Martin Schmettow

New statistics for the design researcher

A Bayesian course in tidy R

2018-11-02

Springer

Contents

Part I Preparations

1 Introduction 9
- 1.1 Whom this book is for 9
- 1.2 Assumptions 11
- 1.3 How to read this book 12

2 Elements of quantitative design research 15
- 2.1 Studies 19
- 2.2 Observations and measures[TBC] 19
- 2.3 Decision making under uncertainty 22

3 Elements of Bayesian statistics 39
- 3.1 Descriptive statistics 40
- 3.2 Bayesian Inferential statistics 56
- 3.3 Bayesian probability theory 58
- 3.4 Statistical models 68
- 3.5 Bayesian estimation 91
- 3.6 What is wrong with classic statistics? [TBC] 94

4 Getting started with R 95
- 4.1 Setting up the R environment 96
- 4.2 Learning R: a primer 104

Part II Models

5 Linear models 151
5.1 Quantification at work: grand mean models 151
5.2 Walk the line: linear regression 163
5.3 A versus B: Comparison of groups 178
5.4 Putting it all together: multi predictor models 187
5.5 Interaction effects 215
5.6 Doing the rollercoaster: polynomial regression 243

6 Multilevel models 251
6.1 Average? Neverage! 252
6.2 The Human Factor: Intercept random effects 256
6.3 Designs and other non-human populations 264
6.4 Crossover: Testing Egans assumption 266
6.5 Measuring entities: psychometric applications [TBC] 272
6.6 Nested random effects 275
6.7 What are random effects? On pooling and shrinkage 280
6.8 Variance in variation: slope random effects [TBC] 286
6.9 Growing with random effects correlations [TBD] 294
6.10 Advanced tricks 294
6.11 Exercises 295

7 Generalized Linear Models 297
7.1 Debunking the Gaussian linear model 299
7.2 Elements of Generalized Linear Models 309
7.3 Case: user testing infusion pumps 313
7.4 Count data 315
7.5 Measures of time 338
7.6 Rating scales 362

Part III Workflow

8 Data management 383
8.1 Modelling questionnaire responses 386
8.2 Modelling experimental responses 389
8.3 Modelling design studies (tbd) 394
8.4 Modelling organizational structures (tbd) 395
8.5 Modelling longitudinal data (tbd) 395
8.6 Doing it in R 395

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 Assumptions

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 @ref(design_research) 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 @ref(bayesian_statistics) 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 @ref(getting_started_r) 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 @ref(linear_models) 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 @ref(multilevel_models). 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 @ref(generalized_linear:models) 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.

The fourth chapter @ref(nonlinear_models) re-iterates on the truth that in an endless universe everything is finite. We leave our fond of high-level regression engine alone and take a closer look at the nuts and bolts it is based upon. By the example of a logistic growth process and efefcts of training, the reader gains a preview on the almighty model specification language Stan.

The third part of the book is primarily dedicated to researchers who deal with complex and high-stake problems, routinely. Elememts of the statistical workflow are spread out in much detail, covering data manipulation, advanced graphics, simulation and model evaluation.

2

Elements of quantitative design research

A *design* is the structure of an artifact or process that people use for a purpose. This definition appears extremely broad as it covers everything from tea kettles to complex information systems, and even silver bullets are included if you wish so. 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. 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.

2.0.0.0.1 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, 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 @ref(bayesian_statistics)).

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 @ref(interaction_effects): 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 @ref(decision_making) 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 @ref(interaction_effects), 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.

2.0.0.0.2 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

2.1 Studies

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

2.2 Observations and measures[TBC]

2.2.1 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.

2.2.2 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 [LMM] 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

2.2.2.1 –>*Satisfaction*

2.2.3 experience [TBD]

2.2.4 design features [TBD]

2.2.5 the human factor [TBD]

2.2.6 situations [TBD]

2.3 Decision making under uncertainty

- 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. 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 unlucky one, who's bungee rope breaks, with fatal consequences.
- You don't know whether the new design will attracts more visitors to your website and serves them better.

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 acknowledgements) 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 recurrance rate of, say, 50% worth?

Once you know the probabilities of events and the respective values, *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* L. As such, it is the average loss one can expect, when following a particular strategy.

In the given case, two strategies exist, namely taking an umbrella versus taking no umbrella, when it is cloudy. We calculate and compare the expected utilities `U` as follows:

$$
\begin{aligned}
P(rain) &= 0.6 \\
P(norain) &= 1 - P(rain) = 0.4 \\
L(carry) &= 2 \\
L(wet) &= 4 \\
U(umbrella) = P(rain)L(carry) + P(norain)L(carry) &= L(carry) = 2 \\
U(noumbrella) &= P(rain)L(wet) = 2.4
\end{aligned}
$$

```
attach(Rainfall)

Actions <-
  data.frame(action = c("umbrella", "no umbrella"))

Events  <-
  data.frame(event  = c("rain", "no rain"),
                      prob   = c(0.6, 0.4))

Loss  <-
  expand.grid(action = Actions$action, event = Events$event) %>%
  join(Events) %>%
  mutate(loss = c(2, 4, 2, 0))

Decision <-
  Loss %>%
  mutate(conditional_loss = prob * loss) %>%
  group_by(action) %>%
  summarise(expected_loss = sum(conditional_loss))

Decision
```

action	expected_loss
no umbrella	2.4
umbrella	2.0

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.

2.3.1 Measuring uncertainty

As we have seen above, a decision requires two investigations: the indeterministic nature of outcomes, and the assigned loss. Assigning loss to decisions is highly context dependent and often requires domain-specific expertize (or even individual judgment). We will do the formalization of it one more time,

then set it aside. The issues of indeterministic processes and the associated uncertainty is basically what the field of *statistics* deals with. 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 wheather 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 wheather), 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 opportunty to average out a single failure at future occasions. In contrast, the investor, who lends you the money for your endeavour, 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 embebedded 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) 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 t o know the chance of rain exactly, which is unrealistic. Estimating parameters from observations is the reign of statistics. From naive calculöations , 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.

Maybe, you have recorded the coincidences of clouds and rainfall over a period, of, let's say, 20 days. You made the following observations:

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

Obs	cloudy	rain
1	TRUE	FALSE
5	TRUE	TRUE
6	FALSE	FALSE
9	FALSE	TRUE
10	TRUE	TRUE
14	TRUE	TRUE
18	FALSE	FALSE
19	TRUE	FALSE

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))
```

```
detach(Rainfall)
```

These probabilities we can feed into the decision framework as outlined above. The problem is, that we obtained just a few obervations 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, $P(rain|cloudy) = 60.05\%$.

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 uncer-

tainty, which involves to *measure the level of certainty* associated with a certain statement. 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: In your experience, 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 and70%.
>
> 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.

2.3.2 Betting on 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 broad (and shallow) or deep (and narrow). Your choice will to some extent, change usability of the website, hence your customer satisfaction, rate of recurrance, revenue etc.

2.3.3 Case: e-commerce A/B

Consider the following example: Violet is development manager of a e-commerce website. At present, there is debate about a major overhaul of the website. Most management team members agree that implementing a better search function will increase the revenue per existing customer. At the opposite, there are considerable development costs involved, which you have estimated to EUR 100.000.

Upper management has a keen eye on shareholder value and wants the costs to be covered by increased revenues within one year. As the company's total revenue per year is 1.000.000 Euro, an increase by factor 1.1 is required to cover the development costs.

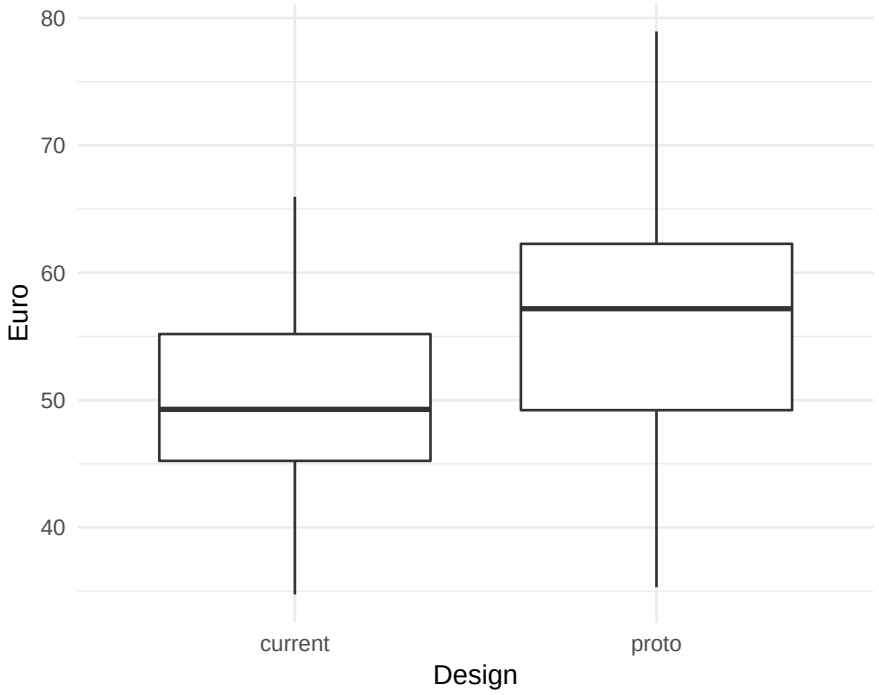
In order to get a rough idea whether this target can be met, Violet conducts a study where she observes the transactions of 50 random customers using the current system with 50 transactions with a prototype of the new system. The measured variable is the money every participant spends during the visit. The question is:

"Do customers in the prototype condition spend 10% more on average?"

The figure below shows the distribution of revenue in the experiment.

```
attach(Rational)
```

```
RD  %>%
  ggplot(aes(y = Euro, x = Design)) +
  geom_boxplot()
```



There seems to be a slight benefit for the prototype condition. But, is it a 10% increase? And how certain can Violet be? A formal regression analysis gives the answer [1].

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

P_1 <-  posterior(M_1)
```

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

fixef	center	lower	upper
Intercept	5.00e+01	4.70e+01	5.30e+01
Designproto	1.13e+00	1.03e+00	1.23e+00

[1] What we run here is a Gamma regression, which we will re-encounter in chapter [GLM]. The coefficients can directly be interpreted as changes in rate.

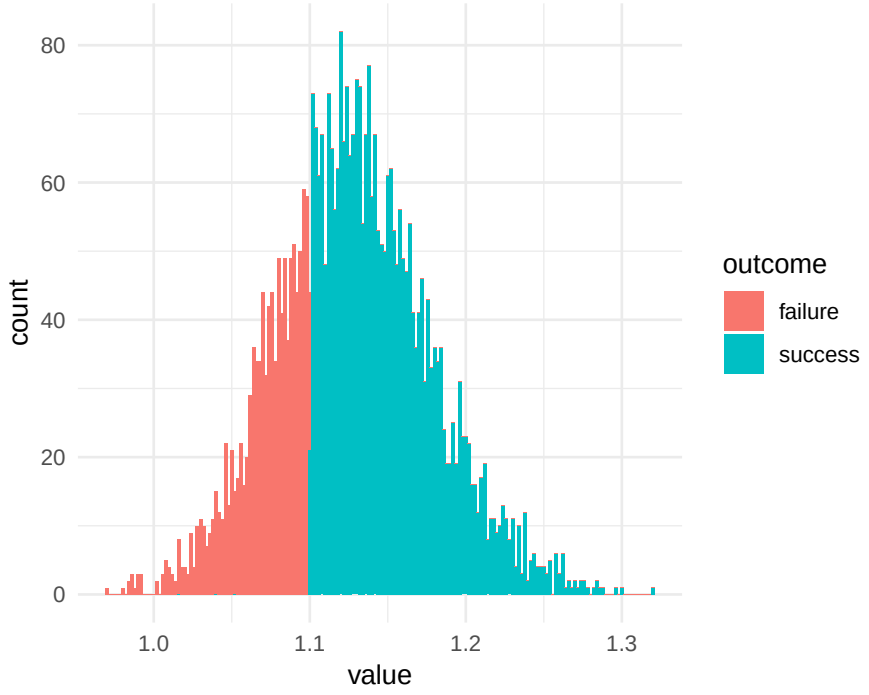
fixef	center	lower	upper
shape	1.77e+09	8.00e+06	1.13e+12

The results tell her that participants spend 49.975 Euro on average with the current design. The prototype seems to increase the revenue per transaction by 1.126. That seems sufficient, but uncertainty is strong, too, as indicated by the 95% credibility intervals. There is a chance of more than 2.5%, that the revenue is just $47.035, 1.035, 8.003 \times 10^6$ (or lower), such that the target is not met. But what precisely is the chance of falling below EUR 5? In Bayesian analysis we usually get the complete posterior distribution, which we can evaluate to get a definite answer:

What is the probability of the increase in revenue being less than 5 EUR?

We inspect the posterior distribution and mark the area of interest

```
N_risk_of_failure_1 <-
  P_1 %>%
  filter(parameter == "Designproto") %>%
  mutate(value = exp(value),
         outcome = value < 1.1) %>%
  select(outcome) %>%
  colMeans

P_1 %>%
  filter(parameter == "Designproto") %>%
  mutate(value = exp(value),
         outcome = ifelse(value < 1.1, "failure", "success")) %>%
  ggplot(aes(x = value, fill = outcome)) +
  geom_histogram(binwidth = .002)
```




```
detach(Rational)
```

The red area represents the danger zone, where revenue increase does not reach EUR 5. The area amounts to 28.65% of the total probability. With this information the manager now has several options:

1) decides that 28.65% is a risk she *dares* to take
2) be confident in herself because *past successes* prove that she should follow her intuition
3) take into account other *sources of evidence*, for example expert opinions or meta studies
4) continue the study until sufficient evidence is gathered, but costs of additional effort have to be accounted for

The first option represents an unspecific tolerance towards risk. Extreme risk tolerance of managers is infamous as one cause for large scale economic crises [BLACK_SWANs]. Her motives could be manifold, still: maybe she was nursed to take risks in her life; or she is confident that consequences in case of failure will be bearable.

The second and third options have something in common: they make use of *external knowledge*. This is explained in the coming section. The fourth

option is another way of making *cumulative use of knowledge*, namely *[adaptive testing: find correct term]*.

2.3.4 Case: 99 seconds

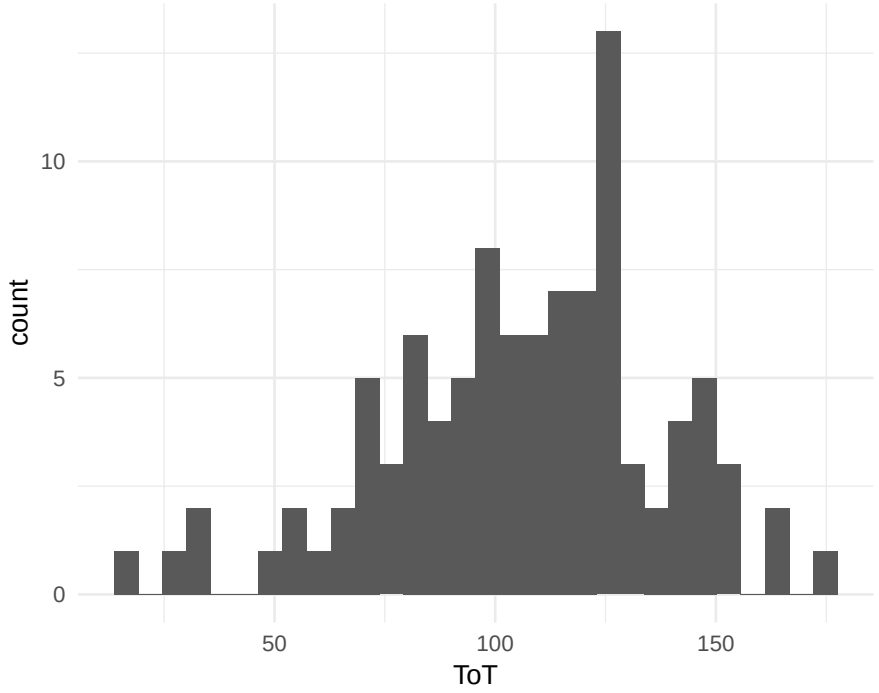
Consider Jane: she is chief engineer at a mega-large rent-a-car company smartr.car that is doing their business nothing but online. Jane was responsible for a large scale overhaul of the customer interface for mobile users. Goal of the redesign was to streamline the user interface, which had grown wild over the years and had become boring. Early customer studies indicated that the app needed a serious visual decluttering 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 requested the following:

> 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: i can find out for you (but we won't violate any rules).

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 what she sees:

```
attach(Sec99)
```

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



Jane, trained in statistics as she is, figures that if she would make such a statement, she wanted to be 90% sure. The performance is not completely off 99 seconds. But given the huge variance in the sample, it seems impossible to hold the desired claim with a certainty of 80%. Still, she gives it a try and runs an estimation (which we will later get to know as a *grand mean model*)

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

P_1 <-
  posterior(M_1)
```

```
T_fixef <-
  fixef(P_1,
        interval = .6)

N_99s <-
  P_1 %>%
  filter(parameter == "Intercept") %>%
  mutate(confirm = value <= 99) %>%
  summarise(certainty = mean(confirm)) %>%
```

```
  mutate(odds = (1-certainty)/certainty) %>%
  as.numeric()

N_111s <-
  P_1 %>%
  filter(parameter == "Intercept") %>%
  mutate(confirm = value <= 111) %>%
  summarise(certainty = mean(confirm)) %>%
  mutate(odds = (1-certainty)/certainty) %>%
  as.numeric()
```

```
detach(Sec99)
```

The results tell her that most likely the average time-on-task is $fixef$. That is not very promising. In fact, the odds that the average user can do it in 99 seconds is a feeble 2%. Luckily, Jane had the idea that the slogan could be changed to *"rent a card in 1-1-1 seconds"*. The certainty that this slogan holds is a favorable 95, a risk that a brave marketing department is surely willing to take.

2.3.5 Prior information

It is rarely the case that we encounter a situation *tabula rasa*. 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. Even the planet is sometimes taken into account: the Pathfinder probe carried a bag of gadgets, but the umbrella was for safe landing only.

In behavioural research it has been general 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. In Bayesian terminology, previous or external knowledge is called *prior information*. Kowledge that comes from the data is called the *likelihood* and both are combined to *posterior* knowledge by multiplication:

$$posterior = likelihood * prior$$

Reconsider Violet, the rational e-commerce development manager. She had this role for years and has supervised several large scale outrolls. Couldn't

she be more daring, taking into account her past successes? This is not a completely new situation, after all.

The impact of prior information can range from negligible to overwhelming. The gain in confidence in our example depends on three factors:

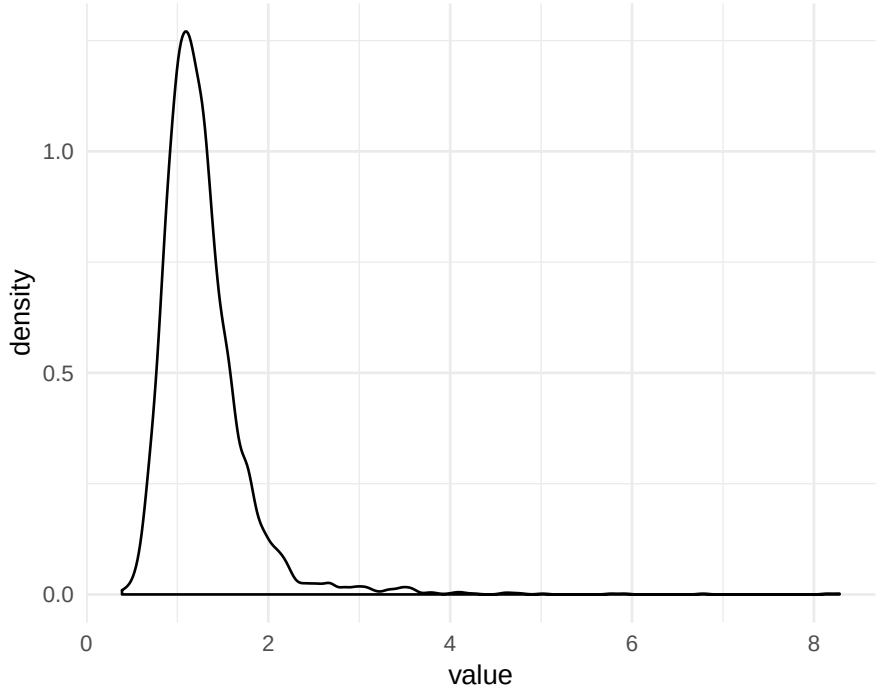
1) the average magnitude of past successes, the higher, the better
2) the similarity of the current project with past projects
3) the amount of evidence (e.g., number of recorded projects)

```
attach(Rational)
```

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

P_prior <- posterior(M_prior)
```

Let's assume, the manager had done 5 projects in the past and recorded the resulting change in revenues. On these five values she does a regression, that we will later call an intercept-only model, and concludes that her *prior belief* is distributed as:

```
P_prior %>%
  filter(parameter == "Intercept") %>%
  mutate(value = exp(value)) %>%
  ggplot(aes(x = value)) +
  geom_density()
```



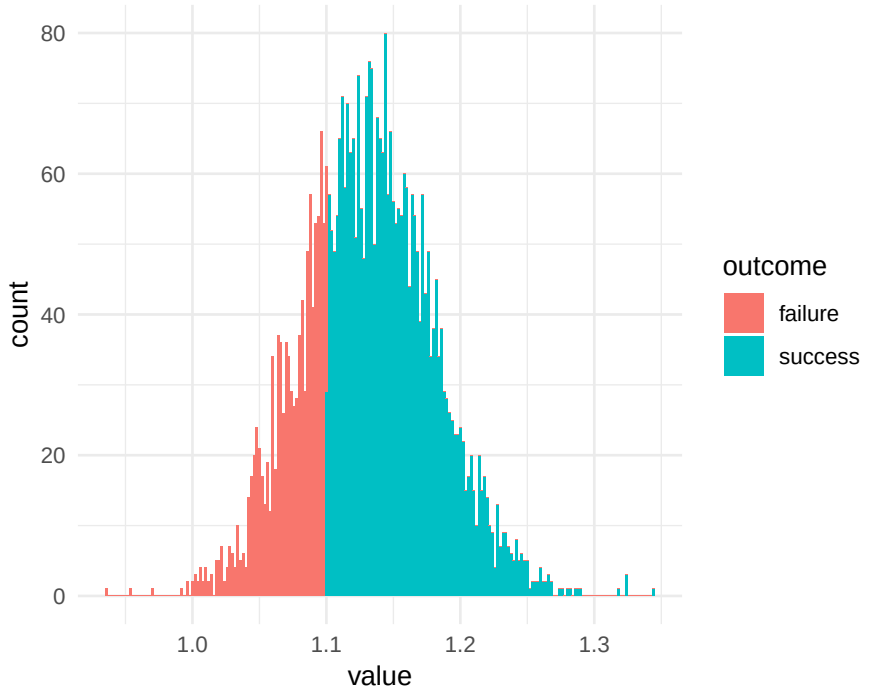
On average, revenues have increased by factor 1.211 by past design improvements. Of course, with only five points of measurement there is uncertainty as well. The mount of uncertainty is the spread of prior distribution. Using this prior information, the development manager runs another regression model on her experimental data, this time using both sources of information, her optimistic prior and the experimental data.

```
M_2 <-
  RD %>%
  stan_glm(Euro ~ Design,
      prior_intercept = normal(0, 100),
      prior = normal(T_prior[[1,2]], T_prior[[1,3]]),
      family = Gamma(link = "log"),
      data = .)

P_2 <- posterior(M_2)
```

```
T_fixef_2 <- fixef(P_2, mean.func = exp)
T_fixef_2
```

fixef	center	lower	upper
Intercept	4.99e+01	4.70e+01	5.31e+01
Designproto	1.13e+00	1.04e+00	1.23e+00
shape	1.66e+09	7.29e+06	1.13e+12

```
N_risk_of_failure_2 <-
  P_2 %>%
  filter(parameter == "Designproto") %>%
  mutate(value = exp(value),
         outcome = value < 1.1) %>%
  select(outcome) %>%
  colMeans

  P_2 %>%
  filter(parameter == "Designproto") %>%
  mutate(value = exp(value),
         outcome = ifelse(value < 1.1, "failure", "success")) %>%
  ggplot(aes(x = value, fill = outcome)) +
  geom_histogram(binwidth = .002)
```



```
detach(Rational)
```

3

Elements of Bayesian statistics

As human beings we make our decisions on what has happened to us earlier in time. 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 skills 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 believe vigorously experience a feeling of certainty, or lack of doubt. That is not to deny, that in a variety of situations, the boundedness of human cognition kick in and we becaome 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. 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 through and through. *Certainty* (or belief or credibility or credence) in Bayesian statistics is formalized as a *probability scale (0 = impossible, 1 = certain)*. Evidence accumulates by two mechanisms, the successive observations in a data set and what has already been learned in the past. When new data arrives and 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.

In the following the essential concepts of statistics and Bayesian analysis will be introduced.

3.1 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. Statistics fall into several broad classes that answer certain types of questions:

- 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?

3.1.1 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, which is directly linked to the level of certainty we can reach.

```
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 amount of data obtained gets larger. Second, taking a valid measure can always fail for a variety of reasons, reasulting 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] 97
```

Another important issue is the distribution of measures across groups in a data set. Again, the number of observations in a group is linked to the certainty we can gain on 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(proportion = n()/n_Gender)
```

Gender	proportion
female	0.59
male	0.41

Summarizing frequencies of metric measures, such as 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))
```

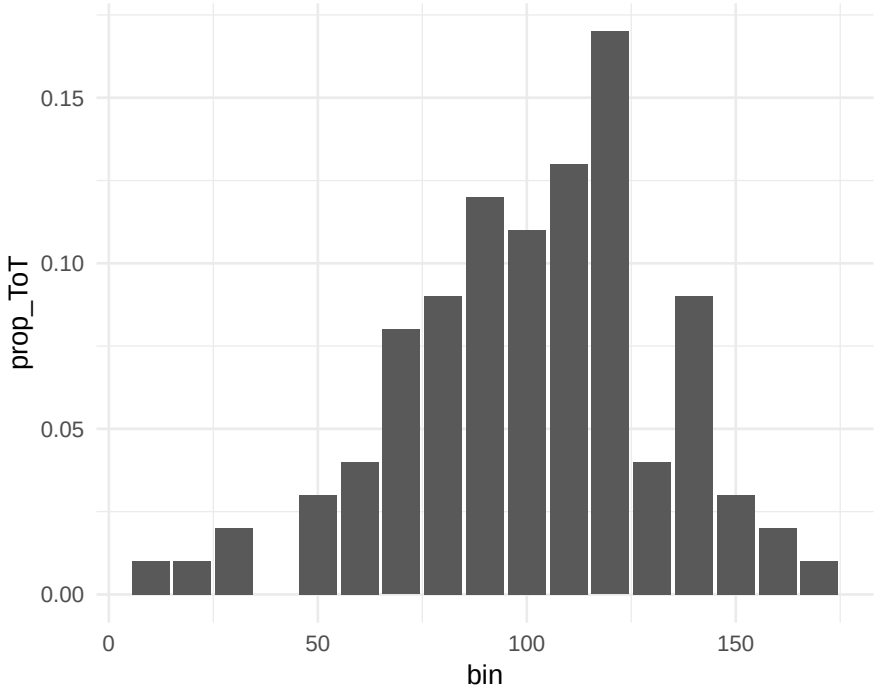
```
## [1] 2
```

```
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. As the histogram reveals, it is not very representative.

```
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(prop_ToT = n()/n_ToT) %>%
  ggplot(aes(x = bin, y = prop_ToT)) +
  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. A plot that shows the density of values in a sequence of bins is called a histogram. It approximates what we will later get to know more closely as a *distribution*.

3.1.2 Central tendency

Reconsider Jane 2.3.4. 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.

```
mean <- function(x) sum(x)/length(x)
```

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

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

Imagine a competitor would go to court. Not an expert in that matter, 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? And here are some other ways to interpret the same slogan:

"*50% (or more) of users* can rent a car in 99 seconds". This is called the *median*. The median is computed by bringing all measures into an order and collect the element right in the center, or the mean of the center pair of values when the number of observations is even.

The median is a special case of so called *quantiles*. 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, .75)
```

```
## 75%
## 125
```

A common pattern in the distribution of measures is that a majority of observations densely 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.

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

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

```
knit_print.tbl_obs <- function(x, ...) {
  #data_set <- deparse(substitute(x))
  n <- min(8, nrow(x))
  tab <- dplyr::sample_n(x, n)
  if("Obs" %in% colnames(tab)) tab <- dplyr::arrange(tab, Obs)
  if("Part" %in% colnames(tab)) tab <- dplyr::arrange(tab, Part)
  cap <- stringr::str_c("Data set",": showing ", n, " of ", nrow(x), " observations")

  res = paste(c("", "", knitr::kable(tab, caption = cap, format = "markdown", ...)),
              collapse = "\n")
  knitr::asis_output(res)
}
```

```
Ver20
```

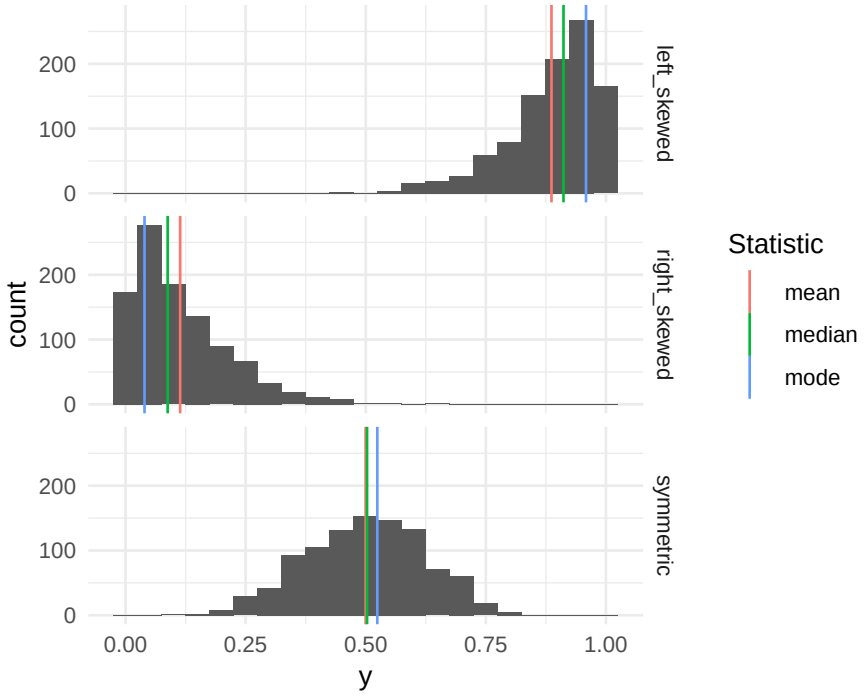
Obs	Part	ToT	age	Gender
32	32	126.1	53	female
44	44	83.2	41	male
49	49	92.1	48	female
51	51	114.7	26	female
67	67	115.1	29	male
78	78	118.9	51	female
80	80	72.0	43	female
82	82	112.7	42	male

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

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.

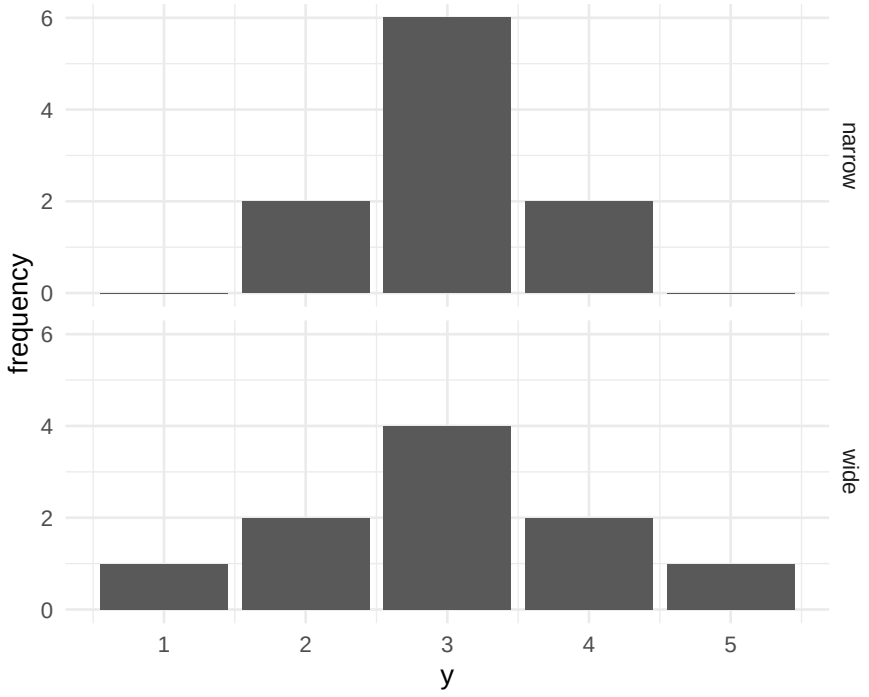
To summarize, the mean is the most frequently used measure of central tendency, one reason being that it is a so called *sufficient statistic*, meaning that it exploits the full information present in the data. The median is frequently used when extreme measures are a concern. The mode is the point in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

3.1.3 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.

The range statistic 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 deviatons 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)
```

3.1.4 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?

A majority of research deals with associations between variables 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. *Categorial 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:

	categorial	metric
categorial	frequency cross tables	differences in mean
-	-	-
metric		covariance, correlation
-	-	-

All forms of associations derive directly from statistics we have already encountered. That is obvious for when one variable is categorial and a little more subtle for two metric variables.

When there are only categories in the game, but no measures, the only way to compare is by frequencies. To illustrate the categorial-categorial 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 and it is recorded whether the sequence was completed correctly or not.

```
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, it is preferred to report proportions within a group. 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

In a similar manner, associations between groups and metric variables are reported by group means. 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 is no 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 associations between two metric variables, *covariance* is a commonly employed statistic for the association between two variables and derives directly from variance, with the difference that the square of deviations is replaced by the multiplication of two deviations. For example, we may want to assess whether there is any relationship between years of experience and ToT:

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

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

```
cov(D_agg$experience, D_agg$ToT)
```

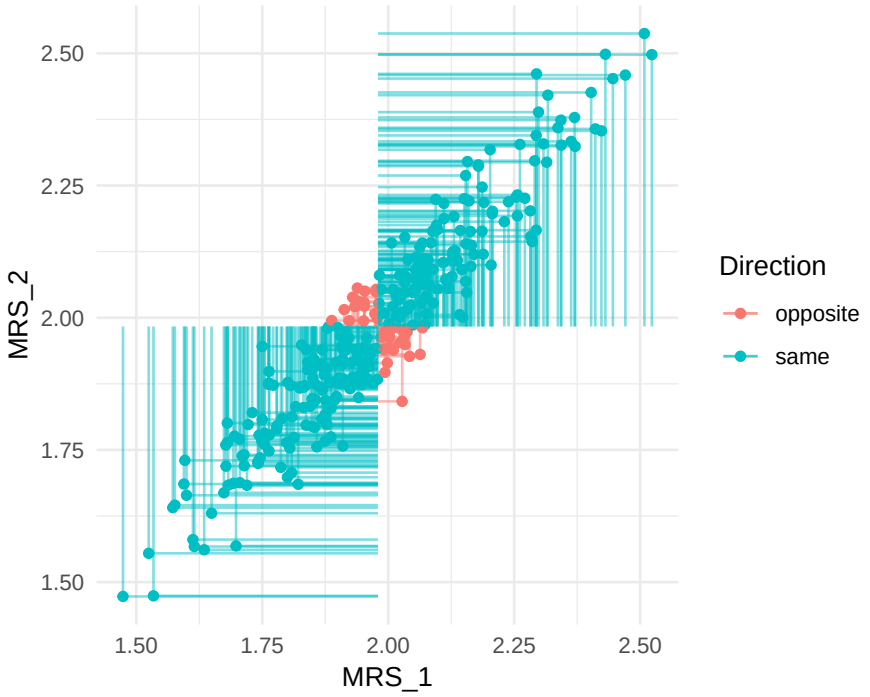
```
## [1] 271
```

When at large the deviations go into the same direction from the respective mean, covariance becomes positive. When the deviations primarily move in opposite direction it is negative. When the picture is mixed, covariance will stay close to zero. The three plots below illustrate three covariances, a strongly positive, a weakly positive and a moderate negative one.

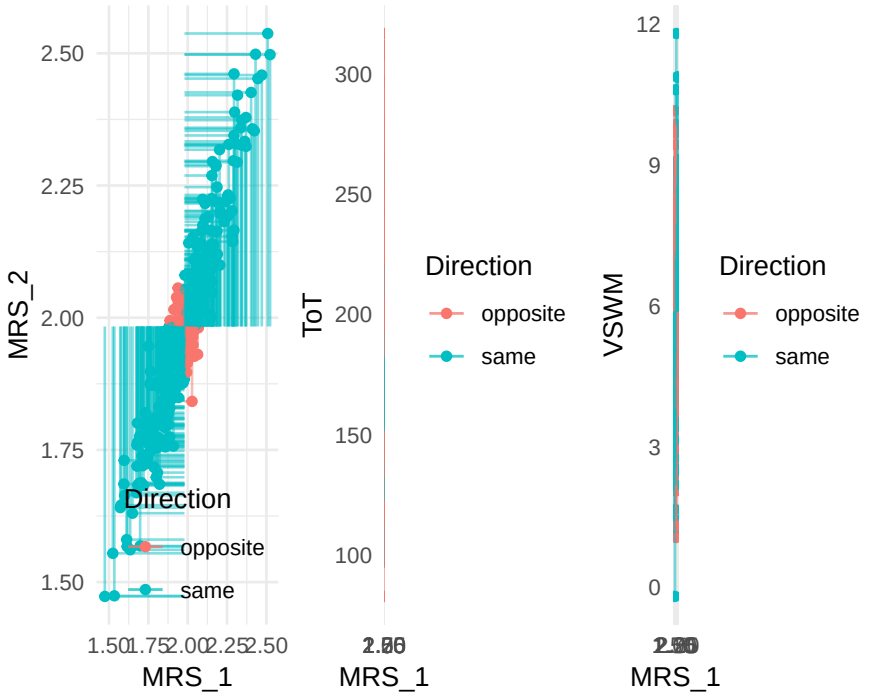
```
cor2cov <- function(cor, sd) diag(sd) %*% cor %*% t(diag(sd))

cor_mat <- matrix(c(1, .95, -.5, .2,
```

```
                    .95, 1, -.5, .2,
                    -.5, -.5,  1, .15,
                    .2, .2, .15, 1), ncol = 4)
sd_vec  <- c(.2, .2, 40, 2)
mean_vec <- c(2, 2, 180, 6)

D_psychomet <-
  mvtnorm::rmvnorm(300, mean = mean_vec, sigma = cor2cov(cor_mat, sd_vec)) %>%
  as.tibble() %>%
  rename(MRS_1 = V1, MRS_2 = V2, ToT = V3, VSWM = V4)
```

```
deviation <- function(x) mean(x) -x
direction <- function(x,y) if_else(sign(deviation(x)) == sign(deviation(y)), "same", "opposi
ggcov     <- function(D, x, y, legend.position = "none")   {
  quo_x <- enquo(x)
  quo_y <- enquo(y)
  out <-
    D %>%
    mutate(Direction = direction(!! quo_x, !! quo_y)) %>%
    ggplot(aes(x = !! quo_x,
               y = !! quo_y,
               col = Direction)) +
    geom_point() +
    geom_segment(aes(xend = mean(!! quo_x), yend = !! quo_y), alpha = .5) +
    geom_segment(aes(xend = !! quo_x, yend = mean(!! quo_y)), alpha = .5)
  out
}

ggcov(D_psychomet, MRS_1, MRS_2)
```




```
grid.arrange(
  D_psychomet %>%  ggcov(MRS_1, MRS_2)  +
    theme(legend.justification=c(1,0), legend.position=c(1,0)),
  D_psychomet %>%  ggcov(MRS_1, ToT),
  D_psychomet %>%  ggcov(MRS_1, VSWM),
  ncol = 3
)
```



The graphs highlight the deviation from the population mean of each variables. A strong covariance emerges whenever there is a strong tendency to deviate in either one direction. In fact, the illustration introduces a geometric interpretation: every data point stretches a rectangle which has an area of the product of the two deviations, just like in the formula of covariance.

As intuitive the idea of covariance is, as unintelligible is the statistic itself for reporting results. Covariance is not a pure measure of association, but is contaminated by the dispersions of X and Y. For that reason, two covariances can hardly be compared. The *Pearson correlation coefficient* r solves the problem by rescaling a covariance by the two standard deviations:

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

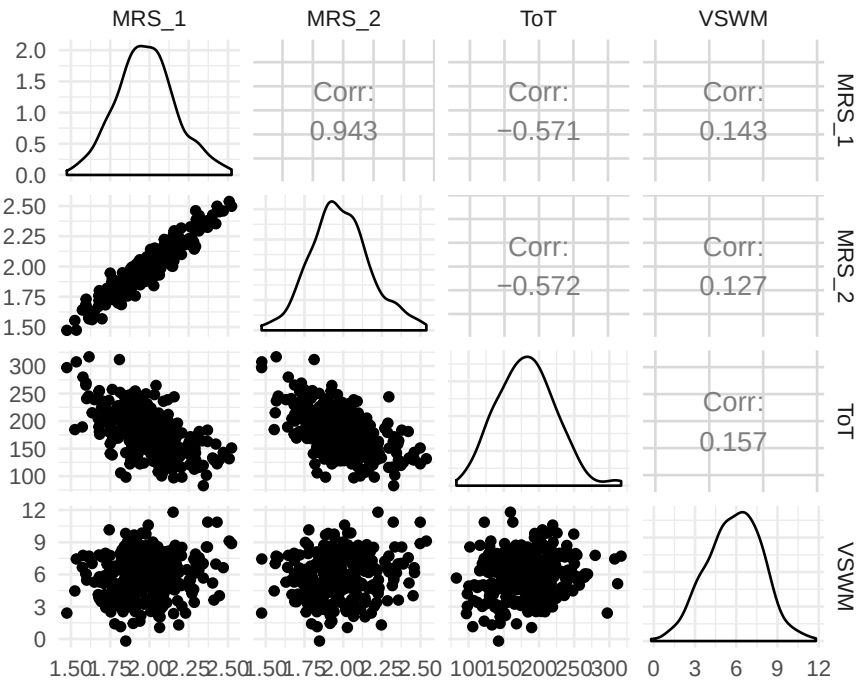
```
cor <- function(x, y)
  cov(x, y)/(sd(x, na.rm = T) * sd(y, na.rm = T))

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

```
## [1] 0.943
```

Due to the standardization of dispersion, r lets the researcher interpret 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 for several purposes.

