such is the case in the IPump study, where every session was composed of a set of tasks, such as starting the device or entering a dose. These tasks were taken from existing training material and including them as a factor could help identify areas for improvement. Although the tasks form a sequence, they are equally important for the operation. Not one can be singled out as default. Treatment coding would force us to name one default task for the Intercept.

The *absolute means model* represents all levels by their absolute means. If you put in a factorial predictor with eight levels, you will get eight coefficients, which are the mean outcomes of every level. Of course, what you can no longer do is find differences between levels.

We have seen in 4.3.2 how to create a AMM dummy variables. IN fact, the linear models formula language this can be done more directly by either of the two option below, (but just leaving out the `1 +` does not suffice):

- `0 + Task`
- `Task - 1`

In the following, we estimate an AMM using the formula method. In the IPump study we had a sample of nurses do a sequence of eight tasks on a medical infusion pump with a novel interface design. For the further development of such a design it may be interesting to see, which tasks would most benefit from design improvements. A possible way to look at it is by saying that a longer task has more potential to be optimized. Under such a perspective, tasks are an equal set and there is no natural reference task for a CGM. Instead we estimate the absolute group means and visualize the marginal postertior distributions.

```
attach(IPump)
```

```
M_AMM_1 <-
  D_Novel %>% stan_glm(ToT ~ 0 + Task,
                       data = .)
```

```
coef(M_AMM_1) %>%
  rename(Task = fixef ) %>%
  ggplot(aes(x = Task, y = center, ymin = lower, ymax = upper)) +
  geom_point(size = 2) +
  geom_errorbar()
```

The plot shows the absolute means and we can easily discover that Task 2 is by far the longest and that tasks differ a lot, indeed. None of these relations can easily be seen in the CGM plot. Note that the AMM is not a different model than the treatment effects model. It is just a *different parametrization*, which makes interpretation easier. Both models produce the exact same predictions.

```
detach(IPump)
```

The choice between CGM and AMM depends on whether a factor represents designed manipulations or whether it is more something that has been collected. Psychological experiments often have fully designed stimuli, because only then can differences between stimuli get an unambiguous causal interpretation. In the Stroop experiment, stimuli show a color word (red) written in a color (Red). Both properties are well-defined and can be manipulated freely by the experimenter. That is what you can call a fully controlled stimulus design. One could argue, that colors and color words have been collected from our perceptual and cultural heredity and are therefore are not really manipulated. This is a valid issue, but it does not matter so much, because the real manipulation is congruency and that is crystal clear:

```
tribble(~Word, ~Color,
        "red", "Red",
        "blue", "Blue",
        "green", "Green") %>%
  tidyr::complete(Word, Color) %>%
  mutate(Condition = if_else(Word == str_to_lower(Color),
                             "congruent",
                             "incongruent"))
```

Word	Color	Condition
blue	Blue	congruent
blue	Green	incongruent
blue	Red	incongruent
green	Blue	incongruent
green	Green	congruent
green	Red	incongruent
red	Blue	incongruent
red	Green	incongruent
red	Red	congruent

In more common version of the Stroop task, a neutral condition is added, where the task stays the same, but the word is a non-color word. Almost every word in a language is a non-color word. And among all those words, color neutrality is not so clear cut. All objects around us show colors, and color symbolism pervades all areas of our lives. We can easily imagine that "tree" is a less neutral word (green leaves), but to what extent is "hood" (little red riding ...) or "sad" (blue) color-neutral. As long as we don't know this precisely for a word, it is not controlled by manipulation, and we can give no default. That being said, I do not recommend you examine the words by a factor, as we have done for tasks. Instead, multi-level models apply best when factors represent a collection of natural objects 5.

4.3.5 Ordered Factors Models

Factors usually are not metric, which would require them to have units (like years or number of errors) and an order. Age, for example, has the unit of years, which makes statements possible such as: "*per year of age*, participants slow down by ...". The same cannot be said for levels of education. We could assign these levels the numbers 0, 1 and 2 to express the order, but we cannot assume that going from Low to Middle is the same amount of effective education as going from Middle to High. Factorial models are indifferent towards orders and therefore can simply be used for ordered factors. For level of education, we could just use a CGM or AMM, the only issue being that the graphics and regression engines order factors alphabetically: High, Low, Middle.

```
attach(BrowsingAB)

BAB1 %>%
  ggplot(aes(x = Education, y = ToT)) +
  geom_boxplot()
```

The following changes the order of levels of GGplot engine is respecting that, and so is the regression engine, putting the intercept on level Low.

```
BAB1$Education <- factor(as.character(BAB1$Education),
                         levels = c("Low", "Middle", "High"))

BAB1 %>%
  group_by(Education) %>%
  summarize(mean_ToT = mean(ToT)) %>%
  ggplot(aes(x = as.integer(Education), y = mean_ToT)) +
  geom_step() +
  scale_x_continuous(breaks=1:3)
```

Note that R also knows a separate variable type called ordered factors. This only seemingly is useful. In fact, if we run a linear model with an ordered factor as predictor, the estimated model will be so unintelligible that I will not attempt to explain it here.

```
M_OFM_1 <-
  BAB1 %>%
  stan_glm(ToT ~ 1 + Education, data = .)
```

```
coef(M_OFM_1)
```

parameter	fixef	center	lower	upper
Intercept	Intercept	211.2	199.9	223.21
EducationMiddle	EducationMiddle	-13.1	-29.8	2.91
EducationHigh	EducationHigh	-29.0	-44.2	-14.33

```
detach(BrowsingAB)
```

A basic ordered factor model is just a CGM where the coefficients are shown in the desired order. The second and third coefficient carry the respective difference towards level Low. EducationHigh is *not* the difference towards EducationMiddle. In this case it makes sense to understand Middle and High as a smaller step or a larger step up from Low. It is not always like that. Sometimes, the only way of moving from the reference group to some other level implies going through all the intermediates, just like walking up a stairway. Then it makes more sense to use a model where coefficients are individual steps. In the IPump study, we looked at the speed of learning of a novel interface design by letting the participants repeat a set of tasks in three successive

sessions. here the three sessions make a stairway: going from the fist to the third session always involves the second session. Before we come to that, we first have to see, why Session must be an ordered factor and not a metric predictor.

The first idea that could come to mind is to take session as a metric predictor and estimate a LRM – there is an ordering and it is the same amount of training per step, which you could call a unit. The thing with learning processes is that they are curved, more precisely, they gradually move towards an asymptote. The following curve shows the effect of a hypothetical training over 12 sessions. What we see is that the steps are getting smaller when training continues. While the amount of training is the same, the effect on performance declines, which is also called a curve of diminishing returns. The asymptote of this curve is the *maximum performance* the participant can reach, which theoretically is only reached in infinity. The following code defines an exponential learning curve function and renders an example.

```
learning_curve <-
  function(session, amplitude, rate, asymptote)
    amplitude * exp(-rate * session) + asymptote

tibble(session = as.integer(1:12)) %>%
  mutate(ToT = learning_curve(session, 10, .3, 2)) %>%
  ggplot(aes(x = session, y = ToT)) +
  geom_step() +
  scale_x_continuous(breaks=1:12)
```

LRMs can only do straight lines, which means constant effects, whereas learning curves have diminishing effects. For short learning sequences, we can use ordered factorial models, where every session becomes a level. As these levels get their own coefficients, the steps no longer have to be constant. When levels are ordered, the two endpoint levels (first session, last session) can serve as a natural reference group for the intercept. However, how useful would it be to express the performance in session 3 as differences to reference level (session 1). It is more natural to think of learning to take place incrementally, like *walking up stairways*, where the previous step always is your reference.

```
attach(IPump)

D_Novel %>%
  group_by(Session, session) %>%
  summarize(mean_ToT = mean(ToT)) %>%
  ggplot(aes(x = as.integer(Session), y = mean_ToT)) +
```

```
  geom_step() +
  scale_x_continuous(breaks=1:3)
```

This is what a factorial model with *stairway dummy coding* does. The first coefficient β_0 is the starting point, for example the first session, and all other coefficients (β_1, β_2) are a sequence of step sizes. The expected value μ_i for session K, using stairways dummies K_0, K_1, K_2 is:

$$\begin{aligned} \mu_i = & K_{1i}\beta_0 & + \\ & K_{2i}(\beta_0 + \beta_1) & + \\ & K_{2i}(\beta_0 + \beta_1 + \beta_2) \end{aligned}$$

Thinking of these dummy variables as switches once again: Recall that treatment dummies have an always-on reference level and exclusive switches for the other levels 4.3.2. Stairways dummies are like a *incremental switches*: when switch K is on, this implies all previous switches are on, too. *Stairways-down* dummies are made as follows:

```
D_Novel <-
  D_Novel %>%
  mutate(Session_1 = 1,
         Step_1 = as.integer(session >= 1),
         Step_2 = as.integer(session >= 2))

D_Novel %>%
  distinct(session, Session_1, Step_1, Step_2) %>%
  arrange(session) %>%
  as_tibble()
```

session	Session_1	Step_1	Step_2
0	1	0	0
1	1	1	0
2	1	1	1

Now we can run a factorial model using these stairway-down dummies, where the intercept is the upper floor and we are loosing height at every step:

```
M_OFM_2 <- stan_glm(ToT ~ Session_1 + Step_1 + Step_2, data = D_Novel)
```

```
coef(M_OFM_2)
```

parameter	fixef	center	lower	upper
Intercept	Intercept	23.73	21.62	25.89
Step_1	Step_1	-10.39	-13.37	-7.32
Step_2	Step_2	-2.38	-5.47	0.69

The Intercept is the performance in the first level, which is *initial performance*. The first step is huge, almost reducing ToT by one half. The second step is much smaller than the first and tiny compared to initial performance. We see that high performance can be reached after just a few training sessions. Clearly, the device is easy to learn.

Another question that arises is what level of performance is reached in the end. Is *maximum performance* good enough, actually? Strictly, this would require a non-linear learning curve model, which would contain an estimate for maximum performance. With an OFM, the best guess we have is *final performance*. Because the second step was already small, we may believe that the asymptote is not so far any more. And we know that final performance is a conservative estimate for maximum performance. With a *stairway-up model* model, the Intercept is the basement an we walk up step-by-step.

```
D_Novel <-
  D_Novel %>%
  mutate(Session_3 = 1,
         Step_1 = as.integer(session <= 1),
         Step_2 = as.integer(session <= 0))

D_Novel %>%
  distinct(session, Session_3, Step_1, Step_2) %>%
  arrange(desc(session)) %>%
  as_tibble()
```

session	Session_3	Step_1	Step_2
2	1	0	0
1	1	1	0
0	1	1	1

```
M_OFM_3 <- stan_glm(ToT ~ Session_3 + Step_1 + Step_2, data = D_Novel)
```

```
coef(M_OFM_3)
```

parameter	fixef	center	lower	upper
Intercept	Intercept	10.96	8.807	13.05
Step_1	Step_1	2.42	-0.563	5.43

parameter	fixef	center	lower	upper
Step_2	Step_2	10.30	7.387	13.37

The Intercept is an estimate of final performance and we can ask whether this level of efficiency is actually good enough. From a methodological perspective the results of this study indicate that it might often be worth-while to let participants do multiple session and observe the learning process. In particular, when users do their tasks routinely with a device, like the nurses, initial performance can be a very poor estimate for long-term performance.

```
detach(IPump)
```

To wrap it up: Factorial models use dummy variables to make factor levels numerical. These dummy variables can be understood as arrays of switches that can be arranged in different patterns:

1. Exclusive on-off switches produce an AMM. An AMM is the least specified among factorial models, all levels are equal. Often this is the best choice when the levels were drawn from a population.
2. One always-on Intercept and exclusive on-off switches produces a CGM with treatment effects. When a default level can be identified, such as the placebo condition in clinical trials, treatment contrasts are a good choice.
3. Stairway dummies produce an OFM, taking one end-point as first coefficient and stepping up (or down). This is particularly useful for short learning curves.

4.4 Putting it all together: multi predictor models

Design researchers are often collecting data under rather wild conditions. Users of municipal websites, consumer products, enterprise information systems and cars can be extremely diverse. At the same time, Designs vary in many attributes, affecting the user in many different ways. There are many variables in the game, and even more possible relations. With *multi predictor models* we can examine the simultaneous influence of everything we have recorded. First, we will see, how to use models with two or more continuous predictors. Subsequently, we address the case of multi-factorial designs. Finally, we will see examples of models, where metric predictors and factors make a bunch of regression lines.

4.4.1 On surface: multiple regression models

Productivity software, like word processors, presentation and calculation software or graphics programs have evolved over decades. For every new release, dozens of developers have worked hard to make the handling more efficient and the user experience more pleasant. Consider a program for drawing illustrations: basic functionality, such as drawing lines, selecting objects, moving or colourizing them, have practically always been there. A user wanting to draw six rectangles, painting them red and arranging them in a grid pattern, can readily do that using basic functionality. At a certain point of system evolution, it may have been recognized that this is what users repeatedly do: creating a grid of alike objects. With the basic functions this is rather repetitive and a new function was created, called "copy-and-arrange". Users may now create a single object, specify rows and columns of the grid and give it a run.

The new function saves time and leads to better results. Users should be very excited about the new feature, should they not? Not quite, as [%Carroll in Rosson] made a very troubling observation: adding functionality for the good of efficiency may turn out ineffective in practice, as users have a strong tendency to stick with their old routines, ignoring new functionality right away. This troubling observation has been called the *active user paradox (AUP)* [%AUP].

Do all users behave that way? Or can we find users of certain traits that are different? What type of person would be less likely to fall for the AUP? And how can we measure resistance towards the AUP? We did a study, where we explored the impact of two user traits *need-for-cognition (ncs)* and *geekism (gex)* on AUP resistance. To measure AUP resistance we observed users while they were doing drawing tasks. A behavioural coding system was used to derive an individual AUP resistance score. Basically, we counted behaviour associated with exploration and elaboration during the task and produced a single score. So, are users with high need-for-cognition and geekism more resistant to the AUP? We first look at the two predictors, separately:

As we will see later, it is preferable to build one model with two simultaneous predictors. For instructive purposes we begin with two separate LRMs, one for each predictor. Throughout the regression models we use z-transformed scores. Neither the personality nor the resistance scores truly have a metric interpretation, so nothing is lost in translation.

$$M1 : \mu_i = \beta_0 + \beta_{\text{ncs}} x_{\text{ncs}}$$
$$M2 : \mu_i = \beta_0 + \beta_{\text{gex}} x_{\text{gex}}$$

```
attach(AUP)
M_1 <-
  AUP_1 %>%
  stan_glm(zresistance ~ zncs, data = .)

M_2 <-
  AUP_1 %>%
  stan_glm(zresistance ~ zgex, data = .)
detach(AUP)
```

@ref(tab:AUP_coef) shows the two separate effects (`M_1` and `M_2`). Due to the z-transformation of predictors, the intercepts are practically zero. Both personality scores seem to have a weakly positive impact on AUP resistance.

Next, we estimate a model that regards both predictors simultaneously. For linear models, that requires nothing more than to make a sum of all involved predictor terms (and the intercept). The result is a `multiple regression model (MRM)`:

$$\mu_i = \beta_0 + \beta_{\text{ncs}} x_{\text{ncs}} + \beta_{\text{gex}} x_{\text{gex}}$$

In R's regression formula language, this is similarly straight-forward. The `+` operator directly corresponds with the `+` in the likelihood formula.

```
attach(AUP)
M_3 <-
  AUP_1 %>%
  stan_glm(zresistance ~ zncs + zgex, data = .) #<--

detach(AUP)
```

For the comparison of the three models we make use of a feature of the package bayr: the posterior distributions of arbitrary models can be combined into one multi-model posterior object, by just stacking them upon each other. The coefficient table of such a multi-model posterior gains an additional column that identifies the model:

```
attach(AUP)

P <-
  bind_rows(posterior(M_1),
            posterior(M_2),
            posterior(M_3))
```

```
T_coef_3 <- P %>%
  posterior() %>%
  coef()
T_coef_3
```

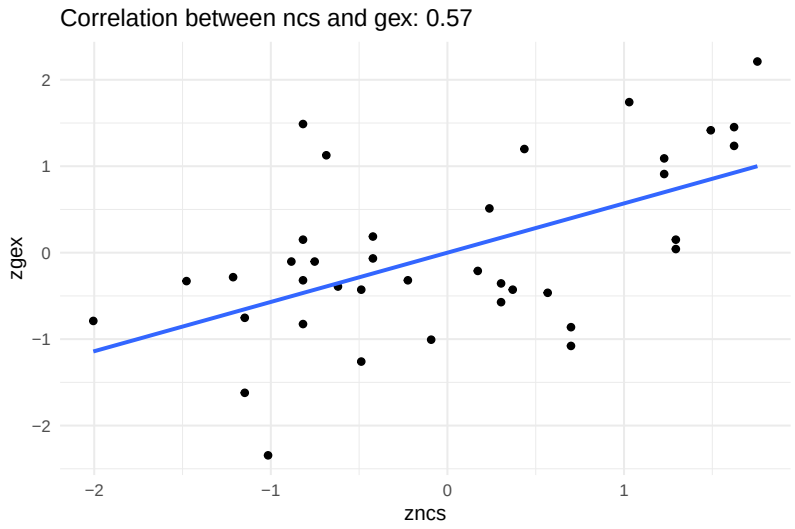
model	parameter	fixef	center	lower	upper
M_1	Intercept	Intercept	-0.003	-0.298	0.312
M_1	zncs	zncs	0.374	0.059	0.682
M_2	Intercept	Intercept	0.002	-0.312	0.315
M_2	zgex	zgex	0.294	-0.037	0.620
M_3	Intercept	Intercept	0.002	-0.307	0.302
M_3	zncs	zncs	0.299	-0.074	0.673
M_3	zgex	zgex	0.118	-0.267	0.500

The intercepts of all three models are practically zero, which is a consequence of the z-transformation. Recall, that the intercept in an LRM means that the predictor is zero. In MRM this is just the same: here, the intercept is the predicted AUP resistance score, for when NCS and GEX are both zero.

When using the two predictors simultaneously, the overall positive tendency remains. However, we observe major and minor shifts: in the MRM, the strength of the geekism score is reduced to less than half: $0.12[-0.27, 0.5]_{CI95}$. NCS has shifted, too, but lost only little of its original strength: $0.3[-0.07, 0.67]_{CI95}$.

For any researcher who has carefully conceived a research question this appears to be a disappointing outcome. The reason is that the two *predictors are correlated*. In this study, participants who are high on NCS also tend to have more pronounced geekism. @ref(AUP_corr_predictors) reveals the situation:

```
G_eda_4 <-
  AUP_1 %>%
  ggplot(aes(x = zncs, y = zgex)) +
  geom_point() +
  geom_smooth(method = "lm", se = F) +
  labs(title = str_c("Correlation between ncs and gex: ",
                     round(cor(AUP_1$zncs, AUP_1$zgex), 2)))

G_eda_4
```




```
detach(AUP)
```

Participants with a higher NCS also tend to score higher on geekism. Is that surprising? Actually, it is not. People high on NCS love to think. Computers are a good choice for them, because these are complicated devices that make you think. (Many users may even agree that computers help you think, for example when analyzing your data with R.) In turn, geekism is a positive attitude towards working with computers in sophisticated ways, which means such people are more resistant towards the AUP.

```
[NCS: love to think] --> [GEX: love computers] --> [resist AUP]
```

When such a causal chain can be established without doubt, some researchers speak of a *mediating variable* GEX. Although a bit outdated [%mediator_analysis], *mediator analysis is correct when the causal direction of the three variables is known.* Then, a so-called step-wise regression is performed to find the pure effects. A better alternative to that is structural equation modelling.

Unfortunately, in the situation here, the causal direction is partly ambiguous. We can exclude that the resistance test has influenced the personality scores, because of the order of appearance in the study. But, causally speaking, geekism may well preceed NCS. For example, computers reward you for thinking hard and, hence, you get used to it and make it your lifestyle. If you like thinking hard, then you probably also like the challenge that was given in the experiment.

```
[GEX: love computers] --> [NCS: love to think] --> [resist AUP]
```

In the current case, we can not distinguish between these two competing theories with this data alone. This is a central problem in empirical research. An example, routinely re-iterated in social science methods courses is the observation that people who are more intelligent tend to consume more fresh vegetables. Do carrots make us smart? Perhaps, but it is equally plausible that eating carrots is what smart people do. The basic issue is that a particular direction of causality can only be established, when all reverse directions can be excluded by logic. Behavioural science reseachers know of only two ways to do so:

1. By the *arrow of time*, it is excluded that a later event caused a preceding one. In the AUP study, there is no doubt that filling out the two personality questionnaires cause the behaviour in the computer task, because of the temporal order.
2. In *strictly controlled experiments*, participants are assigned to the conditions, randomly.

To come back to the AUP study: There is no way to establish a causal order of predictors NCS and Gex. If nothing is known but covariation, they just enter the model simultaneously, as in model `M_3`. This results in a redistribution of the overall covariance and the predictors are *mutually controlled*. In `M_2` the effect of GEX was promising at first, but now seems spurious in the simultaneous model. Most of the strength was just borrowed from NCS by covariation. The model suggests that loving-to-think has a considerably stronger association with AUP resistance than loving-computers.

That *may* suggest, but not prove, that geekism precedes NCS, as in a chain of causal effects, elements that are closer to the final outcome (AUP resistance) tend to exert more salient influence. But, without further theorizing and experimenting this is weak evidence of causal order.

If I would want to write a paper on geekism, NCS and the AUP, I might be tempted to report the two separate LRMs, that showed at least moderate effects. The reason why one should not do that is that separate analyses suggest that the predictors are independent. To illustrate this at an extreme example, think of a study where users were asked to rate their agreement with an interface by the following two questions, before ToT is recorded:

1. Is the interface beautiful?
2. Does the interface have an aesthetic appearance?

Initial separate analyses show strong effects for both predictors. Still, it would not make sense to give the report the title: “Beauty and aesthetics predict usability”. Beauty and aesthetics are practically synonyms. For Gex and NCS this may be not so clear, but we cannot exclude the possibility that they

are linked to a common factor, perhaps a third trait that makes people more explorative, no matter whether it be thoughts or computers.

So, what to do if two predictors correlate strongly? First, we always report just a single model. Per default, this is the model with both predictors simultaneously. The second possibility is to use a disciplined method of *model selection* and remove the predictor (or predictors) that does not actually contribute to prediction. The third possibility is, that the results with both predictors become more interesting when including conditional effects **??**

4.4.2 Crossover: multifactorial models

The very common situation in research is that multiple factors are of interest. In 4.3.5, we have seen how we can use an OGM to model a short learning sequence. What if I tell you now, that in this study, we have compared two designs against each other, and both were tested in three sessions. That makes 2 x 3 conditions. Here, I introduce a multi-factorial model, that has *main effects only*. Such a model actually is of very limited use for the IPump case, where we need *conditional effects* to get to a valid model.

We take as an example the BrowsingAB study: the primary research question regarded the design difference, but the careful researcher also recorded gender of participants. One can always just explore variables that one has. The following model estimates the gender effect alongside the design effect

```
attach(BrowsingAB)
```

```
M_mfm_1 <-
  BAB1 %>%
  stan_glm(ToT ~ 1 + Design + Gender, data = .)
```

```
coef(M_mfm_1)
```

parameter	fixef	center	lower	upper
Intercept	Intercept	203.397	191.8	215.0
DesignB	DesignB	-15.085	-28.3	-2.6
GenderM	GenderM	0.058	-12.5	12.4

By adding gender to the model, both effects are estimated simultaneously. In the following *multi-factorial model (MFM)* the intercept is a reference group, once again. Consider that both factors have two levels, forming a $2x2$ matrix.

```
tribble(~Condition, ~F, ~M,
        "A", "reference","difference",
        "B", "difference", "")
```

Condition	F	M
A	reference	difference
B	difference	

The first one, A-F, has been set as reference group. The intercept coefficient tells that women in condition A have an average ToT of $203.4[191.84, 214.97]_{CI95}$ seconds. The second coefficient says that design B is slightly faster and that there seemingly is no gender effect.

How comes that the model only has three parameters, when there are four groups? In a CGM, the number of parameters always equals the number of levels, why not here? We can think of the 2 x 2 conditions as flat four groups, A-F, A-M, B-F and B-M and we would expect four coefficients, say absolute group means. But with this model I thought of the two effects and how they change the default condition. In this model they do so, but without ever influencing each other. Design is assumed to have the *same effect* for men and women.

In many multi-factorial situations, one is better advised to use a model with conditional effects. Broadly, with conditional effect we can assess, how much effects influence each other. Basically, a model with conditional effects is a re-parametrization of *a multifactorial AMM* (MAMM), with the following dummy coding:

DesignA	DesignB	GenderF	GenderM
1	0	1	0
1	0	0	1
0	1	1	0
0	1	0	1

For your convenience, there also exists a an R formula to estimate an MAMM. This formula supresses the intercept and uses an interaction term without main effects (as will be explained in 4.5.2).

```
M_amfm_1 <- stan_glm(ToT ~ 0 + Design:Gender, data = BAB1, iter = 500)
```

```
coef(M_amfm_1)
```

parameter	fixef	center	lower	upper
DesignA:GenderF	DesignA:GenderF	202	189	215
DesignB:GenderF	DesignB:GenderF	188	174	202
DesignA:GenderM	DesignA:GenderM	204	192	216

parameter	fixef	center	lower	upper
DesignB:GenderM	DesignB:GenderM	187	176	199

The coefficient table carries the four group means and be can further processed as a *conditional plot*.

```
coef(M_amfm_1) %>%
  separate(parameter, into = c("Design", "Gender")) %>%
  ggplot(aes(x = Design, col = Gender, y = center)) +
  geom_point(size = 2) +
  geom_line(aes(group = Gender))
```

If the two effects were truly independent, these two lines had to be parallel, because the effect of Gender had to be constant. What this graph now suggests is that there is an interaction between the two effects. There is a tiny advantage for female users with design A, whereas men are faster with B with about the same difference. Because these two effects cancel each other out, the combined effect of Gender in model `M_mpm_1` was so close to zero.

```
detach(BrowsingAB)
```

4.4.3 Line-by-line: grouped regression models

Recall, that dummy variables make factors compatible with linear regression. We have seen how two metric preditors make a surface and how factors can be visualized by straight lines in a conditional plot. And that is precisely what happens when a factor is combined with a metric predictor: we get a group of lines, one per factor level. For example, we can estimate the effects age and design simultaneously:

```
attach(BrowsingAB)
```

```
M_grm_1 <-
  BAB1 %>%
  stan_glm(ToT ~ 1 + Design + age_shft, data = .)
```

```
coef(M_grm_1)
```

parameter	fixef	center	lower	upper
Intercept	Intercept	184.462	169.38	199.74
DesignB	DesignB	-15.088	-27.67	-2.20
age_shft	age_shft	0.634	0.24	1.03

Once again, we get an intercept first. Recall, that in LRM the intercept is the the performance of a 20-year old (age was shifted!). In the GCM it was the mean of the reference group. When marrying factors with continuous predictors, the *intercept is point zero in the reference group*. The predicted average performance of 20-year old with design A is $184.46[169.38, 199.74]_{CI95}$. The age effect has the usual meaning: by year of life, participants get $0.63[0.24, 1.03]_{CI95}$ seconds slower. The *factorial effect* B is a *vertical shift of the intercept*. A 20-year old in condition B is $15.09[27.67, 2.2]_{CI95}$ seconds faster.

It is important to that this is a model of parallel lines, implying that the age effect is the same everywhere. The following model estimates intercepts and slopes separately for every level, making it an *absolute mixed-predictor model (AMPM)*. The following formula produces such a model:

```
M_ampm_1 <-
  BAB1 %>% stan_glm(ToT ~ (0 + Design + Design:age_shft) , data = .)
```

```
coef(M_ampm_1)
```

parameter	fixef	center	lower	upper
DesignA	DesignA	194.363	176.157	213.059
DesignB	DesignB	156.895	137.843	175.350
DesignA:age_shft	DesignA:age_shft	0.299	-0.279	0.827
DesignB:age_shft	DesignB:age_shft	1.051	0.510	1.596

It turns out, the intercepts and slopes are very different as it can be. The first two coefficients represent the two Design intercepts: for a 20 year old, design B works much better, but at the same time design B puts a much stronger penalty on every year of age. With these coefficients we can also produce a conditional plot, with one line per Design condition.

```
coef(M_ampm_1) %>%
  select(fixef, center) %>%
  mutate(Design = str_extract(fixef, "[AB]"),
         Coef = if_else(str_detect(fixef, "age"), "Slope", "Intercept")) %>%
  select(Design, Coef, center) %>%
```

```
  spread(key = Coef, value = center) %>%
  print() %>%
  ggplot() +
  geom_abline(aes(color = Design, intercept = Intercept, slope = Slope)) +
  geom_point(data = BAB1, aes(x = age, col = Design, y = ToT))
```

```
## # A tibble: 2 x 3
##   Design Intercept Slope
##   <chr>      <dbl> <dbl>
## 1 A           194. 0.299
## 2 B           157. 1.05
```

Note

- how the coefficient table is first made flatter with `spread`, where Intercept and Slope become variables.
- that the Abline geometry is specialized on plotting linear graphs, but it requires its own aesthetic mapping (the global will not work).

So, if we can already fit a model with separate group means (an AMFM) or a bunch of straight lines (AMPM), why do we need a more elaborate account of conditional effects, as in chapter 4.5.2? The answer is, that conditional effects often carry important information, but are notoriously difficult to interpret. As it will turn out, conditional effects sometimes are due to rather trivial effects, such as saturation. But, like in this case, they can give the final clue. It is hard to deny that design features can work differently to different people. The hypothetical situation is BrowsingAB is that design B uses a smaller font-size, which makes it harder to read with elderly users, whereas younger users have a benefit from more compactly written text.

And, sometimes, experimental hypotheses are even formulated as conditional effects, like the following: some control tasks involve long episodes of vigilance, where mind wandering can interrupt attention on the task. If this is so, we could expect people who meditate to perform better at a long duration task, but showing no difference at short tasks. In a very simple experiment participants reaction time could be measured in a long and short task condition.

```
detach(BrowsingAB)
```

4.4.4 Empirical versus statistical control

Fundamental researchers have a knack for the experimental method. An *experiment*, strictly, is a study where you measure the effects of variables you

manipulate. Manipulation is, almost literally, that it is *in your hands*, who receives the treatment. The fantastic thing about manipulation is that it allows for *causal conclusions*. A *strictly controlled experiment* is when all influencing variables are either manipulated or kept constant. That is an ideal and would not even be the case if you test the same person over-and-over again (like researchers in psychophysics often do). You never jump into the same river twice.

Sometimes an influencing variable lends itself to be kept constant. For example, in cognitive psychological experiments environment and equipment is usually kept constant. For applied research, keeping things constant comes at a major disadvantage: it limits the possible conclusions drawn from the study. Imagine, you tested a smartphone app with participants, all students, comfortably sitting in a quiet environment. Would you dare to make conclusions on how any users perform in real life situations, say while driving a car? When keeping things constant, *ecological validity* and *generalizability* suffer.

In most applied design studies we need ecological validity and generalizability. If performance differs under certain conditions, you certainly want to know that. The solution is to *let conditions vary and record them* as variables, as good as possible. For example, if you were to compare two voice-controlled intelligent agent apps, you could manipulate the ambient noise level, if you are in the lab.

In practically all applied studies, variables may exist which you cannot manipulate. Especially, user traits are impossible to manipulate; If someone has an extrovert character or did a lot of gaming in the past, you cannot change that. Diversity of users is a fact and people come as they are.

Field studies usually aim for high ecological validity. Participants are supposed to use the system in the situations they encounter. If a smartphone app is being used sitting, walking, driving or at a secret place, it is crucial to observe all situations. Consider a car navigation system that is tested in a long, lonely highway situation only. How much would the results tell your for performance in dense city traffic? Design researchers frequently need results that are highly representative for various users and situations of use.

Fundamental lab researchers are afraid of individual differences, too. The reasons are different, though: all non-manipulated influencing factors add noise to the study, which makes it harder to find the effects of interest. While lab researchers do there best to keep the environment constant, they cannot keep all participant traits constant. Lab researchers have two solutions to the problem: matching and randomized control.

With *pair matching*, potentially relevant participant traits are recorded upfront; then participants are assigned to conditions such that groups have about the same composition. For example, one makes sure that the age distribution is about the same and both genders are equally represented. When all other

influencing variables are constant between groups, the lab researcher can be sure that the effect is unambiguously caused by the manipulation. So they say and routinely record participants age, gender and nationality.

However, there are better alternatives: the best pair match is the person herself. Experimental studies that expose the same person to several conditions are called *within-subject*. In the special case that all participants encounter all conditions, the variable is *complete within-subject*. In the following chapter, we use mixed-effects models 5 to deal with within-subject designs, gracefully.

In design research, pair matching applies for situations where designs are compared. In the simple situation that a design is evaluated against a set standard (e.g. 111 seconds to rent a car), it is more important to do *population matching*. The sample of participants is drawn to be *representative for the target population*. Representativeness comes in two levels: *coverage representation* is reached when all influencing properties have occurred a few times during observation. So, if your target population contains several subgroups, such as age groups, experience or people with different goals, they should all be covered to some extent. *Proportional representation* means all user and situational properties are covered *and* they have about the same proportion in the sample as in the population.

You can only match what you can measure and you only measure what you expect. Human behaviour in everyday life is influenced by many factors in complex ways. Although a plethora of personality inventories exists, doing them all prior to the real study is impossible. It would probably not even be effective. Never have I seen a design research study, where even the most established personality tests explain more than a few percent of variation. As another example, take the primacy effect: what you experienced first, has the strongest influence. In real life, impressions are constantly pouring on people and you will never be able to record and match that to a reasonable extent.

When influencing variables cannot be measured for matching or statistical control, the last resort is *randomized control*. This is a misleading term, insofar as what the researcher actually does is to *let go to chance*. Indeed, if the process of drawing participants and assigning them to manipulations is completely left to chance, then *in the long-term*, the sample will be proportional representative and all groups will have the same composition of traits. *Randomization* works well with larger samples. With small samples, it can still easily happen that one ends up with more or less biased samples or heterogeneous groups. Just by chance, more higher-educated people could have ended up in condition A of BrowsingAB.

Using manipulation, matching or randomization in in-the-wild research may work in some cases. In other cases, it will be ineffective or impractical. The ultimate problem is the attempt to keep things constant. In applied design research the questions rarely come down to a “Is A better than B?”. If there is an age effect, you may certainly want to know it and see how the design effect

compares to it. But, you can only examine what is varied and recorded. The approach of *statistical control* is to record (instead of manipulate) all variables that may influence the results and add them to the statistical model. As we have seen in this section now, the linear model puts no limits on the number of predictors. That allows us to use control variables and evaluate multiple research questions in a single model. And this is also how it should be done.

In the next section we will take multi-predictor models to a new level. As we have seen, multiple effects can be conditional upon each other and I gave you a straight-forward way to check this with AMFMs and AMPMs. What you could not do with these models is interpret how strong the conditional effect is. In the following section, I will elaborate on what conditional effects can mean and how these can be quantified as differences, using treatment contrasts.

4.5 Conditional effects models

With the framework of MPM, we can use an arbitrary number of predictors. These can represent properties on different levels, for example, two design proposals for a website can differ in font size, or participants differ in age. So, with MPM we gain much greater flexibility in handling data from applied design research, which allows us to examine user-design interactions more closely.

The catch is that if you would ask an arbitrary design researcher:

> Do you think that all users are equal? Or, could it be that one design is better for some users, but inferior for others?

you would in most cases get the answer:

> Of course users differ in many ways and it is crucial to know your target group.

Some will also refer to the concept of usability by the ISO 9241-11, which contains the famous four words:

> "... for a specified user ..."

The definition explicitly requires you to state for *for whom* you intended to design. It thereby implicitly acknowledges that usability of a design could be very different for another user group. In other words, statements on usability are by the ISO 9241-11 definition *conditional* on the target user group.

In statistical terms, conditional statements have this form:

the effect of design *depends on* who the user is.

In regression models, conditional statements like these are represented by *conditional effects*. Interactions between user properties and designs are central in design research, and deserve a neologism: *differential design effects models (DDM)*. Sometimes, conditional effects are needed for a less interesting reason: *saturation* occurs when physical (or other) boundaries are reached and the steps are getting smaller, for example, the more you train, the less effect it seems to have. Saturations counter part is *amplification*, a rare one, because is like compound glue: it will harden only if the two components are present. And finally, we will look at an experimental example, where a conditional effect can be linked to a more generic truth.

4.5.1 Conditional multiple regression

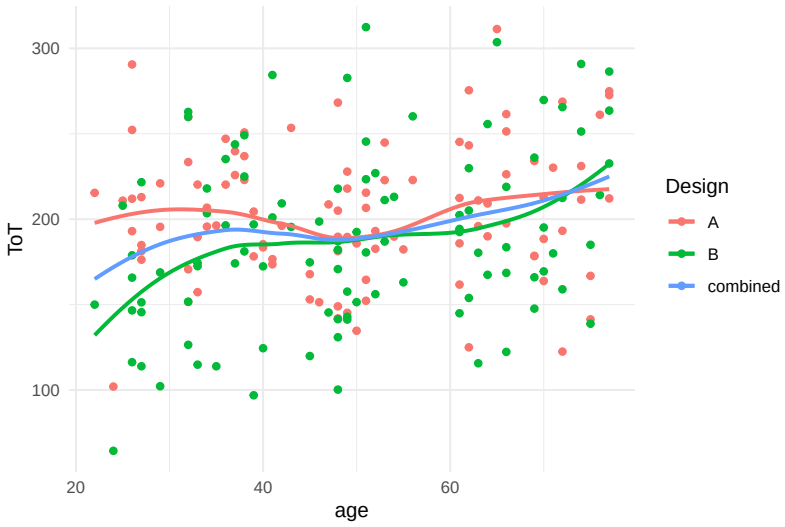
In section 4.2 we have seen how the relationship between predictor and outcome variable can be modelled as a linear term. We analysed the relationship between age and ToT in the (fictional) BrowsingAB case and over both designs combined and observed just a faint decline in performance, which also seemed to take a wavey form.

It is commonly held that older people tend to have lower performance than younger users. A number of factors are called responsible, such as: slower processing speed, lower working memory capacity, lower motor speed and visual problems. All these capabilities interact with properties of designs, such as legibility, visual simplicity and how well the interaction design is mapped to a user's task. It is not such a stretch to assume that designs can differ in how much performance degrades with age.

In turn, a design can also comtain compromises that limit the performance of younger users. For example, assume that main difference between design A and B in the BrowsingAB example is that A uses larger letters than B. Would that create the same benefit for everybody? It is not unlikely, that larger letters really only matter for users that have issues with farsightedness, which is associated with age. Maybe, there is even an adverse effect for younger users, as larger font size takes up more space on screen and more scrolling is required. We take a first look at the situation:

```
attach(BrowsingAB)

BAB1 %>%
  ggplot(aes(x = age,
             col = Design,
             y = ToT)) +
  geom_point() +
```

```
  geom_smooth(se = F) +
  geom_smooth(se = F, aes(col = "combined"))
```



The graph suggests that designs A and B differ in the effect of age. Design B appears to perform much better with younger users. At the same time, it seems as if A could be nmore favorable for users at a high age. By adding the conditional effect `Design:age_shft` the following model estimates the linear relationship for the designs separately. This is essentially the same model as the absolute mixed-predictor model M_ampm_1 4.4, which also had four coefficients, the intercepts and slopes of two straight lines. We have already seen how the GRM and the AMPM produce different fitted responses. Predictions are independent of contrast coding, but coefficients are not. The following conditional model uses treatment contrasts, like the GRM, and we can compare the coefficients side-by-side.

```
M_cmrm <-
  BAB1 %>%
  stan_glm(ToT ~ Design + age_shft + Design:age_shft,
                 data = .)

# T_resid <- mutate(T_resid, M_cmrm = residuals(M_cmrm))
```

```
P_comb <-
  bind_rows(
    posterior(M_grm_1),
    posterior(M_cmrm)
  )
```

```
coef(P_comb)
```

model	parameter	fixef	center	lower	upper
M_cmrm	Intercept	Intercept	195.458	176.933	214.332
M_cmrm	DesignB	DesignB	-37.781	-63.045	-12.494
M_cmrm	age_shft	age_shft	0.250	-0.300	0.831
M_cmrm	DesignB:age_shft	DesignB:age_shft	0.771	0.008	1.547
M_grm_1	Intercept	Intercept	184.462	169.383	199.742
M_grm_1	DesignB	DesignB	-15.088	-27.671	-2.197
M_grm_1	age_shft	age_shft	0.634	0.240	1.034

The conditional model shares the first three coefficients with the unconditional model, but only the first two, Intercept and DesignB have the same meaning. and we regard them first. The intercept is the performance of an average twenty-year-old using design A, but the two models diverge in where to place this and the conditional model is less in favor of design A ($195.46[176.93, 214.33]_{CI95}$ seconds). Conversely, the effect of design B at age of 20 improved dramatically: accordingly, a twenty-year-old is $37.78[63.04, 12.49]_{CI95}$ faster with B.

The third coefficient Age_shift appears in both models, but really means something different. The GRM assumes that both designs have the same slope of 0.63 seconds per year. The conditional model produces one slope per design and here the coefficient refers to design A only, as this is the reference group. Due to the treatment effects, `DesignB:age_shft` is the *difference in slopes*: users loose 0.63 seconds per year with A, and on top of that $0.25[-0.3, 0.83]_{CI95}$ with design B.

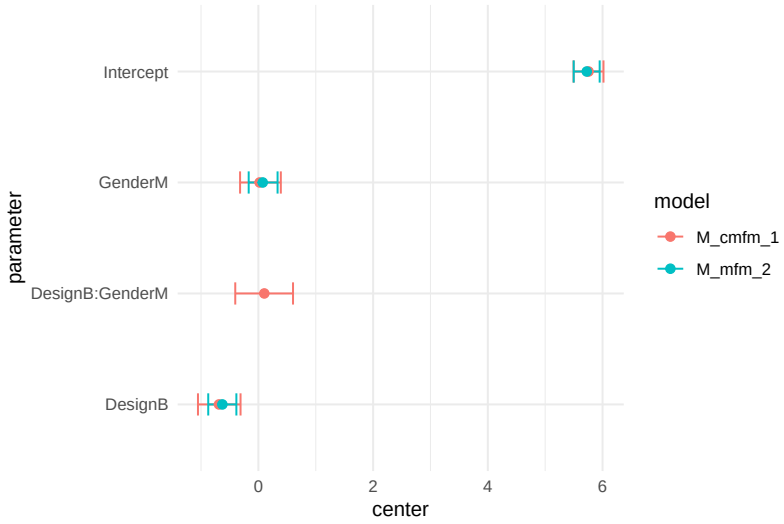
```
detach(BrowsingAB)
```

4.5.2 Conditional multifactorial models

In a conditional multifactorial model (CMFM), the treatment effect depends on another factor. When the second factor changes in level, this influences the coefficients. Because of that a CMFM is more flexible. Actually, a full CMFM has as many coefficients as there are multi-level groups and is flexible enough that all group means can be completely independent, just like an AMM does it. Let us see this on an almost trivial example, first. In the fictional BrowsingAB case, a variable `rating` has been gathered. Let us imagine this is a vague emotional rating in the spirit of user experience. Some claim that emotional experience is what makes the sexes different, so one could ask whether this makes a difference for the comparison two designs A and B.

```
attach(BrowsingAB)
```

```
BAB1 %>%
  ggplot(aes(y = rating, x = Gender, color = Design)) +
  geom_boxplot()
```



In a first exploratory plot it looks like the ratings are pretty consistent across gender, but with a sensitive topic like that, we better run a model, or rather two, a plain MFM and a conditional MFM:

```
M_mfm_2 <-
  BAB1 %>%
  stan_glm(rating ~ Design + Gender,
                  data = .)

M_cmfm_1 <-
  BAB1 %>%
  stan_glm(rating ~ Design + Gender + Design:Gender,
                  data = .)

# T_resid <- mutate(T_resid, M_ia2 = residuals(M_ia2))
```

```
T_ratings <-
  bind_rows(
    posterior(M_mfm_2),
    posterior(M_cmfm_1)) %>%
  coef()
```

```
T_ratings %>%
  ggplot(aes(y = parameter, col = model,
             xmin = lower, xmax = upper, x = center)) +
  geom_errorbarh(height = .2) +
  geom_point(size = 2)
```

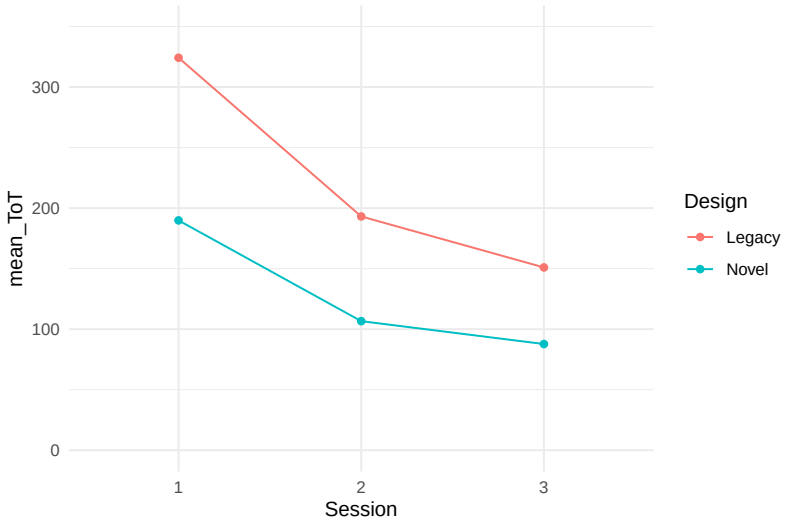
The CLU plots above show both models in comparison. Both models use treatment effects and put the intercept on female users with design A. We observe that there is barely a difference in the estimated intercepts. The coefficient DesignB means something different in the models: in the MFM it represents *the* difference between Designs. In the CMFM, it is the difference by design *with female users*. The same is true for GenderM, which is a general effect in MFM and a local effect in the CMFM, the gender difference with design A. For that reason, it is *not useful* to speak of these coefficients as *main effects*. They are main effects in a plain MFM, but once the effects become conditional, there is nothing such as a main effect any more. At least, this is the case for treatment effect coding and stairways coding (as we will see next).

All three coefficients of the MFM barely change by introducing the conditional effect `DesignB:GenderM`. Recall that in the MFM, the group mean of design B among men is calculated by adding the two main effects to the intercept. This group mean is fixed. The conditional effect `DesignB:GenderM` is the difference to the fixed group in the MFM. It can be imagined as an adjustment parameter, that gives the fourth group its own degree of freedom. In the current CMFM the conditional coefficient is very close to zero, with a difference of just $0.1[-0.4, 0.6]_{CI95}$.

It seems we are getting into a lot of null results here. If you have a background in classic statistics, you may get nervous at such a point, because you remember that in case of null results someone said: "one cannot say anything". This is true when you are testing null hypothesis significance testing. But, when you interpret coefficients, you are speaking quantities and zero is a quantity. What the MFM tells us is that male users really don't give any higher or lower ratings, *in total*, although there remains some uncertainty.

Actually, the purpose of estimating a CMFM can just be to show that some effect is unconditional. As we have seen earlier 4.4.2, conditional effects can cancel each other out. Take a look at the following hypothetical results of the study. Here, male and female users do not agree. If we would run an MFM in such a situation, we would get very similar coefficients, but would overlook that the relationship between design and rating is just poorly rendered.

```
tribble(~Design,     ~Gender, ~mean_rating,
        "A", "F",     5.6,
        "A", "M",     5.6 + .4,
        "A", "Total", mean(c(5.6, 6.0)),
        "B", "F",     5.6 - .3,
        "B", "M",     5.6 + .4 -.3 -.6,
        "B", "Total", mean(c(5.3, 5.1))) %>%
  ggplot(aes(x = Design, col = Gender, y = mean_rating)) +
  geom_point(size = 2) +
  geom_line(aes(group = Gender))
```



If something like this happens in a real design study, it may be a good idea to find out, why this difference appears and whether there is a way to make everyone equally happy. These are questions a model cannot answer. But a CMFM can show, when effects are conditional and when they are not. Much of the time, gender effects is what you rather don't want to have, as it can become a political problem. If conditional adjustment effects are close to zero, that is proof (under uncertainty) that an effect is unconditional. If that is the case, modelling it as a true main effect in a plain MFM is justified, and one is out of the trouble.

Let's see a more complex example of conditional MFMs, where conditional effects are really needed. In the IPump study, two infusion pump designs were compared in three successive sessions. In 4.3.5 we saw how a factorial model can render a learning curve using stairway dummies. With two designs, we can estimate separate learning curves and make comparisons. Let's take a look at the raw data:

```
attach(IPump)
```

```
D_agg %>%
  group_by(Design, Session) %>%
  summarize(mean_ToT = mean(ToT)) %>%
  ggplot(aes(x = Session, y = mean_ToT, color = Design)) +
  geom_point() +
  geom_line(aes(group = Design)) +
  ylim(0,350)
```

We note that the learning curves do not cross, but are not parallel either, which means the stairway coefficients will be different. We need a conditional model.

The first choice to make is between treatment dummies and stairway dummies and both have their applications. With treatment effects, we would get an estimate for the total learning between session 1 and 3. That does not make much sense here, but could be interesting to compare trainings by the total effect of a training sequence.

We'll keep the stairway effects on the sessions, but have to now make a choice on where to fix the intercept, and that depends on what aspect of learning is more important. If this were any walk-up-and-use device or a website for making your annual tax report, higher initial performance would indicate that the system is intuitive to use. Medical infusion pumps are used routinely by trained staff. What matters here is long-term performance, and the final session is the best estimate we have for that. We create stairway dummies for session and make this conditional on Design:

```
T_dummy <-
  tribble(~Session, ~Session3, ~Step3_2, ~Step2_1,
          "1", 1, 1, 1,
          "2", 1, 1, 0,
          "3", 1, 0, 0)

D_agg <-
  left_join(D_agg,
            T_dummy,
            by = "Session")


M_cmfm_2 <-
  stan_glm(ToT ~ 1 + Design + Step3_2 + Step2_1 +
             Design:(Step3_2 + Step2_1), data = D_agg)
```

```
coef(M_cmfm_2)
```

parameter	fixef	center	lower	upper
Intercept	Intercept	150.9	121.902	181.8
DesignNovel	DesignNovel	-63.1	-106.357	-20.8
Step3_2	Step3_2	42.0	0.095	85.6
Step2_1	Step2_1	130.2	86.750	172.8
DesignNovel:Step3_2	DesignNovel:Step3_2	-23.2	-82.801	36.7
DesignNovel:Step2_1	DesignNovel:Step2_1	-48.2	-107.636	12.4

```
detach(IPump)
```

Note that ...

- here I demonstrate a different technique to attach dummies to the data. First a coding table `T_dummy` is created, which is then combined with the data, using a (tidy) *join* operation.
- we have expanded the factor Session into three dummy variables and we have to make every single one conditional. `Design:(Step3_2 + Step2_1)` is short for `Design:Step3_2 + Design:Step2_1`. But, you should *never* use the fully factorial expansion (`Factor1 * Factor2`), a this would make dummy variables conditional.

In conditional learning curve model, the intercept coefficient tells us that the average ToT with the Legacy in the final session is $150.9[121.9, 181.75]_{CI95}$ seconds. Using the Novel design the nurses were $63.09[106.36, 20.84]_{CI95}$ seconds faster and that is our best estimate for the long-term improvement in efficiency.

Again, the learning step coefficients are not "main" effects, but is local to Legacy. The first step `Step2_1` is much larger than the second, as is typical for learning curves. The adjustment coefficients for Novel have the opposite direction, meaning that the learning steps in Novel are smaller. That is not as bad as it sounds, for two reasons: first, in this study, the final performance counts, not the training progress. Second, an more generally, we have misused a linear model to smooth a non-linear model. Learning processes are exponential. $\beta_0 - \beta_1 x_{1i}$ is a linear term. But, when we put it into an exponent, like $\exp(\beta_0 - \beta_1 x_{1i})$ this is the same as the *quotient* $\exp(\beta_0)/\beta_1 x_{1i}$. Linear models and "linear-in-exponent" models differ in one more property: linear models can be negative or positive, depending on which number is larger. But the linear-in-exponent model will always stay in the positive range.

Learning curves are saturation processes, which can look linear when viewed in segments, but unlike linear models, they never cross the lower boundary.

This is simply, because there is a maximum performance limit, which can only be reached asymptotically. In teh following section, I will argue that basically all measures we take have natural boundaries. Under common circumstances, this can lead to conditional effects which are due to saturation. In chapter 6, we will pick up again the idea of putting the linear term into the exponent. This is what some Generalized Linear Models do to avoid crossing natural boundaries of measures.

4.5.3 Hitting the boundaries of saturation

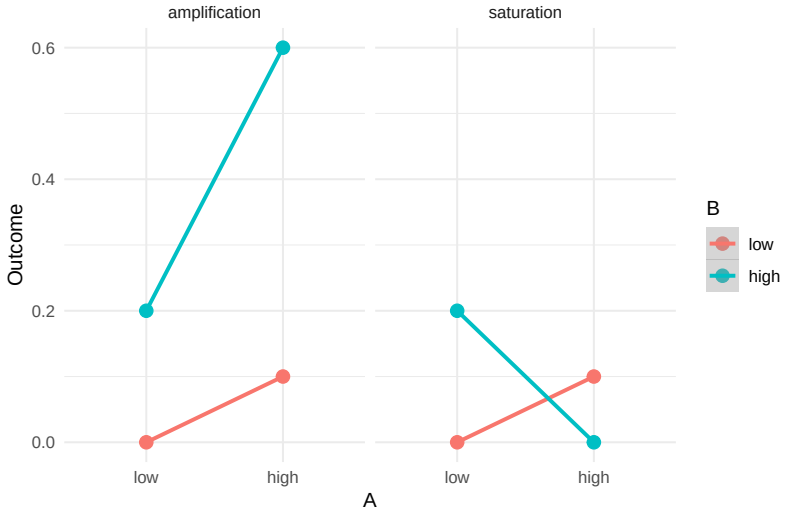
Most statistically trained researchers are aware of some common assumptions of linear regression, such as the normally distributed residuals and variance homogeneity. Less commonly regarded is the assumption of linearity, which arises from the basic regression formula:

$$y_i = \beta_0 + \beta_1 x_{1i}$$

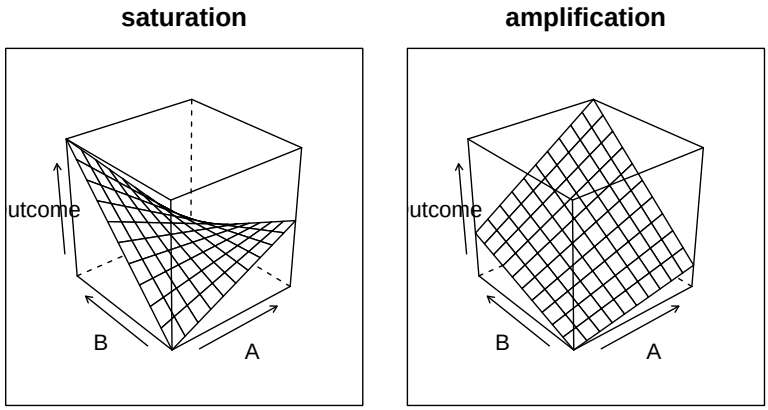
The formula basically says, that if we increase x_1 (or any other influencing variable) by one unit, y will increase by β_1. It also says that y is composed as a mere sum. In this section, we will discover that these innocent assumptions do not hold.

In this and the next section, we will use conditional effects to account for non-linearity. We can distinguish between *saturation effects*, which are more common and *amplification effects*.

```
    mascutils::expand_grid(effect = c("saturation", "amplification"),
                           A = c(0,1),
                           B = c(0,1)) %>%
        left_join(tibble(effect = c("saturation","amplification"),
                                      beta_1 = c(.1,.1),
                                      beta_2 = c(.2,.2),
                                      beta_3 = c(-0.3, 0.3)))  %>%
        mutate(Outcome = A * beta_1 + B * beta_2 + A * B * beta_3) %>%
        mutate(B = factor(B, labels = c("low", "high")),
                    A = factor(A, labels = c("low", "high"))) %>%
        ggplot(aes(x = A, col = B, y = Outcome)) +
        geom_point(size = 3) +
        geom_smooth(aes(group = B, col= B), method = "lm") +
        facet_grid(.~effect)
```




```
Interactions <- expand.grid(effect = c("saturation", "amplification"),
                            A = seq(0,1, length.out = 11),
                            B = seq(0,1, length.out = 11)) %>%
    left_join(tibble(effect = c("saturation","amplification"),
                                    beta_1 = c(.1,.1),
                                    beta_2 = c(.2,.2),
                                    beta_3 = c(-0.3, 0.3)))  %>%
    mutate(Outcome = A * beta_1 + B * beta_2 + A * B * beta_3)

library(lattice)
grid.arrange(
    wireframe(Outcome ~ A + B,
                            data = filter(Interactions, effect == "saturation"),
                            main = "saturation"),
    wireframe(Outcome ~ A + B,
                            data = filter(Interactions, effect == "amplification"),
                            main = "amplification"),
    ncol = 2
)
```



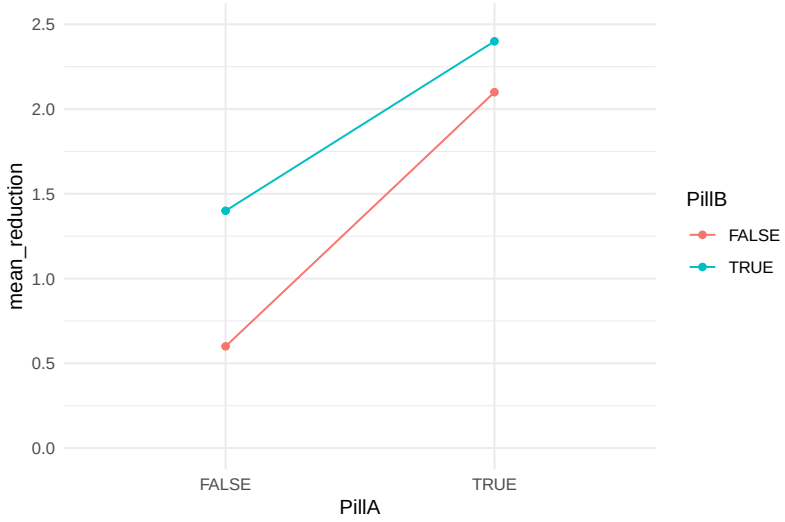
A major flaw with the linear model is that it presumes the regression line to rise or fall infinitely. However, *in an endless universe everything has boundaries*. Just think about your performance in reading this text. Several things could be donme to improve reading performance, such as larger font size, simpler sentence structure or translation into your native language. Still, there is a hard lower limit for time to read, just by the fact, that reading involves saccades (eye movements) and these cannot be accelerated any further. The time someone needs to read a text is limited by fundamental cognitive processing speed. We may be able to reduce the inconvenience of deciphering small text, but once an optimum is reached, there is no further improvement. Such boundaries of performance inevitably lead to non-linear relationships between predictors and outcome.

Modern statistics knows several means to deal with non-linearity, some of them are introduced in 6). Still, most researchers use linear models, and it often can be regarded a reasonable approximation under particular circumstances. Mostly, this is that measures keep a distance to the hard boundaries. Because if performance is pushed to the limits, *saturation* occurs. When there is just one treatment repeatedly pushing towards a boundary, we get the diminishing returns effect seen in learning curves 4.3.5. If two or more variables are pushing simultaneously, saturation appear as conditional effects.

Before we turn to a genuine design research case, let me explain saturation effects by an example that I hope is intuitive. The hypothetical question is: do two headache pills have twice the effect of one? Consider a pharmaceutical study on the effectiveness of two pain killer pills A and B, taking place in the aftermath of a huge party on a university campus. Random strolling students are asked to participate. First, they rate their experienced headache

on a Likert scale ranging from "fresh like the kiss of morning dew" to "dead highway opossum". Participants are randomly assigned to four groups, each group getting a different combination of pills: no pill, only A, only B, A and B. After 30 minutes, headache is measured again and the difference between both measures is taken as the outcome measure: headache reduction. We inspect the position of four group means graphically:

```
attach(Headache)

T_means <-
  Pills %>%
  group_by(PillA, PillB) %>%
  summarise(mean_reduction = round(mean(reduction),1))

T_means %>%
  ggplot(aes(x = PillA, col = PillB, mean_reduction)) +
    geom_point() +
  geom_line(aes(group = PillB)) +
  ylim(0,2.5)
```



When neither pill is given a slight spontaneous reduction seems to occur, which is the placebo-effect. Both pills alone are much stronger than te placebo and giving them both has the most beneficial effect. However, the combined effect is just a tad stronger than the effect of A and stays far from being the sum. One could also say, that the net effect of B is weaker when A has been given first. When the effect of one predictor depends on the level of another, this is just a conditional effect.

Does that make sense? Do the effects of headache pills simply add up like this? This question is easily answered by contemplating what may happen with not two but five headache pills. If we assume linear addition of effects, participants in the group with all pills would even experience the breathtaking sensation of *negative* headache. So, certainly, the effect cannot be truly linear. Al headache pills are pushing into the direction of the boundary called no-headache.