In the field of psychometrics, correlations are ubiquotously employed to represent *reliability* and *validity* of tests. For example, when the long-term career of an individual is dependent on the score of an assessment, it would be inexceptable if test scores differed strongly from day to day. *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 is a stable, such that we can use it for long-term predictions of whether someone will become a good surgeon. A reliability of .95 will probably satisfy most psychometricians. Validity of a test means that it predicts what it was intended for. 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 is already part of the 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.

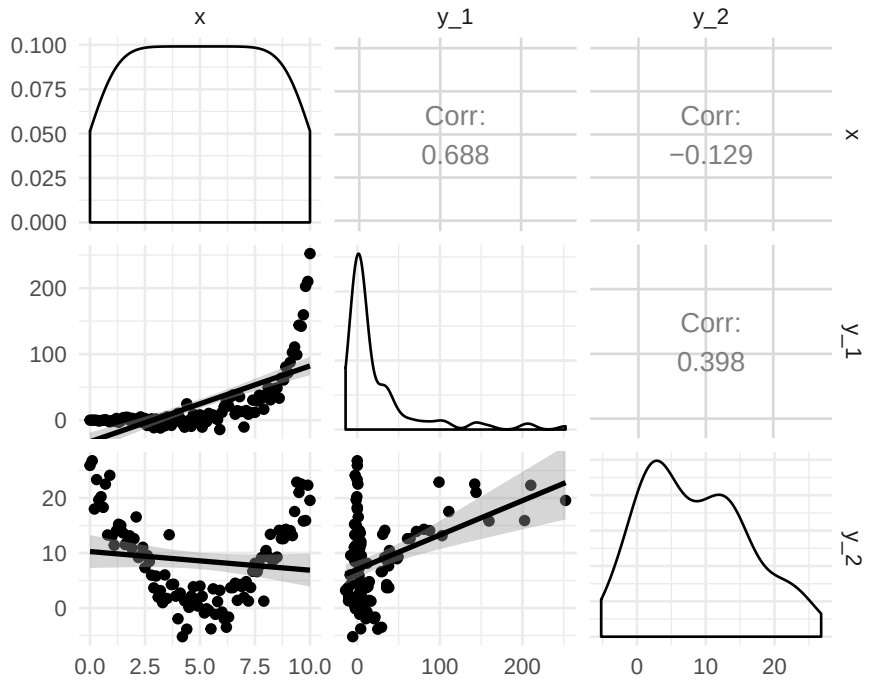
```
D_psychomet %>%
  GGally::ggpairs()
```



Correlations allow psychometricians to employ absolute standards for the quality of measures. 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 ubitutous 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.

```
data_frame(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"))
```



3.1.4.1 [HERE]

3.2 Bayesian Inferential statistics

Bayesian statistics can be reduced to three elements:

1. the *prior belief* is what you believe before (little rain in summer)
2. the *likelihood* is what you learn by observation (the cloudy sky)
3. the *posterior belief* is your adjusted believe after seeing the data

Bayesian statistics formalizes the transition from prior belief to posterior belief in a remarkably simple formula:

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

Note that $\propto$ here means *proportional to*. In very plain words this is:

> what you believe now is a combination of what you knew before and what you have just seen in the data.

The data is usually the present observation or study. But, that does not exclude that prior knowledge grounds on data, too. In experimental Psychology researchers entertain themselves repetitions of the very same experimental paradigm, with slight variations maybe. 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 repeat the experiment another time and find the reverse effect, no one would seriously take this as a debunk. This extreme example illustrates another principle that follows from Bayes rule:

> Today's posterior is tomorrow's prior.

There is no principled difference between prior and posterior. They are just levels of belief (credences) at different points in time. Both differ in strength: prior knowledge can be firm when it rests on an abundance of past evidence. The same holds for the likelihood in the present data: the more observations, the stronger the evidence. This is why larger sample sizes are usually preferred. Prior belief and likelihood can be congruent or contradict each other. When they are congruent, prior belief is strengthened by the data. When they are contradicting each other, prior belief is weakened.

Whatever happens in the individual case, it is generally accepted that scientific progress is incremental over large periods of time. Under this perspective the idea of updating one's belief is even trivial. It is common sense, that if you are too uncertain about a situation you better gather more information. Once you have reached a satisfactory level of certainty, you proceed to act (or publish).

Readers with a firm background in classic statistics may (or my not) recall that once you have settled down on a sample size (and a level of significance), you absolutely must test precisely this number of participants. Stopping early (because you have reached the desired p-value) or adding to the sample is strictly forbidden. Doesn't that bother you under the perspective of accumulative evidence? It should bother you that incremental collection of evidence is impossible when doing frequentist statistics. On the other hand, in Bayesian statistics, accumulation of evidence is a core feature.

Neither is there a way to express one's prior belief when doing a t-test, nor can you just continue your data collection until satisfactory certainty is reached. In the fictional example of the Stroop task, the classic data analysis pretends as if no one has ever done such an experiment before. At the same time, it is strictly forbidden to invite further participants to the lab, when the test results point into the right direction, but evidence is still to weak. If you planned the study with, say, $N = 20$, this is what you have to 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 (e.g., $p = .52$), the only permissable action is dump your data and start over from zero. There are many other ways that frequentist statistics is flawed, including some deeply philosophical ones. For a common person the denial of incremental progress is deeply counter-intuitive and for a common researcher it is a millstone around the neck.

Reconsider Jane and Andrew. What did they know about the current state of affairs when running a particular session. Using some time-on-task measures they disproved the claim "rent a car in 99 seconds". Recall how precisely the question was phrased: on average, users had to be able to complete the transaction in 99 seconds. The statistic of interest is the mean. This was debunked with almost no effort, by calculating:

$$\frac{\hat{M}_{ToT} = 1}{n * \sum ToT = 105.975}$$

But how about the updated slogan: "rent a car in 111 seconds". Can we be sure it holds, when someone repeats the study? We can only to a degree. It could still happen, that the belligerent competitor comes to a different result, just because they have a different sample of participants. Even if they would test a fully matching sample of participants, the measures will differ, simply because an array of smaller and larger impact factors is continuously hitting the central and peripheral nervous systems of your participants. The result is randomness in the data. Fortunately, randomness is often found to be well-behaved in that recurrent patterns emerge. These patterns are called distributions of randomness and I will introduce a whole bunch of them later in this chapter @**??**).

3.2.0.1 [HERE]

3.3 Bayesian probability theory

Probability is an elusive concept and a source of mental suffering and heated debates, especially when people use intuition. The reason is that in the human mind, probability can be rooted in two different ways: in frequentist thinking, probability is represented by relative frequencies, whereas in Bayesian school of thought it is the level of certainty for some event to happen.

This is a Bayesian book and the following considerations are meant to convince the reader that the Bayesian use of probability has a wider domain of application than the frequentist. I also see this as one of the more rare situations where introducing some formalism is supportive in that it dissolves the subtle preoccupations that often accompany intuitive (or, I'd rather say:

exemplified) understanding. In addition, the mathematical definition is bare of any real world notions, like observed frequencies or experienced certainties and (figuratively speaking) makes the different perspectives converge. The algebraic definition of probability is given by the three Kolmogorov axioms. Before we come to that, let me undertake two intermediate steps: a deep bow to the discipline of mathematics, and introducing some set-theoretic concepts.

First, be reminded that mathematical systems of belief are empty. It often helps, to associate a mathematical system of statements to with some more tangible ideas. For example, I remember how my primary school teacher introduced the sum of two numbers as moving elements from one stack onto another, piece by piece. Later, I found this to be an exact embodidment of how the sum follows from the Peano axioms, that define the set of natural numbers. Second, we need to set the stage with a few set-theoretic concepts.

3.3.1 Some set theory

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:

- empty like all opponents who can defeat Chuck Norris, $\{\}$ or $\oslash$
- finite like the possible states of a traffic light
- 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 uncaountable 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?

Obs	Timely	Harm	Success
4	TRUE	TRUE	FALSE
6	TRUE	TRUE	FALSE
11	TRUE	FALSE	TRUE
17	TRUE	FALSE	TRUE
22	FALSE	TRUE	FALSE
23	FALSE	TRUE	FALSE
24	FALSE	FALSE	TRUE
30	FALSE	FALSE	FALSE

Note how the the data table makes use of logical values to denote their membership to a set. Based on these three criteria, we can extract subsets of observations:

```
library(sets)
```

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

The first basic set operator is the *complementary set*. It selects all elements from the domain that are not part of a given set:

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

Once there is more than one set in the game, set operators can be used to create all kinds of new sets. First, the *union* of two sets collects the elements of two separate sets into one new set, for example, the set of all tasks 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 succesful task completions:

```
Success & Timely
```

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

The *set difference* removes elements of one set from another, for example the set of all successful observations with no harmful side effects:

```
Success - Harmless
```

```
## {}
```

It turns out that all succesful observations are also harmless. That indicates that the set of successful events is a *subset* of harmless events. 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). A distinction is made between subsets and *proper subsets*, where one set contains all elements of another, but never the other way round. If two sets are mutual subsets (and therefore improper), they are *equal*.

```
# 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 another important element of set theory: the empty set, which has the special property of being a subset of all other set:

```
set() < Success
```

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

Among other uses, the empty set is important for intersections. 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* and complementary sets are 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. We can easily conceive a set that has elements that are sets. 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 introduction of probability, we need two concepts related to sets o 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 conplementary 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)
```

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 measures. Loosely speaken, 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 have their counterparts in the numerical domain. Probabilities are measures and in the next section we will see how numerical operations on probabilities relate to set operations in a Σ algebra.

3.3.2 Probability

Before we get to a formal presentation of probability, we can develop the idea on the fictional validation study 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, Harm, Timely) %>%
  summarize(n = n()) %>%
  ungroup() %>%
  complete(Success, Harm, Timely, fill = list(n = 0)) %>% # adds empty events
  mutate(pi = n/sum(n))

D_freq
```

Success	Harm	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 disjunct subsets

2. All subsets got a real number assigned, by the operation of relative frequencies, that is between (and including) zero and one.
3. The sum of these numbers is one.

The common mathematical theory of probability assumes a set of outcomes Ω and a Σ algebra F on Ω, like the power set. An element E of F is called an *event*. Note the difference between outcomes, which are singular outcomes, and events, which are sets of outcomes, like the eight outcome categories above. Also note that these eight sets are a partition of Ω, but not a Σ-algebra. However, we can easily construct a Σ-algebra by adding all possible unions and intersections. Or we use the power set of outcomes in the data set right-away.

The *first Kolmogorov axiom* states that a probability is a non-negative real number assigned to every event. Take a look at the table above, to see that this is satisfied for the relative frequencies.

While the first axiom defines a lower border of zero for a probability measure, the *second Kolmogorov axiom* cares for the upper limit (although somewhat 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 probabilites. 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, is the set of all successful observations is 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 give just one 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:

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

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 occuring 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 occurance 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, we even have 100% certainty in our guess, because an observation was only rated a success if there was no harm. But we can easily think of more gradual ways to make a better guess.

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

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

Harm	n	pi
FALSE	18	1

Perhaps, there is a relationship between Timely and Harm in the manner of a speed-accuracy trade-off. 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, Harm) %>%
  summarize(n = n()) %>%
  mutate(pi = n/sum(n)) %>%
  ungroup()
```

Timely	Harm	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 sitation is called *independence of events* and it means that knowing about one variable does not help in guessing the other. In statistics, *conditional probability* is an important concept. In particular, it will carry us away from the

set theoretic interpretation towards the Bayesian interpretation of probability as states of knowledge.

- joint probability
- conditional probability (events in a sequence –> knowledge of A is knowledge of B)
- independence
- Bayes theorem

Other definitions and basic numerical operations on probabilities can be inferred in similar ways.

The basic operations on probability then give rise to more advanced laws of probability:

3.3.3 Certainty as probability

Inferential statistics serves rational decision making under uncertainty by attaching information on the *level of certainty* to a parameter of interest. A central difference between frequentist and Bayesian statistical theory is how the elusive concept of *certainty* emerges. Frequentists just stick to the notion of relative frequencies to express a level of certainty. A common way to express ones level of certainty about a parameter (say, the population mean) is a *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 by convention that the 95% confidence interval is given and its definition is rooted in 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.*

As much as I embrace parsimony, 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. (They are all imagined.) When we turn to Bayesian statistics, we find that it generously elevates certainty to be a basic quantity, rather than a derived. A Bayesian 95% *credibility interval* represents the level of certainty 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 EURO 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 concept of certainty is a probability in mathematical terms, which liberates our reasoning from the requirement to think of long-running series of the same experiment. Let's face the truth: much of the time, we have only this one shot. In the next section we will see how probability theory is used to operate on certainties using Bayes famous theorem.

3.3.4 Bayes theorem

3.4 Statistical models

It is a scientific 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 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 next. Of course, such an entity could never exist, because it were actually a computer that matches the universe in size. In addition, there are limits to how precisely we can measure the current state, although physicist and engineers have pushed this very far.

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 to pin down the neural state of affairs. Her lack of knowledge is profound but still not a problem as the research question was gross, too, not asking for more than the difference in *average* duration. Instead, imagine Violet and a collegue had invented a silly game where they both guess the time-on-task of individual participants. Who comes closest wins. As both players are clever 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 the average ToT in all previous sessions. As gross as this is, it qualifies as a model, more precisely the grand mean model [LM]. The model explains all observations by the population mean.

Of course, Violet would never expect her grand mean model to precisely predict the outcome of a session. Still, imagine a device that has perfect knowledge of the car rental website, the complete current neural state of a the participant and the physical environment both are in. The device would also have a complete and valid psychological theory. With this device, Jane could always make a perfect prediction of the outcome. Unfortunately, real design researchers are far from Laplace demoism. 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 a myriad of *small unrelated forces (SMURF)* can push or hinder the user towards completion.

Laplace demon has perfect knowledge of all SMURF trajectories and therefore can produce a perfect prediction. Violet is completely ignorant of any SMURFs and her predictions will be off many times. A common way to conceive this situation is that observed values y_i are composed of the *expected value* under the model μ_i and a *random part*, ϵ_i

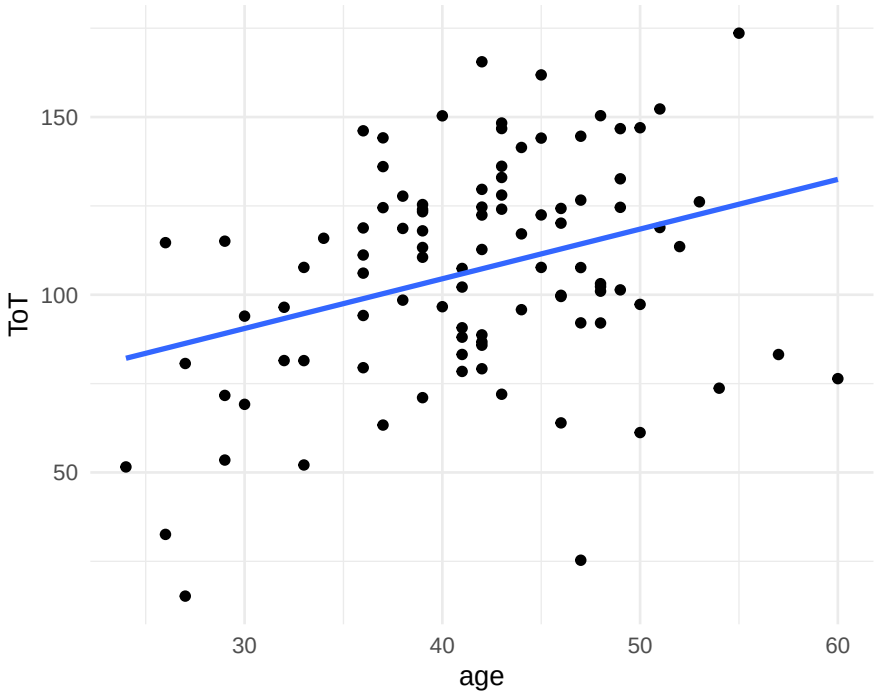
$$y_i = \mu_i + \epsilon_i$$

Generally, statistical models consist of these two parts: the *likelihood* to describe the association between predictors and expected values and the random part, which describes the overall influence of the unexplained SMURFs.

3.4.1 Predictions and likelihood

The likelihood function states the dependency of outcome on the predictor variables. The dependency can be a complex mathematical function of multiple predictors, or as simple as the population average. A common likelihood function is the linear function. For example, in their guessing game, Violet could try to improve her population model, by also taking age of participants into account. Older people tend to be slower. Violet creates 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 30 year old persons have an average ToT of around 90 seconds, which increases to around 120 seconds for 50 year olds. Arithmetically, this is an increase of around 1.5 seconds per year of age.

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



Violet can use this information to improve her gambling. Instead of stoically calling the population mean, she uses a linear function as predictor: $90 + (age - 30) 1.5 $. In Bayesian statistics, this is called a *likelihood function* and the general form for a single linear likelihood function is:

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

Likelihood functions connect the *expected value* μ with *observed variables* $x_{i1}, x_{i2}, ..., x_{ik}$, and (to be estimated) parameters, e.g. β_0, β_1. The likelihood function is often called the *deterministic part* of a model, because its prediction strictly depends on the observed values and the predictors, but nothing else. For example, two persons of age 30 will always be predicted to use up 90 seconds. Apparently, this is not the case for real data.

The linear model is very common in statistical modelling, but likelihoods can basically take all mathematical forms. For example:

- the grand mean model, Violet used before: $\mu_i = \beta_0$
- 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 likelihood function is the interesting part of the model, 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 staionary 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 likelihood functions is that μ_i and x_i have indicators i. Potentially, every observation i has their own realization of predictors and gets a unique expected value, whereas the parameters β_0, β_1 asf. are single values that apply for all observations at once. In fact, we can conceive statistical models as operating on multiple levels, where there is always the two: the observation level and the population level. When introducing multi-level models, we will see how this principle extends to more than these two levels. Another related idea is that parameters summarizes patterns found in data. Any summary implies repetition and that is what the likelihood expresses: the pattern that repeats across observations and is therefore predictable.

3.4.2 Distributions: patterns of randomness

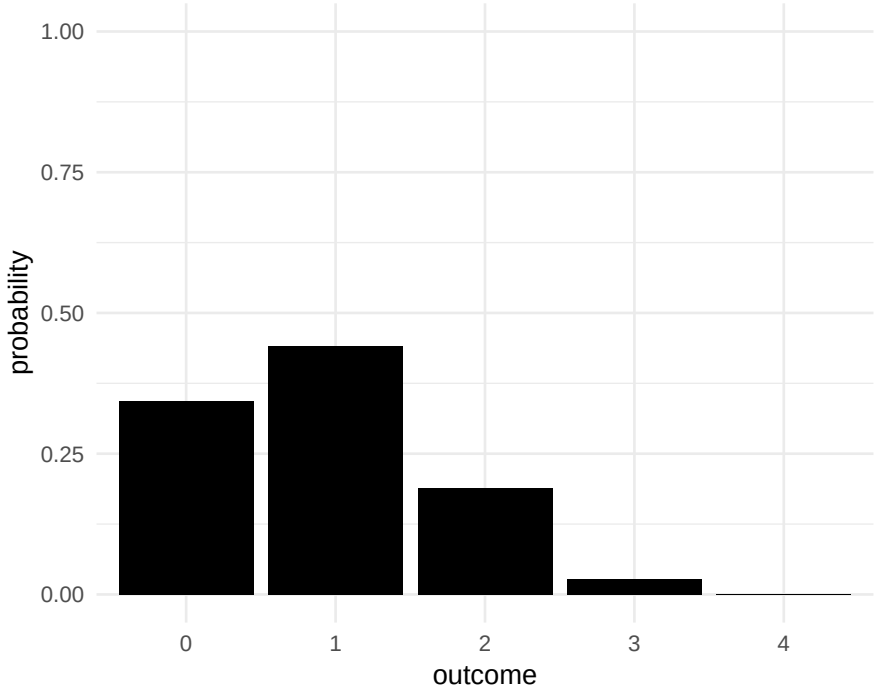
The random part of a statistical model is what changes between observation and is not predictable. When using the grand mean model, the only information we are using is that the person is from the target population. Everything else is left to the unobserved SMURFs and that goes into the random part of the model. Fortunately, SMURFs don't work completely arbitrary. Frequently, recognizable patterns of ramdomness emerge. These patterns can be formulated mathematically as probability and density distributions. A probability distribution is typically characterized as a probability mass function 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

3.4.2.1 Probability distributions

Probability distributions are mathematical functions that assign probabilities to the outcome of a measured variable y. Consider a participant who is asked to complete three tasks of constant difficulty, such that there is a chance of 30 percent for each one to be solved. The outcome variable of interest is the number of correct completions (0, 1, 2 or 3). Under idealized consitions, the following random distribution gives the probability of every possible outcome.

```
D_three_tasks <-
  data_frame(y = 0:4,
             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")
```

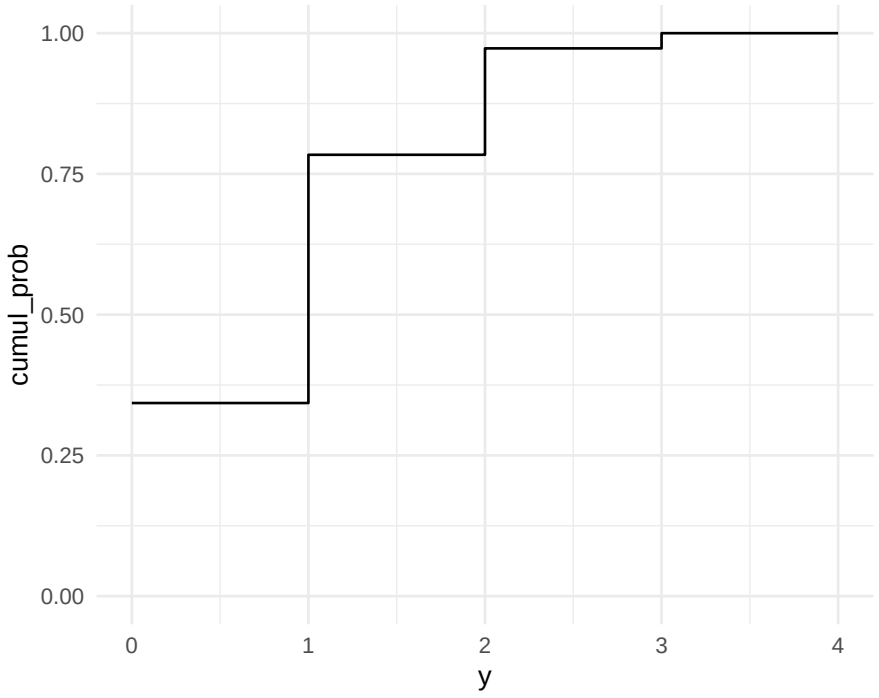


Further, we observe that the most probable outcome is exactly one correct task, which occurs with a probability of $P(y = 1) = 0.441$. At the same time, there is ample possibility for all failures, $P(y = 0) = 0.343$. We may also look at *combined events*, say the probability for less than two correct. 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, between zero and three and certainty means the probability is 1, or: $P(0 \leq y \leq 3) = 1$. As a matter of fact, all probability (and density) distributions fulfill that property.

More precisely, the area under the PMF must be exactly one and that brings us directly to a second form of characterizing the random distribution: the *cumulative distribution distribution (CDF)* renders the probability for the outcome to be smaller or equal to y. In the case of discrete outcomes, this is just stacking (or summing) over all outcomes, just as we did for $P(y \leq 1)$ above. The CDF of the three-tasks example is shown in the graph below. We recognize the left starting point, which is exactly $P(y = 0)$ and observe large jump to $P(y \leq 1)$. Finally, at $y \leq 3$ the function reaches the upper limit of 1, which is full certainty.

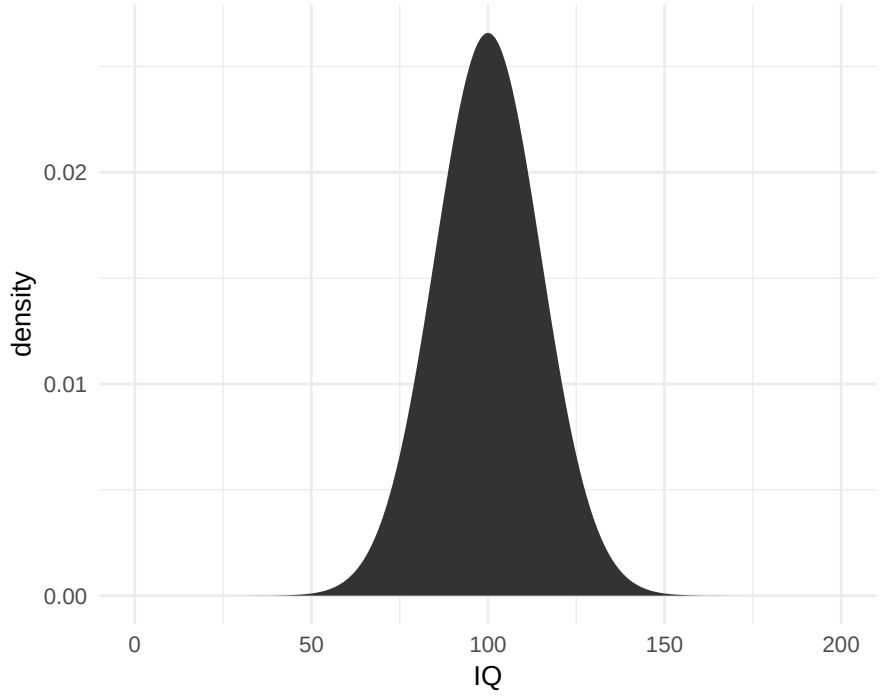
```
D_three_tasks %>%
  ggplot(aes(x = y, y = cumul_prob)) +
  geom_step() +
  ylim(0,1)
```



3.4.2.2 Density distributions

In the three-tasks example, reading and recombining probabilities is like counting blocks and stacking them upon each other. This is how most children learn basic arithmetics. when the outcome measure is *continuous*, rather than discrete, some high school math is required. The most common continuous measure is probably durations. As we will see, durations take quite tricky random patterns, so for the sake of simplicity, consider the distribution of intelligence quotients (IQ). Strictly spoken, the IQ is *not* continuous, as one usually only measures and reports whole number scores. Still, for instructional purposes, assume that the IQ is given in arbitrary precision, be it 114.9, 100.0001 or $\pi * 20$.

```
D_IQ <- data_frame(IQ = 0:200,
                   density = dnorm(IQ, 100, 15),
                   cdf = pnorm(IQ, 100, 15),
                   SE = (IQ > 85) * (IQ < 115) * density,
                   PDF_085 = (IQ < 85) * density,
                   PDF_115 = (IQ < 115) * density)

D_IQ %>%
  ggplot(aes(x = IQ, y = density)) +
  geom_area()
```

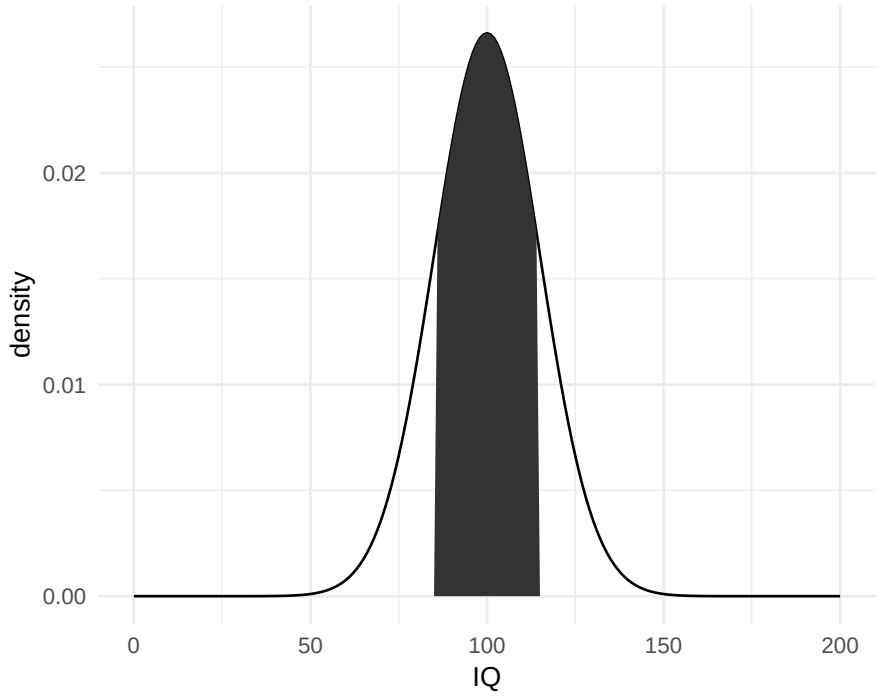


We observe that the most likely IQ is 100 and that almost nobody reaches scores higher than 150 or lower than 50. But, how likely is it to have an IQ of exactly 100? Less than you might think! With continuous measures, we can no longer think in blocks that have a certain area. In fact, the probability 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. Generally, with continuous outcome variables, We can no longer read probabilities directly. Therefore, probability mass distributions don't apply, but the association between outcome and probability is given by what is called *probability density functions*. What PDFs share withg PMFs is that the area under the curve

is always exactly one. They differ in that PDFs return a density for every possible outcome, which by itself is not as useful as probability, but can be converted into probabilities.

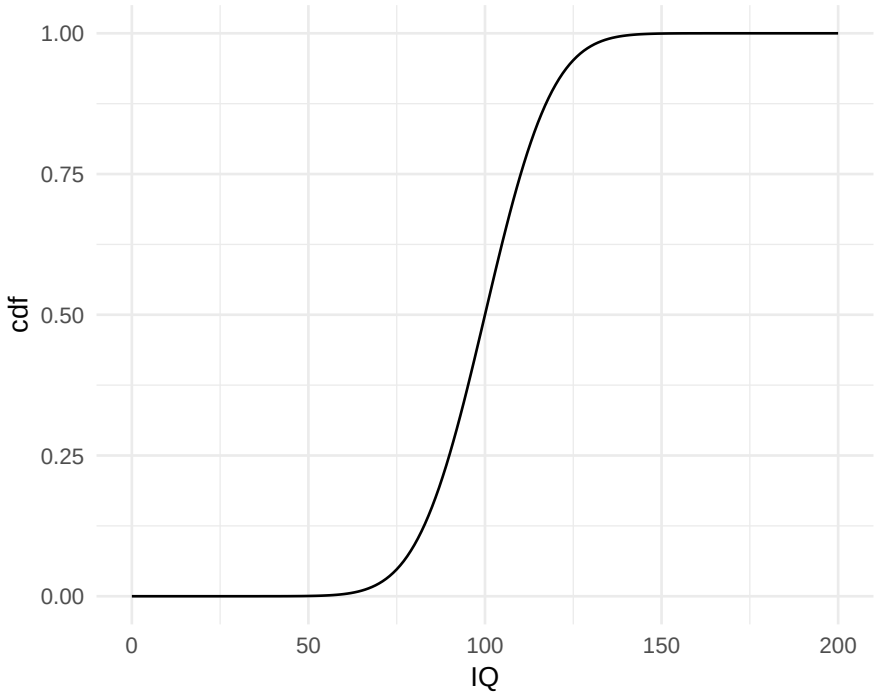
Practically, nobody is really interested in infinite precision. When asking *"what is the probability of IQ = 100?"*, the answer is *"zero"*, but what was really meant was: *"what is the probability of an IQ in a close interval around 100?"*. Once we speak of intervals, we clearly have areas larger than zero. The graph below shows the area in the range of 85 to 115.

```
D_IQ %>%
  ggplot(aes(x = IQ, y = density)) +
  geom_line() +
  geom_area(aes(y = SE))
```



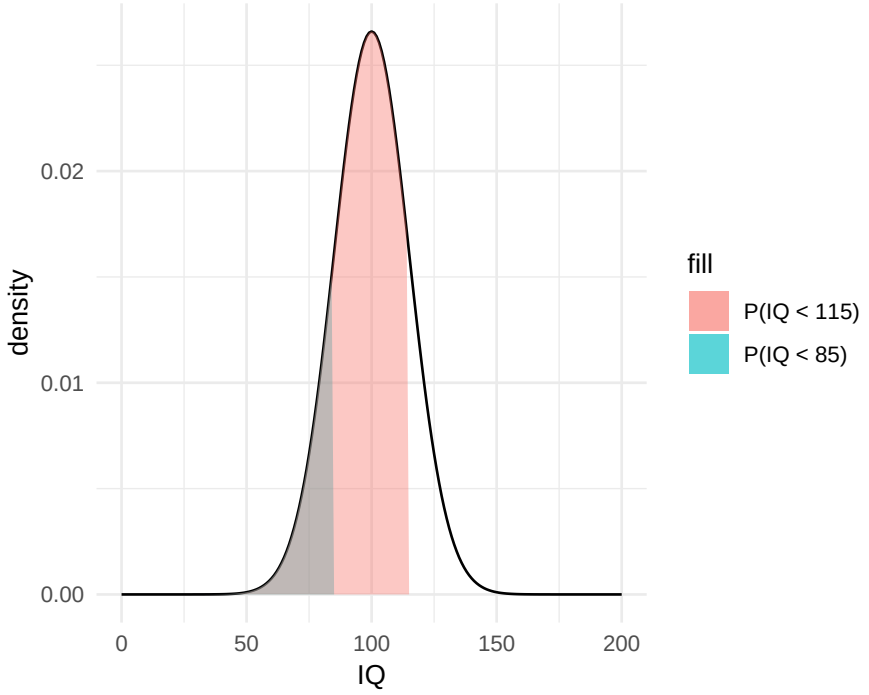
But, how large is this area exactly? As the distribution is curved, we can no longer simply count virtual blocks. Recall that the CDF gives the probability mass, i.e. the area under the curve, for outcomes up to a chosen point. Continuous distributions have CDFs, too, and the the graph below shows the 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()
```



Take a look at the following graph. It shows the two areas $IQ \leq 85$ and $IQ \leq 115$. The magic patch in the center is just the desired interval.

```
D_IQ %>%
  ggplot(aes(x = IQ, y = density)) +
  geom_line() +
  geom_area(aes(y = PDF_085, fill = "P(IQ < 85)"), alpha = .4) +
  geom_area(aes(y = PDF_115, fill = "P(IQ < 115)"), alpha = .4)
```



And here the CDF comes into the play. To any point of the PDF, the CDF yields the area up to this point and we can compute the area of the interval by simple subtraction:

$$P(IQ \leq 115) = 0.159 P(IQ \leq 85) = 0.841 P(85 \leq IQ \leq 115) = P(IQ \leq 115) - P(IQ \leq 85) = 0.683$$

Probability and density distributions usually are expressed as mathematical functions. For example, the function for the case of task completion is the binomial distribution, which gives the probability for y successes in k trials when the success rate is p:

$$Pr(y|p,k) = \binom{k}{y} p^y (1-p)^{k-y}$$

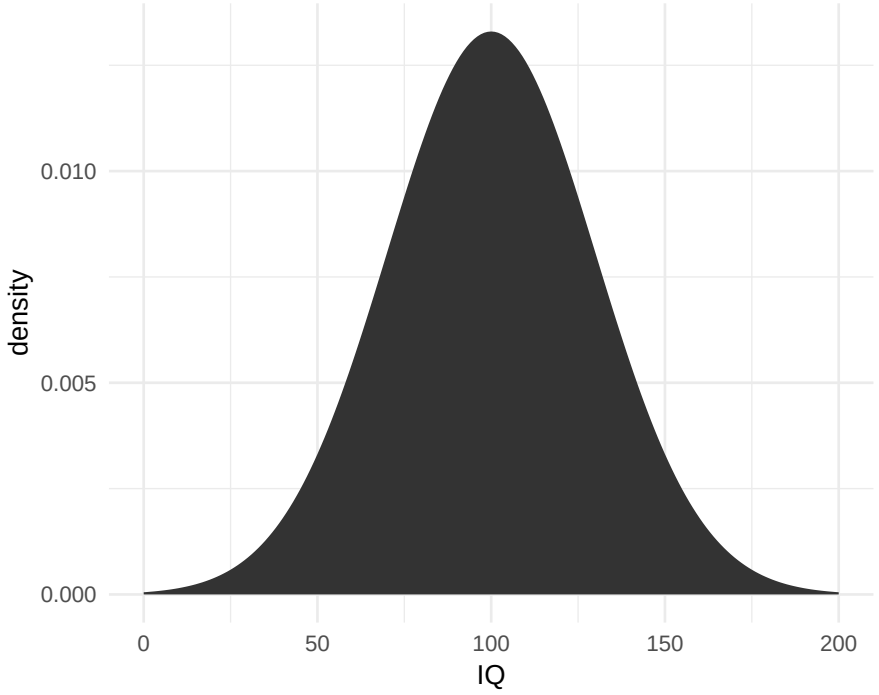
In most cases where the binomial distribution applies, base probability p is the parameter of interest, whereas the number of of trials is known beforehand and therefore does not require estimation. For that reason, the binomial distribution is commonly taken as a one-parameter distribution. When discussing the binomial in more detail, we will learn that p determines the location of the distribution, as well as how widely it is dispersed (preview Figure XY).

The distribution that approximated the IQs is called the Gaussian distribution (or Normal). The Gaussian distribution function takes two parameters, *mu* determines the location of the distribution, say average IQ being 98, 100 or 102 and *sigma* which gives the dispersion, independently (preview Figure XY).

3.4.2.3 Location and dispersion

Location and *dispersion* are two immediate properties of plotted distributions that have intuitive interpretations. Location of a distribution usually reflects where the most typical values come to lie (100 in the IQ example). When an experimenter asks for the difference of two designs in ToT, this is purely about location. Dispersion can either represent *uncertainty* or *variation* in a population, depending on the research design and statistical model. The most common interpretation is uncertainty. The basic problem with dispersion is that spreading out a distribution influences *how typical* the most typical values are. The fake IQ data basically is a perfect Gaussian distribution with a mean of 100 and a standard deviation of 15. The density of this distribution at an IQ of 100 is 0.027. If IQs had a standard deviation of 30, the density at 100 would fall 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 dispersion is low.

```
data_frame(IQ = 0:200,
           density = dnorm(IQ, 100, 30)) %>%
  ggplot(aes(x = IQ, y = density)) +
  geom_area()
```



The perspective of uncertainty routinely occurs in the experimental comparison of conditions, e.g. design A compared to design B. What causes experimenters worry is when the *residual distributions* in their models is widely spread. Roughly speaking, residuals are the variation that is not predicted by the model. The source of this variation is unknown and usually called *measurement error*. It resides in the realm of the SMURFs. With stronger measurement error dispersion, the two estimated locations get less certainty assigned, which blurs the difference between the two.

The second perspective on dispersion is that it indicates *variation by a known source*. Frequently, this source is differences between persons. The IQ is an extreme example of this, as these tests are purposefully designed to have the desired distribution. In chapter [LMM] we will encounter several sources of variation, but I am really concerned about human variation, mostly. Commonly, experimental researchers are obsessed by differences in location, which to my mind confuses "the most typical" with "in general". Only when variation by participants is low, this gradually becomes the same. We will re-encounter this idea when turning to multi-level models.

Most distributions routinely used in statistics have one or two parameters. Generally, if there is one parameter this determines both, location and dispersion, whereas two-parameter distributions can vary location and dispersion independently, to some extent. The Gaussian distribution is a special case

as μ purely does location, whereas σ is just dispersion. With common two-parametric distributions, both parameters influence location and dispersion in more or less twisted ways. For example, mean and variance of a two-parametric binomial distributions both depend on chance of success p and number of trials k, as $\mathrm{M} = pk$ and $\mathrm{Var} = kp(1-p)$.

3.4.2.4 Range of support and skewness [###]

In this book I advocate the thoughtful choice of distributions rather than doing batteries of goodness-of-fit to confirm that one of them, the Gaussian, is an adequate approximation. It usual is trivial 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. Statistical distributions can be classified the same way: having no bounds (Gaussian, t), one bound (usually the lower, Poisson, exponential) or two bounds (binomial, beta).

3.4.2.5 Data generating process

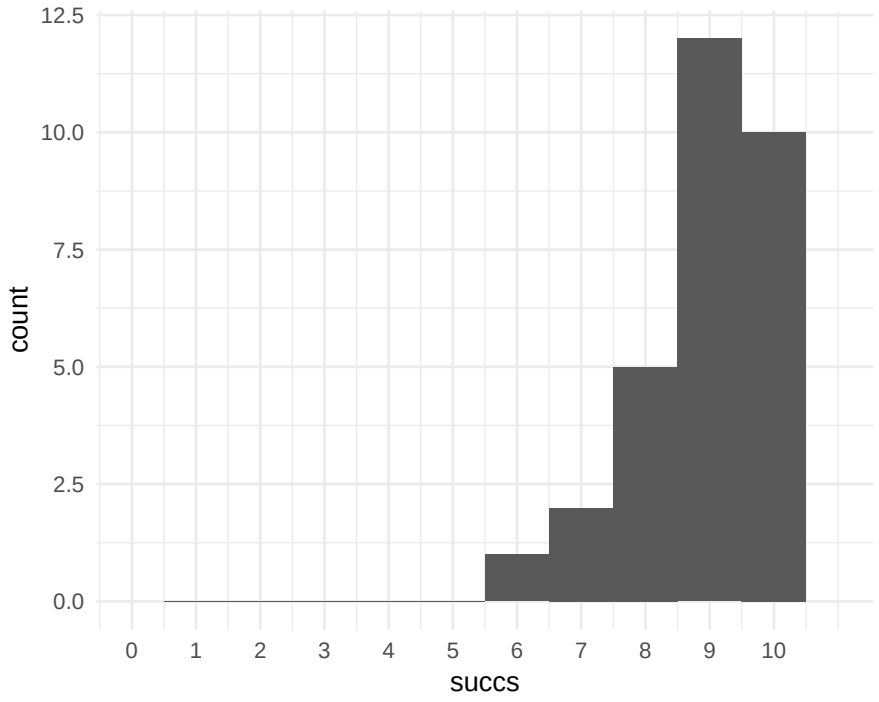
Many dozens of PMFs and PDFs are known in statistical science and are candidates to choose from. First orientation grounds on superficial characteristics of measures, such as discrete/continuous or range, but that is sometimes not sufficient. For example, the pattern of randomness in three-tasks falls 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 severely mistaken statistical models.

For most distributions, strict mathematical definitions exist for under which circumstances randomness takes this particular pattern. 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 . . . . This is particularly the case for the canonical four random distributions that follow. Why are these canonical? The pragmatic answer is: they cover the basic types of measures: chance of success in a number of trials (binomial), counting (Poisson) and continuous measures (exponential, Gaussian).

3.4.2.6 Binomial distributions

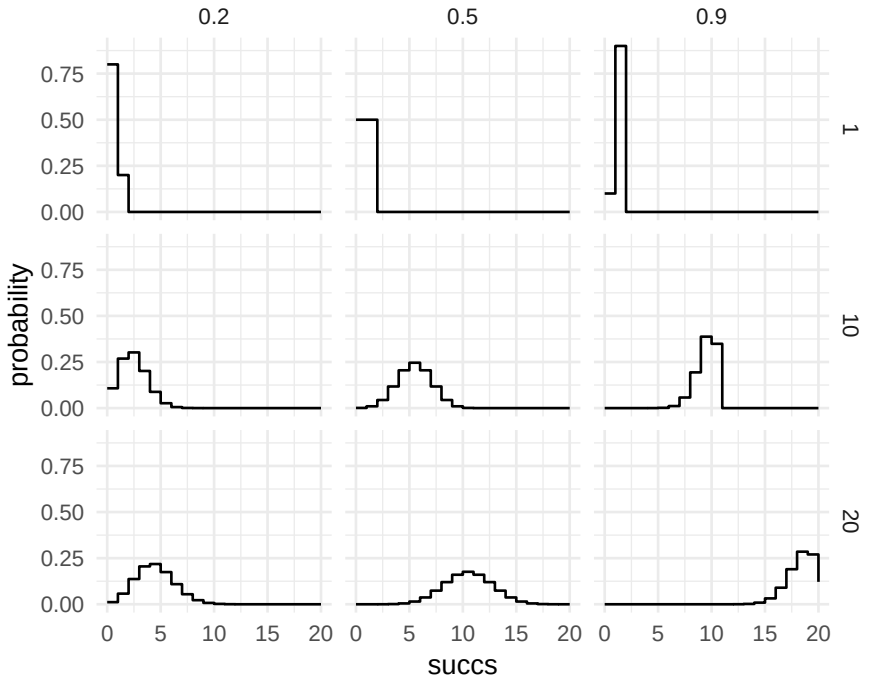
A very basic performance variable in design research is task success. Think of devices in high risk sitations such as medical infusion pumps in surgery. These devices are remarkably simple, giving a medication at a certain rate into the bloodstream for a given time. Yet, they are operated by humans under 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 tpo 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)

data_frame(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 modelling, like on/off, red/blue, male/female. Imagine, Jane's big boss needs a catchy phrase for an investor meeting. Together they decide that the return rate of customers could be a good measure, translating into a statement such as *eighty percent of customers come back*. To prove (or disprove) the claim, Jane uses the customer data base and divides all individuals into two groups: those who have precisely one record and those who returned (no matter how many times). This process results in a distribution, 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. Examples are given in the first row of the figure.

```
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)
```



A Binomial distributions has 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. With larger p the distribution moves to the right. The mean of Binomial distributions is the probability scaled by number of attempts, $M = kp$. Logically, there cannot be more successes then k, but with larger k the distribution gets wider. The variance is the odds scaled by number of attempts, $\text{Var} = kp(1-p)$. As mean and variance depend on the exact same parameters, they cannot be set independently. In fact, the relation $Var = M(1-p)$ 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 hierarchical distributions, where the parameter p itself is distributed, rather than fixed. A common distribution for p is the *beta* distribution and the *logitnormal* distribution is an alternative.

The Binomial distribution has two boundaries, zero below and number of attempts k above. While a lower boundary of zero is often natural, one cannot always speak of a number of attempts. For example, the number of times a customer returns to the car rental website does not yield a natural interpretation of number of attempts. Rather, one could imagine the situation as

that any moment is an opportunity to hire a car. At the same time, every single moment has a very, very small chance that a car is hired, indeed. Under these conditions, an infinite (or painstakingly large) number of opportunities and a very low rate, the random pattern is neatly summarized by *Poisson distributions*.

3.4.2.7 Poisson distributions

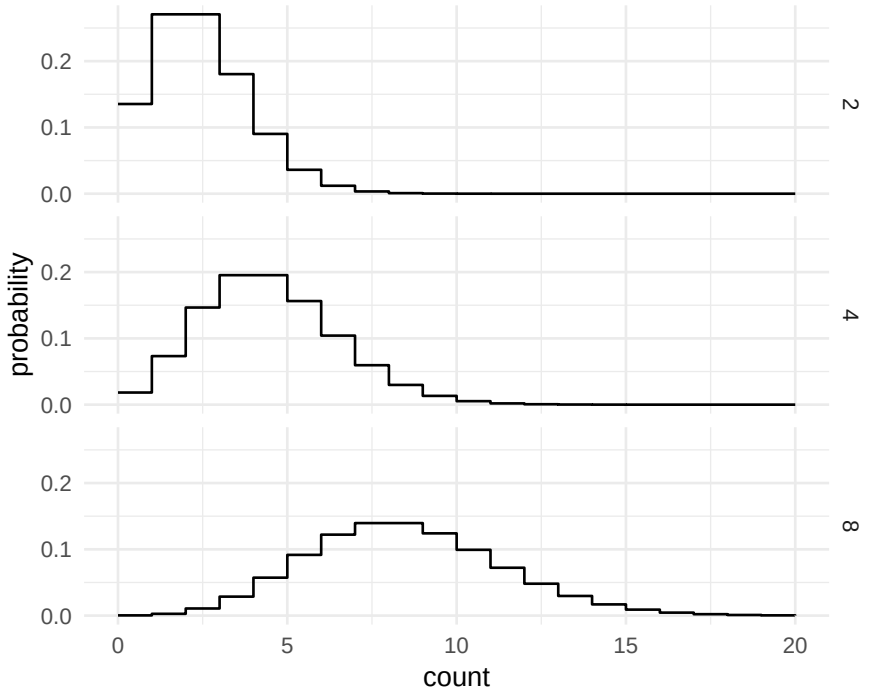
Some counting processes have no natural upper limit like the number of trials in a test. In design research, a number of measures are such *unbounds counts*:

- number of erronous actions
- frequency of returns
- behavioural events, e.g. showing explorative behaviour
- physiological events, such as number of peaks in galvanic skin response

These measures can often be modelled 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. The Poisson distribution is a so called limiting case of the binomial distributions, with infinite k and infinitely small p. Of course, such a situation is completely ideal. Yet, Poisson distributions fit such situations well enough.

Poisson distributions possess only one parameter λ (lambda), that is strictly positive and determines mean and variance of the distribution alike: $\lambda = M = \text{Var}$. As a matter of fact, there cannot be massively dispersed distributions close to zero, nor narrow ones in the far. Owing to the lower boundary, Poisson distributions are *asymmetric*, with the left tail always being steeper. Higher λs push the distribution away from the boundary and the skew diminishes. It is commonly practiced to approximate counts in the high numbers by *normal distributions*.

```
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~.)
```

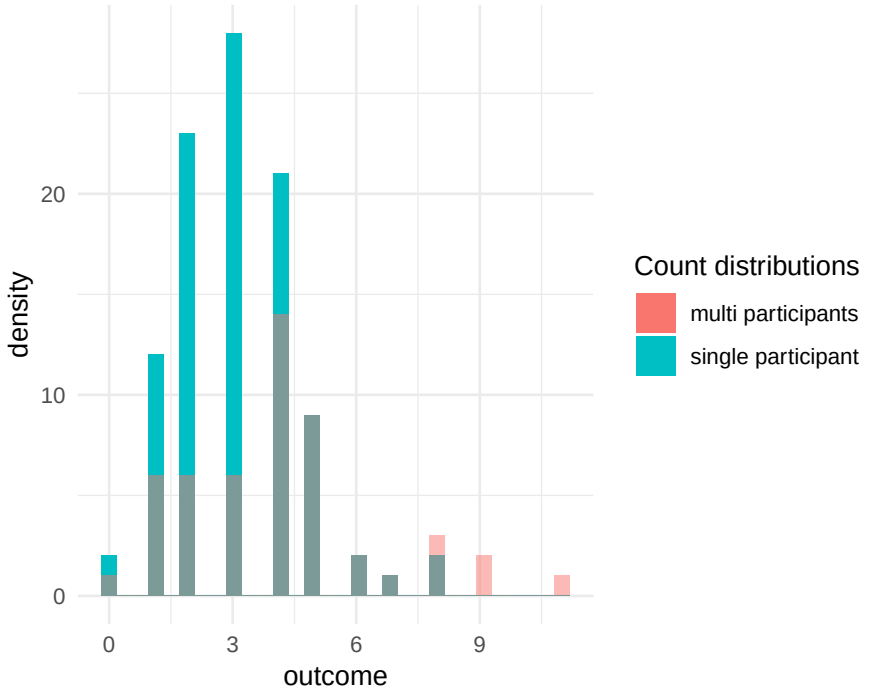


The linkage between mean and variance is very strict. Only a certain amount of randomness can be contained. If there is more randomness, and that is almost certainly so, Poisson distributions are not appropriate. One speaks of *overdispersion* in such a case.

Consider a very simple 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 gane is made to easy. In this experiment, only one player is recorded, and in wonderful ways this player never suffers from fatigue, nor does he get 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, indeed, follow a Poisson distribution, as in the figure below.

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 p. The extra variation is seen in the wider of the two distributions.

```
## FIXME



data_frame(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")
```



Poisson distributions' lower boundary can cause trouble: the measure at hand is truly required to include the lower bound. A person can perform a sequence with no errors, catch zero items or have no friends on facebook. But, you cannot complete an interaction sequence in zero steps or have a conversation with less than two statements. Fortunately, once a count measure has a lower boundary right of zero, the offset is often available, such as the minimum

necessary steps to complete a task. In such a case, the number of errornous steps can be derived and used as a measure, instead:

$$\#\text{errors} = \#\text{steps} - \#\text{neccessary steps}$$

Another lower bound problem arises, when there are hurdles. In traffic research, the frequency of use public transport certainly is an interesting variable. A straight-forward assessment would be to ask bus passengers "How many times have you taken the bus the last five days?". This clearly is a count measure, but it cannot be zero, because the person is sitting in the bus right now. This could be solved by a more inclusive form of inquiry, such as approaching random households. But, the problem is deeper: actually, the whole population is of two classes, those who use public transport and those who don't.

3.4.2.8 Exponential distribution [TBC]

Exponential distributions apply for measures of duration. Exponential distributions have the same generating process as Poisson distributions, except, that the *duration for an event to happen* is the variable of interest, rather than events in a given time. The same idealized conditions of a completely unaffected subway smurfer player and constant catchability of items, the duration between any two catches is exponentially distributed. In more general, the chance for an event to happen is the same at any moment, completely independent of how long one has been waiting for it. For this property, the exponential distribution is called *memoryless*.

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 two reasons: first, the exponential distribution shares with Poisson, that it does not allow variance between participants. Second, the distribution always starts at zero, whereas human reactions always include some basic processing, and be this just the velocity of signals passing nerve cells, which is far below speed of sound (in air).

3.4.2.9 Gamma distribution [TBD]

3.4.2.10 Normal distributions

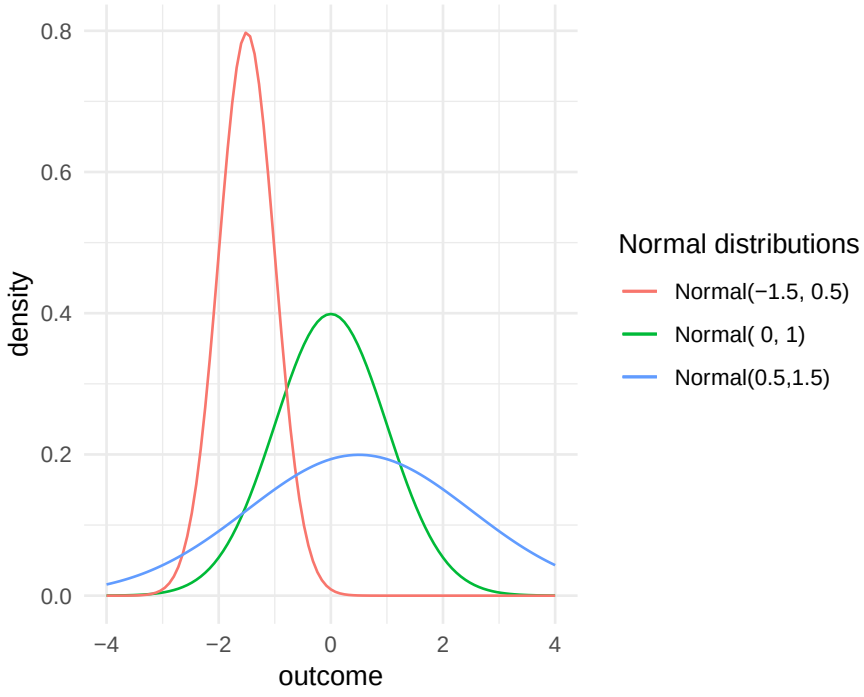
The best known distributions are *normal distributions* or *Gaussian distributions*. These distributions arise mathematically under the assumption of a

myriad of small unrelated 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.

Normal distributions have two parameters: μ marks the center and mean of the distribution. The linear models introduced later are aiming at predicting μ. 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 wider.

Different to Poisson and Binomial distributions, mean and variance of the distribution can be set independently and overdispersion is never an issue.

```
ggplot(data.frame(x = c(-4, 4)), aes(x = x)) +
  stat_function(fun = dnorm, args = list(mean = 0, sd = 1), mapping = aes(colour = "Normal(
  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")
```



Normal distributions have the compelling interpretation of summarizing the effect of SMURFs. They serve to capture randomness in a broad class of regression models and other statistical approaches. The problem with normal

distributions is that they only capture the pattern of randomness under two assumption. The first assumption is that the outcome is continuous. While that holds for duration as a measure of performance, it would not hold for counting the errors a user makes. The second assumption is that the SMURFs are truly additive, like the forces add up, when two pool balls collide. This appears subtle at first, but it has the far reaching consequence that the outcome variable must have an infinite range in both directions, which is impossible.

The normal distribution is called "normal", because people normally use it. Of course not. It gots its name for a deeper reason, commonly known (and held with awe) as 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 normal distribution. In practice, infinity is relative. In some cases, it is reasonable to trade in some fidelity for convenience and good approximations make effective statisticians. As a general rule, the normal distribution 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. However, these rules are no guarantee and careful model criticism is essential.

Measurement is prime and specialized (non-central-limit) distributions remain the first recommendation for capturing measurement errors. The true salvation of normal distributions is their application in *multi-level models*. While last century statistics was reigned by questions of location, and variance considered nuisance, new statistics care for variation. Most notably, amount of variation in a population is added as a central idea in multi-level modelling, which is commonly referred to as *random effects*. These models can become highly complex and convenience is needed more than ever. Normal distributions tie things together in multi-level models, as they keep location and dispersion apart, tidy.

The dilemma is then solved with the introduction of *generalized linear models*, which is a framework for using linear models with appropriate error distributions. Fortunately, MLM and GLM work seemlessly together. With MLM we can conveniently build graceful likelihood models, using normal distributions for populations. The GLM part is a thin layer to get the measurement scale right and choose the right error distribution, just like a looking glass.

3.4.2.11 t distribution [TBD]

3.4.2.12 Exercises:

1. Google it: speed of sound and signal velocity in nerve cells.

3.5 Bayesian estimation

Frequentist statistics falls short on recognizing that research is incremental. Bayesian statistics embraces the idea of gradually increase in certainty. Why has it not been adopted earlier? The reason is it was unfeasible. The innocent multiplication of prior and likelihood is a complex integral, which in most cases has no analytic solution. If you have enjoyed a classic statistics education, you may remember that the computation of sum of squares (explained and residual) can be done by paper and pencil in reasonable time. And that is precisely how statistical computations has been performed before the advent of electronic computing machinery. In the frequentist statistical framework (some call it a zoo), ingenious mathematicians have developed procedures that were rather easy to compute. That made statistical data analysis possible in those times. It came at costs, though:

1. procedures make more or less strong assumptions, limiting their applicability. Procedures become islands.
2. procedures are asymptotically accurate with inference being accurate at large sample sizes only
3. common researchers do not understand crucial elements, for example deriving the F distribution

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 analytical 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, or the Newton-Rapson optimizer can be used to find the maximum likelihood estimate. However, these procedures are too limited as they fail for highly multidimensional problems as they are common in Linear Mixed-Effects Models. Moreover, they do not allow to approximate integrals.

Most Bayesian estimation engines these days ground on a numerical procedure called Markov-Chain Monte-Carlo sampling. The method differs from the earlier mentioned in that it basically is a random number generator. 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 rainfall case above, are estimated within a few minutes. Linear-mixed effects models, which we will encounter later in this book, can take hours and large psychometric models (which are beyond the scope), can take up to a few days.

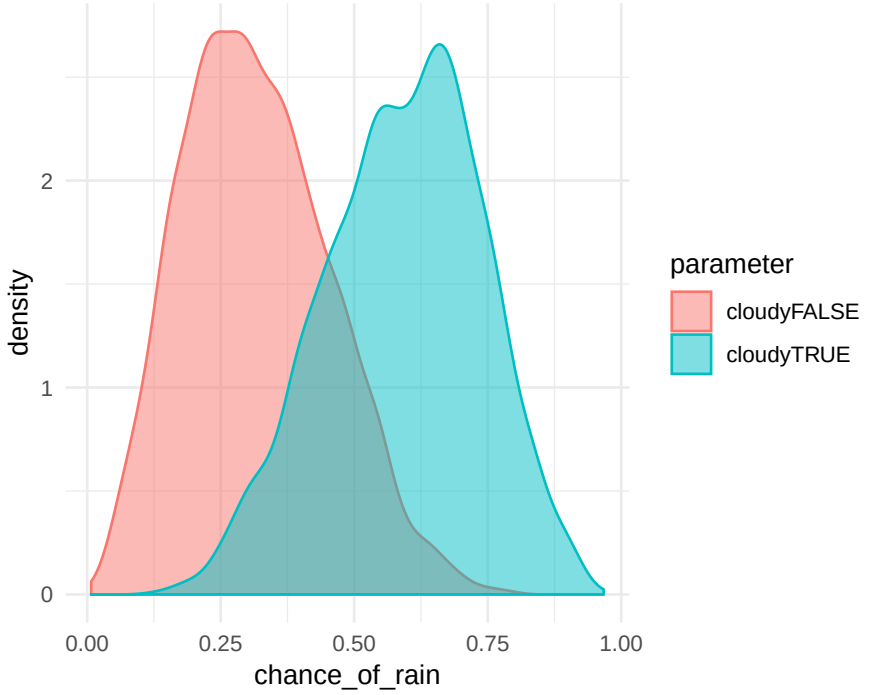
Still, the MCMC algorithm not only delivers some accurate point estimates, it produces the full posterior distribution. This lets us characterize a parameters magnitude and degree of (un)certainty. Let's run an analysis on the 20 rainfall observations to see how this happens.

```
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)
```




```
detach(Rainfall)
```

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 is the same as the *maximum likelihood estimate* when prior knowledge is absent.
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 make non-standard evaluations on the posterior distribution, for example: How certain is it that $P(rain|cloudy) < 0.7$? We'll demonstrate the use of this in the next section.

Coming back to MCMC: how is this distribution actually produced. In plain words, MCMC makes a random walk through parameter space. Regions where the true value is more likely to be are just visited more often. The posterior distribution plots above are actually just frequency plots.

[ILLUSTRATE RANDOM WALK]

3.6 What is wrong with classic statistics? [TBC]

The p-value [...] Imagine giant replication study that showed that 35% of published psychology studies are not replicable. In fact, such a study exists and it has far-reaching consequences. [...] [REF] shows that rejection bias is one reason: studies that have not reached the *magic .05 level* of significance, have a lower chance of publication. [REF] sees as a reason that author's implicitly trick the system by *the garden of forking paths* strategy. The research project collects an array of variables, followed by a fishing expediture. Theory is conventiently considered last, but written up in a pseudo-a prior way.

4

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.

4.1 Setting up the R environment

First, we make sure that you have 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"

4.1.1 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.

4.1.2 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.

4.1.3 A first statistical program

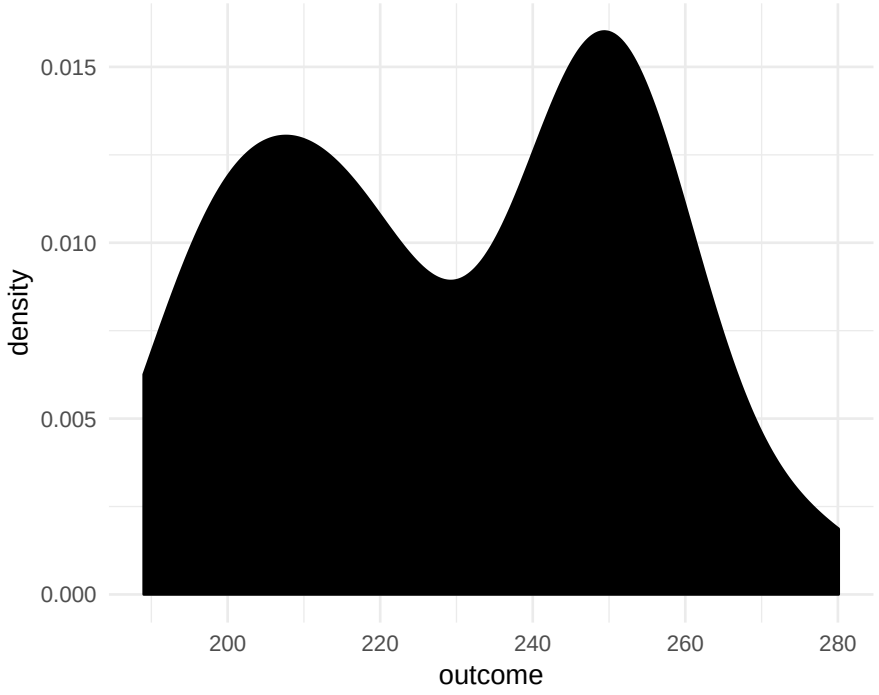
After you have set up your R environment, you are ready to run your first R program (you will not yet understand all the code, but as you proceed with this book, all will become clear):

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)

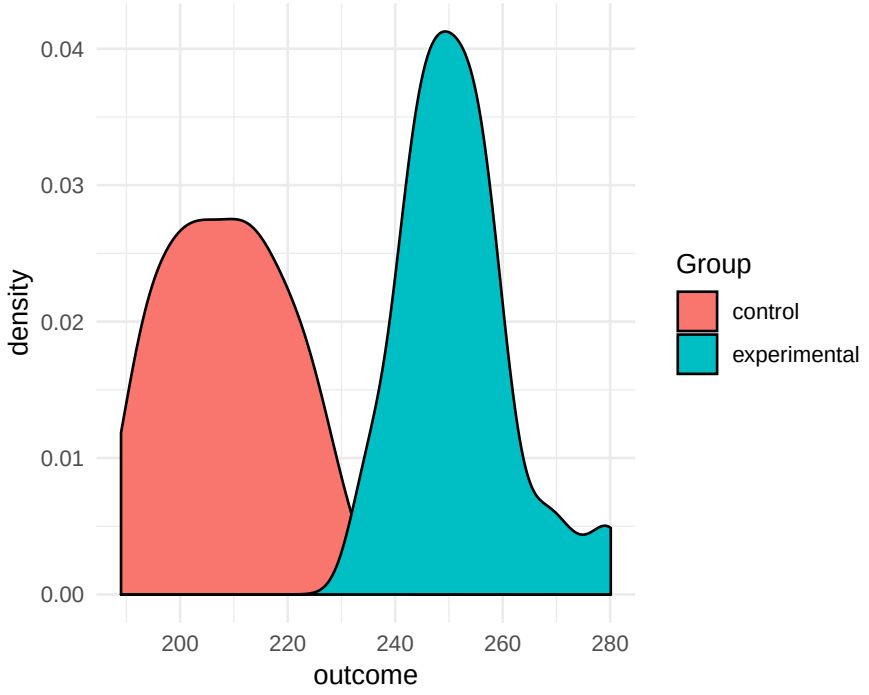
```
## Simulation of a data set with 100 participants
## in two between-subject conditions
N <- 100
levels <- c("control","experimental")
Group <- rep(levels, N/2)
age <- round(runif(N, 18, 35),0)
outcome <- rnorm(N, 42 * 5, 10) + (Group == "experimental") * 42
Experiment <- data_frame(Group, age, outcome)

Experiment %>% sample_n(8)
```

Group	age	outcome
experimental	32	250
experimental	25	247
experimental	35	242
control	32	217
control	26	192
experimental	19	250
control	27	189
control	25	212

```
## 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	211.9	209.1	214.7
Groupexperimental	38.5	34.6	42.3

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.

4.1.4 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.

4.2 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 [CLM], 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 <- data_frame(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] 1 0 0 0 1 0 0 1 0 0 0 1 0 1 0 0 0 0 0 0 0 0 1 0 0
##
## [[2]]
##  [1] 0 0 0 1 1 0 0 1 0 0 1 1 1 0 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 0 1 1 0 0 0
## [36] 1
##
## [[3]]
```

```
##  [1] 0 0 1 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0
## [36] 0 0 0 0 1 0 0 0 0 0 1 0 0 1
```

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.

4.2.1 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] "<000000004213F610>"
```

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

```
## [1] "<00000000421B0930>"
```

4.2.2 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] 1.96
```

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.*

4.2.3 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 use in [LM: dummy variables].

```
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 data manipulation functions, like `mean` are aware of missing values and act conservatively in their presence: the programmer has to explicitly ask to ignore any `NA` values. While this appears slightly inconvenient, it helps to catch errors early in data analysis, and creates better transparency.

```
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.

4.2.4 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%
## 0.947 2.963
```

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

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

D_2 = data_frame(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.

4.2.5 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.

Data frames are objects. By just calling a data frame it gets printed to the screen. This is a good moment to reflect on one idiosyncrasy in R, a feature, of course. R is primarily used interactively, which has the immense advantage that the programmer can incrementally build the data analysis. This implies, that the programmer often wants to quickly check what currently is in a variable. Now, observe what happens when we call `Experiment`:

```
Experiment
```

Group	age	outcome
control	23	199
experimental	35	268
control	27	204
experimental	31	256
control	23	196
experimental	25	247
control	27	206
experimental	23	248
control	30	224
experimental	20	242
control	28	229
experimental	23	245
control	21	199
experimental	22	235
control	20	217
experimental	35	268
control	26	208
experimental	24	251
control	24	202
experimental	22	234
control	26	192
experimental	34	242
control	22	222
experimental	19	253
control	27	189
experimental	30	241
control	32	200
experimental	30	259
control	32	195
experimental	28	242
control	26	213
experimental	20	257
control	34	212
experimental	24	257
control	22	206
experimental	25	259
control	30	205
experimental	19	279
control	27	193
experimental	22	251
control	20	192
experimental	22	253
control	34	222
experimental	19	250
control	24	224
experimental	28	262
control	21	220
experimental	25	250
control	21	199
experimental	19	245
control	21	197
experimental	19	273
control	34	203
experimental	22	247
control	27	212

This is useful information printed to the screen, we see sample size, names of objects and their classes, and the first ten observations as examples. Obviously, this is not the data itself, but some sort of summary. It would be a complete disaster, if R would pass this information on when the call is part of an assignment or other operation on the data, for example: `NewExp <- Experiment`. Apparently, R is aware of whether a called object is part of an operation, or purely for the programmers eyes. For any object, if *called to the console in an intercative session*, R silently uses the `print` function on the object. The following would give precisely the same results as above.

```
print(Experiment)
```

Such `print` functions exist for dozens of classes of objects. They represent what the developers thought would be the most salient or logical thing to print in an interactive session. By virtue of the object system, R finds the right print function for the object at hand.

However, when working interactively, most of the time you do a data transformation and assign it to a new variable, thus you check it immediately. As the assignment is always silent (unless there occurs an error), you are forced to type a print statement afterwards. There is shortcut for that: if you embrace the assignment statement in brackets, the result will be printed to the screen. As convenient as this is, especially when creating longer data transformation chains, it introduces some visual clutter. In this book I use it sparingly, at most.

The dedicated print function does what way back a developer thought would be the most useful information in most cases. Depending on the situation, it can provide too much, too little or just the wrong information. For example: when simulating a data set, like in [First Exercise], we want to check whether the resulting data frame has the desired length and whether the variables have the correct class. The standard implementation of the print command, dumps the whole data frame to the screen, which can be much more than ever needed. The newer dplyr implementation `data_frame` (from the tidyverse, also called *tibble*) you have seen in action already, is much better suited for this situation, as it displays dimensions and variable classes. In contrast, if the job is to spot whether there are many missing values (`NA`), the classic print performs better.

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

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

```
##     Group                age          outcome
##  Length:100         Min.   :18.0   Min.   :189
##  Class :character   1st Qu.:22.0   1st Qu.:209
##  Mode  :character   Median :25.0   Median :232
##                     Mean   :25.7   Mean   :230
##                     3rd Qu.:30.0   3rd Qu.:250
##                     Max.   :35.0   Max.   :280
```

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

```
str(Experiment)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    100 obs. of  3 variables:
##  $ Group  : chr  "control" "experimental" "control" "experimental" ...
##  $ age    : num  23 35 27 31 23 25 27 23 30 20 ...
##  $ outcome: num  199 268 204 256 196 ...
```

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] 230
```

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]
```

outcome
199

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

age
23
35
27

```
Experiment[  1,  ]
```

Group	age	outcome
control	23	199

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 `data_frame`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
```

Group	age
control	23
experimental	35

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

Group	age	outcome
control	23	199

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

```
##   [1] "control"      "experimental" "control"      "experimental"
##   [5] "control"      "experimental" "control"      "experimental"
##   [9] "control"      "experimental" "control"      "experimental"
##  [13] "control"      "experimental" "control"      "experimental"
##  [17] "control"      "experimental" "control"      "experimental"
```

```
##  [21] "control"      "experimental" "control"      "experimental"
##  [25] "control"      "experimental" "control"      "experimental"
##  [29] "control"      "experimental" "control"      "experimental"
##  [33] "control"      "experimental" "control"      "experimental"
##  [37] "control"      "experimental" "control"      "experimental"
##  [41] "control"      "experimental" "control"      "experimental"
##  [45] "control"      "experimental" "control"      "experimental"
##  [49] "control"      "experimental" "control"      "experimental"
##  [53] "control"      "experimental" "control"      "experimental"
##  [57] "control"      "experimental" "control"      "experimental"
##  [61] "control"      "experimental" "control"      "experimental"
##  [65] "control"      "experimental" "control"      "experimental"
##  [69] "control"      "experimental" "control"      "experimental"
##  [73] "control"      "experimental" "control"      "experimental"
##  [77] "control"      "experimental" "control"      "experimental"
##  [81] "control"      "experimental" "control"      "experimental"
##  [85] "control"      "experimental" "control"      "experimental"
##  [89] "control"      "experimental" "control"      "experimental"
##  [93] "control"      "experimental" "control"      "experimental"
##  [97] "control"      "experimental" "control"      "experimental"
```

Predictability and a few other useful tweaks made me prefer `data_frame` 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_data_frame()
```

While `data_frame[]` 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 `data_frame` 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)
```

Group	age	outcome
control	-99	199
experimental	35	660
control	-99	204
experimental	31	256
control	23	196
experimental	25	987

```
## 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.

4.2.6 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.

4.2.7 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()
```

Obs	Part	Task	Design	Gender	Education	age	Far_sighted	Small_font	ToT	clicks	returns	rating
29	29	1	A	F	Low	63	FALSE	FALSE	196	8	2	6
51	51	1	A	F	High	32	FALSE	FALSE	171	10	1	5
113	113	1	B	M	High	61	FALSE	TRUE	145	5	2	4
133	133	1	B	F	Middle	32	TRUE	TRUE	260	14	4	6
166	166	1	B	F	Low	65	TRUE	TRUE	304	21	6	7
177	177	1	B	F	High	33	TRUE	TRUE	174	10	2	5
187	187	1	B	F	High	27	FALSE	TRUE	114	9	3	4
195	195	1	B	M	Low	22	FALSE	TRUE	150	8	1	4

```
detach(BrowsingAB)

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

Obs	Part	font_size	font_color	mu	ToT
5	5	10pt	gray	60	62.0
7	7	10pt	gray	60	67.6
8	8	12pt	gray	48	47.5
10	10	12pt	gray	48	47.7
12	12	12pt	gray	48	59.4
13	13	10pt	gray	60	53.1
18	18	12pt	gray	48	34.7
36	36	12pt	black	46	37.4

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
```

Obs	Part	font_size	font_color	mu	ToT
3	3	10pt	gray	60	61.8
7	7	10pt	gray	60	67.6
8	8	12pt	gray	48	47.5
14	14	12pt	gray	48	46.6
20	20	12pt	gray	48	54.6
24	24	12pt	black	46	52.1
28	28	12pt	black	46	37.2
40	40	12pt	black	46	46.2

```
simulate(N = 6)  %>% as_tbl_obs() ## simulates first 6 participants only
```

Obs	Part	font_size	font_color	mu	ToT
1	1	10pt	gray	60	66.9
2	2	12pt	gray	48	45.2
3	3	10pt	gray	60	61.8
4	4	12pt	black	46	49.2
5	5	10pt	black	50	52.0
6	6	12pt	black	46	45.5

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 <-
##       data_frame(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: 0x0000000031993230>
```

```
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.

4.2.8 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()
```

Obs	Part	A	B
1	1	4	1
2	2	4	4
3	3	6	3

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. This is shown in chapter [DM]. 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*. 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

4.2.9 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()
```

Obs	Condition	age	outcome
4	experimental	23	248
18	experimental	34	242
22	experimental	30	241
26	experimental	28	242
55	control	30	193
64	experimental	20	238
77	control	34	218
81	control	27	203

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()
```

Obs	Condition	age	outcome	age_cntrd
1	experimental	31	256	5.35
14	experimental	24	251	-1.65
19	control	22	222	-3.65
21	control	27	189	1.35
34	experimental	19	279	-6.65
46	experimental	19	245	-6.65
67	control	18	230	-7.65
94	experimental	35	242	9.35

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	209	11.3
experimental	251	10.2

4.2.10 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:

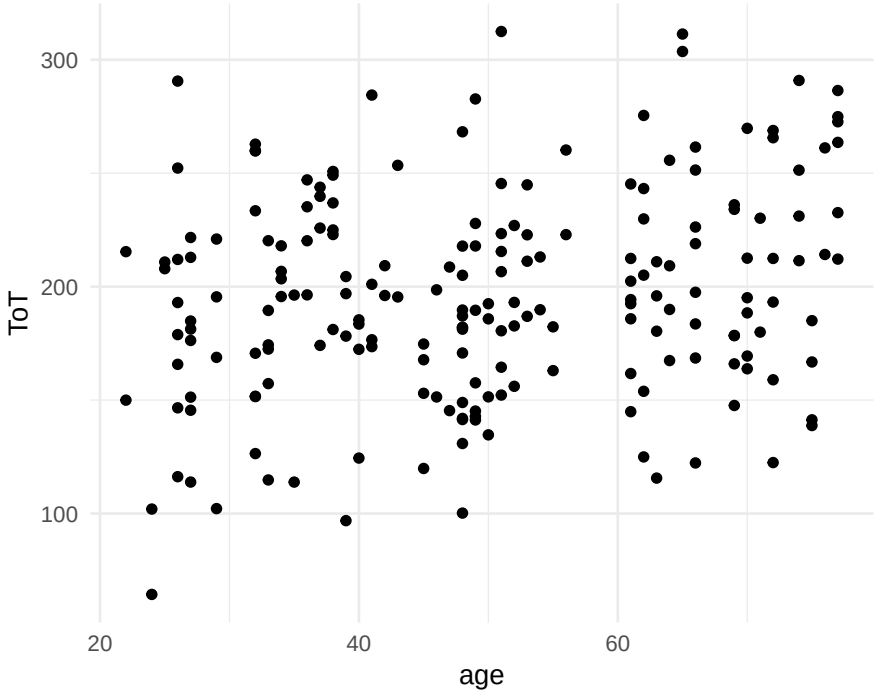
```
attach(BrowsingAB)
BAB1 %>% as_tbl_obs()
```

Obs	Part	Task	Design	Gender	Education	age	Far_sighted	Smartell_To	Flicks	returns	rating
1	1	1	A	M	Middle	49	TRUE	FALSE	190	12	4	6
32	32	1	A	M	Middle	34	TRUE	FALSE	196	12	0	6
46	46	1	A	M	High	61	FALSE	FALSE	245	13	1	6
71	71	1	A	F	High	48	FALSE	FALSE	149	9	3	5
110	110	1	B	M	Low	51	FALSE	TRUE	223	10	2	5
116	116	1	B	M	Middle	75	TRUE	TRUE	185	10	4	6
127	127	1	B	F	High	35	FALSE	TRUE	114	5	1	3
152	152	1	B	F	Low	77	TRUE	TRUE	233	13	2	6

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.