At the example of headache pills, I will now demonstrate that saturation can cause a severe bias when not accounted for by a conditional effect. We estimate both models: a factorial unconditional MFM and a conditional MFM.

```
M_mfm <- stan_glm(reduction ~ 1 + PillA + PillB, data = Pills)
M_cmfm <- stan_glm(reduction ~ 1 + PillA + PillB + PillA:PillB, data = Pills)
# M_3 <- stan_glm(reduction ~ 0 + PillA:PillB, data = Pills, iter = 100)

P_1 <- bind_rows(
  posterior(M_mfm),
  posterior(M_cmfm) #,
  #posterior(M_3)
)
```

The following table puts the center estimates of both models side-by-side.

```
coef(P_1) %>%
  select(model, fixef, center) %>%
  spread(key = model, value = center)
```

fixef	M_1
Intercept	106

```
# %>%  arrange(c(1,2,4,3))
```

Both intercepts indicate that headache diminishes due to the placebo alone, but `M_mfm` over-estimates the placebo effect. At the same time, the treatment effects PillA and PillB are under-estimated. That happens, because the unconditional model averages over two conditions, under which pill A or B are given: with the other pill or without. As `M_cmfm` tells, when taken with the another pill, effectiveness is reduced by -0.37. The effectiveness of two pills is not their sum, but less than that. One can have headache to a certain degree or no headache at all. If it's gone, any more pills have no additional effects.

```
detach(Headache)
```

In general, if two predictors work into the same direction (here the positive direction) and the intercation effect has the opposite direction, this is likely a

saturation effect: the more of similar is given, the closer it gets to the natural boundaries and the less it adds. Remember that this is really not about side effects in conjunction with other medicines. Quite the opposite: if two type of pills effectively reduce headache, but in conjunction produce a rash, this would be an amplification effect. Amplification effects are theoretically interesting, not only for pharmacists. Saturation effects are boring. When they happen, they only tell us that we have been applying *more of the similar* and that we are running against a set limit of how much we can improve things.

Back to design research with a another hypothetical study that works similar to the Pills case. Imagine a study aiming at ergonomics of reading for informational websites. In a first experiment, the researcher found that 12pt font effectively reduces reading time as compared to 10pt by about 5 seconds.

```
D_reading_time <-
  tibble(font_size = c(4, 10, 12, 14, 16, 18),
           observed_time = c(NA, 40, 30, NA, NA, NA),
           predicted_time = 60 - font_size/4 * 10)

D_reading_time
```

font_size	observed_time	predicted_time
4	NA	50
10	40	35
12	30	30
14	NA	25
16	NA	20
18	NA	15

It should be clear by now, that these expectations have no ground: for normally sighted persons, a font size of 12 is easy enough to decipher and another increase will not have the same effect. Taking this further, one would even arrive at absurdly short or impossible negative reading times. At the opposite, a font size of four point may just render unreadable on a computer screen. Instead of a moderate increase by 10 seconds, participants may have to decipher and guess the individual words, which will take much longer.

Researching the effect of font sizes between 8pt and 12pt font size probably keeps the right distance, with approximate linearity within that range. But what happens if you bring a second manipulation into the game with a functionally similar effect? A likely outcome is that the fundamental assumption of *predictors sum up* no longer holds, but *saturation* occurs.

Still, another option to improve readability of text is to improve the contrast, using black-on-white fonts, instead of the more fanbcy looking grey-on-white.

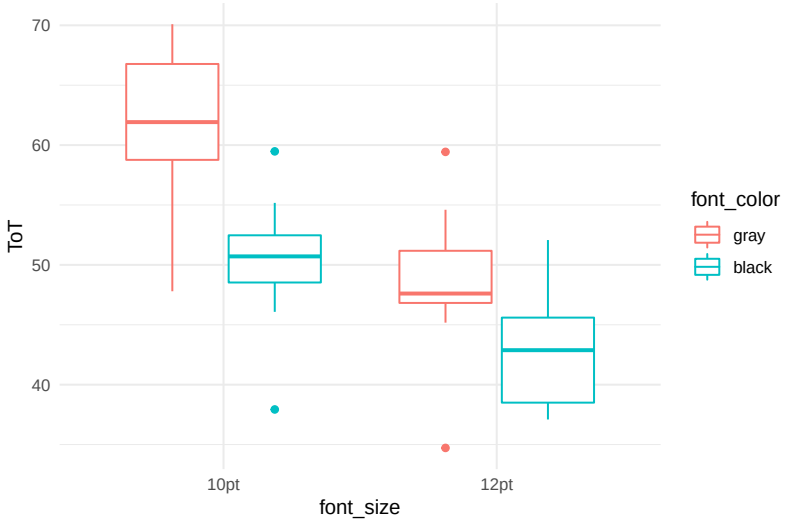
Let's turn to a fictional, yet realistic problem in design research. Design of systems is a matter of compromises. A common conflict of interests is between

the aesthetic appearance and the ergonomic properties. From an ergonomic point of view, one would probably favor a typesetting design with crisp fonts and maximum contrast. However, if a design researcher were now to suggest using 12pt black Arial on white background as body font, this is asking for trouble. Someone in charge will probably insist on a fancy serif font in an understating blueish-grey tone. For creating a relaxed reading experience, the only option left is to increase the font size.

The general question arises: can one sufficiently compensate lack of contrast by setting the text in the maximum reasonable font size 12pt, as compared to the more typical 10pt? In the fictional study Reading, this is examined in a 2x2 experimental design: the same page of text is presented in four versions, with either 10pt or 12pt, and grey versus black font colour. Performance is here measured as time-on-task of reading the full page.

```
attach(Reading)

D_1
```

```
D_1 %>%
  ggplot(aes(col = font_color,
             x = font_size,
             y = ToT)) +
  geom_boxplot()
```



We see immediately, that both design choices have an impact: black letters, as well as larger letters are faster to read. But, do they add up? Or do both factors behave like headache pills, where more is more, but less than the sum.

Clearly, the 12pt-black group could read fastest on average. Neither with large font, nor with optimal contrast alone has the design reached a boundary, i.e. saturation. We run two regression models, a plain MFM and a conditional MFM, that adds an interaction term. We extract the coefficients from both models and view them side-by-side:

```
M_mfm <- D_1 %>%
  stan_glm(ToT ~ 1 + font_size + font_color, data = .)

M_cmfm <- D_1 %>%
  stan_glm(ToT ~ 1 + font_size + font_color + font_size : font_color, data = .)
```

```
T_read_fixef <-
  bind_rows(posterior(M_mfm),
            posterior(M_cmfm)) %>%
  coef()
T_read_fixef
```

modelparameter	fixef	center	lower	upper
M_cmfmIntercept	Intercept	61.20	57.40	65.00
M_cmfmfont_size12pt	font_size12pt	-12.59	-17.80	-6.98
M_cmfmfont_colorblack	font_colorblack	-10.89	-16.35	-5.56
M_cmfmfont_size12pt:font_colorblack	font_size12pt:font_colorblack	5.32	-2.33	12.89
M_mfmIntercept	Intercept	59.84	56.29	63.40
M_mfmfont_size12pt	font_size12pt	-9.89	-13.92	-5.93
M_mfmfont_colorblack	font_colorblack	-8.22	-12.15	-4.23

The estimates confirm, that both manipulations have a considerable effect in reducing reading time. But, as the conditional effect works in the opposite direction, this reeks of saturation: in this hypothetical case, font size act as contrast are more-of-the-similar and the combined effects is less than the sum. This is just like taking two headache pills.

If this was real data, we could assign the saturation effect a deeper meaning. It is not always obvious that two factors work in a similar way. From a psychological perspective this would indicate that both manipulations work on similar cognitive processes, for example, visual letter recognition. Knowing more than one way to improve on a certain mode of processing can be very helpful in design, where conflicting demands arise often and In the current

case, a reasonable compromise between ergonomics and aesthetics would be to either use large fonts or black letters. Both have the strongest ergonomic net effect when they come alone.

Conditional effects are notoriously neglected in research and they are often hard to grasp for audience, even when people have a classic statistics education. Clear communication is often crucial and conditional models are best understood by using conditional plots. A conditional plot for the 2x2 design contains the four estimated group means. These can be computed from the linear model coefficients, but often it easier to just estimate an AGM alongsite the CGM:

```
M_amm <-
  D_1 %>%
  stan_glm(ToT ~ 0 + font_size : font_color,
           data = .)
```

```
coef(M_amm)
```

parameter	fixef	center	lower	upper
font_size10pt:font_colorgray	font_size10pt:font_colorgray	60.8	56.8	64.8
font_size12pt:font_colorgray	font_size12pt:font_colorgray	48.2	44.2	51.9
font_size10pt:font_colorblack	font_size10pt:font_colorblack	49.8	45.8	53.5
font_size12pt:font_colorblack	font_size12pt:font_colorblack	42.7	38.8	46.5

```
T_amm <-
  coef(M_amm) %>%
  separate(fixef, c("font_size", "font_color"), sep = ":") %>%
  mutate(font_size = str_replace(font_size, "font_size", ""),
         font_color = str_replace(font_color, "font_color", ""))

G_amm <- T_amm %>%
  ggplot(aes(x = font_color,
             color = font_size, shape = font_size,
             y = center)) +
  geom_point() +
  geom_line(aes(group = font_size))
```

Note that in a CLU table the column `fixef` stores two identifiers, the level of font-size and the level of font_color. For putting them on different GGplot aesthetics we first have to rip them apart using `separate` before using `mutate` and `str_replace` to strip the group labels off the factor names.

Since the coefficient table also contains the 95% certainty limits, we can produce a conditional plot with overlayed credibility intervals (`geom_errorbar`). These limits belong to the group means, and generally cannot be used to tell about the treatment effects.

```
G_amm + geom_errorbar(aes(ymin = lower, ymax = upper), width = .2)
```

Still, it gives the observer some sense of the overall level of certainty. And, when two 95% CIs do not overlap, that means that the difference is different from zero with 95% credibility, at least. Another useful Ggplot geometry is violin plots, as these make the overlap between CIs visible and reduce visual clutter caused by all these vertical error bars.

However, a violin plot requires more that just three CLU estimates. Recall from 4.1.1 that the posterior object, obtained with `posterior` stores the full certainty information gained by the MCMC estimation walk. The CLU estimates we so commonly use, are just condensing this information into three numbers (CLU). By pulling the estimated posterior distribution into the plot, we can produce a conditional plot that conveys more information and is easier on the eye.

```
P_amm <-
  posterior(M_amm) %>%
  filter(type == "fixef") %>%
  select(fixef, value) %>%
  separate(fixef, c("font_size", "font_color"), sep = ":") %>%
  mutate(font_size = str_replace(font_size, "font_size", ""),
         font_color = str_replace(font_color, "font_color", ""))


G_amm +
  geom_violin(data = P_amm,
              aes(y = value,
                  fill = font_size),
              alpha = 0.5,
              position = position_identity(),
              width = .2)
```

```
detach(Reading)
```

As the figure shows, ergonomics is maximized by using large fonts and high contrast. Still, there is saturation and therefore it does little harm to go with the gray font, as long as it is 12pt.

We have seen in the Headache example that conditional effects occur as non-linearity. The more a participant approaches the natural boundary of zero headache, the less benefit is created by additional effort. This we call *saturation*. Saturation is likely to occur when multiple factors influence the same cognitive or physical system or functioning. In quantitative comparative design studies, we gain a more detailed picture on the co-impact of design interventions and can come to more sophisticated decisions.

If we don't account for saturation by introducing interaction terms, we are prone to underestimate the net effect of any of these measures and may falsely conclude that a certain treatment is rather ineffective. Consider a large scale study, that assesses the simultaneous impact of many demographic variables on how willing customers are to take certain energy saving actions in their homes. It is very likely that subsets of variables are associated with similar cognitive processes. For example, certain action requires little effort (such as switching off lights in unoccupied rooms), whereas others are time-consuming (drying the laundry outside). At the same time, customers may vary in the overall eagerness (motivation). For high effort actions the impact of motivation level probably makes more of a difference than when effort is low. Not including the conditional effect would result in the false conclusion that suggesting high effort actions is rather ineffective.

4.5.4 More than the sum: amplification

Saturation effects occur, when multiple impact factors act on the same system and work in the same direction. When reaching the boundaries, the change per unit diminishes. We can also think of such factors as exchangeable. *Amplification* conditional effects are the opposite: It only really works, if all conditions are fulfilled. Conceiving good examples for amplification effects is far more challenging as compared to saturation effects. Probably this is because saturation is a rather trivial phenomenon, whereas amplification involves some non-trivial orchestration of cognitive or physiological subprocesses. Here, a fictional case on technology acceptance will serve to illustrate amplification effects. Imagine a start-up company that seeks funding for a novel augmented reality game, where groups of gamers compete for territory. For a fund raising, they need to know their market potential, i.e. which fraction of the population is potentially interested. The entrepreneurs have two hypotheses they want to verify:

1. Only technophile persons will dare to play the game, because it requires some top-notch equipment.

2. The game is strongly cooperative and therefore more attractive for people with a strong social motif.

Imagine a study, where they asked a larger set of participants to rate their technophily and sociophily. They were then given a description of the planned game and were asked how much they intended to participate in the game.

While the example primarily serves to introduce amplification effects, it is also an opportunity to get familiar with conditional effects between metric predictors. Although this is not very different to conditional effects on groups, there are a few peculiarities, one being that we cannot straight-forwardly make an exploratory plot. For factors, we have used box plots, but these do not apply for metric predictors. In fact, it is very difficult to come up with a good graphical representation. One might think of 3D wire-frame plots, but these transfer poorly to the 2D medium of these pages. Another option is to create a scatter-plot with the predictors on the axes and encode the outcome variable by shades or size of dots . These options may suffice to see any present main effects, but are too coarse to discover subtle non-linearity. The closest we can get to a good illustration is to create groups and continue as usual. Note, that turning metric predictors into factors is just a hack to create exploratory graphs. By no means do I intend to corroborate the use of group-mean models on metric data.

```
attach(AR_game)
```

```
D_1 %>%
  mutate(Sociophile = forcats::fct_rev(ifelse(sociophile > median(sociophile), "high", "low
         Technophile = forcats::fct_rev(ifelse(technophile > median(technophile), "high", "
  ggplot(aes(y = intention, x = Technophile, col = Sociophile)) +
  geom_boxplot() +
  ylim(0, 0.5)
```

From the boxplot it seems that both predictors have a positive effect on intention to play. However, it remains unclear whether there is a conditional effect. In absence of a better visualization, we have to rely fully on the numerical estimates of a conditional linear regression model (CMRM).

```
M_cmrm <-
  D_1 %>%
  stan_glm(intention ~ 1 + sociophile + technophile + sociophile : technophile,
           data = .)
```

```
coef(M_cmrm)
```

parameter	fixef	center	lower	upper
Intercept	Intercept	0.273	0.258	0.288
sociophile	sociophile	0.114	0.073	0.153
technophile	technophile	0.182	0.153	0.212
sociophile:technophile	sociophile:technophile	0.163	0.080	0.243

The regression model confirms that sociophilia and technophilia both have a moderate effect on intention. Both main effects are clearly in the positive range. Yet, when both increase, the outcome increases over-linearly. The sociophile-technophile personality is the primary target group.

While plotting the relationship between three metric variables is difficult, there are alternative ways to illustrate the effect. For example, we could ask: how does intention change by .1 unit sociophilia for two imaginary participants with extreme positions on technophilia (e.g., .2 and .8). We could calculate these values arithmetically using the model formula and the center estimates. The better and genuinely Bayesian way of doing it is sampling from the *posterior predictive distribution* (PPD). This distribution is generated during parameter estimation. While the posterior distribution (PD) represents the predictors, the PPD is linked to the outcome variable. More specifically, the PPD is the models best guess of the expected value μ. Once the posterior has been estimated, we can draw from it with any values of interest. As these can be combinations of values that have never been observed, we can truly speak of prediction. Regression engines usually provide easy means to simulate from a PD and generate predictions. In the following example, we simulate some equidistant steps of sociophilia for three participants with technophilia scores from .1 to .9.

```
D_2 <-
  mascutils::expand_grid(technophile = seq(.1, .9, by = .1),
                         sociophile = seq(.3, .6, by = .1)) %>%
  arrange(technophile)

T_comb_pred <-
  post_pred(M_cmrm, newdata = D_2, thin =2) %>%
  predict()

D_2 <-
  D_2 %>%
  mutate(intention = T_comb_pred$center)
```

```
D_2 %>%
  mutate(technophile = as.factor(technophile)) %>%
  ggplot(aes(x = sociophile, col = technophile, y = intention)) +
  geom_point() +
  geom_line(aes(group = technophile))
```

The effect is not stunning, but visible. The lines diverge, which means they have different slopes. With high technophilia, every (tenth) unit of sociophilia has a stronger effect on intention to play.

```
detach(AR_game)
```

Saturation effects are about declining net effects, there more similar treatments pile up, that can be the same amount of training (which gives a curve of diminshing returns) or two similar treatments. Amplification effects are more like two-component glue. When using only one of the components, all you get you get is a smear. The only way of getting a strong hold is to put them together. This has a parallel in formal logic. The logical `AND`, requires both operands to be `TRUE` for something to happen operator returns `TRUE` only when both operands A and B are `TRUE`. Instead, a saturation process can be imagined as logical `OR`. If A is already `TRUE`, B no longer matters.

```
T_bool_interaction <-
tibble(A = c(F, F, T, T),
           B = c(F, T, F, T)) %>%
  as_tibble() %>%
  mutate("A OR B" = A | B) %>%
  mutate("A AND B" = A & B)

kable(T_bool_interaction)
```

A	B	A OR B	A AND B
FALSE	FALSE	FALSE	FALSE
FALSE	TRUE	TRUE	FALSE
TRUE	FALSE	TRUE	FALSE
TRUE	TRUE	TRUE	TRUE

For decision making in design research, the notion of saturation and amplification are equally important. Saturation effects can happen with seemingly different design choices that act on the same cognitive (or other) processes. That is good to know, because it allows the designer to compensate one design feature with the other, should there be a conflict between different requirements, such as easthetics and readability of text. Amplification effects are interesting, because they break barriers. Only if the right ingredients are present, a system is adopted by users. Many technology break-throughs can

perhaps be attributed to adding the final necessary ingredient. Sometimes you can also ask: what ingredient has been missing for some earlier technology flops. For example, the first commercial smart phone (with touchscreen, data connectivity and apps) has been the IBM Simon Personal Communicator, introduced in 1993, which was discontinued after only six months on the market. It lasted more than ten years before smartphones actually took off. What were the magic ingredients added?

A feature that must be present for the users to be satisfied (in the mere sense of absence-of-annoyance) is commonly called a *necessary user requirements*. That paints a more moderate picture of amplification in everyday design work: The peak, where all features work together usually is not the magic break-through; it is the strenuous path of user experience design, where user requirements whirl around you and not a single one must be left behind.

4.5.5 Conditional effects and theory

Explaining or predicting complex behaviour with psychological theory is a typical approach in design research. Unfortunately, it is not an easy one. While design is definitely multifactorial, with a variety of cognitive processes, individual differences and behavioural strategies, few psychological theories cover more than three associations between external or individual conditions and behaviour. The design researcher is often forced to enter a rather narrow perspective or knit a patchwork model from multiple theories. Such a model can either be loose, making few assumptions on how the impact factors interact which others. A more tightened model frames multiple impact factors into a conditional network, where the impact of one factor can depend on the overall configuration. A classic study will now serve to show how conditional effects can clarify theoretical reasoning.

Vigilance is the ability to remain attentive for rarely occurring events. Think of truck drivers on lonely night rides, where most of the time they spend keeping the truck on a straight 80km/h course. Only every now and then is the driver required to react to an event, like when braking lights flare up ahead. Vigilance tasks are among the hardest thing to ask from a human operator. Yet, they are safety relevant in a number of domains.

Keeping up vigilance most people perceive as tiring, and vigilance deteriorates with tiredness. Several studies have shown that reaction time at simple tasks increases when people are tired. The disturbing effect of noise has been documented as well. A study by [%Corcoran (1961)] examined the simultaneous influence of sleep deprivation and noise on a rather simple reaction task. They asked:

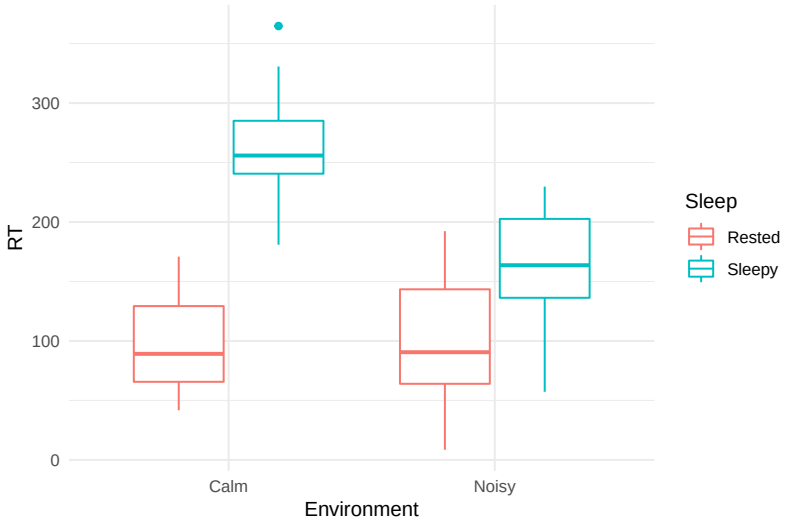
> will the effects of noise summate with those of loss of sleep to induce an even greater performance decrement or will noise subtract from the performance decrement caused by loss of sleep?

The central argument is that sleep deprivation deteriorates the central nervous arousal system. In consequence, sleep deprived persons cannot maintain the necessary level of energy that goes with the task. Noise is a source of irritation and therefore usually reduces performance. At the same time, noise may have an arousing effect, which may compensate for the loss of arousal due to sleep deprivation. To re-iterate on the headache pills analogy 4.5.3, noise could be the antidote for sleepiness.

The Sleep case study is a simplified simulation of Corcoran's results. Participants were divided into 2x2 groups (quiet/noisy, rested/deprived) and had to react to five signal lamps in a succession of trials. In the original study, performance measure gaps were counted, which is the number of delayed reactions ($> 1500ms$). Here we just go with (simulated) reaction times, assuming that declining vigilance manifests itself in slower reactions.

```
attach(Sleep)
```

```
D_1 %>%
  ggplot(aes(x = Environment,
             color = Sleep,
             y = RT)) +
  geom_boxplot()
```



Using a 2x2 model including a conditional effect, we examine the conditional association between noise and sleepiness.

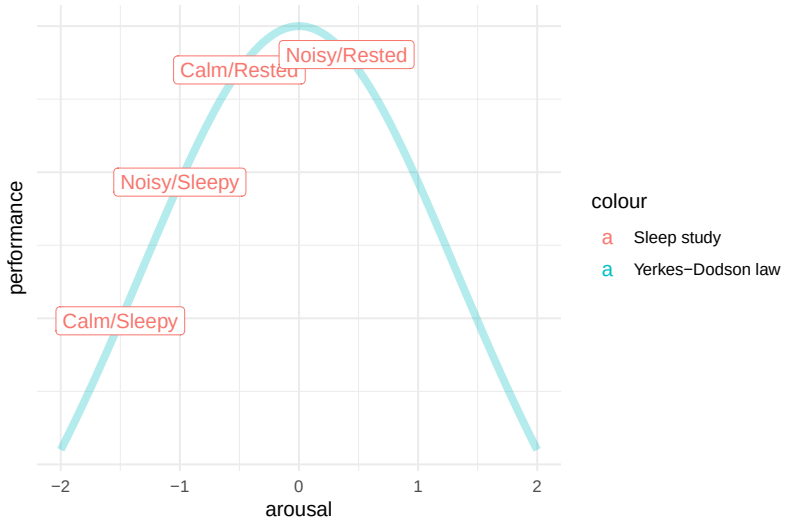
```
M_1 <-
  D_1 %>%
  stan_glm(RT ~ Environment * Sleep, data = .)
```

Note that the `*` operator in the model formula is an abbreviation for a fully factorial model `1 + Environment + Sleep + Environment:Sleep`.

parameter	fixef	center	lower	upper
Intercept	Intercept	98.545	63.5	134.8
EnvironmentNoisy	EnvironmentNoisy	0.532	-49.9	50.8
SleepSleepy	SleepSleepy	160.667	110.9	209.2
EnvironmentNoisy:SleepSleepy	EnvironmentNoisy:SleepSleepy	-98.096	-171.0	-30.2

Recall, that treatment contrasts were used, where all effects are given relative to the reference group quiet-rested (intercept). The results confirm the deteriorating effect of sleepiness, although its exact impact is blurred by pronounced uncertainty $160.67[110.94, 209.24]_{CI95}$. Somewhat surprisingly, noise did not affect well-rested persons by much $0.53[-49.91, 50.77]_{CI95}$. Note however, that we cannot conclude a null effect, as the credibility limits are wide. Maybe the lack of a clear effect is because steady white noise was used, not a disturbing tumult. The effect of sleepiness on RT is partly reduced in a noisy environment $-98.1[-170.97, -30.2]_{CI95}$. This suggests that the arousal system is involved in the deteriorating effect of sleep deprivation, which has interesting consequences for the design of vigilance tasks in the real world.

These findings reverb with a well known in Psychology of Human Factors, the Yerkes-Dodson law. The law states that human performance at cognitive tasks is influenced by arousal. The influence is not linear, but better approximated with a curve as shown in the figure below. Performance is highest at a moderate level of arousal. If we assume that sleepy participants in Corcona's study showed low performance due to under-arousal, the noise perhaps has increased the arousal level, resulting in better performance. If we accept that noise has an arousing effect, the null effect of noise on rested participants stands in opposition to the Yerkes-Dodson law: if rested participants were on an optimal arousal level, additional arousal would usually have a negative effect on performance. There is the slight possibility, that Corcona has hit a sweet spot: if we assume that calm/rested participants were still below an optimal arousal level, noise could have pushed them right to the opposite point.




```
detach(Sleep)
```

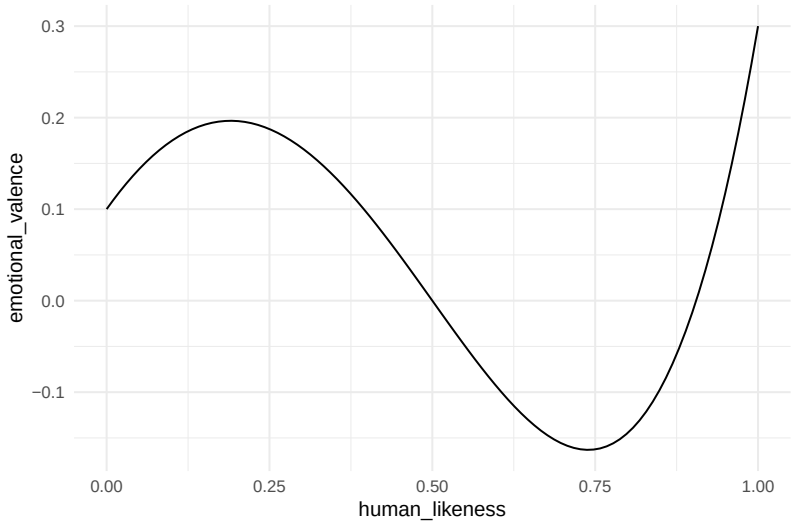
To sum it up, saturation and amplification effects have in common that performance is related to design features in a monotonous increasing manner (, albeit not linearly increasing). Such effects can be interpreted in a straightforward manner: when saturation occurs with multiple factors, it can be inferred that they all impact the same underlying cognitive mechanism and are therefore interchangeable to some extent, like compensating letter size with stronger contrast. In turn, amplification effects indicate that multiple cognitive mechanisms (or attitudes) are necessarily involved and must be regarded during design. The Sleep study demonstrates that conditional effects can also occur in situations with *non monotonously increasing* relationships between design features and performance. When such a relationship takes the form of a parabole, like the Yerkes-Dodson law, the designer (or researcher) is faced with the more complex problem of finding the sweet spot. Central to the Yerkes-Dodson law is that arousal is a gradually increasing condition, and so are noise level and degree of sleep deprivation. A consequential follow-up study would be one, where these levels are manipulated (or measured) on more levels than the original could serve to identify the position of optimal performance more accurately. In 4.6 we have encountered a similar case: the trough of the Uncanny Valley effect can be estimated and this estimate can help researchers to avoid the critical region.

4.6 Doing the rollercoaster: polynomial regression models

In the preceding four sections, we used linear models to render processes that are not linear, but curved. These non-linear processes fell into two classes: dull diminishing return curves for learning curve OFMs or saturation effects and the more sudden peaks of amplification. But, what can we do when a process follows more complex curves, with more ups-and downs? In the following I will introduce polynomial regression models, which still can be formulated as linear models, but can take a wide variety of shapes.

Robots build our cars and sometimes drive them. They mow the lawn and may soon also deliver parcels to far-off regions. Prophecy is that robots will also enter social domains, such as care for children and the elderly. One can assume that in social settings emotional acceptance plays a significant role for technology adoption. Next to our voices, our faces and mimic expressions are the main source of interpersonal messaging. Since the dawn of the very idea of robots, anthropomorphic designs have been dominant. Researchers and designers all around the globe are currently pushing the limits of human-likeness of robots. One could assume that emotional response improves with every small step towards perfection. Unfortunately, this is not the case. [%Mori] discovered a bizarre non-linearity in human response: people's emotional response is proportional t human-likeness, but only at the lower end. A robot design with cartoon style facial features will always beat a robot vacuum cleaner. But, an almost anatomically correct robot face may provoke a very negative emotional response, which is called the *uncanny valley*.

```
tibble(hl = seq(-1, 1, length.out = 100),
             emotional_valence = -.5 * hl + .6 * hl^3 + .2 * hl^4) %>%
  mutate(human_likeness = (hl + 1)/2) %>%
  ggplot(aes(x = human_likeness, y = emotional_valence)) +
  geom_line()
```



[%Mathur et al.] study aimed at rendering the association between human-likeness and liking at full range. They collected 60 pictures of robots and attached a score for human likeness to them. Then they asked participants how much they liked the faces. Fort the data analysis they calculated an average score of likability per robot picture. Owing to the curved shape of the uncanny valley, linear regression is not applicable to the problem. Instead, Mathur et al. applied a third degree polynomial term.

A polynomial function of degree k has the form:

$$y_i = \beta_0 x_i^0 + \beta_1 x_i^1 + ... + \beta_k x_i^k$$

In fact, you are already familiar with two polynomial models. The zero degree polynomial is the grand mean model. This follows from $x_i^0 = 1$, which makes β_0 a constant, the intercept. Also, a first degree polynomial is simply the linear model. By adding higher degrees we can introduce more complex curvature to the association.

```
D_poly <-
  tibble(x = seq(-2, 3, by = .1),
              degree_0 = 2,
              degree_1 = 1   *  degree_0 + 3 * x,
              degree_2 = 0.5 * (degree_1 + 2 * x^2),
              degree_3 = 0.5 * (degree_2 + -1 * x^3),
              degree_4 = 0.4 * (degree_3 + 0.5 * x^4),
              degree_5 = 0.3 * (degree_4 + -0.3 * x^5)) %>%
  gather(polynomial, y, degree_0:degree_5) %>%
  arrange(polynomial, y, x)
```

```
D_poly %>%
  ggplot(aes(x, y)) +
  geom_line() +
  facet_wrap(~polynomial)
```

Mathur et al. argue that the Uncanny Valley curve possesses two stationary points, with a slope of zero: the valley is a local minimum and represents the deepest point in the valley, the other is a local maximum and marks the shoulder left of the valley. Such a curvature can be approximated with a polynomial of (at least) third degree, which has a constant term β_0, a linear slope $x\beta_1$, quadratic component $x^2\beta_2$ and a cubic component $x^3\beta_3$.

While R provides high-level methods to deal with polynomial regression, it is instructive to build the regression manually. The first step is to add variables to the data frame, which are the predictors taken to powers ($x_k = x^k$). These variables are then straight-forwardly added to the model term, as if they were independent predictors. For better clarity, we rename the intercept to be x_0, before summarizing the fixed effects.

```
attach(Uncanny)
```

```
M_poly_3 <-
  RK_2 %>%
  mutate(huMech_0 = 1,
         huMech_1 = huMech,
         huMech_2 = huMech^2,
         huMech_3 = huMech^3) %>%
  stan_glm(avg_like ~ 1 + huMech_1 + huMech_2 + huMech_3,
           data = ., iter = 2500)
  # stan_glm(avg_like ~ poly(huMech, 4),
  #          data = ., iter = 100)

P_poly_3 <- posterior(M_poly_3)
```

```
coef(P_poly_3)
```

parameter	fixef	center	lower	upper
Intercept	Intercept	-0.452	-0.510	-0.392
huMech_1	huMech_1	0.161	-0.262	0.568
huMech_2	huMech_2	-1.114	-2.012	-0.184
huMech_3	huMech_3	0.941	0.339	1.533

parameter	fixef	center	lower	upper
Intercept	Intercept	-0.452	-0.510	-0.392
huMech_1	huMech_1	0.161	-0.262	0.568
huMech_2	huMech_2	-1.114	-2.012	-0.184
huMech_3	huMech_3	0.941	0.339	1.533

We can extract the fixef effects table as usual. The four coefficients specify the polynomial to approximate the average likeability responses. The polynomial parameters have little explanatory value, neither one alone relates to a relevant property of the uncanny valley. One relevant property would be the location of the deepest point of the uncanny valley, its trough. The trough is a local minimum of the curve and we can find this point with polynomial techniques.

Finding a local minimum is a two step procedure: first, we must find all *stationary points*, which includes local minima and maxima. Then, we determine which of the resulting points is the local minimum. Stationary points occur, where the curve bends from a rising to falling or vice versa. At these points, the slope is zero, neither rising nor falling. Therefrore, stationary points are identified by the derivative of the polynomial, which is a second degree (cubic) polynomial:

$$f'(x) = \beta_1 + 2\beta_2 x + 3\beta_2 x^2$$

The derivative $f'(x)$ of a function $f(x)$ gives the slope of $f(x)$ at any given point x. When $f'(x) > 0$, $f(x)$ is rising at x, with $f'(x) < 0$ it is falling. Stationary points are precisely those points, where $f'(x) = 0$ and can be found by solving the equation. The derivative of a third degree polynomial is of the second degree, which has a quadratic part. This can produce a parabolic form, which hits point zero twice, once rising and once falling. A rising encounter of point zero indicates that $f(x)$ has a local minimum at x, a falling one indicates a local maximum. In consequence, solving $f'(x) = 0$ can result in two solutions, one minimum and one maximum, which need to be distinguished further.

If the stationary point is a local minimum, as the trough, slope switches from negative to positive; $f'(x)$ crosses $x = 0$ in a rising manner, which is a positive slope of $f'(x)$. Therefore, a stationary point is a local minimum, if $f''(x) > 0$.

Mathur et al. followed these analytic steps to arrive at an estimate for the position of the trough. They did so on the point estimates they got with their frequentist estimation method. In the following, we will do the same operations on the center estimates of the CLU table. After that, we apply it to the posterior distribution which enables us to make uncertainty statements.

```
library(polynom)

poly     <- polynomial(T_fixef_1$center) # UC function on center
dpoly    <- deriv(poly)                  # 1st derivative
ddpoly   <- deriv(dpoly)                 # 2nd derivative
stat_pts <- solve(dpoly)                 # finding stat points
slopes   <- as.function(ddpoly)(stat_pts)# slope at stat points
trough   <- stat_pts[slopes > 0]         # selecting the local minimum

cat("The trough is most likely at a huMech score of ", round(trough, 2))
```

```
## The trough is most likely at a huMech score of  0.71
```

Note how the code uses high-level functions from package `polynom` to estimate the location of the trough, in particular the first and second derivative `d[d]poly`

This procedure repeats the original analysis, but there is a limitation: using the center estimates, we get a point estimate for the position of the trough, but without any information on uncertainty. The solution is that we let the trough calculation work on every single draw of the posterior distribution. That will produce a posterior distribution of the derived trough position.

Every step of the MCMC walk produces a simultaneous draw of the four parameters `huMech_[0:3]`, and therefore fully specifies a third degree polynomial. The posterior distribution of the position of the trough can be computed, by computing the position for every iteration. For the convenience, the R package Uncanny contains a function `trough(coef)` that includes all the above steps. The following code creates a data frame with one row per MCMC draw and the four huMech variables, the function `trough` acts on this data frame as a matrix of coefficients and returns one trough point per row. We have obtained the PD of the trough.

```
devtools::install_github("schmettow/Uncanny")
```

```
P_trough <-
  P_poly_3 %>%
  filter(type == "fixef") %>%
  select(chain, iter, fixef, value) %>%
  spread(fixef, value) %>%
  select(Intercept, starts_with("huMech")) %>%
  mutate(trough = uncanny::trough(.)) %>%
  gather(key = parameter)
```

This posterior distribution we can plot, or put it into a CLU table:

```
P_trough %>%
  group_by(parameter) %>%
  summarize(center = median(value, na.rm = T),
            lower = quantile(value, .025, na.rm = T),
            upper = quantile(value, .975, na.rm = T))
```

parameter	center	lower	upper
huMech_1	0.161	-0.262	0.568
huMech_2	-1.114	-2.012	-0.184
huMech_3	0.941	0.339	1.533
Intercept	-0.452	-0.510	-0.392
trough	0.709	0.648	0.782

The 95% CI is a conventional measure of uncertainty and may be more or less irrelevant. The most generous display on uncertainty is a density plot on the full posterior. The density function just smooths over the frequency distribution of trough draws, but makes no arbitrary choices on where to cut it.

```
RK_2$M_poly_3 <- predict(M_poly_3)$center

gridExtra::grid.arrange(
  RK_2 %>%
    ggplot(aes(x = huMech, y = avg_like)) +
    geom_point(size = .3) +
    geom_smooth(aes(y = M_poly_3), se = F),

  P_trough %>%
    filter(parameter == "trough") %>%
    ggplot(aes(x = value)) +
    geom_density() +
    xlim(0, 1),

  heights = c(.8, .2)
)
```

With reasonable certainty, we can say that the trough is at approximately two-thirds of the huMech score range. In contrast, the illustration of the uncanny valley as they used to be perpetuated from the original source, place the trough at about four quarters of the scale. The Uncanny Valley effect seems to set in "earlier" than we thought.

A closer look at the scatterplot above reveals a problem with the data set: It seems that data is sparsest right where the valley is deepest. Since there

also is a lot of noise, the concern is that there actually is no trough. This can be tested on the same posterior. The `uncanny::trough` function returns a missing value, when no minimum stationary point could be found. Hence, the proportion of non-`NA`s is the certainty we have that a trough exists:

```
print("Probability that there is a trough:", 1 - mean(is.na(P_trough)) )
```

```
## [1] "Probability that there is a trough:"
```

```
detach(Uncanny)
```

4.6.1 Make yourself a test statistic

Generally, in design research we are interested in real world impact and this book takes a strictly quantitative stance. Rather than testing the hypothesis whether any effect exists or not, we interpret coefficients by making statements on their magnitude and uncertainty. Most often, when a higher polynomial models is required, the coefficients do not have a clear meaning. We evaluated the position of the local minimum, the trough. The theory goes that the Uncanny Valley effect is a disruption of a slow upwards trend, the disruption creates the shoulder and culminates in the trough. But, there is no single coefficient telling us directly that there actually are a shoulder and a trough.

Polynomial theory tells us that a cubic function *can* have two stationary points, but it can also just have one or zero. After all, straight line is a cubic, too, if we set the quadratic and cubic coefficients to zero. The fact that a cubic poylnomial fits the data best is a good indicator that exactly two stationary points are needed. But, compared to the straight-in-the-face Uncanny Valley this is unsatisfyingly indirect. Wouldn't it be bold if we could say: the Uncanny Valley exists *with a certainty of . . .*?

Recall, that when a cubic model is estimated, the MCMC walk makes random visits in a four-dimensional coefficient space 4.1.1 (five-dimensional, if we count the error variance). These coordinates are stored *per iteration* in a posterior distribution object. Every iteration represents one possible polynomial.

```
attach(Uncanny)
```

```
post_pred(M_poly_3, thin = 10) %>%
  left_join(RK_2, by = "Obs") %>%
  ggplot(aes(x = huMech, y = value, group = iter)) +
  stat_smooth(geom='line', alpha=0.2, se=FALSE)
```

All we have to do is count the number of MCMC visits, that have a trough and a shoulder. The function `trough` in the Uncanny package (on Github) is designed to return the position, when it exists and simply returns `NA` otherwise. The same goes for the function `shoulder`, which finds the local maximum, if any.

With these two functions, we can create two simple test statistics, by counting how many of the MCMC draws represent a cubic polynomial *with* shoulder and trough.

```
# devtools::install_github("schmettow/uncanny")
library(uncanny)

P_wide <-
  P_poly_3 %>%
  filter(type == "fixef") %>%
  #as_tibble() %>%
  select(iter, parameter, value) %>%
  spread(key = parameter, value = value) %>%
  select(Intercept, starts_with("huMech")) %>%
  mutate(trough     = uncanny::trough(.),
         shoulder   = uncanny::shoulder(.),
         is_Uncanny = !is.na(trough) & !is.na(shoulder) )

print("The probability that the Uncanny Valley does exist is:")
```

```
## [1] "The probability that the Uncanny Valley does exist is:"
```

```
print(mean(P_wide$is_Uncanny))
```

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

So, with the our data we can be pretty sure that the Uncanny Valley effect is present. Probably, there is a very tiny chance that it does not exist, which we would only catch by increasing the resolution of the posterior, i.e. running more MCMC iterations. This conclusion is even more interesting from a philosophy-of-science point-of-view. It was in 1970, when Masahiro Mori published his theory on the relation between human likeness and emotional response. Fifty years later this article all but outdated in how lucidly it anticipates the emerge of human-like robots and virtual characters [%https://spectrum.ieee.org/automaton/robotics/humanoids/the-uncanny-valley]. But for a modern reader in Social Sciences the article abruptly stops, right where we would expect the experimental part confirming

the theory. It seems that Mori's theory sprang just from his own feelings. Introspection as a scientific method is likely to give seasoned researcher another uncanny feeling. But, we must not mistaken here, Mori used his inner world for finding a theory, not for testing it. Once the world was ready it, Mori's theory turned out to provable and immensely useful for design.

Still, I argue that we have not yet fully confirmed Mori's theory. Strictly spoken, the data of Mathur & Reichling only prove that *on average* the effect exists, that is if we aggregate over participants. It would be much stronger to state: *everyone* experiences the Uncanny Valley. In essence, we could estimate the same cubic models, but *one per participant*. That requires non-aggregated data, because the analysis of very participant requires the data from every participant. The next chapter will introduce *multi-level models*, which can simultaneously estimate a model on population level and participant level. At the end of the following chapter, we will return to the Uncanny Valley with more data to feed our chains. Spoiler alert: the Uncanny Valley effect could be *universal*.

5

Multilevel models

In the previous chapters we have seen several examples of conditional effects: groups of users responding differently to design conditions, such as font size, noise and emerging technology. Dealing with differential design effects seems straight forward: identify the relevant property, record it and add an conditional effect to the model. Identifying the relevant property is, in fact, a catch-22: how would you know what is relevant before you actually conducted the research. Researchers routinely record basic demographic properties such as age and gender, but these frequently show little effects, or the effects are obvious, i.e. not interesting. In addition, such predictors are rarely more than approximations of the properties that make the real difference. Older people have weaker vision *by tendency*, but the individual differences in any age group are immense. Boys tend to be more technophile, but there are some real geeky girls, too.

Identifying properties that matter upfront requires careful review of past research or deep theorizing, and even then it remains guesswork. Presumably, hundreds of studies attempted to explain differences in usage patterns or performance by all sorts of psychological predictors, with often limited results. That is a big problem in design research, as variation in performance can be huge and good predictors are urgently needed. Identifying the mental origins of being fast versus slow, or motivated versus bored, is extremely useful to improve the design of systems to be more inclusive or engaging.

As we will see in this chapter, individual differences can be accounted for and measured accurately without any theory of individual differences. For researchers trained in experimental social sciences it may require a bit of getting used to theory-free reasoning about effects, as it is always tempting to ask for the *why*. But in applied evaluation studies, what we often really need to know is by *how much* users vary. The key to measuring variation in a population is to create models that operate on the level of participants, in addition to the population level, for example

- on population level, users prefer design B over A on average ($\beta_1 = 20$)
- on the participant-level, participant i preferred B over A ($\beta_{1i} = 20$), j preferred A over B ($\beta_{1j} = -5$), +

When adding a participant-level effects, we still operate with coefficients, but in contrast to single-level linear models, every participant gets their own estimate ($\beta_{1.}$). The key to estimating individual parameters is simply to regard participant (`Part`) a grouping variable on its own, and introduce it as a factor.

The subsequent two sections introduce the basics of estimating multi-level linear models, first introducing intercept-only participant-level effects 5.1 and then slope (or group difference) effects 5.2. Typically, fixed and random effects appear together in a linear multi-level model. It depends on the research question whether the researcher capitalizes on the average outcome, the variation in the population or participant-level effects.

The participant-level is really just the factor and once it is regarded alongside the population level, a model is multi-level. However, in multi-level linear modelling we usually use a different type of factor, for the particpant level. The additional idea is that the levels of the factor, hence the individuals, are part of a *population*. The consequences of this perspective, will be discussed in 5.7: a population is a set of entities that do vary to some extent but also clump around a typical value. And that is precisely what *random effects* do: levels are drawn from an overarching distribution, usually the Gaussian. This distribution is estimated simultaneously to the individual parameters ($\beta_{1.}$), with some interesting consequences. We will return to a more fundamental research case, the Uncanny Valley, and examine the *universality* of this strange effect 5.4.

Once it is clear what the concept of random effects means for studying participant behaviour, we will see that it transfers with grace to *non-human populations*, such as designs, teams or questionnaire items. Three sections introduce multi-population multi-level models: In 5.5 we will use a random effects model with four populations and compare their relative contribution to overall variance in performance. Section @ref(re_nested) will show how multiple levels can form a hierarchy and in @ref(re_psychometrics) we will see that multi-level models can be employed the development of *psychometrics tests*, that apply for people. Finally, we will see how to treat tests to compare designs, which I call *designometrics* 5.8.4.

5.1 The Human Factor: Intercept random effects

Design science fundamentally deals with interaction between systems and humans. Every measure we take in a design study is an encounter of an individual

with a system. As people differ in many aspects, it is likely that people differ in how they use and perform with a system. In the previous chapter we have already dealt with differences between users: in the BrowsingAB case, we compared two designs in how inclusive they are with respect to elderly users. Such a research question seeks for a definitive answer on what truly causes variation in performance. Years of age is a standard demographic variable and in experimental studies can be collected without hassle. If we start from deeper theoretical considerations than that, for example, we suspect a certain personality trait to play a significant role, this can become more effort. Perhaps, you need a 24-item scale to measure the construct, perhaps you first have to translate this particular questionnaire into three different languages, and perhaps you have to first invent and evaluate a scale. In my experience, personality scales rarely explain much of the variation we see in performance. It may be interesting to catch some small signals for the purpose of testing theories, but for applied design research it is more important to quantify the performance variation within a population, rather than explaining it.

At first, one might think that the grand mean model would do, take β_0 as the population mean and σ_ϵ as a measure for individual variation. But that is mistaken, as the residual variance collects all random sources, not just variance between individuals, in particular residuals are themselves composed of:

- inter-individual variation
- intra-individual variation, e.g. by different levels of energy over the day
- variations in situations, e.g. responsiveness of the website
- inaccuracy of measures, e.g. misunderstanding a questionnaire item

What is needed, is a way to separate the variation of participants from the rest. Reconsider the principles of model formulations: the likelihood captures what is repeatable, what does not repeat goes to the random term. For the problem of identifying individual differences, we simply apply this principle: to pull out a factor from the random term, repetition is needed. For estimating users' individual performance level, all that is needed is repeated measures.

In the IPump study we have collected performance data of 25 nurses, operating a novel interface for a syringe infusion pump. Altogether, every nurse completed a set of eight tasks three times. Medical devices are high-risk systems where a single fault can cost a life. It is required that user performance is on an *uniformly* high level. We start the investigation with the global question:

> What is the average ToT in the population?

```
attach(IPump)
```

```
D_Novel %>%
summarize(mean_Pop = mean(ToT))
```

mean_Pop
16

The answer is just one number and does not refer to any individuals in the population. This is called the population-level estimate or fixed effect estimate. The following question is similar, but is grouped by the participant level:

What is the average ToT of individual participants?

```
D_Novel %>%
group_by(Part) %>%
summarize(mean_Part = mean(ToT)) %>%
  sample_n(5)
```

Part	mean_Part
15	17.2
5	16.0
20	19.1
12	18.9
23	12.0

Such a grouped summary can be useful for situations where we want to directly compare individuals, like in performance tests. In experimental research, individual participants are of lesser interest, as they are exchangeable entities. What matters is the total variation within the sample, representing the population of users. Once we have participant-level effects, the amount of variation can be summarized by the standard deviation:

```
D_Novel %>%
  group_by(Part) %>%
  summarize(mean_Part = mean(ToT)) %>%
  ungroup() %>%
  summarize(sd_Part   = var(mean_Part))
```

sd_Part
13.8

Generally, these are the three types of parameters in multi-level models: the population-level estimate (commonly called *fixed effects*), the participant-level estimates (*random effects*) and the participant-level *standard deviation*.

```
detach(IPump)
```

Obviously, the variable `Part` is key to build such a model. This variable groups observations by participant identity and, formally, is a plain factor. A naive approach to multi-level modelling would be to estimate an AGM, like `ToT ~ 0 + Part`, grab the center estimates and compute the standard deviation. Different to the descriptive analysis above and the naive approach, a multi-level model estimates fixed effects, random effects and random standard deviation *simultaneously*. Random effects are really just factors, with the difference that they are assumed to follow a Gaussian distribution. This will further be explained in section 5.7.

For the IPump study we can formulate a GMM model with participant-level random effect β_{p0} as follows:

$$\begin{aligned} \mu_i &= \beta_0 + x_p\beta_{p0} \\ \beta_{p0} &\sim \text{Gaus}(0, \sigma_{p0}) \\ y_i &\sim \text{Gaus}(\mu_i, \sigma_\epsilon) \end{aligned}$$

There will be as many parameters β_{p0}, as there were users in the sample, and they have all become part of the likelihood. The second term describes the distribution of the participant-level group means. And finally, there is the usual random term. Before we examine further features of the model, let's run it. In the package `rstanarm`, the command `brm()` is dedicated to estimate multi-level models with the extended formula syntax.

However, I will now introduce another Bayesian engine and use it from here on. The Brms package provides the Brm engine, which is invoked by the command `brm()`. This engine covers all models that can be estimated with `stan_glm` or `stan_glmer` and it uses the precise same syntax. All models estimated in this chapter, will also work with `stan_glmer`. That being said, Brms supports an even broader set of models, some of which we will encounter in chapter 6.

For multi-level models, the reason to switch Brms is a selfish one. It became to tedious, to support both Rstanarm and Brms with the Bayr package. The only downside of Brms is that it has to compile the model, preceding the estimation. For simple models, as in the previous chapter, the chains are running very quickly, and the extra step of compilation creates much overhead. For the models in this chapter, the chains run much longer, such that compilation time becomes almost negligible.

```
attach(IPump)
```

```
M_hf <- brm(ToT ~ 1 + (1|Part), data = D_Novel)
P_hf <- posterior(M_hf)
```

The posterior of the multi-level model contains four types of variables:

1. the *fixed effect* captures the population average (Intercept)
2. *random effects* capture how individual participants deviate from the population mean
3. *random factor variation* (or group effects) captures the overall variation in the population.
4. the residual standard deviation captures the amount of noise

With the `bayr` package these parameters can be extracted using the respective commands:

```
fixef(P_hf)
```

model	type	fixef	center	lower	upper
M_hf	fixef	Intercept	16	14.6	17.5

```
ranef(P_hf) %>% sample_n(5)
```

re_entity	center	lower	upper
14	-0.032	-3.30	3.05
9	0.353	-2.09	4.57
15	0.091	-2.90	3.53
18	-0.282	-4.26	2.32
13	0.210	-2.49	4.40

```
grpef(P_hf)
```

model	type	fixef	re_factor	center	lower	upper
M_hf	grpef	Intercept	Part	1.5	0.077	3.76

Random effects are factors and enter the model formula just as linear terms. What sets them apart, is that they are estimated together with the overarching Gaussian distribution. To indicate that to the regression engine, a dedicated

syntax is used in the model formula (recall that `1` represents the intercept parameter):

```
(1|Part)
```

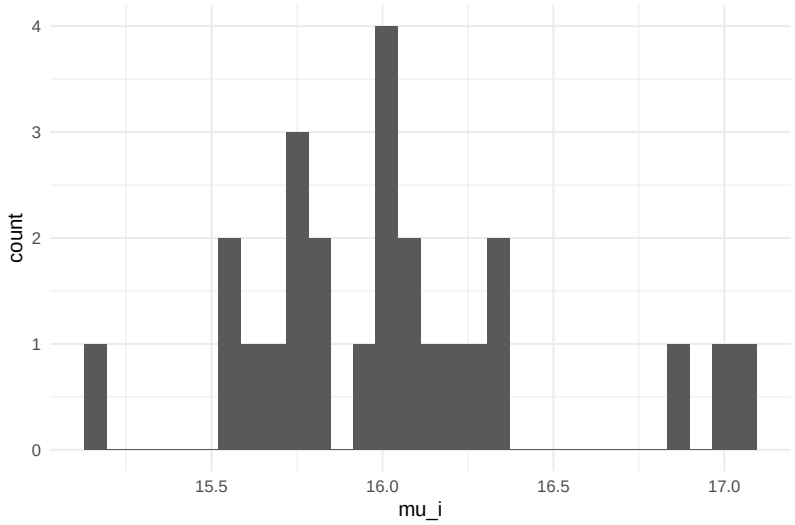
In probability theory expressions, such as the famous Bayes theorem, the `|` symbol means that something to the left is conditional on something to the right. Random effects can easily be conceived as such conditional effects. Left of the `|` is the fixed effect that is conditional on (i.e. varies by) the factor to the right. In the simplest form the varying effect is the intercept and in the case here could be spoken of as:

> ToT depends on the participant you are looking at

Speaking of factors: so far, we have used *treatment contrasts* most of the time, which represent the difference towards a reference level. Analogue to a factor model, we could estimate the first participants performance level and make it the reference group, intercept β_0. All other average scores, we would express as differences to the reference participant. This seems odd and, indeed, has two disadvantages: first, whom are we to select as the reference participant? The choice would be arbitrary, unless we wanted to compare brain sizes against the grey matter of Albert Einstein, perhaps. Second, most of the time the researcher is after the factor variation rather than differences between any two individuals, which is inconvenient to compute from treatment contrasts.

The solution is to use a different contrast coding for random factors: *deviation contrasts* represent the individual effects as *difference (δ) towards the population mean*. As the population mean is represented by the respective fixed effect, we can compute the absolute individual predictions by adding the fixef effect to the random effect:

```
tibble(mu_i = ranef(P_hf)$center +
             fixef(P_hf)$center) %>%
  ggplot(aes(x = mu_i)) +
  geom_histogram()
```



Finally, we can assess the initial question: are individual differences a significant component of all variation in the experiment? Assessing the impact of variation is not as straight-forward as with fixed effects. Two useful heuristics are to compare group-level variation to the fixed effects estimate (Intercept) and against the residual variance:

```
P_hf %>%
  filter(type %in% c("grpef", "disp", "fixef")) %>%
  clu()
```

parameter	type	fixef	re_factor	center	lower	upper
Intercept	fixef	Intercept	NA	16.0	14.556	17.48
sigma_resid	disp	NA	NA	16.4	15.489	17.38
Sigma[Part:Intercept,(Intercept)]	grpef	Intercept	Part	1.5	0.077	3.76

```
detach(IPump)
```

The variation due to individual differences is an order of magnitude smaller than the Intercept, as well as the standard error. This lets us conclude that the novel interface works pretty much the same for every participant. If we are looking for sources of variation, we have to look elsewhere. As we have seen already in section 4.3.5, the main source of variation is learning.

5.2 Slope random effects: variance in change

So far, we have dealt with Intercept random effects that capture the gross differences between participants of a sample. We introduced these random effects as conditional effects like: "the overall performance depends on what person you are looking at". However, most research questions rather regard differences between conditions.

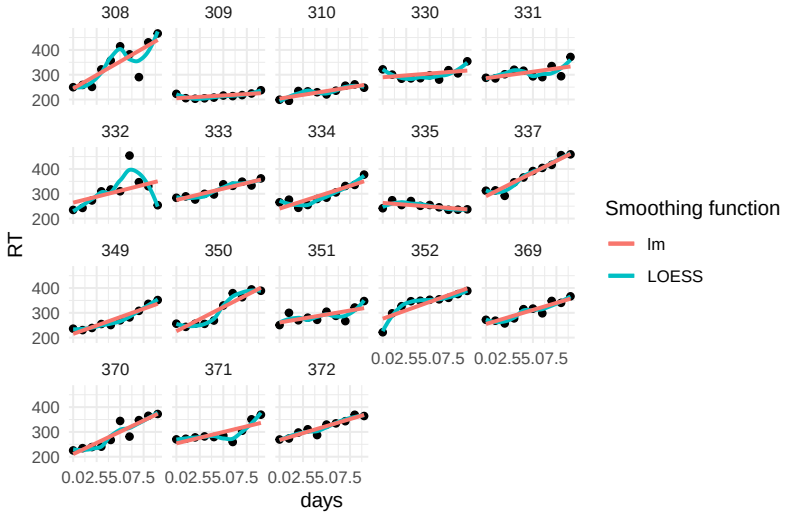
Slope random effects represent, how much individuals differ in their change of response, when conditions change. Consider case BrowsingAB, where the population averages of the designs were not that far apart. That can mean they are truly not that different. But it can also mean that some users do a lot better with A, and others with B. Which design is preferred could largely depend on the person.

For an illustration of slope ramdom effects, we take a look at a data set that ships with package Lme4 (which provides a non-Bayesian engine for multi-level models). 18 participants underwent sleep deprivation on ten successive days and the average reaction time on a set of tests has been recorded per day and participant. The research question is: what is the effect of sleep deprivation on reaction time and, again, this question can be asked on population level and participant level.

The participant-level plot below shows the individual relationships between days of deprivation and reaction time. For most participants a increasing straight line seems to be a good approximation, so we can go with a parsimonous linear regression model, rather than an ordered factor model. One noticeable exception is participant 352, which is fairly linear, but reaction times get shorter with sleep deprivation. (What would be the most likely explanation? Perhaps 352 is a cheater, who slept well every night and only gained experience in doing the tests).