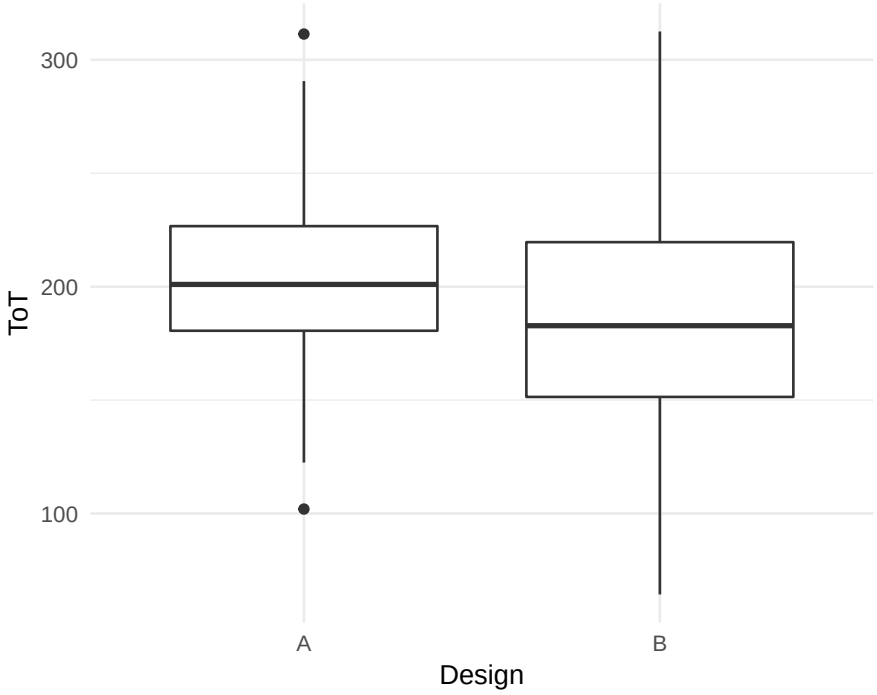
```
aes(x = age, y = ToT)
```

```
## 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:

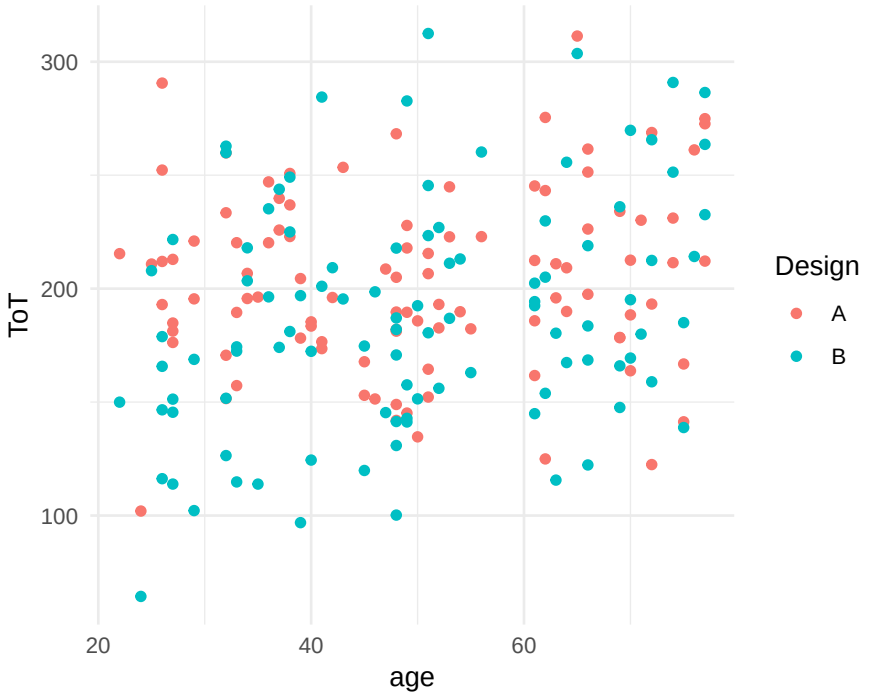
```
BAB1 %>%
  ggplot(aes(x = Design,  y = ToT)) +
  geom_boxplot()
```



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.

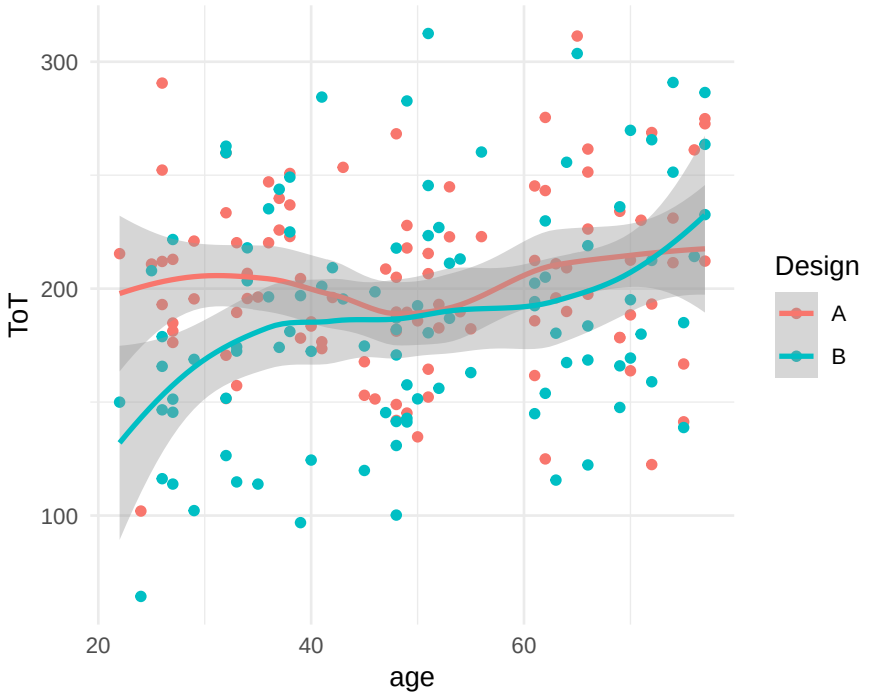
Now, we combine all three variables in one plot: how does the association between ToT and age differ by design? As we have two quantitative variables, we stay with the scatter plot for now. As we intend to separate the groups, we need a property of points to distinguish them. Points offer several additional 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.

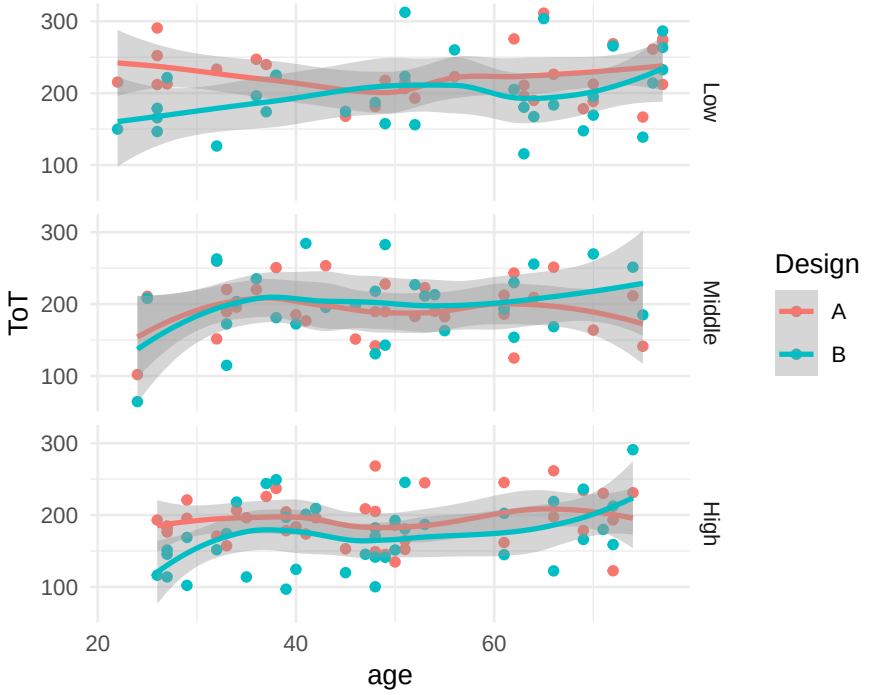
```
BAB1 %>%
  ggplot(aes(x = age,  y = ToT, color = Design)) +
  geom_point() +
  geom_smooth()
```



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

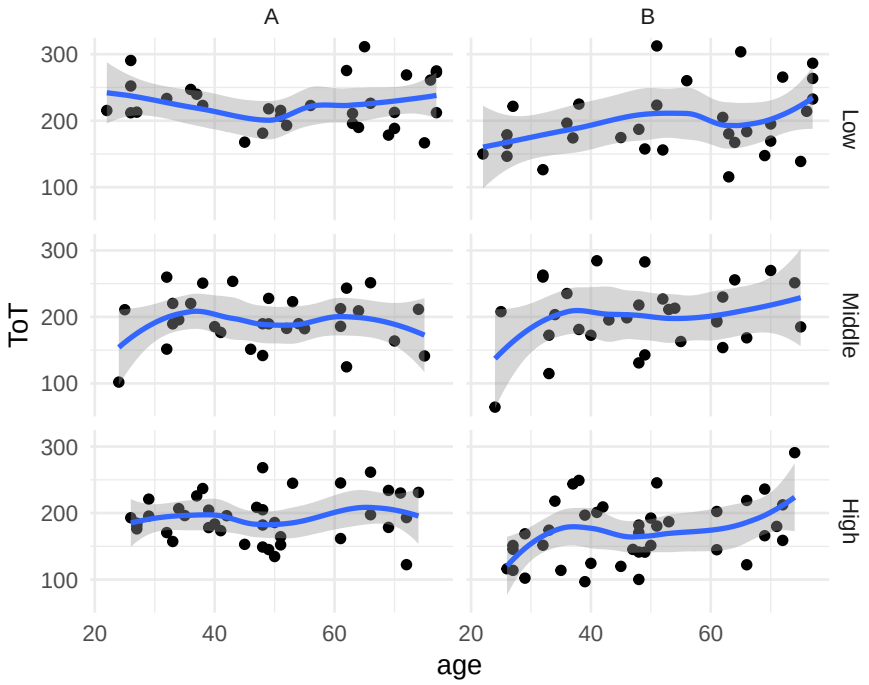
Now that we have represented three variables with properties of geometries, what if we wanted to add a fourth one, say education level? Formally, we could use another aesthetic, say shape of points, to represent it. You can easily imagine that this would no longer result in a clear visual figure. For 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.

4.2.11 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 <-
  data_frame(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 [LM], 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	274.0	266	282.4
ConditionExperimental	-46.1	-58	-34.5

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	206.60	126.1	285.79
age	1.58	-1.2	4.46

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	231.69	199.310	265.46
ConditionExperimental	-46.21	-56.362	-36.43
age	1.51	0.344	2.66

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.

4.2.12 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.

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

````
# Results

In the study we examined

1. whether on average users can rent a car in 99 seconds
2. and if we can say that with sufficient certainty

## Data exploration


```{r}
summary(D)
```

```{r}
summary(D)
```

All variables show good variation.

## Regression

```{r}
M_1 <- stan_lm(ToT ~ 1, data = D)
```

```{r}
coef(M_1) %>% knitr::kable()
```


Our main observations are:

+ the 99 seconds claim is not supported
+ time-on-task measures are not normally distributed
````

Fig. 4.1. A minimal statistical report in markdown

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.

4.2.13 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.

4.2.14 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.

Part II

Models

5

Linear models

5.1 Quantification at work: grand mean models

Reconsider Andrew and Jane. They were faced with the problem that potential competitors could challenge the claim "rent a car in 99 seconds" and drag them to court. More precisely, the question was: "will users on average be able ... ", which is not about individual users, 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 think of it as the "grandmother of all models". Although its is the simplest of all, it is of useful application in design research; here are a few more 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 brake lights of a car must be designed to let the following driver react in a certain time frame

The GMM predicts the expected level of performance when the only thing you know is the population someone is from. Prediction here means that the estimated grand mean we take as a best guess for the population average, that is all realizations we have *not* observed. So, it is a generalization regarding all people we have not invited to the lab, but also potential performance differences by situation, for example the daily shape people are in or their current level of motivation. Just consider the many ways people differ in abilities, experience, strategies, preferences, situations, wishes etc. All these differences may also vary for the same person from day to day and influence performance. All these aspects have not been recorded in the Sec99 study and are therefore

not taken into account for prediction. All linear models capture the unpredicted variation in a separate parameter σ, the *residual standard deviation*. σ measures the amount of variance that is left after subtracting all explanatory variables. Formally, the likelihood and random formulas of the grand mean model are written with the following likelihood and random term:

$$\mu = \beta_0 y_i \sim \text{Norm}(\mu_i, \sigma_\epsilon)$$

The larger σ is, the more questionnable it is that the grand mean is truly *representative* for the population. From the GM we will depart in to directions. First, in the remainder of this chapter, we will add further predictors to the model, for example age of participants or a design condition. These models will still make statements on population averages, although in a more detailed way. In the following chapter on mixed-effects models, individuals will get spot light. Still, what all the advanced models have in common is that they move variance in the outcome variable from the error variance to predictors. An optimist would say: today's error is tomorrow's predictor.

So, when estimating the grand mean model, we estimate the intercept β_0 and the standard deviation of the Gaussian distributed error term $\sigma\epsilon$. In R, the analysis of the 99 seconds problem 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 = Ver20`. As the `stan_glm` command applies to a huge variety of regression model, the desired model needs further specification. For that purpose, R has its own formula language. The formula of the grand mean model is `ToT ~ 1`. Left of the `~` (*tilde*) operator is the outcome variable. The right hand side specifies the deterministic part. The 1 here has nothing to do with the natural number neighboured by 0 and 2. In R's formula language it represents the *intercept*.

```
attach(Sec99)
```

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

```
summary(M_1)
```

```
##
## Model Info:
##
##  function:     stan_glm
##  family:       gaussian [identity]
##  formula:      ToT ~ 1
##  algorithm:    sampling
##  priors:       see help('prior_summary')
```

```
##  sample:       4000 (posterior sample size)
##  observations: 100
##  predictors:   1
##
## Estimates:
##                 mean   sd     2.5%   25%    50%    75%    97.5%
## (Intercept)    106.0    3.1   99.9  103.8  106.0  108.1  112.0
## sigma           31.5    2.3   27.5   29.9   31.3   33.0   36.5
## mean_PPD       106.0    4.5   97.2  103.0  106.0  109.0  114.9
## log-posterior -494.3    1.0 -497.1 -494.7 -494.0 -493.6 -493.3
##
## Diagnostics:
##               mcse Rhat n_eff
## (Intercept)   0.1  1.0  2711
## sigma         0.0  1.0  2709
## mean_PPD      0.1  1.0  3258
## log-posterior 0.0  1.0  1382
##
## For each parameter, mcse is Monte Carlo standard error, n_eff is a crude measure of ef
```

The object `M_1` is the model object created by `stan_glm`. When you call the summary you complex listings that represent different aspects of the regression. These aspects, and more are saved inside the object in a hierarchy of lists. The central result of the estimation is the *posterior distribution (PD)*. It is an array of variables over MCMC runs. We extract the posterior distribution from the model object using the `posterior` command (package: `bayr`).

```
P_1 <- posterior(M_1)
P_1
```

** tbl_post: 4000 samples in 1 chains

model	parameter	type	fixef	entities
M_1	1	fixef	Intercept	1

parameter
sigma_resid

The posterior object identifies itself by telling the number of MCMC samples, and the variables contained in the model. In the case here, there is just the intercept (representing the grand mean) and the standard deviation of errors.

Much of the time a researcher doesn't want to deal with the posterior, directly, but desires a brief summary of location and uncertainty. Coefficient tables are frequently used, just like one one shown below.Coefficient tables report the central tendency of every coefficient, which is an indicator for the 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 coefficient table is produced by the `coef` command of the bayr library:

```
coef(P_1)
```

parameter	type	fixef	center	lower	upper
Intercept	fixef	Intercept	106.0	99.9	112.0
sigma_resid	disp	NA	31.3	27.5	36.5

```
detach(Sec99)
```

5.1.1 Reading coefficient tables

Coefficient tables are the standard way to report regression models. They contain all parameters (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).

The authors of Bayesian books and the various Bayesian libraries have different opinions on what to report in a coefficient table. Most seem to prefer the posterior mode or the median, only some use the mean.

A disadvantage of the *mean* is that it may change, under many 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 **??**.

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% quartile 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 values only changes when the order changes, i.e. a non-monotonous function was applied.

For center estimates I use the posterior median by default, for its simplicity. Researchers who desire downward compatibility with classic regression engines, can easily switch to the mode by using the *estimate* argument. The following uses the `shorth` command form the package `modeest`.

```
attach(Sec99)
```

```
coef(P_1, estimate = modeest::shorth)
```

parameter	type	fixef	center	lower	upper
Intercept	fixef	Intercept	106.0	99.9	112.0
sigma_resid	disp	NA	31.1	27.5	36.5

Expressing the level of certainty of the posterior distribution makes statistics *inferential*. When the posterior is widely spread, you will still bet on values close to the center, but keep your bid low. For the spread of a distribution, the standard deviation may come to mind of some readers. The standard deviation has teh 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. In this book, *2.5% and 97.5% certainty quantiles* are routinely used to form 95% credibility intervals. Again, another method exists to obtain CIs. Some authors prefer to report the *highest posterior interval*, 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.

The `coef` command by default gives 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 `coef`, 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	99.9	112.0
sigma_resid	31.3	27.5	36.5

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 "significant" certainty for the parameter of interest. Furthermore, classic regression engines don't yield and 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 make the case. This is why we should be glad that Bayesian engines report uncertainty on all parameters involved.

It is common practice to not just drop coefficient tables, but explain and interpret them in written form. My suggestion of how to *report regression results* is to simply walk through the table row-by-row and for every parameter make two statements: a quantitative statement based on the central tendency, and an uncertainty statement. In the present case that would be:

1. The *intercept* β_0 is in the region of 106 seconds, which is pretty off the target of 99 seconds.
2. 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:

1. The population mean is rather not representative for the observations, as the standard error is almost one third of it.
2. We can be pretty certain that this is so.

```
detach(Sec99)
```

5.1.2 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* mu_i and predictors. It is common to call predictors with the Greek letter β (beta). If there are more than one predictors, these are marked with subscripts, starting at zero. The "best guess" is called the *expected value* and is denoted with μ (mu_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 a 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.

$$y_i \sim N(\mu_i, \sigma_\epsilon)$$

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 often 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):

$$y_i = \mu_i + \epsilon_i\mu_i = \beta_0\epsilon_i \sim \text{Norm}(0, \sigma_\epsilon)$$

First, it is to say, that the two ways of formula 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 Normal distribution. Residual distributions are a highly useful concept in modelling as they can be used to check a given model. When residuals are such a highly useful concept, 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 right from the start.

5.1.3 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 table. While it is convenient to have a standard procedure it turns out very useful to understand how these estimates came to life. In Bayesian estimation, the *posterior distribution (PD)* is the central point of departure for any such statements. The 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 cases, 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).

In fact, the `stan_glm` command returns a large object that stores, among others, the full random walk. This random walk represent the posterior distribution almost directly. The following code extracts this the posterior distribution from the regression object prints it. When calling the new object (class: tbl_post) directly, it provides a compact summary of all variables in the model, here this is the intercept and the residual standard error.

```
attach(Sec99)
```

```
P_1 <-  posterior(M_1)
P_1
```

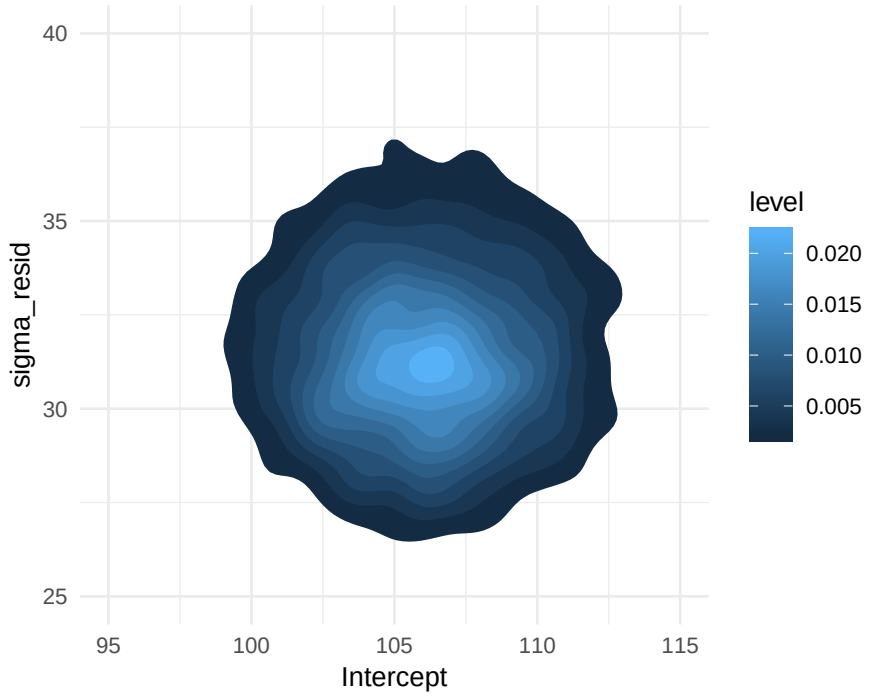
** tbl_post: 4000 samples in 1 chains

model	parameter	type	fixef	entities
M_1	1	fixef	Intercept	1

parameter
sigma_resid

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 do a sufficient job:

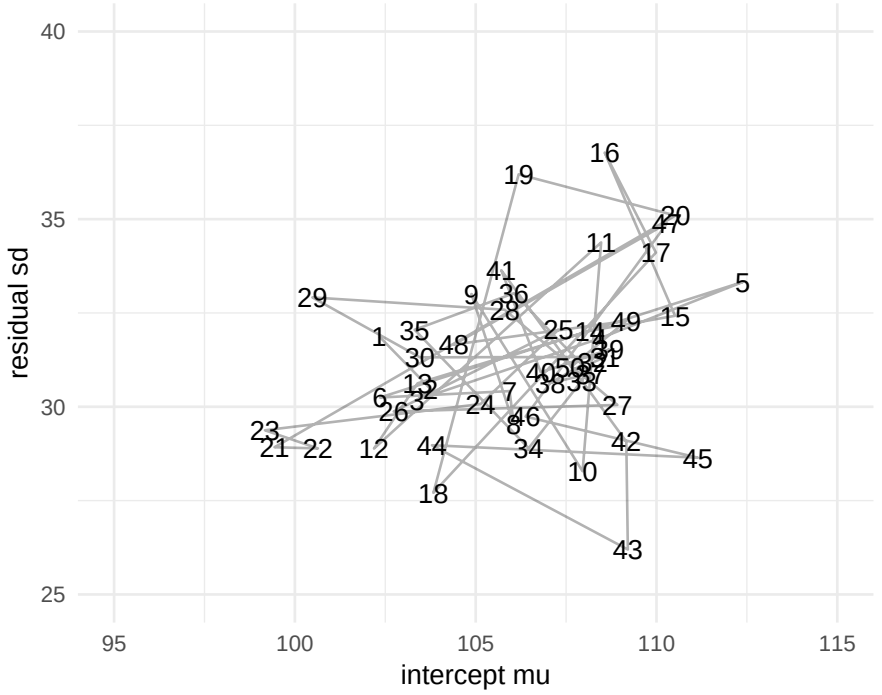
```
P_1 %>%
  select(chain, iter, parameter, value) %>%
  spread(parameter, value) %>%
  ggplot(aes(x = Intercept, y = sigma_resid, fill = ..level..)) +
  stat_density_2d(geom = "polygon") +
  xlim(95, 115) + ylim(25, 40)
```



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, because it is either carried out, or not, and that depends on a bet. If the new target is in a highly likely region, it is carried out with a higher chance. This sounds circular, but it works and, of course, it has been proven mathematically that it works.

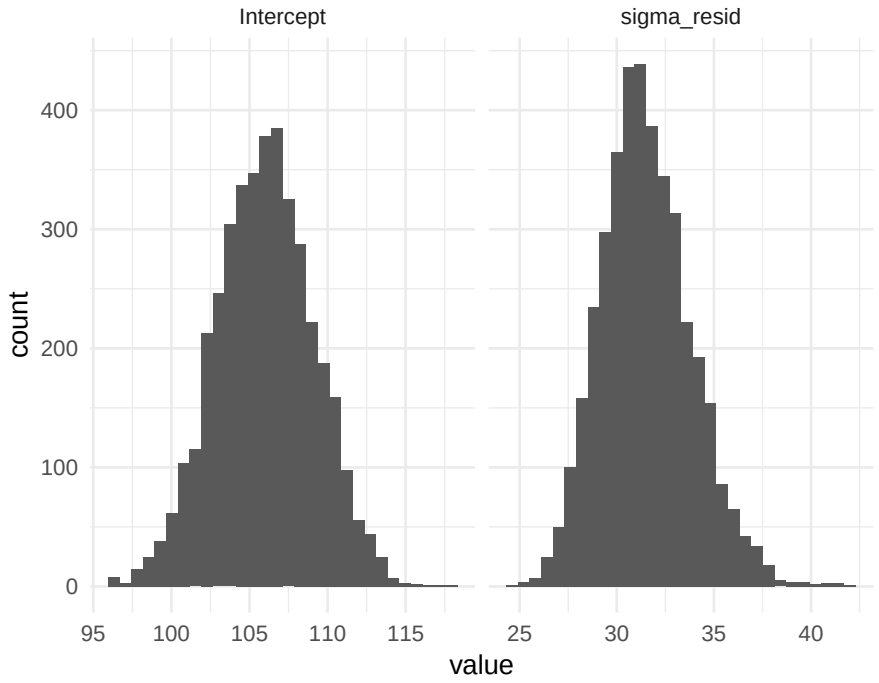
The regression object stores the MCMC results as a long series of positions in parameter space. For any range of interest, it is the relative frequency of visits that represents its certainty. The first 50 hundred steps of the MCMC random walk are shown in @ref(99_seconds_random_walk)‘. Apparently, the 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 a three parameter dimensions, which is difficult to visualize. Multi-level models 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 MPDs is the violin plot, which packs a bunch of densities and therefore can be used for models of higher dimension. Still, for simple models histograms do the job:

```
P_1 %>%
  ggplot(aes(x = value)) +
  geom_histogram() +
  facet_grid(. ~ parameter, scales = "free")
```



In our example, in @ref(99_seconds_post) we can spot that the most likely value for average time-on-task is 106.1. 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 `fixef()` does not answer the question, accurately. 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
0.986	0.054

It turns out that the certainty for average time-on-task above the 99 is an overwhelming 0.986. The alternative claim, that average completion time is better than 111 seconds, has a rather moderate risk to err (0.054).

```
detach(Sec99)
```

5.2 Walk the line: linear regression

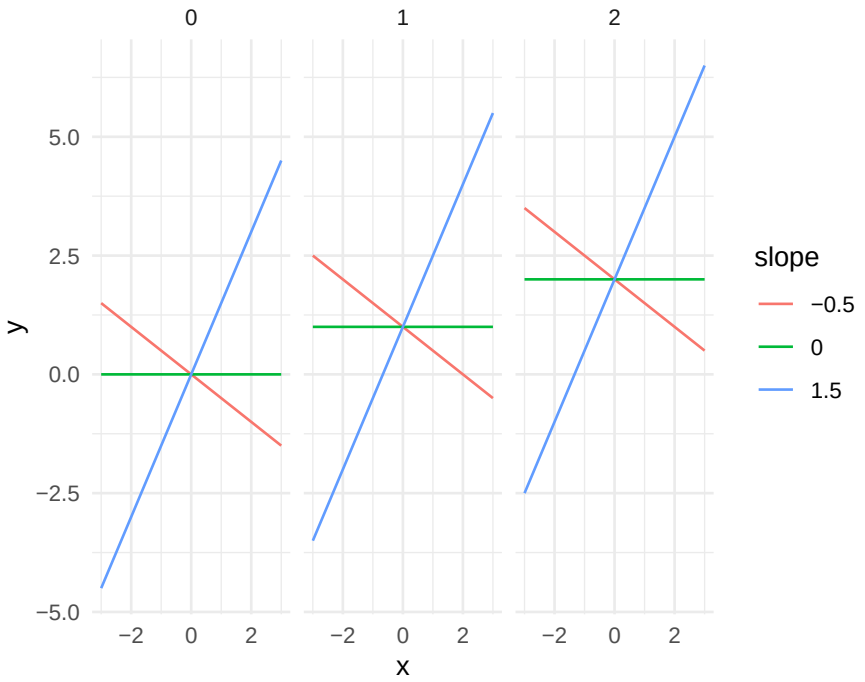
In the previous section we have introduced the mother of all regression models: the grand mean model. It assigns rather coarse predictions, without any real predictors. Routinely, design researchers desire to predict performance based on *metric variables*, such as:

- 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 of font size on reading performance, when all participants are psychology students or 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 for its simplicity. 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 1, the line will raise by this amount when one moves one step to the right.

$$f(x_1 + 1) = \beta_0 + \beta_1 x_{1i} + \beta_1$$

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 being constant over the whole range. This shows that The LRM can be conceived a generalization of the GM model: $\mu_i = \beta_0$.

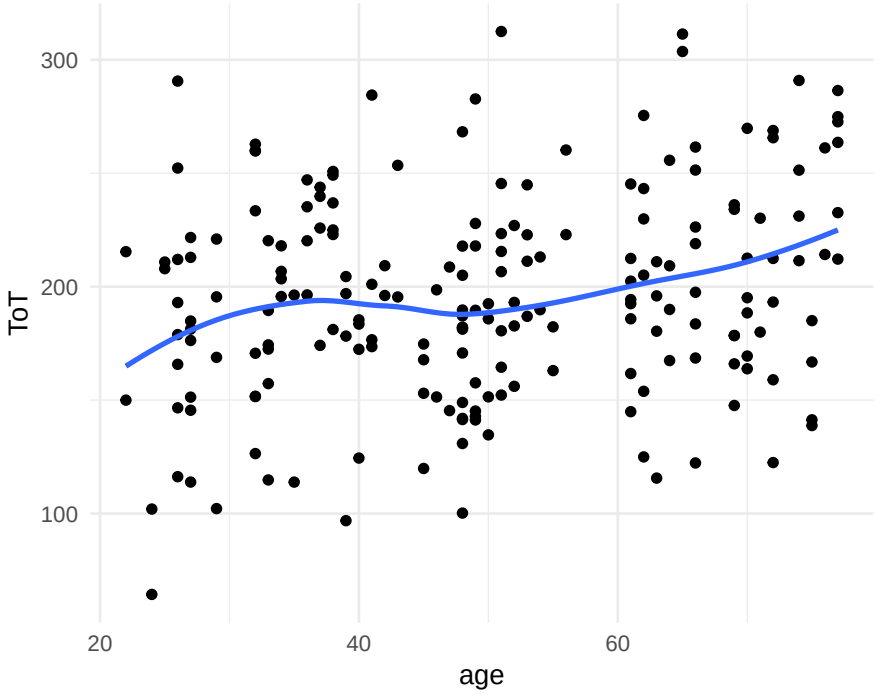
Linear regression gives us the opportunity to discover how ToT can be predicted by age (x_1) in the BrowsingAB case. IN this synthetic experiment, two designs A and B are compared, but this is what we ignore for now. Instead, we ask: are older people slower on the internet? or: is there a linear relationship between age and ToT? The likelihood and random terms of the LRM are:

$$\mu_i = \beta_0 + \beta_1 x_{1i}$$
$$Y_i = N(\mu_i, \sigma)$$

This literally means: with every year of age, ToT increases by the 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 lever fixed at a pivotal point, LOESS is a pipe cleaner. 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)
```

```
G_eda_1 <-
  BAB1 %>%
  ggplot(aes(x = age, y = ToT)) +
  geom_point()+
  geom_smooth(se = F, fullrange = F)

G_eda_1
```



In fact, the BrowsingAB data 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 currently 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, here. Therefore, it is treated as a metric predictor or *covariate*.

```
M_age <-
  BAB1 %>%
  stan_glm(ToT ~ 1 + age,
           data = .)
```

```
T_age <- fixef(M_age)
T_age
```

fixef	center	lower	upper
Intercept	164.090	143.152	185.18
age	0.642	0.234	1.04

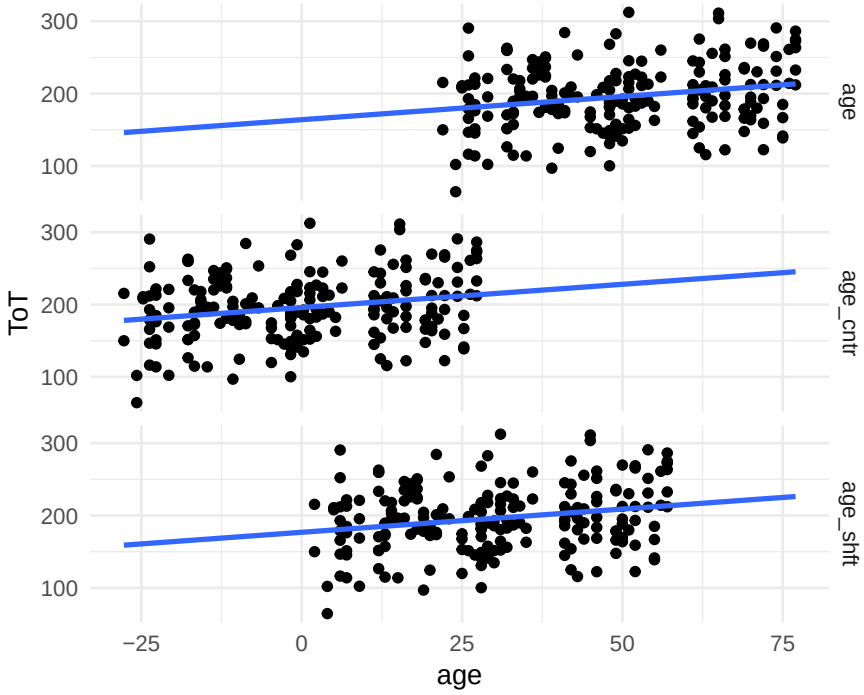
Is age associated with ToT? The coefficient table tells us that with every year of age, users get 0.64 seconds slower, which is considerable. It also tells us that the predicted performance at age = 0 is 164.09.

5.2.1 Transforming variables

The intercept parameter refers to the predicted ToT of a new born. That is a bizarre prediction and we would never seriously put that forward in a stakeholder presentation, or in the conclusion of a scientific paper, would we not? Besides that, the intercept estimate is rather uncertain, with a wide 95% interval, $164.09[143.15, 185.18]_{CI95}$.

Both, implausibility and high certainty are rooted in the same problem: the model puts a parameter, where there is no data. The broad region of the intercept is as empty as the Khali desert, because observations are impossible or have not been recorded. Fortunately, there is a pragmatic solution to the problem: *shifting the predictor*. "Shifting" literally means that the age predictor is moved to the right or the left, such that the zero point is in a region populated with observations. In the case, here, 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. Figure XY shows a shift of -20 and a centering shift 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*:

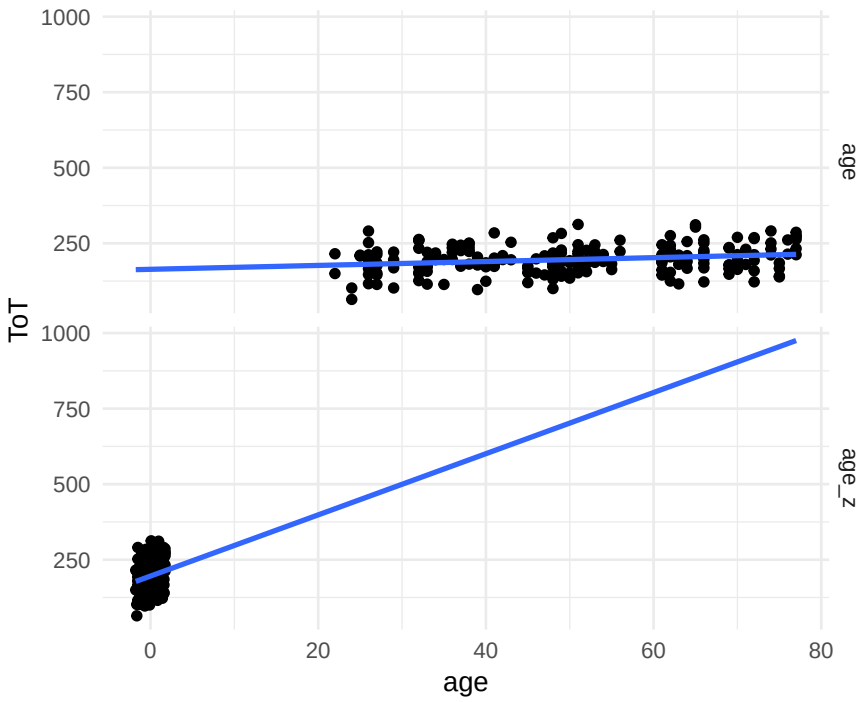
```
P_age <-
  bind_rows(posterior(M_age),
            posterior(M_age_shft),
            posterior(M_age_cntr))


T_age <- fixef(P_age)
T_age
```

model	fixef	center	lower	upper
M_age	Intercept	164.090	143.152	185.18
M_age	age	0.642	0.234	1.04
M_age_cntr	Intercept	195.888	189.843	202.09
M_age_cntr	age_cntr	0.641	0.265	1.05
M_age_shft	Intercept	176.688	163.262	190.25
M_age_shft	age_shft	0.645	0.256	1.04

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 scarcely populated with data for the very reason. Centering on the other hand results in a highly certain estimate, no surprisingly, as the region is densily populated. At the same time, the slope parameter practically does not change, neither in magnitude nor in certainty.

An even stronger standardization is *z-transformation*, where the predictor is centered at zero and all values are divided by the standard deviation, which results in a change of spread:

```
BAB1 <-
  BAB1 %>%
  mutate(age_z = (age - mean(age))/sd(age),
         age_cntr = age - mean(age))

BAB1 %>%
  tidyr::gather("predictor", "age", age, age_z) %>%
  ggplot(aes(x = age, y = ToT)) +
  facet_grid(predictor~.) +
  geom_point() +
  geom_smooth(se = F, method = "lm", fullrange = T)
```



A z-transformed variable is centered on zero and has a standard deviation of one. As can be seen, z-transformation has a considerable effect on the slope. Pulling the data cloud together pulls up the slope. Again, we run an LRM and compare against the original model:

```
M_age_z <-
  stan_glm(ToT ~ 1 + age_z, data = BAB1)
```

```
P_age <-  bind_rows(P_age, posterior(M_age_z))
T_age <- fixef(P_age)
T_age
```

model	fixef	center	lower	upper
M_age	Intercept	164.090	143.152	185.18
M_age	age	0.642	0.234	1.04
M_age_cntr	Intercept	195.888	189.843	202.09
M_age_cntr	age_cntr	0.641	0.265	1.05
M_age_shft	Intercept	176.688	163.262	190.25
M_age_shft	age_shft	0.645	0.256	1.04
M_age_z	Intercept	195.873	189.852	202.37
M_age_z	age_z	10.118	3.720	16.24

As expected, the intercept matches that of the centered variable. The small deviations are due to the [random walk] (#random_walk) and disappear when running longer MCMC chains. The slope parameter has inflated dramatically. That, of course, is not a magical trick to obtain more impressive numbers, it is simply the effect of dividing the original variable by its standard deviation. A step of one on `age` is exactly one year, whereas in `age_z` it is *one standard deviation*. Z-transformation adjusts a variable by its observed dispersion, measured as standard deviations. In the case of age, this is not very desirable.

Years of age is a natural and commonly understood unit and my advice would be to use centering, leaving the unit size untouched. When dealing with variables that are pseudo-metric, rating scales in particular, z-transformation makes sense. Imagine you were stranded on a tropical island. While putting together a shelter, you realize of useful a meter stick is. The *ur meter* is thousands of kilometers away. Obviously, you can bootstrap yourself by picking up a reasoable straight stick and mark 10 (or eight) equally spaced sections. With z-transformation you are picking your unit of measurment on what you find, too. Still, "one standard deviation" is hard to swallow for anyone untrained. If we can assume the measurement to be normally distributed, which is frequently a good approximation for rating scale data, we can derive relative statements. As it happens, the central two standard deviations cover around two thirds of the full area, leaving one sixth to each tail. That allows to make quantile statements, such as: Within central two thirds regarding age, ToT moves up by $2\beta_1 = 10.12$ seconds, if age is truly Normal, of course.

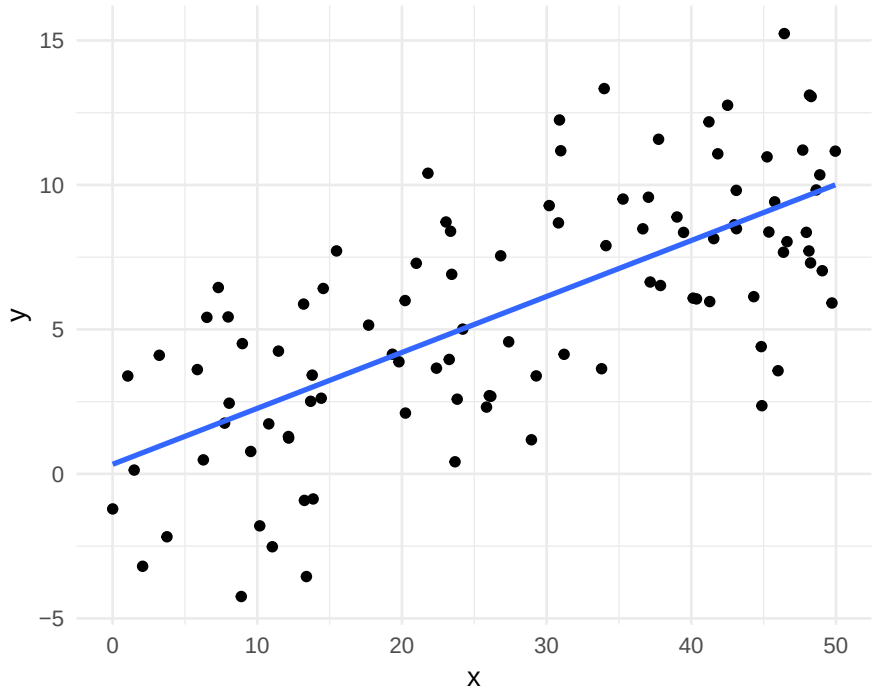
A general useful application of z-scores is to bootstrap a common unit for a diverse set of measures. Think of a battery of Likert scales, that all choose their own ranges. With z-scores we can compare the relative impact of several predictors in a population. In fact, that even makes sense for the age model. As it happens, the random variation parameter σ is given in standard deviation units, too. We can conclude that the age effect causes just a moderate fraction of all the variation observed. More research is needed.

5.2.2 Correlations

LRM render the relationship between two metric variables. A commonly known statistic that seems to do the same is Pearson's correlation coefficient 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 smimulated data set where X and Y are linearly linked:

```
D_cor <-
  data_frame(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 and that a covariance is the mean squared deviance from the population mean. This is how the correlation coefficient is decontaminated from the idiosyncracies of the involved measures, their location and dispersion. In contrast, slope parameter in an LRM is a measure of association, too. It is agnostic of the overall location of measures, because this is captured by the intercept. However, dispersion remains intact. That makes that slope and intercept together retain information about location, dispersion and association of data and we can ultimately make predictions. Still, there is a tight re.lationship between Pearson r and a slope coefficient β_1, namely:

$$r = \beta_1 \frac{sd_X}{sd_Y}$$

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
## 0.697
```

The clue with Pearson r is that it normalized the slope coefficient by the variation found in the sample. This reminds of z-transformation as was introduced in 5.2.1. In fact, when both, predictor and outcome, are z-transformed before estimation, the coefficient equals Pearson's r, right away:

```
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
## 0.697
```

In regression modelling the use of coefficients prevales because they allow for predictions. However, correlation coefficients play a major role in two situations: in exploratory data analysis and in multilevel models. With the latter we will deal in @??re_correlations). For exploratory data analysis it is recommended to inspect pairwise 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 colinearity.

5.2.3 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 an 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 magnitude 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°C (or 0K). 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 LRM. From a formal perspective, the ELA 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. From figure [G_eda_1] we have seen that the increase 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 summarize the monotonous relationship by the slope coefficient. In 5.4.3 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 more detailed picture and see disruptions of linearity as indicators for interesting psychological processes. An uncanny example of theoretical work will be given in polynomial regression. For the rest of us, 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.