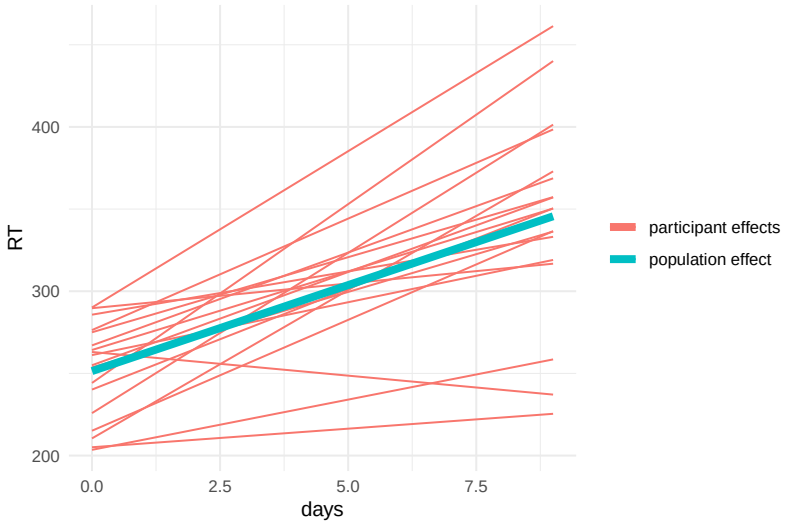
```
attach(Sleepstudy)
```

```
D_slpstd %>%
  ggplot(aes(x = days, y = RT)) +
  facet_wrap(~Part) +
  geom_point() +
  geom_smooth(se = F, aes(color = "LOESS")) +
  geom_smooth(se = F, method = "lm", aes(color = "lm")) +
  labs(color = "Smoothing function")
```



A more compact way of plotting multi-level slopes is the spaghetti plot below. By superimposing the population level effect, we can clearly see that participants vary in how sleep deprivation delays the reactions.

```
D_slpstd %>%
  ggplot(aes(x = days,
             y = RT,
             group = Part)) +
  geom_smooth(aes(color = "participant effects"), size = .5, se = F, method = "lm")+
  geom_smooth(aes(group = 1, color = "population effect"), size = 2, se = F, method = "lm")
  labs(color = NULL)
```



For a single level model, the formula would be `RT ~ 1 + days`, with the intercept being RT at day Zero and the coefficient `days` representing the change per day of sleep deprivation. The multi-level formula retains the population level and adds the participant-level term as a conditional statement: the effect depends on whom you are looking at.

```
RT ~ 1 + days + (1 + days|Part)
```

Remember to always put complex random effects into brackets, because the `+` operator has higher precedence than `|`. We estimate the multi-level model using the rstanarm engine.

```
M_slpsty_1 <- brm(RT ~ 1 + days + (1 + days|Part),
                  data = D_slpstd,
                  iter = 2000)
```

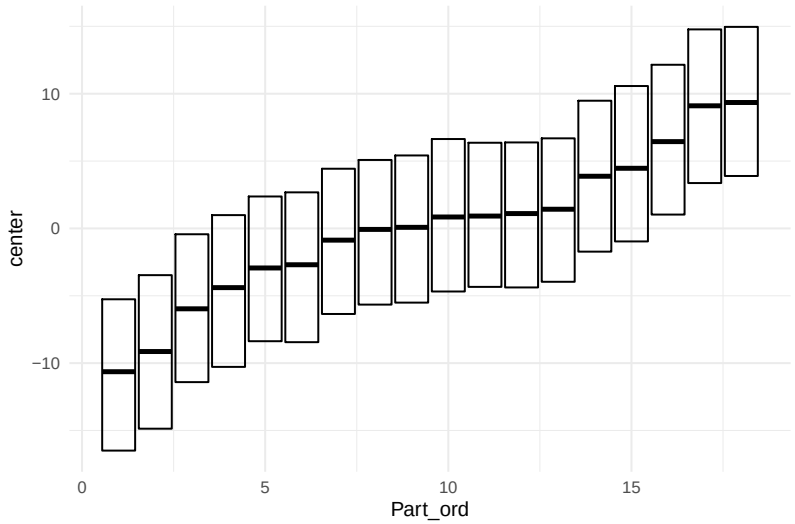
The Bayr package provides a specialized command for multi-level tables. `fixef_ml` extracts the population-level estimates in CLU form and adds the participant-level standard deviation. The overall penalty for sleep deprivation is around ten milliseconds per day, with a 95% CI ranging from 7ms to 14ms. At the same time, the participant-level standard deviation is around 6.5ms, which is considerable. Based on the assumption that the central two standard deviations of a Gaussian distribution contain two-thirds of the total mass, we can expect that roughly one third of the population has a penalty of smaller than 4ms *or* larger than 17ms.

```
fixef_ml(M_slpsty_1)
```

fixef	center	lower	upper	SD_Part
Intercept	251.4	237.72	265	23.11
days	10.4	7.08	14	6.58

The following plot shows the slope random effects, ordered by the center estimate.

```
ranef(M_slpsty_1) %>%
  filter(fixef == "days") %>%
  mutate(Part_ord = rank(center)) %>%
  ggplot(aes(x = Part_ord, ymin = lower, y = center, ymax = upper)) +
  geom_crossbar()
```



The multi-level regression model is mathematically specified as follows. Note how random coefficients $\beta_{.(Part)}$ are drawn from a Gaussian distribution with their own standard deviation, very similar to the errors ϵ_i.

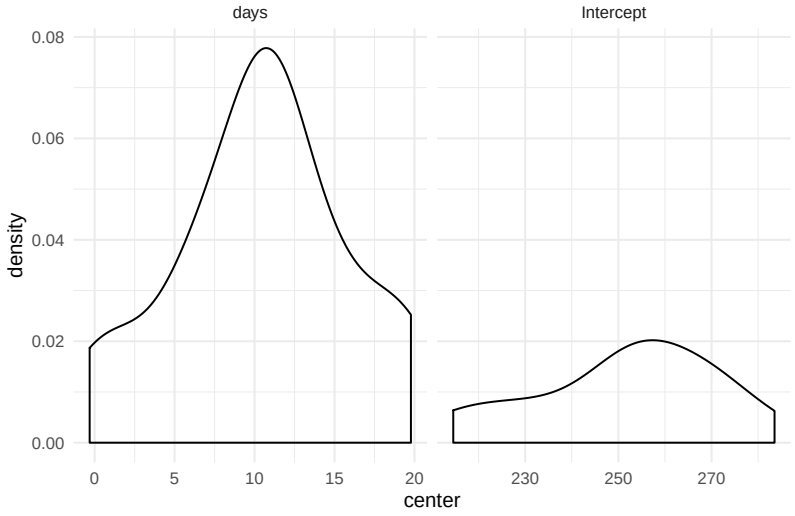
$$
\begin{aligned}
y_i &= \mu_i + \epsilon_i \\
\mu_i &= \beta_0 + \beta_{0(Part)} + x_1\beta_1 + x_1\beta_{1(Part)} \\
\beta_{0(Part))} &\sim \text{Gaus}(0, \sigma_{0(Part)}) \\
\beta_{1(Part))} &\sim \text{Gaus}(0, \sigma_{1(Part)}) \\
\epsilon_i &= \text{Gaus}(0, \sigma_\epsilon)
\end{aligned}
$$

The second line can also be written as:

$$\mu_i = \beta_0 + \beta_{0(Part)} + x_1(\beta_1 + x_1\beta_{1(Part)})$$

which underlines that random coefficients are additive correction terms to the population-level effect. Whereas the `ranef` command reports only these corrections, it is sometimes useful to look at the total scores per participant. In package Bayr, the command `re_scores` computes total scores on the level of the posterior distribution. The following plot uses this command and plots the distribution.

```
posterior(M_slpsty_1) %>%
  re_scores() %>%
  clu() %>%
  ggplot(aes(x = center)) +
  facet_grid(~fixef, scales = "free") +
  geom_density()
```



5.3 Thinking multi-level

There is a lot of confusion about the type of models that we deal with in this chapter. They have also been called hierarchical models or mixed effects models. The "mixed" stands for a mixture of so called fixed effects and random effects. The problem is: if you start by understanding what fixed effects and random effects are, confusion is programmed, not only because there exist several very different definitions. In fact, it does not matter so much whether an estimate is a fixed effect or random effect. As we will see, you can construct a multi-level model by using just plain descriptive summaries. What matters is that a model contains estimates on population level and on participant

level. The benefit is, that a multi-level model can answer the same question for the population as a whole and for every single participant.

For enetering the world of multi-level modelling, we do not need fancy tools. More important is to start thinking multi-level on a familiar example: the IPump case. A novel syringe infusion pump design has been tested against a legacy design by letting trained nurses complete a series of eight tasks. Every nurse repeated the series three times on both designs. Time-on-task was measured and the primary research question is:

> Does the novel design lead to faster execution of tasks?

```
attach(IPump)
```

To answer this question, we can compare the two group means using a basic CGM:

```
M_cgm <-
  D_pumps %>%
  stan_glm(ToT ~ 1 + Design, data = .)
```

```
fixef(M_cgm)
```

fixef	center	lower	upper
Intercept	27.9	25.8	29.8
DesignNovel	-11.9	-14.6	-9.1

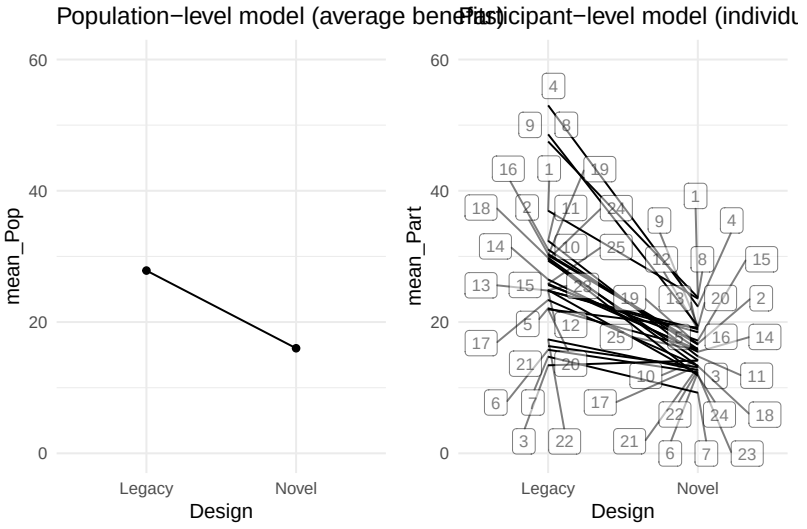
This model is a single-level model. It takes all observations as "from the population" and estimates the means in both groups. It further predicts that with this population of users, the novel design is faster *on average*, that means taking the whole population into account, (and forgetting about individuals).

An average benefit sounds promising, but we should be clear what it precisely means, or better what it does not mean: That there is a benefit for the population does not imply, that every individual user has precisely that benefit. It does not even imply that every user has a benefit at all. In extreme case, a small subgroup could be negatively affected by the novel design, but this could still result in a positive result on average. In the evaluation of high-risk devices like infusion pumps concerns about individual performance are real and this is why we designed the study with within-subject conditions, which allows to estimate the same model on population level and participant level. The following code produces a *multi-level descriptive model*. First, a summary on participant level is calculated, then it is summarized to obtain the population

level. By putting both summaries into one figure, we are doing a multi-level analysis.

```
T_Part <-
  D_pumps %>%
  group_by(Part, Design) %>%
  summarize(mean_Part = mean(ToT))

T_Pop  <-
  T_Part %>%
  group_by(Design) %>%
  summarize(mean_Pop = mean(mean_Part))
```

```
gridExtra::grid.arrange(nrow = 1,
             T_Pop %>%
  ggplot(aes(x = Design, group = NA,
             y = mean_Pop)) +
  geom_point() +
  geom_line() +
  ggtitle("Population-level model (average benefits)") +
  ylim(0, 60),
 T_Part %>%
  ggplot(aes(x = Design,
             y = mean_Part,
             group = Part, label = Part)) +
  geom_line() +
  ggrepel::geom_label_repel(size = 3, alpha = .5) +
  ggtitle("Participant-level model (individual benefits)") +
  ylim(0, 60))
```



Note

- how with `gridExtra::grid.arrange()` we can multiple plots into a grid, which is more flexible than using facetting
- that `ggrepel::geom_label_repel` produces non-overlapping labels in plots

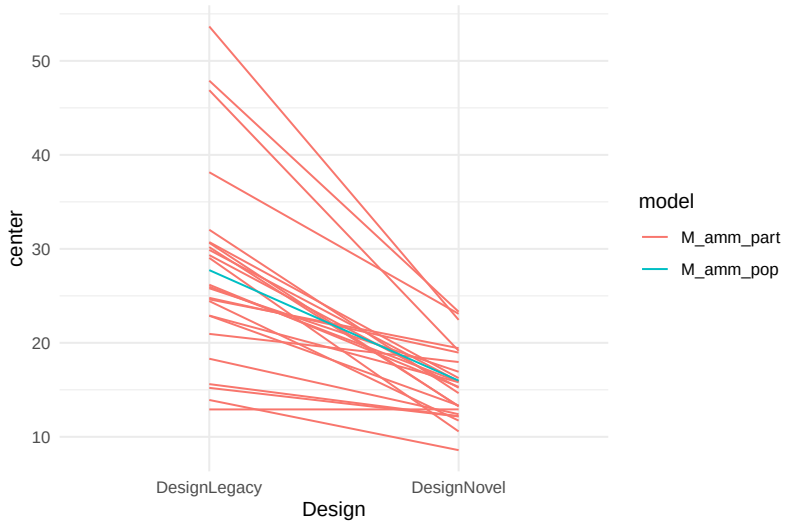
This is a full multi-level analysis, as it shows the same effect on two different levels *alongside*. In this case, the participant-level part removes all worries about the novel design. With the one small exception of participant 3, all users had net benefit from using the novel design and we can call the novel design universally better. In addition, some users (4, 8 and 9) seem to have experienced catastrophes with the legacy design, but their difficulties disappear when they switch to the novel design.

If you look again at the participant-level *spaghetti plot* and find it similar to what you have seen before, you are right: This is an design-by-participant *conditional plot*. Recall, that conditional effects represent the change of outcome, depending on another factor. In this multi-level model, this second factor simply Part(icipant). That suggests that it is well within reach of plain linear models to estimate design-by-participant conditional effects. Just for the purpose of demonstration, we can estimate a population level model, conditioning the design effect on participants. Ideally, we would use a parametrization giving us separate Intercept and DesignNovel effects per participant, but the formula interface is not flexible enough and we would have to work with dummy variable expansion. Since this is just a demonstration before we move on to the multi-level formula extensions, I use an AMM instead. A plain linear model can only hold one level at a time, which is why we have to estimate the

two separate models for population and participant levels. Then we merge the posterior objects, produce a combined CLU table for plotting.

```
M_amm_pop <-
  D_pumps %>%
  stan_glm(ToT ~ 0 + Design, data = ., iter = 50, chains = 2)

M_amm_part <-
  D_pumps %>%
  stan_glm(ToT ~ (0 + Design):Part, data = ., iter = 50, chains = 2)
```

```
T_amm <-
  bind_rows(posterior(M_amm_pop),
            posterior(M_amm_part)) %>%
  fixef() %>%
  separate(fixef, into = c("Design", "Part"))


T_amm %>%
  ggplot(aes(x = Design, y = center, group = Part, color = model)) +
  geom_line()
```



In the first place, the convenience of (true) multi-level models is that both (or more) levels are specified and estimated as one model. For the multi-level models that follow, we will use a specialized engine, `brm()` (generalized multi-level regression) that estimates both levels simultaneously and produce multi-level coefficients. The multi-level CGM we desire is written like this:

```
M_mlcgm <-
  D_pumps %>%
  brm(ToT ~ 1 + Design + (1 + Design|Part), data = ., iter = 100)
```

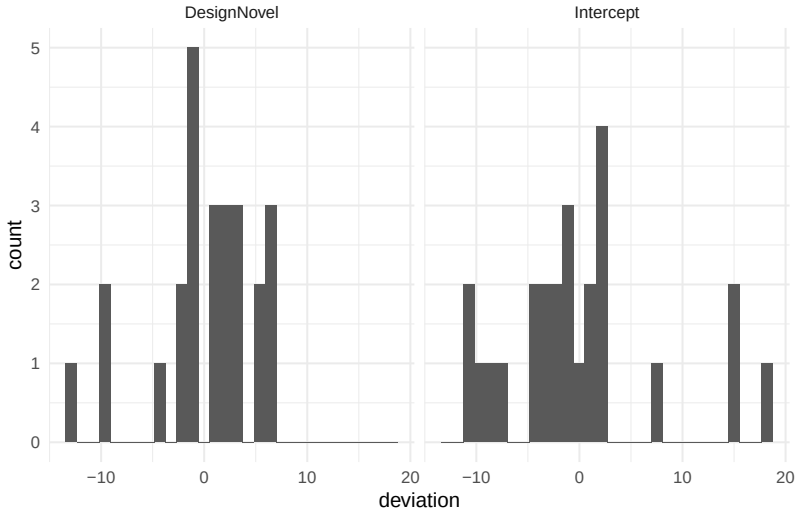
In the formula of this multi-level CGM the predictor term (`1 + Design`) is just copied. The first instance is the usual population-level averages, but the second is on participant-level. The `|` operator in probability theory means "conditional upon" and here this can be read as *effect of Design conditional on participant*.

For linear models we have been using the `coef()` command to extract all coefficients. Here it would extract all coefficients on both levels. With multi-level models, two specialized command exist to separate the levels: we can extract population-level effects using the `fixef()` command (for "fixef effects"). All lower level effects can be accessed with the `ranef` command, which stands for *random effects*. Here, the population level coefficients are absolute means, whereas random effects are *not*. Usually, random effects are *differences towards the population-level*. This is why random effects are always *centered at zero*. In the following histogram, the distribution of the DesignNovel random effects are shown. This is how much users deviate from the average effect in the population.

```
fixef(M_mlcgm)
```

fixef	center	lower	upper
Intercept	28.1	24.4	31.6
DesignNovel	-12.1	-15.0	-8.3

```
ranef(M_mlcgm) %>%
  rename(Part = re_entity, `deviation` = center) %>%
  ggplot(aes(x = deviation)) +
  facet_grid(~fixef) +
  geom_histogram()
```



The distribution of random effects should resemble a *Gaussian distribution*. It is usually hard to tell with such small sample sizes, but it seems that the Intercept effects have a left skew. As we will see in chapter 6, this problem is not surprisung and can be resolved. The distributions are also centered at zero, which is not a coincidence, but the way random effects are designed: deviations from the population mean. That opens up two interesting perspectives: first, random effects look a lot like residuals 4.1.2, and like those we can summarize a random effects vector by its *standard deviation*, using the `grpef` command from package Bayr.

```
bayr::fixef_ml(M_mlcgm)
```

fixef	center	lower	upper	SD_Part
Intercept	28.1	24.4	31.6	8.45
DesignNovel	-12.1	-15.0	-8.3	6.07

Most design research is located on the population level. We want to know how a design works, broadly. Sometimes, stratified samples are used to look for conditional effects in (still broad) subgroups. Reporting individual differences makes little sense in such situations. The standard deviation summarizes individual differences and can be interpreted the *degree of diversity*. The command `bayr::fixef_ml` is implementing this by simply attaching the standard deviation center estimates to the respective population-level effect. As coefficients and standard deviations are on the same scale, they can be compared. Roughly speaking, a two-thirds of the population is contained in an interval *twice as large* as the SD.

```
fixef_ml(M_mlcgm)
```

fixef	center	lower	upper	SD_Part
Intercept	28.1	24.4	31.6	8.45
DesignNovel	-12.1	-15.0	-8.3	6.07

That having said, I believe that more researchers should watch their participant levels more closely. Later, e will look at two specific situations: psychometric models have the purpose of measuring individuals 5.8 and those who propose universal theories (i.e., about people *per se*) must also show that their predictions hold for each and everyone 5.4.

```
detach(IPump)
```

5.4 Testing universality of theories

Often, the applied researcher is primarily interested in a population-level effect, as this shows the *average* expected benefit. If you run a webshop, your returns are exchangeable. One customer lost can be compensated by gaining a new one. In such cases, it suffices to report the random effects standard deviation. If user performance varies strongly, this can readily be seen in this one number.

In at least two research situations, going for the average is just not enough: when testing hazardous equipment and when testing theories. In safety critical research, such as a medical infusion pump, the rules are different than for a webshop. The rules are non-compensatory, as the benefit of extreme high performance on one patient cannot compensate the costs associated with a single fatal error on another patient. For this asymmetry, the design of such a system must enforce a *robust* performance, with no catastrophes. The multilevel analysis of the infusion pumps in 5.3 is an example. It demonstrated that practically all nurses will have a benefit from the novel design.

The other area where on-average is not enough, is theory-driven experimental research. Fundamental behavioural researchers are routinely putting together theories on The Human Mind and try to challenge these theories. For example the Uncanny Valley effect **??**: one social psychologist's theory could be that the Uncanny Valley effect is caused by religious belief, whereas a cognitive psychologist could suggest that the effect is caused by a category confusion on a fundamental processing level (seeing faces). Both theories make universal statements, about all human beings. *Universal statements* can never be proven, but can be is tested by finding counter-evidence. If there is one participant who is provenly non-religious, but falls into the Uncanny Valley, our social psychologist would be proven wrong. If there is a single participant at

all, who does not fall for the Uncanny Valley, the cognitive psychologist was wrong.

Obviously, this counter-evidence can only be found on participant level. In some way, the situation is analog to robustness. The logic of universal statements is that they are false if there is one participant who breaks the pattern, and there is no compensation possible. Unfortunately, the majority of fundamental behavioural researchers, have ignored this simple logic and still report population-level estimates when testing universal theories. In my opinion, all these studies should not be trusted, before a multi-level analysis shows that that the pattern exists on participant level.

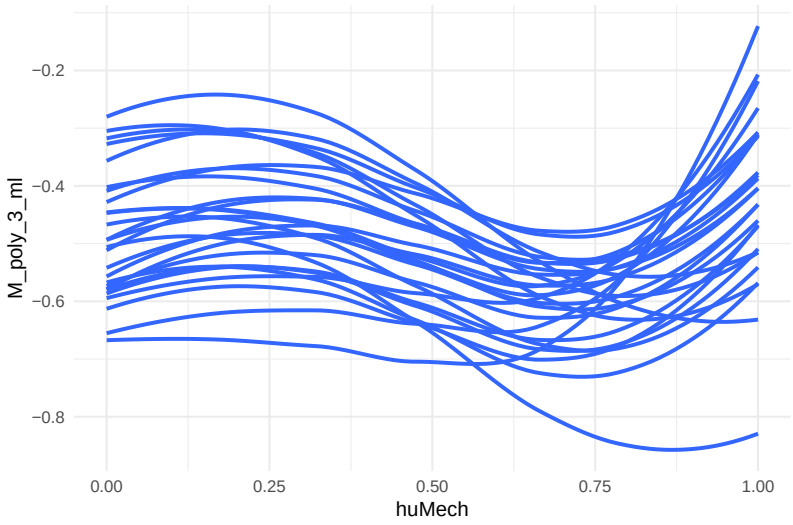
In 4.6, the Uncanny Valley effect has been demonstrated on population level. This is good enough, if we just want to confirm the Uncanny Valley effect as an observation, something that frequently happens, but not necessarily for everyone. The sample in our consisted of mainly students and their closer social network. It is almost certain, that many of the tested persons were religious and others were atheists. If the religious-attitude theory is correct, we would expect to see the Uncanny Valley in several participants, but not in all. If the category confusion theory is correct, we would expect all participants to fall into the valley. The following model performs the polynomial analysis as before 4.6, but multi-level:

```
attach(Uncanny)
```

```
M_poly_3_ml <-
  RK_1 %>%
  brm(response ~ 1 + huMech1 + huMech2 + huMech3 +
                (1 + huMech1 + huMech2 + huMech3|Part),
              data = ., iter = 2500)

P_poly_3_ml <- posterior(M_poly_3_ml)

PP_poly_3_ml <- post_pred(M_poly_3_ml, thin = 5)
```

One method for testing universality is to extract the fitted responses (`predict`) and perform a visual examination: can we see a valley for every participant?

```
T_pred <-
  RK_1 %>%
  mutate(M_poly_3_ml = predict(PP_poly_3_ml)$center)

T_pred %>%
  ggplot(aes(x = huMech, y = M_poly_3_ml, group = Part)) +
  geom_smooth(se = F)
```

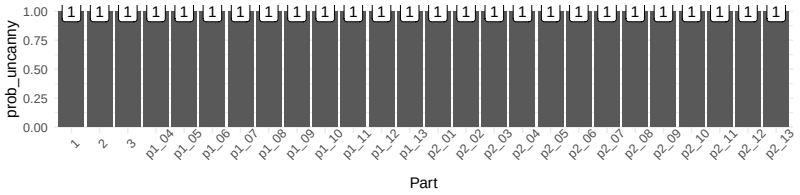


This spaghetti plot broadly confirms, that all participants experience the Uncanny Valley. For a more detailedanalysis, a facetted plot would be better suited, allowing to inspect the curves case-by-case. We proceed directly to a more formal method of testing universality: In **??** we have seen how the posterior distributions of shoulder and trough can be first derived and then used to give a more definitive answer on the shape of the polynomial. It was argued that the unique pattern of the Uncanny Valley is to have a shoulder left of a trough. These two properties can be checked by identifying the stationary points. The proportion of MCMC iterations that fulfill these properties can is evidence that the effect exists. For testing universality of the effect, we just have to run the same analysis on participant-level. Since the participant-level effects are deviations from the population-level effect, we first have to add the population level effect to the random effects (using the Bayr command `re_scores`), which creates absolute polynomial coefficients. The two command `trough` and `shoulder` from package Uncanny ((github.com/schmettow/uncanny)[http://github.com/schmettow/uncanny]) require a matrix of coefficients, which is done by spreading out the posterior distribution table.

```
T_univ_uncanny <-
  P_poly_3_ml %>%
  re_scores() %>%
  select(iter, Part = re_entity, fixef, value) %>%
  tidyr::spread(key = "fixef", value = "value") %>%
  select(iter, Part, huMech0 = Intercept, huMech1:huMech3) %>%
  mutate(trough = uncanny::trough(select(.,huMech0:huMech3)),
         shoulder = uncanny::shoulder(select(.,huMech0:huMech3)),
```

```
          has_trough = !is.na(trough),
          has_shoulder = !is.na(shoulder),
          shoulder_left = trough > shoulder,
          is_uncanny = has_trough & has_shoulder & shoulder_left)
```

```
T_univ_uncanny %>%
  group_by(Part) %>%
  summarize(prob_uncanny = mean(is_uncanny),
            prob_trough = mean(has_trough),
            prob_shoulder = mean(has_shoulder)) %>%
  ggplot(aes(x = Part, y = prob_uncanny)) +
  geom_col() +
  geom_label(aes(label = prob_uncanny)) +
  theme(axis.text.x = element_text(angle = 45))
```



The above plot shows the probability that a participant experiences the Uncanny Valley, as defined by polynomial stationary points. Everyone in our sample experienced the Uncanny Valley effect. Every single MCMC step (remember, every step is a complete polynomial) is positive. This complete absence of counter-evidence may raise suspicions. If this is all based on a random walk, we should at least see a few deviations. The reason for that is simply that the number of MCMC runs puts a limit on the resolution. If we increase the number of iterations enough, we would eventually see few deviant sample drop and measure the tiny chance that a participant does not fall for the Uncanny Valley.

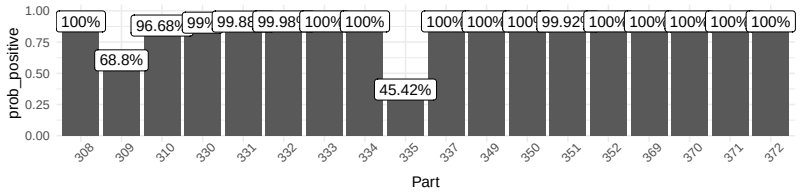
```
detach(Uncanny)
```

While this is great news for all scientists who think that the Uncanny Valley effect is innate (rather than cultural), it does not demonstrate the actual identification of deviant participants. Therefore, we briefly re-visit case Sleepstudy, for which we have estimated a multi-level linear regression model to render individual detoriation of reaction time as result of sleep deprivation. By visual inspection, we identified a single deviant participant who showed an improvement over time. However, the fitted lines a based on point estimates, only (the median of the posterior). Using the same technique as above, it

is possible to calculate the participant-level probabilities for the slope being positive, as expected.

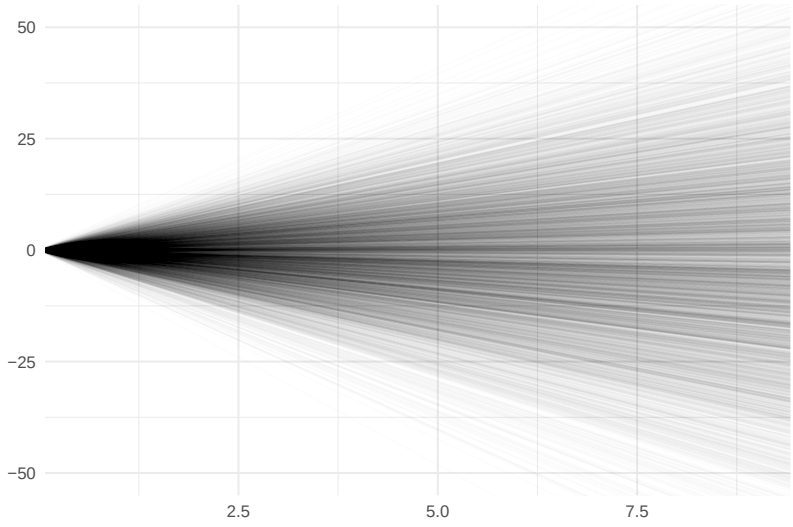
```
attach(Sleepstudy)
```

```
P_scores <-
  posterior(M_slpsty_1) %>%
  re_scores() %>%
  mutate(Part = re_entity)

P_scores %>%
  filter(fixef == "days") %>%
  group_by(Part) %>%
  summarize(prob_positive = mean(value >= 0)) %>%
  mutate(label = str_c(100 * round(prob_positive, 4), "%")) %>%
  ggplot(aes(x = Part, y = prob_positive)) +
  geom_col() +
  geom_label(aes(label = label), vjust = 1) +
  theme(axis.text.x = element_text(angle = 45))
```



All, but participants 309 and 335 almost certainly have positive slopes, experiencing a detoriation of reaction time. Participant 335 we had identified earlier by visual inspection. Now, that we account for the full posterior distribution, it seems a little less suspicious. Basically, the model is almost completely undecisive whether this is a positive and negative effect. The following plot shows all the possible slopes the MCMC random walk has explored. While participant 335 clearly is an outlier, there is no reason to get too excited and call him or her a true counter-example from the rule that sleep deprivation reduced performance.

```
P_scores %>%
  filter(Part == 335, fixef == "days") %>%
  ggplot() +
  xlim(0.5, 9) +
  ylim(-50, 50) +
  geom_abline(aes(intercept = 0, slope = value), alpha = .01)
```




```
detach(Sleepstudy)
```

That being said, the method of posterior-based test statistics can also be used for *analysis of existence*. In the Sleepstudy case a hypothetical question of existence would be that there exist persons who are completely unsensitive to sleep deprivation. Why not? Recently, I saw a documentary about a guy who could touch charged electric wires, because due to a rare genetic deviation, his skin had no sweat glants. Universal statements can only be falsified by a counter-example. Statements of existence can be proven by just a single case. For example, in the 1980 dyslexia became more widely recognized as a defined condition. Many parents finally got an explanation for the problems their kids experienced in school. Many teachers complained that many parents would just seek cheap excuses for their lesser gifted offsprings. And some people argued that dyslexia does not exist and that the disability to read is just a manifestation of lower intelligence. According to the logic of existence, a single person with otherwise good functioning, but slow in learning how to read suffices to proof dyslexia. These have been found in the meantime.

5.5 Non-human populations and cross-overs

With multi-level models design researchers can examine how a design affects the population of users as a whole, as well as on individual level. If there is little variation between users, it can be concluded that the effect is uniform in the population of users. In this section we will generalize the term *population*

and extend the application of multi-level modelling to other types of research entities, such as designs, questionnaire items and tasks.

Many studies in, what one could call *fundamental design research* seek to uncover general laws of design that may guide future system development. Fundamental design research is not concerned with choosing between individual designs, whether they are good enough or not, but with seperating the population of possible designs into good ones and bad ones by universal statements, such as "For informational websites, broad navigation structures are better than deep ones". Note how this statement speaks of designs (not users) in an unspecified plural. It is framed as a universal law for designs.

Comparative evaluation studies, such as the IPump case, are not adequate to answer such questions, simply because you cannot generalize from a sample of two to all possible designs. This is only possible under strict constraints, namely that the two designs under investigation only differ in one design property. For example two versions of a website present the same web content in a deep versus a wide hierarchy, but layout, functionality are held constant. And even then, we should be very careful with conclusions, because there can be interaction effects. For example, the rules could be different for a website used by expert users.

If the research question is universal, i.e. aiming at general conclusions on all designs (of a class), it is inevitable to see designs as a population from which we collect a sample. The term population suggests a larger set of entities, and in fact many application domains have an abundance of existing designs and a universe of possible designs. Just to name a few: there exist dozens of note taking apps for mobile devices and hundreds of different jump'n'run games. Several classes of websites count in the ten thousands, such as webshops and municipal websites.

Whereas we can define classes in any way and for everything we want, the term population, in a statistical not biological sense, has a stronger implication. A population contains individuals and these individuals vary only to some extent. At the same time, it is implied that we can identify some sort of typical value for a population, such that most individuals are clumped around this typical value. Essentially, if it looks similar to one of the basic statistical distributions 3.4.2, we can call it a population. To illustrate the difference between a class and a population. *Vehicles* are a class of objects that transport people or goods. This broad definition covers many types of vehicles, including bicycles, rikshas, cars, buses, trucks and container vessels. If the attribut under question is the weight, we will see a distribution spreading from a 10 kilograms up to 100 tons. That is a a scale of 1:10.000 and the distribution would spread out like butter on a warm toast. Formally, we can calculate the average weight of a vehicle, but that would in no way be a typical value.

It does not stop here. While design research is clearly about the user-design encounter, other research entities exist that we could call a population. The

most routinely used *non-human populations* in design research besides *designs* are *tasks*, *situations* and *questionnaire items*. We briefly characterize the latter three and proceed to a case study that employs three populations at once.

Tasks: Modern informational websites contain thousands of information pieces and finding every one of those can be considered a task. At the same time, it is impossible to cover them all in one study, such that sampling a few is inevitable. We will never be able to tell the performance of every task, but using random effects it is possible to estimate the variance of tasks. Is that valuable information? Probably it is in many cases, as we would not easily accept a design that prioritizes on a few items at the expense of many others, which are extremely hard to find.

Situations: With the emerge of the web, practically everyone started using computers to find information. With the more recent breakthrough in mobile computing, everyone is doing everything using a computer in almost every situation. People chat, listen to music and play games during meals, classes and while watching television. They let themselves being lulled into sleep, which is tracked and interrupted by smart alarms. Smartphones are being used on trains, while driving cars and bikes, however dangerous that might be, and not even the most private situations are spared. Does the usability of messaging, navigation and e-commerce apps generalize across situations? Again, a study to examine performance across situations would best sample from a population of situations.

Questionnaire items: Most design researchers cannot refrain from using questionnaires to evaluate certain elucive aspects of their designs. A well constructed rating scale consists of a set of items that trigger similar responses. At the same time, it is desireable that items are unsimilar to a degree, as that establishes good discrimination across a wide range. In ability tests, for example to assess people's intelligence or math skills, test items are constructed to vary in difficulty. The more ability a person has, the more likely will a very difficult item be solved correctly. In design research, rating scales cover concepts such as perceived mental workload, perceived usability, beauty or trustworthiness. Items of such scales differ in how extreme the proposition is, like the following three items that could belong to a scale for aesthetic perception:

1. The design is not particularly ugly.
2. The design is pretty.
3. The design is a piece of art.

For any design it is rather difficult to get a positive response on item 3, whereas item 1 is little of a hurdle. So, if one thinks of all possible propositions about beauty, any scale is composed of a sample from the population of beauty propositions.

If we look beyond design research, an abundance of non-human populations can be found in other research disciplines, such as:

- products in consumer research
- phonemes in psycholinguistics
- test items in psychometrics
- pictures of faces in face recognition research
- patches of land in agricultural studies

In all these cases it is useful (if not inevitable) to ask multi-level questions not just on the human population, but on the encounter of multiple populations. In research on a single or few designs, such as in A/B testing, designs are usually thought of (and modelled) as common fixed-effects factors. However, when the research question is more fundamental, such that it regards a whole class of designs, it is more useful to think of designs as a population and draw a sample. In the next secion we will see an example, where a sample of users encounters a sample of designs and tasks.

In experimental design research, the reserach question often regards a whole class of designs and it inevitable (although often done wrong) to view designs as a population. As we usually want to generalize across users, that is another sample. A basic experimental setup would be to have every user rate (or do a task) on every design, which is called a complete (experimental) design, but I prefer to think of it as a complete *encounter*.

Every measure is an encounter of one participant and one design. If a multi-item rating scale is used, measures are an encounter between three populations. Every measure combines the impact from three members from these populations. With a single measure, the impact factors are inseparable. But if we have many measures, we can apply a *cross-classified multi-level model* (CRMM). An intercept-only CRMM just adds intercept random effects for every population.

As we will see in 5.8.4, the individual random coefficients of a CRMM can be used for psychometric evaluation of rating scales. In the following example, the question is a comparison of diversity across populations. Three decades ago, [%Dennis Egan] published one of the first papers on individual differences in computer systems design and made the following claim:

> 'differences among people usually account for much more variability in performance than differences in system designs' [1]

[1] Egan, D. Individual differences in human-computer interaction. In M. Helander, ed., Handbook of Human Computer interaction. Elsevier Science Publishers, Amsterdam, The Netherlands, 1988, 543–568.

What is interesting about this research question is that it does not speak about effects, but about *variability of effects* and seeks to compare variability of two totally different populations. In the following we will see how this claim can be tested by measuring multiple encounters between designs and users, apply an intercept-only CRMM and compare the random factor standard deviations.

Egan's claim has been cited in many papers that regarded individual differences and we were wondering how it would turn out in the third millenium, with the probably most abundant of all application domains: informational websites. For the convenience, we chose the user population of student users, with ten unversity websites as our design sample. Furthermore, ten representative tasks on such websites were selected, which is another population. During the experiment, all 41 participants completed 10 information search items such as:

> On website [utwente.nl] find the [program schedule Biology].

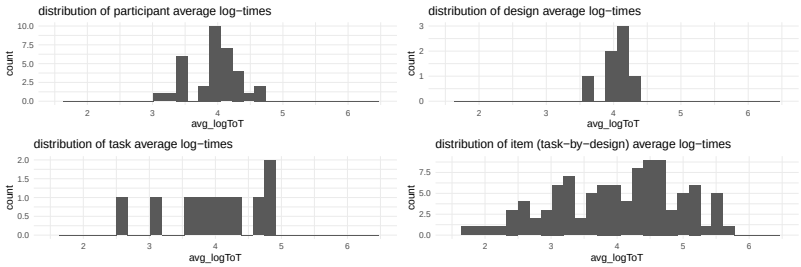
```
attach(Egan)
```

```
D_egan <- D_egan %>% mutate(logToT = log(ToT))
D_egan %>% as_tbl_obs()
```

Note that ToT has been log-transformed for compliance with the assumptions of Linear Models. Generally, the advice is to use a Generalized Linear Model instead 6.3.2.

Egan's claim is a two-way encounter to which we added the tasks as a third population. However, our data seemed to require a fourth random effect, which essentially is an interaction effect between tasks and websites: how easy a task is, largely depends on the webite where it is carried out. For example, one university website could present the library on the homepage, whereas another websites hides it deep in its navigation structure. The following grid of histogram shows the marginal distributions of human and non-human populations. The individual plots were created using the following code template:

```
  D_egan %>%
    group_by(Part) %>%
    summarize(avg_logToT = mean(logToT)) %>%
    ggplot(aes(x = avg_logToT)) +
    geom_histogram() +
    labs(title = "distribution of participant average log-times") +
    xlim(1.5,6.5)
```

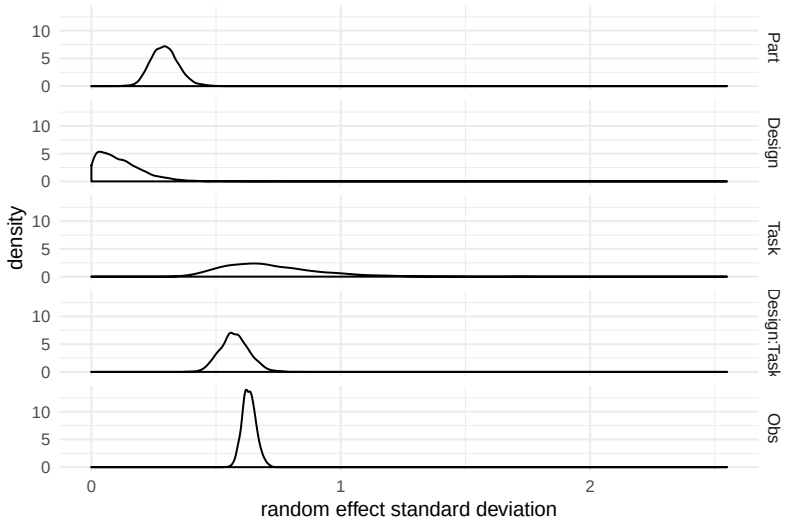


There seems to be substantial variation between participants, tasks and items, but very little variation in designs. We build a GMM for the encounter of the four populations.

```
M_1 <- D_egan %>%
  brm(logToT ~ 1 + (1|Part) + (1|Design) + (1|Task) + (1|Design:Task),
              data = .)

P_1 <-  posterior(M_1)
```

A Bayesian multi-level model estimates the standard deviation alongside with coefficients, such that we can compare magnitude and certainty of variability. In addition, we can always compare a random factor standard deviation to the standard error.

```
P_1 %>%
  filter(type == "grpef"  | type == "disp") %>%
  mutate(re_factor = if_else(type == "disp", "Obs", re_factor),
         # re_factor = factor(re_factor,
         #                    levels = c("Obs", "Part", "Design", "Task", "Design:Task")),
         re_factor = mascutils::reorder_levels(re_factor, c(4, 2, 5, 3, 1))) %>%
  ggplot(aes(x = value)) +
  geom_density() +
  labs(x = "random effect standard deviation") +
  facet_grid(re_factor~.)
```



The outcomes of our study are indecisive regarding Egan's claim. Variance of participants is stronger thnan variance of designs, but not by much. Both factors also produce much less variability in measures than does the noise, which we here regard a observation-level random effect. Tasks seem to have the overall strongest effect, but this comes with huge uncertainty. The strongest variability is found in the sample of Design-Task pairs, which is interesting

A secondary observation on the posterior plot is that some effects are rather certain, such as Obs and Design:Task, whereas others are extremely uncertain, especially Task. There is a partial explaination for this: the variation is estimated from the "heads" in thehuman or non-human population. It therefore strongly depends on respective sample size. Design and Task have a meager $N = 10$, which is why the estimate are so uncertain. With $N = 41$ the participant level estimate has more data and reaches better certainty, same for the pairs ($N = 100$). The observations level can employ to all 410 observations, resulting in a highly certain estimate.

We proceed with a formal check of Egans claim, using the same technique as in **??** and **??**. What is the probability, that Egan is right? We create a Boolean variable and summarize the proportion of MCMC draws, where $\sigma_{\text{Part}} > \sigma_{\text{Design}}$ holds.

```
C_Egan_is_right <-
  P_1 %>%
  filter(type == "grpef", re_factor %in% c("Part", "Design")) %>%
  select(chain, iter, re_factor, value) %>%
  spread(re_factor, value) %>%
  summarize(Prop_Egan_is_right = mean(Part > Design)) %>%
  c()
```

```
C_Egan_is_right
```

```
## $Prop_Egan_is_right
## [1] 0.944
```

A chance of 0.94375 can count as good evidence in favour of Egan's claim, although it certainly does not match the "much more" in the original quote. However, if we take the strong and certain Design:Task effect into account, the claim could even be reversed. Apparently, the difficulty of a task depends on the design, that means, it depends on where this particular designer has placed an item in the navigation structure. That clearly is a design feature and therefore counts as strong counter-evidence.

```
detach(Egan)
```

5.6 Nested random effects

In some research designs, we have populations, where every member contains a population by itself. A classic example is from educational research: a sample of schools is drawn and inside every school a sample of students is selected. Like cross-classified models, nested models consist of multiple levels. The difference is that if one knows the lowest (or: a lower) level of an observation, the next higher level is unambiguous, like:

- every student is in exactly one class
- every participant is from exactly one professional group

Nested random effects (NRE) represent nested sampling schemes. As we have seen above, cross-classified models play an important role in design research, due to the user/task/design encounter. NREs are more common in research disciplines where organisation structures or geography plays a role, such as education science (think of the international school comparison studies PISA and TIMMS), organisational psychology or political science. One examples of a nested sampling structure in design research is the CUE8 study, which is the eighth instance of Comparative Usability Evaluation (CUE) studies by Rolf Molich [%CUE8]. Different to what the name might suggest, not designs are under investigation in CUE, but usability professionals. The over-arching question in the CUE series is the performance and reliability of usability professionals when evaluating designs. Earlier studies sometimes came to devastating results regarding consistency across professional groups when it comes to identifying and reporting usability problems. We always knew, qualitative

analysis is much harder to do in an objective way, did we not? The CUE8 study lowered the bar, by asking whether several professional groups will arrive at consistent measures for time-on-task.

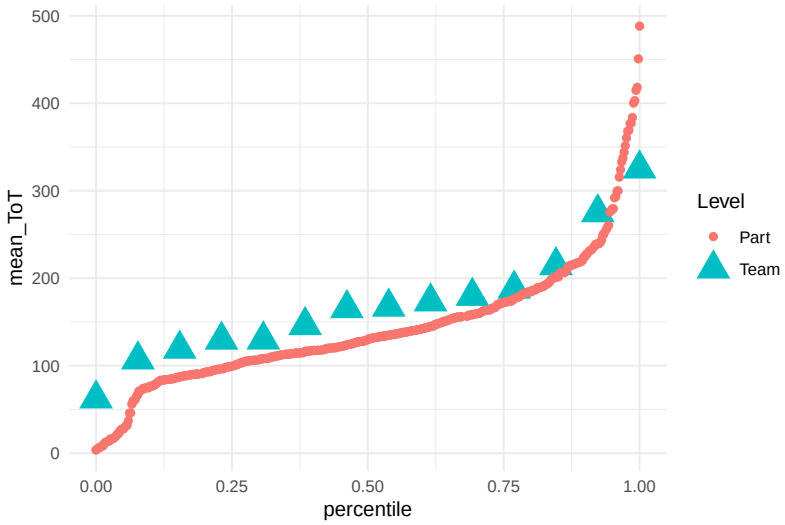
The CUE8 study measured time-on-task in usability tests, which had been conducted by 14 different teams. The original research question was: How reliable are time-on-task measures across teams? All teams used the same website (a car rental company) and the same set of tasks. All teams did moderated or remote testing (or both) and recruited their own sample of participants. So, the analysis can performed on three levels: the population level would tell us the overall performance on this website. That could be interesting for the company running it. Below that are the teams and asking how they vary is the primary research question. At the same time, participants make another level of analysis. Because every participant is assigned to exactly one team, we can call this a nested situation. If that weren't the case, say there is one sample of participants shared by the teams, that would be just cross-classification.

As the original research question is on the consistency across teams, we can readily take the random effect variance as a measure for the opposite: when variance is high, consistency is low. But, how low is low? It is difficult to come up with an absolute standard for inter-team reliability. Because we also have the participant-level, we can resort to a relative standard: how does the variation between teams compare to variation between individual participants?

Under this perspective, we examine the data. This time, we have real time-on-task data and as so often, it is highly skewed. Again, we use the trick of logarithmic transformation to obtain a more symmetric distribution of residuals. The downside is that the outcome variable may not be zero. For time-on-task data this is not an issue. In fact, the original CUE8 data set contains several observations with unrealistically low times. Before proceeding to the model, we explore the original variable `ToT` on the two levels (Participant and Team): In the following code the mean ToT is computed for the two levels of analysis, participants and teams and shown in ascending order.

```
attach(CUE8)

D_cue8

D_part_mean <-
  D_cue8 %>%
  group_by(Part, Condition) %>%
  summarize(mean_ToT = mean(ToT), na.rm = T,
            n_Obs = n()) %>%
  ungroup() %>%
  rename(Unit = Part) %>%
```

```
  mutate(percentile = percent_rank(mean_ToT),
         Level = "Part")

D_team_mean <-
  D_cue8 %>%
  group_by(Team, Condition) %>%
  summarize(mean_ToT = mean(ToT, na.rm = T),
            n_Obs = n()) %>%
  ungroup() %>%
  rename(Unit = Team) %>%
  mutate(percentile = percent_rank(mean_ToT),
         Level = "Team")

D_task_mean <-
  D_cue8 %>%
  group_by(Task, Condition) %>%
  summarize(mean_ToT = mean(ToT, na.rm = T),
            n_Obs = n()) %>%
  ungroup() %>%
  rename(Unit = Task) %>%
  mutate(percentile = percent_rank(mean_ToT),
         Level = "Task")


bind_rows(D_team_mean,
          D_part_mean) %>%
  ggplot(aes(x = percentile,
             y = mean_ToT,
             col = Level,
             shape = Level,
             size = Level)) +
  geom_point()
```



It seems there is ample variation in ToT for participants, with mean ToT ranging from below 100 to almost 500 seconds. There also is considerable variation on team level, but the overall range seems to be a little smaller. Note, however, that the participant level contains all the variation that is due to teams. A model with nested random effects can separate the sources of variation. When two (or more) levels are nested, a special syntax applies for specifying nested random effects. `1|Team/Part`.

```
M_1 <-
  D_cue8 %>%
  brm(logToT ~ Condition + (1|Team/Part),
      data = .)
# logToT ~ Condition + 1|Part/Team

P_1 <- posterior(M_1)
```

Note that the model contains another feature of the CUE8 study, as participants were tested in one of two conditions: moderated and remote testing sessions. As participants are strictly divided into these groups, a participant-level random effect is not useful.

```
P_1
```

** tbl_post: 4000 samples in 4 chains

model	parameter	type	fixef	re_factor	entities
M_1		ranef	Intercept	Team	14
M_1		ranef	Intercept	Team:Part	523
M_1	b_Conditionmoderated	fixef	Conditionmoderated	NA	1
M_1	b_Intercept	fixef	Intercept	NA	1
M_1	sd_Team:Part___Intercept	grpef	Intercept	Team:Part	1
M_1	sd_Team___Intercept	grpef	Intercept	Team	1

The posterior object reveals two random factors, one for teams and one for participants. The coefficients are in no way different to cross-classified random effects. In both cases, the absolute group mean for a certain participant is obtained by adding up all two coefficients. The syntax really is just a safe way to deal with nested samples, where participant identifiers are re-used within teams. Nothing stops us from doing the following:

```
D_cue8 %>%
  mutate(Part = str_c(Team, Part, sep = "_")) %>%
  brm(logToT ~ Condition + (1|Part) + (1|Team), data = .)
```

```
##  Family: gaussian
##   Links: mu = identity; sigma = identity
## Formula: logToT ~ Condition + (1 | Part) + (1 | Team)
##    Data: . (Number of observations: 2486)
## Samples: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
##          total post-warmup samples = 4000
##
## Group-Level Effects:
## ~Part (Number of levels: 523)
##               Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sd(Intercept)     0.43      0.02     0.39     0.47 1.00     1665     2742
##
## ~Team (Number of levels: 14)
##               Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sd(Intercept)     0.64      0.15     0.41     1.01 1.00     1343     2528
##
## Population-Level Effects:
##                    Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## Intercept              4.60      0.26     4.07     5.11 1.01     1588     2045
## Conditionmoderated     0.36      0.36    -0.33     1.08 1.01     1478     1994
##
## Family Specific Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sigma     0.61      0.01     0.59     0.63 1.00     4378     3123
```

```
##
## Samples were drawn using sampling(NUTS). For each parameter, Bulk_ESS
## and Tail_ESS are effective sample size measures, and Rhat is the potential
## scale reduction factor on split chains (at convergence, Rhat = 1).
```

Let's take a closer look at the results regarding consistency of ToT measures across teams. We would always expect participants to show variation, but if team averages show strong variation, then we can suspect that there are biases. It turns out that the variation by team is by a factor of 1.5 larger than individual differences. And it is on par with the measurement error (sigma).

```
P_1 %>%
  filter(type %in% c("fixef", "grpef")) %>%
  clu()
```

parameter	type	fixef	re_factor	center	lower	upper
b_Intercept	fixef	Intercept	NA	4.611	4.059	5.150
b_Conditionmoderated	fixef	Conditionmoderated	NA	0.344	-0.340	1.033
sd_Team__Intercept	grpef	Intercept	Team	0.617	0.415	1.014
sd_Team:Part__Intercept	grpef	Intercept	Team:Part	0.427	0.389	0.469

It is not surprising to see the test users vary greatly in performance. In contrast, the discordance between professional teams is concerning, which even arises after controlling for an experimental factor that could explain the variance, remote or moderated. The opposite seems to be the case. Although one can think of good reasons why these two forms of testing would differ, the difference is rather small (between moderated and remote testing is weak and highly uncertain ($0.34[-0.34, 1.03]_{CI95}$).

```
T_fixef <- fixef(P_1)
T_fixef
```

fixef	center	lower	upper
Intercept	4.611	4.06	5.15
Conditionmoderated	0.344	-0.34	1.03

```
detach(CUE8)
```

Actually, it is not even necessary to specify that participants are nested within teams. If we make sure that participant identifiers are unique across the whole study (not just within a team unit), we can also just use `1|Part + 1|Team`.

If the participant identifier is only unique within a team, it is either to be recoded, e.g. `mutate(Part = str_c(Team, Part))`.

In this section we introduced a new perspective on multi-level models. Here, the question was to quantify and compare samples (rather than conditions) as sources of variation. With multi-level models, we can separate sources of variation and conveniently quantify them as standard errors. This builds on how random effects are constructed, as factor levels drawn from a Gaussian distribution. As this matches how measurement errors are represented. In result, we could separate and compare the three levels of variation: participants, team and residuals. In the following section, we will delve deeper into the matter of random effects.

5.7 What are random effects? On pooling and shrinkage

At least half a dozen of defintions exist for the term random effect. This is so confusing that some authors refrain to use the term altogether. Here, the definition is conceptually based on the idea of a population. Technically, it is compatible with the implementation found in `rstanarm` and other engines. Unfortunately, the very terms *random effect* and *random factor* are highly misleading, as there is nothing more or less random in a random factors as compared to fixed factors. The opposite is the case: as we have seen above, a random effects model pulls a variance component from the random term and explains it by assigning coefficients to entities (teams or users). The best advice is to not contemplate over what makes a factor random. It is just a name and because random factors are so amazingly useful, they should be called fonzy factors, instead.

When a data set contains a factor that we may wish to add to the model, the question is: fixed effect or random effect? Above, I have introduced the heuristic of populations. If one can conceive tasks, designs, or whatever set of items as a population, there is clumping to some degree, but also variation. The more clumping there is, the better is the guess for unobserved members by observing some members. In such a case, the predictor is introduced as a fonzy factor. Now it is time to more formally conceive when a set of things is a population.

Obviously, we would never speak of a population, when the objects of interest are from different classes. Entities gathering on super market parking lots, like persons, cars, baskets and and dropped brochures, we would never see as a population. People, we would generally see as a population, as long as what we want to observe is somewhat homogenous. When the question is, how fast persons can do a 2000 meter run at the olympic games, we would certainly want one population per discipline (swimming, running, etc). Why is that

so? It is because we expect members of a population to have some similarity, with the consequence that, if you already have observed some members of the population, this tells you something about any unobserved members.

Reconsider the Bayesian principle of prior knowledge by an experiment of thought: Consider, a UX expert with experience in e-commerce is asked to estimate how long it takes users to do the checkout, *without being shown the system*. The expert will probably hesitate briefly, and then come up with an estimate of, let's say, 45 seconds. Without any data, the expert made some reasonable assumptions, e.g. that a disciplined design process has been followed, and then relies on experience. The experts personal experience has formed prior to the study by observing many other cases. Now, we confront the expert with an even more bizzare situation: guess the time-on-task for an unknown task with an unseen system of unknown type! The expert will probably refuse to give an answer, arguing that some systems have tasks in the millisecond range (e.g. starting a stopwatch), whereas other processes easily run for hours or days (e.g. doing data exploration). This is agreeable, and we provide the expert with average ToT of four other tasks within the same system:

$ToT_{1-4} = 23, 45, 66, 54$

Now, the expert is confident that ToT be around 50 seconds and that is probably a good guess. What has happened is that prior belief about the unkown task parameter has been formed not externally, but *by data* as it arrived. The likely value of one unit has been learned from the other units and this appears pretty reasonable. The same principle applies when removing outliers. When staring at a boxplot or scatterplot the mind of the observer forms a gestalt that covers the salient features of data, for example: almost all points are located in the range 100 - 500. Once this pattern has formed, deviant points stand out.

However, the salience of the gestalt may vary. Consider a situation where ToT has been measured by the same procedure, but using five different stop watches. Stop watches are so incredibly accurate that if you know one measure, you basically know them all. What many researchers do with repeated measures data, is take the average. This is the one extreme called *total pooling*. In the stopwatch case the average of the five measures would be so highly representative, that total pooling is a reasonable thing to do.

In other cases, the levels of a factor are more or less independent, for example tasks in a complex system, where procedure duration ranges from seconds to hours. Guessing the duration of one task from a set of others is highly susceptible and the average duration across tasks is not representative at all. The best choice then is to see tasks as factor levels, that are independent. This extreme of *no pooling* is exactly represented by fixed effect factors as they have been introduced in 4.

Random effects sit right between these two extremes of no and total pooling and implement *partial pooling*: the more the group mean is representative for the units of the group, the more it is taken into account. The best thing about partial pooling is that, unlike real priors, there is not even the need to determine the amount of pooling in advance. The variation of entities has been observed. The stronger the enities vary, the less can be learned from the group level. The variation is precisely the group-level standard deviation of the random effect. So, we can think of random factors as factors where there is a certain amount of cross-talk between levels. The random effect estimate then draws on two sources of evidence: all data from the overarching population and data that belongs just to this one entity. As that is the case, the exploratory analysis of individual performance in SOV does not resemble a true random effect, as group means were calculated independently.

How are random effects implemented to draw on both sources? Obviously, the procedure must be more refined than just putting dummy variables in a likelihood formula. In the Bayesian framework a remarkably simple trick suffices, and it is even a familiar one. By the concept of prior distributions, we already know a way to restrict the range of an effect based on prior knowledge. For example, intelligence test results have the prior distribution $IQ \sim \text{Gaus}(100, 15)$, just because they have been empirically calibrated this way. In most other cases, we do have rough ideas about the expected magnitude and range in the population, say: healthy human adults will finish a 2000m run in the range of 5-12 minutes.

As prior knowledge is external to the data, it often lacks systematic evidence, with the exception of a meta analyses. This is why we tend to use weak informative priors. Like priors, random effects take into account knowledge external to the entity under question. But, they draw this knowledge from the data, which is more convincing after all. The basic trick to establish the cross-talk between random factor levels, is to *simultaneously estimate factor levels and random factor variation*. This has several consequences:

All random effects get a more or less subtle trend towards the population mean. As a side effect, the random factor variance is usually smaller than variance between fixed factors, or naive group means. This effect is called *shrinkage*. When the random factor variation is small, extreme factor levels are pulled stronger towards the population mean, resulting in stronger shrinkage. Or vice versa: When random variation is large, the factor levels stand more on their own.

The random factor variation is an estimate and as such it is certain only to a degree. As we have seen in 5.5, the more levels a random factor comprises, the more precise is the estimate of random factor variation. The strongest shrinkage occurs with few observations per factor levels and highly certain random factor variation.

Previously, I have stressed how important repeated measures design is, as the number of observations per entity plays a role, too. The more observations there are, the less is the group mean overruled by the population mean. Less shrinkage occurs. This is why multi-level models gracefully deal with imbalanced designs. Groups with more observations are just gradually more self-determined. Taking this to the opposite extreme: when a factor level contains no data at all, it will just be replaced by the population mean. This principle offers a very elegant solution to the problem of missing data. If you know nothing about a person, the best guess is the population mean.

Under the perspective of populations as a more or less similar set of entities, these principles seem to make sense. Within this framework, we can even define what fixed effects are:

> a fixed effect is a factor where levels are regarded so unsimilar, that the factor-level distribution can be practically considered infinite.

We routinely approximate such a situation when using non-informative prior distributions, like β_0 Gaus$(0, 10000)$. In the extreme case, a uniform distribution with an infinite upper boundary truly has an infinite variance: $\beta_0\, U(0, \infty)$. A finite population mean doesn't even exist with such a distribution.

So, when a design researcher has observed that with design A, ToT is approximately distributed as ToT_A Gaus$(120, 40)$, is it realistic to assume that design B has ToT in the range of several hours? Would a cognitive psychologist see it equally likely that the difference in reaction time on two primitive tasks is 200ms or 2h? Probably not. Still, using fixed effects for factors with very few levels is a justifyable approximation. First of all, priors can be used at any time to factor in reasonable assumptions about the range. Second, with very few estimates, the random factor variation cannot be estimated with any useful certainty. Very small shrinkage would occur and the results would practically not differ.

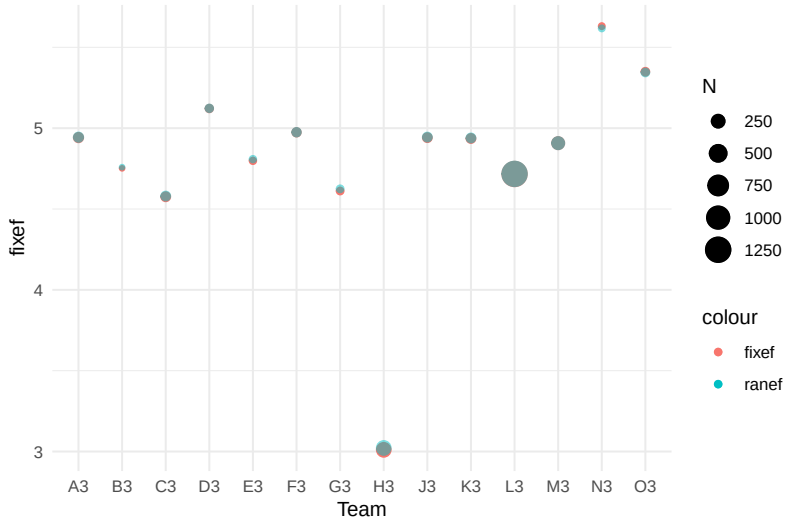
The CUE8 study makes a case for seeing shrinkage in action: Teams of researchers were asked to conduct a performance evaluation on a website. Tasks and website were the same, but the teams followed their own routines. Some teams tested a few handful of participants, whereas others tested dozens remotely. Teams, as another non-human population (sic!) differ vastly in the number of observations they collected. We can expect differences in shrinkage. To see the effect, we compare the team-level group means as fixed factor versus random factor. All teams have enough participants tested to estimate their mean with some certainty. At the same time, the group sizes vary so dramatically that we should see clear differences in tendency towards the mean.

We estimate two models, a random effects (RE) model and a fixed effects (FE) model. For the RE model, the absolute group means are calculated on the posterior. Figure XY shows the comparison of FE and RE estimates.

```
attach(CUE8)
```

```
M_2 <-
  D_cue8 %>%
  stan_glm(logToT ~ Team - 1, data = .)

M_3 <-
  D_cue8 %>%
  brm(logToT ~ 1 + (1|Team), data = ., iter = 100)
```

```
P_2 <- posterior(M_2, type = "fixef")
P_3_fixef <-  posterior(M_3, type = "fixef")
P_3_ranef <-  posterior(M_3, type = "ranef")

## Creating a derived posterior with absolute team-level random effects
P_3_abs <-
  left_join(P_3_ranef, P_3_fixef,
    by = c("chain", "iter", "fixef"),
    suffix = c("","_fixef"))

P_3_abs$value <-  P_3_abs$value + P_3_abs$value_fixef


T_shrinkage <-
  D_cue8 %>%
  group_by(Team) %>%
  summarize(N = n()) %>%
  mutate(fixef = fixef(P_2)$center,
         ranef = ranef(P_3_abs)$center,
         diff = fixef - ranef)


sd_fixef <- sd(T_shrinkage$fixef)
sd_ranef <- sd(T_shrinkage$ranef)


T_shrinkage %>%
  ggplot(aes(x = Team, y = fixef, size = N, col = "fixef")) +
  geom_point() +
  geom_point(aes(y = ranef, col = "ranef"), alpha = .5)
```




```
detach(CUE8)
```

Team H is far off the population average, but almost no shrinkage occurs due to the large number of observations. Again, no shrinkage occurs for Team L, as it is close to the population mean, and has more than enough data to speak for itself. Team B with the fewest observation (a genereous $N = 45$, still), gets noticable shrinkage, although it is quite close to the population mean. Overall, the pattern resembles the above properties of random effects: groups that are far off the population mean and have comparably small sample size get a shrinkage correction. In the case of CUE8, these correction are overall negligible, which is due to the fact that all teams gathered ample data. Recall the SOV simulation above, where the set of tasks every user did was beyond control of the researcher. In situations with quite heterogeneous amount of missing data per participant, shrinkage is more pronounced and more information is drawn from the population mean.

At the same time, shrinkage adjusts the estimates for variation, with $sd_{RE} = 0.583 < sd_{FE} = 0.588$. The random effects estimate is an unbiased estimate for the population variance, whereas fixed effects variation would be overestimating. [%REF]

So, random effects are factors with the additional assumption of Gaussian distribution. When a multi-level model is estimated, the population level effect, the random effects levels and the variance of the distribution are estimated simultaneously. This creates two particular advantages of multi-level models with random effects:

1. In unbalanced research designs (with unequal number of observations per subject) small groups are corrected towards the population mean.
2. Strong outliers are corrected towards the population mean.

In conclusion, whereas classical techniques for repeated measures often require additional tweaks to work well with unbalanced designs and outliers, multilevel models with random effects handle those situations gracefully.

5.8 Psychometrics and designometric models

Up to to this point, we have characterized random factors mainly by their variance. However, a random effect is just like a factor, where the levels are typically entities in a sample. In a multi-level model, these levels are represented as coefficients. Every entity gets their own estimate which represents the level of functioning of the entity. These values can either be compared against an external benchmark, or they are compared to each other. When human individuals are being compared this is called *psychometrics*.

Traditionally, psychometrics deals with the valid and reliable measurement of personal characteristics, such as individual levels of performance, motivation, socio-cognitive attitude and the like. Advanced statistical models have been devised, such as confirmatory factor analysis or item response models. Formally, these applications establish an order among population entities, from low to high. The least one is aiming for is that for any pair of entities A and B, we can tell which entity has the more pronounced property. This is called an *ordering structure*. Fewer applications go beyond mere ranking and establish metric interpretations. For example, in an *interval-scaled* test, the researcher may compare differences between participants.

In design research, multi-item validated scales are routinely used for one of two purposes:

1. A design-related research questions involve traits or abilities of users. For example: Do social network users with high Openness have more connections? A six-item test for Openness is used on every individual in the sample and the scores are compared the number of connections. This is the basic *psychometric situation*, which is an *encounter of persons and items*.
2. In design research one frequently assesses properties of a design by using multi-item questionnaires. One example would be the comparison of user experience among a set of e-commerce homepages using scales such as the AttrakDiff (the hedonic dimension). Another example would be to map the uncanny valley effect by letting participants judge a set of artificial faces with a multi-item scale to measure eeriness. Apparently, when the

aim is to measure a design, the situation is an encounter of users, items and designs. We call this the psychometric situation.

We begin with the first case, standard psychometrics to assess user characteristics. For example, one could ask how a persons visual-spatial abilities are related to performance in navigating a complex hypertext environment, exploring body cavities during surgical procedures or monitoring the scattered displays in air traffic control centers.

In the case of visual-spatial ability, the researcher could administer a test for visual-spatial abilities: for example, participants solve a set of mental rotation tasks and reaction times are collected as a score for spatial processing speed; this would later be compared to performance in a real (or simulated) task, let's say the number of times an obstacle is hit in a driving simulator. The straight-forward approach would be to take the average measure per person as a score for mental rotation speed.

```
n_Part <- 20
n_Trial <- 10
n_Obs <- n_Part * n_Trial

D_Part <- tibble(Part = 1:n_Part,
                 true_score = rnorm(n_Part, 900, 80))

D_Trial <- tibble(Trial = 1:n_Trial)

D_CTT <-
  mascutils::expand_grid(Part = D_Part$Part,
                                Trial = D_Trial$Trial) %>%
  left_join(D_Part) %>%
  mutate(RT = rnorm(n_Obs,
                    mean = true_score,
                    sd = 50)) %>%
  mascutils::as_tbl_obs()

D_CTT %>%
  group_by(Part) %>%
  summarize(score = mean(RT))
```

Part	score
1	1039
2	875
3	850
4	788
5	960
6	853
7	797
8	830
9	883
10	887
11	740
12	920
13	834
14	950
15	867
16	965
17	1008
18	1053
19	968
20	814

Note how the table only contains the identifiers, but no additional information. The trials in the mental rotation task are assumed to be exchangeable. This is how the so-called *classical test theory* approach works. In classical test theory, the observed test score y_i for participant i is composed of the the true score of participant i, μ_i, and a Gaussian measurement error ϵ_{ij}.

$$y_{ij} = \mu_i + \epsilon_{ij}$$

The following model implements CTT as an absolute group means model **??**, with the only difference that the person factor is a random effect, i.e. it assumes a Gaussian distribution of person scores.

CTT assumes that all items function precisely the same way and can therefore be ignored, which is a bold claim. For a set of experimental trials it may (or may not) be true that they are all equally hard. In most other situations item equality is highly questionnable. Two items from the same scale can differ in several aspects, one of which is how hard (or strong) an item is. Consider the following two items from a fictional user experience scale; most likely, the second item would get lower ratings on average, because it is stronger:

1. The interface is nice.
2. The interface is really cool.

One problem with CTT is that by averaging scores, the CTT swallows any information on item functioning. In contrast, the families of *Item response models (IRM)*, as well as *factor analysis models (FAM)*, do not take for granted that all items act the same. As diverse and elaborated these models are today, they all have in common, that items are modelled explicitly and get their own estimates. Discussing these models in more depth would require a separate book. Still, a simple item response model is nothing but an encounter of persons and test items. With only two populations involved, the standard psychometric model simply is a crossover of person and item random effects.

Some years ago, I proposed a novel personality construct *geekism*, which states that users differ in how enthusiastic they are about tinkering with technology. The hope was that we could explain differences in user behaviour, such as how they react when having to learn a new software application. A qualitative study with self-proclaimed geeks and several psychometric studies resulted in rating scale with 32 items. The Hugme case is one of the quantitative follow-up studies, where the Geekism scale was used together with the Need for Cognition scale (NCS), which assesses the tendency to enjoy intellectual puzzles in general. We were interested in (1) how the items function, (2) how reliable the scale is and (3) how Geekism correlates with Need-for-cognition.

```
attach(Hugme)

D_quest <- D_quest %>% mutate(Session = as.factor(session))
```

One important thing to note at this point is that psychometricians like to put things in matrices. An item response matrix is squared, whereas we need the long format for the regression engine. As is shown below, the long form can be transformed into a matrix and vice versa.

```
D_long <- expand_grid(Part = 1:8,
                      Item = 1:5) %>%
  mutate(rating = rnorm(40)) %>%
  mascutils::as_tbl_obs()
D_long

D_long %>%
  select(Part, Item, rating) %>%
  spread(key = Item, value = rating)
```

Part	1	2	3	4	5
1	-0.601	-0.434	0.728	-0.248	1.090
2	1.175	1.171	0.640	-1.627	-0.201
3	-0.267	0.539	1.252	-0.637	-1.098
4	0.083	0.007	-1.969	-1.391	-0.869
5	-0.702	-2.037	0.003	1.770	-0.127
6	-2.171	0.240	1.766	0.496	0.851
7	-0.241	0.263	-0.972	-0.382	-0.015
8	-1.602	0.675	1.352	0.383	-1.063

Psychometric programs often require matrix data, but for a multi-level models we need the long format. IRM models regard items as populations, too, and the basic IRT model is a cross-classified intercept-only model 5.5.

```
D_psymx_1 <-
  D_quest %>%
  filter(Scale == "Geek", Session == 1)


M_psymx_1 <-
  D_psymx_1 %>%
  brm(rating ~ 1 + (1|Part) + (1|Item), data = .)
```

With such an IRT model, we can extract the person scores, if we want to compare persons. Psychometric evaluation of a rating scale also draws upon items scores. In the following I will demonstrate two psychometric evaluations, using multi-level models:

1. *Test coverage* of a scale can be assessed by comparing the distribution of item scores with the distribution of person scores
2. *Test reliability* can be estimated by comparing scores across two sessions of testing.
3. *Test validity* can be estimated as person score correlations between scales.

5.8.1 Coverage

Geekism was assumed to vary widely in the population of users and we wanted to be able to cover the whole range with good precision. In IRT psychometrics, items and persons are actually scored on the same scale. The person-level coefficients represent the persons' level of geekism. The item-level effects can best be called item sensitivity. A rule in IRT psychometrics is that for accuracy in a certain range, this range has to be covered by items with a matching sensitivity. An item with consistently high ratings gets a high score, and is able to distinguish low levels of geekism. But, that makes it barely useful for

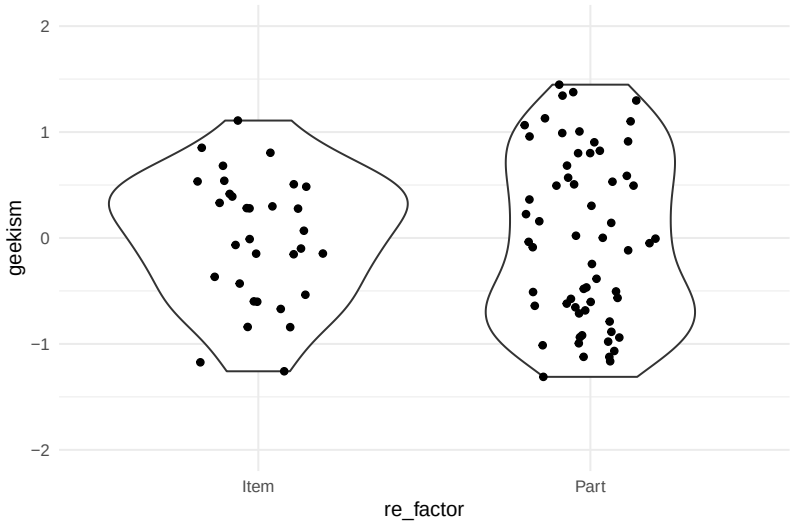
discriminating between geeks on high levels. Just think of how poorly a very simple arithmetic question, like "Which of the following numbers is divisible by 3? [2, 3, 5, 7, 9]" would be able to diagnose the math skills of you, the readers of this book. The inverse is also true: an item with a very strong proposition, like

> I always build my own computers

may be great to distinguish between amateur and pro level geekism, but the majority of persons will just say No. So, item sensitivity and a person tendency are inverse. In order to derive a picture on test coverage, it is useful to take the inverse effects and call this *strength*.

We have a linear model, where the rating is weighted sums of person tendency and item sensitivity. A high rating can mean two things (or both): coming from a very geek person, indeed, or it was a very sensitive item. For a good test coverage we need sensitive items for levels of low geekism and strong, i.e. *less* sensitive, items for the pros. Because random effects are centered at zero, we can simply reverse the scale with *item strength* being the negative sensitivity. Now we can compare the distributions of person and item scores side-by-side and check how the person tendencies are covered by item strength. *Note* that for obtaining the absolute scores, we can use the Bayr function `re_scores`, but for psychometric analysis, the deviation from the population average is sufficient, hence `ranef`.

```
P_psymx_1 <- posterior(M_psymx_1)

T_ranef <-
  ranef(P_psymx_1) %>%
  rename(geekism = center) %>%
  mutate(geekism = if_else(re_factor == "Item", -geekism, geekism)) # reversing


T_ranef %>%
  ggplot(aes(x = re_factor,
             y = geekism,
             label = re_entity)) +
  geom_violin() +
  geom_jitter(width = .2) +
  ylim(-2, 2)
```



It turns out that the 32 items of the test cover the range of very low to moderately high geekism quite well. The upper 20 percent are not represented so well, as it seems. If we were to use the scale to discriminate between geeks and totally geeks, more strong item had to be added.

5.8.2 Reliability

Next, we examine the reliability of the Geekism scale. Reliability is originally a CTT concept and means that the measurement error is small. For example, a reliable personality scale produces almost exactly the same score when applied to a person on different occasions. Is the Geekism score reliable? In our study we asked participants to fill out the questionnaire twice, with an experimental session in-between. If reliability of Geekism is good, the correlation of scores between sessions should be very strong.

In order to obtain the scores per session, we add an effect to the model. For reliability we are interested in correlation between person scores, so it suffices to add the Session random effect to the participant level, only. However, the same model can be used to do assess stability of item scores, too. This is rarely practiced, but as we will see, there is an interesting pattern.

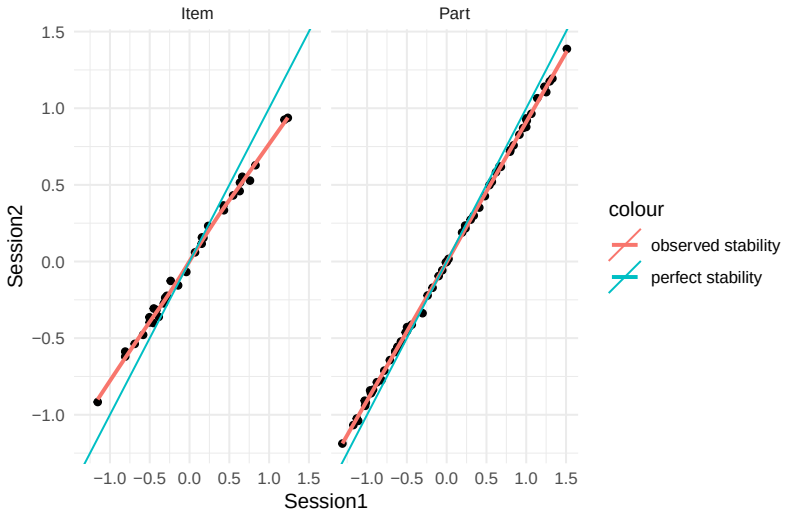
```
D_psymx_2 <-
  D_quest %>% filter(Scale == "Geek")

M_psymx_2 <-
  D_psymx_2 %>%
  brm(rating ~ 0 + Session + (0 + Session|Part) + (0 + Session|Item),
      data = .)
```

We extract the random effects and plot test-retest scores for participants and items. The red line in the plots indicates perfect stability for comparison.

```
T_ranef <-
  ranef(M_psymx_2) %>%
  select(re_factor, re_entity, Session = fixef, score = center) %>%
  spread(key = Session, value = score)

sample_n(T_ranef, 5)
```

re_factor	re_entity	Session1	Session2
Part	30	-0.497	-0.453
Part	7	-0.614	-0.559
Part	19	0.236	0.216
Part	68	-0.617	-0.556
Item	Geek23	-0.583	-0.478

```
plot_stability <-
  function(T_ranef) T_ranef %>%
                    ggplot(aes(x = Session1, y = Session2)) +
                    facet_grid(.~re_factor) +
                    geom_point() +
                    geom_smooth(aes(color = "observed stability"), se = F) +
                    geom_abline(aes(intercept = 0, slope = 1,
                                    color = "perfect stability"))

T_ranef %>% plot_stability()
```



The participant scores are highly reliable. If you measure the score of a person, you almost precisely know the result of another measure a few hours later. At least in short terms, the Geekism construct - whatever it may truly be - can be measured with almost no error. Only ever so slightly is there a trend that lower scores get higher the second time and higher get lower, which could be called a trend towards the average. Something during the experiment has led participants to report a more mediocre image of themselves.

The psychometric model further contains intercept and slope random effects for items, and we can examine test-retest patterns in the same way. We see the same trend towards the average, but much stronger. In psychometric analysis it is common to assess participant-level test-retest reliability, but rarely is that done on items. In the present case, we see that this can be a mistake. Here it seems that the trend towards mediocracy does not produce a bias on the population mean, because it is bi-directional and the item and participant scores are nicely symmetric around the center of the scale. Not every test may have these properties and any asymmetric wear-off effect of items would produce more serious biases.

Another situation where item stability matters is when a person doing the test is actually learning from it. Usually, it is not desired that a test can be learned, because that means people can train for it. would be unfortunateis unlikely to occur in a regular math or intelligence test, but when the items are real-world tasks, like programming a medical infusion pump or driving a car, the participant get a lot of feedback and will learn.

The example of test-retest stability shows one more time, how useful plots are for discovering patterns in data. More formally, test-retest stability is reported as a correlation. We can produce a correlation estimate by using the standard `cor` command on the participant-level random effects:

```
T_ranef %>%
  group_by(re_factor) %>%
  summarize(cor = cor(Session1, Session2))
```

re_factor	cor
Item	0.998
Part	1.000

Unfortunately, this lacks information about the degree of certainty. The better way is to let the regression engine estimate all correlations between random factors that are on the same level (Part, Item). The rgression engine `brm` fromn package Brms package does that by default, and this is why it has been used, here. The following code extracts the posterior distributions of all correlations in the model, creates an estimates table and a plot of 95% certainties.

```
clu_cor <-
  function(model){
    model %>%
    posterior() %>%
    filter(type == "cor") %>%
    mutate(parameter = str_remove_all(parameter, "cor_")) %>%
    group_by(parameter) %>%
    summarize(center = median(value),
              lower = quantile(value, .025),
              upper = quantile(value, .975)) %>%
    separate(parameter, into = c("re_factor", "between", "and"), sep = "__")
}


M_psymx_2 %>%
  clu_cor()
```

re_factor	between	and	center	lower	upper
Item	Session1	Session2	0.984	0.915	0.999
Part	Session1	Session2	0.995	0.968	1.000

With random effects correlations assessing test-retest-stability is straightforward. If test and retest ramdom effects correlate strongly, we can be sure that the error of measurement is low and we can call it a reliable scale. Good reliability is necessary, but not sufficient to also call a scale valid.

5.8.3 Validity

Reliability doesn't say anything about what the scale actually measures. In psychometric studies, *validity* of a scale is routinely evaluated by comparing the scores to external measures. In a perfect world, it would be assessed how scores are related to relevant real-world behaviour, such as:

1. Are high-Geek persons more enthusiastic to learn a programming language?
2. Do high-Geek persons perform better in computer jobs?
3. Are high-Geek persons more likely to buy robot toys for their offsprings?

In the real world, researchers in the field of personality are usually content with relating their scales to other, validated personality scales. In the Hugme study, participants were also asked to rate themselves on the Need-for-Cognition scale (NCS). In very brief NCS measures how much a person enjoys intellectual puzzles. Since computers are intellectual puzzles, sometimes in a good way, often not, we thought that high-Geek persons must also score high on NCS.

At the same time, a very strong correlation between Geek and NCS would indicate that the two scales render the same property, which would make one of them redundant, probably the newcomer. The following model estimates the person scores per scale and we can extract the correlation.

```
M_psymx_3 <-
  D_psymx_3 %>%
  brm(rating ~ 0 + Scale + (0 + Scale|Part),  data = .)
```

```
M_psymx_3 %>% clu_cor()
```

re_factor	between	and	center	lower	upper
Part	Intercept	ScaleNCS	-0.524	-0.698	-0.3

We observe a weakly positive association between Geek and NCS, just as was hoped for.

```
detach(Hugme)
```

5.8.4 Design-o-metrix

Psychometrics, as it was introduced above, deals with comparing human individuals. In Design Research, this may be of interest, but the real stake is to *compare designs*. As we will see in this section, psychometric concepts can well be extended to *designometric problems*. However, there is one twist, which has up til now been overlooked in most of Design Research: in designometric studies the target population is designs, not humans. In a typical psychometric study, measurements are an encounter of humans with items, with the ultimate goal of measuring humans. A designometric measurement is the encounter of three popluations, humans, items and, ultimately, designs. Classic psychometric tools use a 2-dimensional matrix as input and cannot deal with a third dimension. Multi-level models have no such limits. All we have to do, is crossing in another non-human population **??**non-human-populations).

We revisit the Uncanny Valley data set 4.6. The experiment used eight items from the Eeriness scale [%MacDorman] to ask the judgment of participants on 82 stimuli showing robot faces. In one of our experiments (RK_1), participants simply rated all robots face in three separate session. Here are a few example observations:

```
attach(Uncanny)
```

```
RK_1 %>%
  select(Part, Item, Session, response) %>%
  sample_n(8)
```

Part	Item	Session	response
p2_04	nE7	2	-0.083
p2_09	nE6	3	-0.182
p2_10	nE6	2	-0.667
p2_07	nE7	3	-0.390
p2_09	nE3	3	-0.660
p2_05	nE2	1	-0.293
p1_10	nE1	2	-0.757
p2_12	nE4	2	-0.503

With this data we seem to be standing on familiar psychometric grounds: Items are used on persons and we have three measures over time. We can calculate test-retest stability of items and persons using a multi-level model. Voila! Here are your correlations, person and item stability - with credibility limits. Wait a second! What is being measured here? Persons? No, robot faces. The original question was, how human-likeness of robot faces is related to perceived eeriness of robot faces and the Eeriness scale intended purpose is the comparison of designs, not persons. For example, it could be used by robot designers to check that a design does not trigger undesireable emotional responses. Without knowing the humen-likeness scores, robot faces become just a naked *sample of designs* **??**:

```
UV_dsgmx <-
  RK_1 %>%
  rename(Design = Stimulus) %>%
  select(Part, Item, Design, Session, response) %>%
  as_tbl_obs()

UV_dsgmx
```

Measures in the Uncanny experiment are an encounter of three samples: Part, Item and Design, and Design is the one of interest. That means we need a model that produces Design-level scores. For the user of multi-level models that just means to add a Design random effect to the psychometric model (Part, Item). Models, where a design random factor sits on top of a psychometric model, I call from here on a *designometric models*. The most basic designometric model is a three-way cross-classified intercept-only model, from which design scores can be extracted. By extending the test-retest psychometric model `M_psymx_2`, we can estimate test-retest stability.

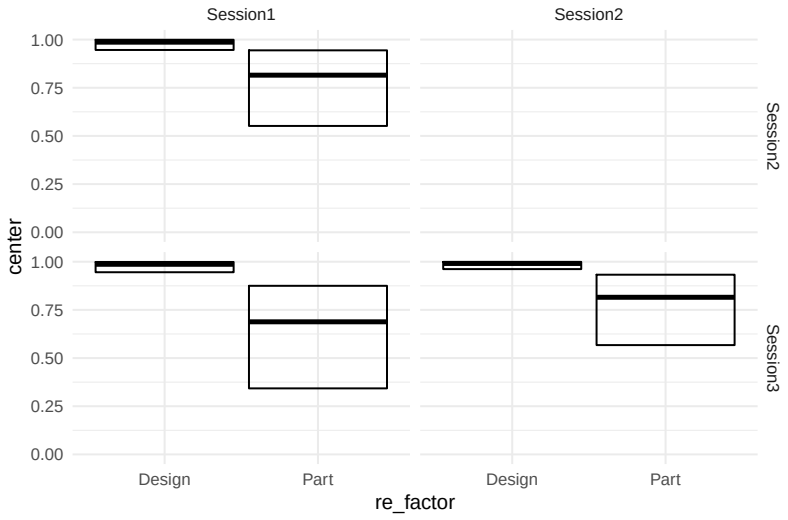
```
M_dsgmx_1 <-
  UV_dsgmx %>%
  brm(response ~ 0 + Session +
        (0 + Session|Design) +
```

```
        (1 + Item) +
        (0 + Session|Part), data = .)
```

Like in the psychometric situation, we extract the correlations. Since we have three sessions, we get two stability scores per level.

```
M_dsgmx_1 %>%
  posterior() %>%
  clu_cor() %>%
  print() %>%
  ggplot(aes(x = re_factor,
             y = center,
             ymin = lower,
             ymax = upper)) +
  facet_grid(and  ~ between) +
  geom_crossbar() +
  ylim(0,1)
```

```
## # A tibble: 6 x 6
##   re_factor between  and      center lower upper
##   <chr>     <chr>    <chr>     <dbl> <dbl> <dbl>
## 1 Design    Session1 Session2  0.987 0.946 0.999
## 2 Design    Session1 Session3  0.987 0.945 0.999
## 3 Design    Session2 Session3  0.991 0.962 0.999
## 4 Part      Session1 Session2  0.815 0.552 0.944
## 5 Part      Session1 Session3  0.688 0.343 0.875
## 6 Part      Session2 Session3  0.815 0.567 0.933
```



```
detach(Uncanny)
```

The test-retest stability for designs is very reassuring. Ratings on the Eeriness scale are highly reproducable and the error will be very small. To a lesser, but still sufficient degree are person scores stable.

But, what does the person score (and its stability) actually mean? It describes the tendency of a person to give high ratings on Eeriness. Should a researcher want to assess how vulnerable a person is to the Uncanny Valley effect, the Eeriness scale is also reliable for measuring persons. Many scales in design research lend themselves to be looked at from a designometric and psychometric perspective. For example, a hypothetical scale to measure comfort of sitting can be used to evaluate seats, but can also be used to measure how comfortable a person is with sitting.

No seat fits every person, or put differently: the comfort of a seat depends on the person sitting in it. This points us at one of many possible extensions to carry out deeper designometric analysis. If the difficulty of an item in a psychometric test depends on who is being tested, this is called *differential item functioning*. For example, the large international student evaluations PISA and TIMMS routinely check their test items for cultural differences. The aim is to formulate test questions in such a way that they are equally comprehensible with all cultural backgrounds. Most likely, this is a desireable property for designometric scales, too. In a multi-level designometric model, this could be incorporated as an interaction effect between cultural background and item-level coefficients.

That all being said about designometric models, my observation is that practically all published rating scales in design research have been validated under a psychometric perspective, rather than a designometric. This is a mistake! If the purpose of the scale is to compare designs, the scale validation must be carried out on the design scores. In the worst case, a designometric scale is evaluated by a study, where a sample of participants and a sample of items do only encounter a single design, rather than a sample. It worries me that practically all purportedly designometric scales out there have been validated under the wrong perspective, and I call this the *psychometric fallacy* in design research.

6

Generalized Linear Models

In the preceding chapters we got acquainted with the linear model as an extremely flexible tool to represent dependencies between predictors and outcome variables. We saw how factors and covariates gracefully work together and how complex research designs can be captured by multiple random effects. It was all about specifying an appropriate (and often sophisticated) right-hand side of the regression formula, the predictor term. Little space has been dedicated to the outcome variables, except that sometimes we used log-transformed outcomes to accomodate the Gaussian error term. That is now going to change, and we will start by examining the assumptions that are associated with the outcome variable.

Have you wondered about why I have been using so many simulated data sets up to this point? The reason for using simulated data is: the linear model, as introduced so far, makes assumptions that are *never* truly met by real data. The simulated data sets so far were meant to demonstrate some features found in real data sets, but generously wiped over some other frequent peculiarities.

Another question is probably lurking in the minds of readers with some classic statistics training: what has happened to the assumptions of ANOVA and the like, and where are all the neat tests that check for Normality, constant variance and such? The Gaussian linear model, which we used throughout **??** and **??**, makes many assumptions, but in my view, the three crucial assumptions are:

1. *Linearity* of the association between predictors and outcome variable.
2. *Gaussian distribution* of responses
3. *constant variance* of response distribution

In the next section we will review these assumptions and lead them ad absurdum. Simply put, in the real world there is no such thing as Gaussian distribution and true linearity. Checking assumptions on a model that you

know is inappropriate, seems a futile exercise, unless better alternatives are available, and that is the case: with *Generalized Linear Models* (GLMs) we extend our regression modelling framework once again, this time focussing on the responses and their shape of randomness.

As we will see, GLMs solves some common problems with linearity and gives us more choices on the shape of randomness. To say that once and for all: What GLMs do not do is relax the assumptions of linear models. And because I have met at least one seasoned researcher who divided the world of data into two categories, "parametric data", that meets ANOVA assumptions, and "non-parametric data" that does not, let me get this perfectly straight: *data is neither parametric nor non-parametric*. Instead, data is the result of a process that distributes measures in some form and a good model aligns to this form. Second, a *model is parametric*, when the statistics it produces have a useful interpretations, like the intercept is the group mean of the reference group and the intercept random effect represents the variation between individuals. All models presented in this chapter (and this book) fulfill this requirement and *all are parametric*. There may be just one counter-example, which is polynomial regression **??**, which we used for its ability to render non-monotonic curves. The polynomial coefficients have no interpretation in terms of the cognitive processes leading to the Uncanny Valley. However, as we have seen in **??**, they can easily be used to derive meaningful parameters, such as teh positions of shoulder and trough. A clear example of a non-parametric method is teh Mann-Withney U-test, which operates on sums of ranks, which typically has no useful interpretation.

The GLM framework rests on two extensions that bring us a huge step closer to our valuable data. The first one is a minor mathematical trick to establish linearity, the *link function*. The second is the expected *shape of randomness*. After we established the GLM framework 6.1, I will introduce a good dozen of model classes, that leaves little reason to crudely approximate with Gaussian distributions, data transformations, or unintelligible non-parametric procedures. There almost always is a clear choice right at the beginning that largely depends on the properties of the response variable, for example:

- *Poisson LM* is the first choice for outcome variables that are counted (with no upper limit), like number of errors.

- *Binomial (aka logistic) LM* covers the case of successful task completion, where counts have an upper boundary.

These two GLM families have been around for more than half a century. The quest for a good model for reaction time and time-on-task was more difficult, as there does not seem to be a generally accepted default. Luckily, with recent developments in Bayesian regression engines the choice of random distributions has become much broader. For RT and ToT, I will suggest exponentially-

modified Gaussian *(ExGauss)* models or, to some extent, *Gamma* models. For rating scales, where responses fall into a few ordered categories, *ordinal logistic regression* is a generally accepted approach. For (quasi)continuous rating scales will I make a novel suggestion, the *Beta LM*.

Too many choices can be a burden, but as we will see, most of the time the appropriate model family is obvious. For the impatient readers, here is the recipe: Answer the following three questions about the outcome variable. And, to make it even easier, it is always safe to answer Yes to the third question.

1. Is the outcome variable discrete or continuous?
2. What are the lower and upper boundaries of outcome measures?
3. Can we expect over-dispersion?

Then use the graph below to identify the correct distribution family and jump to the respective section.

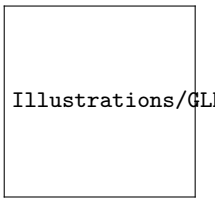


Fig. 6.1. GLM decision chart

6.1 Elements of Generalized Linear Models

GLM is a *framework for modelling* that produces a *family of models*. Every member of this family uses specific *link functions* to establish linearity and chooses a particular *random distribution*, that has an adequate shape and mean-variance relationship.

Sometimes GLMs are mistaken as a way to relax assumptions of linear models, (or even called non-parametric). They are definitely not! Every member makes precise assumptions on the level of measurement and the shape of randomness (see Table A). One can even argue that Poisson, Binomial and exponential regression are stricter than Gaussian, as they use only one parameter, with the consequence of a tight association between variance and mean. A few members of GLM are classic: Poisson, Binomial (aka logistic) and exponential regression have routinely been used before they were united under the hood of GLM. These and a few others are called *canonical* GLMs, as they possess some

convenient mathematical properties, that made efficient estimation possible, back in the days of expensive computer time.

For a first understanding of Generalized Linear Models, you should know that linear models are one family of Generalized Linear Models, which we call a Gaussian linear model. The three crucial assumptions of Gaussian linear models are encoded in the model formula:

$$\mu_i = \beta_0 + \beta_1 x_{1i} + \cdots + \beta_k x_{ki}$$
$$y_i \sim \text{Gaus}(\mu_i, \sigma)$$

The first term, we call the likelihood and it represents the systematic quantitative relations we expect to find in the data. When it is a sum of products, like above, we call it linear. *Linearity* is a frequently under-regarded assumption of linear models and it is doomed to fail in 6.1.1. The second term defines the pattern of randomness and it hosts two further assumptions: *Gaussian distribution* and *constant error variance* of the random component.

In classic statistics education, the tendency is still to present these assumptions as preconditions for a successful ANOVA or linear regression. The very term *pre*condition suggest, that they need to be checked upfront and the classic statisticians are used to deploy a zoo of null hypothesis tests for this purpose, although it is widely held among statisticians that this practice is illogical. If an assumptions seems to be violated, let's say Normality, researchers then often turn to non-parametric tests. Many also just continue with ANOVA and add some shameful statements to the discussion of results or humbly cite one research paper that claims ANOVAs robustness to violations.

The parameters of a polynomial model usually don't have a direct interpretation. However, we saw that useful parameters, such as the minimum of the curve, can be derived. Therefore, polynomial models can be called *semiparametric*, at least. As an example for a *non-parametric* test, the Mann-Whitney U statistic is composed of the number of times observations in group A are larger than in group B. The resulting sum U usually bears little relation to any real world process or question. Strictly speaking, the label non-parametric has nothing to do with ANOVA assumptions. It refers to the usefulness of parameters. A research problem, where U as the sum of wins has a useful interpretation could be that in some dueling disciplines, such as Fencing, team competitions are constructed by letting every athlete from a team duel every member of the opponent team. We could call the U-test parametric, and perhaps, the group means turn out to be meaningless.

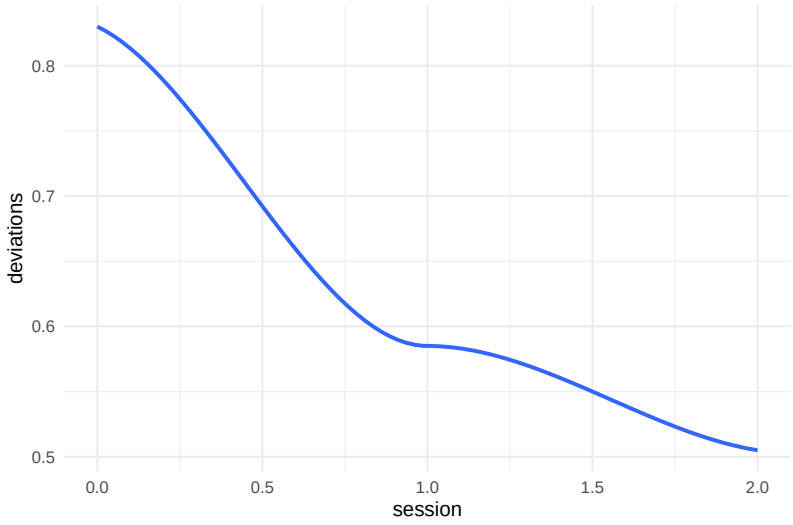
6.1.1 Re-linking linearity

In the chapter on Linear Models, we encountered several situations where linearity itself was violated:

- in 4.3.5 we used ordered factors to estimate a learning curve
- in 4.5.3, we used conditional effects when two or more interventions improve the same process, e.g. visual recognition of letters
- and in **??** we used polynomials to estimate wildly curved relationships

The case of Polynomial regression is special in two ways: first, the curvature itself is of theoretical interest (e.g. finding the "trough" of the Uncanny Valley efect). Second, a polynomial curve (of second degree or more) is no longer monotonously increasing (or decreasing). In contrast, learning curves and saturation effects have in common that in both situations performance steadily increases when we add more to the predictor side. There is a limit to performance, which is reached asymptotically, but there always is a monotonous trend.

```
attach(IPump)

D_Novel %>%
  ggplot(aes(x= session, y = deviations)) +
#  geom_jitter() +
  geom_smooth(se = F)
```



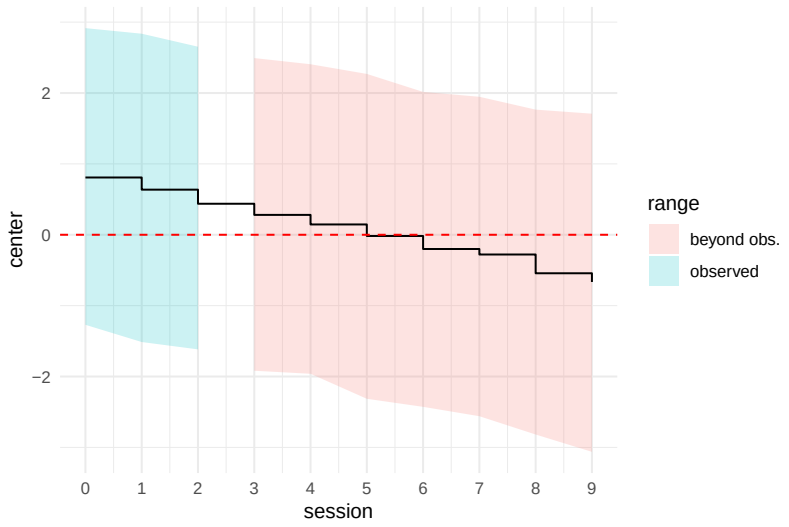
We used an OFM with stairways coding to account for this non-linearity, but that has one disadvantage. From a practical perspective it would interesting to know, how performance improves when practice continues. What would be performance in (hypothetical) sessions 4, 5 and 10. Because the OFM just makes up one estimate for every level, there is no way to get predictions beyond the observed range.

With an LRM, the slope parameter applies to all steps, which gives us the possibility of deriving predictions beyond the observed range. To demonstrate this on the deviations from optimal path, the following code estimates a plain LRM and then injects some new (virtual) data to get predictions beyond the observed range:

```
M_LRM_1 <- stan_glm(deviations ~ 1 + session, data = D_Novel)
```

```
D_new <-
  tibble(session = as.integer(c(0:9)),
         range = if_else(session < 3, "observed", "beyond obs.")) %>%
  as_tbl_obs()

predict(M_LRM_1, newdata = D_new) %>%
  print() %>%
  left_join(D_new) %>%
  ggplot(aes(x = session, y = center, ymin = lower, ymax = upper)) +
  geom_ribbon(aes(fill = range), alpha = .2) +
  geom_step() +
  geom_hline(aes(yintercept = 0), color = "red", linetype = 2) +
  scale_x_continuous(breaks = 0:10)
```

```
## **  10 predictions (scale: resp) with 95% credibility limits (five shown below)
##   Obs center lower upper
## 1   1  0.780 -1.36  2.99
## 2   3  0.507 -1.56  2.63
## 3   6 -0.050 -2.26  2.13
## 4   9 -0.477 -2.73  1.83
## 5  10 -0.701 -2.97  1.66
```




```
detach(IPump)
```

When we use a linear model when there truly is an asymptotic curve, negative values are produced, which are impossible. As the graph shows, this is most pronounced the further we move away from the observed range. But, it also effects the observed levels, as the lower credibility limits are negative, too.

Such a non-linearity happens to all outcome variables that have natural lower or upper boundaries, and that includes all outcome variables in the universe, except its very own spatial extension, perhaps. Speed of light can only be approached asymptotically (if you have some mass to carry around) and even the darkest intergalactic spaces are lit by cosmic background radiation and therefore have a temperature just slightly above absolute zero. It is an unescapable truth, that all our ephemeral measures in design research have boundaries and therefore suffer from wrong predictions:

- Errors and other countable incidences are bounded at zero
- Rating scales are bounded at the lower and upper extreme item
- Task completion has a lower bound of zero and upper bound is the number of tasks.
- Temporal measures formally have lower bound of zero, but psychologically, the lower bound always is a positive number.

The bare linear term is always inadequate, which is unfortunate, because its strength is the endless versatility in specifying relations between predictor variables and outcome. How can we escape the problem of boundaries, without sacrificing such a useful tool? Generalized linear models use a simple

mathematical trick that keeps linear terms, but confines the fitted responses to the natural boundaries of the measures. In linear models, the linear term μ is mapped directly to fitted responses: $\mu_i = \beta_0 + x_1\beta_1$. In GLMs, a layer is drawn between the fitted response μ and the linear term, *linear predictor* θ. The *link function* ϕ transforms between μ and θ. In order to transform back to the scale of measurement, the inverse of the link function, the *mean function* is used.

$$\theta_i = \beta_0 + x_1\beta_1$$
$$\mu_i = \phi(\theta_i)$$

In arithmetics an abundance of functions exists for every possible purpose. However, a link function ϕ must fulfill two criteria:

1. mapping from the (linear) range $[-\infty; \infty]$ to the range of the response, e.g. $[0; \infty]$.
2. be monotonically increasing

Intuitively speaking, a monotonically increasing function always preserves the order, such that the following holds for a link function.

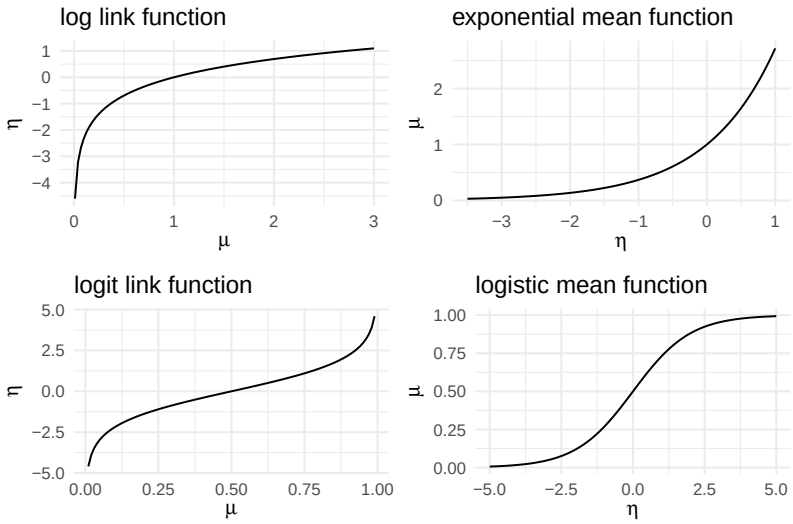
$$\theta_i > \theta_j \rightarrow \phi(\theta_i) > \phi(\theta_j) \rightarrow \mu_i > \mu_j$$

One reason for monotonicity is that for a link function ϕ there must exist the inverse, which is called the mean function (ϕ^{-1}). For example, x^2 is not a proper link function, because its inverse, $\sqrt{x}$ can take *two* values (e.g. $\sqrt{x} = [2, -2]$) and therefore is not a function on the desired range, strictly speaking.

The most typical cases are that there is a lower boundary of zero, or there are two boundaries. An adequate link function for count variables would map the range of natural numbers (only lower bound) to the *linear range* of η that is $[-\infty; \infty]$. The *logarithm* is such a function and its inverse is the *exponential* function, which bends the linear range back into the boundary. Other variables, like success rates or rating scales, have lower and upper boundaries. A suitable pair of functions is the *logit* link function and the *logistic* mean function.

```
plot_glmfun <- function(f = log,
                        title = "log link function",
                        lower = .01, upper = 3,
                        dir = "link"){
  out <-
    tibble(x = seq(lower, upper, (upper - lower)/100)) %>%
    ggplot(aes(x)) +
```

```
    stat_function(fun = f) +
    labs(title = title) +
    labs(x = expression(mu), y = expression(eta))
  if(dir == "mean") out <- out + labs(x = expression(eta),
                                      y = expression(mu))
  out
}

gridExtra::grid.arrange(
  plot_glmfun(),
  plot_glmfun(f = exp, "exponential mean function",  -3.5, 1, dir = "mean"),
  plot_glmfun(f = logit, "logit link function", 0.01, .99),
  plot_glmfun(f = inv_logit, "logistic mean function",  -5, 5, dir = "mean"))
```



Using the link function comes at a cost: the linear coefficients β_i is losing its interpretation as increment-per-unit and no longer has a natural interpretation. Later we will see that logarithmic and logit scales gain an intuitive interpretation when parameters are exponentiated, $\exp(\beta_i)$ (**??** and **??**

6.1.2 Choosing patterns of randomness

The second term of a linear model, $y_i \sim Norm(\mu_i, \sigma)$ states that the observed values are drawn from Gaussian distributions(see @ref(resid_normality)). But Gaussian distributions have the same problem as the linearity assumption: the range is $[-\infty; \infty]$. in fact, a Gaussian distribution can only be a reasonable ap-

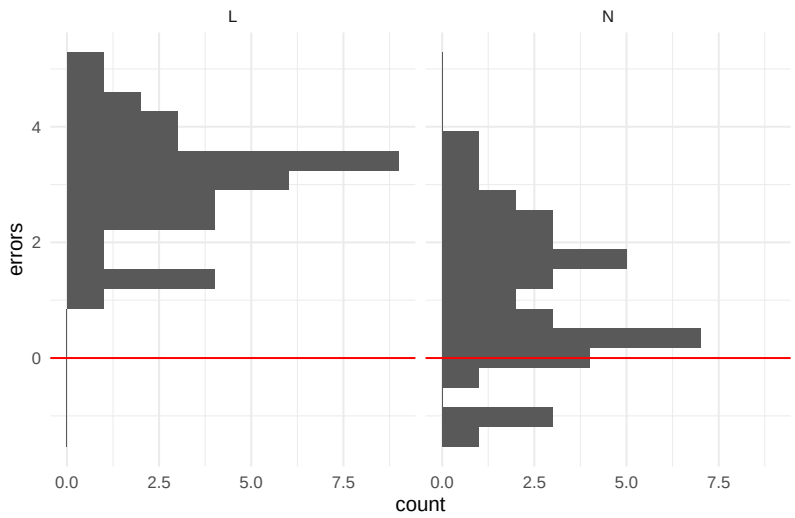
proximation when the measures are far off the boundaries of measures and the error is much smaller than the predicted values (@ref(normal_distributions)).

The problem can be demonstrated by simulating observations, using a Gaussian pattern of randomness, and see how this fails to produce realistic data. Imagine a study comparing a novel and a legacy interface design for medical infusion pumps. The researchers let trained nurses perform a single task on both devices and count the errors. Assuming, the average number of errors per tasks is $\mu_L = 3$ for the legacy device and $\mu_N = 1.2$ for the novel device, with standard deviation of $\sigma = .8$. We can simulate a basic data set as:

```
N = 80
D_pumps_sim <-
  tibble(Design = rep(c("L", "N"), N/2),
              mu = if_else(Design == "L", 3, 1.2),
              errors = rnorm(N, mu, sd = 1))
```

We illustrate the data set using histograms:

```
D_pumps_sim %>%
  ggplot(aes(x = errors)) +
  facet_grid(~Design) +
  geom_histogram(bins = 20) +
  geom_vline(col = "red", xintercept = 0) +
  coord_flip()
```



We immediatly see, that simulation with Gaussian distributions is rather inappropriate: a substantial number of simulated observations is *negative*, which

strictly makes no sense for error counts. The pragmatic and impatient reader may suggest to adjust the standard deviation (or move the averages up) to make negative values less unlikely. That would be a poor solution as Gaussian distributions support the full range of real numbers, no matter how small the variance is (but not zero). There is always a chance of negative simulations, as tiny as it may be. Repeatedly running the simulation until `pumps` contains exclusively positive numbers (and zero), would compromise the idea of random numbers itself. The second reason is that the simulations very purpose was to express and explore expectations from the linear model (CG). We can simply conclude that any model that assumes normally distributed errors must be wrong when the outcome is bounded below or above, which means: always.

Recall how linearity is gradually bent when a magnitude approaches its natural limit. A similar effect occurs for distributions. Distributions that respect a lower or upper limit get squeezed like chewing gum into a corner when approaching the boundaries. Review the sections on Binomial 3.4.2.3 and Poisson distributions 3.4.2.4 for illustrations. As a matter of fact, a lot of real data in design research is skewed that way, making Gaussian distributions a poor fit.

A common misconception is that random distributions approach the Gaussian distribution with larger sample sizes. But, what really happens only, is that increasing the number of observations renders the true distribution more clearly.

In chapter 3.4.2 a number of random distributions were introduced, together with conditions of when they arise. The major criteria were related to properties of the outcome measure: how it is bounded and whether it is discrete (countable) or continuous. Generalized Linear Models give the researcher a larger choice for modelling the random component and the following table lists some common candidates.

boundaries	discrete	continuous
unbounded	NA	Normal
lower	Poisson	Exponential
lower and upper	Binomial	Beta

That is not to say that these five are the only possible choices. Many dozens of statistical distributions are known and these five are just making the least assumptions on the shape of randomness in their class (mathematicians call this *maximum entropy distributions*). In fact, we will soon discover that real data frequently violates principles of these distributions. For example, count measures in behavioural research typically show a variance that exceeds the mean, which speaks against the Poisson distributions. As we will see in 6.2.3 and **??**, Poisson distribution can still be used in such cases with some additional tweaks borrowed from multi-level modelling (observation-level random effects).

As we will see, response times in design research are particularly misbehaved, as they do not have their lower boundary at zero, but at the lowest human possible time to solve the task. In contrast, most continuous distributions assume that measures near zero are at least possible. In case of response times, we will take advantage of the fact, that modern Bayesian estimation engines support a larger range of distributions than ever seen before (i.e., in classic statistics). The `stan_glm` regression engine has been designed with downwards compatibility in mind, which is why it includes fewer distributions. Luckily, there is a sibling engine in the package `brms`, which is more progressive and gives many more choices, such as the Exponential-Gaussian distribution.

6.1.3 Mean-variance relationship

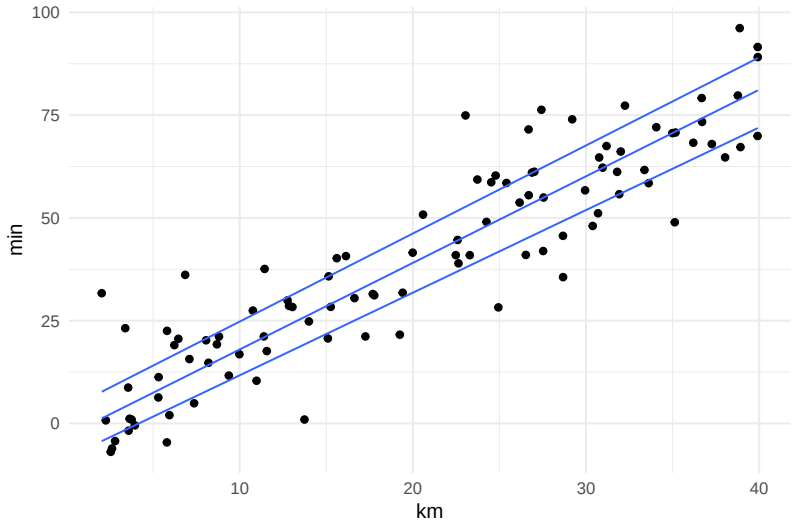
The third assumption of linear models is rooted in the random component term as well. Recall, that there is just one parameter σ for the dispersion of randomness and that any Gaussian distribution's dispersion is exclusively determined by σ. That is more harmful as it may sound. In most real data, the dispersion of randomness depends on the location, as can be illustrated by the following simulation.

Imagine a survey on commuter behaviour that asks the following questions:

1. How long is the route?
2. How long does it *typically* take?
3. What are the maximum and minimum travel times you remember?

If we simulate such data from a linear model, the relationship between length of route and travel time would look like a evenly wide band, which is due to the constant variance:

```
N = 100
tibble(Obs = as.factor(1:N),
           km  = runif(N, 2, 40),
           min = rnorm(N, km * 2, 10)) %>%
  ggplot(aes(x = km, y = min)) +
  geom_point() +
  geom_quantile(quantiles = c(.25, .5, .75))
```

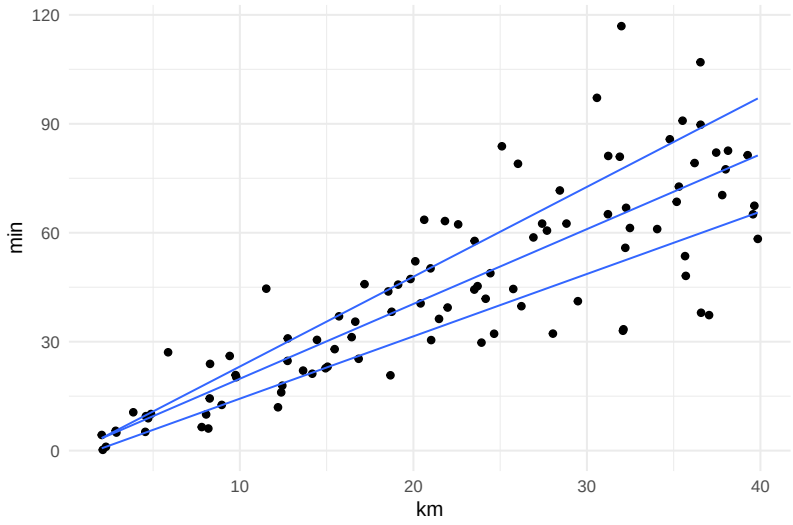


Again, we get some impossible negative data points, but what is also unrealistic is that persons who live right around the corner experience the same range of possible travel times than people who drive dozens of kilometers. Most of the time, we intuit the dispersion of randomness to increase with the magnitude.

Most other distributions do not have constant variance. For example, a Gamma distribution takes two parameters, shape α and scale τ and both of them influence mean and variance of the distribution, such that the variance increases by the mean by square:

$$\begin{aligned} Y &\sim \text{Gamma}(\alpha, \theta) \\ E(Y) &= \alpha\theta \\ \text{Var}(Y) &= \alpha\theta^2 \\ \text{Var}(Y) &= E(Y)\theta \end{aligned}$$

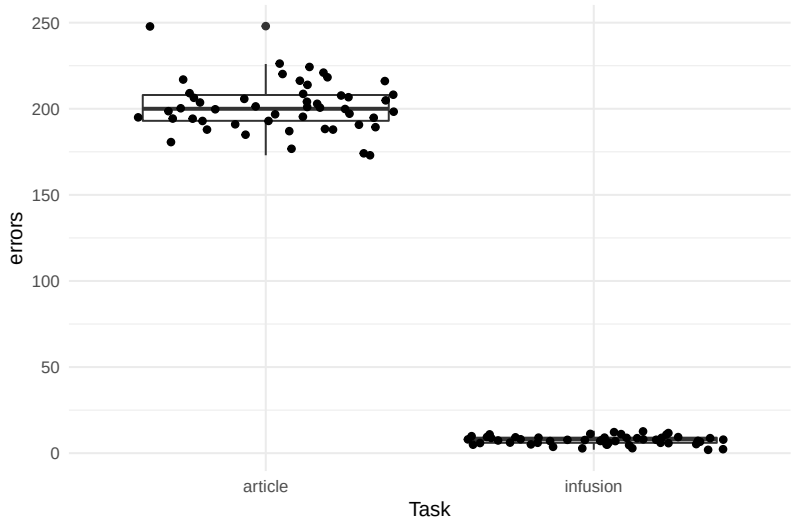
```
tibble(Obs = as.factor(1:100),
           km  = runif(100, 2, 40),
           min = rgamma(100, shape = km * .5, scale = 4)) %>%
  ggplot(aes(x = km, y = min)) +
  geom_point() +
  geom_quantile(quantiles = c(.25, .5, .75))
```



A similar situation arises for count data. When counting user errors, we would expect a larger variance for complex tasks and interfaces, e.g. writing an article in a word processor, as compared to the rather simple situation like operating a medical infusion pump. For count data, the Poisson distribution is often a starting point and for Poisson distributed variables, mean and variance are both exactly determined by the Poisson rate parameter λ, and therefore linearly connected.

$$Y \sim \text{Poisson}(\lambda)$$
$$\text{Var}(Y) = E(Y) = \lambda$$

```
tibble(Obs = as.factor(1:100),
           Task  = rep(c("article", "infusion"), 50),
           errors = rpois(100, lambda = if_else(Task == "article", 200, 8))) %>%
  ggplot(aes(x = Task, y = errors)) +
  geom_boxplot() +
  geom_jitter()
```

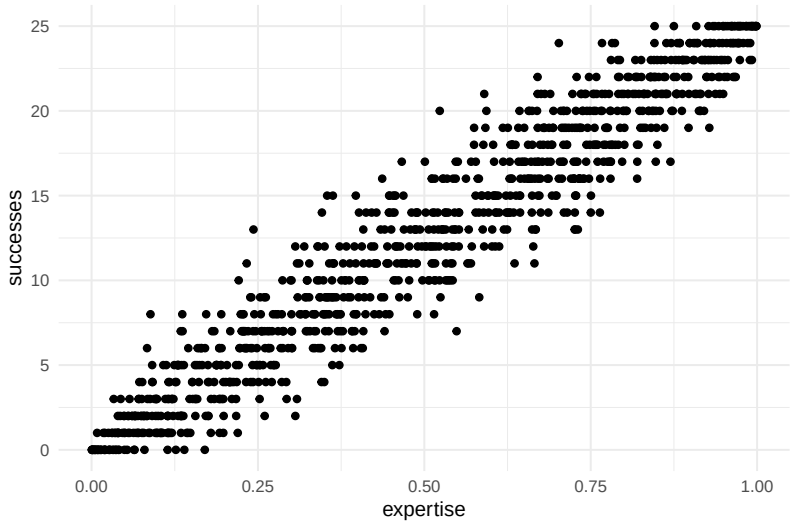


Not by coincidence, practically all distributions with a lower boundary have variance increase with the mean. Distributions that have two boundaries, like binomial or beta distributions also have a mean-variance relationship, but a different one. For binomially distributed variables, mean and variance are determined as follows:

$$\begin{aligned} Y &\sim \text{Binom}(p, k) \\ E(Y) &= pk \\ \text{Var}(Y) &= p(1-p)k \\ \text{Var}(Y) &= E(Y)(1-p) \end{aligned}$$

To see this, imagine a study that examines the relationship between user expertise (for the convenience on a scale of zero to one) and success rate on ten tasks. The result is a cigar-like shape. For binomial distributions, variance gets largest, when the chance of success is centered at $p = .5$. This is very similar for other distributions with two boundaries, such as beta and logit-Gaussian distributions.

```
tibble(expertise  = runif(1000, 0, 1),
           successes = rbinom(1000, 25, expertise)) %>%
  ggplot(aes(x = expertise, y = successes)) +
  geom_point()
```



In conclusion, the Gaussian distribution assumption of constant variance is flawed in two aspects: real distributions are typically asymmetric and have mean and variance linked. Both phenomena are tightly linked to the presence of boundaries. Broadly, the deviation from symmetry gets worse when observations are close to the boundaries (e.g. low error rates), whereas differences in variance is more pronounced when the means are far apart from each other.

Still, using distributions that are not Gaussian sometimes carries minor complications. Gaussian distributions have the convenient property that the amount of randomness is directly expressed as the parameter σ. That allowed us to compare the fit of two models A and B by comparing σ_A and σ_B. In random distributions with just one parameter, the variance of randomness is fixed by the location (e.g. Poisson λ or Binomial p). For distributions with more than one parameter, dispersion of randomness typically is a function of two or more parameters, as can be seen in teh formulas above. For example, Gamma distributions have two parameters, but these do not pull location and dispersion as neatly apart as Gaussian distributions do. Instead, mean and variance depend on both parameters for Gamma distributions.

Using distributions with entanglement of location and dispersion seems to be a step back, but frequently it is necessary to render a realistic association between location of fitted responses and amount of absolute randomness. Most distributions with a lower bound (e.g. Poisson, exponential and Gamma) increase variance with mean, whereas double bounded distributions (beta and binomial) typically have maximum variance when the distribution is centered and symmetric. For the researcher this all means that selecting a distribution class for a Generalized Linear Model, the choice determines the shape of randomness *and* the relation between location and variance.

The following sections are organized by type of typical outcome variable (counts, time intervals and rating scales). Each section (except rating scales) first introduces a one-parametric model (e.g. Poisson). A frequent problem with these models is that the location-variance relation is too strict. When errors are more widely dispersed than is allowed, this is called over-dispersion and one can either use a trick borrowed from multi-level models, observation-level random effects @(olre) or select a two-parametric distribution class (e.g., Negative-Binomial).