5.2.4 Posterior predictions

In Bayesian regression models, *posterior predictions* are a simple, yet powerful concept to compare the fitted model with the original data. As [McElreath] points out, one should rather call them *retrodictions*, because they regard the real data one has, not the future. In yet another perspective they are *idealized*

responses and as such address both, present and future. Posterior predictions are routinely compared to *observed values* y_i which are known beforehand, but still contain the randomness component. Fitting a linear model basically means separating the idealized effect from the randomness. When the model is somewhat well-specified, this succeeds but comes at the costs of uncertainty: the *predicted value is a random variable, too.*

In the BrowsingAB case, we have repeatedly recorded age of participant. LRM found the repeating pattern, that with every unit of age, ToT increased by 0.645 seconds. This pattern is what the model predicts, now and forever. If we ask: what is predicted for a person of age 45? We get the predicted value μ_i:

$$\mu_{age=45} = 176.688 + 0.645 * 45 = 205.707$$

The predicted value is our best guess under the LRM model. Like coefficients, it is uncertain to some degree, but we are setting this aside for the moment. What we know for sure is the *observed value* . A primitive procedure to get a best guess for someone of age 45 is to find one matching case in the data and take this as face value. The data set contains a few of those participants, but the situation is rather scattered and the prediction is not clear.

```
BAB1 %>%
  select(Part, age, ToT) %>%
  filter(age == 45) %>%
  kable()
```

Part	age	ToT
28	45	153
68	45	168
128	45	120
168	45	175

A more disciplined way to obtain predicted values is to use the standard command `predict` on the model object:

```
T_pred_age <-
  BAB1 %>%
  select(Part, age, ToT) %>%
  mutate(mu = predict(M_age_shft)$center)

T_pred_age %>%
  filter(age == 45) %>%
  kable()
```

Part	age	ToT	mu
28	45	153	191
68	45	168	192
128	45	120	193
168	45	175	191

We see that all participants of age 45 get the same prediction. What differs is the observed value, as this contains the influence of all random sources at the time of measuring. We will return to residuals in the next section.

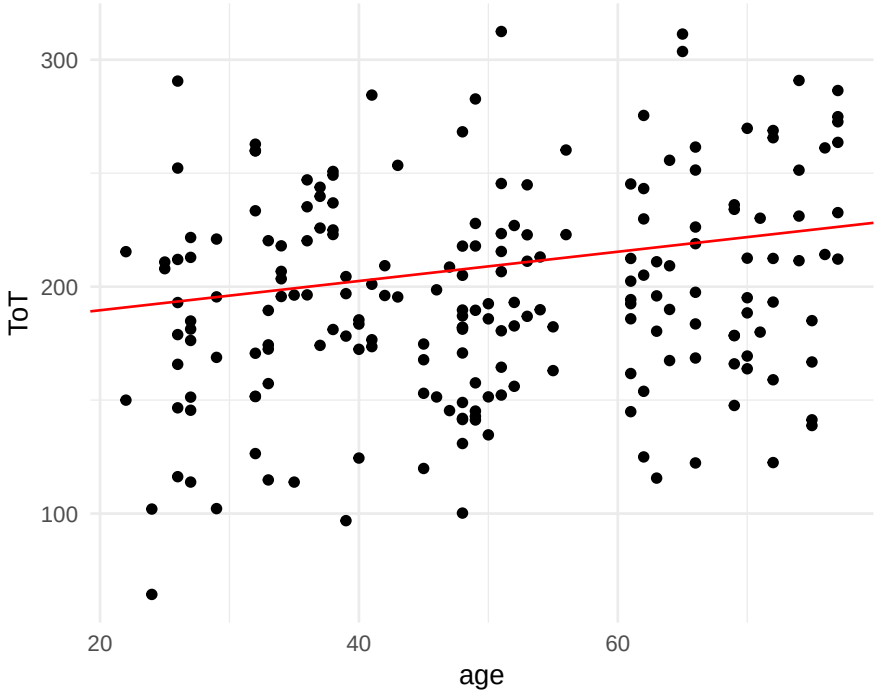
What if we wanted to get a best guess for an age that did not occur in our data set, say 43.5? Using the LR likelihood function above, we can estimate the expectation for any value we want. By entering the intercept and age estimates (`C_age`), we obtain:

```
C_age <- fixef(M_age_shft)$center
C_age[1] + 43.5 * C_age[2]
```

```
## [1] 205
```

With the same procedure, it is possible to plot observed and expected values next to each other:

```
G_pred_age <-
  BAB1 %>%
  ggplot(aes(x = age, y = ToT)) +
  geom_point() +
  geom_abline(intercept = C_age[1],
              slope = C_age[2],
              col = "red")

G_pred_age
```




```
detach(BrowsingAB)
```

This figure does not look too convincing. The regression line is rather flat, indicating a small effect of age. In addition, it looks like a lonely highway in a vast forest area. Just visually, a completely flat line or a slight negative slope would be equally credible.

- resid-predict plots as a general form to discover trends
- qq plots

5.2.5 Exercises

1. Examine the linear parameters of model `M_age_rtrn` and derive some impossible predictions, as was done in the previous section.
2. The BAB1 data set contains another outcome variable where the number of clicks was measured until the participant found the desired information. Specify a LR with age and examine the residual distribution. Is the Normality assumption reasonably defensible? What is the difference to home returns, despite both variables being counts?

3. Review the two figures in the first example of [GSR]. The observations are bi-modally distributed, nowhere near Gaussian. After (graphically) applying the model they are well-shaped. What does that tell you about checking residual assumptions before running the model?

5.3 A versus B: Comparison of groups

Another basic liknear model 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, like gender or whether they have a high school degree
- comparing situations, like whether someone uses an app on the go, or sitting still behind a computer

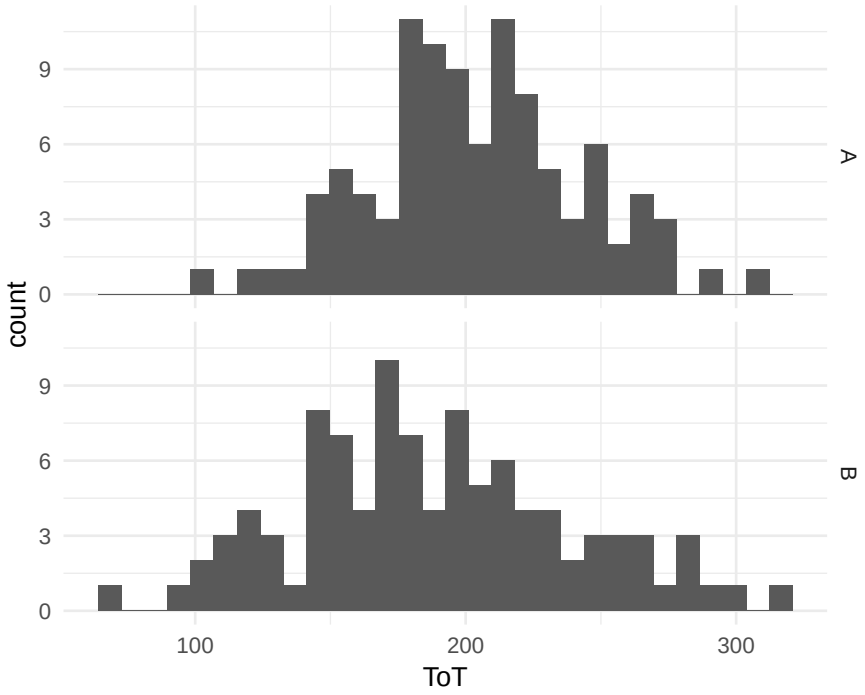
In order to perform a CGM, a variable is needed that establishes the groups. This is commonly called a *factor*. A factor is a variable that identifies members of groups, like "A" and "B" or "male" and "female". The groups are called *factor levels*. In the BrowsingAB case, the prominent factor is Design 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 over time has grown into a jungles, to the disadvantage for the 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 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 predictors (education) and 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_Design`.
2. With the `fixef` command the estimates are extracted and interpreted

```
M_Design <-
  BAB1 %>%
  stan_glm(ToT ~ 1 + Design,
           data = .)
```

```
T_Design <- fixef(M_Design)
T_Design
```

fixef	center	lower	upper
Intercept	203	194.7	212.21
DesignB	-15	-27.4	-3.05

```
detach(BrowsingAB)
```

The model contains two parameters, the first one being the *intercept*, again. How can you have a "crossing point zero", when there is no line, but just two groups. You can't. When speaking of pure factor models, like the one above or the multifactorial models of the next section, the intercept has a different meaning, it is the *mean of a reference group*. Per default, stan_glm chooses the alphabetically first group label as the reference group, design A. We can therefore say that design A has an average performance of $203.36[194.7, 212.21]_{CI95}$.

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 $15.02[27.36, 3.05]_{CI95}$ less time to complete the task. 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.

5.3.1 Not stupid: dummy variables

Are we missing something so far? We have not seen the model formula, yet. The CGM is a linear model, but this is not so apparent as it contains a factor, a non-metric variable. 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 $\times$ β_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 converzing factor levels to numbers. A dumma variable represents only one level of a factor, by asking the simple question: *Does this observation belong to group DesignB?* The answer is yes/no and can be coded by a Boolean variable. For every level, a separate dummy variable is constructed to answer the simple yes/no question of membership. But, can Boolean variables enter arithmetic equations? At least, R assumes that this is safe, as they can always be converted to ones and zeros using `as.numeric`. Routinely, we can leave the whole dummy variable business to the linear model engine. Still, it is instructive to do it yourself once.

```
attach(BrowsingAB)
```

```
BAB1 <-  BAB1 %>%
  mutate(d_A = as.numeric(Design == "A"),
         d_B = as.numeric((Design == "B")))
BAB1 %>%
  select(Obs, Design, d_A, d_B, ToT) %>%
```

```
sample_n(8) %>%
kable()
```

Obs	Design	d_A	d_B	ToT
82	A	1	0	122
133	B	0	1	260
109	B	0	1	199
42	A	1	0	252
6	A	1	0	182
191	B	0	1	260
136	B	0	1	154
116	B	0	1	185

Vice versa, numbers in a logical context are always interpreted in complete reverse.

$$v = 1 \mapsto \text{TRUE} v = 0 \mapsto \text{FALSE}$$

Boolean variables can be used in all contexts, including numerical and categorical. It is unfortunate that we call them dummies in such a belittling manner, as they are truly bridges between the world of categories and arithmetic. They identify groups in our data set and switch on or off the effect in the linear term. For a factor G with levels A and B and zero/one-coded dummy variables `d_A` and `d_B`, the likelihood is written as:

$$\mu_i = d_{Ai}\beta_A + d_B\beta_{Bi}$$

When $d_{Ai} = 1, d_{Ai} = 0$, the parameter β_A is switched on, 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. All arithmetic commands in R do an implicit typecast when encountering a Boolean variable, e.g. `sum(TRUE, FALSE, TRUE)` (the result is 2). In contrast, regression engines interpret Boolean variables as categorical. Therefore, dummy variables have to be passed on as explicitly numeric to the regression engine. Only when a variables truly is zeroes and ones, it will be interpreted as desired. This has been done that with the explicit typecast `as.numeric` above.

Most research deals with estimating effects as differences and all regression engines quietly assume that what you want is a model with a reference group. When expanding the dummy variables manually, this automatism gets into the way and the needs to be switched off, explicitly. In Rs linear models formula language, that is done by adding the term `0 +` to the formula.

```
M_dummy <-
  stan_glm(ToT ~ 0 + d_A + d_B,
     data = BAB1)
```

```
fixef(M_dummy)
```

fixef	center	lower	upper
d_A	203	194	213
d_B	188	179	197

In its predictions, the model `M_dummy` is equivalent to the fomer model `M_design`, but contains two effects that are exactly the group means. This model we call an *absolute group means model (AGM)*.

Regression engines expand dummy variables automatically, when they encounter a factor. However, when formally specifying the likelihood, dummy variables must be made explicit. When doing an AGM on a factor x_1 with $1, ..., k$ levels, the likelihood function becomes:

$$\mu_i = d_1\beta_{1[1]} + ... + d_k\beta_{1[k]}$$

In the remainder of the book, we are dealing with more complex models (e.g., multifactorial models in the next section), as well as factors with many levels (random effects in multi-level models 6). With expanded dummy variables, the likelihood can become unduly long. However, many other textbooks avoid this altogether and the model is unambiguously specified in Rs regression formula language.

Up to this point, I have introduced dummy variables at the example of AGMs, where at any moment only one factor level is switched on. A more common CGMs would have a the following likelihood specification (with a factor of k levels):

$$\mu_i = \beta_0 + d_1\beta_{1[1]} + ... + d_k\beta_{1[k-1]}$$

For a factor with k levels in a CGM with intercept, $k - 1$ dummy variables need to be constructed in the way shown above. As all non-reference levels are seen as difference towards the reference, *the reference level is set to always on*. To see so, we extract the dummy variables from the CGM on design with the standard command `model.matrix`. As you can see, for all observations the `(Intercept)` column takes the value `1`, whereas level `DesignB` is switched on and off.

```
BAB1 %>%
  select(Part, Design, ToT) %>%
  cbind(model.matrix(M_Design)) %>% ## <---
  as_data_frame() %>%
  sample_n(8) %>%
  kable()
```

Part	Design	ToT	(Intercept)	DesignB
176	B	222	1	1
106	B	182	1	1
190	B	156	1	1
161	B	219	1	1
131	B	124	1	1
175	B	208	1	1
189	B	180	1	1
144	B	236	1	1

Regression engines take care of factors, automatically, expanding the dummy variables under the hood. Still, by understanding the concept, we gained additional flexibility in specifying factorial models. One variant is to estimate an AGM, which leaves out the intercept. Of course, this can also be done right-away with the formula `ToT ~ 0 + Design`. Later, we will see a very practical application of AGMs, when drawing interaction plots. The second benefit is that we can specify the reference level as we want. It just depends on which dummy variable one sets to always-on. In the next section we will gain even more control over what the effects mean, by setting contrasts.

```
detach(BrowsingAB)
```

5.3.2 Getting it sharper with contrasts [TBC]

The default behavior of regression engines when encountering a factor is to select the first level as reference group and estimate all other levels relative to that. This fully makes sense when you are after the effect of a treatment and is therefore called *treatment contrasts*. Treatment contrasts do not have anything special to them. They are just a default of the regression engines, because so many people work experimentally.

Before we come to an understanding what contrasts are, let me point out what they are not: In @ref(dummy_variables) it was mentioned, that the AGM and CGM make exactly the same predictions (even though the formulas differ by the `+0` term). For contrasts, this holds in general. Setting contrasts never changes the predictions μ_i. In contrast, *contrast change what the coefficients*

β_i *mean*. Recall, that in an CGM, β_1 has the meaning of "difference to the reference group", whereas in an AGM, it is "the mean of the second group".

In the following, I will illustrate contrasts that represent two different types of questions:

- To what extent does a group deviate from the overall mean in the sample?
- With three ordered factor levels, how large is the rolling effect, i.e. from level 1 to level 2, and the average of 1 and 2 to level 3?

So, how are contrasts set? Although, this is a bit odd, they are neither set by modifications to the regression formula, nor by additional command arguments. Instead, *contrasts are set as attributes of the factor variable*, which we can retrieve by and set by the standard command `contrasts`. For example, the variable Education has the three levels "Low", "Middle", "High", in that order:

```
attach(BrowsingAB)
```

```
levels(BAB1$Education)
```

```
## [1] "Low"    "Middle" "High"
```

```
contrasts(BAB1$Education) %>%
  kable()
```

	Middle	High
Low	0	0
Middle	1	0
High	0	1

If you believe to have seen something similar before, you are right. Contrast tables are related to dummy variables (@ref(dummy_variables)). The column Low is omitted in the contrasts table as it is the reference level with an always-on intercept. In the previous section I mentioned that you can choose the reference level freely by constructing the dummy variables accordingly. However, that would mean to always bypass the regression engines dummy handling. The package `forcats` provides a set of commands to change the order of levels. In the following code, the factor levels are reversed, such that High becomes the reference group:

```
BAB1 <-
  BAB1 %>%
  mutate(Education_rev = forcats::fct_rev(Education)) # <--
levels(BAB1$Education_rev)
```

```
## [1] "High"   "Middle" "Low"
```

```
contrasts(BAB1$Education_rev) %>%
  kable()
```

	Middle	Low
High	0	0
Middle	1	0
Low	0	1

```
detach(BrowsingAB)
```

If we were running a CGM on `Education_rev`, the intercept would represent level High, now. Again, this model would make the very same predictions and residuals. In that respect, it is the same model as before, only the meaning (and values) of the coefficients changes and become differences to the level High.

Sometimes, it is useful to have contrasts other than treatment, as this more closely matches the research question. A plethora of contrasts is known, I will introduce *deviation contrasts* and *successive difference contrasts* in the following.

[Example for deviation coding]

Successive difference coding (SDC) applies when one is interested in effects of progressive effects, such as performance gain in a number of sessions. Consider the following research situation. Shortly after the millennium, medical infusion pumps became infamous for killing people. Infusion pumps are rather simple devices that administer medication to a patients body in a controlled manner. Being widely used in surgery and intensive care, development of these devices must comply to national and international regulations. Unfortunately, the regulations of those days almost completely overlooked the human factor. While those devices would always function "as described in the user manual", they contained all kinds of severe violations of user-interface design rules, just to name few: foil switches with poor haptic feedback, flimsy alphanumeric LCD screenlets and a whole bunch of modes. Imagine a medical devices company has partnered with some renowned research institute to develop the infusion pump of the future. Users got interviewed and observed, guidelines were consulted, prototypes developed, tested and improved. At the end of the process the design was implemented as an interactive simulation. In the meantime, national agencies had reacted, too, and regulations now demand a routine user-oriented development cycle. One of the new rules says: "the design must undergo validation testing with trained users".

That means you have to first introduce and train your users to get fluent with the device, then test them. We [REF] thought that the training process itself

is of immense importance. Why not test it, then? In the real study we tested everyone three times and traced individual progress. This requires a repeated measures analysis and we are not quite there, yet [see LMM].

[TODO:

- simulate IPump case, non-repeated
- demonstrate succ diff contr
- find example for deviation contrasts
- Contrasts extend the idea of dummy variables.
- Dummy variables become continuous, thereby more flexible in their effects

]

5.3.3 Sharper on the fly: derived quantities [TBD]

Contrasts are classic in aligning regression estimates with research questions that state a differences. With two groups A and B the following hypotheses of difference can be formed:

```
A - 0
B - 0
A - B
```

5.3.4 Exercises

1. The `simulate` function in case environment BrowsingAB lets you change the residual error (in standard deviations). Simulate three data sets with different residual variance, and estimate them by the same model. See how the uncertainty of effects behaves.
2. BrowsingAB, simulate several data sets of very small sample size. Observe how strongly the composition of education and age varies.
3. BAB1 contains the variable Education, which separates the participants by three education level (Low, Middle, High). Construct the dummy variables and run an AGM.
4. Specify the expanded CGM likelihood for education. Construct the dummy variables and run a regression.
5. Consider you wanted to use education level High as reference group. Create dummy variables accordingly and run the regression.

5.4 Putting it all together: multi predictor models

Design researchers are often forced to obtain their 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 metric covariates. Subsequently, we address the case of multi-factorial designs. Finally, we will see examples of models, where covariates and factors peacefully co-reside.

5.4.1 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 colorize 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 routinely 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 right so, 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).*

Do all users behave that way? Or can we find users of certain traits that are involved. 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 moderately complex behavioral coding system was used to derive an individual AUP resistance score. Are users with high need-for-cognition and geekism more explorative and resistant to the AUP?

As we will see later, it is preferable to build a model with two simultaneous predictors. For instructive purposes we begin with two separate LRM, one for each predictor. Throughout the regression models we use z-transformed scores. Neither the personality nor the resistance scores have a natural interpretation, so nothing is lost in translation.

$$\mu_i = \beta_0 + \beta_{\text{ncs}} x_{\text{ncs}} \mu_i = \beta_0 + \beta_{\text{gex}} x_{\text{gex}}$$

```
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 Rs regression formula language, this likewise 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() %>%
  fixef()
T_coef_3
```

model	fixef	center	lower	upper
M_1	Intercept	0.005	-0.308	0.306
M_1	zncs	0.377	0.054	0.699
M_2	Intercept	-0.004	-0.312	0.311
M_2	zgex	0.298	-0.029	0.605
M_3	Intercept	0.001	-0.306	0.317
M_3	zncs	0.300	-0.090	0.686
M_3	zgex	0.121	-0.273	0.508

The intercepts of all three models are practically zero, which is a consequence of the z-transformation. Recall, that the intercept in an LRM is the predicted value, when 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.

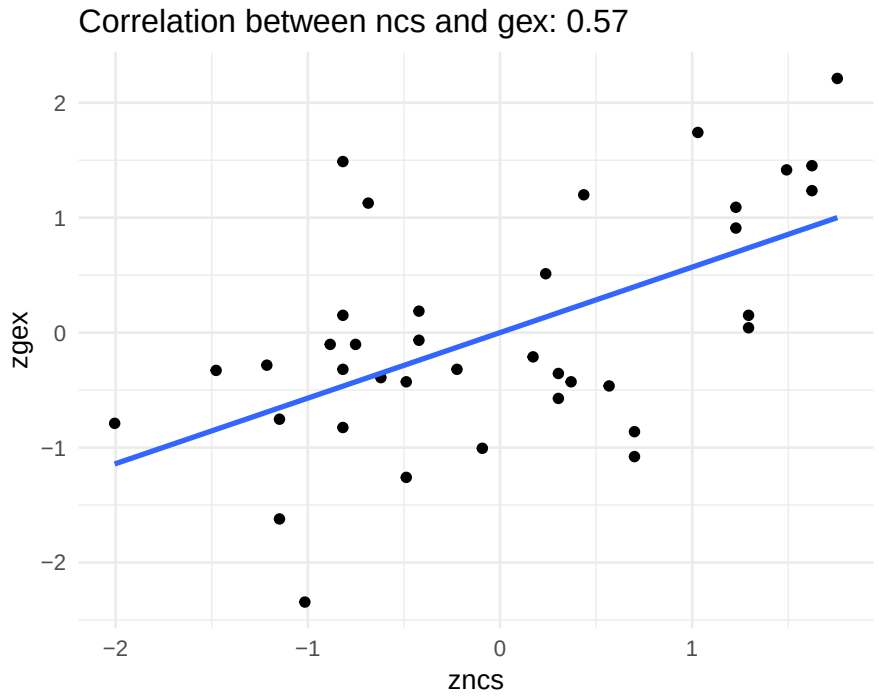
When using the two predictors simultaneously, the overall positive tendency remains. However, we observe major and minor shifts: in the MRM, the strength of the geekism score is reduced to less than half: $0.12[-0.27, 0.51]_{CI95}$. NCS has shifted, too, but lost only little of its original strength: $0.3[-0.09, 0.69]_{CI95}$.

For any researcher who has carefully conceived research questions that appears to be a disappointing outcome. In fact, there is a reason for the loss of strength and ignoring this issue will mislead the interpretation of results.

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 have more 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, some researchers speak of a *mediating variable* GEX. Although a bit outdated [REF], *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 is structural equation modelling, SEM).

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 apperance 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 by 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 issue is that a particular direction of causality can only be established, when all reverse directions can be excluded. 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. That has (virtually) happened in the BrowsingAB case, which gives it an unambiguous causal direction. This was not so, if participants had been allowed to choose themselves which design to test.

To come back to the AUP study: There is no way to establish a causal order of predictors. The correct procedure is to regard the predictors simultaneously (not step-wise), 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 then became 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 quite 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.

The readers of a paper on geekism, NCS and the AUP will probably be more impressed by the (still moderate) effects in the separate models we had run, initially. 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. The interface is beautiful
2. The interface has an aesthetic appearance?

Initial separate analysis shows 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, when 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* @ref(model_selection) and remove this (or those) predictors that do not actually contribute to prediction. The third possibility is, that the results with both predictors become more interesting when including interaction effects @ref(interaction_effects)

5.4.2 Crossover: multifactorial models [TBC]

The only necessary precondition for statistical control is that you recorded the influencing variable. This has happened in the BrowsingAB study: the primary research question regarded the design difference, but the careful researcher also recorded the gender of participants.

What happened to the likelihood function when we moved from GMM to CGM and LRM? The effect of age was simply added to the intercept. For model on education level effects, we expanded the dummy variables and then added them all up. Indeed, the linear model is defined as a succession of linear terms $x_i\beta_i$ and nothing keeps us from adding further predictors to the model. Seeing is believing! The following code estimates a model with design and gender as predictors.

```
attach(BrowsingAB)
```

```
M_mpm_1 <-
  BAB1 %>%
  stan_glm(ToT ~ Design + Gender, data = .)
```

```
T_fixef_mpm_1 <- fixef(M_mpm_1)
T_fixef_mpm_1
```

fixef	center	lower	upper
Intercept	203.349	191.9	215.10
DesignB	-15.050	-27.8	-2.68
GenderM	0.196	-12.8	12.71

By adding gender to the model, both effects are estimated simultaneously. In a *factorial MPM* the intercept is a reference group, too. Consider that both factors have two levels, forming a $2x2$ design with groups: A-F, A-M, B-F, B-M. The first one, A-F, has been set as reference group. Women in condition A have an average ToT of $203.35[191.94, 215.1]_{CI95}$ seconds. The other two fixed effects are, once again, differences to the reference. Here, nor does gender do much to performance, nor does the design effect really change, compared to the CGM.

[likelihood][interaction plot]

5.4.3 Line-by-line: regression in groups [TBC]

Recall that dummy variables make factors compatible with linear regression. No barriers are left for combining factors and covariates in one model. For example, we can estimate the effects age and design simultaneously:

```
M_mpm_2 <-
  BAB1 %>%
  stan_glm(ToT ~ Design + age_shft, data = .)
```

```
T_fixef_mpm_2 <-  fixef(M_mpm_2)
T_fixef_mpm_2
```

fixef	center	lower	upper
Intercept	184.208	169.940	198.79
DesignB	-14.981	-27.804	-2.14
age_shft	0.648	0.254	1.03

```
detach(BrowsingAB)
```

Once again, we get an intercept, first. Recall, that in LRM the intercept is the the performance of a 20-year old (age shifted). In GCM it was the mean of the reference group. When *marrying factors with a covariates, the intercept is point zero in the reference group.* The predicted average performance of 20-year old with design A is $184.21[169.94, 198.79]_{CI95}$. The age effect has the usual meaning: by year of life, participants get $0.65[0.25, 1.03]_{CI95}$ seconds slower. The *factorial effect* B is a *vertical shift of the intercept.* 20-year old in condition B are $14.98[27.8, 2.14]_{CI95}$ seconds faster. In fact, this holds for all ages, as can be seen in the following figure. The model implies that the age affect is the same with both designs, which is not true, as we will see later.

[likelihood][interaction plot]

5.4.4 Residual analysis

5.4.4.1 Assessing predictive power

```
attach(BrowsingAB)
```

When residuals are pronounced, predictions are inaccurate, which is undesireable. A model that reduces the residuals, may provide better predictions. In the current case, we may wonder: does the age predictor effectively reduce the residual standard error σ_ϵ as compared to the GMM? We fit the GMM for comparison purposes and compare the standard error. This does not look convincing, as the reduction is marginal.

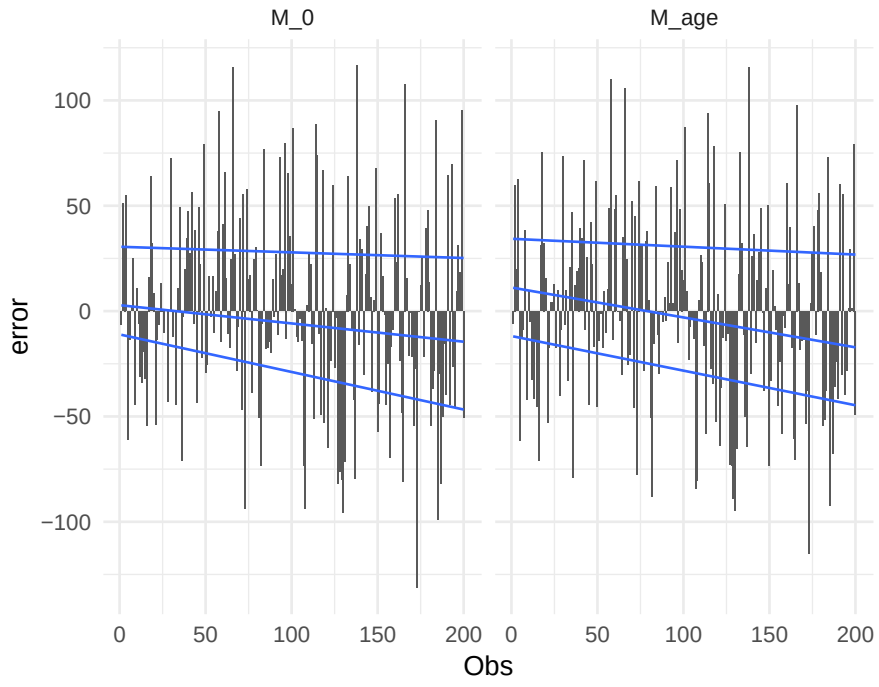
```
M_0 <-
  BAB1 %>%
  stan_glm(ToT ~ 1, data = .)
```

```
bind_rows(
  posterior(M_0),
  posterior(M_age_cntr)) %>%
  coef(type = "disp")
```

model	center	lower	upper
M_0	46.1	42.0	51.2
M_age_cntr	45.2	41.1	49.9

Does the age variable really have that little predictive value? Seeing is believing. The following graph shows the individual residuals for both models as a column plot. The two caterpillars have about the same overall width and there is no noticable reduction in residual magnitude by adding the age predictor.

```
T_resid <-
  BAB1 %>%
  select(Obs, age) %>%
  mutate(
    M_0 = residuals(M_0),
    M_age = residuals(M_age))

G_resid_age_1 <-
  T_resid %>%
  mutate(Obs_ordered = min_rank(age)) %>%
  gather(model, error, -Obs, -Obs_ordered, -age) %>%
  ggplot(aes(x = Obs, y = error)) +
  facet_grid(~model) +
  geom_col(position = "dodge") +
  geom_quantile()
G_resid_age_1
```



When adding a predictor to a model, the least one would expect is a noticable reduction in error and @ref(BAB1_resid_age_1) confirms that the age predictor is pretty useless.

```
detach(BrowsingAB)
```

5.4.4.2 Normal distribution

So far, we have seen two linear models, the GMM and the LRM. They only differ in how they sketch the relation between predictor variables and predicted values. In the remainder of this chapter on linear models, we will encounter a few more building blocks for the likelihood part and these will allow us to specify a wealth of complex models. However, the random term will stoically stay the same:

$$y_i \sim N(\mu_i, \sigma_\epsilon)$$

In words, the random term says: *observed values y_i are drawn from a Normal distribution with the predicted value μ_i as mean and a fixed standard deviation σ_ϵ*. As we have seen in 3.4.2, Normal distributions are one pattern of randomness among many and this choice may therefore be appropriate, or not. One heuristic that may justify this choice is that the observed values are located rather in the center of the scale of measurement. That is certainly much better, than to blindly stick to the Normal random pattern just for convenience. Even better is to check the assumption of Normally distributed randomness. In the following, we will examine the underlying assumptions closer and apply graphical techniques for verification. Before we delve into more depth, I would like to contrast the overall tenor in this section (and [MODSEL]) to the workflow frequently encountered in classic statistics. Boldly speaken, classicically trained researchers often sem to imply, that such assumptions needed to be checked beforehand. In the process called *assumption checking*, arcane non parametric tests are carried out, before the researcher actually dares to hit the button labelled as *RUN ANOVA*. As we will see now, the order of actions is just the other way round. In the workflow called *model criticism*, we start by contemplating what may be a reasonable model, using heuristics or even past data. Then the model is immediatly executed and the estimated model itself undergoes a routine checkup. In linear models, verifying the random pattern assumptions grounds on extracting the estimated residuals from the model object and is hence called *residual analysis*.

In the notation displayed above, there are possibly as many distributions as there are observed values (due the subscript in μ_i). It appears impossible to evaluate not just one distribution but such many. However, an equivalent

notation is routinely used for linear models, that specifies just one *residual distribution*. For the LRM that is:

$$\mu_i = \beta_0 + \beta_1 x_1 y_i = \mu_i + \epsilon_i \epsilon_i \sim N(0, \sigma_\epsilon)$$

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 Normal 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 @ref(generalized_linear_models) cannot be specified that way. But, for the purpose of residual analysis, it appears more intuitive.

The first assumption of randomness underlying the linear model simply is that the distribution follows Normal distribution. Visually, Normal distributions is characterized by:

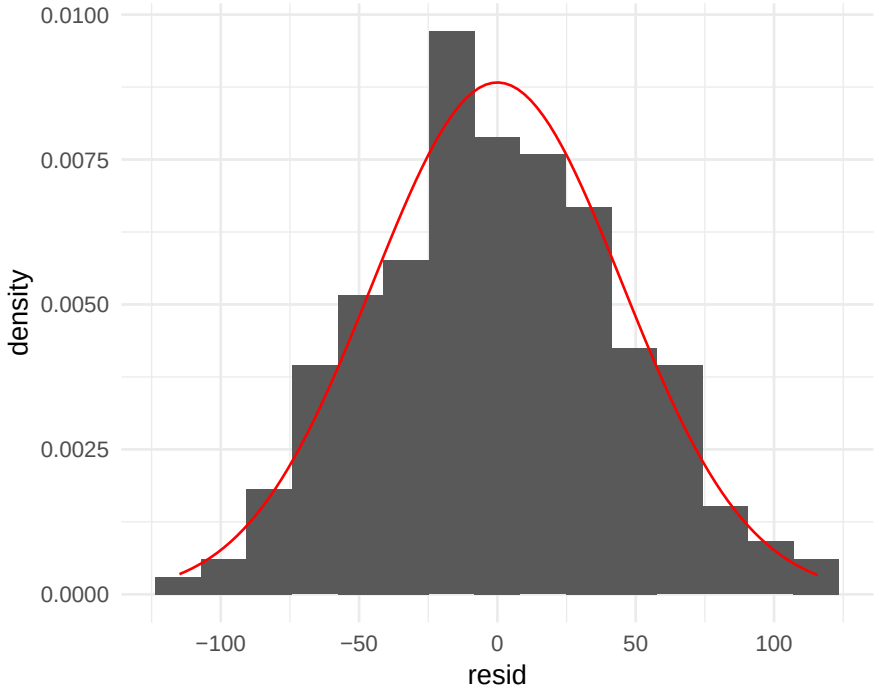
- one curved peak (unimodality)
- from which density smoothly declines towards both ends (smoothness)
- at same rates (symmetry)

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 theoretical 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 Normal distribution, which is centered at zero. The standard error σ_ϵ has been estimated alongside the coefficients and is extracted using the function `bayr::coef`.

```
attach(BrowsingAB)
```

```
G_resid_age_shft <-
  data.frame(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 = coef(M_age_shft, type = "disp")$center),
                colour = "red")
```

```
G_resid_age_shft
```



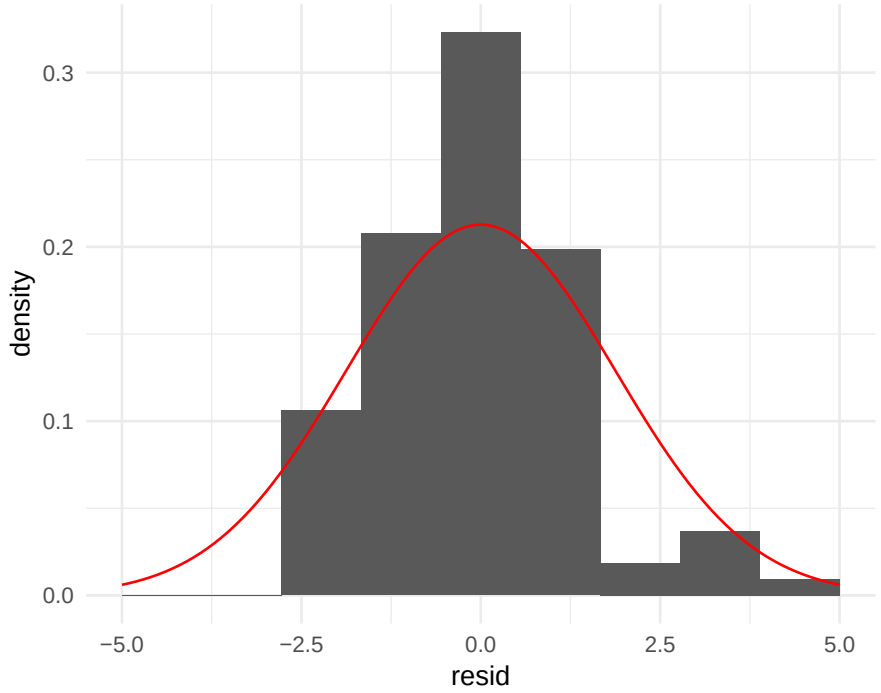
The match of residual distribution with the theoretical distribution is not perfect, but overall this model seems to sufficiently satisfy the Normality assumption. To give a counter example, we estimate the same model using the outcome variable `returns`, which captures the number of times a participant 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 <- coef(P_age_rtrn)
T_age_rtrn
```

parameter	type	fixef	center	lower	upper
Intercept	fixef	Intercept	2.563	1.99	3.099
age_shft	fixef	age_shft	-0.004	-0.02	0.013
sigma_resid	disp	NA	1.875	1.71	2.073

```
C_age_rtrn_disp <-
  T_age_rtrn %>%
  filter(type == "disp") %>%
  select(center) %>%
  as.numeric()

G_resid_age_rtrn <-
  data_frame(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_disp),
                colour = "red") +
  xlim(-5, 5)
G_resid_age_rtrn
```




```
detach(BrowsingAB)
```

The estimation produces the usual coefficients, as well as a standard error. However, the residuals not even remotely resemble the theoretical curve. While

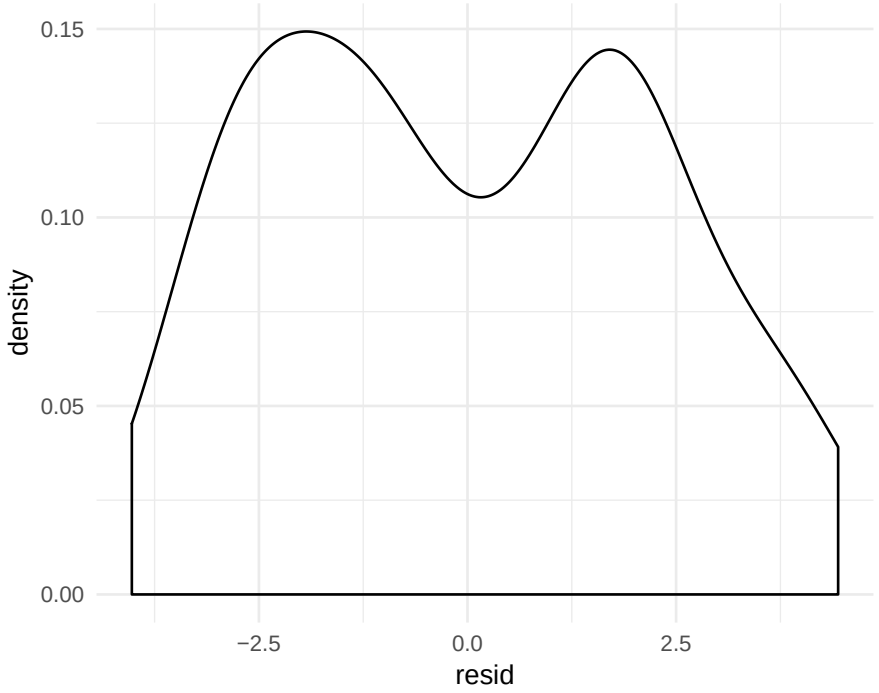
it is unimodal, it appears rather asymmetric, with a steep rise to the left and a long tail to the right. That is a typical outcome when count measures get too close to the left boundary. How about unimodality? We have not discussed any multimodal theoretical distributions in 3.4.2, but one has been displayed in @ref(first_program). 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(
    data_frame(Group = "A", y = rnorm(50, 4, 1)),
    data_frame(Group = "B", y = rnorm(50, 8, 1))
  )
```

```
M_bimod <- stan_glm(y ~ 1, data = D_bimod, iter = 200)
```

```
D_bimod  %>%
  mutate(resid = residuals(M_1)) %>%
  ggplot(aes(x = resid)) +
  geom_density()
```



These two deviations from Normal 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 7. 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. Mulitmodality 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, in almost comes natural to distinguiish between users who have used the design before, and those who didn't. If the grouping variable is recorded, the solution is group comparison models 5.3, soon to be introduced.

Visual assessment of symmetry and unimodality is simple and effective in many cases. But, Normal distributions are not the only to have these properties. At least logistic distributions and t distributions have these, too, with subtle different in curvature. Normal and t distributions differ in how quickly probability drops in the tails. Normal distributions drop much faster such that extreme events are practically impossible. With t-distributions, extreme val-

ues drop in probability, too, but the possibility of catastrophies (or wonders) stays substantial for a long time.