6.2 Count data

Gaussian distributions assume that the random variable under investigation is continuous. For measures, such as time, it is natural and it can be a reasonable approximation for measures with fine-grained steps, such as average scores of self-report scales with a large number of items. Other frequently used measures are clearly, i.e. naturally, discrete, in particular everything that is counted. Examples are: number of errors, number of succesfully completed tasks or the number of users. Naturally, count measures have a lower bound and frequently this is zero. A distinction has to be made, though, for the upper bound. In some cases, there is no well defined upper bound, or it is very large (e.g. number of users) and Poisson regression applies. In other cases, the upper bound is given by the research design, for example the number of tasks given to a user. When there is an upper bound, Binomial distributions apply, which is called logistic regression.

6.2.1 Poisson regression

If data can be considered successes in a fixed number of trials, logistic regression is the model type of choice. When the outcome variable is of type count, but there is no apparent upper limit, Poisson regression applies.

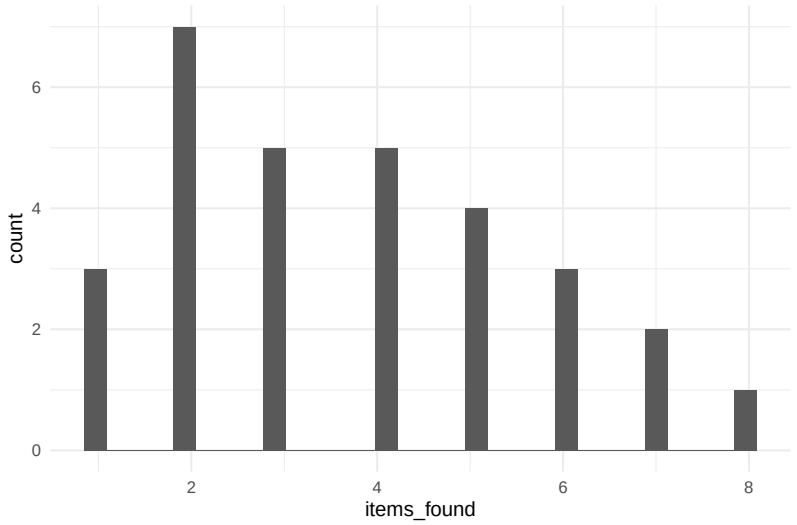
In brief, Poisson regression has the following attributes:

1. The outcome variable is bounded at zero (and that must be a possible outcome, indeed).
2. The linear predictor is on a logarithmic scale, with the exponential function being the inverse.
3. The random component follows a Poisson distribution.
4. Variance of randomness increases linearly with the mean.

The link function is the logarithm, as it transforms from the non-negative range of numbers to real numbers. For a start, we have a look at a Poisson

GMM. Recall the smart smurfer game from section 3.4.2.4. Imagine that in an advanced level of the game , items are well hidden from the player and therefore extremely difficult to catch. To compensate for the decreased visibility of items, the level carries an abundance them. In fact, the goal of the designers is that visibility and abundance are so carefully balanced that, on average, a player finds three items. We simulate a data set for one player repeating the level 30 times and run our first Poisson model, which is a plain GMM.

```
set.seed(6)
D_Pois <-
  tibble(Obs = 1:30,
  items_found = rpois(30, lambda = 3.4))

D_Pois %>%
  ggplot(aes(x = items_found)) +
  geom_histogram()
```




```
M_Pois <-
  stan_glm(items_found ~ 1,
           family = poisson,
           data = D_Pois)
```

```
fixef(M_Pois)
```

model type	fixef	center	lower	upper
object	fixef Intercept	1.31	1.16	1.48

Poisson distributions have only one parameter λ (lambda), has a direct interpretation as it represents the expected mean (and variance) of the distribution. On the contrary, the regression coefficient is on a logarithmic scale to ensure it has no boundaries. To reverse to the scale of measurement, we use the exponential function is the *mean function* 6.1.1:

```
fixef(M_Pois, mean.func = exp)
```

model type	fixef	center	lower	upper
object	fixef Intercept	3.72	3.18	4.41

The exponentiated intercept coefficient can be interpreted as the expected number of items found per session. Together with the credibility limits it would allow the conclusion that the items are slightly easier to find than three per session.

Finally, let's take a look of the formalism of the Poisson GMM:

$$\begin{aligned}\theta_i &= \beta_0\\ \mu_i &= \exp(\theta_i)\\ y_i &\sim \text{Pois}(\mu_i)\end{aligned}$$

In linear models, the first equation used to directly relate fitted responses μ_i to the linear term. As any linear term is allowed to have negative results, this could lead to problems in the last line, because Poisson λ is strictly non-negative. *Linear predictor* θ_i is taking the punches from the linear term and hands it over to the fitted responses μ_i via the exponential function. This function takes any number and returns a positive number, and that makes it safe for the last term that defines the pattern of randomness.

6.2.1.1 Speaking multipliers

To demonstrate the interpretation of coefficients other than the intercept (or absolute group means), we turn to the more complex case of the infusion pump study. In this study, the deviations from a normative path were counted as a measure for error-proneness. In the following regression analysis, we examine

the reduction of deviations by training sessions as well as the differences between the two devices. As we are interested in the improvement from first to second session and second to third, successive difference contrasts apply 4.3.3.

```
attach(IPump)
```

```
M_dev <-
  stan_glmer(deviations ~ Design + session + session:Design +
               (1 + Design + session|Part) +
               (1 + Design|Task) +
               (1|Obs), ## observation-level ramdom effect
             family = poisson,
             data = D_pumps)
```

Note that in order to account for over-dispersion, observation-level random effect (`1|Obs`) has been used, see 6.2.3. For the current matter, we only need to inspect population-level coefficients:

```
fixef(M_dev)
```

fixef	center	lower	upper
Intercept	0.829	0.233	1.405
DesignNovel	-1.522	-2.307	-0.740
session	-0.236	-0.337	-0.133
DesignNovel:session	-0.075	-0.244	0.090

These coefficients are on a logarithmic scale and cannot be interpreted right away. By using the exponential mean function, we reverse the logarithm and obtain the following table:

```
fixef(M_dev, mean.func = exp)
```

fixef	center	lower	upper
Intercept	2.291	1.262	4.076
DesignNovel	0.218	0.100	0.477
session	0.790	0.714	0.875
DesignNovel:session	0.927	0.784	1.094

The intercept now has the interpretation as the number of deviations with the legacy design in the first session. However, all the other coefficients are no

longer summative, but *multiplicative*. It would therefore be incorrect to speak of them in terms of differences.

$$\begin{aligned}\mu_i &= \exp(\beta_0 + x_1\beta_1 + x_2\beta_2) \\ &= \exp(\beta_0)\exp(x_1\beta_1)\exp(x_2\beta_2)\end{aligned}$$

It has alwyays been more natural to speak of these effects in multiplicative form. If we would say "With the novel interface 1.8 fewer errors are being made", that means nothing. 1.8 fewer than what? Instead, the following statements make perfect sense:

1. In the first session, the novel design produces 2.291 *times* the deviations than with the legacy design.
2. For the legacy design, every new training session reduces the number of deviations *by factor* 0.79

3. The reduction rate per training session of the novel design is *92.73% as compared to the legacy design.

```
detach(IPump)
```

To summarize: reporting coefficients on the linearized scale is usually not an option As we are not used to think in logarithmic terms, any quantitative message would get lost. By applying the mean function, we get back to the original scale. As it turns out, speaking multiplicative is natural and conveys more information. As we will see next, linearized scales can even describe non-linear relationships, such as training progress.

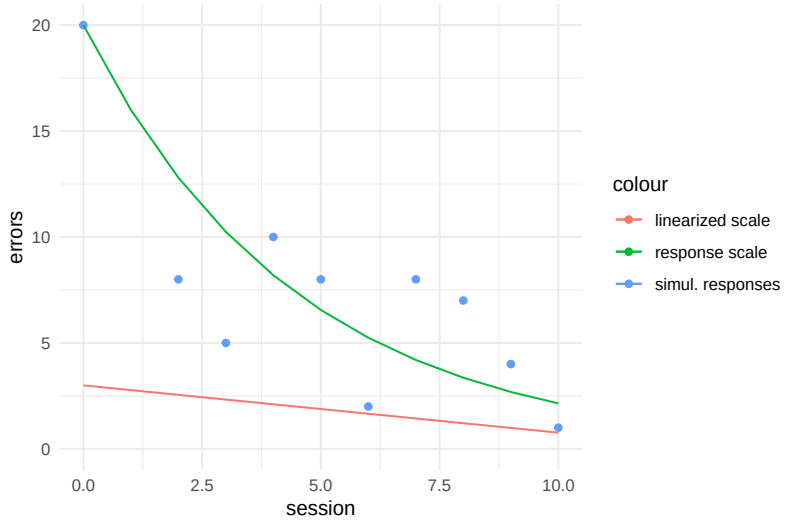
6.2.1.2 Linearizing learning curves

The Achilles heel of Gaussian linear models is the linearity assumption. All measures in this universe are finite, which means that all processes eventually hit a boundary. Linearity is an approximation that works well if you stay away from the boundaries. If you can't, saturation effects happen and that means you have to add interaction effects or ordered factors to your model. Recall the IPump case, where we saw a learning curve over three sessions of practice with design Legacy and Novel. Learning curves are non-linear and we had to use an ordered factor model. Learning curves also have a lower boundary, which is often called the asymptote. As we will see now, learning curves can be described by one log-linearized slope coefficient, if the lower boundary is Zero.

The idea of replacing the OFM with a *linear___ized___ regression model (LzRM), *is attractive. For one, with such a model we can obtain valid* forecasts* of the learning process. And second, the LzRM is more *parsimonous* 7.2.1. For any sequence length, an LzRM just needs two parameters: intercept and slope, whereas the OFM requires one coefficient per session.

What if I also told you, that exponential functions make pretty good learning curves? (Review the simulation in 4.3.5) Even the idea of a multiplicative effect bears some good intuition for learning processes, for example, if we say that by every session, errors are reduced to, say 80%, compared to the previous. This can be demonstrated by simulation of a learning experiment. This simulation takes a constant step size of $\log(.8) = -0.223$ on the linearized scale, resulting in a reduction of 20% per session.

```
initial_deviations <- 20
learning_rate <- .8

D_learn <- tibble(session = 0:10,
                  theta = log(initial_deviations) +  session * log(learning_rate),
                  mu = exp(theta),
                  errors = rpois(11, mu))

D_learn %>%
  ggplot(aes(x =  session, y = errors)) +
  geom_point(aes(col = "simul. responses")) +
  geom_line(aes(y = mu, col = "response scale")) +
  geom_line(aes(y = theta, col = "linearized scale")) +
  ylim(0,20)
```



While the linear predictior scale is a straight line, the response scale clearly is a curve-of-diminishing returns. That opens up the possibility that learning the novel pump design has a constant multiplier (or rate) on the response scale, which would result in a constant difference on the linearized scale. In the following, we estimate two Poisson models, one linearized OFM (OzFM) (with stairway dummies 4.3.5) and one LzRM. Then we will assess the model fit (using fitted responses). If the learning process is linear on the log scale, we can expect to see the following:

1. The two step coefficients of the OzFM become similar (they were wide apart for ToT).
2. The slope effect of the LzRM is the same as the step sizes.
3. Both models fit similar initial performance (intercepts)

We estimate both models as usual, with conditional effects for Design:

```
attach(IPump)
```

```
D_agg <-
  D_agg %>%
  mutate(Step_1 = as.integer(session >= 1),
         Step_2 = as.integer(session >= 2))

## Ordered  regression model
M_pois_cozfm <-
  D_agg %>%
  brm(deviations ~ 0 + Design + Step_1:Design + Step_2:Design,
      family = "poisson", data = .)

## Linear regression model
M_pois_clzrm <-
  D_agg %>%
  brm(deviations ~ 0 + Design + session:Design, family = "poisson", data = .)
```

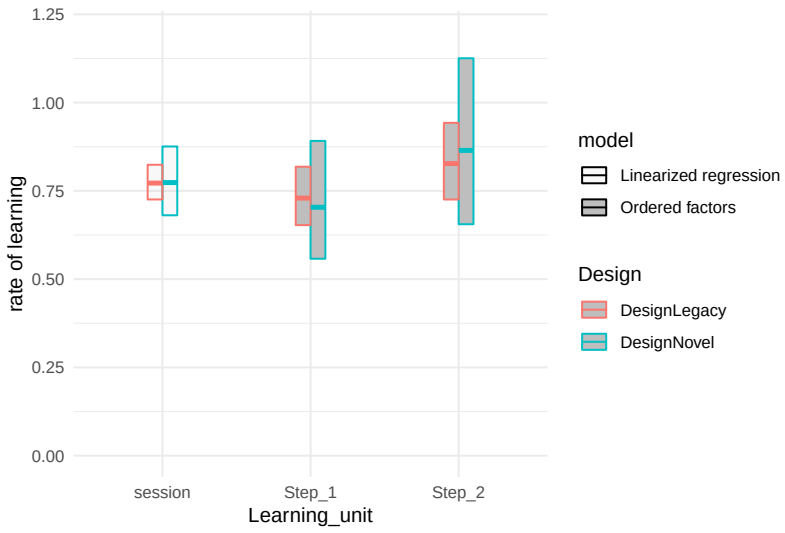
For the question of a constant rate of learning, we compare the one linear coefficient of the regression model with the two steps of teh ordered factor model:

```
T_fixef <-
  bind_rows(
    posterior(M_pois_cozfm),
    posterior(M_pois_clzrm)
  ) %>%
  fixef(mean.func = exp) %>%
```

```
  separate(fixef, into = c("Design", "Learning_unit"), sep = ":") %>%
  mutate(model = if_else(str_detect(model, "lz"),
                         "Linearized regression", "Ordered factors")) %>%
  filter(!is.na(Learning_unit)) %>%
  arrange(Design, Learning_unit, model) %>%
  discard_redundant()

T_fixef
```

model	Design	Learning_unit	center	lower	upper
Linearized regression	DesignLegacy	session	0.772	0.726	0.824
Ordered factors	DesignLegacy	Step_1	0.730	0.653	0.818
Ordered factors	DesignLegacy	Step_2	0.827	0.726	0.943
Linearized regression	DesignNovel	session	0.773	0.681	0.876
Ordered factors	DesignNovel	Step_1	0.704	0.558	0.891
Ordered factors	DesignNovel	Step_2	0.865	0.656	1.126

```
T_fixef %>%
  ggplot(aes(x = Learning_unit, y = center, col = Design,
             alpha = model, ymin = lower, ymax = upper)) +
  geom_crossbar(position = "dodge", width = .2, fill = "grey") +
  labs(y = "rate of learning") +
  ylim(0, 1.2)
```



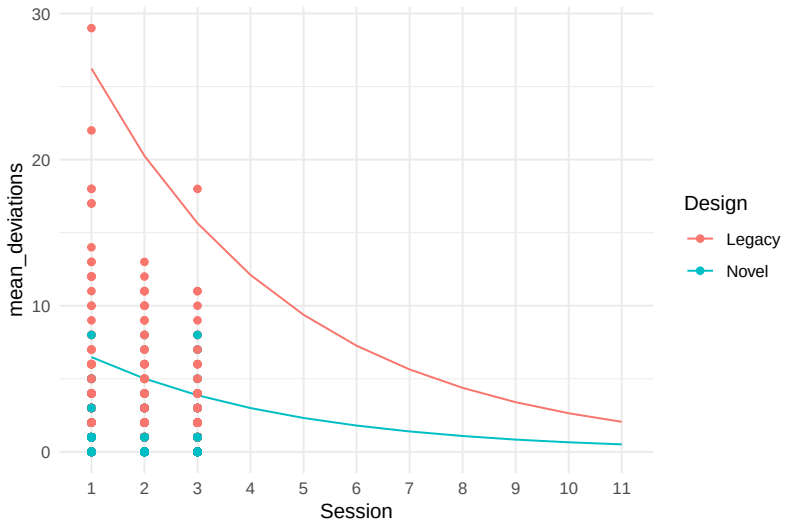
Indeed, the coefficients Step_1 and Step_2 appear to be in a similar region for both designs, although learning at step 1 might be a little bit stronger. The OFM puts a huge amount of uncertainty on the coefficients. For Novel

there even is quite some possibility that the learning rate falls in the region larger than One, which would mean performance gets worse. This makes little sense. What makes sense is that in both models the learning rate is the same for both designs. This indicates that the linearized regression model can even be reduced further, with just one parameter for learning rate (but with conditional intercepts):

```
M_pois_lzrm <-
  D_agg %>%
  brm(deviations ~ 0 + Design + session, family = "poisson", data = .)
```

We will return to this case in section 7.2.3 and show that the unconditional model, with just one learning rate parameter, is the best fit for the data. For now, the bottom line is that variables that a log-linearized model can remove nasty saturation effects and reduce model complexity. In addition, we replaced an ordered factor model with a smooth linear regression model, which has another advantage: The ordered factor model confines us to the number of learning sessions we have observed. From teh OFM, there is no way to predict, how the learning curve would progress with additional sessions. But, with a linearized regression model, we can produce *forecasts* for future sessions. For this purpose, we inject fabricated data (predictors only) into the model, extract the fitted responses and plot the results:

```
D_forecast <-
  expand_grid(session = c(0:10),
              Design = c("Novel", "Legacy"),
              Part = 1:50) %>%
  as_tbl_obs() %>%
  mutate(Session = as.factor(session + 1))

T_pred <-
  post_pred(M_pois_lzrm, newdata = D_forecast) %>%
  left_join(D_forecast, by = "Obs") %>%
  group_by(Design, Session) %>%
  summarize(mean_deviations = mean(value))

T_pred %>%
  mutate(linetype = if_else(as.numeric(Session) <= 3, "retrospective", "forecast")) %>%
  ggplot(aes(x = Session, col = Design, y = mean_deviations)) +
  geom_line(aes(group = Design)) +
  geom_point(aes(y = deviations), data = D_pumps)
```




```
detach(IPump)
```

Normally, fitted responses are just retrospective. Here, we extrapolate the learning curve by fake data and obtain real *forecasts*. We can make more interesting comparison of the two devices. For example, notice that initial performance with Novel is around five deviations. With Legacy this level is reached only in the seventh session. We can say that the Novel design is always seven sessions of training ahead of Legacy.

The conclusion is that log-linearized scales can reduce or even entirely remove saturation effects, such that we can go with a simpler models, that are easier to explain and potentially more useful. Potentially, because we can not generally construct parametric learning curves with log-linear models. The crucial property here is that the lower bound is Zero. Some measures have a positive lower bound, which is constant and known, and can be translated to a lower bound of Zero. For example, path length, the minimum number of steps to find something on the internet is One, Path length can just shifted by One, e.g. `mutate(addional_steps = steps + 1)` to create a lower bound of Zero. This is different for Time-on-Task, which always has a strictly positive lower bound, which we don't know and which probably varies between individuals. Learning curves that approach strictly positive asymptotes have the following mathematical form:

```
# devtools::install_github("schmettow/asymptote")
asymptote::ARY
```

```
## perf ~ ampl * exp(-rate * trial) + asym
```

```
## <environment: namespace:asymptote>
```

The offset to Zero is in the summand `asym`, and because it is a summand this term cannot be linearized in a straight-forward manner. For general learning curves a truly non-linear model is required, not just a linearized. This can be constructed with the Brm engine, but I considered this to be beyond the scope of this book.

6.2.2 Logistic (aka Binomial) regression (#logistic-reg)

In the last section, we have seen how Poisson regression applies, when outcome variables are count numbers. More strictly, Poisson regression applies to count data, when there is no upper limit to counts (or if this limit is extremely large, as in the Smart Smurfer example). When the outcome variable is counts, but an upper limit exists and is known, *logistic regression* is an appropriate model. Such a situation often arises, when the counts are successes in a fixed number of trials. More brief, logistic regression has the following properties:

1. The outcome variable has a zero lower bound and a fixed upper bound, e.g. number of trials k.
2. The linear predictors are on a *logit scale* also called *log-odds*, which is reversed by a *logistic function*.
3. The random component follows a *binomial distribution*.
4. Due to the former, the variance of randomness is largest at $\mu = 0.5$ or $\eta = 1$ and declines towards both boundaries, taking a characteristic cigar shape.

Logistic regression applies for discrete outcomes, just like Poisson regression. The difference is that logistic regression has a finite number of possible outcomes, which is teh number of trials plus one. In the following section, I will first introduce logistic regression for when there is only one trial per observation, with two possible outcomes. That is called a *dichotomous outcomes*. This has some interesting applications for outcomes that are not exactly quantitative, like any other outcome variable covered in this book. Namely, we can apply logistic regression to variables that denote two classes. In the subsequent section, we will look at logistic regression for when there is more than one trial. The most difficult part of logistic regression is to report the estimated coefficients in an intelligible manner, which will be covered in the final section.

6.2.2.1 Dichotomous outcomes

The most simple form of successes-in-trials measure is when there is only one trial. This is called a *dichtotomous* variable, and that is very common:

- a user is successful at a task, or fails
- a visitor returns to a website or does not
- a usability problem is discovered or remains unseen
- a driver brakes just in time or crashes
- a customer recommends a product to a friend or does not
- a user starts searching on a website by keyword or by traversing links

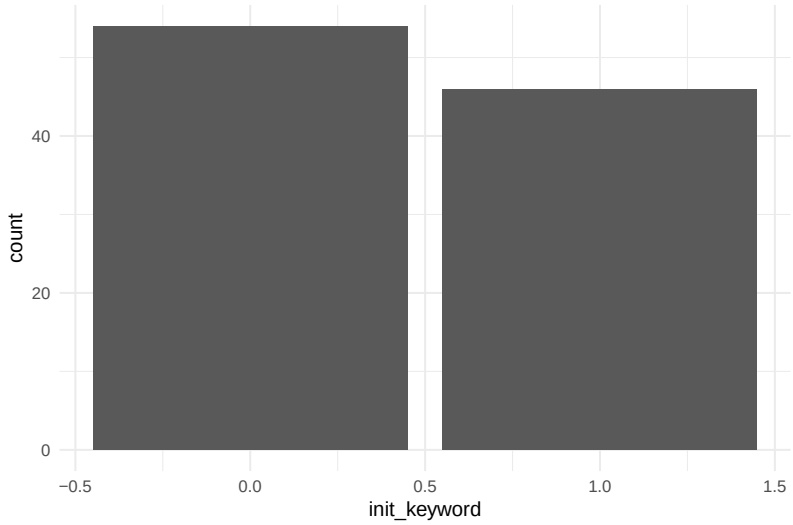
Often, dichotomous outcome variables have a quantitative notion in the sense of more or less desireable.. When the outcome casts a positive light on the design, by convention it is coded as 1, otherwise 0. But, the dichotomy can also be two equal alternatives, such as whether a user starts a web inquiry by keyword search or by following a link. Let's take this as an example. Research on search strategies of web users revealed that they are quite ecclectic regarding their method to find a piece of information. In particular, most people use keyword search and link navigation at occasion. Web users are also known to be impatient companions, who build a first judgment very quickly and swiftly turn to a competitor's site, when the first impression leaves something to be desired. Therefore, it can be valuable to know what method the majority of users prefer, initially.

For this purpose, we can classify users by what method they start with when given a search task during a usability study. As there exist only two options, keyword search or following links, we can capture the outcome in a dichotomous response variable. Below is the simulation of a small data set, where 40% of users initially prefer keyword search:

```
set.seed(42)
D_web <- tibble(visitor = as.factor(1:100),
                init_keyword = rbinom(100, 1, .4))

D_web %>% sample_n(6) %>% kable()
```

visitor	init_keyword
63	1
22	0
99	1
38	0
91	1
92	0

```
D_web %>%
  ggplot(aes(x = init_keyword)) +
  geom_bar()
```



For estimating the proportion of the two classes of users, we run a logistic regression grand mean model and inspect the coefficient table. Note that logistic regression is not called after its shape of randomness and we have to pass the *binomial family* to the regression engine.

```
M_web <- D_web %>% stan_glm(init_keyword ~ 1, data = .,
                            family = binomial)
```

```
clu(M_web)
```

parameter	type	fixef	center	lower	upper
Intercept	fixef	Intercept	-0.165	-0.562	0.261

Clearly, the Intercept parameter is not a proportion, as that forbids negative values. Like with Poisson regression, the coefficient is on a linearized scale. As proportions are in a range from zero to one, a particular link function is sits between the measures and the linear term, stretching the bounded into an unbounded range. For logistic regression, the *logit* functions maps the fitted responses $\mu_i \in [0; 1]$ onto the *linear predictor* scale $\eta_i \in [-\infty; \infty]$.

$$\eta_i = \text{logit}(\mu_i) = \log \frac{\mu_i}{1 - \mu_i}$$

Note that the fraction $\frac{\mu_i}{1-\mu_i}$ is the proportion of keyword search divided by the proportion of following links and is called an *odds*. The logit function is

therefore often called *log-odds*. In section 6.2.2.3, we will see how we can report logistic regression results as odds. In the case here, we can directly report the results as proportions, which requires to apply the *mean function*, which is the inverse of the logit, also known as the *logistic function*:

$$\mu_i = \text{logit}^{-1}(\eta_i) = \frac{\exp \eta_1}{\exp \eta_i + 1}$$

In GMM, $\eta_i = \beta_0$ and we can directly obtain the estimated proportion by applying the logistic function to the Intercept. The `clu` command lets you pass on a mean function. However, this logistic mean function is only useful for GMM and other absolute group means models, as we will see in 6.2.2.3.

```
posterior(M_web) %>% clu(mean.func = inv_logit)
```

model	parameter	type	fixef	center	lower	upper
M_web	Intercept	fixef	Intercept	0.459	0.363	0.565

From the GMM we retrieve one estimate that reflects the proportion to start by keyword search. As a side note, proportions could also be called probabilities, like "with 40% probability a user starts by keyword search." However, I urge anyone to avoid speaking of logistic regression coefficients as probabilities. While mathematically this is correct, for the audience it can easily cause confusion with certainty or, beware of this, the p-value.

The apt reader may have noticed that the returners data set has been simulated with an exact return rate of 40%. Despite the sample size of 100, the center estimate seems rather off and hampered by considerable uncertainty. In computer science jargon, every dichotomous observation accounts to a *bit*, which is the smallest amount of information ever possible. Because the information of a single dichotomous observation is so sparse, large samples are important when dealing with dichtomous outcomes. Large samples can mean testing many users, or giving every user more than one trial.

6.2.2.2 Successes in a number of trials

If we repeatedly observe a dichotomous response, we can summarize the results as *successes-in trials*, like:

```
responses <- c(0, 1, 1, 1, 0, 1)
cat(sum(responses), "successes in", length(responses), "trials")
```

```
## 4 successes in 6 trials
```

Imagine we had conducted an extended version of the previous experiment, where users get set of ten tasks and we observe their initial behaviour every time they open a new website. As such tasks sometimes take very long, it may also happen that a participant cannot finish all ten tasks within time. That means, we potentially have a different number of attempts per participant, which we simulate as:

```
set.seed(42)
D_web_ex <-
  tibble(
    Session = 1:100,
    trials = round(runif(100, 7, 10), 0),
    init_keyword = rbinom(100, trials, .4),
    init_link = trials - init_keyword) %>%
  mascutils::as_tbl_obs()

D_web_ex
```

In order to estimate a model with more than one trial per observation, we have to specify how many trials there were. That is not done directly, but via the number of "failures", in this case this is the number of trials where a link was followed. The response side of the model formula takes this in as an array with two columns, which is generally constructed as `cbind(successes, failures)`. We estimate a GMM:

```
M_web_ex <- stan_glm(cbind(init_keyword, init_link) ~ 1,# <--
                     family = binomial,
                     data = D_web_ex)
```

```
fixef(M_web_ex, mean.func = inv_logit)
```

model type	fixef	center	lower	upper
object	fixef Intercept	0.407	0.375	0.441

With repeated trials the estimate for tzhe proportion of initial keyword search gets much closer to real value and credibility intervals tighten up, too. By using teh inverse logit, we can readily report the results as proportions. But, make no mistake, when predictors come into play and additional coefficients are being estimated, reporting proportions does no longer work. Instead, we have to learn to speak in odds.

6.2.2.3 Talking odds

When presenting results of a statistical analysis, the linear predictor is likely to cause trouble, at least when the audience is interested in real quantities. Coefficients on a logit-linearized scale have only very general intuition:

- zero marks a 50% chance
- positive values increase the chance, negative decrease
- bigger effects have larger absolute values

That is sufficient for purely ranking predictors by relative impact (if on a comparable scale of measurement), or plain hypothesis testing, but it does not connect well with quantities a decision maker is concerned with. Let's see this at the example of the infusion pump study:

1. What is the expected frequency of failure on first use?
2. The novel design reduces failures, but is it sufficient?
3. Is frequency of failures sufficiently reduced after two training sessions?

In the comparison of two medical infusion pumps (@ref(slope_RE)) 25 nurses completed a set of eight tasks repeatedly over three sessions. In @ref(slope_RE) a multi-level model was estimated on the workload outcome. It is tempting to apply the same structural model to success in task completion, using binomial random patterns and logit links.

```
completion ~ Design*Session + (Design*Session|Part) + (Design*Session|Task)
```

Such a model is practically impossible to estimate, because dichtomous variables are so scarce in information. Two populations encounter each other in the model: participants and tasks, with 6 observations per combination (6 bit). We should not expect to get reasonably certain estimates on that level and, in fact, the chains will not even mix well. The situation is a little better on the population level: every one of the six coefficients is estimated on 400 bit of raw information. We compromise here by estimating the full model on population level and do only intercept random effects to account for gross differences between participants and tasks.

```
attach(IPump)
```

```
M_cmpl <-
  D_pumps %>%
  stan_glmer(completion ~ Design * Session +
               (1|Part) + (1|Task),
             family = binomial,
             data = .)
```

```
fixef(M_cmpl)
```

fixef	center	lower	upper
Intercept	1.388	0.069	2.755
DesignNovel	0.404	0.092	0.716
Session2-1	0.676	0.147	1.212
Session3-2	-0.069	-0.616	0.463
DesignNovel:Session2-1	-0.300	-1.055	0.478
DesignNovel:Session3-2	0.287	-0.485	1.068

The result is one absoulte group mean, the Intercept, and five effects, which are mean differences on the logit-linearized scale η_i. This is a boundless scale, where we can freely sum over effects to obtain group means. If we want to report group means, we can use the invers elogit function to obtain proportions, but for that we have to *first do the linear combination followed by the transformation*, for example:

- the completion rate in the first legacy session is 0.8
- in novel/session 1: `logist(Intercept + DesignNovel)` = 0.857
- in novel/session 2: `logist(Intercept + DesignNovel + Session2-1 + DesignNovel:Session2-1)` = 0.897
- in legacy/session 3: `logist(Intercept + DesignNovel + Session2-1)` = 0.88

Above we have used the mean logistic mean function to elevate the absolute group means to proportions. This is an intuitive scale, but unfortunately, the mean function does not apply to individual effects. It is for example, *incorrect* to apply it like: "the novel pumps proportion of failures in the first session increases by `logist(DesignNovel)` = 0.6".

Now, it comes into play that the logit is a compound function, the logarithm of an odds. The inner part of the function, the *odds*, are the chance of success divided by the chance of failure. Especially in the anglo-american culture, odds are a rather common way to express ones chances in a game, say:

- odds are 1 against 1 that the coin flip produces Head. If you place €1 on Head, I put €1 on tail.
- odds are 1 against 12 that Santa wins the dog race. If you place 1€ on Santa, I place €12 against.

If the coefficients are log-odds, than we can extract the odds by the inverse of the logarithm, the exponential function, like in the following call of `fixef`:

```
fixef(M_cmpl, mean.func = exp)
```

fixef	center	lower	upper
Intercept	4.008	1.071	15.72
DesignNovel	1.497	1.096	2.05
Session2-1	1.966	1.159	3.36
Session3-2	0.933	0.540	1.59
DesignNovel:Session2-1	0.741	0.348	1.61
DesignNovel:Session3-2	1.333	0.615	2.91

But is it legitimate to apply the transformation on individual coefficients in order to speak of changes of odds? The following arithmetic law tells that what is a sum on the log-odds scale, is multiplication on the scale of odds:

$$\exp(x + y) = \exp(x) \exp(y)$$

Consequently, we may speak of changes of odds using *multiplicative language*:

- If you place €100 on failure in the next task with the legacy design in session 1, I place €400.761 on success.
- The odds of success with the novel design increase by *factor* 1.497. Now, I would place $400.761 \times 1.497 =$ €599.962 on success.
- On success with the novel design in session 2, I would place $400.761 \times 1.497 \times 1.966 \times 0.741 =$ €874.002 on success.

Once, we have transformed the coefficients to the odds scale, we can read coefficients as multipliers and speak of them in hard currency.

```
detach(IPump)
```

To summarize: Logistic regression applies when the basic observations falls into two classes. For any research design involving such outcomes, repetition is highly recommended, and outcomes can be summarized into successes-in-trials. Reporting coefficients on the logit scale is only useful when nobody is interested in intelligible effects sizes. How to report the results depends on the research question. If one is interested in proportions per group, the inverse logit applies to the absolute group meeans and this can be easily understood. If one wants to talk about effects or differences, such as the amount of improvement with a novel design, only the logarithm of log-odds is inversed, and effects are reported as odds. Depending on teh audience, this may be more or less intuitive, but it can always be embedded in a wager for illustration.

While logistic regression is not easy to master it has important areas of application, as well as interesting extensions.

- In epidemiologic research, logistic regression is the undispensible tool for several central outcomes, such as hospitalization, mortality, infection and recovery.
- In psychometrics, the famous Rasch model applies for measuring a persons ability by teh number of correct answers in a test. A Rasch model is just a cross-classified multilevel logistic regression 5.8.4.
- If the outcome is a classification with more than two classes, *multi-nomial regression* is an extension of logistic regression.
- In section 4.3.5, we will encounter *ordinal logistic regression*, which applies for classifications with an order, such as responses on Likert scales.

One frequent problem when using logistic regression on successes-in-trials outcomes is that the assumption of a Binomial shape of randomness is violated by *over-dispersion*. Like Poisson distributions, Binomial distributions have a variance tightly linked to the mean, but frequently there is more variance than allowed, for example when tasks or test items vary in difficulty. In the following section two solutions to the problem are introduces: *beta-binomial regression* and *observation-level random effects*.

6.2.3 Modelling overdispersion

Poisson and binomial distributions are one-parameter distributions. As there is only one parameter, it is impossible to choose location and dispersion independently. In effect, both properties are tightly entangled. For Poisson distributions they are even the same.

$$\begin{aligned} Y &\sim \text{Poisson}(\lambda) \\ \text{Var}(Y) &= \text{Mean}(Y) = \lambda \end{aligned}$$

For binomial variables, mean and variance both depend on probability p and are entangled in cigar shaped form, as the dispersion shrinks when approaching either two boundaries. Binomial variance is also affected by the number of trials k, but that hardly matters as the value of k is usually not up for estimation, but known a prior.

$$\begin{aligned} Y &\sim \text{p, k}(\lambda) \\ \text{Mean}(Y) &= kp \\ \text{Var}(Y) &= kp(1-p) \\ &= \text{Mean}(Y)(1-p) \end{aligned}$$

In real count data, we often see similar relationship between variance and mean, except that variance is inflated by some additional positive factor, which is called *overdispersion*. Poisson or Binomial distribution cannot render inflated data, and using them on over-dispersed data is a serious mistake. Fortunately, there exist two solutions to the problem, which I will introduce in the following three sections. In the first two sections, we will *replace the one-parameter distribution with a two-parameter distribution*, where the second parameter represents the factor of variance inflation. The second method is to use *observation-level ramdom effects*, which draws from the multi-level modelling toolbox.

Let me give you an example to illustrate the two methods. It is common saying that some people attract mosquito bites more than others. But is that really true? A simple lab experiment would do to test the "Sweet Blood" theory. A sample of participants are exposed to a pack of mosiquitos under carefully controlled conditions (time of day, environmental condition, hungriness of mosquitos). We don't know the mechanisms that makes the blood sweeter, and hence cannot measure it. In the simulation below, it is just assumed that there is a such a property, but in a real study we would not know. Know imagine a study, where mosquito bites have been counted on 200 boy scouts after an expedition.

The following simulation function works by using a two-parameter distribution. Negative-binomial distributions are discrete distributions with a lower bound of zero, just like Poisson distributions. They also have the same location parameter μ, but a new parameter `size`, which re-scales the scale of measurement. When the scale of measurement is down-scaled, the distribution becomes relatively wider. When size approaches infinity, we are left with a plain Poisson variance. The following data simulation samples Sweet-blood data from a negative-binomial distribution with a size of 6.

```
set.seed(42)
N = 200
avg_sweet <- 6
size = 3

Sweet_blood_nbin <- tibble(Method = "NegBinomial",
                           bites = rnbinom(n = N,
                                           mu = avg_sweet,
                                           size = size))

Sweet_blood_nbin %>%
  summarize(mean(bites), var(bites))
```

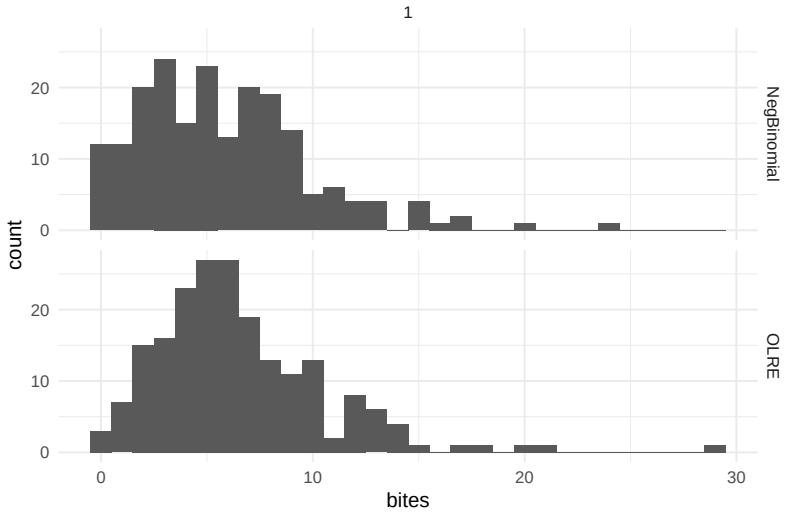
mean(bites)	var(bites)
5.83	16.7

The next simulation first creates an observation-level indicator for blood sweetness, which in real data would not be known to the researcher; it is therefore a *latent variable*. Sweetness is sampled from a Gaussian distribution, that means we can get negative and positive values. Then Poisson random numbers are generated, but `lambda` is not a fixed value. Instead, every observation is sampled with a different value for sweetness. The exponentiation is simply to transform the Gaussian variable to the non-negative range of `lambda`. Taking the logarithm of the Gaussian `mu` puts the resulting distribution at approximately(!) the desired location of average sweetness.

```
set.seed(42)
N = 200
avg_sweet <- 6
size = 4


Sweet_blood_olre <-
  tibble(Method = "OLRE",
         sweetness = rnorm(N, mean = log(6), sd = .5),
         bites = rpois(N, lambda = exp(sweetness)))

Sweet_blood_olre %>%
  summarize(mean(bites), var(bites))
```

mean(bites)	var(bites)
6.56	16.5

```
bind_rows(Sweet_blood_nbin,
          Sweet_blood_olre) %>%
  ggplot(aes(x = bites)) +
  geom_histogram() +
  facet_grid(Method~1)
```



The two methods for dealing with over-dispersion reverses either method of simulation. Either we choose a more flexible distribution, or we estimate the residuals on the linearized scale. The first method can be recommended, because it is leaner. Only one parameter is added, whereas OLRE results in one linearized residual for every observation. The advantage of OLRE is more of a conceptual kind. Not only does it appealing for researchers who are familiar with multi-level models, it also reverbs with the well-known concept of Gaussian residuals and lets us compare (linearized) residual variance to (linearized) ramdom effects variances.

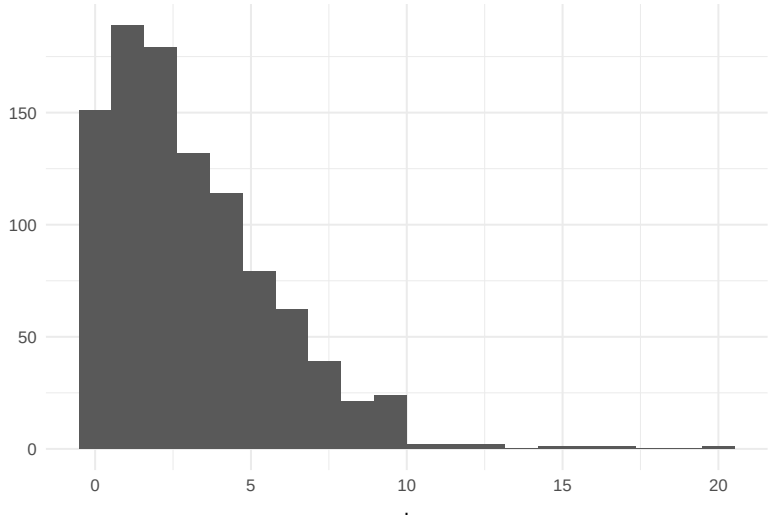
–>

6.2.3.1 Negative-binomial regression for overdispersed counts

When Poisson regression is used for overdispersed count data, the model will produce reliable center estimates, but the credibility limits will be too narrow. The model suggests better certainty than there is. To explain that in simple terms: The model "sees" the location of a measure, which makes it seek errors in a region with precisely that variance. There will be many measures outside the likely region, but the model will hold on tight, regard these as (gradual) outliers and give them less weight. A solution to the problem is using a response distribution with *two parameters*. A second parameter usually gives variance of the distribution more flexibility, although only Gaussian models can set it loose, entirely. For the Poisson case (i.e. counts without an upper limit) *negative binomial distributions* do the job, for binomial distributions the beta-binomial applies. Both distributions are so-called *mixture distributions*. In mixture distributions, the parameter of the "outer" distribution is

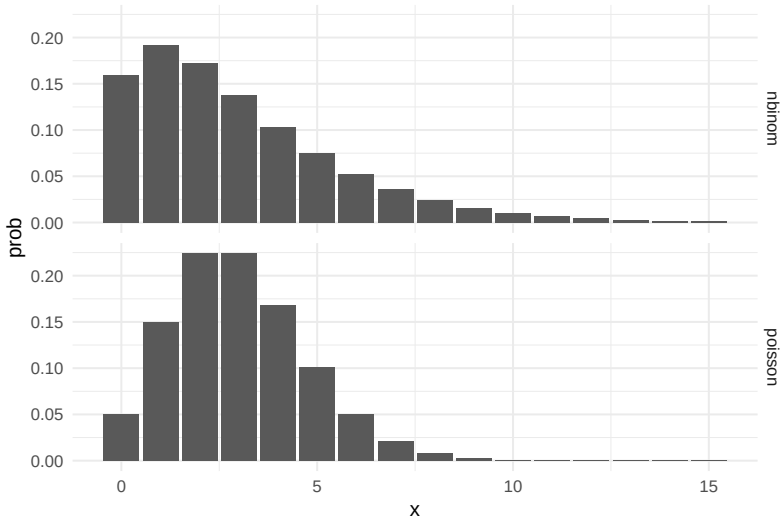
not constant, but allowed to vary by a distribution itself. Under this perspective, negative binomial distribution is equivalent to a Poisson distribution, if we let parameter λ follow a gamma distribution, like this:

```
rnegbinom <- function(n, mu, size){
  shape <- size
  scale <- mu/size
  lambdas <- rgamma(n, shape = shape, scale = scale)
  rpois(n, lambda = lambdas)
}

rnegbinom(1000, mu = 3, size = 2) %>%  qplot(bins = 20)
```



The figure below shows a negative binomial distribution and Poisson distribution with the same mean. The additional parameter size reduces the scale and makes the distribution wider.

```
tibble(x = 0:15,
           nbinom = dnbinom(x, mu = 3, size = 2),
           poisson  = dpois(x, lambda = 3)) %>%
  gather(distribution, prob, -x) %>%
  ggplot(aes(x = x, y = prob)) +
  facet_grid(distribution ~ .) +
  geom_col(position = "dodge")
```

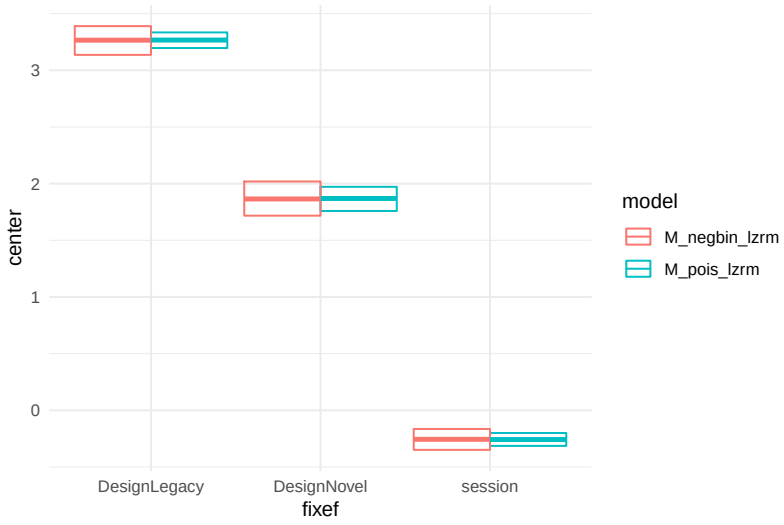


In 6.2.1.2 we have seen how log-linearization can accomodate learning curves, using a Poisson model. It is very likely that this data is over-dispersed and that the Poisson model was not correct. To demonstrate overdispersion, we estimate the unconditional learning curve model one more time, with a negative-binomial pattern of randomness:

```
attach(IPump)
```

```
M_negbin_lzrm <-
  D_agg %>%
  brm(deviations ~ 0 + Design + session, family = "negbinomial", data = .)
```

```
bind_rows(
  posterior(M_pois_lzrm),
  posterior(M_negbin_lzrm)) %>%
  filter(type == "fixef") %>%
  clu() %>%
  ggplot(aes(x = fixef, y = center, ymin = lower, ymax = upper, col = model)) +
  geom_crossbar(position = "dodge")
```



We observe that the center estimates are precisely the same. But, credibility limits are much wider with an underlying negative-binomial distribution. In the CLU table we also see that the neg-binomial model got another parameter `phi`, which is another name for the size parameter, controlling over-dispersion relative to a Poisson distribution as:

$$\text{Variance} := \mu + \mu^2/\phi$$

Due to the reciprocal term, the *smaller* ϕ gets, the *more* overdispersion had to be accounted for. From this formula alone it may seem that neg-binomial distributions could also account for under-dispersion, when we allow negative values. But, in most implementations ϕ must be non-negative. That is rarely a problem, as under-dispersion only occurs under very rare circumstances. Over-dispersion in count variables in contrast, is very common, if not ubiquitous. Negative-binomial regression solves the problem with just one additional parameter, which typically need not be interpreted. Reporting on coefficients uses the same principle as in plain Poisson regression: inversion by exponentiation and speaking *multiplicative*.

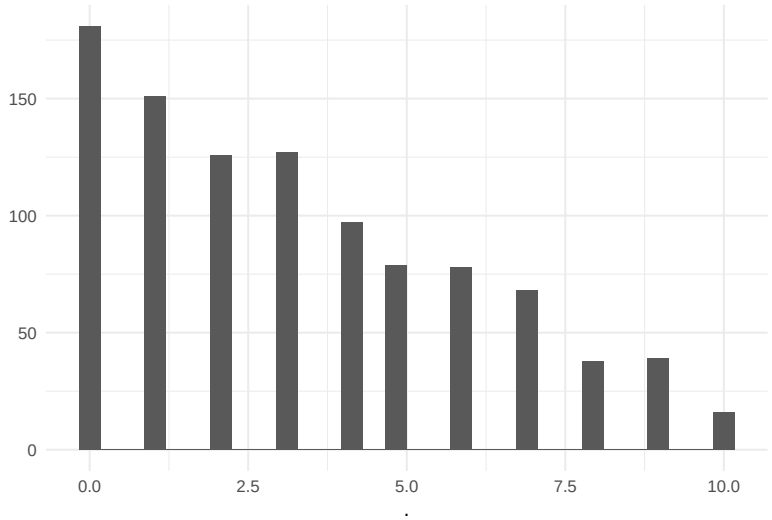
```
detach(IPump)
```

6.2.3.2 Beta-binomial regression for successes in trials

Beta-binomial regression follows a similar pattern as neg-binomial. A two parameter distribution allows to scale up the variance relative to a binomial

model **??**. A beta-binomial distribution is created by replacing binomial parameter p by a *betadistribution*, with parameters a and b:

```
rbetabinom <- function(n, size, a, b) rbinom(n, size, rbeta(n, a, b))
rbetabinom(1000, 10, 1, 2) %>% qplot()
```



The Brms regression engine currently only implements the negative binomial, but not the beta-binomial family. That is a good opportunity to applaud the author of the Brms package for his ingenious architecture, which allows custom families to be defined by the user. The only requirement is that the distribution type is implemented in Stan [%Stan], which is the underlying general-purpose engine behind Brms. The following code is directly taken from the Brms documentation and adds beta-binomial distribution to the available shapes of randomness.

```
# define a custom beta-binomial family
beta_binomial2 <- custom_family(
  "beta_binomial2", dpars = c("mu", "phi"),
  links = c("logit", "log"), lb = c(NA, 0),
  type = "int", vars = "trials[n]"
)

# define custom stan functions
bb_stan_funs <- "
  real beta_binomial2_lpmf(int y, real mu, real phi, int N) {
    return beta_binomial_lpmf(y | N, mu * phi, (1 - mu) * phi);
  }
```

```
  int beta_binomial2_rng(real mu, real phi, int N) {
    return beta_binomial_rng(N, mu * phi, (1 - mu) * phi);
  }
"
```

Note that Beta-binomial distribution are usually parametrized with two shape paramneter a and b, which have a rather convoluted relationship with mean and variance. For a GLM a parametrization is required that has a mean parameter (for μ_i). Note, how the author of this code created a `beta_binomial2` distribution family, which takes μ and a scale parameter ϕ.

Defining the two function is sufficient to estimate beta-binomial models with Brms. In the following I simulate two outcomes from nine trials, `y` is sampled from a beta-binomial distribution, whereas `ybin` is from a Binomial distribution. Both have the same mean of .1 (10% correct). Subsequently, a beta-binomial and a binomial grand mean models are estimated.

```
set.seed(42)
D_betabin <- tibble(y = VGAM::rbetabinom(1000, 9, prob = .1, rho = .3),
                    n = 9) %>%
  as_tbl_obs()

M_betabin <-
  D_betabin %>%
  brm(y | trials(n) ~ 1, family = beta_binomial2,
      stan_funs = bb_stan_funs, data = .)

M_bin <- brm(y |trials(n) ~ 1, family = "binomial", data = D_betabin)
```

The following CLU table collects the estimates from both models, the true beta-binomial and the binomial, which does not account for over-dispersion in the data.

```
bind_rows(
  posterior(M_bin),
  posterior(M_betabin)) %>%
  clu(mean.func = inv_logit)
```

model	parameter	type	fixef	center	lower	upper
M_betabin	b_Intercept	fixef	Intercept	0.706	0.663	0.747
M_betabin	phi	shape	NA	0.998	0.983	1.000
M_bin	b_Intercept	fixef	Intercept	0.706	0.679	0.734

When comparing the two intercept estimates, we notice that the center estimate is not affected by over-dispersion. But, just like with Poisson models, the binomial model is too optimistic about the level of certainty.

We have seen To summarize: one-parameter distributions usually cannot be used to model count data due to extra variance. One solution to the problem is to switch to a family with a second parameter. These exist for the most common situations. When we turn to modelling durations, we will use the Gamma family to extend the Exponential distribution 6.3.1. Gamma distributions have another problem: while extra variance can be accounted by a scale parameter, we will see that another property of distribution families can be to rigid, the skew. The solution will be to switch to a three-parameter distribution family to gain more flexibility.

Another technique to model over-dispersion does not require to find (or define) a two-parametric distribution. Instead, *observation-level random effects* borrow concepts from multi-level modelling and allow to keep the one-parameter distributions.

6.2.3.3 Using observation-level random effects

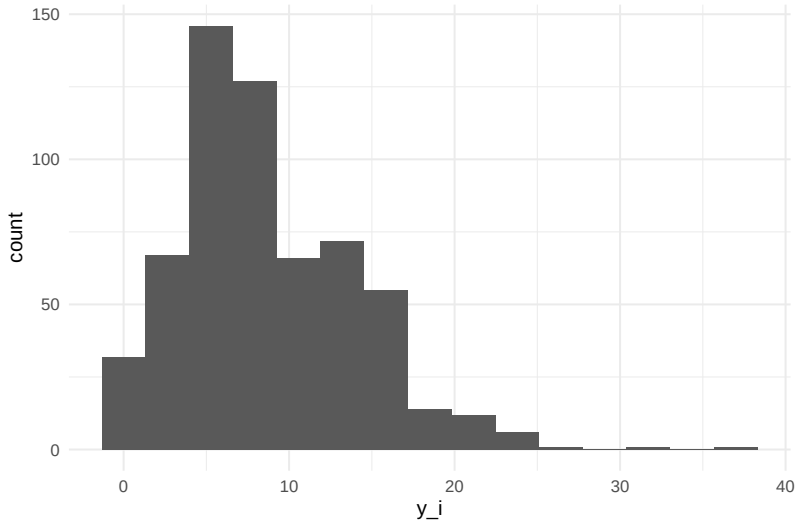
As we have seen in chapter 5, random effects are often interpreted towards variance in a population, with a Gaussian distribution. On several occasions were multi-level models used to separate sources of variance, such as between teams and participants in CUE8. *Observation-level random effect* (OLRE) use the same approach by just calling the set of observation a *a population*.

Using random effects with GLMs is straight-forward, because random effects (or their dummy variable representation, to be precise), are part of the linear term, and undergo the log or logit linearization just like any population-level effect in the model.

For demonstration of the concept, we simulate from an overdispersed Poisson grand mean model with participant-level variation and observation-level variation.

```
sim_ovdsp <- function(
  beta_0 = 2,    # mu = 8
  sd_Obs = .3,
  sd_Part   = .5,
  N_Part = 30,
  N_Rep  = 20,
  N_Obs =  N_Part * N_Rep,
  seed = 1){
  set.seed(seed)
  Part <- tibble(Part = 1:N_Part,
```

```
                     beta_0p = rnorm(N_Part, 0, sd_Part)) ## participant-level RE
  D <- tibble(Obs  = 1:N_Obs,
                    Part = rep(1:N_Part, N_Rep),
                    beta_0i = rnorm(N_Obs, 0, sd_Obs),   ## observeration-level RE
                    beta_0 = beta_0) %>%
    left_join(Part) %>%
    mutate(theta_i = beta_0 + beta_0p + beta_0i,
           mu_i    = exp(theta_i),               ## inverse link function
           y_i     = rpois(N_Obs, mu_i))
  D %>% as_tbl_obs()
}

D_ovdsp <- sim_ovdsp()

D_ovdsp %>%
  ggplot(aes(x = y_i)) +
  geom_histogram(bins = 15)
```



The above code is instructive to how OLREs work:

1. A participant-level random effect is created as `beta_0p`. This random effect can be recovered, because we have repeated measures. This variation will not contaminate Poisson variance.
2. An observation-level random effect is created in much the same way.
3. Both random effects are on the linearized scale. The linear predictor `theta_i` is just the sum of random effects (and Intercept). It could take negative values, but ...

4. ... applying the inverse link function (`exp(theta_i)`) ensures that all responses are positive.

The extra variation comes from two sources: participant-level, with repeated measures and observation-level, without repeated measures. When random effects were introduced in the previous chapter, I said that estimating random effects requires repeated measures. For Gaussian models, that is true (and probably also for other two-parametric families). In contrast, one-parameter distributions precisely specify their variance at any point of location and therefore the OLREs can be recovered. The following model contains an participant-level random effects and an OLRE:

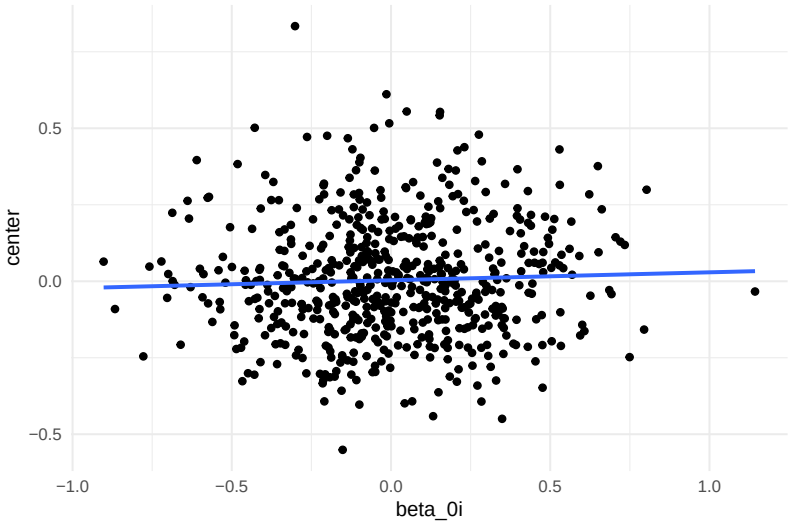
```
M_ovdsp <-
  D_ovdsp %>%
  stan_glmer(y_i ~ 1 + (1|Part) + (1|Obs), data = .,
             family = poisson)
```

```
grpef(M_ovdsp)
```

re_factor	center	lower	upper
Obs	0.305	0.264	0.350
Part	0.536	0.403	0.719

Let's first take a look at the two random effect standard errors above. It seems that we got a fair recovery on the center estimates (for standard deviation). For OLRE certainty is also good, even better than for the participant-level, which is simply due to the fact that there are more levels. Random effect variation is accurately recovered from the simulated data, but can we also recover the full vector of factor levels? The following plot shows that In the following I am extracting observation-level random effects and plot them against the simulated (linearized) coefficients:

```
OLRE <-
  posterior(M_ovdsp) %>%
  filter(type == "ranef", re_factor == "Obs") %>%
  clu()

D_ovdsp %>%
  bind_cols(OLRE) %>%
  ggplot(aes(x = beta_0i, y = center)) +
  geom_point() +
  geom_smooth(method = "lm", se = F)
```



This shows, that even the observation-level deviations can be recovered quite well. For every single observation, the model can determine how much it has been pushed by unrecorded sources of variation. These OLRE levels can be seen as generalized residuals, or *linearized residuals*.

Linearized residuals can be used for different purposes, such as outlier detection or to compare sources of variation. Frequently, I reminded the reader to interpret parameters quantitatively by translating their magnitude to statements of practical relevance. For random effects variance this is not always straight forward, especially when we are on a linearized scale. One way is to make comparative statements on the sources of variance, like "the variance due to individual differences exceeds the measurement error". OLREs are on the same scale as all other random effects in the model, which makes it a suitable reference source of variation.

```
## [1] "sim_ovdsp" "D_ovdsp"
```

6.3 Duration measures

Time is a highly accessible measure, as clocks are all around us: on your wrist, in transport stations, in your computers and a very big (and accurate) one is hosted at the Physikalisch-Technischen Bundesanstalt in Braunschweig (Physical-technological federal institute in Braunschweig, Germany). Duration measures often carry useful information; especially, *Reaction time (RT)* measures are prime in experimental cognitive studies and have revealed fascinating phenomena of the human mind, such as the Stroop effect, memory

priming, motor learning and the structure of attention. In design research, reaction times are also frequently used in experiments, but more common is *time-on-task (ToT)* as a measure of task efficiency. Formally, both outcome types measure a period of time. I am deliberately making a distinction between the two, because the data generating process of reacting to a simple task (like naming a color) may be different to a complex task, like fidning information on a website.

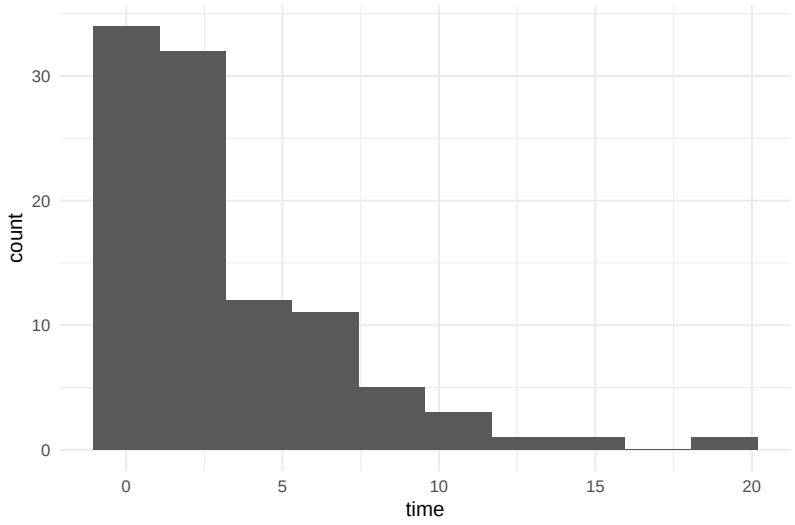
Temporal variables are practically continuous (as long as one measures with sufficient precision), but always have lower bounds. First, I will introduce two classic types of models that use exponentially or Gamma distributed error terms. Both distribution families assume a lower bound at zero, which is problematic, as we will see. Modern Bayesian estimation engines offer an increasing variety of more exotic response distributions. Among those are Exgaussian response distributions, which works well when the lower bound is positive.

6.3.1 Exponential and Gamma regression

Exponential distributions arise from basic random processes under some very idealized conditions. First, the lower boundary must be zero and second, the rate at which events happen is assumed to be constant rate, just like Poisson distributions assumes a constant λ.

Reconsider the subway smurfer example 3.4.2, where players collect items in a jump and run game. We have already seen how collection counts can be modelled using Poisson or binomial regression. Another way to look at it is the time between two events of item collection. For demonstration only, we assume such idealized conditions in the subway smurfer example and generate a data set. Exponential distributions are determined by one parameter, the *rate* parameter λ, which is strictly positive. The mean of an exponential distribution is the reciprocal $1/\lambda$ and the variance is $\text{Var} = 1/\lambda^2$. Like with Poisson regression, variance is strictly tied to the mean.

```
set.seed(20)
D_exp <-
  tibble(Obs = 1:100,
  time = rexp(100, rate = 1/3))

D_exp %>%
  ggplot(aes(x = time)) +
  geom_histogram(bins = 10)
```




```
mean(D_exp$time)
```

```
## [1] 3.29
```

```
var(D_exp$time)
```

```
## [1] 12.4
```

As the `stan_glm` engine does not support exponential response distributions, we use `brm`, instead, and recover the parameter.

```
M_exp <- brm(time ~ 1,
             family = "exponential",
             data = D_exp)
```

```
fixef(M_exp, mean.func = exp)
```

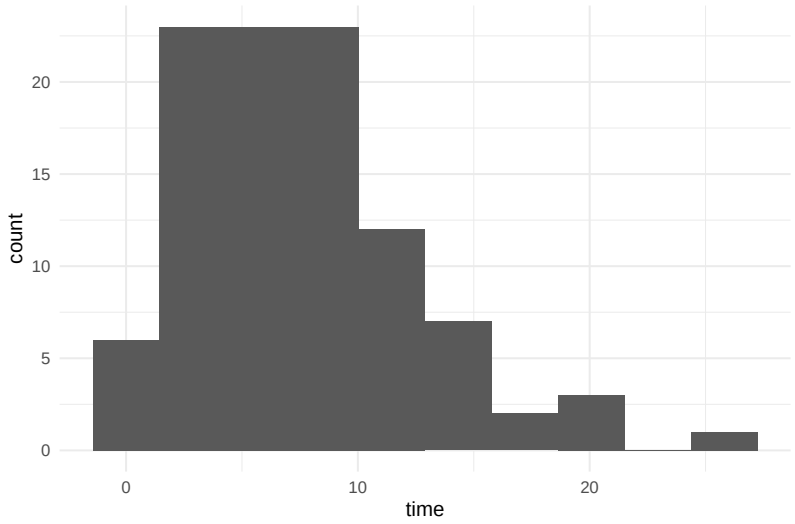
type	fixef	center	lower	upper
fixef	Intercept	3.29	2.73	4

Exponential disribution are rarely used in practice for two shortcomings: first, the strict mean-variance relation makes it prone to over-dispersion. This can be resolved by using observation-level random effects 6.2.3.3 or using Gamma

distributions, which accounts for extra variance by a second parameter. The second problem is the lower boundary of Zero, which will be resolved by using Exgaussian error distributions, instead.

Exponential regression has a single parameter and therefore has the same problem as seen with Poisson and binomial regression before. Only if all events have the same rate to occur, will an exponential distribution arise, which means for behavioural research: never. A general solution to the problem is introducing an observation-level random effect 6.2.3.3. Here we will tackle the problem by using continuous, zero-bounded distributions with two parameters, the Gamma family of distributions. While the two parameters rate and shape do not directly translate into location and dispersion as with Gaussian, it provides the extra degree of freedom to set them almost independently. The only limitation is that variance rises with the mean, but as we have argued in **??**, this is rather a desired feature than a problem. In the following, we simulate Gamma distributed observations.

```
set.seed(20)
D_gam <-
  tibble(Obs = 1:100,
  time = rgamma(100, rate = 1/3, shape = 2))

D_gam %>%
  ggplot(aes(x = time)) +
  geom_histogram(bins = 10)
```



```
mean(D_gam$time)
```

```
## [1] 7.61
```

In comparison to the exponential distribution above, a significant difference is that the mode of the gamma distribution (its peak) is not fixed at zero, but can move along the x-axis. That makes it appear a much more realistic choice for temporal data in behavioural research. We estimate a simple gamma GMM on the simulated data. For historical reasons, `brm` uses the inverse link function ($\theta = 1/\mu$) for Gamma regression per default, but that does not actually serve the purpose of link functions to stretch μ into the range of real numbers. Instead, we explicitly demand a log link, which makes this a multiplicative model.

```
M_gam <- brm(time ~ 1,
             family = Gamma(link = log),
             data = D_gam)

M_gam
```

```
fixef(M_gam, mean.func = exp)
```

type	fixef	center	lower	upper
fixef	Intercept	7.64	6.7	8.71

Both, Exponential and Gamma distributions support the range of real numbers including zero. The weak point of both models is that they have zero as their natural starting point. As we will see in the following section, this assumption is usually violated with RT and ToT data. So, what are they good for, after all? These two models are routinely used for the *time intervals (TI)* between events that are triggered independently. In nuclear physics the individual triggers are atoms, each one *deciding on their own* when to decay. If you measure the interval between two decays the time interval is exponentially distributed. (And if you count the neutrons per time interval, the result is a Poisson distribution).

Analog situations can be found in service design and logistics. Take the example of for customer support systems. Customers are like atoms in that their decision to file a request is usually independent from each other. Just by chance it can truly happen that two customers call the service center practically the same moment, so that the lower bound of Zero can actually be reached by some observations. Overwhelmed hotline queues do not make people happy,

if they have technical problems. When planning a support system, the risk of angry customers has to be weighed against the costs of over-staffing. A good design would hit a certain sweet spot and in the ideal case there would be a predictive model of inflow rate of customers.

6.3.2 ExGaussian regression

The problem with RT and ToT data is that Zero is not a possible outcome, as any task uses up a minimum time to complete. For example, the table below shows the minimum ToT for finding the academic calendar on ten university websites (case Egan). This varies a lot between designs, but is never even close to zero. The last column puts the minimum observed ToT in relation to the observed range. On two of the websites, the offset was even larger than the obvserved range itself, hence the problem of positive lower boundaries is real in user studies.

```
attach(Egan)
```

```
D_egan %>%
  filter(success,
         Task == "academic calendar") %>%
  group_by(Task, Design) %>%
  summarize(min_time = min(ToT),
            range = max(ToT) - min_time,
            min_time/range) %>%
  kable()
```

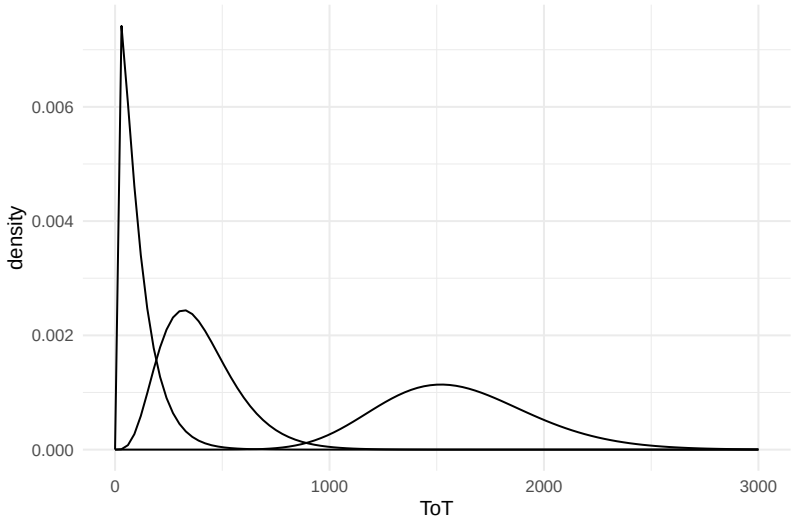
Task	Design	min_time	range	min_time/range
academic calendar	VU Brussel	21	130	0.162
academic calendar	KU Leuven	52	40	1.300
academic calendar	UGent	8	70	0.114
academic calendar	University of Antwerp	3	7	0.429
academic calendar	UHasselt	21	48	0.438
academic calendar	Leiden University	130	181	0.718
academic calendar	VU Amsterdam	207	188	1.101
academic calendar	RUG	119	132	0.902
academic calendar	University Tilburg	24	39	0.615

```
detach(Egan)
```

On the first glance, that does not seem to pose a major problem for Gamma distributions, as the left tail vanishes the more a Gamma distribution is shifted to the right, the impossible regions get smaller. However, Gamma distributions

inevitably become more symmetric at larger values. A Gamma distribution far to the right has almost equally long tails and we may eventually use a Gaussian distribution, instead. As there is no separate parameter controlling the skewness of the curve it may happen that the random component captures the amount of variance, but overdoes the left tail, which introduces a bias on the coefficients. The following graphic illustrates the mean-variance-skew relationship on three Gamma distributions that move from left to right (`M`), keeping the variance constant (`V`):

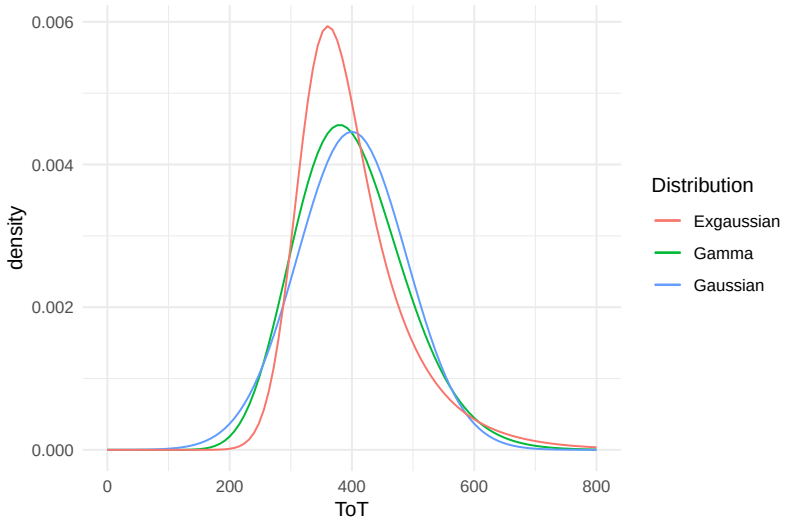
```
M = c(100,200, 400)
V = 8000

## gamma
rate = M/V
shape = rate^2 * V

ggplot(data.frame(x = c(0, 3000)), aes(x = x)) +
  stat_function(fun = dgamma,
                args = list(rate = rate[1], shape = shape[1])) +
  stat_function(fun = dgamma,
                args = list(rate = rate[1], shape = shape[2])) +
  stat_function(fun = dgamma,
                args = list(rate = rate[1], shape = shape[3])) +
  labs(x = "ToT", y = "density")
```



We have seen so far, that distributions with one parameter (Poisson, binomial, exponential) have a fixed relationship between location and dispersion.

In order to vary location and dispersion independently, a second parameter is needed (neg-binomial, beta-binomial, Gamma, Gaussian). Only three-parameter distributions can do the trick of setting skewness separately. So called *exponentially modified Gaussian* (Exgaussian) distributions are convolutions of a Gaussian distribution and exponential distribution and have three parameters, μ, σ and rate β. Very roughly, the Gaussian component controls location and dispersion whereas the exponential part adjusts the skew. When β is large in comparison to μ, the distribution is more left skewed. With this additional degree of freedom we can simulate (and estimate) distributions that are far to the right, have strong dispersion *and* strong skewness. The following plot repeats the right-furthest Gamma distribution from above and adds a Gaussian and Exgaussian distributions with all three having the exact same mean and variance.

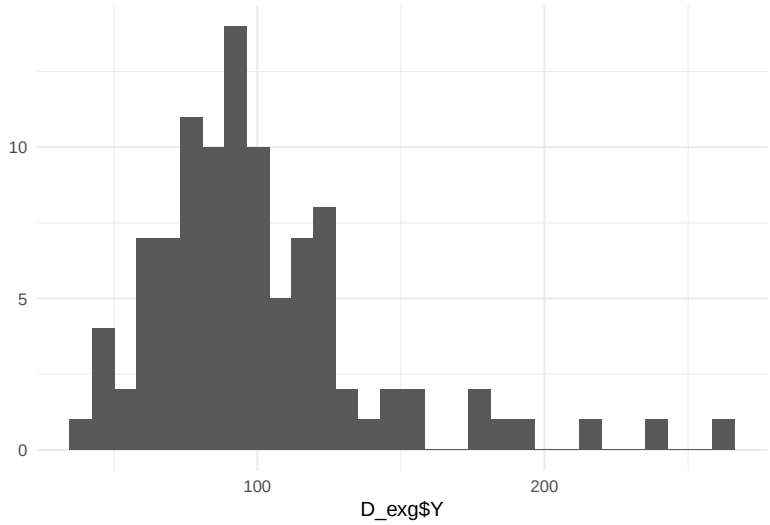
```
M = 400
V = 8000

## Exgaussian
mu = M
beta = 80
sigma = sqrt(V - beta^2)

## Gamma
rate = M/V
shape = rate^2 * V

ggplot(data.frame(x = c(0, 800)), aes(x = x)) +
  stat_function(fun = dgamma,
                args = list(rate = rate, shape = shape),
                mapping = aes(colour = "Gamma")) +
  stat_function(fun = dnorm,
                args = list(mean = M, sd = sqrt(V)),
                mapping = aes(colour = "Gaussian")) +
  stat_function(fun = brms::dexgaussian,
                args = list(mu = M,
                            sigma = sigma,
                            beta = beta),
                mapping = aes(colour = "Exgaussian")) +
  labs(colour="Distribution", x = "ToT", y = "density")
```



The Gamma distribution in this example starts approaching a the perfect bell curve of the Gaussian distribution. In contrast, the exgaussian distribution takes a steep left climb followed by a long right tail, which is caused by its pronounced exponential component. We do the usual exercise to simulate a grand mean model and recover the parameters with the help of the `brm` engine:

```
attach(Chapter_GLM)
```

```
D_exg <- tibble(Y = rexgaussian(100, mu = 100, sigma = 20, beta = 30))
qplot(D_exg$Y)
```




```
M_exg <- brm(Y ~ 1,
             family = exgaussian,
             data = D_exg)
```

```
fixef(M_exg)
```

type	fixef	center	lower	upper
fixef	Intercept	99.4	92.9	107

```
detach(Chapter_GLM)
```

Noteworthy, for Exgaussian models the brm engine uses the identity link function by default. While this is rather convenient for interpretation, it could theoretically lead to impossible predictions. As we will see later, the exgaussian is not immune, but *robust to impossible predictions* because of its tiny left tail. Any linear impact factor, like an experimental treatment can push it 150 ms to the left with insignificant risk of impossible predictions.

All GLM family members introduced so far are established, have been presented in many textbooks and are routinely used in research of all kinds. The Exgaussian is a newcomer. It does not come with a solid background in physical systems and may very well be considered just a hack. The following two sections examine Exgaussian models more closely and sets them in competition against Gamma and Gaussian error terms. We will be using primarily

graphical methods here, but will come back these cases in chapter 7.2.4 with a more formal approach.

6.3.2.1 Reaction times

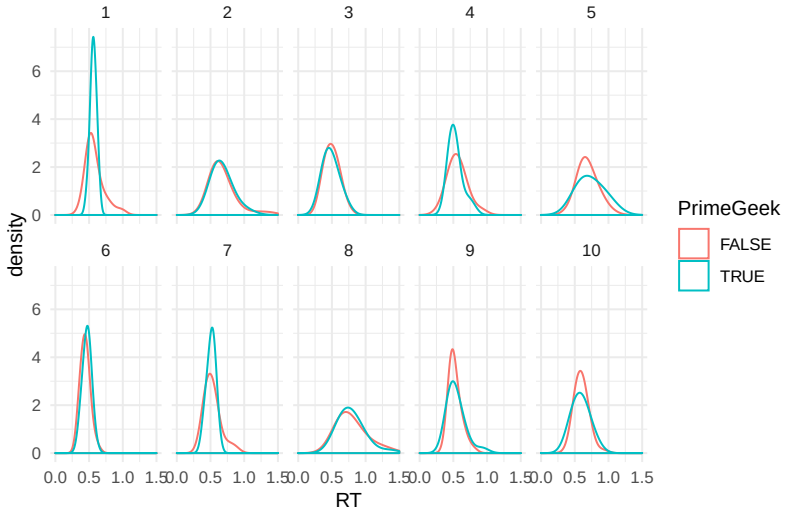
In experimental studies, the *inertia of the nervous system* sets a limit larger than zero for reaction times. This is partly due to to some hard electrochemical and biomechanical limits of the peripheral systems. (Octopuses have decentralized nervous system for a reason!) Nerve cells and muscle fibers are slow working horses. The same goes for our minds. Reportedly, they are blazingly fast at complex tasks, such as recognizing colors and written words. Still, there always is a minimum time necessary to collect an idea from the memories and activate the surrounding nodes. Therefore, experimental reaction times have a positive minimum and it is not much of a stretch to imagine that this is a sharp limit.

In the Hugme case, we tried to pin down the hypothetical Geek personality. We used Need-for-cognition as a predictor for reaction times from the semantic Stroop task. Like in the original Participants must name the color of words, but these are non-color words from two categories (geek/non-geek). These words are preceded by Geek/non-geek pictures. The theory goes that a real geek, when seeing an open computer case followed by "root" will briefly reminisce and be distracted from the primary task. It did not work this way at all and we only saw a few miniscule effects. That is good news for the analysis here. The only effect was that Geek primes caused a minimal delay. Because there are no other effects, we can use a rather simple multi-level CGM to compare how Gaussian, Gamma and Exgaussian fit reaction times. Let's take a first look some parts of the data:

```
attach(Hugme)
```

Most of the participant-level frequency distributions of RT have a clear cut-off at around .25 seconds. The steepness of the left climb varies between participants, but some at least are rather sharp, with a right tail that is leveling off slowly. When compared to the illustrations above, it seems that an Exgaussian model could accomodate this data well.

```
D_hugme %>%
  group_by(Part) %>%
  filter(Part <= 10) %>%
  ggplot(aes(x = RT, color = PrimeGeek)) +
  facet_wrap(~Part, nrow = 2) +
  geom_density(adjust = 2) +
  xlim(0, 1.5)
```



Even the efect of geek primes is barely visible, but we clearly observe a left skew in most of the participants. In the following, we run three CGM models with Exgaussian, Gamma or Gaussian response distributions. For the subsequent analysis, multi-model posterior distributions and posterior predictive distributions are extracted and merged into one multi-model posterior object `P_1`.

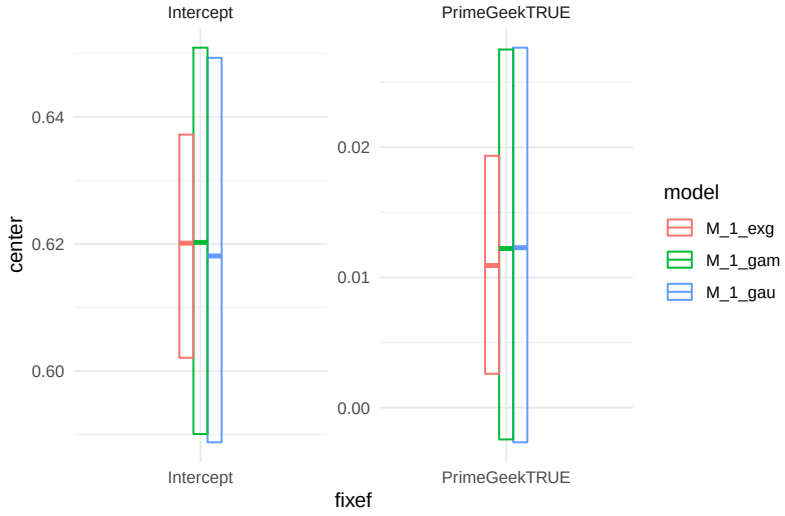
```
memory.limit(16000)

F_1 <- formula(RT ~ 1 + PrimeGeek + (1 + PrimeGeek|Part))

M_1_gau <- D_hugme %>%
  brm(F_1,
      family = gaussian,
      data = .)

M_1_gam <- D_hugme %>%
  brm(F_1,
      family = Gamma(link = identity),
      data = .)

M_1_exg <- D_hugme %>%
  brm(F_1,
      family = exgaussian,
      data = .)

P_1 <- bind_rows(
```

```
  posterior(M_1_gau),
  posterior(M_1_gam),
  posterior(M_1_exg)
)

T_1_predict <-
  bind_rows(
    post_pred(M_1_gau, thin = 5),
    post_pred(M_1_gam, thin = 5),
    post_pred(M_1_exg, thin = 5)
  ) %>%
  predict()
```

Note that the predictive posterior distributions runs over thousands of observation, which creates very large objects in your computer's RAM. To prevent running into a memory limit, we crank up the memory limit (`memory.limit`) and thin out the number of posterior predictive samples by factor 5.

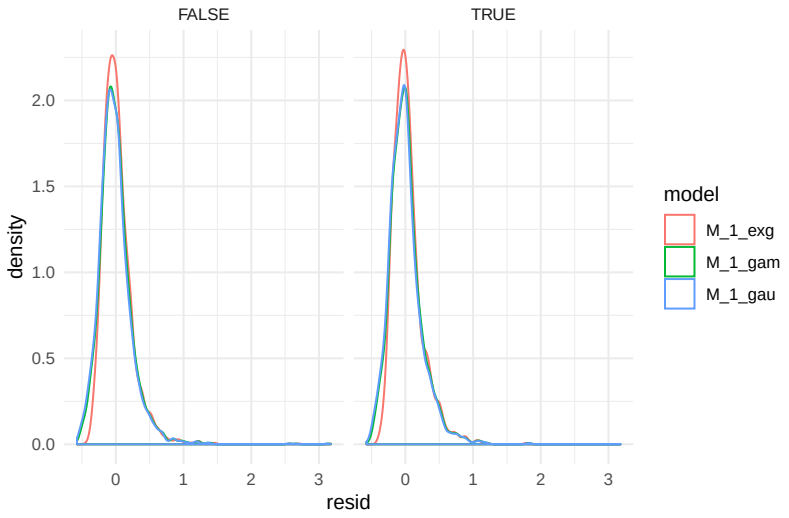
The below plot shows the population-level effects for the three models. The center estimates are very close, which means that neither of the models has a significant bias. However, the Exgaussian model produces much tighter credibility intervals. We have seen such an effect before, when on over-dispersed data, a Poisson model produced tighter intervals than the Negbinomial model. Here it is the other way round: the model with more parameters produces better levels of certainty.

```
fixef(P_1) %>%
  ggplot(aes(y = center, ymin = lower, ymax = upper,
             x = fixef,
             color = model)) +
  facet_wrap(~fixef, scales = "free") +
  geom_crossbar(width = .2, position = "dodge")
```



If the Exgaussian model has a better fit, we should primarily see that in how the residuals are shaped. The Exgaussian distribution has one more degree of freedom, which can be used to set an arbitrary skew. The following reveals that the extra flexibility of the Exgaussian has been employed. Both, Gaussian and Gamma are almost symmetric, whereas the Exgausian takes a steeper left climb. The three distributions have almost the same right tail, but the left tail of the Exgaussian is smaller and the extra probability mass has moved to the center.

```
D_hugme <- D_hugme %>%
  left_join(T_1_predict) %>%
  mutate(resid = RT - center)

D_hugme %>%
  ggplot(aes(x = resid, color = model)) +
  facet_wrap(~PrimeGeek) +
  geom_density() +
  labs()
```



We can carefully conclude that the Exgaussian may be very useful for analyzing pychological experiments as it seems to better accomodate reaction times. Given the novelty of Exgaussian models, it is recommended that researchers carry out a careful multi-model analysis. In 7.2.4 we will come back to this case with a more formal approach and confirm that from the three response distributions, the Exgaussian has the best predictive accuracy.

```
detach(Hugme)
```

6.3.2.2 Time-on-task

Experimental psychologists call the Stroop task a complex one. But, essentially it is a decision between three options and minimal processing time is rather short. Compared to tasks in usability studies, such as finding information on websites or renting cars online, this is almost nothing. Also, the complex dynamics of the task are rather different. For example, a single user error at the begin of a task sequence can have dramatic consequences, such as getting lost on a website. While ToT data also has a strictly positive lower boundary (the fastest way of achieving the goal), it often has a much wider spread and more pronounced than in RT data. In the following we will repeat the informal model comparison from the last section for ToT data.

We compare the three patterns of randomness on the CUE8 data set, which contains ToT measures on five tasks on a car rental website. In this study 14 professional teams took part with two conditions: remote and moderated sessions. As data from the remote condition is contaminated with cheaters, we only use the moderated sessions. In order to compare the impact of the

chosen distribution on the coefficient estimates, we include the factor Task as a fixed effect (with treatment contrasts), despite this not being the most meaningful.

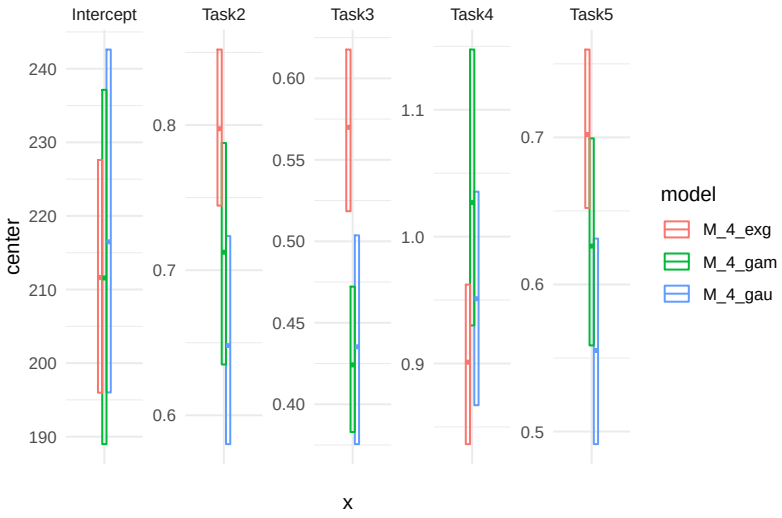
```
attach(CUE8)
```

```
D_cue8_mod <- D_cue8 %>%
  filter(Condition == "moderated", !is.na(ToT)) %>%
  as_tbl_obs()
```

```
F_4 <- formula(ToT ~ 1 + Task + (1|Part))

M_4_gau <- D_cue8_mod %>%
  brm(F_4,
      family = gaussian(link = log),
      data = ., iter = 2000)

M_4_exg <- D_cue8_mod %>%
  brm(F_4,
      family = exgaussian(link = log),
      data = ., iter = 2000)

M_4_gam <- D_cue8_mod %>%
  brm(F_4,
      family = Gamma(link = log),
      data = ., iter = 2000)

P_4 <- bind_rows(
  posterior(M_4_gau),
  posterior(M_4_gam) %>% mutate(value = if_else(value %in% c("fixef", "ranef"), exp(value),
  posterior(M_4_exg)
)


T_4_predict <- bind_rows(
  post_pred(M_4_gau, thin = 5),
  post_pred(M_4_gam, thin = 5),
  post_pred(M_4_exg, thin = 5)) %>%
  predict()
```

Note that the Gamma model caused trouble when estimated with an identity link. For this reason, all three models were estimated on a log-linearized scale. This makes the back-transformed coefficients multiplicative, which actually makes more sense, as we have seen in 6.2.1.1.

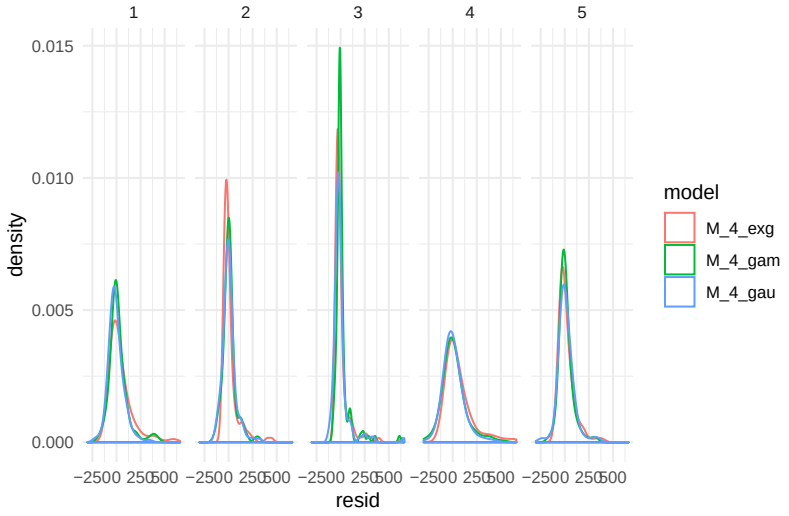
```
fixef(P_4, mean.func = exp) %>%
  ggplot(aes(y = center, ymin = lower, ymax = upper,
             x = " ",
             color = model)) +
  facet_wrap(~fixef, scales = "free", nrow = 1) +
  geom_crossbar(width = .2, position = "dodge")
```



In the pr4evious section, we have seen, that the three models agreed on the center estimates. For the ToT data here, three models produce rather different coefficients. It seems that the models disgree on all but the Intercept.

Inspecting the residual distributions yields a similar patter, once again: there is visible left skew, and the Exgaussian model has the sharpest left rise.

```
left_join(T_4_predict, D_cue8_mod, by = "Obs") %>%
  mutate(resid = ToT - center) %>%
  ggplot(aes(x = resid, color = model)) +
  facet_wrap(~Task, nrow = 1) +
  geom_density(adjust = 2)
```




```
detach(CUE8)
```

In general, it seems that Exgaussian models for RT and ToT accomodates left skewness better and produces estimates that are more conservative and certain at the same time. But, These are just informal comparisons. In chapter 7, we will apply formal criteria for selecting between distributions. As will turn out, Gamma distribution is the preferred distribution for ToT in CUE8.

One last issue remains to get clarified: using the identity link for Exgaussian models is very convenient and is probably much safer as compared to Gaussian models with their longer left tails. Still, impossible predictions can arise. But, how much of a risk is there? We check this on the posterior predictive distributions of both studies, CUE8 and Hugme. The following table shows the proportion of observations, that get a negative 2.5% credibility limit assigned.

```
bind_rows(Hugme$T_1_predict, CUE8$T_4_predict) %>%
  group_by(model) %>%
  summarize(mean(lower < 0)) %>%
  kable()
```

model	mean(lower < 0)
M_1_exg	0.000
M_1_gam	0.000
M_1_gau	0.000
M_4_exg	0.015
M_4_gam	0.000
M_4_gau	0.712

For our RT data, impossible predictions is not a big issue with any of the models, as all 2.5% quantiles are positive. That is different for ToT: while the gamma model is inherently immune to negative predictions, the exgaussian model produced a few impossible lower 2.5% limits (around 3%). The Gaussian model is extremely off: more than 70% of all predictions have impossible lower 2.5% limits.

In the scientific literature, the coverage on what random pattern to use for RT and ToT data is meager at this moment. Probably, that is due to the lack of user-friendly engines supporting the more exotic GLM family members, gamma or exgaussian regression. The brms engine covers a much broader set of distributions than any other implementation before and researchers have the choice. This chapter attempted to provide theoretical arguments as well as empirical indications that the exgaussian regression can be a better choice than Gaussian and gamma. First of all, it accomodates the strong left skew of RT and ToT much better than the gamma, which takes a too symmetric form when far from the left boundary. Second, it is reasonably robust to impossible predictions, even when using the convenient identity link function. Third, and that is almost too good to be true, it massively improves certainty in predictors. It seems that Exgaussian models are more efficient for carving out delicate effects in experimental studies.

However, as the discussion has not even fully started, to declare it settled would be premature. In contrast, the aim of this section was to illustrate a semi-formal approach that reseachers can follow to choose among the candidate models for their specific RT and ToT data. Data from other RT paradigms might take different shapes. For example, when measuring RT by events in EEG signals (rather than actual key presses), motor time plays a much smaller role, pushing RTs closer to the left boundary. Then, the exgaussian model might produce higher rates of impossible predictions and the gamma model could sufficiently accomodate the left skewness. Note that even using a log link on the exgaussian model can produce visits to the negative range. When both accomodate the left skew equally well, the gamma model is to be preferred as it never produces impossible predictions and is more parsimomous.

6.4 Rating scales

In classic design research of the last millenium, die-hard Human Factors researchers have mainly been asking objectively sounding questions, like:

- Can a user achieve accurate results with the system?
- Can they do so in less time?
- Is the number of errors reasonably confined?

For professional, and especially critical, systems, these are highly valid questions, indeed. The purpose of a medical infusion pump is to improve the health status of a patient by accurately and reliably delivering medication into the bloodstream and the extent to which this can be measured dircetly.

However, starting with the 1990s, wave after wave of novel electronic entertainment systems and digital gadgets rolled over the comsumer mass market. The purpose of a video recorder or a smartphone is to deliver joy ... and to be sold in large batches. The sole purpose of a commercial website is to sell and nothing else. With the new millenium, design researchers began to recognize what consumer psychologists had discovered two decades earlier: users are not rational decision makers in a utilitarian sense. When people decide to adopt (or buy) a new system, this is only partly driven by their expectation of productivity. These additional expectations are commonly called *hedonistic values* and cover a broad class of human needs, such as:

- positive emotions
- expression of self
- social connectedness
- aesthetic perception
- personal growth

Whether or not these concepts are well-defined from a psychological point-of-view is beyond the scope of this book. What matters is that these concepts are so elusive that the most sincere researchers have not yet found objective criteria to measure them. Instead, almost everyone resorts to use of *self-report rating scales*, like this one:

How beautiful do you perceive the user interface to be?

unattractive 1 – 2 – 3 – 4 – 5 a piece of art

If you use a 5 point rating scale like this one to measure perceived beauty, participants have to convert their gut feeling into a number, which involves the following three processes somewhere in their minds:

1. anchoring
2. introspection
3. binning

By *anchoring* participants establish an idea of how ugly or beautiful something has be to get an extreme rating of 1 or 5. These imaginary endpoints define the *absolute range* of the rating scale. The researcher might early on give explicit or implicit cues to let the participant guess the range the researcher has in mind. If an experiment is overtly about web design, then probably "very ugly" means the least attractive commercial website the participant can think of. However,

participants old enough to remember web design in its infancy (say the early attempts of disney.com), may end up with a lower anchor than today's kids. If too few cues are given upfront, participants will probably adjust their anchors to the stimuli they see throughout the experiment. Probably, it will make a difference for what 1 or 5 mean, when the set of stimuli contain just websites, or websites *and* impressionist paintings *and* screenshots from 1980 splatter movies.

By *introspection* participants intuitively assess the intensity of their *real feelings* as compared to the anchors. Reportedly, feelings are influenced by:

1. visual simplicity
2. prototypicality
3. second exposure
4. Gestalt principles
5. fluency of processing
6. attribute substitution heuristics
7. color aesthetics
8. fashion
9. previous stimuli
10. current mood
11. a person's history
12. and cultural background

By *binning* the participant mentally divides the absolute range into five categories that are either fuzzy or defined by stereotypes, like "It must look at least as elegant as a certain other website to get a 4."

As the outcome of anchoring, introspection and binning are not under the control of the researcher, the response patterns can vary between participants. Let's consider a few possible patterns of participants (and their dramatic stories):

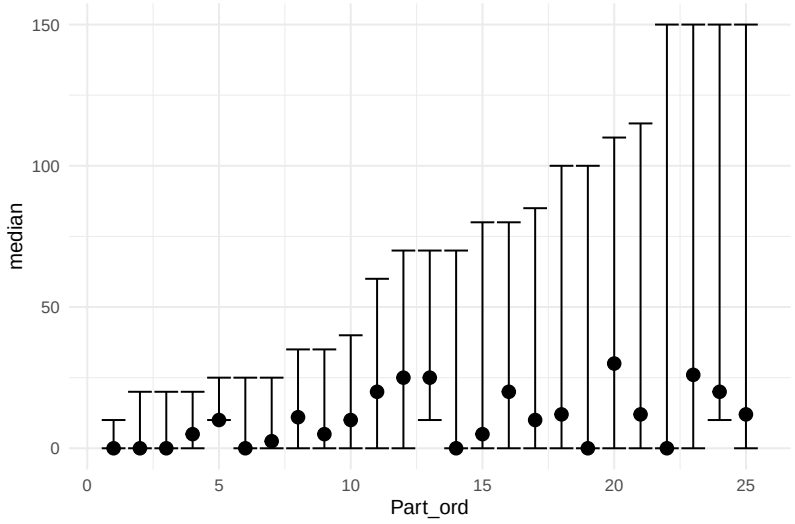
1. A is undecisive and stays in the center region
2. B has a crush on the experimenter and responds slightly more positive
3. C politely avoids the negative extremes
4. D is a human rights activist and habitually treats the seven bins equally.
5. E is annoyed by the experiment (rightly so) and falls into a dichotomous response pattern: 1 or 5
6. F is a surrealist and has a completely unique way to "look at things".

Many unkowns are in the game. Only one special case we can alleviate with the multilevel modelling tools in our hands. Anchoring can (but not necessarily does) result in a constant *shift* between participants. Compare participants A and B: A collects almost all stimuli in categories 2, 3 and 4, whereas B uses

3, 4 and 5. This is not much else than a participant-level intercept random effect. Unfortunately, the situation can be more difficult than that. When participants differ in how extreme they set their endpoints, like C and D, than their responses will differ in variance. The maximum variance, however, will be found in participant E.

To speak of a real case: In the IPump study, a single-item rating scale was used to measure mental workload. The results suggest that all participants used the lower range of the scale, but differed vastly in where they set their upper point. Figure XY orders participants by the maximum value they used. This is obviously related to variance, but seemingly not so much with location. It does not suffice to use a response distribution with mean-variance relationship, as we used to. All these issues make rating scales peculiar and we should not pretend they have the same neat arithmetic properties as objective measures.

```
attach(IPump)

D_pumps %>%
  group_by(Part) %>%
  summarize(min = min(workload),
            max = max(workload),
            median = median(workload)) %>%
  mutate(Part_ord = rank(max, ties.method = "first")) %>%
  ggplot(aes(x = Part_ord, ymax = max, ymin = min, y = median)) +
  geom_errorbar() +
  geom_point(size = 3)
```



```
detach(IPump)
```

Setting the idiosyncratic rating scale responses aside, how does a common rating scale appear in our framework of link functions and patterns of randomness? Rating scales are bounded on two sides and we already know what that means: a suitable model for rating scales will likely contain a logit link function and a distribution of randomness that is bounded on two sides.

A real problem with rating scales is that they often are discrete. Most scales force participants to give their answer as a choice between five or seven ordered levels. When the response variable has just a few levels, *ordinal regression* is a good choice. Ordinal regression itself is a generalization of logistic regression.

However, with most properly designed self-report instruments, a scale comprises several items, because only that can sufficiently reduce measurement error and allow for in-depth psychometric assessments. Take the well-known Cronbach α, which assesses reliability of a scale by correlating every items score with the sum score. Obviously, that only makes sense when there are multiple items. While from a psychometric perspective, single-item scales are susceptible, there can be situations where a researcher may use a validated single item scale for pragmatic reasons. Especially, when measures happen in situ, such as during a usability test or even a real operation, being brief and unobtrusive might be more important than good quality of measures.

With multi-item rating scales, one also has the possibility to build a psychometric multi-level model, where items are considered a sample of a population of possible items. That is actually a very good idea, as the item-level random effects control for differences in item location. For example, the following item is likely to produce generally lower beauty ratings than the one shown earlier, because the anchors have been moved downwards:

How beautiful do you perceive the user interface?

like a screenshot from a splatter movie 1 – 2 – 3 – 4 – 5 quite attractive

Unless one builds such a psychometric multi-level model, ordinal regression is not very suitable for multi-item scales and here is why: The sum (or mean) score is still binned, but more finely grained. A sum score over three seven-binned items already has 21 bins, which would result in an inflation of number of parameters in ordinal regression.

As a rescue, one might well regard a measure with 21 bins as continuous . Furthermore, there actually is no strong reason to use binned rating scales at all. So called *visual analog scales* let participants make continuous choices by either drawing a cross on a line or move a slider control. For sum scores and visual analogue scales, the problem of choice reduces to a logit link function (they still have two boundaries) and a continuous distribution bounded on both sides. That is precisely what is behind *beta regression* and, as we shall

see, this distribution is flexible enough to smooth over several of the rating scale pathologies that were just discussed.

6.4.1 Ordered logistic regression

When the ordinal response has a low number of response categories (between 4 and 7), ordinal regression applies. Recall logistic regression: the response falls into one of two categories, which are coded as 0 and 1. Although not in a strict sense, the two categories can often be thought of as in an order: success is better than failure, presence more than absence and a return better than staying away. Instead of two categories, we can also conceive the situation as a *threshold* between the categories, that needs force to jump over it. Any positive impact factor x_i can then be thought of as such a force that pushes a response probability to the higher category, by the amount β_i (on logit scale). At the same time, the intercept β_0 represents the basic log-odds of falling into category 1 in a default state, that is $x_i = 0$.

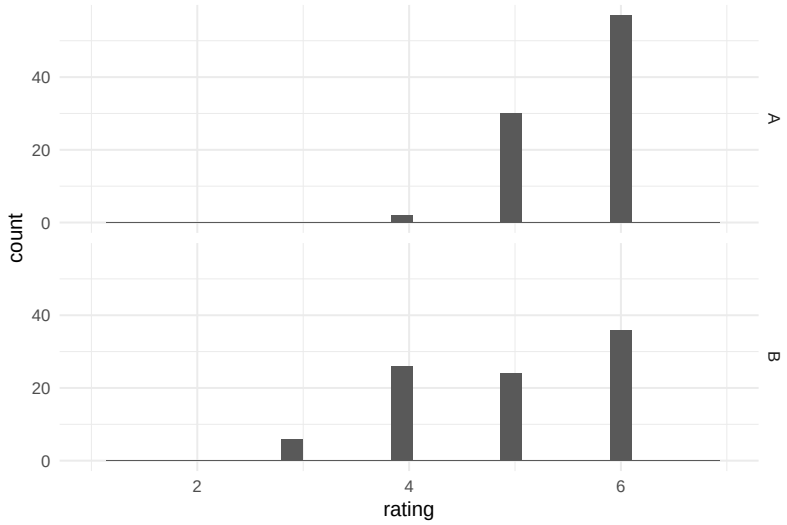
In ordinal regression, this idea extends to cases with more than two ordered response categories. The only arising complication is that with two categories, we have one *threshold* to overcome, whereas with three categories there are two thresholds and generally, with c categories, there are $c-1$ thresholds. Ordinal regression deals with the problem by estimating $c-1$ intercept estimates $\beta_{0[k]}$. Each threshold intercept $\beta_{0[k]}$ represents the probability (on logit scale) that the response falls into category k *or lower*, or formally:

$$\text{logit}(P(y_i \leq k)) = \beta_{0[k]}$$

Let's see this at the example of the BrowsingAB case, first. User ratings have been simulated with seven levels:

```
attach(BrowsingAB)
```

```
BAB1 %>%
  ggplot(aes(x = rating)) +
  facet_grid(Design~.) +
  geom_histogram() +
  xlim(1,7)
```



The brms regression engine implements ordinal regression by the family `cratio` (cumulative odds ratio) with a default logit link function.

```
M_ord_1 <-
  BAB1 %>%
  brm(rating ~ Design,
      family = "cratio",
      data = .)
```

```
fixef(M_ord_1)
```

fixef	center	lower	upper
NA	-13.411	-53.335	-5.928
NA	-13.326	-53.333	-5.896
NA	-4.053	-5.020	-3.251
NA	-2.281	-2.778	-1.796
NA	-1.121	-1.525	-0.734
NA	1.321	0.846	1.851
DesignB	-0.856	-1.317	-0.410

The six intercepts correspond with the thresholds between the seven levels. It is no coincidence that the intercept estimates increase by order, as they are cumulative (the "c" in cratio). The first intercept estimate represents (the logit of) the proportion of responses $y_i \leq 1$, the second $y_i \leq 2$ etc. The Design effect has the usual interpretation as compared to logistic regression,

an increase in logit. The only difference is that it refers to all six reference points. The expected proportion of responses equal to or smaller than 2 for design A is:

$$\begin{aligned}\pi(y_i \leq 2|A) &= \text{logit}^{-1}(\beta_{0[2]}) \\ &= \text{logit}^{-1}(-13.3) = 1.674 \times 10^{-6}\end{aligned}$$

The expected proportion of responses equal to or smaller than 2 for design B we get by the usual linear combination:

$$\begin{aligned}\pi(y_i \leq 2|B) &= \text{logit}^{-1}(\beta_{0[2]} + \beta_1) \\ &= \text{logit}^{-1}(-14.2) = 6.808 \times 10^{-7}\end{aligned}$$

All coefficients are shifting all thresholds by the same amount (on the linear predictor scale). You can picture this as a single puppetier controlling multiple puppets by just one stick, making them dance synchronously. As long as the ordinal scale has only a low number of bins, that keeps the number of parameters at a reasonable level. Just imagine, you were estimating an ordinal multilevel model and all participant-level effects were five or seven-folded, too. However, the equidistancy of effects on bin thresholds is an assumption by itself, and in the presence of response styles on rating scales, it cannot be taken for granted .

Besides that, the ordinal model appears very snug to the structure of the data. It does not wipe over the fact that the response is discrete and the thresholds represent the order. Conveniently, effects are represented by a single estimate, which one can use to communicate direction and certainty of effects. On the downside, communicating absolute performance (that is, including the intercept) is more complicated. When presenting predictions from an ordinal model one actually has to present all thresholds, rather than a single mean. In practice that probably is less relevant than one might think at first, because predictions on self-report scales is less useful than metric performance data. Ordinal data also does not lend itself so much to further calculations. For example, you can use ToT measures on infusion pumps in calculating the required staffing of an intensive care unit, because seconds are metric and can be summed and divided. In contrast, it does not make sense to calculate the cognitive workload of a team of nurses by summing their self-report scores. The only possibility is to compare the strengths of predictors, but that does not require predictions.

```
detach(BrowsingAB)
```

6.4.2 Beta regression

One of the most futile discussions in methodology research to my mind is whether one should use a four, five or seven binned Likert scale. From a pure measurement point of view, more bins give better resolution, the ultimate consequence being not to bin at all, that is using continuous rating scales. At the same time, many rating responses come from multiple item scales, which multiplies the number of bins. Speaking of ordinal regression, it seems reasonable to have seven intercepts for a single item scale, but who would want 14 or 21 for a two or three-item scale? And most scales have more items than that, which is good from a psychometric perspective.

In fact, psychometric research in the process of developing rating scales routinely uses a method called *confirmatory factor analysis*, which derives from the Gaussian linear model and inherits its assumptions. Not surprisingly, most research applying the very same instruments also use plain linear models. It seems fair enough to take a multi-item scale as a continuous measure, but given the framework of GLM, it is unnecessary (to put it mildly) to go along with the assumptions of Normality and linearity. While the link function for a double bounded response variable is simply the logit, the only missing ingredient is a double bounded error distribution. Enter beta distribution!

We demonstrate beta regression on rating scales at the example of the CUE8 study. This study aimed at assessing whether remote usability testing arrives at the same ToT measures as in moderated sessions. As we have seen in 5.6, the difference is marginal. But, rating scales are susceptible for all kinds of cognitive and social biases. For that reason, a golden rule for user test moderators is to constantly remind participants to not blame themselves for errors. Reportedly, test moderators also do help participants (after counting to 10) in order to minimize frustration (and maximize information flow). What could the presence or absence of a moderator do to satisfaction ratings? Perhaps, remote participants feel the lack of assurance and support as higher levels of frustration. Furthermore, it is not unlikely that satisfaction ratings are sensitive to idiosyncratics in the process and setting of the user test, such that we could even expect differences between teams.

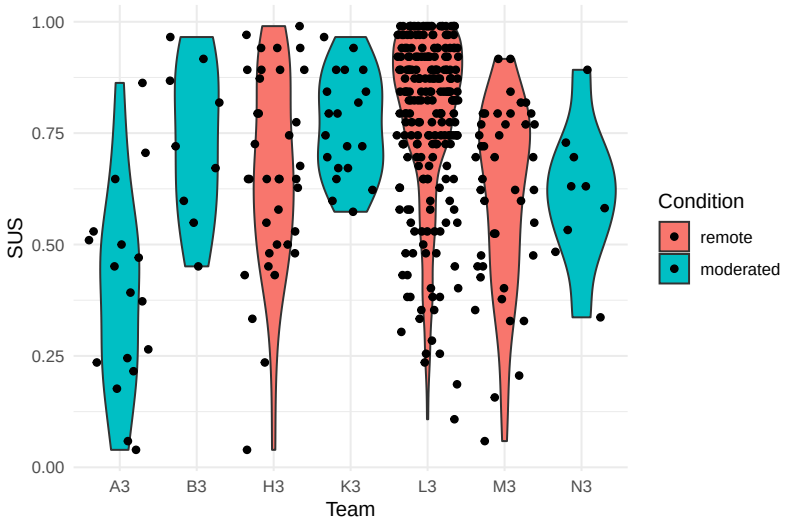
Before we build the model, there are two issues to regard: First, the boundaries are 0 and 1, which requires a rescaling of responses into this interval. The SUS scores are on a scale from 0 to 100 and a divisor of 100 would produce the desired interval. Second, the responses must actually lie *strictly between 0 and 1*, excluding the boundaries. On (quasi)continuous scales, it seems not very likely to have 0 or 1 as response, but it can happen. Indeed, participants in the CUE8 study have responded with a satisfaction rating of 100 quite often.

A practical solution is to scale the responses in such a way as to avoid the two boundaries, which is what the following hack does.

1. add a tiny value to all responses
2. create a divisor by adding twice that value to the maximum value the responses can take
3. divide all responses by that divisor

You may find it inappropriate to mangle a response variable in such an arbitrary way. However, keep in mind that the levels of ordinal responses are highly arbitrary. In terms of measurement theory, all transformations that maintain the order are permitted for ordinal scales. For the following analysis, the data set was further reduced by averaging the scores across tasks and excluding probable cheaters with a ToT < 30s.

```
attach(CUE8)

D_cue8_SUS <-
  D_cue8 %>%
  filter(!is.na(SUS)) %>%
  group_by(Part, Team, Condition) %>%
  dplyr::summarize(ToT = sum(ToT),
            SUS = mean(SUS)) %>%
  ungroup() %>%
  filter(ToT > 30) %>%
  mutate(SUS = (SUS + 1)/(100 + 2)) %>% ## rescaling to ]0;1[
  as_tbl_obs()

D_cue8_SUS %>%
  ggplot(aes(x = Team, y = SUS, fill = Condition)) +
  geom_violin() +
  geom_jitter()
```




```
M_5_bet <-
  D_cue8_SUS %>%
  brm(SUS ~ Condition + (1 | Team),
      family = Beta(link = "logit"), iter = 0, chains = 1,
      data = .)

M_5_bet <-
  D_cue8_SUS %>%
  brm(fit = M_5_bet, data = .)

sync_CE(M_5_bet, Env = CUE8)
```

```
M_5_bet
```

```
##  Family: beta
##   Links: mu = logit; phi = identity
## Formula: SUS ~ Condition + (1 | Team)
##    Data: . (Number of observations: 363)
## Samples: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
##          total post-warmup samples = 4000
##
## Group-Level Effects:
## ~Team (Number of levels: 7)
##               Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sd(Intercept)     0.81      0.39     0.34     1.95 1.01      554      231
##
## Population-Level Effects:
```

```
##                    Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## Intercept              0.79      0.52    -0.51     1.78 1.00      567      234
## Conditionmoderated    -0.34      0.70    -1.70     1.11 1.00     1124     1088
##
## Family Specific Parameters:
##     Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## phi     4.13      0.30     3.56     4.74 1.00     3390     2547
##
## Samples were drawn using sampling(NUTS). For each parameter, Bulk_ESS
## and Tail_ESS are effective sample size measures, and Rhat is the potential
## scale reduction factor on split chains (at convergence, Rhat = 1).
```

```
fixef(M_5_bet)
```

fixef	center	lower	upper
Intercept	0.798	-0.506	1.78
Conditionmoderated	-0.343	-1.699	1.11

```
grpef(M_5_bet)
```

type	fixef	re_factor	center	lower	upper
grpef	Intercept	Team	0.71	0.344	1.95

Do participants in remote sessions feel less satisfied? There seems to be a slight disadvantage, but we cannot confirm this with sufficient certainty. In contrast, the variation between teams is substantial, which indicates that SUS ratings are not independent of the particular setting. That is rather concerning for a widely used and allegedly validated rating scale.

6.5 Beyond mean: distributional models

The framework of GLM, as flexible as it has proven to be up to this point, has one major limitation: it renders the relationship between predictors and the location of the response. We only think in terms of impact factors that improve (or damage) average response times, error rates, satisfaction ratings etc. As we have seen multiple times in chapter 5, variance matters, too. With multi-level models we can estimate variance within a sample and even compare variance across samples, like in the CUE8 case, where more variance is due

to teams rather than due to participants. What cannot be done with plain multi-level models is estimate effects on variance, like: "Do teams in CUE8 differ in the variation of responses?". That brings me to the final feature of modern regression modelling in the scope of this book. With *distributional models*, we can put predictors on any distribution parameter that we want, not just location μ.

All but the one-parameter distributions come with additional parameters that allow to accomodate the dispersion of the error distribution, or even its skew in the case of Exgaussian models. In teh following I will present two application scenarios for distributional models. We start with a designometric problem when using bi-polar rating scales (see 6.4.) In the next section I will illustrate the problem on simulated data from two items with different anchoring. Subsequently, we will see on a real data set, that participants differ in how they exploit the range of a rating scale. by a simple simulation will show what effect item anchoring can have and how a distributional model can deal with such a situation.

6.5.1 Item-level anchoring in rating scales

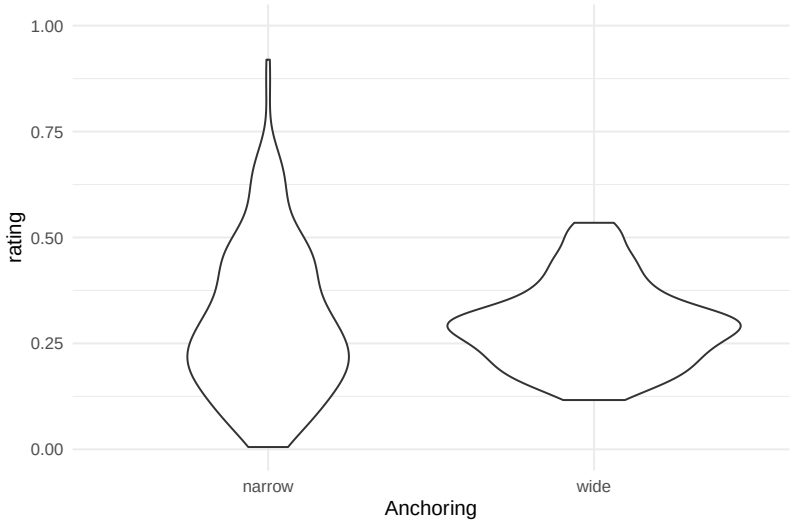
As a first illustration, imagine two versions of a continuous rating scale for visual beauty that differ in how their extreme levels are labelled:

1. like the ugliest website I have ever seen: 0 ——— 1 like the most beautiful website
2. distgusting as a screenshot from a splatter movie: 0 ——— 1 like the most beautiful sunset

For the sake of simplicity (not for a strong research design), let us assume that one participant has rated a sample of 400 websites in two conditions: narrow anchoring and wide anchoring. The following simulates data such that both anchorings have the same location (`mu`), but the narrow anchoring condition produces a much more exploitation of the range. Note that parameter `phi` does not increase variance, but just the opposite: the range of the scale is expanded, which lets the variance shrink. It is comparable to the number of trials in a binomial process. The more trials there are, the relatively tighter the distribution becomes.

```
set.seed(42)

N = 400

Conditions <-
  tribble(~Anchoring, ~mu, ~phi,
```

```
              "wide", .3, 18,
              "narrow", .3, 6) %>%
  mutate(a = mu * phi,
         b = phi - mu * phi)

D_Anchor <-
  tibble(Obs = 1:N,
              Anchoring = rep(c("wide", "narrow"), N/2)) %>%
  left_join(Conditions) %>%
  mutate(rating = rbeta(N, a, b))

D_Anchor %>%
  ggplot(aes(x = Anchoring, y = rating)) +
  geom_violin() +
  ylim(0,1)
```



Note that the `rbeta` command uses a different parametrization of beta distribution, with parameters `a` and `b`, which have are linked to the distribution mean and variance in rather convoluted ways.

The two response distributions have the same location, but the more narrow anchoring produces a wider dispersion of responses. How would we confirm this statistically? The Brms engine can link predictors to *any* other parameter of the response distribution, which teh author of the package calls *distributional models*. They have an immense potential as they relax another assumption of GLM, namely that all variance parameters must strictly follow the mean-variance relationship demanded by a distribution family. As we have seen, one can easily create a case where this assumption is violated.

For Beta distributions, a large ϕ results in reduced dispersion and vive versa. Accordingly, when used in a distributional model, a positive effect decreases variance.

When estimating dispersion or scale parameters, we have to regard that these are positive, strictly. The Brms engine simply extends the principle of link functions to parameters other than the μ and sets a default log link for ϕ. In order to estimate the changes in μ and ϕ simultaneously, the brms engine receives two regression formulas. Having multiple formulas for a regresson model is a notable extension of the R model specification language, which is why Brms brings its own command `bf()` to collect these. We run a distributional Beta regression on the simulated rating scale responses:

```
M_beta <- brm(bf(rating ~ 0 + Anchoring,
                 phi ~ 0 + Anchoring),
              family = Beta(),
              data = D_Anchor)
```

The parameter table below contains the two regular coefficients on location and, as expected, there is little difference in location. The intercept on scale parameter ϕ is the scale in the narrow condition (with wider variance). The treatment effect on ϕ is positive on the log scale, which means it is deflates variance, just as expected.

```
posterior(M_beta) %>%
  select(parameter, value) %>%
  mutate(value = if_else(str_detect(parameter, "phi"),
                         exp(value),
                         inv_logit(value))) %>%
  group_by(parameter) %>%
  summarize(center = median(value),
            lower = quantile(value, .025),
            upper = quantile(value, .975))
```

parameter	center	lower	upper
b_Anchoringwide	0.371	0.313	0.432
b_Intercept	0.421	0.367	0.476
b_phi_Anchoringwide	4.501	2.632	7.562
b_phi_Intercept	5.103	3.481	7.288

Note that as of writing this, the Bayr package has not yet evolved to handle distributional models. One issue is that the two parameters are on different scales ()

With such a distributional model we can discover differences in anchoring. And, even better, we can account for it. It intuitively makes sense to mix

items with extreme and modest anchoring. By merging a distributional model with a designometric multi-level model 5.8.4, we can evaluate and use such heteogeneous scales. The general concept is shown in the next section, where we account for participant-level employment of scale.

6.5.2 Participant-level employment of scale

In the previous section, we have seen how to discover differences in anchoring of items. What is seen frequently in designometric studies is different response patterns in participants, in particular how much they tend to use the extremes. We can account for that by letting the variance parameter vary across participants. The last model in this chapter is putting it all together:

1. a structural part for the mean with a fixed-effects polynomial term and a participant-level random effect for the response curve.
2. a structural part for scale parameter `rho` with a participant-level random effect on the scale parameter `rho`
3. a Beta shape of randomness

```
attach(Uncanny)
```

```
RK_1 <-
  RK_1 %>%
  mutate(response_unit = mascutils::rescale_centered(response, scale = .99) + 1)

M_poly_3_beta  <-
  brm(formula = response_unit ~ 1 + huMech1 + huMech2 + huMech3 +
        (1 + huMech1 + huMech2 + huMech3 | Part),
      family = Beta(),
      data = RK_1,
      inits = 0)

M_poly_3_beta_dist  <-
  brm(formula = bf(response_unit ~ 1 + huMech1 + huMech2 + huMech3 +
        (1 + huMech1 + huMech2 + huMech3 | Part),
                   phi ~ 1 + (1|Part)),
      family = Beta(),
      data = RK_1,
      inits = 0)
```

Note that on such more exotic and complex models, Brm sometimes has difficulties in finding valid good starting values for the MCMC walk. Like here, it often helps to fix all starting values to Zero.