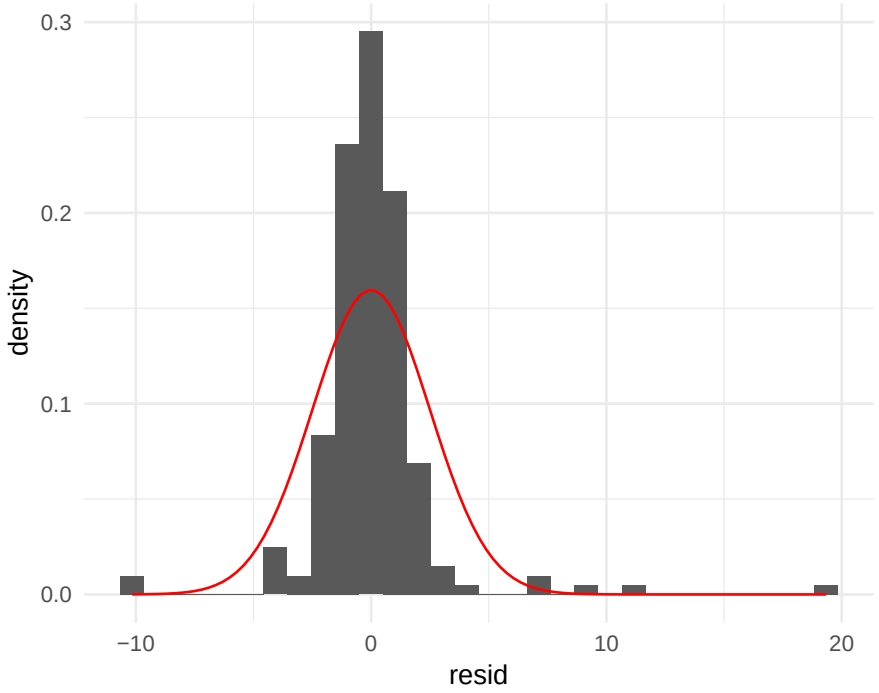
Provided one has a small abundance of data, *quantile-quantile (qq) plots* can be used to evaluate subtle deviations in curvature (and symmetry and unimodality). In qq plots, the observed and theoretical distributions are both flattened and put against each other. This is a powerful and concise method, but it is harder to grasp. The following code illustrates the construction of a qq-plot that compares GMM residuals of a t-distributed measure against the Normal 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 <- data_frame(y = rt(200, 2))
```

```
M_t <- stan_glm(y ~1, data = D_t, iter = 200)
```

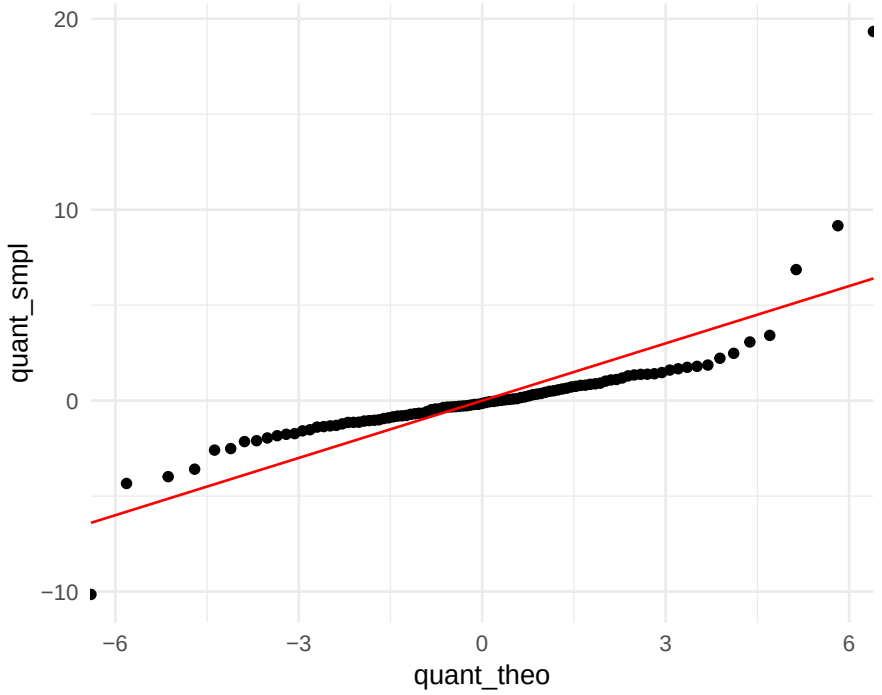
We obtain the following residual and theoretical distributions. It is approximately symmetric and unimodal, but the curvature seems to be a bit off.

```
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.

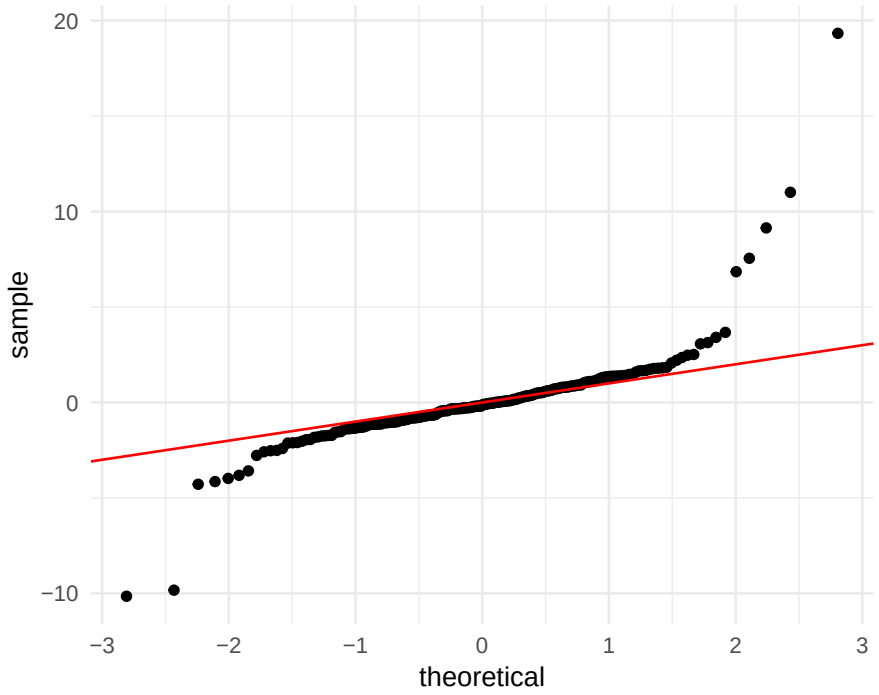
```
D_QQ <- data_frame(step = 0:100/100,
           quant_smpl = quantile(D_t$resid, step),
           quant_theo = qnorm(step, 0, C_sigma))

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 tuzrning points. here. From these points on the theoretical quantiles start lag behind. The lower and upper 1% sampled quantiles go to much more extreme values, ranging from -10 to almost 20, whereas the Normal distribution renders such events practically impossible. Generally, a rotated sigmoid shape is typical for fat tailed distributions. The problem of misusing a Normal 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? Perhaps, researchers used the Normal 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.

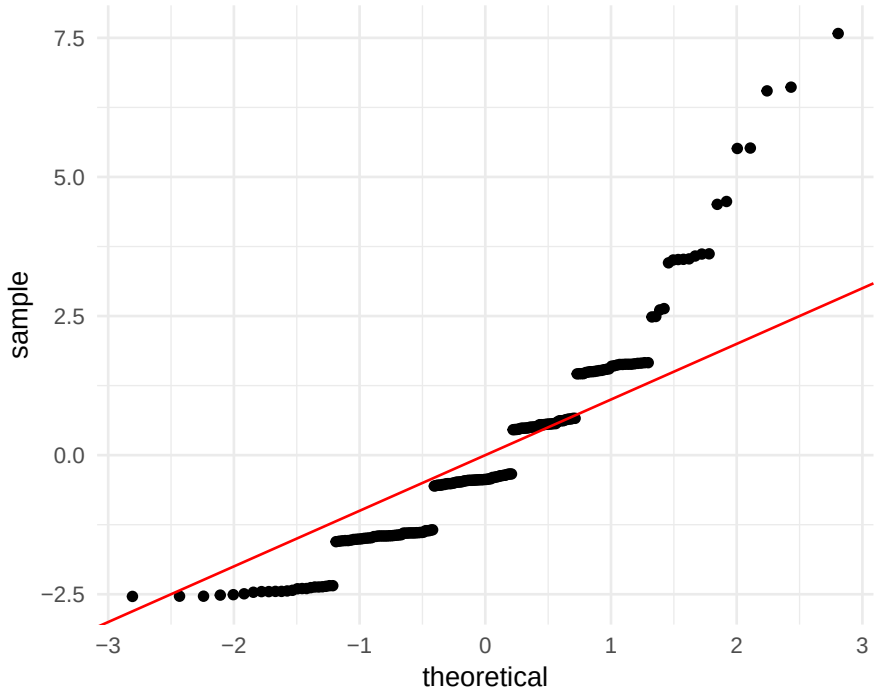
```
D_t %>%
  ggplot(aes(sample = resid)) +
  geom_qq(distribution = qnorm) +
  geom_abline(intercept = 0, slope = 1, col = "red")
```




```
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:

```
  data_frame(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 Normal distribution, but the right tail are much fatter, one 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-spedified, neither predictions nor estimate, 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 @ref(poisson_regression).

5.4.4.3 Constant variance

In @ref(resid_normality) we have assessed one assumption that underlies all linear models, namely Normal 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 grounds in the there being 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:

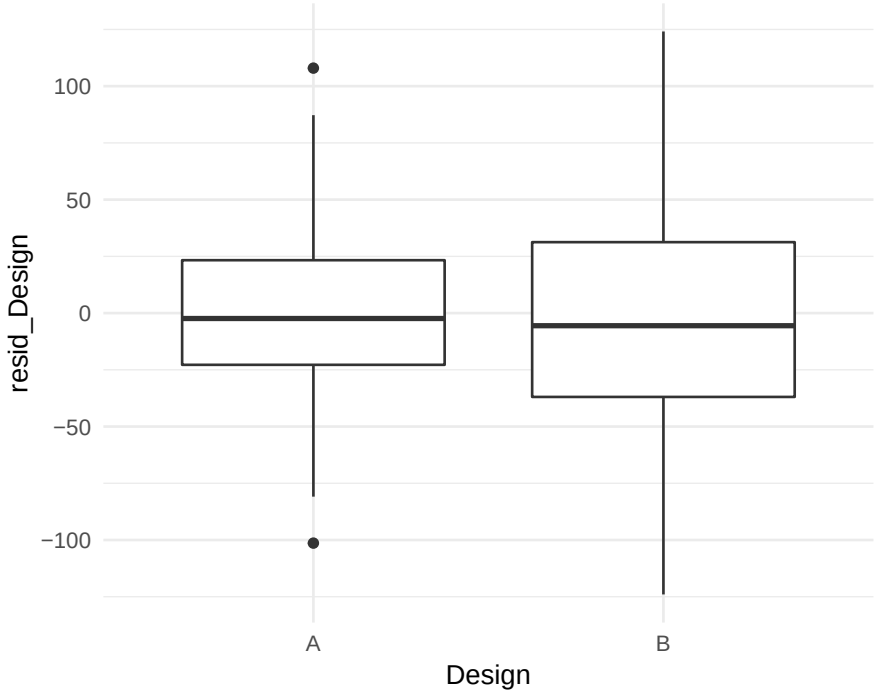
1. Consider people's daily way to work. Suppose you ask a few persons you know: "What is your typical way to work and what is the longest and the shortest duration you remember?". In statistical terms, you are asking for a center estimate and (informal) error dispersion. Is it plausible that a person with typical travel time of 5 minutes experienced the same variation as another person witha 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 as 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 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)
```

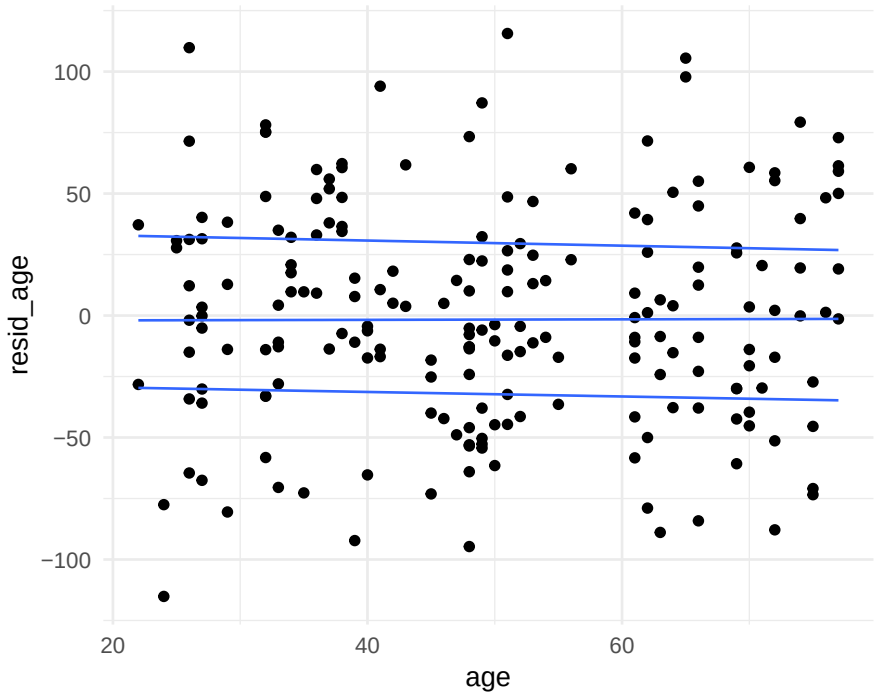
```
BAB1 %>%
  mutate(resid_Design = residuals(M_Design)) %>%
  ggplot(aes(x = Design, y = resid_Design)) +
  geom_boxplot()
```



Both sets of residuals are reasonably symmetric, but it appears that design B produces more widely spread residual. Something in the design causes individual performance to vary stronger from the population mean. The cause of this effect will be disclosed in @ref(differential_design_effects). (In essence, design B is rather efficient to use for younger users, whereas older users seem to have severe issues.)

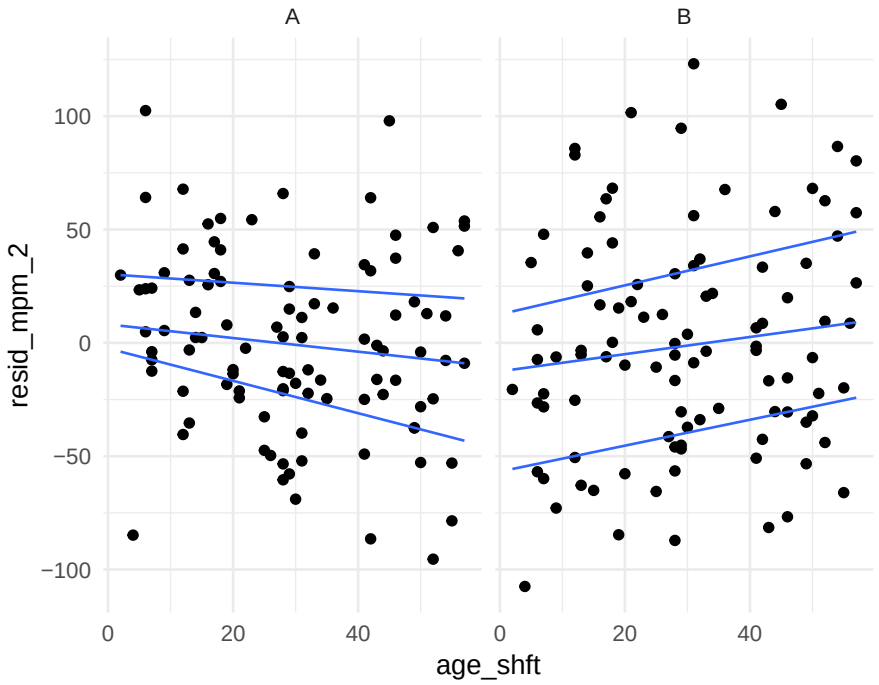
Visual checks of constant variance for factors is straight forward using common boxplots. For continuous predictors, such as age, requires a more uncommon graphical representation known as *quantile plots*. These are not the same as qq plots, but luckily are included with ggplot.

```
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 MPM on age and design, just requires to create a grouped quantile plot. This looks best using facetting, rather than separating by color:

```
BAB1 %>%
  mutate(resid_mpm_2 = residuals(M_mpm_2)) %>%
  ggplot(aes(x = age_shft, y = resid_mpm_2)) +
  facet_grid(~Design) +
  geom_point() +
  geom_quantile()
```



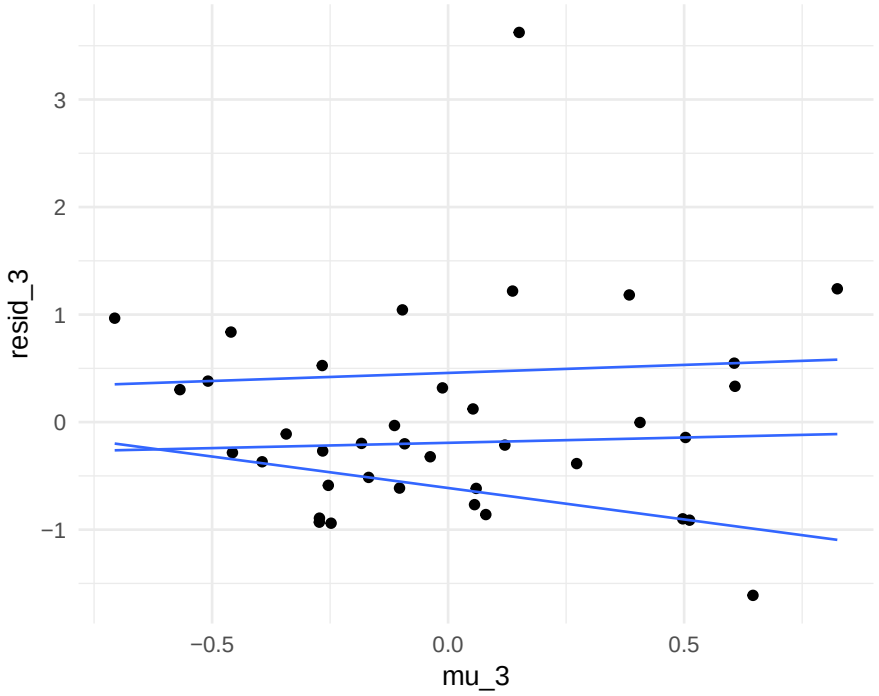
This looks rather worrying. Especially with Design A, the residuals are not constant, but increase with age. In addition, we observe that residuals are not even centered at zero across the whole range. For design A, the residual distribution moves from positive centered to negative centered, design B 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-dimnensional 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, when we turn towards @ref(generalized_linear_models), we will find that these models take 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 interaction effects or heterogeneity in groups.

```
detach(AUP)
```

5.4.5 How to plot MPM? [TBC, move to Interaction effects]

Now that we have seen how we can compare on one factor, we can extend the CGM to more factors. With linear models, we can extend the basic CGM to incorporate multiple of such factors. This is called *multifactorial models* (MFM).

Now, we are looking at a slightly more complicated situation, where browsing is predicted by design and education level of participants at the same time.

First, with the *ggplot* system, we plot the new situation. Recall that

1. mapping variables in the data set to elemental properties in the plot, such as: x position, y position, color, shape etc.
2. selecting an appropriate geometry that carries such properties, e.g. points, bars, box plots

Now we can do the two-factorial ANOVA in much the same way as before. We only have to add the predictor *Education* to the model formula. Then we run the MCMC estimation and get the estimates.

[Interaction plots with groups, IA plots with standard debs, mixed IA plots, using the AMM]

5.4.6 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 variables lends itself to be kept constant. For example in cognitive psychology 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, there exist variables, which you cannot manipulate for one of the following reasons. 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 on a secret place, it is crucial to observe all situations. Consider a car navigation system that is tested in a long, lonely highway situation. How much would the results tell your for performance in dense city traffic? Design researchers frequently need results that are highly representative for 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*. [A special case of within-subject design is *repeated measures*.] In the following chapter, we use mixed-effects models [LMM] to deal with within-subject designs, gracefully.

In design research, pair matching applies for situations, where designs are compared. In the simpler situation that a design is evaluated against 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 teh sample as in the population.

You can only match what you can measure and you only measure what you expect. Human behavior 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 sample, 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 will see now, the linear model puts no limits on the number of predictors. If you believe that user age may play a role for performance, just record it and add it to the model. Statistical control requires at least coverage representation of the sample.

5.4.7 Exercises

1. Rerun `M_3`, this time using the unstandardized resistance scores. Interpret the intercept.
2. The innovators behind the WWW have a long tradition in figurative use of nautical metaphors, like navigator, cybernaut, ... Do people in a world-wide hypermedia system behave like animals roaming known and lesser known territory? Then we would expect performance to be dependent on abilities for spatial cognition, like the visual-spatial working memory. Another theory could draw on the fact that for most websites the written word prevails. Would we not expect people with better verbal processing capabilities to excel? Consider a study `MMN` (for Miller's magic number, an iconic term in working memory research) that examines the influence of working memory capacities on people's browsing performance. Two tests for WM capacity were used: the Corsi task for visual-spatial WM capacity and the Ospan task for verbal WM capacity. Then participants were given five different search tasks on five websites. As a measure of efficiency, total time on task and number of clicks were recorded. As these variables tend to have skewed residual distributions, logarithmic transformation was applied before analysis. *Do LRM on both WM predictors separately, then move on to an MRM..*

5.5 Interaction effects

With the framework of MPM, we can use an arbitrary number of predictors. These can represent properties on different levels, for example, two designs 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 (literally) 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 of the form:

> the effect of design changes with (some) user properties

In regression models, conditional statements like these are represented by *interaction effects*. Interactions between user properties and designs are the most genuine in design research, and deserve a neologism: *differential design effects (DDM)*. They come with some of their siblings. *Saturation* occurs when physical (or other) boundaries are reached and the result is less than the sum. *Amplification*, a rare one, is like compound glue: it will harden only if the two components are present. The final section is a plea for interaction effects in theorizing.

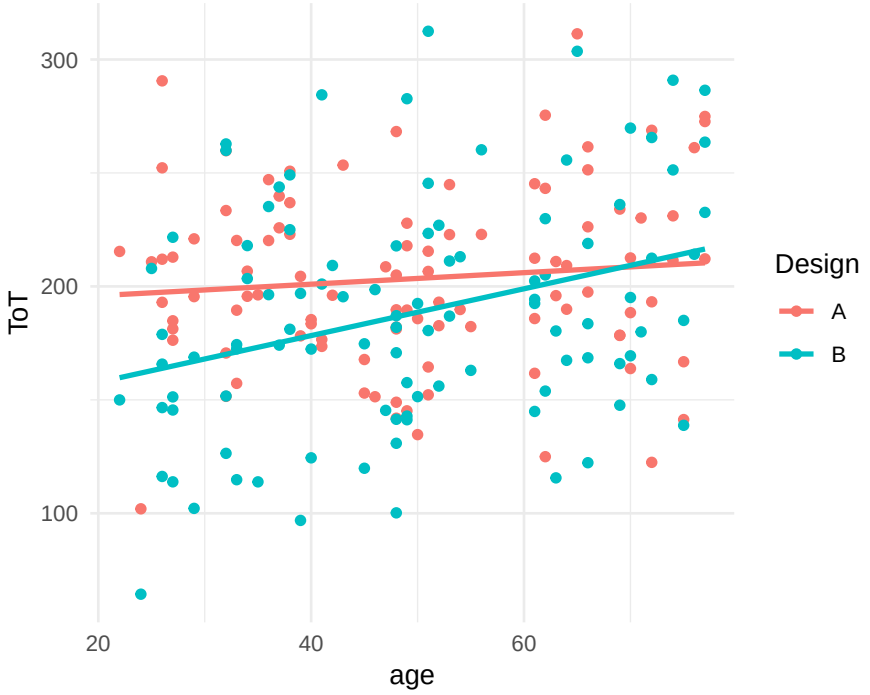
5.5.1 Users differ: differential design effects

Do people differ? Yes. If you showed two web designs to a group of stakeholders, asking to choose individually. Is there any chance they would all agree? No. Is the design of universal systems easy? No. The ideal of universal design is to never put a user group at a disadvantage. If it is an ideal, than designs are likely to differ in by how much they accomplish it.

As to age: 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. Is, perhaps, one of the two designs less of a burden for elderly users? We plot the observed ToT by age, this time forming groups by design.

```
attach(BrowsingAB)

G_slope_ia <-
  BAB1 %>%
  ggplot(aes(x = age,
             col = Design,
             y = ToT)) +
  geom_point() +
  geom_smooth(method = "lm", se = F)
```

```
G_slope_ia
```



As we have seen, with GLM, it is possible to investigate the effect of design properties and user properties simultaneously. For example, assume that main difference between design A and B in the web browsing example is that A uses larger letters than B. Would that create the same benefit for everybody? It is not unlikely, that larger letters only matter for users that have issues with far 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 recall the MPM in BrowsingAB: `M_mpm_2` showed a moderate relationship between age and ToT. The design effect was disappointing (`M_Design`): a classic statistician may have called it "significant", but one can hardly claim practical relevance. By adding the interaction effect `Design:age_shft` to the model, we will now investigate how the age effect differs by design. We call this a *differential interaction effect*, as one of the involved

```
M_ia1 <-
  BAB1 %>%
```

```
  stan_glm(ToT ~ Design + age_shft + Design:age_shft,
                 data = .)

# T_resid <- mutate(T_resid, M_ia1 = residuals(M_ia1))
```

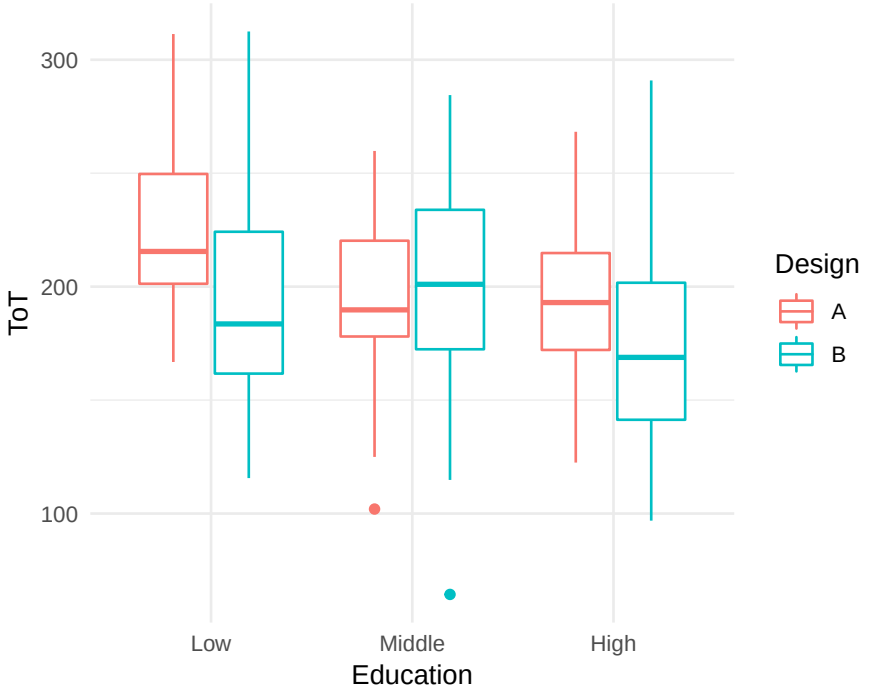
```
T_ia1 <-
  bind_rows(
    posterior(M_mpm_2),
    posterior(M_ia1)) %>%
  fixef()


T_ia1
```

model	fixef	center	lower	upper
M_ia1	Intercept	195.684	176.708	214.337
M_ia1	DesignB	-37.779	-64.695	-12.813
M_ia1	age_shft	0.259	-0.276	0.798
M_ia1	DesignB:age_shft	0.770	0.056	1.557
M_mpm_2	Intercept	184.208	169.940	198.789
M_mpm_2	DesignB	-14.981	-27.804	-2.140
M_mpm_2	age_shft	0.648	0.254	1.033

The intercept still is the performance of an average twen using reference design A. Compared to the MPM, it is less favorable, with $195.68[176.71, 214.34]_{CI95}$ seconds time-on-task. The effect of design B improved dramatically with the DDM: a twen is now $-37.78[-64.7, -12.81]_{CI95}$ faster with it. Is B the clear winner? It is not, because A has much lower age effect. The age parameter does no longer represent both groups, but just reference A, and it is now very small, $0.26[-0.28, 0.8]_{CI95}$. The final coefficient is *not* the age effect in B, but *the difference* to the reference age effect. With design B, users are getting 1.029 seconds slower per year of life. That is a lot!

If we can do it with covariates, like age, we can do it with factors, too. For example, does the overall improvement from design A to B depend on education level?

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



Again, we compare the main-effect model to the one with interaction effects.

```
M_mpm_3 <-
  BAB1 %>%
  stan_glm(ToT ~ Design + Education,
                 data = .)

M_ia2 <-
  BAB1 %>%
  stan_glm(ToT ~ Design + Education + Design:Education,
                 data = .)

# T_resid <- mutate(T_resid, M_ia2 = residuals(M_ia2))
```

```
T_ia2 <-
  bind_rows(
    posterior(M_mpm_3),
    posterior(M_ia2)) %>%
  fixef()


T_ia2
```

model	fixef	center	lower	upper
M_ia2	Intercept	224.57	208.53	239.20
M_ia2	DesignB	-26.34	-48.10	-4.42
M_ia2	EducationMiddle	-29.51	-51.65	-7.64
M_ia2	EducationHigh	-31.49	-51.70	-10.77
M_ia2	DesignB:EducationMiddle	32.98	1.61	64.12
M_ia2	DesignB:EducationHigh	4.36	-24.50	32.74
M_mpm_3	Intercept	218.83	206.34	231.21
M_mpm_3	DesignB	-15.11	-26.84	-3.27
M_mpm_3	EducationMiddle	-13.46	-28.80	2.44
M_mpm_3	EducationHigh	-29.39	-43.19	-14.46

The model has factors only, so there is a reference group. The intercepts in both models represent low education encountering design A. The main effect designB on low-educated users is present in both models. With the interaction term, design B looks much favorable, $-26.34[-48.11, -4.42]_{CI95}$. The effect of middle education has doubled, whereas it remains stable for high education.

That may appear strange at first, but keep in mind, that by the interaction term, the education main effects are no longer "main": they only refer to group A, now. In the main effects model, the same education effects are assumed for both designs. Here, it is conditional on design, which brings us to the two interaction effects `B:Middle` and `B:High`. These are, once again, differences. The effect of middle education in B is 32.98more seconds than in A (-29.51). There practically is no net effect of middle education in B. In contrast, the high education interaction effect is small, with practically makes `DesignB` a *main effect: it is the same in both groups.*

Count the number of parameters in both models! We have six with interaction and four without. Six is just the number of groups and, indeed, with interaction effects all group means can vary freely. That is best demonstrated by estimating a variant of the model, where six parameters represent six group means, directly. In other words, it is an AGM, without an intercept and without main effects:

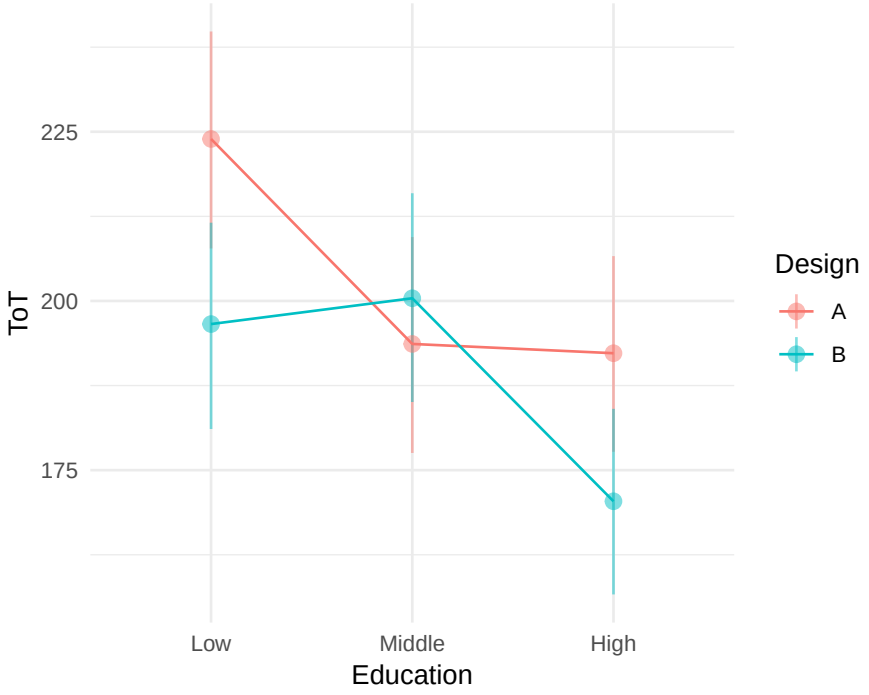
```
M_ia3 <-
  BAB1 %>%
  stan_glm(ToT ~ 0 + Design : Education,
                 data = .)
# T_resid <- mutate(T_resid, M_ia2 = residuals(M_ia2))
```

We extract the coefficients and, with a little preparation, we create an *interaction plot*. It shows the same pattern as the exploratory plot, but is fully inferential, with error bars indicate the 95% credibility interval.

```
P_ia3 <- posterior(M_ia3)
T_ia3 <- fixef(M_ia3)
T_ia3
```

fixef	center	lower	upper
DesignA:EducationLow	224	208	240
DesignB:EducationLow	197	181	212
DesignA:EducationMiddle	194	178	209
DesignB:EducationMiddle	200	185	216
DesignA:EducationHigh	192	178	207
DesignB:EducationHigh	170	157	184

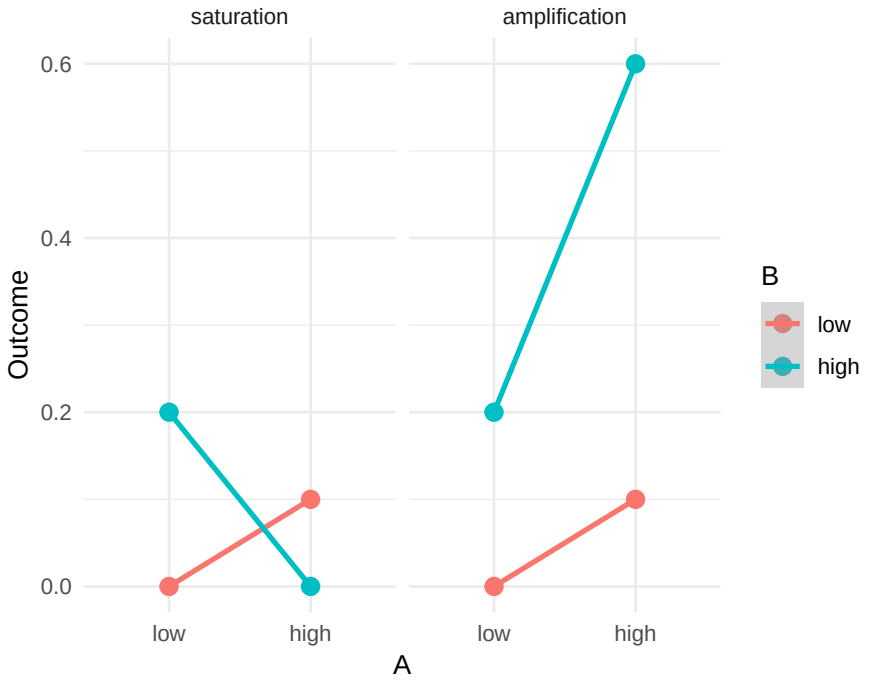
```
T_ia3 %>%
  select(fixef, ToT = center, lower, upper) %>%
  separate(fixef, c("Design", "Education"), ":") %>%
  mutate(Design = str_replace(Design, "Design", ""),
         Education = str_replace(Education, "Education", ""),
         Education = forcats::fct_inorder(Education)) %>%
  ggplot(aes(y = ToT, ymin  = lower, ymax = upper,
             x = Education,
             color = Design, group = Design)) +
  geom_pointrange(alpha = .5) +
  geom_line()
```



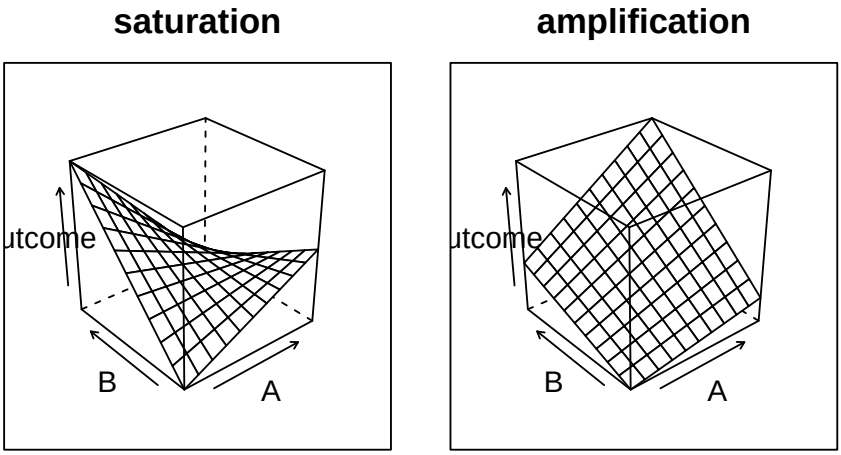

```
detach(BrowsingAB)
```

We can distinguish between *saturation effects* and *amplification effects*.

```
    expand.grid(effect = c("saturation", "amplification"),
                            A = c(0,1),
                            B = c(0,1)) %>%
        join(data.frame(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)) %>%
    join(data.frame(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
)
```



5.5.2 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:

$$y_i = \beta_0 + \beta_1 x_{1i} ...$$

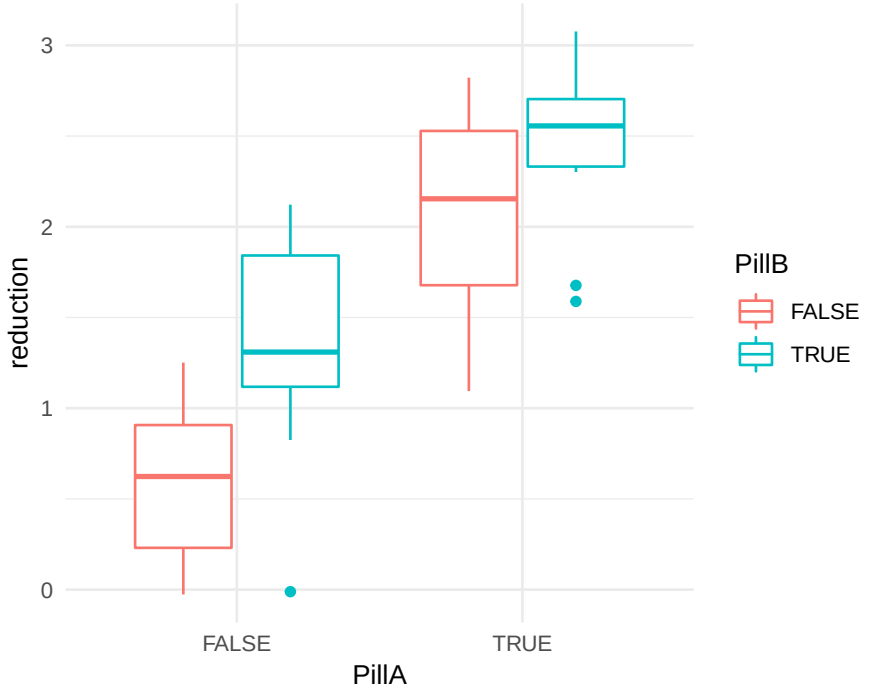
The formula basically says, that if we increase x_1 (or any other influencing variable) by one unit, y will increase by β_1. It also says that y is composed as a mere sum. In this section, we wil discover that these innocent assumption do not hold.

A major flaw with the linear model is that it presumes the regression line to increase and fall infinitely. However, *in an endless universe everything has boundaries*. 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. Boundaries of performance measures 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 later chapters of this book ([GLM, NLM]). Still, most researchers use linear models, and it often can be regarded a reasonable approximation, as long as one keeps reasonable distance to the upper and lower boundaries of the outcome.

Before we turn to a genuine design research case, let me explain saturation effects by an example that I hope is intuitive. It boils down to the question: do two headache pills have the double efect than one? Consider a pharmaceutical study on the effectiveness of two pain killer pills A and B. It takes place in the aftermath of a great 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: A-B, A-Placebo, B-Placebo, Placebo-Placebo. After 30 minutes, experienced 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)

Pills %>%
  ggplot(aes(x = PillA, col = PillB, reduction)) +
    geom_boxplot()
```



When neither pill is given a slight spontaneous reduction seems to occur. Both pills alone seem to be effective in their own way, and giving them both has the most beneficial effect. However, it seems that the net effect of B is slightly weaker when administered together with A. Again, when the effect of one predictor depends on the level of another, the effect is called *conditional*.

Now, we will estimate and compare a set of three models, starting with the standard factorial MPM and proceeding to two models with interaction effects. The third one is an AGM with two factors, actually, although it may not seem so at first.

```
M_1 <- stan_glm(reduction ~ 1 + PillA + PillB, data = Pills)
M_2 <- stan_glm(reduction ~ 1 + PillA + PillB + PillA:PillB, data = Pills)
M_3 <- stan_glm(reduction ~ 0 + PillA:PillB, data = Pills)

P <- bind_rows(
  posterior(M_1),
  posterior(M_2),
  posterior(M_3)
)
```

Imagine, the researcher started with the two-way MGM, estimating the fixed effects PillA and PillsB. The result is:

```
T_fixef_1 <- fixef(M_1)
T_fixef_1
```

fixef	center	lower	upper
Intercept	0.678	0.384	0.975
PillATRUE	1.293	0.959	1.626
PillBTRUE	0.576	0.232	0.907

Given these estimates, we conclude that pill A has an almost double reduction of headache compared to B. The intercept indicates that headache spontaneously diminishes at a rate of $0.68[0.38, 0.98]_{CI95}$. Setting the spontaneous recovery aside, what would you predict the effect to be in the group with both pills?

Thinking linearly you probably came up with:

$$1.87$$

Does that makes sense? Do the effects of headache pills simply add up like this? Consider a scenario, where five headache pills were compared in the same manner. If we assume linear addition of effects, some participants in the group with all pills could even experience the breathtaking sensation of *negative* headache. Certainly, the effect is not linear. But, is it in the long run? In some Hollywood movies the wounded hero consumes handful of pills before the showdown (With the exception of Leon, the Profi, where it is the scary Gary Oldman consuming some harder stuff). With two pills, how close to the boundaries are we? We can just test that by introducing an *interaction term* to the model: `PillaA:PillB`.

```
T_fixef_2 <-  fixef(M_2)
T_fixef_2
```

fixef	center	lower	upper
Intercept	0.592	0.252	0.940
PillATRUE	1.469	0.976	1.940
PillBTRUE	0.746	0.270	1.226
PillATRUE:PillBTRUE	-0.359	-1.017	0.315

We first examine the main effects for both pills. Interestingly, both effects are now considerably stronger than before. Taking either pill alone is more

effective than we thought from the multiple CGM. The fourth coefficient (`PillATRUE:PILLBTRUE`) is the interaction term. With the updated model we, again, predict the net headache reduction when both pills are given. This is as simple as summing up all coefficients from intercept to interaction. The result is a combined effect of both main effects, *reduced* by 0.36. As we would expect, the effectiveness of two pills is not their sum, but less. One can have headache to a certain degree or no headache at all. If it's gone, any more pills have no additional effects. This is called an *saturation effect*: the more is given, the closer it gets to teh natural boundaries and the less it adds. Let me emphasize that this is not some strange (and therefore interesting) metabolic interaction between pharmaceuticals. Saturation effects must not be confused with any "real" interaction, that is theoretically interesting. Still, as we have seen, ignoring saturation effects results in severely biased estimates.

Back to design research! Imagine, you are testing the influence of font size on reading performance. In your experiment, you found that increasing from 8pt to 12pt cuts the required reading time by 10 seconds. According to the linear model, you had to expect the reading time to decrease by another 10 seconds, when enlarging to 16pt.

```
detach(Headache)
```

```
attach(Reading)

D_reading_time <-
  data_frame(font_size = c(4, 8, 12, 16, 24, 28),
             observed_time = c(NA, 40, 30, NA, NA, NA),
             predicted_time = 60 - font_size/4 * 10) %>%
  as_tbl_obs()

D_reading_time
```

Obs	font_size	observed_time	predicted_time
1	4	NA	50
2	8	40	40
3	12	30	30
4	16	NA	20
5	24	NA	0
6	28	NA	-10

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 ar-

rive 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 when you bring a second manipulation into the game, that has a functionally similar effect? A likely outcome is that the fundamental assumption of *predictors sum up* no longer holds, but *saturation* occurs.

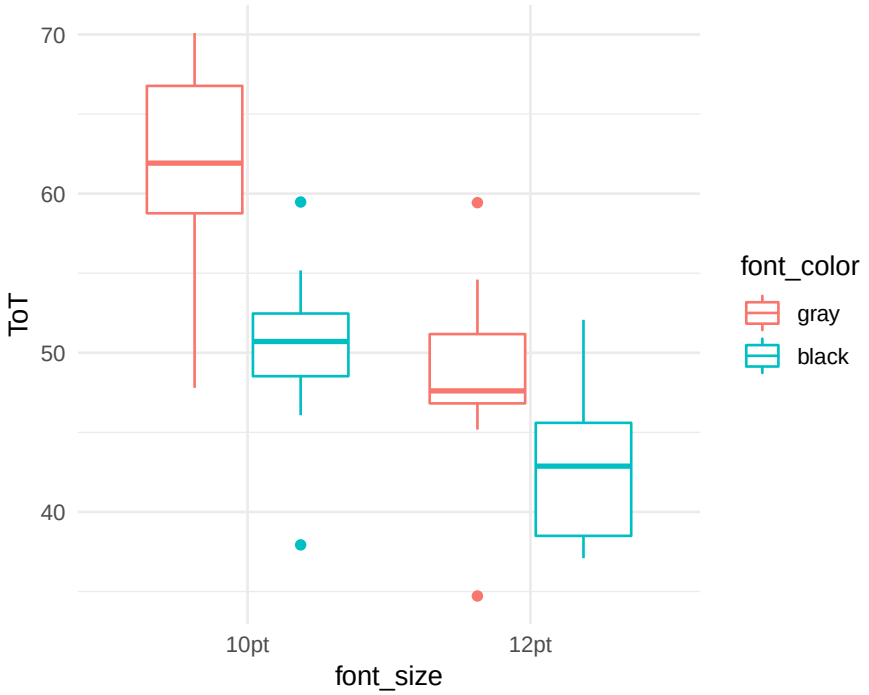
Let's turn to a fictional, yet realistic problem in design research. Design of systems is a matter of compromises. A common conflict of interests is between the aesthetic appearance and the ergonomic properties. Consider the typesetting on web pages. Many commercial websites are equally concerned about the ease of comprehending information and positive emotional response. From an ergonomic point of view, the typesetting of longer text simply requires you to choose the best screen-readable font, find an optimal font size and maximize contrast. If you were now to suggest typesetting a website in black Arial on white background, you are in for trouble with your graphic designer or, much worse, the manager responsible for corporate impression management. In any case, someone will insist on a fancy serif font (which renders rather poorly on low resolution devices) in an understating blueish-grey tone. For creating a relaxed reading experience, the only option left is to increase the font size. This option is limited by another compromise, though, as too large letters requires excessive scrolling.

The 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 between-subject design: the same page of text is presented in four versions, with either 10pt or 12pt, and grey versus black font color. Performance is here measured as time-on-task of reading the full page.

```
D_1
```

Part	font_size	font_color	mu	ToT
2	12pt	gray	48	45.2
10	12pt	gray	48	47.7
13	10pt	gray	60	53.1
20	12pt	gray	48	54.6
25	10pt	black	50	59.5
29	10pt	black	50	52.3
33	10pt	black	50	55.2
34	12pt	black	46	43.0

```
D_1 %>%
  ggplot(aes(col = font_color,
             x = font_size,
             y = ToT)) +
  geom_boxplot()
```



We see immediately, that both design choices have a an impact: more contrast, as well as larger letters lead to improved reading performance. 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. So, neither with large font, nor with optimal contrast alone has the design reached maximum performance, i.e. saturation. Still, there could be an interaction effect, that gradually reduces the positive effect of large font in a high-contrast design. We run two regression model, one with both main effects and one that adds an interaction term. We extract the coefficients from both models and view them side-by-side:

```
M_1 <-
  D_1 %>%
  stan_glm(ToT ~ font_size + font_color,
           data = .)
```

```
M_2 <-
  D_1 %>%
  stan_glm(ToT ~ font_size + font_color + font_size : font_color,
           data = .)
```

```
T_read_fixef <-
  right_join(select(fixef(M_1), fixef, M_1 = center),
             select(fixef(M_2), fixef, M_2 = center),
             by = "fixef")

T_read_fixef
```

fixef	M_1	M_2
Intercept	59.79	61.17
font_size12pt	-9.91	-12.45
font_colorblack	-8.23	-10.78
font_size12pt:font_colorblack	NA	5.22

The estimates confirm, that both manipulations have a considerable effect. And, there is an interaction effect as well, correcting the additive effect of font color and size. The combined effect of high contrast and large font is therefore

$$\mu_{12pt,black} = 61.172 + -12.45 + -10.783 + 5.222 = 43.161$$

In the research scenario, that was not the question, strictly, as we were primarily interested in comparing the effects of font size and contrast. Also, if we see the credibility interval of the interaction effect it is not highly certain ($5.22[-2.25, 12.57]_{CI95}$)) Still, including the interaction term is the right choice, for two reasons: first, both manipulations influence the same cognitive processing, as they both improve visual clarity, with the effect of better visual letter recognition. From a theoretical perspective, we would thus expect saturation. This is simply the application of prior knowledge and in a perfect world one would use a prior distribution for the interaction term, creating a more certain estimate. That being said, the data set is synthetic, and the simulation definitely included the interaction effect. Second, in Table [XY], all other coefficients change considerably with introduction of the interaction effect. Especially, the effects of the two manipulations get considerably underestimated.

Why are the main effects under-estimated, when not including the interaction term? The pure main-effects model has three parameters. This allows

it to represent the same number of independent group means. Formally, the number of groups in the study is four. However, the three-parameter model assumes that the fourth group (black, 12pt) can sufficiently be specified by the existing three parameters. If saturation occurs, the group of participants in the 12pt group is not homogeneous: in the grey group, they experience a stronger improvement than in the black group. The three parameter model is forced to make the best out of situation and adjusts the net effect of font size to be slightly lower than it actually is. The same occurs for the font color effect.

Interaction effects are notoriously neglected in research, and they are hard to grasp for audience with or without a classic statistics education. It is therefore highly recommended to illustrate interaction effects using interaction plots. To begin with, an interaction plot for the 2x2 design contains the four estimated group means. These can be computed from the linear model coefficients, but there is an alternative way: we modify the model, such that it represents the four group means, directly. The re-parametrized model has no intercept (if we want the group means directly, we need no reference group), and no main effects. The model contains just the raw interaction effect, which results in an estimation of the four group means:

```
M_3 <-
  D_1 %>%
  stan_glm(ToT ~ 0 + font_size : font_color,
           data = .)
```

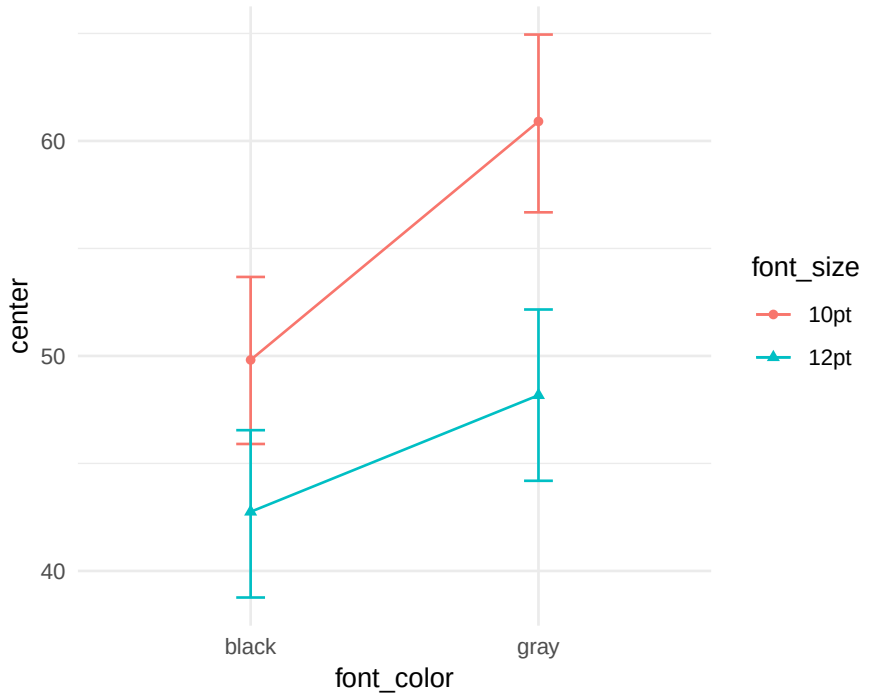
```
fixef(M_3)
```

fixef	center	lower	upper
font_size10pt:font_colorgray	60.9	56.7	64.9
font_size12pt:font_colorgray	48.2	44.2	52.2
font_size10pt:font_colorblack	49.8	45.9	53.7
font_size12pt:font_colorblack	42.8	38.8	46.5

With some basic data transformation on the coefficient table, we create a data frame that encodes the two factors separately. Note how `separate` is used to split the coefficient names into group identifiers. Using `mutate` and `str_replace`, the group labels are stripped the factor names. The resulting coefficient table serves as input to ggplot, creating an interaction plot with overlayed credibility intervals, using `geom_errorbar`.

```
T_2x2_mu <-
  fixef(M_3) %>%
```

```
  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", "")) %>%
  as.data.frame()

G_interaction <-
  T_2x2_mu %>%
  ggplot(aes(x = font_color,
             color = font_size, shape = font_size,
             y = center,
             ymin = lower,
             ymax = upper)) +
  geom_point() +
  geom_line(aes(group = font_size)) +
  geom_errorbar(width = .1)

G_interaction
```




```
detach(Reading)
```

From Figure XY we can easily spot the two main effects, and how they go together. The researcher would conclude that with the desired gray font, increasing font size partly compensates for the lack of contrast. At the same time, we see that some saturation occurs, but still, from a purely ergonomic perspective, large font and high contrast is the preferred design.

We have seen in the Headache example that interaction effects occur as nonlinearity. 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.

In the Reading case, overall utility is a compound of ergonomic quality and aesthetic appeal. It was assumed that a gray-on-white color scheme is more appealing. The researcher would choose the gray-12pt design: higher contrast would increase ergonomic value, but too much at the expense of appeal. Of course, in a perfect world, one would add emotional appeal as a second measure to the study.

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 measure is 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 as when effort is low. Not including the interaction effect would result in the false conclusion that suggesting high effort actions is rather ineffective.

5.5.3 More than the sum: amplification

Saturation effect 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. These impact factors can be similar (headache pills) or unsimilar, like font size and contrast. Still, they are to some extent exchangeable. We can compensate lack of contrast with larger font size.

Amplification interaction effects are the opposite: They occur, when two (or more) factors impact different cognitive processes in a processing chain. 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 their fund raising endeavor 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:

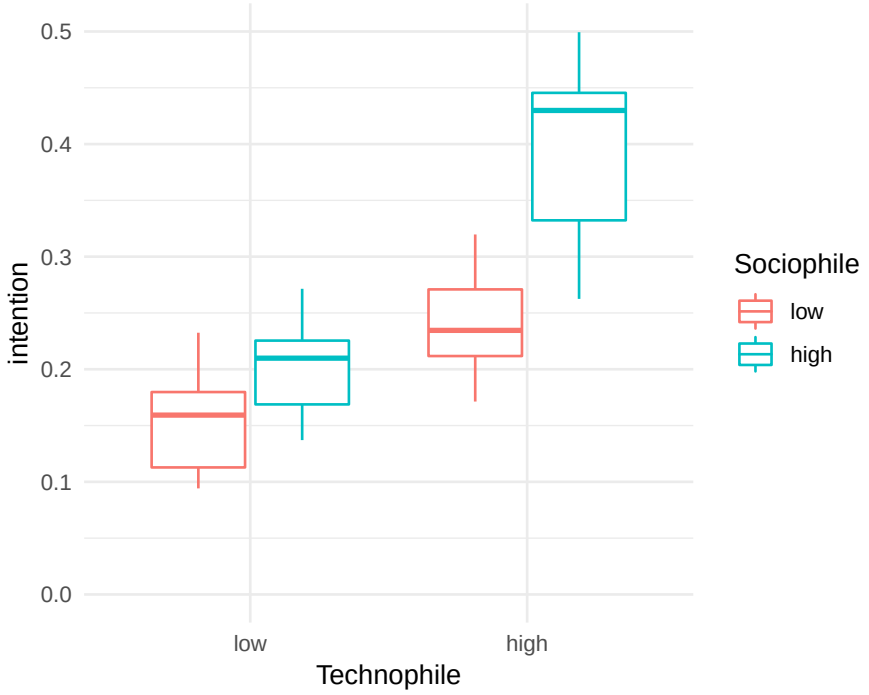
1. Only technophile persons will dare to play the game, because it requires some top-notch equipment.
2. The game is strongly cooperative and therefore more attractive for people with a strong social motif.

In their study they asked a larger set of participants to rate their technophily and sociophily. Then they were given a description of the planned game and were asked how much they intend to participate in the game.

While the example primarily serves to introduce amplification effects, it is also an opportunity to get familiar with interaction effect between metric predictors. While this is not very different to interaction 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 variables. 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 axis 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 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 vague if there is an interaction effect. In absence of a good visualization, we have to rely fully on the numerical estimates of a linear regression with interaction effect.

```
M_1 <-
  D_1 %>%
  stan_glm(intention ~ sociophile * technophile,
           data = .)
```

```
T_1 <- fixef(M_1)
T_1
```

fixef	center	lower	upper
Intercept	0.273	0.259	0.288
sociophile	0.113	0.075	0.153
technophile	0.183	0.152	0.212
sociophile:technophile	0.165	0.082	0.246

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 intention increases overlinearly. 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 true value μ, separated from the random component. 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_1_pred <-
  post_pred(M_1, newdata = D_2, thin =2) %>%
  predict()

D_2 <-
  D_2 %>%
  mutate(intention = T_1_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)
```

Amplification effects are like two-component glue. When using only one of the components you just get a smear. Putting both togeher gives a strong hold. There also is a parallel in Boolean algebra. Recall, the Boolean OR operator returns `TRUE` when one of the operands is `TRUE`, whereas AND requires both operands to be `TRUE`.

```
T_bool_interaction <-
data_frame(A = c(F, F, T, T),
           B = c(F, T, F, T)) %>%
  as_data_frame() %>%
  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

The Boolean OR is an extreme case of saturation. Once one of the factors is present, the result is "positive" and introducing the other one has absolutely no additional effect. Another basic Boolean operation is $AANDB$ with quite the opposite effect: any of the two factors only has an effect, if the other is present, too. Clearly, this is a conditional effect. Rarely will we encounter non-trivial cases where the results are so crisp as in the Boolean world.

5.5.4 Theoretically interesting interaction effects [EDIT]

Explaining or predicting complex behavior with psychological theory is one corner stone of design research. Unfortunately, it is not that easy. While design is definitely multifactorial, with a variety of cognitive processes, individual diferences and behavioural strategies, few psychological theories cover more than three associations between external or individual conditions and behavior. 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 impact of one factor can depend on the overall configuration. A classic study will now serve to show how interaction effects and theoretical reasoning go together.

Vigilance is the ability to endure in attention 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 braking lights flaring 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 ask:

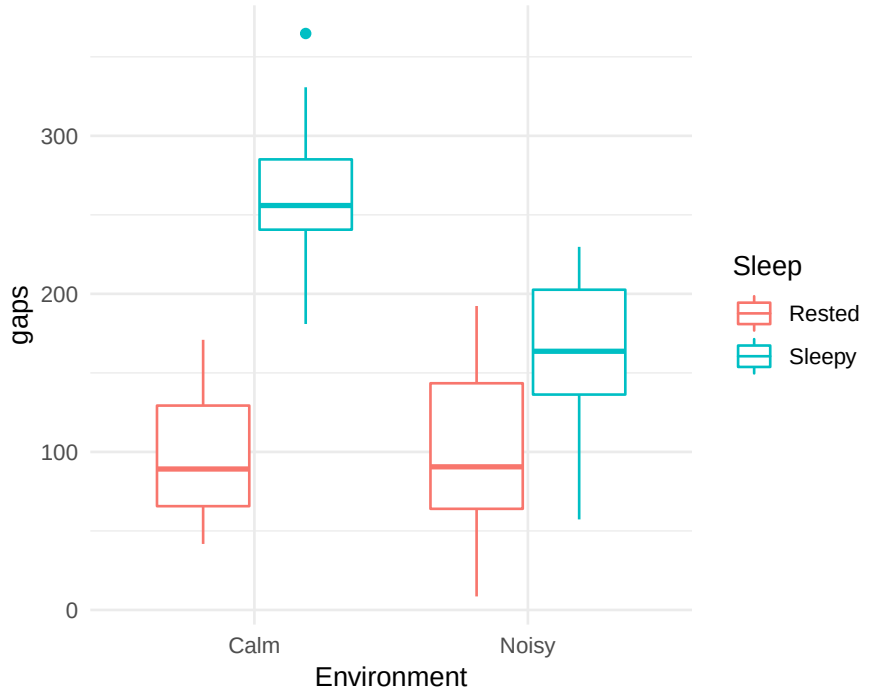
> will the effects of noise summate with those of loss of sleep to induce an even greater performance decrement or will noise subtract from the performance decrement caused by loss of sleep?

The central argument is that sleep deprivation deteriorates a central nervous arousal system. In consequence, sleep deprived persons cannot maintain the necessary level of tension 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, 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 has to react to five signal lamps in a succession of trials. In the original study, as performance measure gaps were counted, which is the number of delayed reactions ($> 1500ms$).

```
attach(Sleep)
```

```
G_expl <-
  D_1 %>%
  ggplot(aes(x = Environment,
             color = Sleep,
             y = gaps)) +
  geom_boxplot()
G_expl
```



Using a 2x2 model including an interaction effect, we examine the conditional association between noise and sleepiness. The `*` operator in the model formula is an abbreviation for the fully factorial term `Environment + Sleep + Environment:Sleep`.

```
M_1 <-
  D_1 %>%
  stan_glm(gaps ~ Environment * Sleep, data = .)
```

```
T_fixef <-  fixef(M_1)
T_fixef
```

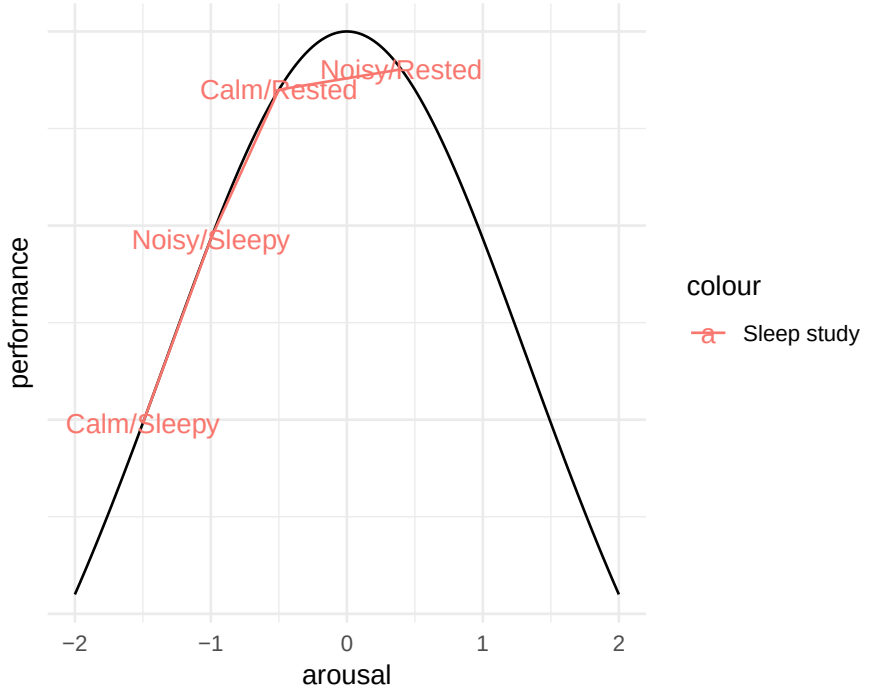
fixef	center	lower	upper
Intercept	99.01	65.1	132.8
EnvironmentNoisy	-0.16	-51.3	51.3
SleepSleepy	160.31	113.5	209.5
EnvironmentNoisy:SleepSleepy	-98.91	-168.6	-29.6

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 $160.32[113.49, 209.55]_{CI95}$. Somewhat surprisingly, noise did not affect well-rested persons by much $-0.16[-51.33, 51.31]_{CI95}$. 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 irritating effect of noise may therefor be minimal. As suggested by Corcoran, the effect of sleepiness on performance is partly reduced in a noisy environment $-98.91[-168.62, -29.62]_{CI95}$. 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.

The finding also relates to the well known Yerkes-Dodson law in ergonomic science. 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 Figure XY. 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 except 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.

Central to the Yerkes-Dodson law is that arousal is a gradually increasing property, but the current experiment only features two levels. A straight line is the only way to unambiguously connect two group means; examining any curved relationships is impossible. We could think of varying noise levels over a wider range for better tracing the non-linear relationship between arousal and performance. The next section introduces polynomial regression for approximating non-linear associations between metric variables.




```
detach(Sleep)
```

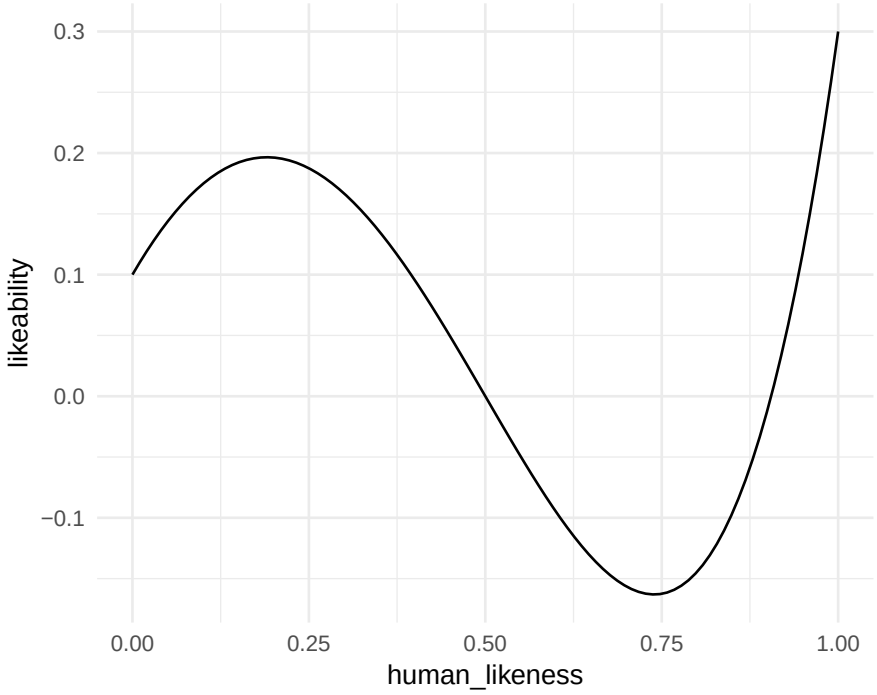
5.5.5 Exercises

1. In case IPump (data set D_agg), two infusion pumps haven been tested against each other (L = legacy, N = novel). Nurses from two professional groups (intensive and general care) completed a sequence of tasks with both devices. In order to account for learning effects, this procedure was repeated in three sessions. Three performance measures were taken: time-on-task, deviations from the optimal path and self-reported workload. Analyze the results using the models that have been introduced so far. Start with basic CGM and LRM, then use these as building blocks for more complex models. At least for the more basic models, always start by an exploratory plot. Then build and run the model. Extract the coefficients table and interpret magnitude and certainty.
2. For the basic models of the previous exercise, extract and plot the residual distribution. Compare the residual distribution by groups.
3. For the more complex models, extract the predicted values and residuals and plot them. Is their an association between the two?

5.6 Doing the rollercoaster: polynomial regression

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 accompany children and care for our seniors. 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 so. [Mori] discovered a bizarre non-linearity in human response: people respond more positive to improvements in human-likeness, but only at the lower end. A feline robot design with recognizable but stylized facial features will always beat a faceless robot with a strong mechanical appeal. Designs on the high end, that are very human-like, but not exactly, face a sudden drop in emotional response, which is called the *uncanny valley*.

```
G_uncanny_illu <-
  data_frame(hl = seq(-1, 1, length.out = 100),
             likeability = -.5 * hl + .6 * hl^3 + .2 * hl^4) %>%
  mutate(human_likeness = (hl + 1)/2) %>%
  ggplot(aes(x = human_likeness, y = likeability)) +
  geom_line()

G_uncanny_illu
```



[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 ranging from mechanical to human appearance. Then they let more than 200 participants rate the faces on how much they liked them. Finally, they created an average score of likability per robot picture and examined the association with human likeness using a statistical model. 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.

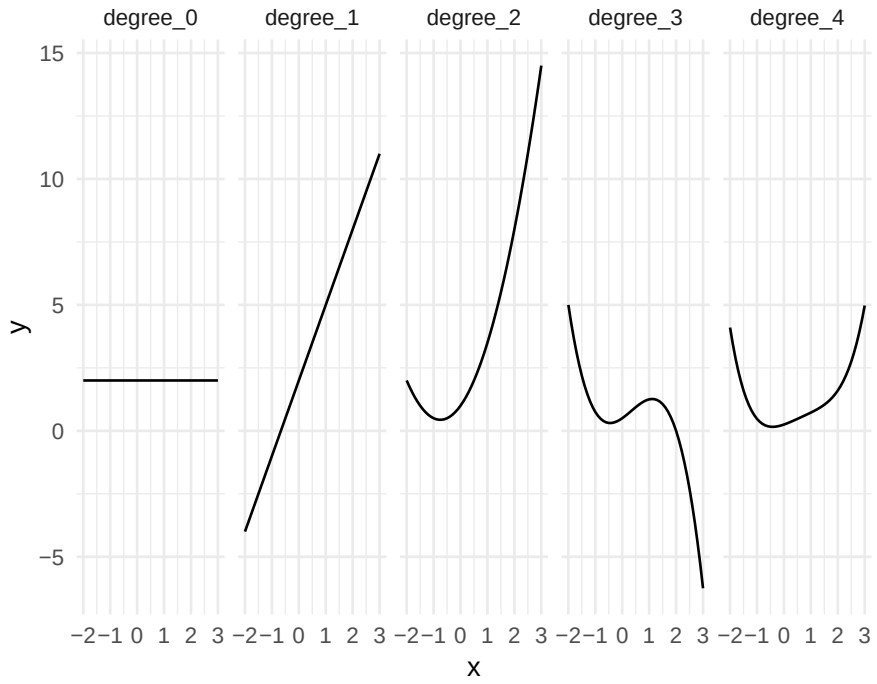
A polynomial function of degree k has the form:

$$y_i = \beta_0 x_i^0 + \beta_1 x_i^1 + ... + \beta_k x_i^k$$

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 just the linear model. By adding higher degrees we can introduce more complex curvature to the association.

```
D_poly <-
  data_frame(x = seq(-2, 3, by = .1),
             degree_0 = 2,
```

```
            degree_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 = 0.5 * (degree_3 + 0.2 * x^4)) %>%
  gather(polynomial, y, degree_0:degree_4) %>%
  arrange(polynomial, y, x)
D_poly %>%
  ggplot(aes(x, y)) +
  geom_line() +
  facet_grid(~polynomial)
```



Mathur et al. argue that the Uncanny Valley curve possesses two stationary points, where the slope is zero. One is a local minimum and represents the deepest point in the valley. The second 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 exponentiated predictors ($x_k = x^k$). 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 <-
  UV_1 %>%
  mutate(huMech_0 = 1,
         huMech_1 = huMech,
         huMech_2 = huMech^2,
         huMech_3 = huMech^3) %>%
  stan_glm(avg_like ~  + huMech_1 + huMech_2 + huMech_3,
           data = ., iter = 400)
  # stan_glm(avg_like ~ poly(huMech, 4),
  #          data = ., iter = 200)

P_poly <- posterior(M_poly)
```

```
# Uncanny$P_1 %>%
#   mutate(parameter = str_replace(parameter, "Intercept", "huMech_0"),
#          fixef = str_replace(fixef, "Intercept", "huMech_0"))
```

```
library(polynom)
```

```
T_fixef_1 <- fixef(P_poly)
T_fixef_1
```

fixef	center	lower	upper
Intercept	-0.207	-0.337	-0.093
huMech_1	-0.525	-0.763	-0.281
huMech_2	0.293	0.060	0.577
huMech_3	0.744	0.368	1.133

We can extract the fixef effects table as usual. The four coefficients specify the polynomial to approximate the average likability responses. The polynomial parameters have little explanatory value. Neither of the parameter 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 with polynomial techniques, we can find this point.

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. They are distinguishable by having a slope of zero, neither rising nor falling. Stationary points can be identified by the derivative of the third degree polynomial, which is a second degree 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, during rise and when falling. A rising encounter of point zero indicates that $f(x)$ has a local minimum at x, a local maximum when falling. In consequence, solving $f'(x) = 0$ can result in two solutions, one minimum and one maximum, which needs 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, when of $f''(x) > 0$.

Mathur et al. followed these analytic steps to arrive at an estimate for the position of the trough. The following code uses several high-level functions from package `polynom` to estimate the location of the trough, by drawing on the first and second derivative `d[d]poly`

```
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  0.37
```

While this procedure is instructive, there is an issue: drawing on the center estimates, which is a summary of the PD, we get a point estimate, only. Statements on certainty are impossible, as a CI is lacking. Recall the 99 seconds study, where we operated directly on the PD to obtain more specific statements on certainty. Another case for directly operating on the PD samples is

to calculate additional statistics. It is another virtue of the MCMC method, that this is possible.

Every PD sample contains simultaneous draw of the four parameters `huMech_[0:3]`, and therefore fully specifies its own third degree polynomial. A PD for the trough parameter can be obtained by performing the above procedure on every sample separately. For the convenience, the case study 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.

```
P_trough <-
  P_poly %>%
  filter(type == "fixef") %>%
  select(chain, iter, fixef, value) %>%
  spread(fixef, value) %>%
  select(Intercept, starts_with("huMech")) %>%
  trough()
```

From the PD, we can derive statements on uncertainty in the usual way: the 95% credibility limits we get by:

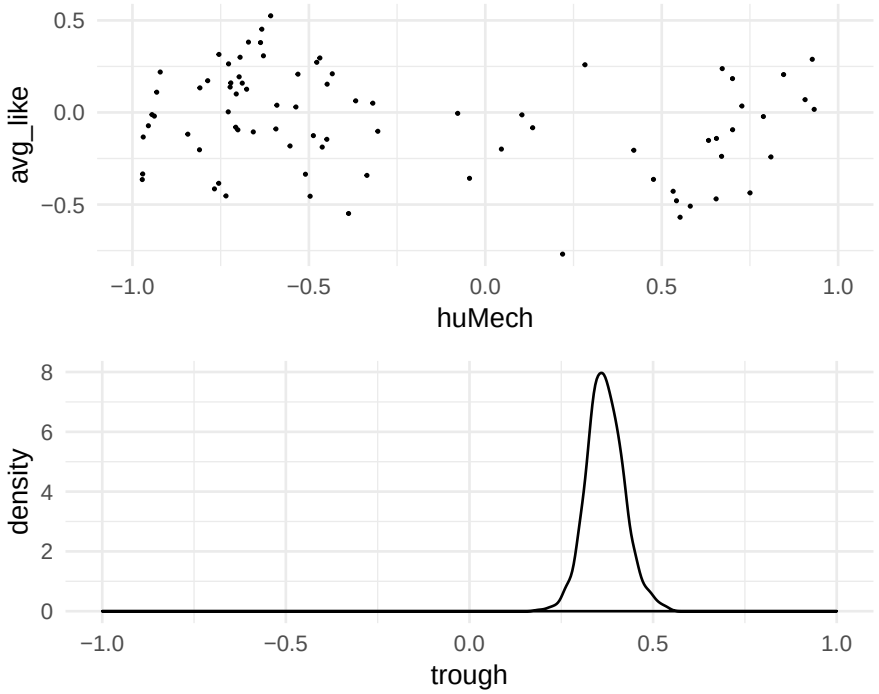
```
quantile(P_trough, c(.025, 0.5, .975), na.rm = T) %>%
  kable()
```

	x
2.5%	0.271
50%	0.367
97.5%	0.479

The 95% CI is a conventional measure of uncertainty and may be more or less irrelevant for the spectator. The most generous account 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.

```
grid.arrange(
  UV_1 %>%
    ggplot(aes(x = huMech, y = avg_like)) +
    geom_point(size = .3) +
    xlim(-1, 1),

  data_frame(trough = P_trough) %>%
    ggplot(aes(x = trough)) +
```

```
    geom_density(aes(x = trough)) +
    xlim(-1, 1)
)
```



With reasonable certainty, we can say that the trough is at approximately two-thirds of the huMech score range. 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. We clearly see that this is not the case in Mathur's study. This might be an artifact, for example in that the huMech score is not linearly related to the perception of human-likeness.

```
detach(Uncanny)
```

6

Multilevel models

In the previous chapters we have seen several examples of differential design 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 interaction effect to the model. Identifying the relevant property is, in fact, a catch-22: how would you know what is relevant before you have conducted the research, actually. 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 difference, really. 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 exist some real geek 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 theoretically derived predictors. For researchers trained in experimental social sciences it may require a bit of Zen to let go of theory-driven reasoning, 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 ($\beta_{1j} = -5$), ...

See how on the participant level, everyone gets their own estimate for preference ($\beta_{1.}$). The key to estimating individual parameters is simply to regard participant (`Part`) a grouping variable on its own, and introduce it as a factor to the linear model. As we will see, it is *almost* that easy. The additional idea is that the levels of the factor, hence the individuals, are part of a *population*. While researchers in fundamental social sciences typically ignore individual variation, there is an independent branch in Psychology called *psychometrics* that deals with precisely that: constructing tests to measuring the individual. Think of IQ tests, which are constructed to have an average of 100, with 67% humans falling in the range 85 - 115. More extreme values become increasingly infrequent. From this perspective, it becomes clear what a population is: a set of entities that do vary but clump around a typical value. These two simple ideas, participants are factors, too, and are part of a population, form what is commonly called *random factors*: a factor where levels are drawn from an overarching distribution, usually the Gaussian. This distribution is estimated simultaneously to the individual parameters ($\beta_{1.}$), which has some interesting consequences, as we will see when discussing shrinkage.

Typically, fixed and random effects appear together in a linear mixed-effects model. As we will see, it depends on the research question whether the researcher capitalizes on the participant-level effects, the average outcome or the variation in the population. As we will also see, the idea of random effects transfers with grace to *non-human populations*, such as designs or questionnaire items. But, first, let me give you an illustration, why caring for individual differences is not just an added value, that you may take or leave, but a necessary perspective to avoid severely wrong conclusions.

6.1 Average? Neverage!

Many studies in design research pretend that all users are alike. This borrowed from experimental social science research. Seminal studies in cognitive psychology and social psychology claim mental structures or processes that hold for all humans. That is the very paradigm of fundamental psychology and many researchers ground their theorizing and experimentation on it. And indeed, several properties of human information processing have been replicated under such diverse conditions that one can assume that they are general. For example, the Stroop observation has been observed a hundred times in various population subgroups [REF], situations [REF] and variations.

The treatments in such experiments can be simplistic to an extreme. Often, participants are asked to respond to colored geometric shapes that light up on a black background by simple key presses. This is called *reductionism* and it has lead to at least some insights how the human mind works. As much as design researchers should draw upon psychological theory, as little useful the reductionist paradigm is, as designs must work in real world situations, often in many different ones, and highly controlled lab experiments just bear to little resemblance to the real world. In particular, by no means can we expect all usersalike to one or more designs. Their minds are complex, too, obscure and unpredictable besides.

> 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.)

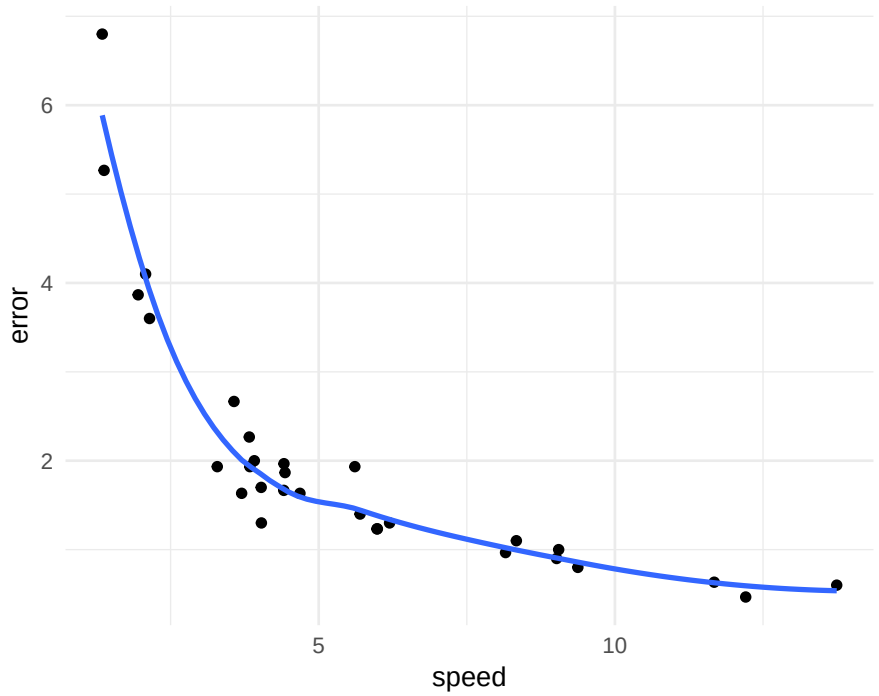
Let me get it straight: People differ! If you plan to move from design A to B, or any other design choice, its is a good idea to also regard how much people vary. If your research question refers to any mental process or its change, the only solid way is to observe how one-and-the-same individuals behave on all conditions. That means within-subject designs should be the rule, not the exception. In contrast, doing a between-subject design is a potential mistake and requires good justification. The following case illustrates how terribly things can go wrong, when within-subject processes are examined in a between-subject manner:

Imagine you set out to determine the association between typing speed and errors. It is common sense that quality decreases with speed, so a positive correlation is expected. During the epxeriment, participants get a number of texts to type. With the first trial they are instructed to type at a relaxed speed. With every further trial they were asked to type slightly faster than at the trial before. The expectation is that

> The faster a person types, the more errors occur

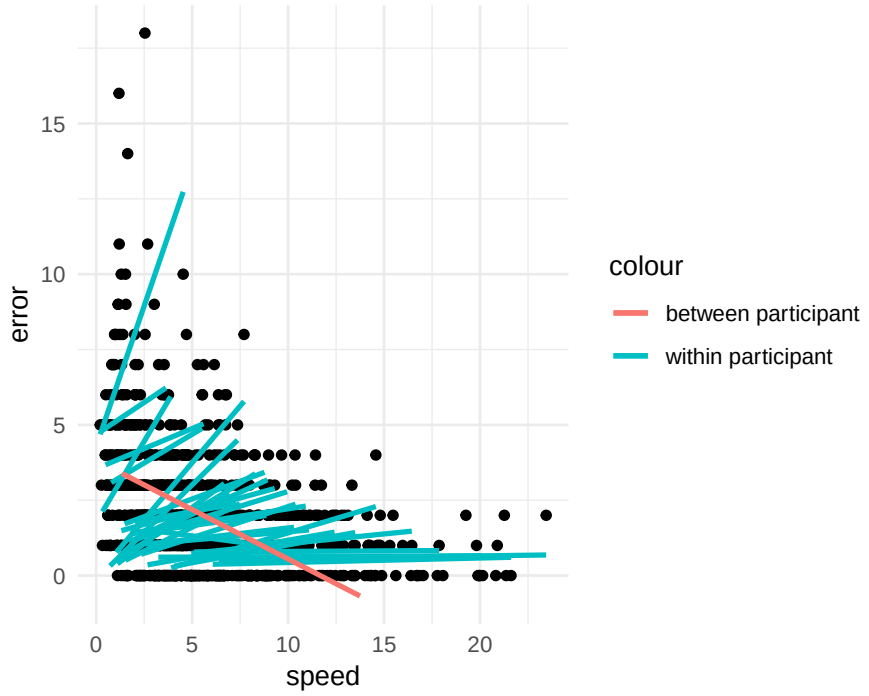
In terms of statistical skills, the researcher is familiar with linear models, and is aware of the fact, that these may not be applied for repeated measures. Therefore our researcher first averages the scores across participants and receives a data set with a neat 30 data points, one per trial. A first exploratory plot reveals a bizarre relation: it seems that the faster participants type, the less error they make. That is completely against expectations, as there almost always is a trade-off between speed and accuracy.