If participants show different employment of scale, we should see that on the participant-level standard deviation of `rho`.

```
bind_rows(
  posterior(M_poly_3_beta),
  posterior(M_poly_3_beta_dist)
) %>%
  filter(type %in% c("grpef", "fixef")) %>%
  group_by(model, parameter) %>%
  summarize(center = median(value),
            lower = quantile(value, .025),
            upper = quantile(value, .975)) %>%
  ungroup()
```

model	parameter	center	lower	upper
M_poly_3_beta	b_huMech1	4.945	3.320	6.613
M_poly_3_beta	b_huMech2	-14.596	-18.284	-11.072
M_poly_3_beta	b_huMech3	10.244	7.991	12.666
M_poly_3_beta	b_Intercept	-0.209	-0.513	0.077
M_poly_3_beta	sd_Part__huMech1	3.496	2.421	4.889
M_poly_3_beta	sd_Part__huMech2	7.513	5.099	10.633
M_poly_3_beta	sd_Part__huMech3	4.942	3.404	6.952
M_poly_3_beta	sd_Part__Intercept	0.692	0.517	0.931
M_poly_3_beta_dist	b_huMech1	4.516	2.975	6.085
M_poly_3_beta_dist	b_huMech2	-13.723	-17.080	-10.462
M_poly_3_beta_dist	b_huMech3	9.750	7.587	11.982
M_poly_3_beta_dist	b_Intercept	-0.171	-0.439	0.114
M_poly_3_beta_dist	b_phi_Intercept	1.388	1.084	1.693
M_poly_3_beta_dist	sd_Part__huMech1	3.373	2.393	4.784
M_poly_3_beta_dist	sd_Part__huMech2	7.372	5.312	10.308
M_poly_3_beta_dist	sd_Part__huMech3	4.966	3.579	6.857
M_poly_3_beta_dist	sd_Part__Intercept	0.668	0.497	0.910
M_poly_3_beta_dist	sd_Part__phi_Intercept	0.752	0.580	1.030

Note that the Bayr package does not fully support distributional models, yet, and we have to make without some of the high-level commands, such as grpef.

The standard deviation of `rho` is positive and even half of the population average. While a regular Beta model adjusts the variance according to the usual mean-variance relationship, the distributional model also accounts for overall broader and narrower distributions. We would probably see some subtle adjustments in predictive plots, but this model is also very complex, as every participant get their own `rho`. Therefore, we defer a deeper inspection to the end of the next chapter, where we demonstrate the superiority of the distributional model using information criteria.

```
detach(Uncanny)
```

6.5.3 Participant-level skew in reaction times

In 6.3.2, the Exgaussian model seemed to sit well with reaction times, accomodating their left skew better than Gaussian or Gamma distributions. But if we review the participant-level plots carefully, we see that the shape of randomness differ between participants. There is visible differences in variance, as well as in skew.

The following distributional model estimates the same location effects (PrimeGeek), but grant all participants their own variance and skew. Two distributional parameters, σ for variance and β for skew are added to the multi-line formula interface. The maximum distributional model would estimate fully separate distributions per every participant and condition. But, since the two response distributions (Geek/nogeek) appear similar in the plots, a reduced predictor term is used, assuming that variance and shape are not affected by the experimental condition, but only the person doing the task. That is further justified by the fact, that the effect of priming categories were meager, at best.

```
attach(Hugme)
```

```
M_1_exg_dist  <-
  brm(formula = bf(RT    ~ 1 + PrimeGeek + (1 + PrimeGeek|Part),
                   sigma ~ 1 + (1|Part),
                   beta  ~ 1 + (1|Part)),
      family = exgaussian(),
      data = D_hugme,
      inits = 0)
```

Next, we extract the fixed efefcts, as well as the group-level standard deviation from the posterior. If participants show different patterns of randomness, we should see that in participant-level variation of σ and β.

```
P_1_exg_dist <- posterior(M_1_exg_dist)

P_1_exg_dist %>%
  filter(type %in% c("fixef","grpef")) %>%
  group_by(parameter) %>%
  summarize(center = median(value),
            lower = quantile(value, .025),
            upper = quantile(value, .975)) %>%
  ungroup()
```

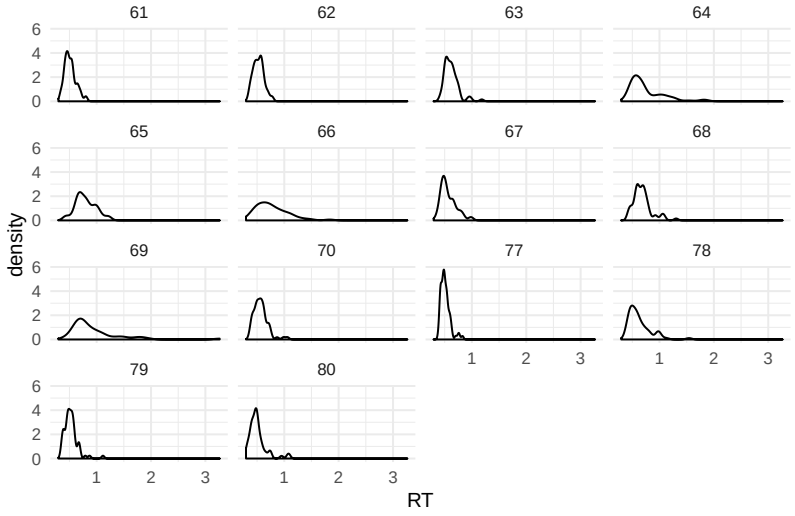
parameter	center	lower	upper
b_beta_Intercept	-2.012	-2.133	-1.877
b_Intercept	0.620	0.595	0.648
b_PrimeGeekTRUE	0.010	0.002	0.018
b_sigma_Intercept	-2.918	-3.039	-2.797
sd_Part__beta_Intercept	0.474	0.394	0.589
sd_Part__Intercept	0.113	0.095	0.137
sd_Part__PrimeGeekTRUE	0.027	0.020	0.035
sd_Part__sigma_Intercept	0.445	0.352	0.564

The estimates confirm, that participants vary significantly in how there RTs are dispersed, as well as how skewed they are. Finally, we can also examine whether there are any associations between location, variance and skew. Recall that parameter `beta` gives the model the flexibility to add extra skew for when the location is far from the left boundary. If that were the case, we would see a correlation between participant-level random effects betwen location and skew. The following extracts the participant-level random effects for `beta`, `mu` and `sigma`:

```
P_1_exg_dist %>%
  filter(type == "ranef" & fixef == "Intercept") %>%
  mutate(dist_par = str_match(parameter, "beta|sigma"),
         dist_par = if_else(is.na(dist_par), "mu", dist_par)) %>%
  group_by(dist_par, re_entity) %>%
  summarize(center = median(value)) %>%
  ungroup() %>%
  spread(key = dist_par, value = center) %>%
  select(-re_entity) %>%
  corrr::correlate()
```

rowname	beta	mu	sigma
beta	NA	0.771	0.040
mu	0.771	NA	0.353
sigma	0.040	0.353	NA

```
D_hugme %>%
  filter(60 < as.numeric(Part) & as.numeric(Part) < 90) %>%
  ggplot(aes(x = RT)) +
  geom_density() +
  facet_wrap(~Part)
```



One could get excited about the strong correlation between `mu` and `beta`, but recall that the amount of skew introduced by `beta` is proportional to the location. At the same time we see a mild correlation between `mu` and `sigma`, which is just the usual mean-variance relationship, which other distributions have factored in. So, from a theoretical point of view, there is not much to gain.

That does not mean the distributional model is a fruitless endeavour. In the following chapter, we will learn how to compare models by predictive accuracy. As it will turn out in 7.2.4, the distributional Beta model in the Hugme case has a better predictive accuracy than the location-only model. The same is true for the Exgaussian distributional model and left to the reader as an exercise.

Distributional models also can do more than just better accomodate data. Applied researchers and experimentalists traditionally embrace on-average effects, but have been mostly blind to effects on variance. In safety relevant systems, strong variance can be a real problem, because this implies that extremely poor performance is becoming more likely, increasing the risk of hazard. For such systems, variance is a crucial parameter and the dispersion effects of design changes should be tracked, carefully. For the readers interested in exercising this technique, several data sets from this book are candidates for such an analysis:

- CUE8: Do teams differ in the variance of ToT variance?
- Egan: What is the most robust web design (the one that reduces variance)?
- IPump: Does training (or the Novel design) reduce variance?

Experimentalists should scratch their heads, too, and take a fresh look at their theories. Systematic variation in variance canm give further clues about what is going on in the minds of their participants. For example, it has been suggested that the Uncanny Valley effect is due to category confusion, which suggests that the longest RTs are to be found in the trough, which [%M&R] have shown to be the case. In [%Keeris], we have fleshed out this theory a little more, basically saying that at every trial the confusion only occurs if the mind picks up enough detail to reject the initial category (Human Face). So, if we reduce the presentation time, we would see more responses where the initial category is kept and the emotional response is positive, rather than extrenmely negative. This hypothesis implies that variance should be largest in the range of the valley, and that with shorter presentation times variance goes up, as responses are a mix of category confusion and category retainers. If we go even shorter, retaining the category will become prominent and variance decreases again.

```
detach(Hugme)
```

In the last three chapters we have met the family of models called *Generalized Multi-level Linear Models* (GMLM). Linear linear models mainly rests on the linear term, or I would rather say: the linear term is the power of linear models. The predictor side of the formula can combine multiple linear terms, at the same time, qualitative data, as captured by factor variables, can be added, too. These simple mechanism unfold into a combinatoric expansion of possibilities.

Technically, multi-level models don't expand the linear term. From a utilitarian perspective, random effects are just useful, because fixef-effects factors do not really shine when factors have an abundance of levels, like participants in an experimental study. But, multi-level models also bring a new perspective, which is painfully missing in many design studies. When we start thinking about users as a population, a population with a good deal of diversity, this makes us think about individuals, too.

Multi-level models also redeem variance as a first-class citizen among model parameters, and we are getting used to think of distributions. Generalized Linear Models expand on that perspective and deal exclusively with what happens on the response side of a model and we learned that for practically all standard measures a Gaussian model is wrong and an (almost) right distribution can just be picked from a whole zoo, using just a few rules. To me, the framework of Generalized Linear Models feel like the sudden relief from pain, when you untie a pair of bad fitting shoes after a full day hike. And sometimes even literally, some real pain, like ordered factors for a learning curve, just goes away.

In the recent past have seen many researchers beginning to see the beauty of the New Statistics approach. Suddenly, people start thinking of models, rather than arcane procedures, and start tinkering. Then their favorite regression engine gets a new feature, say distributional models, and that could be worth a try, could it not?

This book could stop right here, where I leave my readers with an ethusiastic speech, hoping that everybody has reached this page and that some people will read on and write more about interesting models in design research. But, the abundance of possible models leaves the question of when to stop. How do we decise whether introducing, say distributional random effects, really improves the model? As we will see in the final chapter, there are two opposite forces at work, model complexity, which is to be avoided, and model fit. Let's tie our shoes to get the best grip!

7

Working with models

In chapters 4 and 5, we have seen a marvelous variety of models spawning from just two basic principles, linear combination of multiple effects and Gaussian distribution. Chapter 6 further expanded the variety of models, letting us choose response distributions that sit snug. This chapter, is dedicated to methods that assess how snug a model is.

We begin with model criticsm 7.1, where we examine how well one model fits data, we have gathered in the past. In section 7.2 model selection methods are introduces, which help decide which of several models performs better in the future, by providing the most accurate forecasts.

With linear models we can combine and recombine any set of predictors in many ways. We saw examples, where the more simple of two model seemed to work better, but often it was the more complex conditional effects model that was closer to the data. We will use graphical methods to assess *goodness-of-fit*, which helps to scrutinize the structural part of linear models. The random part of linear models is fixed, in that errors follows Gaussian distribution, with a constant variance. Checking these assumptions is based on *residual analysis*. That being said, the section on model criticism introduces techniques that only apply to Gaussian linear models 4, making it a good follow-up for readers who have just worked through that chapter.

Everyone who is already using multi-level models 5 and Generalized Linear Models 6 might want to step right into model selection 7.2, where a more formal approach is presented to compare different models. In contrast to the graphical methods, introduced in @model-criticism, it is applicable for a wider range of models, even wider than the models covered by this book. The problem of comparing multiple models and selecting the fittest first rises the question of what "fittest" actually means and I will settle on the principle of forecasting accuracy. Methods to approximate predictive accuracy are demonstrated and we will see how pruning unnecessary parameters can im-

prove a model. Finally, we will see how model comparison can be used by experimentalists, to test whether a hypotheses is solid.

7.1 Model criticism

For a start, I would like to contrast the overall approach of model criticism to the workflow frequently encountered in classic statistics. Boldly speaking, classicically trained researchers often assume that the "assumptions of ANOVA" need to be checked beforehand. Often a process called *assumptions checking* is carried out before the researcher actually dares to hit the button labelled as *RUN ANOVA*. As we will see in this section, the order of actions is just the other way round. The straight-forward way to check the integrity of a model is to examine the fitted model and find its flaws, which is called *model criticism*. Another symptom of classic assumption checking is the batteries of arcane non-parametric tests that is often carried out. This practice has long been criticized in the statistical literature for its logical flaws and practical limitations. Here, I will fully abandon hypothesis tests for model criticism and demonstrate graphical methods, which are based on two additional types of estimates we can extract from linear models (next to coefficients):

1. with *residuals* we test the assumptions on the shape of randomness
2. with *fitted responses* we check the structural part of the model

7.1.1 Residual analysis

So far, it seems, linear models equip us well for a vast variety of research situations. We can produce continuous and factorial models, account for saturation effects, discover conditional effects and even do the weirdest curved forms. just by the magic of combining linear coefficients. However, the random term stoically stayed the same every time, that it often wasn't worth mentioning:

$$y_i \sim N(\mu_i, \sigma_\epsilon)$$

In words, the randomness term says: *observed values y_i are drawn from a Gaussian distribution located at fitted response μ_i, having a fixed standard deviation σ_ϵ*. In the notation displayed above, there are indeed as many distributions as there are observed values (due the subscript in μ_i). It appears impossible to evaluate not just one distribution, but this many. However, an equivalent notation is routinely used for linear models, that specifies just one *residual distribution*. For the LRM that is:

$$\mu_i = \beta_0 + \beta_1 x_1 y_i = \mu_i + \epsilon_i \epsilon_i \sim \text{Gaus}(0, \sigma_\epsilon)$$

In this notation, observed values y_i are decomposed into predicted values and *individual* residuals ϵ_i. These are frequently called *errors*, hence the greek symbol ϵ. The standard deviation of residuals σ_ϵ is commonly called the *standard error*. The random pattern of the model can now be expressed as a single Gaussian distribution. The reason why I do not use this notation routinely, is that it only works for linear models, but not for models with other random patterns. More specifically, Generalized Linear Models 6 cannot be specified that way. But, for the purpose of residual analysis, it appears more intuitive.

In the previous section we have seen how to extract fitted responses μ_i from estimated models. With fitted responses, we evaluate the structural part of the model, the linear connection of coefficients, which practically always carry the research question. to the main objectives of the research. Residual analysis looks at the pattern of randomness, which the classic linear notation defines as *Gaussian distributed* with *constant variance*.

$$\epsilon_i \sim \text{Gaus}(0, \sigma_\epsilon)$$

As we have seen in 3.4.2, Gaussian distributions are one pattern of randomness among many and this choice may therefore be appropriate, or not. As a heuristic, if observed values are located rather in the middle of the possible range of measurement (and avoid the edges), Gaussian distributions work well. But, like linearity is compromised by saturation, a distribution is affected when squeezed against a hard boundary, as has been discussed in 6.1.

Like with fitted responses, we will use graphical methods, to assess these two assumptions. When either one is violated, the validity of the model is compromised, although maybe not as much as a linear regression model is compromised curved associations like the Unbcanny Valley effect. It is noteworthy at this point, that these two assumptions severely limit what you may actually do with linear models. In some cases, the these assumptions are met well enough, but in many data sets they are visibly violated. Luckily, the linear model with a Gaussian term has been superceded by a class of models that give us more choice from patterns of random ness and let's accomodate pattern of randomness as it is. I may go as far as saying: some readers can skip this part and jump right to chapter **??**.

7.1.1.1 Gaussian residual distribution

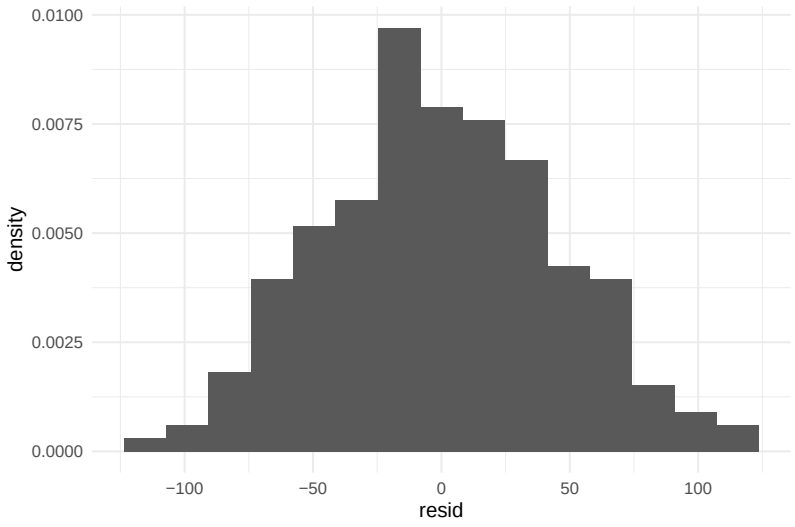
The first assumption of randomness underlying the linear model simply is that the distribution follows a Gaussian distribution. Visually, Gaussian distributions is characterized by:

- one curved peak (unimodality)
- from which density smoothly declines towards both ends (smoothness)
- at same rates (symmetry)

For a rough evaluation of this assumption, it suffices to extract the residuals from the model at hand and plot it as a distribution. The `residuals` command returns a vector of residual values, exactly one per observation. With the vector of residuals at hand, we can evaluate this assumption by comparing the residual distribution to its theoretically expected form, a perfect bell curve. The following command chain extracts the residuals from the model and pipes them into the ggplot engine to create a histogram. With `stat_function` an overlay is created with the theoretical Gaussian distribution, which is centered at zero. The standard error σ_ϵ has been estimated alongside the coefficients and is extracted using the function `clu`.

```
attach(BrowsingAB)
```

```
tibble(resid = residuals(M_age_shft)) %>%
  ggplot(aes(x = resid)) +
  geom_histogram(aes(y = ..density..), bins = 15) +
  stat_function(fun = dnorm,
                args = c(mean = 0,
                         sd = clu(M_age_shft, type = "disp")$center),
                colour = "red")
```



The match of residual distribution with the theoretical distribution is not perfect, but overall this model seems to sufficiently satisfy the normality assumption. To give a counter example, we estimate the same model using the

outcome variable **returns**, which captures the number of times a participant had (desparately) returned to the homepage.

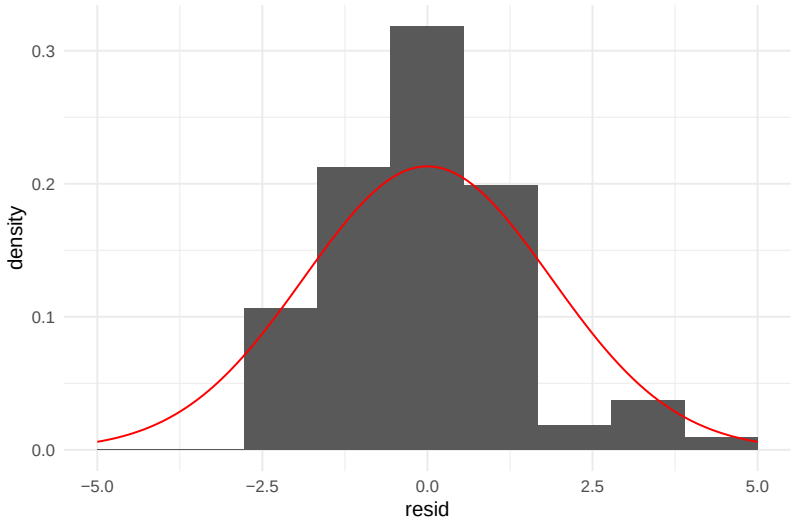
```
M_age_rtrn <-
  stan_glm(returns ~ 1 + age_shft, data = BAB1)
P_age_rtrn <- posterior(M_age_rtrn)
```

```
T_age_rtrn <- clu(P_age_rtrn)
T_age_rtrn
```

parameter	type	fixef	center	lower	upper
Intercept	fixef	Intercept	2.564	2.02	3.113
age_shft	fixef	age_shft	-0.004	-0.02	0.012
sigma_resid	disp	NA	1.872	1.70	2.064

We now graphically compare the distribution of estimated residuals against a perfect Gaussian distribution with the same standard error. The first chain in teh code extracts the center estimate from the table of estimates.

```
C_age_rtrn_sd <-
  T_age_rtrn %>%
  filter(type == "disp") %>%
  select(center) %>%
  as.numeric()

tibble(resid = residuals(M_age_rtrn)) %>%
  ggplot(aes(x = resid)) +
  geom_histogram(aes(y = ..density..), bins = 10) +
  stat_function(fun = dnorm,
                args = c(mean = 0,
                         sd = C_age_rtrn_sd),
                colour = "red") +
  xlim(-5, 5)
```




```
detach(BrowsingAB)
```

The estimation produces the usual coefficients, as well as a standard error. However, the residuals do not even remotely resemble the theoretical curve. While it is unimodal, it appears rather asymmetric, with a steep rise to the left and a long tail to the right. That is a typical outcome when count measures get too close to the left boundary. How about unimodality? We have not discussed any multimodal theoretical distributions in 3.4.2, but one has been displayed in 2.1.3. In brief, a bimodal residual distribution can arise, when two groups exist in the data, which lay far apart. The following code illustrates the situation by simulating a simple data set with two groups, that is fed into a GMM.

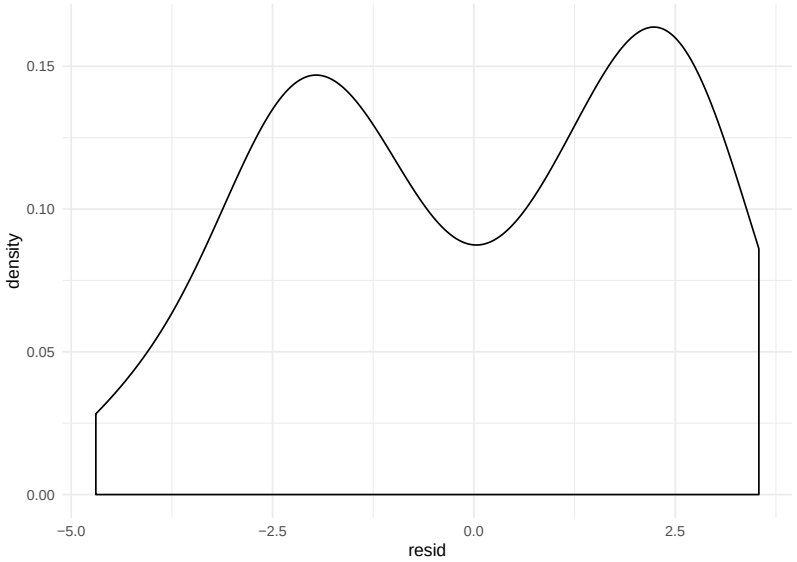
```
attach(Chapter_LM)
```

```
set.seed(42)
D_bimod <-
  bind_rows(
    tibble(Group = "A", y = rnorm(50, 4, 1)),
    tibble(Group = "B", y = rnorm(50, 8, 1))
  )
```

```
M_bimod <- stan_glm(y ~ 1, data = D_bimod, iter = 500)
```

```
D_bimod  %>%
  mutate(resid = residuals(M_bimod)) %>%
```

```
ggplot(aes(x = resid)) +
geom_density()
```



These two deviations from Gaussian distribution have very different causes: asymmetry is caused by scales with boundaries. This is an often arising situation and it is gracefully solved by Generalized Linear Models 6. This is a family of models, where each member covers a certain type of measurement scale. In the above example, Poisson regression, applies, taking care of count measures. Multimodality is caused by heterogeneous groups in the data, like experimental conditions, design or type of user. For a grouping structure to cause distinguished multimodality, differences between groups have to be pronounced in relation to the standard error. It is often the case, that these variables are controlled conditions, such as in an AB test. It is also quite likely that strong grouping structures can be thought of beforehand and be recorded. For example, in usability tests with diverse user samples, it almost comes natural to distinguish between users who have used the design before and those who did not. If the grouping variable is recorded, the solution is group comparison models 4.3.1, already introduced.

Visual assessment of symmetry and unimodality is simple and effective in many cases. But, Gaussian distributions are not the only ones to have these properties. At least logistic distributions and t distributions have these, too, with subtle differences in curvature. Gaussian and t distributions differ in how quickly probability drops in the tails. Gaussian distributions drop much faster, meaning that extreme events are practically impossible. With t-distributions,

extreme values drop in probability, too, but the possibility of catastrophies (or wonders) stays substantial for a long time.

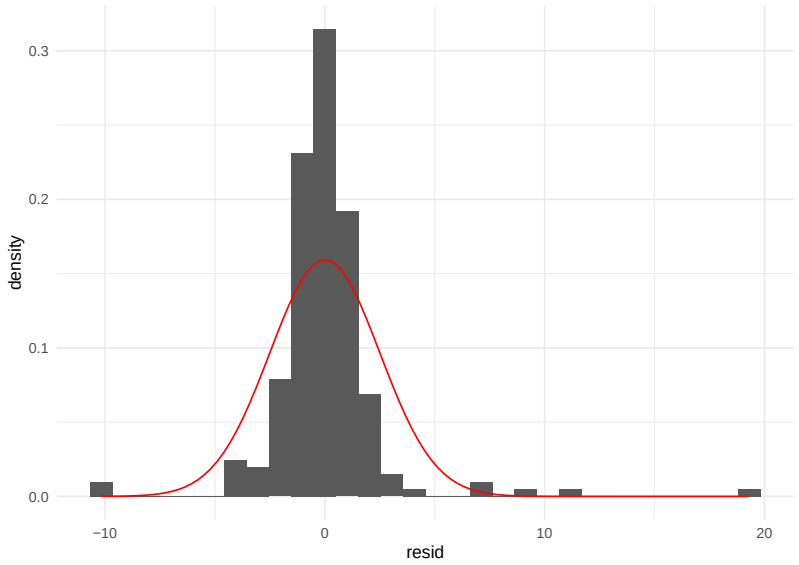
Provided one has a small abundance of data, *quantile-quantile (qq) plots* can be used to evaluate subtle deviations in curvature (and symmetry and unimodality). In qq plots, the observed and theoretical distributions are both flattened and put against each other. This is a powerful and concise method, but it is a bit hard to grasp. The following code illustrates the construction of a qq-plot that compares GMM residuals of a t-distributed measure against the Gaussian distribution. We simulate t-distributed data, run a GMM and extract residuals, as well as the standard error σ_ϵ.

```
set.seed(2)
D_t <- tibble(y = rt(200, 2))
```

```
M_t <- stan_glm(y ~1, data = D_t, iter = 500)
```

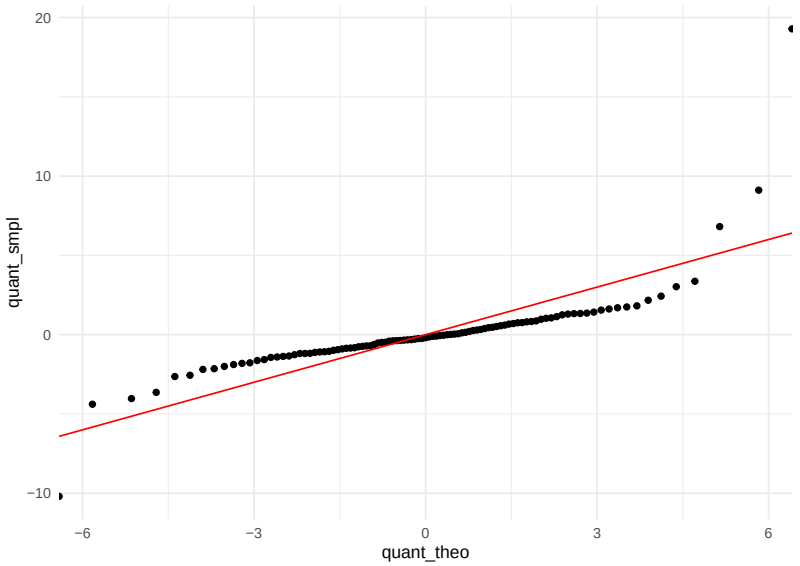
We obtain the following residual and theoretical distributions. It is approximately symmetric and unimodal, but the curvature seems to be a bit off.

```
D_t <-   mutate(D_t, resid = residuals(M_t))

C_sigma <- rstanarm::sigma(M_t)

D_t %>%
  ggplot(aes(x = resid)) +
  geom_histogram(aes(y = ..density..)) +
  stat_function(fun = dnorm,
                args = c(mean = 0,
                         sd = C_sigma),
                colour = "red")
```



The next step is where the two curves get flattened. First, we compute a sequence of quantiles with fixed steps, say 1%, 2%, ... 99%%. Finally, theoretical and observed quantiles are fed into a scatterplot.

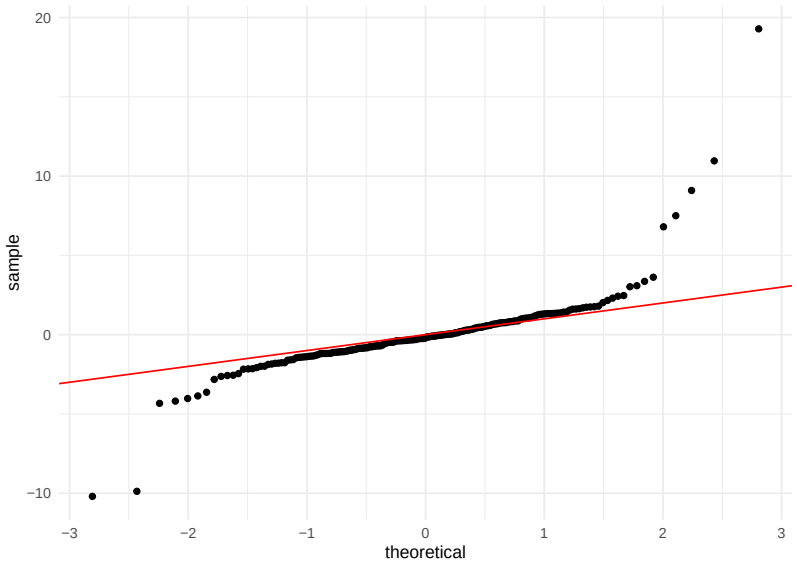
```
D_QQ <- tibble(step = 0:100/100,
               quant_smpl = quantile(D_t$resid, step),
               quant_theo = qnorm(step, 0, C_sigma))

D_QQ %>%
  ggplot(aes(x = quant_theo, y = quant_smpl)) +
  geom_point() +
  geom_abline(slope = 1, intercept = 0, col = "red")
```



In the ideal case, they match perfectly and the quantiles are on a straight line. Instead, we see a rotated sigmoid shape and this is typical for fat-tailed distributions such as t. The shape is symmetric with turning points at around -4 and 4 on the theoretical scale. In the middle part the relation is almost linear, however, not matching a 1-by-1. The t distribution loses probability mass rather quickly when moving from the center to the turning points. From these points on the theoretical quantiles start to lag behind. The lower and upper 1% sampled quantiles go to much more extreme values, ranging from -10 to almost 20, whereas the Gaussian distribution renders such events practically impossible. Generally, a rotated sigmoid shape is typical for fat tailed distributions. The problem of misusing a Gaussian distribution is that it dramatically underestimates extreme events. Have you ever asked yourself, why in the 1990s, the risk for a nuclear meltdown were estimated to be one in 10.000 years, in face of two such tragic events in the past 40 years? Rumor tells, researchers used the Gaussian distribution for the risk models, under-estimating the risk of extreme events.

The ggplot engine provides an easy to use geometry for qqlots, which lets us further explore deviant patterns. Variables with t distribution take an inverse-sigmoid shape due to their fat tails.

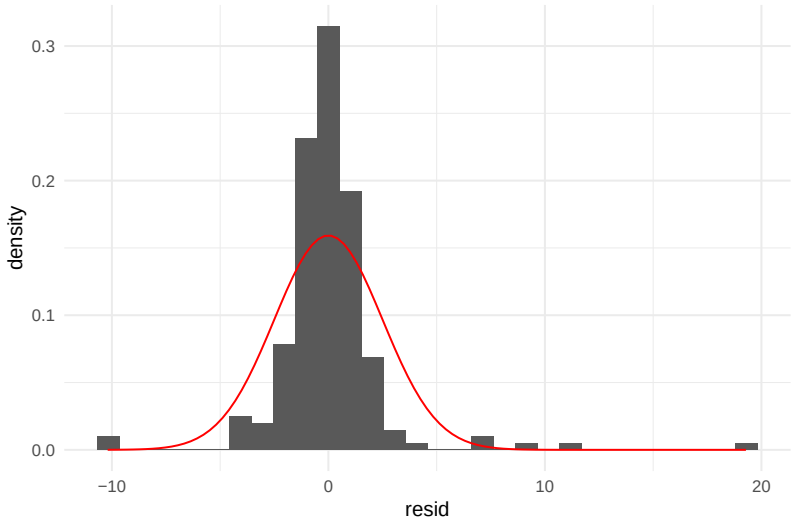
```
D_t %>%
  ggplot(aes(sample = resid)) +
  geom_qq(distribution = qnorm) +
  geom_abline(intercept = 0, slope = 1, col = "red")
```




```
detach(Chapter_LM)
```

Once mastered, the qq-plot is the swiss knife of Normality check. Next to the subtleties, we can also easily discover deviations from symmetry. This is how the residuals of the returns-to-homepage model look like:

```
  tibble(resid = residuals(BrowsingAB$M_age_rtrn)) %>%
  ggplot(aes(sample = resid)) +
  geom_qq(distribution = qnorm) +
  geom_abline(intercept = 0, slope = 1, col = "red")
```



To the left, extreme values have a lower probability than predicted by the Gaussian distribution, but the right tail is much fatter, once again. We also see how residuals are clumped, which is characteristic for discrete (as compared to continuous) outcome measures. This is poor behaviour of the model and, generally, when a model is severely mis-specified, neither predictions nor estimates, nor certainty statements can be fully trusted. A model that frequently fits in case of count numbers is Poisson regression, which will enter the stage in chapter **??**.

7.1.1.2 Constant variance

In **??**, we assessed one assumption that underlies all linear models, namely Gaussian distribution of residuals. The second assumption underlying the linear model random (or residual) is that residual variance is constant throughout the whole range. In both classic and modern notation, this stems from the fact that there is just a single σ_ϵ defined. However large μ_i is, the dispersion of residuals is not supposed to change.

Before we dive into the matter of checking the assumption, let's do a brief reality check using common sense:

1. Consider people's daily commute to work. Suppose you ask a few persons you know: "What is your typical way to work and what is the longest and the shortest duration you remember?". In statistical terms, you are asking for a center estimate and (informal) error dispersion. Is it plausible that a person with typical travel time of 5 minutes experienced the same variation as another person with a typical time of 50 minutes?

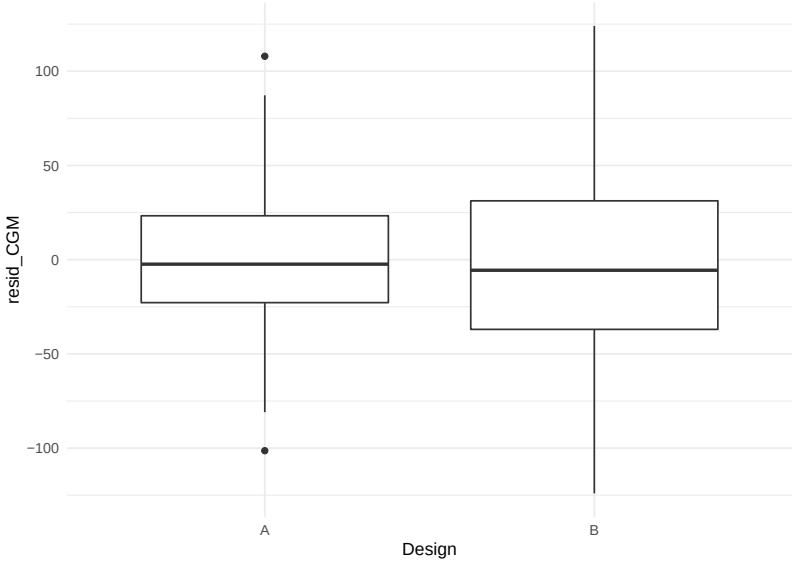
2. Consider an experiment to assess a typing training. Is it plausible that the dispersion of typing errors before the training is the same as after the training?

In both cases, we would rather not expect constant variance and it is actually quite difficult to think of a process, where a strong change in average performance is not associated with a change in dispersion. The constant variance assumption, like the normality assumption is a usual suspect when approximating with linear models. We will come back to that down below.

In a similar way, we can ask: can it be taken for granted that residual variance is constant when comparing two or more groups? Would you blindly assume that two rather different designs produce the same amount of spread around the average? It may be so, but one can easily think of reasons, why this might be different. We check the situation in the CGM of the BrowsingAB study. Do both design conditions have the same residual variance? Again, we extract the residuals, add them to the data set and produce a boxplot by Design condition:

```
attach(BrowsingAB)
```

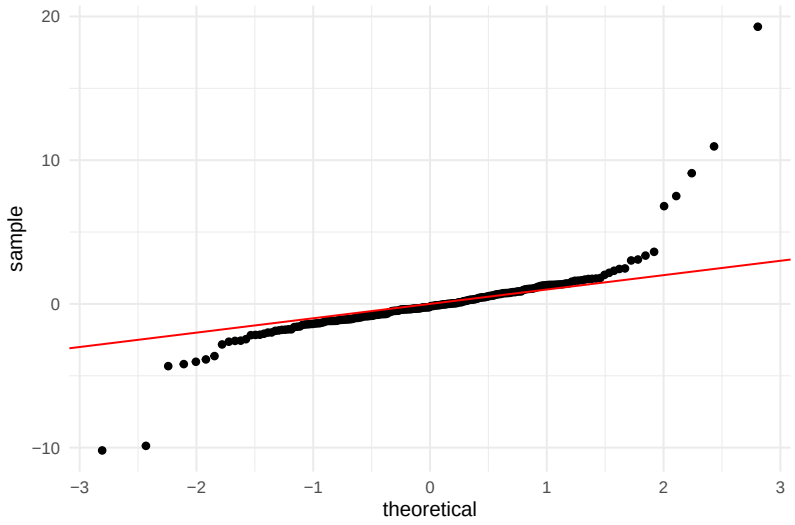
```
BAB1 %>%
  mutate(resid_CGM = residuals(M_CGM)) %>%
  ggplot(aes(x = Design, y = resid_CGM)) +
  geom_boxplot()
```



Both sets of residuals are reasonably symmetric, but it appears that design B produces more widely spread residuals. Something in the design causes individual performance to vary stronger from the population mean. The cause of this effect has been disclosed in 4.5.2. (In essence, design B is rather efficient to use for younger users, whereas older users seem to have severe issues.)

Visual checks of constant variance for factors is straight forward using common boxplots. For continuous predictors, such as age, requires a more uncommon graphical representation known as *quantile plots*. These are not the same as qq plots, but luckily are included with ggplot.

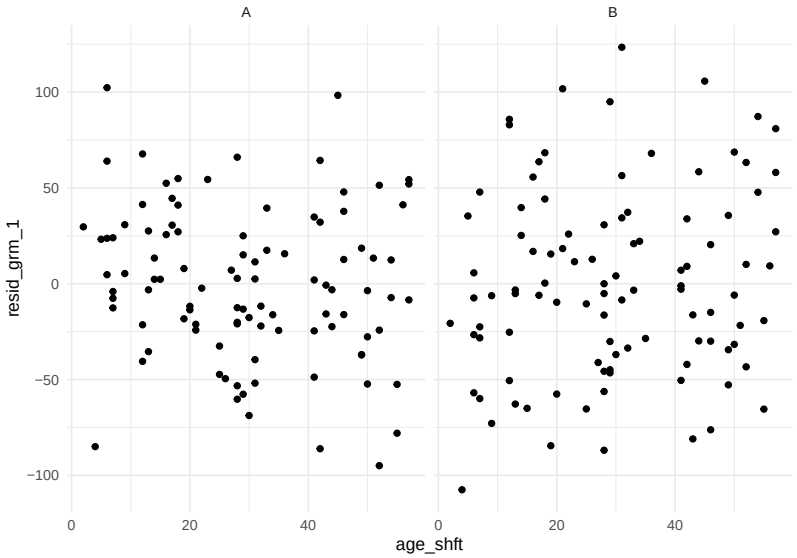
```
BAB1 %>%
  mutate(resid_age = residuals(M_age)) %>%
  ggplot(aes(x = age, y = resid_age)) +
  geom_point() +
  geom_quantile()
```



The quantile plot uses a smoothing algorithm (probably not unlike LOESS) to picture the trend of quantiles (25%, 50% and 75%). Here, the quantiles run almost horizontal and parallel, which confirms constant variance. Taking this as a starting point, we can evaluate more complex models, too. The grouped regression model on age and design just requires to create a grouped quantile plot. This looks best using facetting, rather than separating by colour:

```
BAB1 %>%
  mutate(resid_grm_1 = residuals(M_grm_1)) %>%
  ggplot(aes(x = age_shft, y = resid_grm_1)) +
  facet_grid(~Design) +
```

```
  geom_point() +
  geom_quantile()
```



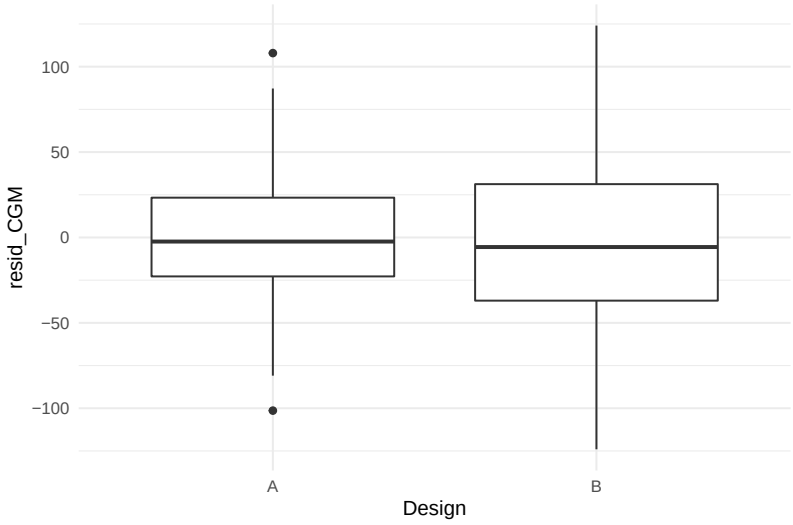
This looks rather worrying. Especially with Design A, the residuals are not constant, but increase with age. In addition, we observe that residuals are not even centered at zero across the whole range. For design A, the residual distribution moves from positive centered to negative centered, design B vice versa. That also casts doubts on the validity of the LRM on age: these contrariwise trends seem to mix into an unsuspicious even distribution. It seems that a lot more has been going on in this study, than would be captured by any of these models.

```
detach(BrowsingAB)
```

Another model type we may want to check with quantile plots is the MRM. With two continuous predictors, one might be tempted to think of a 3-dimensional quantile plot, but this is not recommended. Rather, we can use a generalization of quantile plots, where the x-axis is not mapped to the predictor directly, but the predicted values μ_i. We assess the residual variance on the MRM model on the AUP study, where resistance to fall for the active user paradox has been predicted by geekism tendencies (gex) and need-for-cognition (ncs):

```
attach(AUP)
```

```
AUP_1 %>%
  mutate(resid_3 = residuals(M_3),
         mu_3 = predict(M_3)$center) %>%
  ggplot(aes(x = mu_3, y = resid_3)) +
  geom_point() +
  geom_quantile()
```



We observe a clear trend in quantiles, with residual dispersion increasing with predicted values. Generally, plotting residuals against predicted values can be done with any model, irrespectively of the number and types of predictors. However, interpretation is more limited than when plotting them against predictors directly. In fact, interpretation boils down to the intuition we introduced at the beginning of the section, that larger outcomes typically have larger dispersion. This is almost always a compelling assumption, or even a matter of underlying physics and, once again, a linear model may or may not be a reasonable approximation. Fortunately, Generalized Linear Models 6, provide more reasonable defaults for the relationship between predicted values and dispersion. In contrast, residual variance per predictor allows to discover more surprising issues, such as conditional effects or heterogeneity in groups.

```
detach(AUP)
```

7.1.2 Fitted responses analysis

Frequently, the research question is how strong the effect of a predictor variable is on the response. This strength of effect is what coefficient table tell us. But, how do we know that the model actually fits the process under examination? We have seen several examples of models that do not align well with the observation, in particular when plain MPM were used in the presence of conditional effect. Here, I will introduce technqiue to assess whether a model fits the data well, that is based on this model alone. More precisely, the method is based on *fitted responses*, which is the μ_i that appears in the model formula. We will see that, just like coefficients estimates, fitted responses can be extracted as CLU tables. In essence, by comparing fitted responses to observed responses y_i, we can examine how well a model actually fits the data and identify find potential patterns in the data that the model ignores.

Remember that all linear model assume that an observed value is composed of a structural component, the one that repeats across observations, and a Gaussian distributed error, as can best be seen in the classic notation:

$$y_i = \mu_i + \epsilon_i\mu_i = \beta_0 + \beta_1 x_i \epsilon_i \sim \text{Gaus}(0, \sigma)$$

The first line of the formula does the separation between structural part and pattern of randomness: by estimation, μ_i is the model's idealized response and ϵ_i is the error, taking the form of a zero-centered Gaussian distribution. The purpose of investigating model fit is to find structural features in the obervations that are not rendered by the model at hand. We have already encountered multiple situations like this:

- BrowsingAB: Does an unconditional LRM suffice for the relationship between age and performance?
- IPump: Do both designs have the same learning progress, or do we need conditional effects?
- Uncanny: Does the cubic polynomial provide a right amount of flexibility (number of stationary points), or would a simpler model (a quadratic polynomial) or a more complex model render the relationship better?

Before we come to the method, a warning: *Fitted responses* are more frequently called *predicted values*, and accordinly are extracted by the `predict()` function. This term is in two ways confusing: first, what values? Fair enough, but the second is more profound: The "predicted values" is suggesting that you can use them for forecasting future responses. Well, you can, but only after you tested forecasting accuracy. In 7.2.1 we will see that one can easily create a snug fitting model, without any value for forecasting. (And we will learn how to compare the forecasting ability of models.) As [%McElreath] points

out, one should rather call predicted values *retrodictions*, which is a term that avoids the second I equally like. because they are bound to the current data. I believe the better term is fitted responses, because it reminds us that we are talking about *idealized responses under one model*. That should keep imaginations in check. The match between idealized responses and the *original* observations is called *model fit*. And that is *"fit" as in "fitted"* and not as in "fitness".

We can always compute fitted responses by placing the estimated coefficients into the model formula. A more convenient way is to use the standard command `predict()` on the model object. The package `bayr` provides a tidy version, that produces tidy CLU tables.

```
attach(BrowsingAB)
```

```
T_pred_age_shft <- predict(M_age_shft)
T_pred_age_shft
```

Obs	center	lower	upper
48	191	99.0	281
70	187	95.5	273
95	178	92.0	270
125	184	92.1	275
141	189	100.9	279

There will always be as many fitted responses as there are responses and they come in the same order. They can be attached to the original data, which is very useful for evaluating model fit.

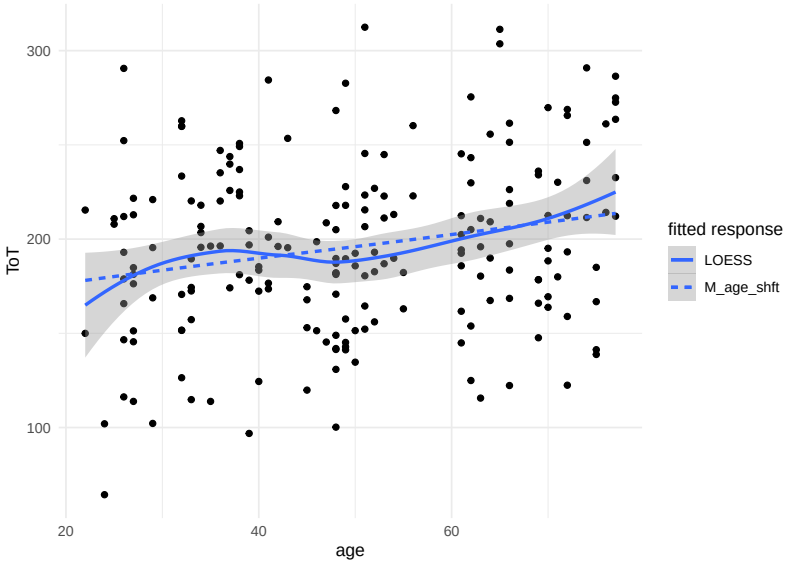
```
BAB1 <- BAB1 %>%
  mutate(M_age_shft = T_pred_age_shft$center)
```

Note that only the center estimate is extracted from the predicted and call the new variable like the model. Later, this will help to collect fitted responses from multiple models.

The evaluation of model fit is a visual task (at this stage). We start with a plot of the raw data, together with a LOESS.

```
BAB1 %>%
  ggplot(aes(x = age, y = ToT)) +
  geom_point() +
  geom_smooth(aes(linetype = "LOESS")) +
```

```
  geom_smooth(aes(y = M_age_shft, linetype = "M_age_shft"), se = F) +
  labs(linetype = "fitted response")
```

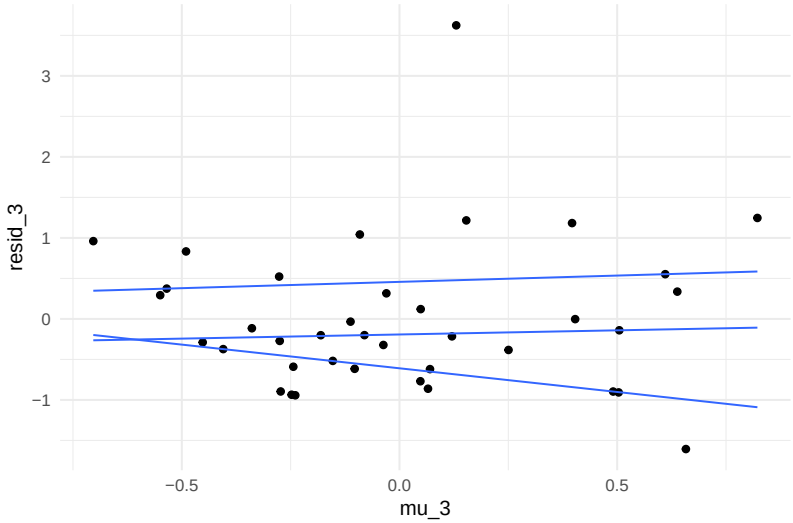


Note that the sequence of smooth geometries all use different sources for the y coordinate. The literal values for the color aesthetic produce the legend; the legend title is created by `labs()`.

The only two features the LOESS smoother and the linear model have in common is their total upward trend and fitted responses at ages 20 and 50. The LOESS indicates some wobbliness in the response, with something that even could be a local maximum at around age 37. One has to be a little careful here, because LOESS is just another engine that produces a set of fitted responses. LOESS i a very flexible model, and as we will see in 7.2.1, flexible models tend to over-fit, which means that they start to pull noise into the structural part. This results in less accurate forecasts. In conclusion, LOESS and LRM tell different stories and we cannot tell which one is closer to the truth without further investigation. Conditional effects are always among the suspects, when in comes to non-linearities. The wobbly relationship could be the result of a mixture of conditions. By pulling the Factor Design into the graph, we can assess how well the unconditional model fits both conditions.

```
G_Design_age <-
  BAB1 %>%
  ggplot(aes(x = age, y = ToT, col = Design)) +
  geom_point() +
  geom_smooth(aes(linetype = "LOESS"), se = F) +
```

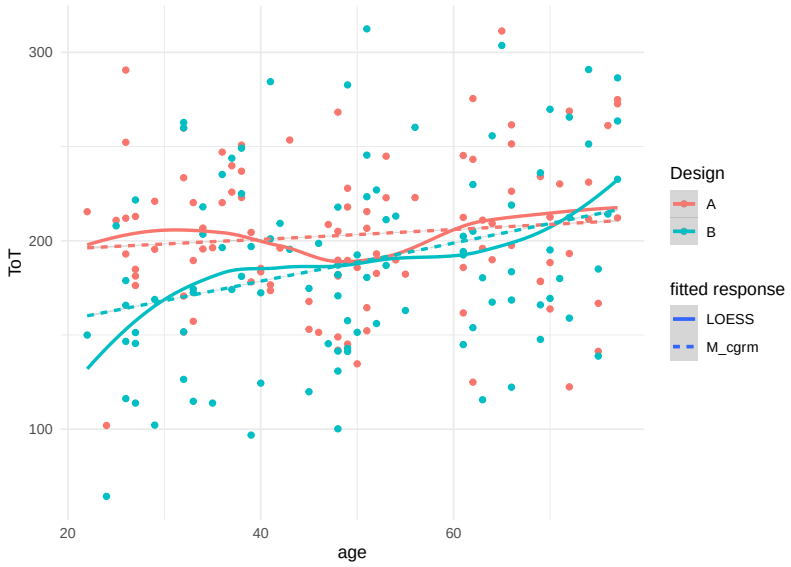
```
  labs(linetype = "fitted response")

G_Design_age + geom_smooth(aes(y = M_age_shft, linetype = "M_age_shft"), se = F)
```



It turns out, that the MRM only fits for part of the obervations, namely those on design B and between age of 35 and 65. Most strikingly, the model seems to fit worst at age of 20, which we identified as a point of good prediction. As it turns out, this is just a good prediction in total, with strongly deviating local intercepts. The model seems in need for a conditional term, like in the following CGRM

```
M_cgrm <- BAB1 %>%
  stan_glm(ToT ~ Design * age, data = .)
```

```
BAB1$M_cgrm <- predict(M_cgrm)$center

G_Design_age %+%
  BAB1 +
  geom_smooth(aes(y = M_cgrm, linetype = "M_cgrm"))
```



The improved model now seems captures the overall increment by age in both conditions. Apart from the age-50 dip, the DesignA condition reasonably fitted. The model also predicts a cross-over point at age of 73, where both designs are equal. In contrast, the model cannot adequately render the association in design B, which appears inverse-sigmoid. These non-linear associations stem from a fancy psychological model I have put together at the end of the long night. Let us take a look at some real wobbles, instead.

```
detach(BrowsingAB)
```

The Uncanny Valley effect is all about non-linearity and we have seen in 4.6 how a complex curves can be captured by higher-degree polynomials. With every degree added to a polynomial, the model gets one more coefficient. It should be clear by now that models with more coefficients are more flexibel. In multi-factorial models, adding a conditional term lets all group means move freely, whereas the flexibility of a polynomial can be measured by how many stationary points are possible, because you need one for every peak and valley. Higher degree polynomials can do even more tricks, such as saddle points, that have a local slope of zero without changing direction.

Mathur & Reichling identified a cubic polynomial as the lowest degree that would render the Uncanny Valley effect, which has at least one local maximum and one local minimum (both are stationary points). In fact, they also conducted a formal model comparison, which approved that adding higher degrees does not make the model better. Such a formal procedure is introduced in **??**testing-theories), whereas here we use visualizations of fitted responses to evaluate the possible models.

In the following, the cubic model is compared to the simpler quadratic model. It could be, after all, that a parabole is sufficient to render the valley. On the other side of things, a polynomial model with the ridiculous degree 9 is estimated, just to see whether there is any chance a more complex model would further improve the fit.

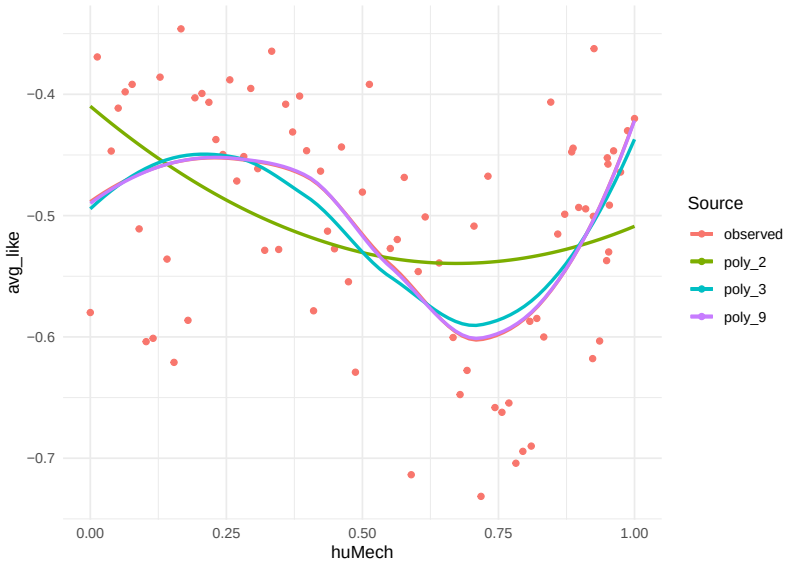
```
attach(Uncanny)
```

```
M_poly_2 <- RK_2 %>%
  stan_glm(avg_like ~ poly(huMech, 2), data = .)

M_poly_3 <- RK_2 %>%
  stan_glm(avg_like ~ poly(huMech, 3), data = .)

M_poly_9 <- RK_2 %>%
  stan_glm(avg_like ~ poly(huMech, 9), data = .)
```

```
sync_CE(Uncanny, M_poly_2, M_poly_3, M_poly_9)
```

```
PP_poly <- bind_rows(post_pred(M_poly_2),
                     post_pred(M_poly_3),
                     post_pred(M_poly_9))

T_fit_poly <-
  predict(PP_poly) %>%
  select(model, Obs, center) %>%
  spread(model, center)
```

```
RK_2 %>%
  bind_cols(T_fit_poly) %>%
  ggplot(aes(x = huMech)) +
  geom_point(aes(y = avg_like, col = "observed")) +
  geom_smooth(aes(y = avg_like, col = "observed"), se = F) +
  geom_smooth(aes(y = M_poly_2, col = "poly_2"), se = F) +
  geom_smooth(aes(y = M_poly_3, col = "poly_3"), se = F) +
  geom_smooth(aes(y = M_poly_9, col = "poly_9"), se = F) +
  labs(col = "Source")
```



Note that:

- all posterior predictive distributions (PPD) are collected into one multi-model posterior predictions object (class tbl_post_pred)
- from the PP, multi-model CLUs are then created at once and turned into a wide table using `spread`, which can be attached to the original data.

We observe that the two lower-degree polynomials deviate strongly from the observed pattern. The quadratic polynomial is a parabole and one could expect it to fit the valley part somewhat, but it does not. The cubic polynmial curve almost seems to snap into the right position. That is confirmed by observing that using a 9-degree polynomial barely changes the fitted curve, although, according to polynomial theory, this model could have as many as eight stationary points, which is a ridiculous amount of flexibility. This polynomial also sits perfectly snug with the LOESS (on the original observations), which means it is at least as flexible. These results confirm that [%Mathur & Reichling] got it just right with using a cubic polynomial. This model captures the salient features of the data, not more, but also not less. One could argue, that since the 9-degree polynomial makes almost the ame predictions as the cubic polynomial, it does no harm to always estimate a more flexible model. As we will see in the 7.2.1, this is problematic, as models with unnecessary flexibility tends to "over-fit", which means seeing structure in the noise, which reduces forecasting accuracy.

To sum it up, visual analysis on fitted responses is an effective way to discover shortcomings of a Gaussian linear model with respect to the structural part. A possible strategy is to start with a basic model, that covers just the main

research questions and explore how well it performs under different conditions. With metric predictors, fitted response analysis can uncover problems with the linearity assumption.

7.2 Model selection

If one measures two predictors x_1, x_2 and one outcome variable y, formally there exist four linear models to choose from:

- `y ~ 1` (grand mean)
- `y ~ x_1` (main effect 1)
- `y ~ x_2` (main effect 2)
- `y ~ x_1 + x_2` (both main effects)
- `y ~ x_1 * x_2` (both main effects and interaction)

For a data set with three predictors, the set of possible models is already 18. This section deals with how to choose the right one. In 4, we have seen multiple times, how a model improves by adding another coefficient, for example a missing conditional effect 4.5 can lead to severe biases. The opposite of such under-specification is when a model carries unnecessary parameters, which causes *over-fitting* and reduces *predictive accuracy*. The subsequent sections accept predictive accuracy as a legitimate measure for model comparison and introduces methods to measure predictive accuracy: simulation, cross validation, leave-one-out cross validation and information criteria. The final two sections show how model selection can be used to test hypothesis, without using the infamous p-value.

7.2.1 The problem of over-fitting

In this chapter we have seen multiple times how a model that is too simple fails to align with the structure presnet in the data. For example, recall the IPump case, where an ordinal model was much more accurate in rendering the learning process than the simpler regression model. In several other cases, we have seen how introducing conditional effects improves model fit. At these examples, it is easy to see how omitting relevant predictors reduces the predictive accuracy of a model.

Too sparse models produce inferior predictions, but that is only one side of the coin: Models that contain *irrelevant predictors* also produce inferior predictions. This is called *over-fitting* and can be understood better if we first recall, that a models job is to divide observed magnitudes into the structural part and the random part 3.4. The structural part is what all observations

have in common, all future observations included. The structural part always is our best guess and the better a model separates structure from randomness, the more accurate our forecasts become. The process of separation is imperfect to some degree. Irrelevant predictors usually get center estimates close to zero in a linear model, but their posterior distribution (or credibility intervals) usually has its probability mass spread out over a range of non-zero values. The irrelevant parameter adds a degree of freedom which introduces additional uncertainty. As an analogy, an irrelevant parameter is to a model, what a loose wheel is to a car. Even on a perfectly flat road it will cause a wobbly driving experience.

For further illustration, we simulate a small data set by drawing two variables from two Gaussian distributions. One variable is the (presumed) predictor, the other is the outcome and because they are completely unrelated, a GMM would be appropriate. But what happens if the researcher assumes that there is a linear relation and adds the irrelevant predictor to the model?

```
attach(Chapter_LM)
```

```
sim_overfit <- function(n = 10, seed = 1317){
  set.seed(seed)
  tibble(pred    = rnorm(n = 10, mean = 0, sd = 1),
         outcome = rnorm(n = 10, mean = 2, sd = 1)) %>%
  as_tbl_obs()}


D_overfit <- sim_overfit()
```

```
M_gmm <-  stan_glm(outcome ~ 1,        data = D_overfit, iter = 2000)
M_lrm <-  stan_glm(outcome ~ 1 + pred, data = D_overfit, iter = 2000)
```

```
P_overfit <- bind_rows(posterior(M_gmm), posterior(M_lrm))
coef(P_overfit)
```

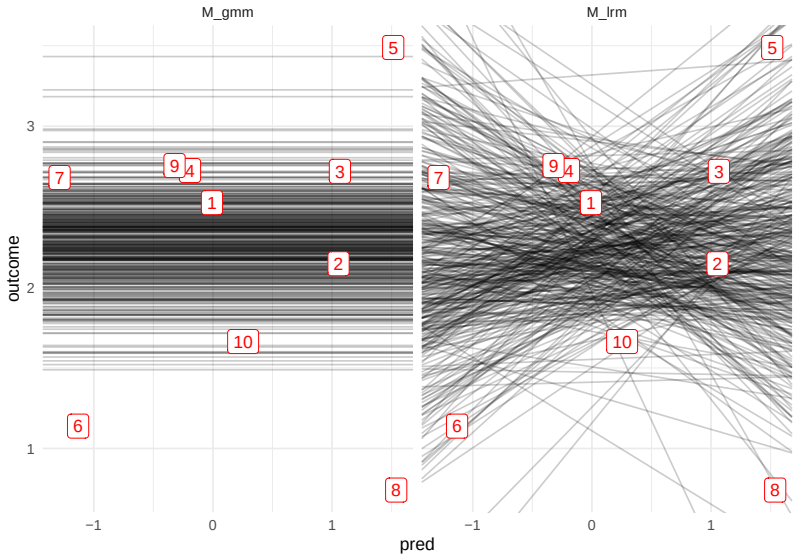
model	parameter	fixef	center	lower	upper
M_gmm	Intercept	Intercept	2.255	1.684	2.840
M_lrm	Intercept	Intercept	2.255	1.609	2.894
M_lrm	pred	pred	0.018	-0.674	0.652

We first examine the coefficients. We observe that both models produce the same center estimates for Intercept. At the same time, the LRM predictor is centered at a point very close to zero. The good news is that irrelevant predictors usually do not add any biases. The credibility intervals are a different

story. The most striking observation is that the LRM is very uncertain about the slope. The possibility of considerable positive or negative slopes is not excluded at all. Next to that, the intercept of the LRM bears more pronounced uncertainty, compared to the GMM.

The following plot illustrates the mechanism behind over-fitting. It is created by extracting intercept and slope parameters from the posterior distributions and plot them as a bunch of linear functions. For the GMM, all slopes are set to Zero, but we observe that the LRM has visited many rather strong slopes. These extreme slopes are mostly caused by the extreme observations Five, Six and Eight, which the LRM tries to reach, while the GMM stays relatively contained, assigning most of these extreme values to the random part. Finally, by the distribution of slopes, the distribution of left end-points is pulled apart and that is what we see as an extra uncertainty in the intercept.

```
P_overfit %>%
  filter(type == "fixef") %>%
  select(model, iter, parameter, value) %>%
  spread(key = parameter, value = value, fill = 0) %>%
  filter( (iter %% 10) == 0 ) %>%
  ggplot() +
  facet_wrap(~model) +
  geom_abline(aes(intercept = Intercept, slope = pred), alpha = .2) +
  geom_point(aes(x = pred, y = outcome), data = D_overfit, col = "Red") +
  geom_label(aes(x = pred, y = outcome, label = Obs), data = D_overfit, col = "Red")
```



Every parameter that is added to a model, adds to it some amount of flexibility. When this parameter is influential within the structure, the extra flexibility improves the fit. When it is not, the extra flexibility grabs on too much randomness, with the consequence of reduced predictive accuracy. *Model pruning* is the process of discarding unnecessary parameters from a model until it reaches its maximum predictive accuracy. That is easy if you use simulated data, but in practice predictive accuracy can only be estimated by throwing new data at an estimated model. Nevertheless, unnecessary parameters in linear models can often sufficiently be identified by two simple rules (which actually are very similar to each other):

1. The center estimate is close to Zero.
2. If the parameter is removed, this causes little change in other parameters.

In the following, I will introduce formal methods to model pruning. These can be used in more complex situations, such as pruning multi-level models or selecting an appropriate error distribution type.

7.2.2 Cross validation and LOO

Recall that coefficients are tied to fitted responses by the structural part. Consequently, stronger uncertainty in coefficients causes stronger uncertainty of predictions. Are the predictions of LRM inferior to the parsimonous GMM? Since we have simulated this data set, the true population mean is known ($\mu = 2$) and we can assess predictive accuracy by comparing the deviation of fitted responses from the true value. A standard way of summarizing the predictive accuracy of models is the *root mean square error (RMSE)*, which we can compute from the *posterior predictive* distributions.

```
RMSE <- function(true, value) {
  se   <- (true - value)^2
  mse  <- mean(se)
  rmse <- sqrt(mse)
  rmse
}
```

```
PP_overfit <-
  bind_rows(post_pred(M_gmm),
            post_pred(M_lrm)) %>%
  left_join(D_overfit, by = "Obs")

PP_overfit %>%
  group_by(model) %>%
  summarize(RMSE = RMSE(true = 2, value))
```

model	RMSE
M_gmm	1.01
M_lrm	1.13

The LRM has a larger error by around 10%, which is a lot for just one additional parameter. The RMSE of the GMM is close to One, which is almost precisely the standard deviation of the simulation function; the GMM has found just the right amount of randomness.

In practice, the central dilemma in evaluating predictive accuracy is that usually we do not know the real value. The best we can do, is use one data set to estimate the parameters and use a second data set to test how well the model predicts. This is called *cross validation* and it is the gold standard method for assessing predictive accuracy. Here, we can simulate the situation by using the simulation function one more time to produce future data, or more precisely: the data new to the model.

```
D_new_data <- sim_overfit(n = 100, seed = 1318)

PP_cross_valid <-
  bind_rows(post_pred(M_gmm, newdata = D_new_data),
            post_pred(M_lrm, newdata = D_new_data)) %>%
  left_join(D_new_data, by = "Obs")

PP_cross_valid %>%
  group_by(model) %>%
  summarize(RMSE = RMSE(value, outcome))
```

model	RMSE
M_gmm	1.14
M_lrm	1.25

Waiting for new data before you can do model evaluation sounds awful, but new data is what cross validation requires. More precisely, cross validation only requires that the forecast data is not part of the sample you trained the model with. Psychometricians, for example, use the split half technique to assess the reliability of a test. The items of the test are split in half, one training set and one forecasting set. If the estimated participant scores correlate strongly, the test is called reliable.

So, model evaluation can be done, by selecting on part of the data to train the model, i.e. estimate the coefficients, and try to forecast the other part of the data. However, data is precious and reserving half of it for forecasting will considerably be at the cost of certainty. Fortunately, nobody actually said it has to be half the data. Another method of splitting has become common, *leave-one-out (LOO) cross validation*. The idea is simple:

1. Remove observation i from the data set.
2. Estimate the model $M_{/i}$.
3. Predict observation i with Model $M_{/i}$.
4. Measure the predictive accuracy for observation i.
5. Repeat steps 1 to 4 until all observations have been left out and forecast once.

The following code implements a generic function to run a LOO analysis using an arbitrary model.

```
do_loo <-
  function(data,
           F_fit,
           f_predict = function(fit, obs) post_pred(fit, newdata = obs))
  {

    model_name <- as.character(substitute(F_fit))
    F_train_sample <- function(obs) data %>% slice(-obs) # Quosure
    F_test_obs  <- function(obs) data %>% slice(obs)  # Quosure
    F_post_pred <- function(model, model_name, newdata, this_Obs) {
      post_pred(model = model,
                model_name = model_name,
                newdata = newdata) %>%
        mutate(Obs = this_Obs)}

    out <-   tibble(Obs          = 1:nrow(data),
                    Model        = model_name,
                    Train_sample = map(Obs, F_train_sample), # training observations
                    Test_obs     = map(Obs, F_test_obs),  # test observation
                    Fitted       = map(Train_sample, F_fit)) %>%   # model objects
      mutate(Post_pred = pmap(list(model = Fitted,
                                   newdata = Test_obs,
                                   model_name = Model,
                                   this_Obs = Obs), F_post_pred))
    return(out)
  }
```

Before we put `do_loo` to use, some notes on the programming seem in order. Despite its brevity, the function is highly generic in that it can compute leave-one-out scores no matter what model you throw at it. This is mainly achieved by using advanced techniques from *functional programming*:

1. The argument `f_fit` takes an arbitrary function to estimate the model. This should work with all standard regression engines.

2. The argument `f_predict` takes a function as argument that produces the predicted values for the removed observations. The default is a function based on `predict` from the bayr package, but this can be adjusted.
3. The two functions that are defined inside `do_loo` are so-called *quosures*. Quosures are functions that bring their own copy of the data. They can be conceived as the functional programming counterpart to objects: Not the object brings the function, but the function brings its own data. The advantage is mostly computational as it prevents data to be copied every time the function is invoked.
4. `map` is a meta function from package purrr. It takes a list of objects and applies an arbitrary function, provided as the second argument.
5. `map2` takes two parallel input lists and applies a function. Here the forecast is created by matching observations with the model they had been excluded from.
6. The function output is created as a *tibble*, which is the tidy re-implementation of data frames. Different to original `data.frame` objects, tibbles can also store complex objects. Here, the outcome of LOO stores every single sample and estimated model, neatly aligned with its forecast value.
7. Other than one might expect, the function does not return a single score for predictive accuracy, but a dataframe with inidividual forecasts. This is on purpose as there is more than one possible function to choose from for calculating a single accuracy score.
8. The function also makes use of what is called non-standard evaluation. This is a very advanced programming concept in R. Suffice it to say that `substitute()` captures an expression, here this is the fitting function argument, without executing it, immediatly. Here the provided argument is converted to character and put as an identifier into the dataframe. That makes it very easy to use `do_loo` for multiple models, as we will see next.

Since we want to compare two models, we define two functions, invoke `do_loo` twice and bind the results in one data frame. As the model estimation is done per observation, I dialed down the number of MCMC iterations a little bit to speed up the process:

```
fit_GMM  <-  function(sample) stan_glm(outcome ~ 1,        data = sample, iter = 500)
fit_LRM  <-  function(sample) stan_glm(outcome ~ 1 + pred, data = sample, iter = 500)

Loo <- bind_rows(do_loo(D_overfit, fit_GMM),
                 do_loo(D_overfit, fit_LRM))

Loo %>% sample_n(5)
```

```
sync_CE(Chapter_LM, Loo)
```

```
## [1] "Loo"
```

This data frame `Loo` is somewhat unusual, because also it stores complex R objects, rather than atomic values. Doing a full LOO run is very computing expensive and therefore it makes a lot of sense to save all the models for potential later use.

Model comparison is again based on the posterior predictive distribution, we are only interested in the posterior predictive distributions. The following code merges all posterior predictions into one multi-model posterior prediction table and joins it with the teh original observations. Now we can compute the RMSE and we even have the choice to do it on different levels. On a global level, the prediction errors of all observations are pooled, but we can also summarize the prediction error on observations level, or plot the predictive distribution.

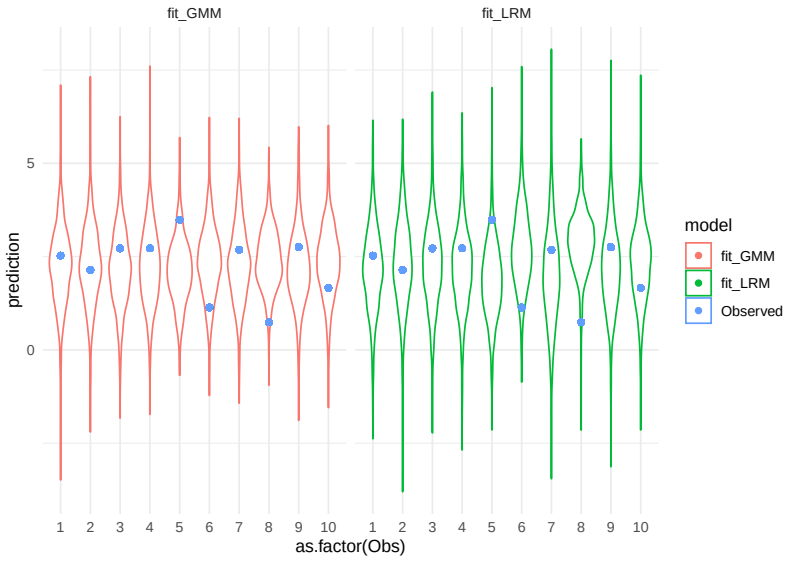
```
PP_Loo <-
  bind_rows(Loo$Post_pred) %>%
  left_join(D_overfit) %>%
  rename(prediction = value)

PP_Loo %>%
  group_by(model) %>%
  summarize(RMSE = RMSE(prediction, outcome))
```

model	RMSE
fit_GMM	1.33
fit_LRM	1.55

```
PP_Loo %>%
  group_by(model, Obs) %>%
  summarize(RMSE = RMSE(prediction, outcome)) %>%
  spread(value = RMSE, key = model)
```

Obs	fit_GMM	fit_LRM
1	1.14	1.21
2	1.09	1.18
3	1.20	1.23
4	1.15	1.26
5	1.64	1.95
6	1.54	1.88
7	1.12	1.50
8	1.85	2.30
9	1.16	1.32
10	1.21	1.25

```
PP_Loo %>%
  ggplot(aes(x = as.factor(Obs), y = prediction, color = model)) +
  facet_grid(~model) +
  geom_violin() +
  geom_point(aes(y = outcome, col = "Observed"))
```



7.2.3 Information Criteria

So far, we have seen that the right level of parsimony is essential for good predictive accuracy. While LOO can be considered gold standard for assessing predictive accuracy, it has a severe downside. Estimating Bayesian models with MCMC is very computing intensive and for some models in this book,

doing a single estimating is in the range of dozens of minutes to more than an hour. LOO estimates the model as many times as there are observations, which quickly leads to unbearable computing time for larger data sets.

Information criteria account for goodness-of-fit, but also penalizes more flexible models. The oldest of all IC is the *Akaike Information Criterion (AIC)*. Compared to its younger siblings, it is less broad, but its formula will be instructive to point out how goodness-of-fit and complexy are balanced within one formula. To represent goodness-of-fit, the AIC employs the *deviance*, which directly derives from the Likelihood (3.3.3). Model complexity is accounted for by a penalty term that is just two times the number of parameters k.

$$\text{Deviance} = -2\log(p(y|\hat{\theta})).\text{Penality} = 2k AIC = D + 2k$$

By these two simple terms the AIC brings model fit and complexity into balance. Note that lower deviance is better and so is lower complexity. In effect, when comparing two models, the one with the *lower AIC wins*. As it grounds on the likelihood, it is routinely been used to compare models estimated by classic maximum likelihood estimation. The AIC formula is ridiculously simple and still has a solid foundation in mathematical information theory. It is easily computed, as model deviance is a by-product of parameter estimation. And if that was not enough, the *AIC is an approximation of LOO cross-validation*, beating it in computational efficiency.

Still, the AIC has limitations. While it covers the full family of Generalized Linear Models 6, it is not suited for Multi-level Models 5. The reason is that in multi-level models the degree of freedom (its flexibility) is no longer proportional to the formal number of parameters. Hence, the AIC penalty term is over-estimating. The *Deviance Information Criterion (DIC)* was the first generalization of the AIC to solve this problem by using an estimate for degrees of freedom. These ideas have more recently been refined into the *Widely Applicable Information Criterion (WAIC)*. Like DIC this involves estimating the penalty term p_{WAIC}. In addition, WAIC makes use of the full posterior predictive distribution, which is results in the *estimated log pointwise predictive density*, $\text{elpd}_{\text{WAIC}}$ as goodness-of-fit measure [%REF]. The standard implementation of WAIC is provided by package Loo, and works with all models estimated with Rstanarm or Brms. When invoked on a model, it returns all three estimates:

```
attach(Chapter_LM)
```

```
loo::waic(M_gmm)
```

```
##
```

```
## Computed from 4000 by 10 log-likelihood matrix
##
##           Estimate  SE
## elpd_waic    -13.9 2.1
## p_waic         1.6 0.6
## waic          27.8 4.2
##
## 1 (10.0%) p_waic estimates greater than 0.4. We recommend trying loo instead.
```

Compared to LOO-CV, the WAIC is blazing fast. However, the WAIC has two major down-sides: First, while the RMSE has a straight-forward interpretation as the standard deviation of the expected error, WAICs are unintelligable by themselves. They only indicate relative predictive accuracy, when multiple models are compared. Second, WAIC as an approximation can be wrong. Fortunately, as can be seen from the warning above, the WAIC command performs an internal check on the integrity of the estimate. When in doubt, the function recommends to try loo instead.

This is not the full LOO-CV method, but another approximation: LOO-IC often is more reliable approximation of the real LOO-CV, than WAIC. It is slower than WAIC, but still has reasonable computation times. Again, the LOO-IC implementation features an extra safety feature, by checking the integrity of results and helping the user to fix problems.

```
Loo_gmm <- loo(M_gmm, cores = 1)
Loo_lrm <- loo(M_lrm, cores = 1)
```

```
Loo_gmm
```

```
##
## Computed from 4000 by 10 log-likelihood matrix
##
##          Estimate  SE
## elpd_loo    -14.0 2.1
## p_loo         1.7 0.6
## looic        28.0 4.3
## ------
## Monte Carlo SE of elpd_loo is 0.1.
##
## Pareto k diagnostic values:
##                          Count Pct.    Min. n_eff
## (-Inf, 0.5]   (good)     9     90.0%   1135
##  (0.5, 0.7]   (ok)       1     10.0%   359
##    (0.7, 1]   (bad)      0      0.0%   <NA>
##    (1, Inf)   (very bad) 0      0.0%   <NA>
```

```
##
## All Pareto k estimates are ok (k < 0.7).
## See help('pareto-k-diagnostic') for details.
```

```
Loo_lrm
```

```
##
## Computed from 4000 by 10 log-likelihood matrix
##
##          Estimate  SE
## elpd_loo    -16.3 3.0
## p_loo         3.8 1.7
## looic        32.5 6.0
## ------
## Monte Carlo SE of elpd_loo is NA.
##
## Pareto k diagnostic values:
##                          Count Pct.    Min. n_eff
## (-Inf, 0.5]   (good)     6     60.0%   1644
##  (0.5, 0.7]   (ok)       3     30.0%   481
##    (0.7, 1]   (bad)      0      0.0%   <NA>
##    (1, Inf)   (very bad) 1     10.0%   15
## See help('pareto-k-diagnostic') for details.
```

The interpretation of LOO-IC is the same as for all information criteria, the value with the smaller IC wins. What often confuses users of information criteria is when they see two ICs that are huge with only a tiny difference, like 1001317 and 1001320. Recall that information criteria all depend on the likelihood, but on a logarithmic scale. What is a difference on the logarithmic scale is a multiplier on the original scale of the likelihood, which is a product of probabilities. And a small difference on the log scale can be a respectable multiplicator:

```
exp(1001320 - 1001317)
```

```
## [1] 20.1
```

For a real application, we pick up the infusion pump case, where we left it in section 6.2.1.2, where we informally compared three Poisson models for the learning rate on path deviations:

- an ordered factor model with four learning rate coefficients (`M_pois_cozfm`)
- a linearized regression with two learning rate coefficients (`M_pois_clzrm`)

- a linearized regression with one learning rate coefficient (`M_pois_lzrm`)

From the coefficients it seemed that the OzFM can best be replaced by the more parsimonous LzRM. Still, the there remains a slight differences in the two steps. This is a good example where formal model selection is useful.

```
attach(IPump)
```

```
L_pois_cozfm <-  loo(M_pois_cozfm)
L_pois_clzrm <-  loo(M_pois_clzrm)
L_pois_lzrm <-  loo(M_pois_lzrm)

loo_compare(x = list(L_pois_lzrm, L_pois_clzrm, L_pois_cozfm))
```

	elpd_diff	se_diff	elpd_loo	se_elpd_loo	p_loo	se_p_loo	looic	se_looic
M_pois_lzrm	0.00	0.000	-482	21.6	7.33	1.14	964	43.1
M_pois_clzrm	-1.49	0.208	-484	21.6	9.21	1.33	967	43.3
M_pois_cozfm	-3.55	2.301	-486	21.3	13.29	1.60	971	42.7

Model comparison with LOO-IC points us at the unconditional LzRM as the most parsimonous model, followed by the conditional LzRM; the conditional OFM goes in last. Model comparison confirms, that we may assume a learning rate that is approximately constant across the learning process *and* the designs.

7.2.4 Choosing response distributions

In model pruning we compare the predicitve accuracy of models that differ by their structural part. With modern evaluation criteria, there is more we can do to arrive at an optimal model. The methods for model comparison introduced so far also allow to compare different response distributions, i.e. the shape of randomness. As we have seen in chapter 6.1, for most outcome variables, choosing an appropriate response distribution is a straight-forward choice, based on just a few properties (continuous versus discrete, boundaries and overdispersion).

In 6.3.2 the properties of ToT and RT data suggested that Gaussian error terms are inappropriate, because the error distribution is highly skewed. While Gamma error distributions can accomodate some skew, this may not be sufficient, because the lower boundary of measures is strongly positive. From an informal analysis of these three models, we concluded that the exgaussian is most suited for ToT and RT outcomes, followed by Gamma, whereas the Gaussian showed a very poor fit. However, visual analysis of residuals on non-Gaussian models is notoriously difficult, due to the variance-mean relationship. With information criteria, we can easily select the response distribution with the best predictive accuracy.

In 6.3.2 we found by informal analysis that Exgaussian regression may be suitable for RT and ToT responses. In the following this is put to a more formal test by using information criteria. This is one case, where *WAIC failed* to approximate predictive accuracy well, but to our convenience, the authors of package Loo provide a slightly less time efficient, but more robust estimate, the *LOO-IC*. One advantage of LOO-IC over WAIC is that it produces observation-level estimates, which can be checked individually. Often LOO-IC only fails on a few observations. The author of package Brms has added a practical fallback mechanism: by adding the argument `reloo = TRUE`, the problematic observations are refitted as real LOO-CVs.

```
attach(Hugme)
```

```
Loo_1_gau <- loo(M_1_gau, reloo = TRUE)
Loo_1_gam <- loo(M_1_gam)
Loo_1_exg <- loo(M_1_exg, reloo = TRUE)
```

The command `loo_compare` puts the results in one table and arranges the models by predictive accuracy. Unfortunately, it is not a tidy command, and a little extra tweaking is required to make it tidy. (`loo_compare` puts the models in row names, instead of using an explicit identifier variable). As expected, the Exgaussian error distribution makes for the best model.

```
loo_compare(Loo_1_gau, Loo_1_gam, Loo_1_exg) %>%
  as.data.frame() %>%
  rownames_to_column("Model") %>%
  select(Model, looic) %>%
  mutate(looic_diff = looic - min(looic))
```

Model	looic	looic_diff
M_1_exg	-12947	0
M_1_gam	-11941	1006
M_1_gau	-6479	6468

```
detach(Hugme)
```

The comparison confirms that the Exgaussian response distribution is best suited for these reaction time data. It may also be better suited for reaction time experiments in more general, but this needs more proof by other data sets. For the time being, experimentalists are advised to estimate several response distributions (especially Exgaussian and Gamma) and select the best fitting distribution using the described method.

The other class of duration measures is ToT data, which is routinely collected in usability tests and other applied design research studies. Formally, it should

have similar properties as RT data, but the underlying processes of data generation may still be very different. We investigate the CUE8 data using formal model selection on the same three response distributions.

In the previous analysis, the WAIC estimate self-reported unreliability, and we traded some computing efficiency for reliability by switching too LOO-IC. If LOO-IC is unreliable for a few observations, these can be fitted by running a few real LOO-CV with `reloo = T`. If more than just a few observations are unreliable, this can get time expensive. In the worst case, we would be thrown back to the very inefficient LOO-CV method. Fortunately, the authors of package Loo provide an alternative fall-back to reloo-ing: In *k-fold cross validation* a model is refit k times, always leaving out and predicting N/k observations. LOO cross validation actually is 1-fold cross validation, which results in fitting N models. 10-fold cross validation stands on middle ground by training the model ten times on a different 90% part of data and testing the model on the remaining 10%:

```
attach(CUE8)
```

```
F10_4_gau <- kfold(M_4_gau, K = 10)
F10_4_gam <- kfold(M_4_gam, K = 10)
F10_4_exg <- kfold(M_4_exg, K = 10)
```

```
loo_compare(F10_4_gau, F10_4_gam, F10_4_exg) %>%
  as.data.frame() %>%
  rownames_to_column("Model") %>%
  mutate(kfoldic = -2 * elpd_kfold) %>%
  select(Model, kfoldic) %>%
  mutate(kfoldic_diff = kfoldic - min(kfoldic))
```

Model	kfoldic	kfoldic_diff
M_4_gam	6587	0
M_4_exg	6765	178
M_4_gau	7159	572

```
detach(CUE8)
```

Interestingly, the Gamma response distributions is favored over the Exgaussian in this case for ToT. Again, this may or may not generalize to future data sets.

Obviously, predictive accuracy can also be used to check for over-dispersion, just compare, for instance, a Poisson model to a Negbinomial model. However, over-dispersion in count data is so common that model selection may be a waste of effort. In addition, should there really be no over-dispersion, the

Negbimomial model will behave almost exactly like a Poisson model and the costs (in terms of parsimony) are negligible.

Therefore, I close this section with a more interesting comparison: In section 6.5 *distributional models* were introduced as a generalization of GLMs, that allow to also link predictors to variance and other shape parameters, such as the Gaussian σ, the Beta ρ and the two parameters of the Exgaussian distribution, σ for dispersion and β for skew. A Beta model with two structural parts (mean and scale) was estimated and their appeared to be significant variance in how participants employed the scale, i.e. the range of responses. However, by individual-level scale parameters the model is massively inflated. Is that for the good of predictive accuracy? Below, we use the LOO-IC estimate to compare the distributional model against a regular Beta regression.

```
attach(Uncanny)
```

```
Loo_beta <- loo(M_poly_3_beta)
Loo_dist <- loo(M_poly_3_beta_dist)

loo_compare(Loo_beta,
            Loo_dist)
```

	elpd_diff	se_diff	elpd_loo	se_elpd_loo	p_loo	se_p_loo	looic	se_looic
M_poly_3_beta_dist	0	0.0	1870	54.5	126.7	4.40	-3740	109
M_poly_3_beta	-936	50.3	935	47.5	96.1	2.93	-1869	95

Despite its opulence, the distributional model out-ranks the simpler model. It confirms that individual response patterns occur and by granting all participants their own scale, we can expect to gain better predictive accuracy.

7.2.5 Testing theories

For everyone who has to publish in Social Science journals, this book has so far left one question unanswered: What is the Bayesian method for testing null hypotheses? To give the short answer: There are no p-values in this statistical framework, but you can always use credibility limits to say "the effect is different from Zero with a certainty larger than 95%". If some pertinent reviewers want p-values, you can argue that a null-hypothesis test is really just model selection, but unfortunately no p-value tests are available for the models that you are using, and then you introduce one of the methods based on predictive accuracy.

But, should you really do null hypothesis testing in design research? Well, if your research implies or even directly targets real costs, benefits and risks for real people, you should continue with quantifying impact factors. That can

even go so far that an unnecessary predictor is left in the model, just to make its irrelevance transparent. Everyone whose final report draws any conclusion of how the results can be used for the better, must speak of quantities at some place.

Still, it seems to make sense to test theories by comparing predictive accuracy of models. Recall the uncanny valley phenomenon. In chapter 4.6 the non-linear relationship between human likeness of robot faces and emotional response was modeled as a 3-deg polynomial, at first, and we have reported a bunch of quantitative results, such as the position of the trough. We encountered the case again for an examination of goodness-of-fit 7.1.2. Our plots of fitted responses suggested that the cubic polynomial fitted than the simpler quadratic. A quadratic function produces a parabole and can therefore also can have a local minimum. But it cannot have the shoulder that makes the effect so abrupt. This abruptness is what has fascinated so many researchers, but is it really true?

```
attach(Uncanny)
```

```
options(mc.cores = 1)
Loo_poly_2 <- loo(M_poly_2)
Loo_poly_3 <- loo(M_poly_3)
```

```
loo_compare(
  Loo_poly_2,
  Loo_poly_3)
```

	elpd_diff	se_diff	elpd_loo	se_elpd_loo	p_loo	se_p_loo	looic	se_looic
M_poly_3	0.00	0.0	88.1	5.56	4.40	0.734	-176	11.1
M_poly_2	-9.43	4.2	78.7	5.24	3.62	0.623	-157	10.5

The cubic polynomial has the better predictive accuracy. It suggests that the Uncanny Valley effect is due to a disruptive cognitive change, like when we suddenly become aware that we have been fooled. This is not a direct test of a given theory, but it favors all theories that contain an abrupt process. One of the theories that does not contain a disruptive process goes that the UV effect is caused by religious beliefs. Another theory explains the effect as a the startling when our mind suddenly becomes aware of a deception and re-adjusts.

```
detach(Uncanny)
```

8

Reflections

A

Cases

This book comes with eight research cases with real data and eight simulated cases. They will be provided as R environments saved as Rda files. To use these environments, you have to download the Rda file and load it:

```
load("Cases/Uncanny.Rda")
```

The loading puts the environment into your R session and you can see the content using the Environment tab in Rstudio. Or you issue the `ls` command:

```
ls(Uncanny)
```

```
##  [1] "D_UV"               "DK_1"               "Loo_beta"
##  [4] "Loo_dist"           "Loo_poly_2"         "Loo_poly_3"
##  [7] "M_dsgmx_1"          "M_poly_2"           "M_poly_3"
## [10] "M_poly_3_beta"      "M_poly_3_beta_dist" "M_poly_3_ml"
## [13] "M_poly_9"           "P_poly_3"           "P_poly_3_ml"
## [16] "PP_poly_3_ml"       "PS_1"               "RK_1"
## [19] "RK_2"               "T_poly_3_ml"        "T_univ_uncanny"
## [22] "UV_1"               "UV_dsgmx"
```

In order to use the environment content in your R session, you have to attach it.

```
attach(Uncanny)
```

Real data cases contain one or more data sets as dataframes (tibbles), such as:

```
UV_1
```

Before switching to a different case environment, it is recommended to detach the present environment:

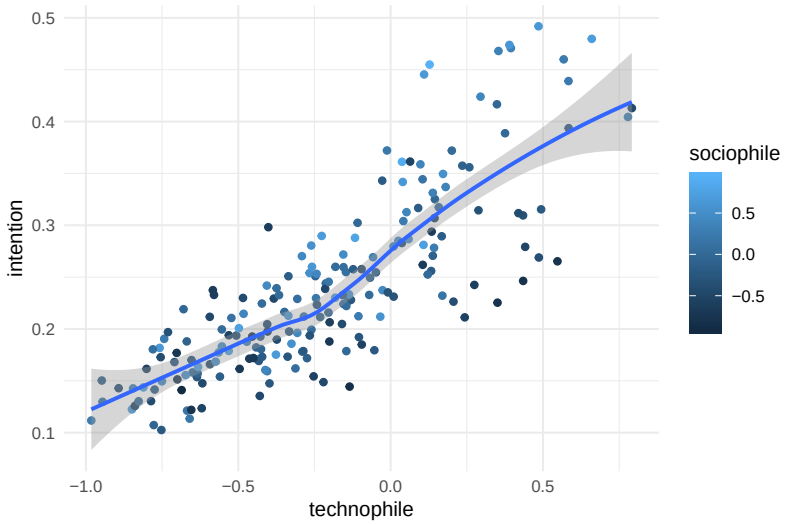
```
detach(Uncanny)
```

Synthetic data cases contain a simulation function that precisely produces the data set as it has been used in this book. Sometimes, the simulation function also provides additional arguments to make changes to the data set.

```
load("Cases/AR_game.Rda")
attach(AR_game)
```

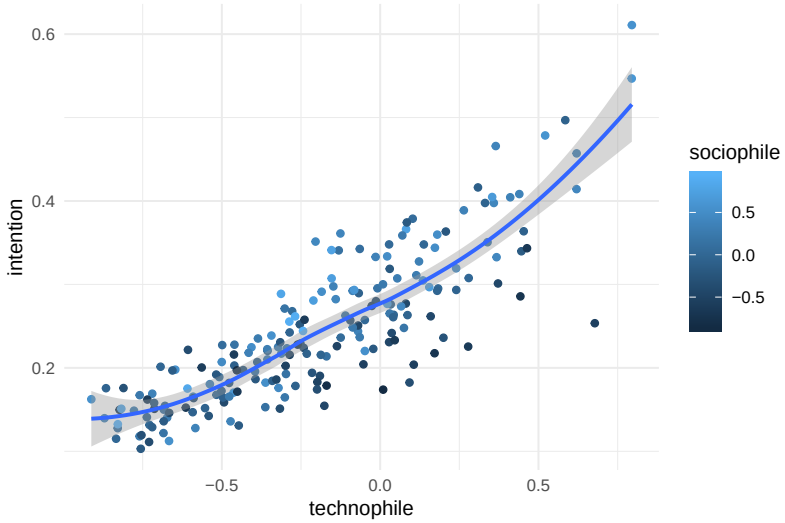
The following simulates the AR_game data set, exactly as it was used in section [amplification].

```
simulate() %>%
  ggplot(aes(x = technophile, color = sociophile, y = intention)) +
  geom_point() +
  geom_smooth()
```



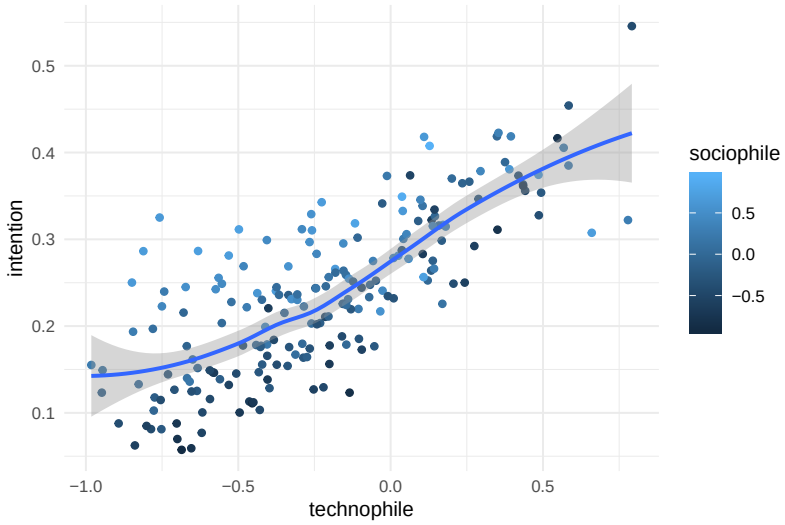
All simulation functions provide the argument `seed`, which sets the random number generator(s) to a specific value. Using a different seed value produces a data set with the same structure, but different values.

```
simulate(seed = 1317) %>%
  ggplot(aes(x = technophile, color = sociophile, y = intention)) +
  geom_point() +
  geom_smooth()
```



Additional arguments can be used to change the structure of the data set. In the present example, the amplification effect can be turned into a [saturation] effect, by changing the beta argument:

```
simulate(beta = c(-1, 1, .4, -1)) %>%
  ggplot(aes(x = technophile, color = sociophile, y = intention)) +
  geom_point() +
  geom_smooth()
```



If you want to understand how a simulation function works and how it can be controlled, you can display the code of the function, just by calling it without parentheses:

```
simulate
```

```
## function(N = 200,
##            beta = c(-1, 1, .4, .6),
##            sigma = .2,
##            seed = 42)
##     {
##     set.seed(seed)
##     out <-
##       tibble(Part = 1:N,
##                  technophile = rbeta(N, 2, 3) * 2 - 1,
##                  sociophile = rbeta(N, 2, 2) * 2 - 1) %>%
##       mutate( eta = beta[1] +
##                 beta[2] * technophile +
##                 beta[3] * sociophile +
##                 beta[4] * technophile * sociophile,
##               intention = mascutils::inv_logit(rnorm(N,  eta, sigma))) %>%
##       as_tbl_obs()
##
##     class(out) <- append(class(out), "sim_tbl")
##     attr(out, "coef") <- list(beta = beta,
##                               sigma = sigma)
##     attr(out, "seed") <- seed
```

```
##
##     out %>% as_tbl_obs()
##     }
## <bytecode: 0x000000004b903b68>
```

Real and synthetic case environments provide all data used in this book, but also all models are included that have been estimated. When working through this book, this saves you the effort to run the models by yourself.

```
M_cmrm
```

```
## stan_glm
##  family:       gaussian [identity]
##  formula:      intention ~ 1 + sociophile + technophile + sociophile:technophile
##  observations: 40
##  predictors:   4
## ------
##                        Median MAD_SD
## (Intercept)            0.3    0.0
## sociophile             0.1    0.0
## technophile            0.2    0.0
## sociophile:technophile 0.2    0.0
##
## Auxiliary parameter(s):
##       Median MAD_SD
## sigma 0.0    0.0
##
## ------
## * For help interpreting the printed output see ?print.stanreg
## * For info on the priors used see ?prior_summary.stanreg
```

```
detach(AR_game)
```

A.1 Real cases

A.1.1 CUE8

CUE8 is one of the long series *Comparative Usability Evaluation* studies conducted by Rolf Molich CUE8. Previous studies had shown that usability experts differ a lot in identification and reporting of usability problems. In CUE8, Molich and colleagues put the bar much lower and asked, whether different professional teams would obtain consistent measures of time-on-task. Eight

independent Teams were given the same test scenario, consisting of five user tasks on a car rental website. Teams were otherwise free to design the test. In particular, some teams conducted remote usability tests, whereas others did standard moderated testing.

A.1.1.1 Measures

Time-on-task was measured in seconds and are also provided on a logarithmic scale, because of violation of Gaussian distribution of errors. In addition, satisfaction has been measured using the Systems Usability Scale with the range of $[0, 100]$.

```
load("Cases/CUE8.Rda")
attach(CUE8)
D_cue8
detach(CUE8)
```

A.1.2 Uncanny Valley

The Uncanny Valley effect is an astonishing emotional anomaly in the cognitive processing of artificial faces, like ... robot faces. Intuitively, one would assume that people would always prefer faces that are more human-like. As it turns out, this is only up to a certain point, where increasing human-likeness causes a feeling of eerie.

For the first time, the UV effect could be rendered in an experiment by Mathur & Reichling. They measured the emotional response to pictures of robot faces using a simple rating scale (Likeability).

A.1.2.1 Experimental design

In order to pursue deeper into the UV effect, two more experiments, PS and RK, have been conducted in our lab. Both experiments have in common that participants see a sequence of robot faces and give an emotional response. Experiments DK and PS aimed at identifying the cognitive level of processing that makes the UV occur and collected the responses under manipulation of presentation time. The presentation times were 50, 100, 200 and 2000ms. Experiment RK kept the presentation time constant at 2000ms, but presented all stimuli three times. This was to collect enough data for verifying that the UV phenomenon is universal, i.e. occurs for every participant. All three experiments used a full within-subject design. However, every trial presented just one item from the Eeriness scale.

A.1.2.2 Stimuli

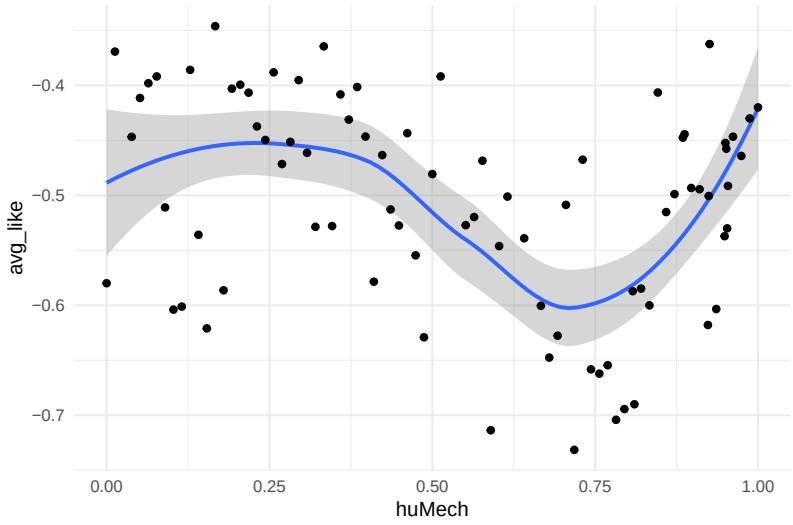
The stimuli in the experiments are pictures of robot faces that vary from totally not human-like (like the casing of a robotic vacuum cleaner) to almost indistinguishable from human. Mathur & Reichling collected and tagged the major part of the set of stimuli. The two studies, PS [PS] and RK [RK], successively added stimuli to the set in order to increase the precision in the range where the effect occurs.

The central predictor in these experiments is the human-likeness of a robot face. Mathur & Reichling produced these measures by use of a rating scale, humano-mechanical scale (huMech). Stimuli that were added by PS and RK were rated by two experts, using MR collection as a a baseline. Variable huMech has been normalized to $[0; 1]$.

A.1.2.3 Measures

The Eeriness scale of Ho & MacDorman [REF] has been used to measure the emotional response. This scale contains eight items and has specifically been designed to observe the UV effect. The scale was implemented as a visual analog scale. Because the Eeriness scales direction is reverse to the original Likeability scale of Mathur & Reichling, responses have been reversed (negative Eeriness) and normalized to the interval $[-1; 0]$. In addition, reaction times have been recorded.

```
load("Cases/Uncanny.Rda")
attach(Uncanny)
Uncanny$RK_2 %>%
  ggplot(aes(x = huMech, y = avg_like)) +
  geom_smooth() +
  geom_point()
```




```
detach(Uncanny)
```

A.1.3 IPump

Medical infusion pumps are unsuspicious looking devices that are en-mass installed in surgery and intensive care. Their only purpose is controlled injection of medication in the blood stream of patients. Pumps are rather simple devices as infusion is not more than a function of volume and time. They are routinely used by trained staff, mostly anaesthesiologists and nurses. We should have great faith in safe operation under such conditions. The truth is, medical infusion pumps have reportedly killed dozens of people, thousands were harmed and an unknown number of nurses lost their jobs. The past generation of pumps is cursed with a chilling set of completely unnecessary design no-gos:

- tiny 3-row LCD displays
- flimsy foil buttons without haptic marking or feedback
- modes
- information hidden in menus

For fixing these issues no additional research is needed, as the problems are pretty obvious to experienced user interface designers. What needs to be done, though, is proper validation testing of existing and novel interfaces, for example:

- is the interface safe to use?

- is it efficient to learn?
- is a novel interface better than a legacy design? And by how much?

We conducted such a study. A novel interface was developed after an extensive study of user requirements and design guidelines. As even the newest international standards for medical devices do not spell precise quantitative user requirements (such as, a nurse must be able to complete a standard task in t seconds and no more than e errors may occur), the novel interface was compared to a device with a legacy design.

A.1.3.1 Experimental design

Eight successive user tasks were identified from training material and documentation. All participants were trained nurses and they were asked to complete the series of tasks with the devices. In order to capture learnability of the devices, every nurse completed the sequence of tasks in three consecutive sessions.

A.1.3.2 Measures

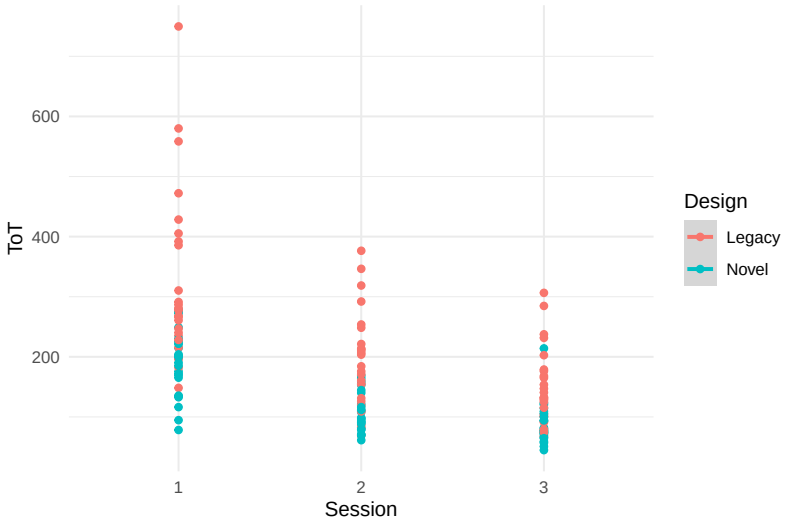
A number of performance measures were recorded to reflect safety and efficiency of operation:

1. *task completion*: for every task it was assessed whether the nurse had completed it successfully.
2. *deviations from optimal path*: using the device manual for every task the shortest sequence was identified that would successfully complete the task. The sequence was then broken down into individual operations that were compared to the observed sequence of operations. An algorithm called *Levenshtein distance* was used to count the number of deviations.
3. *time on task* was recorded as a measure for efficiency.
4. *mental workload* was recorded using a one-item rating scale.

Furthermore, several participant-level variables have been recorded:

1. professional group: general or intensive care
2. level of education (Dutch system): MBO, HBO and university
3. years of job experience as a nurse

```
load("Cases/IPump.Rda")
attach(IPump)
IPump$D_agg %>%
  ggplot(aes(x = Session, color = Design, y = ToT)) +
  geom_smooth() +
  geom_point()
```



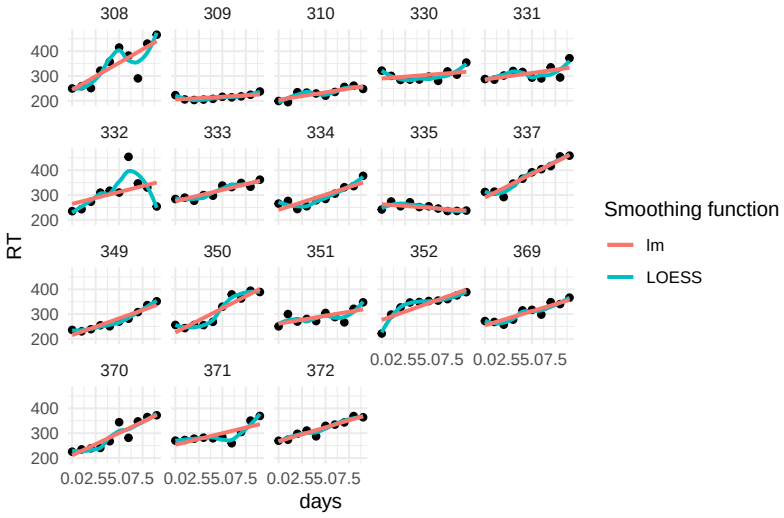

```
detach(IPump)
```

A.1.4 Case Sleepstudy

This data set ships with the lme4 package and has only been converted to the coding standards used throughout. Eighteen participants underwent sleep deprivation on ten successive days and the average reaction time on a set of tests has been recorded per day and participant. For further information on the data set, consult the documentation (`?lme4::sleepstudy`). Variable names have been changed to fit the naming scheme of this book.

```
load("Cases/Sleepstudy.Rda")
attach(Sleepstudy)
D_slpstd %>%
  ggplot(aes(x = days, y = RT)) +
  facet_wrap(~Part) +
  geom_point() +
  geom_smooth(se = F, aes(color = "LOESS")) +
```

```
  geom_smooth(se = F, method = "lm", aes(color = "lm")) +
  labs(color = "Smoothing function")
```




```
detach(Sleepstudy)
```

A.1.5 Egan

in the beginning 1990ies Dennis Egan examined the variability in performance that is due to individual differences Egan. He concluded that individual differences are the greater source of performance variance than design differences are. Twenty-five years we put that to a test [test_Egan].

A.1.5.1 Research Design

We selected ten university websites and identified ten typical information search tasks. 41 student users took part and performed the tasks. Our design is a full within-subject design, with *planned missing values*. Instead of all 100 possible combinations of websites and tasks, every participant got a set of trials, where websites and tasks were paired, such that every website and every task appeared *exactly once* per participant.

A.1.5.2 Measures

Four usability measures were taken per trial:

- task success
- number of clicks (clicks)
- number of tims user returns to homepage
- workload (measured by a one-item scale)

```
load("Cases/Egan.Rda")
attach(Egan)
Egan$D_egan
detach(Egan)
```

A.1.6 Case: Millers Magic Number

Miller's magic number says that the short term memory capacity is 7 ± 2. Later research by Baddeley & Hitch found that the so-called *working memory* is a multi-component system, that stores visual and verbal information separately. Following up on the experimental research by [Freudenthal], we were interested how differences in capacity of the verbal and visual subsystem explain performance differences in a real web search tasks.

A.1.6.1 Research Design

We selected five municipal websites and five search tasks. For creating the trials, websites and tasks were paired in ush c way, that all participants see every website and every task exactly once.

A.1.6.2 Measures

For visual working memory capacity, we used the Corsi block tapping task. For verbal capacity, the Ospan task was used and both scoring schemes, A and B, were applied. Number of clicks and time to completion were recorded as performance measures.

```
load("Cases/MMN.Rda")
attach(MMN)
MMN_2
detach(MMN)
```

A.1.7 AUP

In their seminal article, [Carroll & Rosson] coin the term *Active User Paradox* for their observations, that users stick to their habits and are reluctant to put energy into learning a computer system. This seems irrational at first, as users miss a chance to increase their long-term efficiency. (Later research by [Fu & Gray] found that their is a hidden rationality to the AUP.) Still, there are users out there who are enthusiastic about computers and love to solve the intellectual puzzles they provide. We wanted to see whether people of that kind would be more likely to over-win the AUP.

A.1.7.1 Measures

For measuring the personality of users, we used

- the *need-for-cognition scale*

- the *Geekism (gex)* scale.
- a scale for *Computer Anxiety Scale*
- a scale for *Utilitarianism*

Users were given two complex tasks on a computer. Their behavour was observed and events were coded, e.g. “User uses documentation of the system”. Events were than rated, counted and aggregated into two scales:

- seeking challenges
- explorative behaviour

By combining these two scales, we created a total score for *resistance to the AUP*, per participant.

```
load("Cases/AUP.Rda")
attach(AUP)
MMN$AUP_1
```

```
## NULL
```

```
detach(AUP)
```

A.1.8 Hugme

As swift and enthusiastic most people are with words, the more clumsy and desinterested many are with computers. Today's consumer computing devices are unmatched in usability and most people do not need to think harder than they like, when hunting for the best online prices, stream movies and enjoy their communications. However, there is a minority of computer users, who call themselves the geeks. The hypothetical geek personality feels attracted to the inner workings of a computer system itself, rather than the applications and media entertainment it delivers. A geek person seeing a computer is more likely to have certain memories or associations. For example, remembering how it was to build your first own computer, or the intellectual joy of learning a new programming language. If this were true, we thought, then showing a word that is related to how geeks perceive computers (e.g. "explore", "play", "create") should create a brief nostalgic moment, resulting in a delayed response.

A.1.8.1 Measures

Before the experiment, participants filled out teh multi-item NCS and Geekism questionnaires. The geekism questionknaire was given a second time after the experiment, originally to assess test-retest reliability.

As response we had chosen recation time in a Stroop task, where participants were first primed by a picture shown before the Stroop task. These pictures were from two conditions: either showing computers in a geek-ish way (for example, an open case or programming code on the screen) or as regular product images. Furthermore, we presumed that geeks like to think hard and used the need-for-cognition scale as a second predictor. It was expected that participants with high NCS scores would recognize computers as a source of delightful hard thinking and hence have slower reaction times, when priming image and target word are both from the category Geek.

A.1.8.2 Stimuli

Stimuli were composed of a priming picture and the word presented in the Stroop task, with prime/word pairs generated randomly for each observation.

Ninety words were selected, 30 for each of the following three categories and translated into English, German and Dutch:

1. hedonism (stylish, braging, wanting, ...)
2. utilitarianism (useful, assisting, optimizing)

3. geekism (configuring, technology, exploring)

Seventy-two prime pictures were selected, with 24 for each of the following categories:

1. control (pictures unrelated to technology)
2. geek (pictures showing computers in a geekish setting, e.g. an open computer case or a terminal console)

A.1.8.3 Data modelling in a nutshell

The Hugme experiment is a multi-population study: participants, pictures, words and items (from two scales) all provide information that is relevant for the statistical model. For analyzing the relationship between geekism, NCS on te one hand and the difference in reaction times between word and picture categories, we need the table `D_hugme`, which provides the predictor data for every observation from the experiment. The way this table was constructed is an instructive example of how researchers can logically structure and efficiently transform data from complex studies.

The Table `R_exp` is the raw data that we got from the experiment. It contains the encounter of participants with primes and words, but misses the classification of words and primes, as well as participant-level variables. The classification of words and primes is stored separately in three *entity tables*: `E_Words`, `E_Primes`, whereas a `E_part` contains demographic data:

```
load("Cases/Hugme.Rda")
attach(Hugme)
```

```
E_Part %>% sample_n(8)
```

Part	gender	age
18	female	23
16	male	54
2	male	27
60	male	23
64	female	23
57	male	24
3	female	24
50	male	24

```
E_Words %>% sample_n(8)
```

Word	Word_DE	Word_NL	WordCat
rebuild	umbauen	wijzigen	geek
effective	effektiv	effectief	util
manageable	handlich	handig	util
commanding	beherrschen	beheersen	geek
modifying	verändern	veranderen	geek
usable	brauchbar	bruikbaar	util
serving	bedienen	besturen	util
planning	planen	plannen	util

```
E_Primes %>% sample_n(8)
```

All entity tables capture the information in the sample of precisely one population, with one row per member and a unique identifier. This identifier is called a *key* and is crucial for putting all the information together, using *joins*. Any response measure, like RT, is something that happens at every encounter. More abstractly, we could say, that this encounter is a *relation* between participants, words and primes, and the response is an *attribute* of the relation. `R_exp` is a relationship table which stores responses together with the encounter.

In fact, the encounters between Part, Word and Prime can be called a key, because the combination truly identifies every observation:

```
R_exp %>% as_tbl_obs()

cat("The number of unique encounters is",
    dim(distinct(R_exp, Part, Word, Prime))[1])
```

```
## The number of unique encounters is 4027
```

What is still missing in the table are all the variables that further describe or classify members of any of the three samples. Think of the püredictors you need for running group-means analysis or linear regression on the data. Table `R_exp` provides the scaffold and the other variables we can pull in from the entity tables by using *join* operations. A join always operates on two tables that share at least one key variable. The following code takes the `R_exp` the relationship table to the left and merges in additional data from the entity table, picking on the key variable. This works successively on the three entity tables:

```
R_exp %>%
  left_join(E_Words, by = "Word") %>%
  left_join(E_Primes, by = "Prime") %>%
  left_join(E_Part, by = "Part")
```

```
detach(Hugme)
```

Now you know how Entity Relationship Modelling works. This paradigm came up with the second generation of data base systems in the 1990s (The first generation used strictly hierarchical data structures) and has been fully adopted by the packages Dplyr and Tidyr from the Tidyverse.

A.2 Synthetic data sets

A.2.1 Rainfall

We don't want to get wet, which is why we use the weather forecast. If we want to make our own forecasts, we need data. This simulation records rainfall on 20 days. On around 60% of these days, the sky was cloudy in the morning. With a blue sky in the morning, the chance of rain is 30%. On cloudy days it is 60%.

A.2.2 99 seconds

The maerketing department of a car rental website claims that "You can rent a car in 99 seconds." In this simulation, time-on-task measures are taken from 100 test users. These are Gaussian distributed with a mean of 105 and a standard error of 42. ToT also correlates with age, while there is no gender difference.

```
load("Cases/Sec99.Rda")
attach(Sec99)
Ver20 %>%
  ggplot(aes(x = age, y = ToT)) +
  geom_point()
```

```
detach(Sec99)
```

A.2.3 Rational

A project needs a predicted increase in revenue by a factor of 1.1 for management to give green light. The simulation produces a between-subject comparison study, where 50 users each see the current version of the website or the prototype. Revenue in both groups is Gamma distributed with a mean of 100 for the current version and a multiplyer of 1.1 for the prototype.

A.2.4 BrowsingAB

A.2.5 Headache

This simulation takes perceived headache measured on 16 participants before and after an administration of headache pills. Participants either get both pills A and B, only A, only B or no pill (placebo). Pill A and B are both effective on their own, but there is a saturation effect, when both pills are taken. Baseline headache is generated from Beta distributions. In order to avoid unplausible (i.e. negative) values, reduction in headache involves a log transformation.

```
load("Cases/Headache.Rda")
attach(Headache)
simulate() %>%
  ggplot(aes(x = PillA, color = PillB, y = reduction)) +
  geom_boxplot()
```

```
detach(Headache)
```

A.2.6 Reading time

This simulation covers an experiment where participants got to read a text on screen and their reading time is recorded. 40 participants are divided over four experimental conditions, where the font size is either 10pt or 12pt and where the font color is either black (high contrast) or gray. Small and gray font results in an average reading time of 60 seconds. 12pt is read 12s faster and black font is read 10s faster. Due to a saturation effect, 12pt and black combined do not result in 22s, but only 14s. Reading time (ToT) is generated with Gaussian distribution.

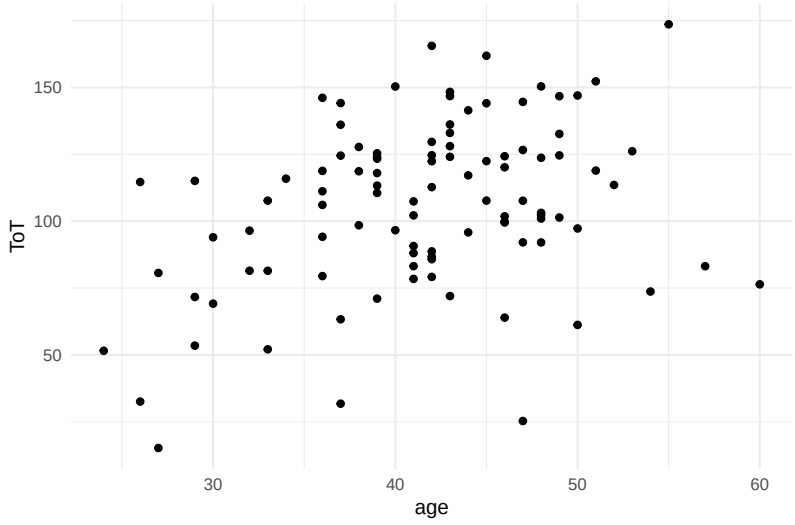
```
load("Cases/Reading.Rda")
attach(Reading)
simulate() %>%
  ggplot(aes(col = font_color,
             x = font_size,
             y = ToT)) +
  geom_boxplot()
```

```
detach(Reading)
```

A.2.7 AR_game

A company seeks their customer profile for a novel Augmented Reality game. 200 participants rate how technophile or sociophile they are (generated from Beta distributions) and rate their intention to buy the product. The coefficients are set to create a slight benefit (for intention) of being sociophile or technophile and an amplification effect for participants that are both. Intention is sampled from a Gaussian distribution, but with an inverse logit transformation to create boundaries at $[0; 1]$.

```
load("Cases/AR_game.Rda")
attach(AR_game)
simulate() %>%
  mutate(technophile_grp = technophile > median(technophile),
         sociophile_grp = sociophile > median(sociophile)) %>%
  ggplot(aes(x = sociophile_grp,
             color = technophile_grp,
             y = intention)) +
  geom_boxplot()
```




```
detach(AR_game)
```

A.2.8 Sleep

This simulation is loosely modelled after an experiment Corcoran [D. W. J. Corcoran (1962) Noise and loss of sleep, Quarterly Journal of Experimental Psychology, 14:3, 178-182, DOI: 10.1080/17470216208416533] who measured the combined effects of sleep deprivation and noisy environments. It turned out that noise and sleep deprivation both increase reaction times, but that noise helps when someone is very tired. Outcomes were simulated from Gaussian distributions.

These results can be explained by the Yerkes-Dodson law, which states that performance on cognitive tasks is best under moderate arousal. It is assumed that arousal increases energy, but also causes loss of focus. These two counteracting forces reach an optimal point somewhere in between. The two lines Energy and Focus have been produced by a logistic function, whereas Performance is the product of the two.

```
load("Cases/Sleep.Rda")
attach(Sleep)
simulate() %>%
  ggplot(aes(x = Environment,
             color = Sleep,
             y = RT)) +
  geom_boxplot()
```

```
detach(Sleep)
```