```
attach(Typing)

Type_1_avg %>%
  ggplot(aes(x = speed,
             y = error)) +
  geom_point() +
  geom_smooth(se = F)
```



Could it be true that the faster a person types, the less errors happen? Of course not. The problem is how precisely we ask the question. When averaging over persons, we actually answer a different question, which is on a *population-level*: Do persons who can type faster make fewer errors? This question sees every person as one data point with two performance measures, speed and errors, both representing the ability to type. Now, the above results make more sense: people, who are trained in typing are at the same time fast and more accurate. But, what the experiment was out for is the trade-off between speed and accuracy *within a person*. The experiment provokes this trade-off by asking the same person to accelerate in typing. And, when we visualize the results on an individual level, the speed-accuracy trade-off becomes immediately visible:

```
Type_1 %>%
  ggplot(aes(x = speed,
             y = error,
             group = Part)) +
  geom_point() +
  geom_smooth(se = F, method = "lm", aes(col = "within participant")) +
  geom_smooth(data = Type_1_avg,
              aes(col = "between participant", x = speed, y = error, group = NA),
              se = F, method = "lm")
```




```
detach(Typing)
```

The example here is constructed to be extreme: the population-level effect has the opposite direction of individual level effects. However, the situation is fully plausible and points us to an important principle: Whenever a research question capitalizes on a change in mental state or behaviour, the only solid way to research is a within-subject design. Therefore again: treating averaged data as if it represents within-persons changes is a severe mistake. In the A/B case, for example, one must always be clear about the level. Is it true that

- on average, users prefer B over A?

- or all users individually prefer B over A?

The bizzare situation with the Typing study will have to wait, because we will need all elements of multilevel modelling to resolve it. However, the figure above features the core elements of mutlilevel models:

- persons have different levels for errors at relaxed typing (varying intercepts)
- persons differ in how much the error frequency goes up with speed (varying slopes)
- the better a person performs at relaxed typing, the stronger the number of errors goes up when the person types faster (correlated ramdom effects)

6.2 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 almost every aspect, it is likely that people differ in how they use 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. The data set included further variables, education and gender, that may help to predict performance.

Imagine, you had gathered no predictors at all and observed massive amount of variation in performance. It is reasonable to assume that much of this variation stems from individual differences. That, in turn would mean that a fraction of users perform extremely well, whereas many users fail miserably. If that were true, you would give the advice to invest into redesign that is more inclusive, ironing out the differences, would you not? To make your point, you had to prove that individual differences are the main source of variation.

At first, one might think that the grand mean model would do, take β_0 as the population mean and σ as a measure for individual variation. Unfortunately, it is not valid to take the residual variance as variance between individuals, the reason being that random variation captured by σ is composed of multiple sources, in particular:

- inter-individual variation
- intra-individual variation, e.g. by different levels of energy over the day
- variations in situations, e.g. responsiveness of website
- inaccuracy of measures

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.

Imagine, a pre-study to BrowsingAB was done on the existing design A. Data was gathered in live operation of the website. Real users were tracked on eight tasks. While it was possible to identify the task a user was on and identity of users could be tracked, too. However, as a consequence of privacy protection not a single demographic variable could be gathered. With the simulation function of the BrowsingAB case environment we generate a new data set `SOV` (sources of variance), reduce it to design A and remove all predictors.

```
attach(BrowsingAB)
```

```
n_Part = 25
n_Task = 8
n_Obs = n_Part * n_Task


SOV <-
simulate(n_Part, n_Task) %>%
filter(Design == "A") %>%
select(Obs, Part, Task, ToT) %>%
  as_tbl_obs()
```

We start with some data exploration. The first question we can ask is:

> What is the average ToT in the population?

```
SOV %>%
summarize(mean_Pop = mean(ToT)) %>%
  kable()
```

mean_Pop
146

This question is rather global and does not refer to any individuals in the population. The following question does:

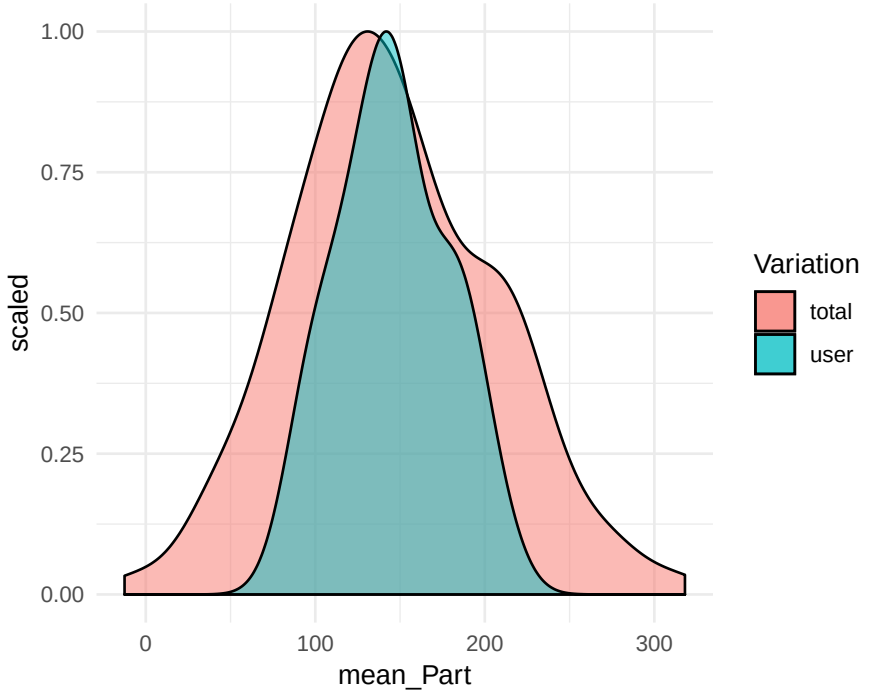
> What is the average ToT of individual participants?

```
SOV %>%
group_by(Part) %>%
summarize(mean_Part = mean(ToT)) %>%
  sample_n(5) %>%
  kable()
```

Part	mean_Part
7	145
14	101
6	128
22	127
12	148

This question differs in that it is *grouped* by participant. Now, that we have the individual means, we can plot the variation in task performance. For comparison, the overall variation is added to the plot. It is apparent that individual differences make only part of the overall variance.

```
G_sources_variation <-
  SOV %>%
  group_by(Part) %>%
  summarize(mean_Part = mean(ToT)) %>%
  ungroup() %>%
  ggplot(aes(x = mean_Part)) +
  geom_density(data = SOV, aes(x = ToT, y=..scaled..,
                               fill = "total"),alpha = 1/2) +
  geom_density(aes(fill = "user", y=..scaled..),alpha = 1/2) +
  labs(fill = "Variation")
G_sources_variation
```




```
detach(BrowsingAB)
```

Above, we have claimed that our data set is bare of predictors. Is it? The variable `Part` groups observations by participant identity and therefore is a plain factor. If we can compute user means like above, what keeps us from adding a factor, like:

```
SOV ~ 1 + Part
```

While formally, a participant is just a group of observations, there is one apparent, one practical and one subtle difference compared to factors as we know them so far. The apparent difference is that before we had just very few levels and many observations. With participants we had to estimate dozens or hundreds of coefficients with just very few observations per level. In consequence, the posterior distribution will become spread out like butter on a toast and certainty will be abysmal. The practical difference is that, while we are interested in the overall variation, the ability of individual users is rather uninteresting. We actually have no use for dozens of user ability scores. The subtle difference is that users form a population. That sounds rather obvious than subtle, but is key for the solution once we understand what being *member of a population* means, statistically. I will give a brief account here and return to the topic in a later section.

Imagine the following situation: you are seated in a university bistro and you get the task to guess the intelligence quotient of every person entering the bistro. After every trial you are disclosed the real IQ of the person. You know that the average IQ is 100 and you give a bonus of 5 owing to the fact it is at a university. The first five persons have the IQs:

```
IQ <- c(106, 108, 103, 115, 110)
```

It seems the bonus of five is an understatement and the average is closer to 110. You adjust your best guess accordingly. That sounds trivial, but for the specification of a regression model it is not. The crucial point is that any guess k depends on the information you received on trials $1...(k-1)$. Review the model formulation for comparison of means models. There is nothing in the linear term that transfers information between levels of factors. The group means are estimated in complete independence.

The key is is that participants are *members of a population*. In a reasonably large population, extremes such as an IQ of 160 will eventually happen, but they are very unlikely. by far most people, roughly two thirds, are clumped together in the range 85 to 115. Imagine you are travelling to an African country you have never heard of before. Africa has the richest gene pool of all and ethnicies differ a lot in tallness and skin tone. Once you have left the plane and saw a few people, you get an impression of how tall and dark people are in this place. The same principle holds for human attributes or capabilities in design research. Having seen a bunch of participants completing a task between two and five minutes gives a best guess for the next particiant. It is not excluded that this person will need 25 minutes or 20 seconds for the task, but it is less likely.

Factors where the individual levels vary, but are more or less clumped around a population mean are best modelled as *random factors* and the individual levels are called *random effects*. Random effects enter the likelihood of the linear model specification in a similar additive way as fixed effects. The difference is that the levels are assumed to stem from a normal distribution, which represents the clumping. The mean of this distribution is zero, similar to the distribution of residuals, but the standard deviation of this distribution, representing the amount of variation in the population, is *estimated alongside* the individual levels.

For the pre-study we have simulated above we can formulate a GM model with participant-level random effect β_{p0} as follows:

$$\mu_i = \beta_0 + x_p\beta_{p0}\beta_{p0} \sim N(0, \sigma_{p0})y_i \sim N(\mu_i, \sigma_\epsilon)$$

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 levels. 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 `stan_glmer()` is dedicated to estimate mixed-effects models with the extended formula syntax.

```
attach(BrowsingAB)
```

```
M_1_SOV <- stan_glmer(ToT ~ 1 + (1|Part), data = SOV)
P_1_SOV <- posterior(M_1_SOV)
```

The posterior of the mixed-effects 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_1_SOV)
```

model	type	fixef	center	lower	upper
M_1_SOV	fixef	Intercept	146	133	159

```
# ranef(P_1_SOV) # a long list
grpef(P_1_SOV)
```

model	type	fixef	re_factor	center	lower	upper
M_1_SOV	grpef	Intercept	Part	26.1	15.5	40.4

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)
```

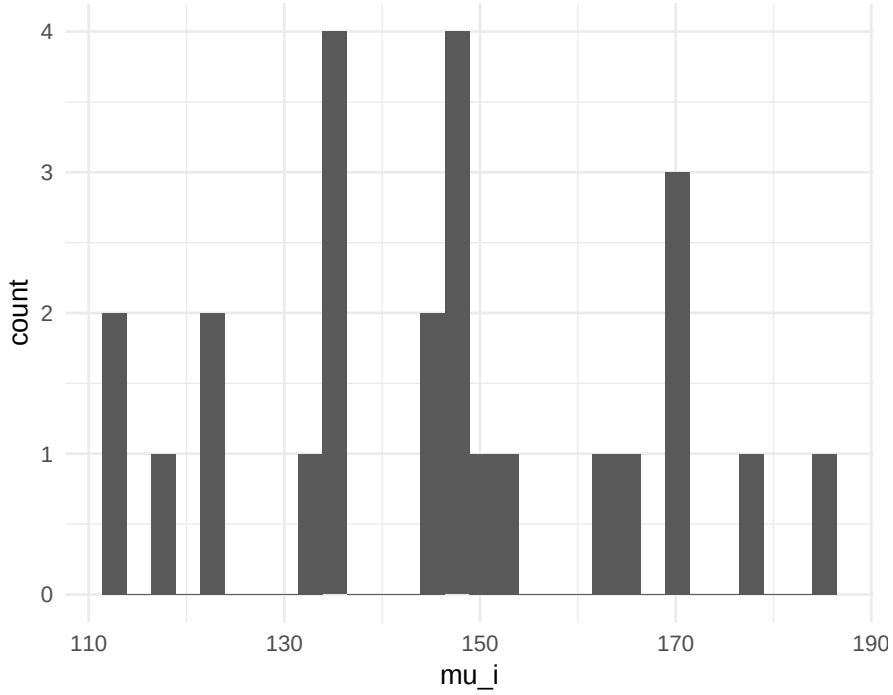
In probability theory expressions, such as famous Bayes theorem, the | symbol means that something to the left is conditional on something to the right. Random effects can easily be conceived as such conditional effects. Left of the | is the fixed effects that is conditional on (i.e. varies by) the factor 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 gray 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:

```
data_frame(mu_i = ranef(P_1_SOV)$center +
             fixef(P_1_SOV)$center) %>%
  ggplot(aes(x = mu_i)) +
  geom_histogram()
```



Finally, we can assess the initial question: is 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. One heuristic is to compare it against the residual variance, which is:

```
T_sov_vc <- coef(P_1_SOV, type = c("grpef", "disp"))
T_sov_vc
```

parameter	type	fixef	re_factor	center	lower	upper
sigma_resid	disp	NA	NA	54.5	49.3	60.9
Sigma[Part:Intercept,(Intercept)]	grpef	Intercept	Part	26.1	15.5	40.4

```
detach(BrowsingAB)
```

The variation due to individual differences is half of the noise, which is considerable. It seems in order to further investigate of why and how users vary in performance, as this is the key to improving the design for all users.

Exercises:

6.3 Designs and other non-human populations

With the introduction of random effects design researchers can make more precise statements about how a design affects the population of users. The population-level average effect gets amended by a measure of variation. Only when this variation is small can one conclude that the effect is homogenous for the whole population. In the BrowsingAB case, this gives valuable information on how well this particular design serves the population of users and whether one should be satisfied with it or not. This is a typical research question in applied design research where one seeks to choose one from a small set of designs. Generally, A/B studies seek to answer the question: *is one design better for the user population?*. This question has two entities: the user and the design. However, it is not symmetric, as users are provided as a sample from which one wishes to generalize, whereas the two designs are exhaustive. There is the choice between the two and they are both present.

But, that is not all there is in design research. 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 the bad. A/B studies, as we know them, are not adequate to answer such questions, simply because there is no reasonable way to generalize from a sample of two to all possible designs. This would only be possible under one strict constraint: the design property in question, let's say font size, must be the only property where A and B differ. Under this constraint, the population of designs is exactly all designs that are constant except for font size. It is left to the reader to contemplate why that is of little value.

If the research question is fundamental, aiming at general conclusions on design, it is almost inevitable to test it on a larger sample of designs. Once we start speaking about samples of designs, there must be a *population of designs*, too. 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 [...]. Several classes of websites count thousands of members, such as webshops and municipal websites.

Even in highly standardized domains such as automotive cockpits the differences between generations, manufacturers and models are noticable to be called individuals from a population. That is precisely the idea of a random effect: entities vary around a clump of commonalities. In consequence, when testing a fundamental design rule, like *large fonts are better to read*, the proposed research strategy is to sample from the population of websites, classify sample members by font size, and test every design with a sample of users. Given our definition of random effects it is easy to see that the regression

model will contain two random effects (at least), one across users, the other across websites. More on that later.

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 population 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 any information. With the more recent break through 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 aspects of their designs. I do not refrain from being very critical about that, however: a well constructed scale (which unfortunately is rather rule than exception in design research) 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:

- phonemes in psycholinguistics
- test items in a psychometrics
- pictures of faces in face recognition research
- products in consumer research
- patches of land in agricultural studies
- flowers in biology

However, most of the time these one or two of these populations are involved at once, whereas in design research the encounter of three is rather typical, and not even the limit, as we will see in the following case study

6.4 Crossover: Testing Egans assumption

In design research, the encounter of users and designs is almost inevitable. All sampling schemes with an encounter of otherwise independent populations produce so called *cross-classified random effects* (CRE). Random effects, that is the individual entities in the sample, are usually of lesser interest, but the group-level variation is a natural measure of diversity in the sample. When performance is diverse, we gain little faith that the situation serves all users well.

But, is variance in performance due to diversity of users, necessarily? At least this is what Dennis Egan claimed almost three decades ago:

> 'differences among people usually account for much more variability in performance than differences in system designs' [1]

This claim has been cited in many papers that regarded individual differences. We were wondering how it would turn out in the third millenium, with the probably most abundant of all application domains: informational websites.

[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.

For the convenience, we chose the very accessible user population of student users. The design population consisted of ten unversity websites, and ten representative tasks on those were selected. During the experiment, every one of 41 participants completed 10 information search items, being of the folling form:

> On website [utwente.nl] find the program schedule [Biology].

As this is a real study, there is some inconvenience ahead. Most synthetic case studies so far have outcome measures that are too good to be true. Recall that the distribution of residuals is assumed to be Gaussian in linear regression models. Unfortunately, that is not how the noise of time-on-task measures and other "temporal" variables is shaped, frequently. The cheap trick of *log transformation* often serves the purpose when not much is at stake. A more disciplined approach to work with time variables is given in [GLM - Time variables].

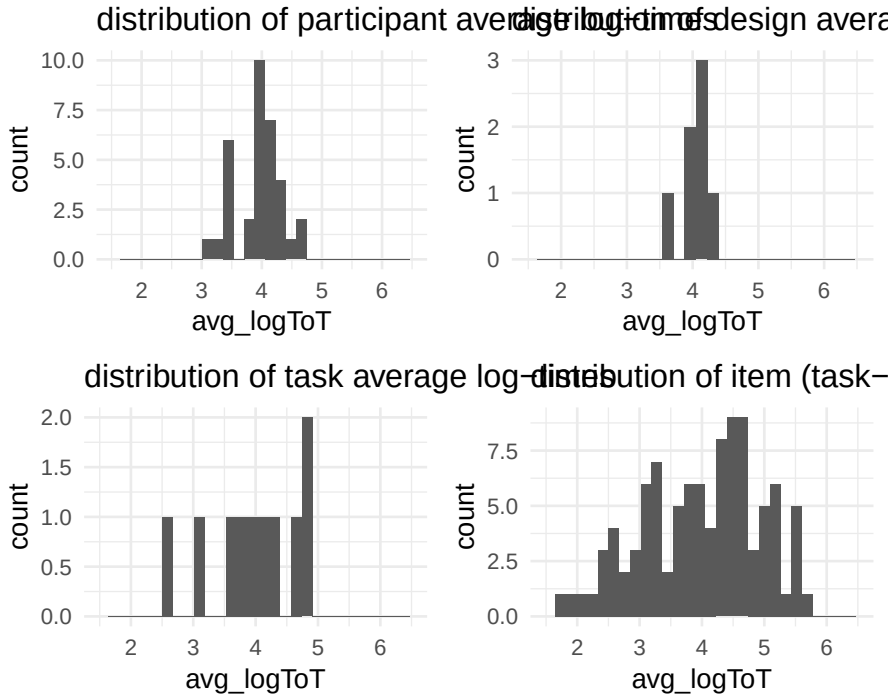
```
attach(Egan)
```

```
D_egan <- D_egan %>% mutate(logToT = log(ToT))
D_egan %>% as_tbl_obs()
```

Obs	Part	Gender	age	trial	Design	Task	success	ToT	clicks	returns	return_home	workload	logToT
39	16	woman	23	5	Erasmus University	student psychologist	TRUE	61	4	0	0	3.50	4.11
82	8	man	22	6	VU Brussel	schedules	TRUE	89	7	1	0	3.00	4.49
391	22	man	24	4	Erasmus University	student psychologist	TRUE	74	3	0	0	3.50	4.30
11	17	woman	21	10	UGent	number of libraries	TRUE	16	2	0	0	2.50	2.77
266	28	woman	20	4	UHasselt	ombudsperson or complaints desk	FALSE	70	2	0	0	2.50	4.25
64	30	man	22	6	Erasmus University	number of faculties	TRUE	6	1	0	0	3.25	1.79
173	32	man	23	10	KU Leuven	opening hours library	TRUE	61	4	0	0	4.00	4.11
250	39	woman	21	4	VU Brussel	ombudsperson or complaints desk	TRUE	99	6	2	0	3.25	4.59

The study originally gathered demographic data and even design features, but for the question of which population varies the most, they are of no further interest. The basic factors in the study were exclusively from human and non-human populations. The latter consisted of designs and tasks in the first place. However, a fourth non-human population has been added: after all, participants get items, which are pairs of tasks and websites. Only seemingly redundant, items will enter the model as a separate random effect. Similar to interaction effects, it may not hold that a task has the same difficulty on every website, and a particular website may present one task in an accessible way, but hide another. The following grid of histogram shows a graphical approximation of the distribution of human and non-human populations. The individual plots were created using the following code:

```
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)
```



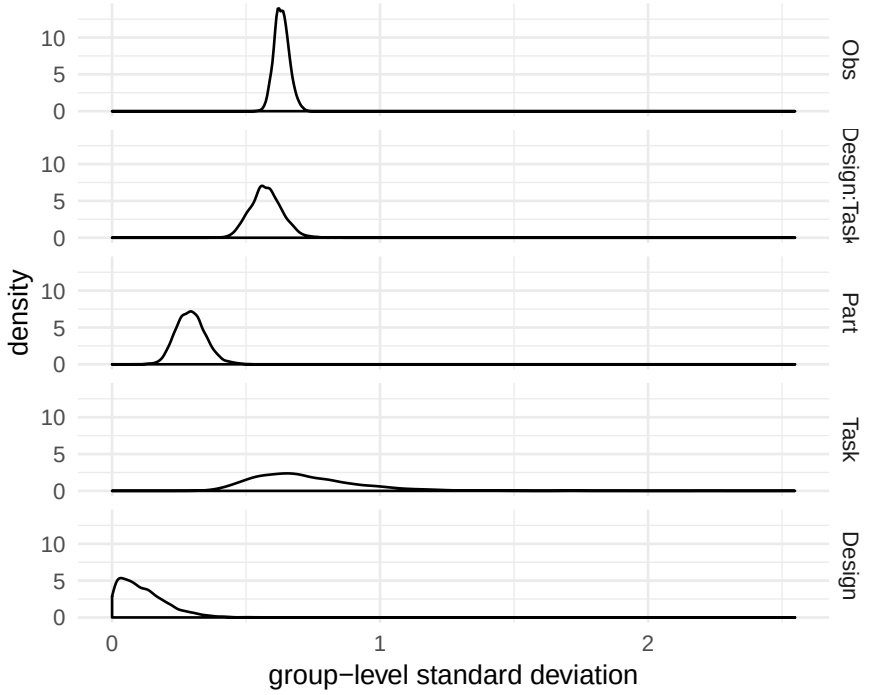
There seems to be substantial variation between participants, tasks and items, but very little variation in designs. We build a GMM for the encounter of the four populations.

$$\mu_i = \beta_0 + x_{Pi}\beta_P + x_{Di}\beta_D + x_{Ti}\beta_T + x_{D:Ti}\beta_{D:T}\beta_P \sim N(0,\sigma_P)\beta_D \sim N(0,\sigma_D)\beta_T \sim N(0,\sigma_T)\beta_{DxT} \sim N(0,\sigma_{DxT})y_i$$

```
M_1 <- D_egan %>%
  stan_glmer(logToT ~ 1 + (1|Part) + (1|Design) + (1|Task) + (1|Design:Task),data = .)

P_1 <-  posterior(M_1)
```

The group-level variations are recorded as standard deviations and in Bayesian estimation framework, such variance parameters have their posterior distributions like any other. While the `grpef` command would conveniently extract a summary of the posteriors, here we choose to get in full contact with the posteriors. In contrast to the linear coefficients, variance parameters are bounded at zero. For this reason, it usually makes little sense to ask whether there is possibly zero variance in a random effect. But, to answer our question, we should rather compare the posterior distributions of all $\sigma_.$, including the error term (Obs).

```
P_1 %>%
  filter(type == "grpef"  | type == "disp") %>%
  mutate(re_factor = if_else(type == "disp", "Obs", re_factor)) %>%
  arrange(order) %>%
  mutate(re_factor = forcats::fct_inorder(re_factor)) %>%
  ggplot(aes(x = value)) +
  geom_density() +
  labs(x = "group-level standard deviation") +
  facet_grid(re_factor~.)
```



Before we formally evaluate Egan's claim, there are a few noteworthy remarks on how to read the posteriors, when the parameter is a group-level standard deviation. First, standard deviation is on the same scale as the measured variable. Usually, this makes standard deviations rather intuitive to interpret. This is, *unless* you have obscured your variable by log transformation, which is the major disadvantage of this procedure. Second, the posterior graph is too easily misinterpreted by confusing the posterior distribution for the group-level distribution. Instead, the amount of variation in a random factor is reflected by the location on the x-axis, while the spread is uncertainty. The population of designs seems to have the lowest variation of all, with users being close second. The observation-level variation (Obs) is nothing but the standard error, σ_ϵ. The observation-level variation sometimes serves well as a reference point to make quantitative statements on variation. For example, there is little confirmation for an overwhelming "users differ!" in this particular study, as there is much more variation from one situation to the other than between participants. A third observation on the posterior plot is that some posteriors are rather certain, e.g. Obs, whereas others are extremely uncertain, especially task-level variation. There is a pattern to explain this: uncertainty of group-level variation estimates heavily depends on sample size from the respective population. Design and Task have a meager $N = 10$, which is why there is so much uncertainty. At the other extreme, every single one of the 410

observations contributes to estimation of σ_{Obs}, resulting in a rather certain estimate.

All interaction effects are potential assaults on generalizability and so are items (`Design:Task`). Without this interaction effect, the model would assume that all ten tasks had the same relative difficulty regardless of the website. And the same goes for website. Given the countless possibilities to structure information on a website, this is a bold assumption. It were as if all ten information architects had previously agreed on how well to support any of the tasks. And, as the results show, there is substantial conditional variation, information architects do not fully agree. Regarding the interpretation, the question is: are we willing to add the *conditional* design effects under what Egan called "designs", or can only complete websites be called designs, with everything underneath being singular forces that amplify of erase each other? This depends on whether you are an optimist or realist designer. The realist thinks that design is a wicked problem science, where a good design a bunch of reasonable compromises, but is never maximum. You may use large fonts to improve reading speed, but that comes at more scrolling time. You may choose to place an awesome video presentation of your university on its homepage, but you are using costly real estate you could use for other frequently used information. In such a view, a design is a set of carefully calibrated trade-offs and must therefore be taken as a whole.

We proceed with a formal check of Egans claim. It can be stated as: is the random effect variation of users larger than variation of designs, $\sigma_{\text{Part}} > \sigma_{\text{Design}}$. One might be tempted to somehow refer to the figure above and evaluate how much the two posterior distributions overlap. It is much easier than that. Recall, that the joint posterior distribution covers all parameters simultaneously at every MCMC draw. Usually, we evaluate the marginal posterior of one parameter at a time for inference. But, it is equally valid to derive further quantities at every MCMC draw and make inference. In the present case, this quantity is a variable with two states: either Egan's claim holds, or it does not. The marginal posterior of such a logical variable is of type Bernoulli, capturing 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 0.944 can count as good evidence in favor of Egan's claim, although it is outshined by other factors, as we have discussed above.

In this case study we have illustrated a number of useful concepts in design research:

1. Design research deals with samples of several human and non-human populations at once. Cross-classified random effects models capture these structures.
2. Group-level standard deviations are captured by the posterior in much the same manner as linear coefficients. The only difference being that they are bounded to zero, which makes null hypothesis useless and can lead to strongly skewed marginal posteriors.
3. When testing Egan's claim, we saw how an exotic hypothesis can easily be answered through deriving further quantities from the joint posterior. The same technique we have previously employed to find the troughs in te uncanny valley polynomials.

```
detach(Egan)
```

6.5 Measuring entities: psychometric applications [TBC]

Traditionally, psychometrics deals with the valid and reliable measurement of person 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*. Few applications even go beyond mere ranking and establish metric interpretations. For example, in an *interval-scaled* test, the researcher may compare differences between participants.

A psychometric test usually contains several items. For example, a test to measure a child's arithmetic skills would contain a set of arithmetic problems and a simple test score would be the number of correctly answered items. In design research, self-report questionnaires are routinely used for one of two purposes: first, research questions sometimes involve traits or abilities of users. For example, one could ask how a persons visual-spatial abilities are related to navigation performance, be it in a complex hypertext environment, body cavities during surgical procedures or the scattered displays in air traffic control centers. The researcher would then administer two tests: an innate ability

test, such as a mental rotation task and a performance test on, let's say, the surgical procedure. This is just twice the standard psychometric procedure. Second, in design research one frequently assesses properties of a design by using standardized 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 scale to measure eeriness.

We begin with the first case, standard psychometrics to assess user characteristics. A well designed test always employs a set of items. The primary reason for multiple items is a familiar one. In classical test theory, the observed test score y_i for participant i is assumed to be composed of the the true score μ_i and the measurment error ϵ_i as:

$$y_i = \mu_i + \epsilon_i$$

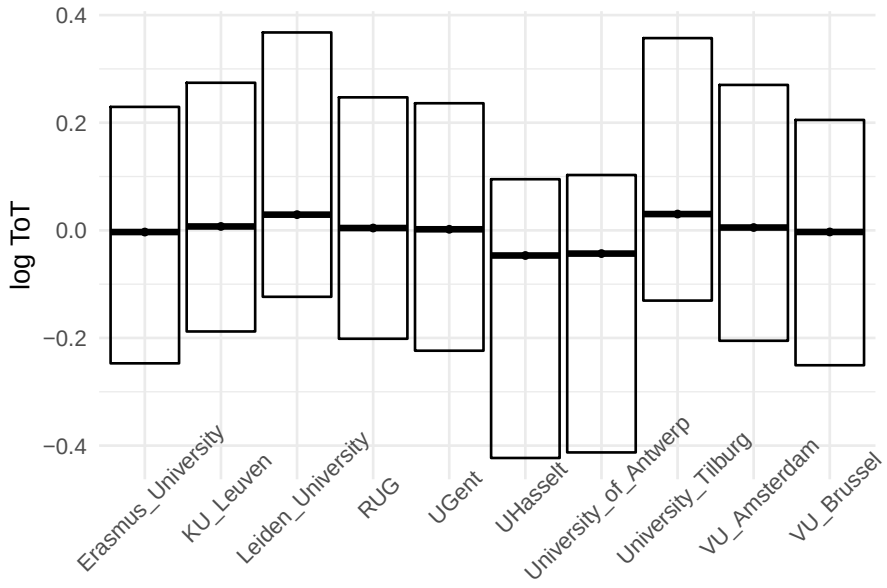
The same principle holds as for linear models: the researchers aim is to separate μ_i from ϵ_i and that is only possible if there is repeated measures.

In our terminology, psychometric models are the encounter of persons and test items. With only two populations involved, the standard psychometric model is a crossover of person and item random affects, a simplification of the quadrupel crossover encountered in the Egan study. What sets psychometric models apart is how random effects are being used. While in the Egan study inference was based on the population-level distribution, i.e. $\sigma.$, in psychometrics applications one is interested in the individual scores. If we wanted to compare two individuals, e.g. which of the two is more likely to become the better surgeon based on some ability test, we need individual scores and that is precisely what random effects are on their lowest level.

[semantic Stroop]

Measuring participant performance is a useful application in some design research fields, such as human performance in critical systems, e.g. identify persons with the best talent for being an airspace controller. However, in design research, it is frequently designs that are to be compared. For example, in the Egan case, one might be interested to select the best of the ten website designs, in order to use it as a blueprint. Such a research question is no longer psychometric, literally, but formally the only difference is that participants can be considered test items, rather than test objects. At least, when using ramdom effects for psychometric purposes, there is nothing special about the participant random effects compared to design random effects. Let's pretend we had conducted the stufy in order to identify a good template for university websites. The following code extracts design random effects posteriors, summarizes them in the usual way and plots.

```
attach(Egan)

ranef(P_1) %>%
  filter(re_factor == "Design") %>%
  rename(Design = re_entity) %>%
  ggplot(aes(x = Design, y = center, ymax = upper, ymin = lower)) +
  geom_point(size = 1) +
  geom_crossbar() +
  ylab("log ToT") +
  theme(axis.text.x = element_text(angle = 45))
```



The most simple form is the Rasch model in IRT, where participants respond to a set of items and the response is either correct or wrong. The outcome variable response is usually coded as 0 = wrong and 1 = correct. Apparently, such a variable does nowhere near satisfy the assumption of linear models. It turns out that the Rasch model can be interpreted as a cross-classified random effects in a *logistic regression* @ref(logistic_regression). Logistic regression is a member of the General*ized* Linear family of models, which will be introduced in the next chapter.

As innocent as the Egan study seems, there is some psychometrics involved. First, cognitive workload was measured using the NASA TLX questionnaire

which has four items. With four items, we would hardly want to speak of a population. The trick is to view the four items as instantiations from all a virtual set of possible questions (on that matter). A typical question in psychometric research is that of *item consistency*, which for a small set of items can be measured by the correlation between items. Inter-item correlations indicate that a scale is reliable, which basically means it predicts well. Item correlations are mainly used during scale development, as a criterion to sort out renegade items, those that do not move in accordance with the others.

```
detach(Egan)
```

6.6 Nested random effects

Nested random effects (NRE) represent nested sampling schemes. A classic example from school research is: 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

As we have seen above, cross-classified models play a primary role in design research, due to the user/task/design encounter. NRE 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 case of a nested sampling structures are some of the Comparative Usability Evaluation (CUE) studies, pioneered 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. 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, didn't we? The CUE8 study lowered the bar, by asking whether several professional groups are able to 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 were responsible for recruiting their own sample of participants. The nested sampling scheme

is therefore: every participant is assigned to exactly one team. The original analysis [CUE8] focussed on the consistency across teams. Using hierarchical random effects, we can extend the analysis. Conjsider the research question: strong systematic variation between teams is considered a sign of low reliability in the field. But, how strong is strong? It is difficult to come up with an absolute standard for inter-team reliability. Again, we can resort to a relative standard: how does the variation between teams compare to variation between individual participants?

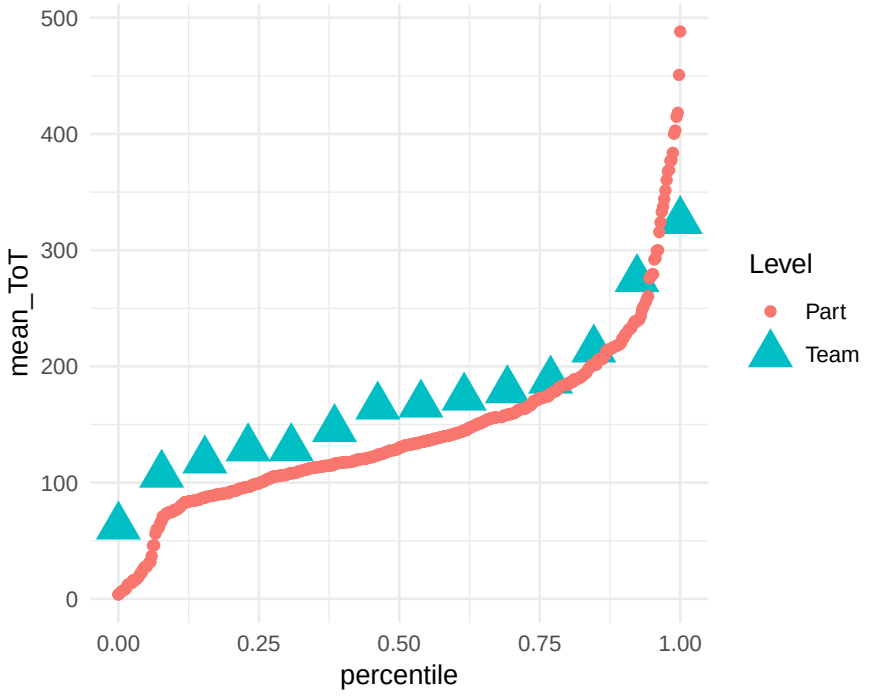
Under this perspective, we examine the data. This time, we have real time-on-task data and as so often, it is highly skewed. Again, we use a cheap trick of logarithmic transformation to obtain more symmetric distribution of residuals. The downside is that the outcome variable may not be zero. For time-on-task 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, exploring the original variable `ToT` on the two levels is in order. The mean ToT is computed for the two random effect levels, participants and teams and plotted in that order.

```
attach(CUE8)

D_cue8
```

Obs	Team	Part	Condition	SUS	Task	ToT	logToT
140	B3	28	moderated	87.5	5	120	4.79
303	D3	61	remote	NA	3	80	4.38
377	E3	76	moderated	NA	2	91	4.51
440	F3	88	remote	NA	5	280	5.63
956	K3	192	moderated	97.5	1	611	6.42
1511	L3	303	remote	80.0	1	111	4.71
1897	L3	380	remote	100.0	2	79	4.37
2416	M3	484	remote	55.0	1	289	5.67

```
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()
```




```
detach(CUE8)
```

It seems there is ample variation in ToT for participants, with mean ToT ranging from below 100 to almost 500 seconds. There is variation in team-wise ToT, too, although less pronounced. This observation is congruent with the *law of small numbers*, which will be discussed in the following section.

The model is easily specified with the familiar syntax. In fact, it is not even necessary to specify that participants are nested within teams. The only requirement is that participant identifiers are unique across the whole study (not just within a team unit). The model below contains another feature of ths CUE8 study. Participants were tested in one of two conditions: moderated and remote testing sessions. Mixing fixed and random effects, we examine all influences simultaneously.

Note that this treatment is between-subject for participants. There is also just one team that did both conditions. For specifying hierarchical random effects we can use the same syntax as before, *if* all participants have a unique identifier. If the participant identifier is only unique within a team, it is either to be recoded, e.g. `mutate(Part = str_c(Team, Part))`, or one uses the nesting operator `Part/Team`.

```
attach(CUE8)

M_1 <-
  D_cue8 %>%
  stan_lmer(logToT ~ Condition + (1|Part) + (1|Team),
        data = .)

P_1 <- posterior(M_1)

detach(CUE8)
```

After running the model, we compare the sources of variation. Recall, that LMM assume that the ramdom effects are drawn from a normal distribution and the amount of systematic variation is represented by σ. As this is precisely the way, normal distributed errors are represented, we can actually distinguish three levels of variation: participants, team and observations (residuals). The table below tells that the strongest variation is still on observation-level. Almost the same amount of variation is due to teams. And, surprisingly, the variation due to teams is considerably smaller than participant-level variation.

```
attach(CUE8)
T_sov <- coef(P_1, type = c("grpef", "disp"))
T_sov
```

parameter	type	fixef	re_factor	center	lower	upper
sigma_resid	disp	NA	NA	0.611	0.592	0.632
Sigma[Part:Intercept,(Intercept)]	grpef	Intercept	Part	0.427	0.388	0.468
Sigma[Team:Intercept,(Intercept)]	grpef	Intercept	Team	0.589	0.405	0.940

It is hard to deny that, at least for consumer systems, people vary greatly in performance. That is the whole point about universal design. The discordance, between professional teams is concerning. And that arises after controlling for an experimental factor, remote or moderated. By the way, the difference between moderated and remote testing is $0.33[-0.31, 1.05]_{CI95}$. The fixed effect is rather weak and highly uncertain.

```
T_fixef <- fixef(P_1)
T_fixef
```

fixef	center	lower	upper
Intercept	4.614	4.093	5.10
Conditionmoderated	0.331	-0.311	1.05

```
detach(CUE8)
```

6.7 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 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, observing some members gives a more or less good guess for unobserved members. In such a case, the predictor is introduced as a fonzy factor. Now, it is in order, 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, 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, in other words: if you have observed how some members of the population behave, you get an idea about the unobserved.

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 given 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 researcher do with repeated measures data is average over the repetitions. This is the one extreme called *total pooling*. In the stopwatch case the average across 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 effects factors as they have been introduced in @ref(linear_models).

Random effects sit right between these two extremes of no and partial 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 over-arching 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 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 $IQ \sim N(100, 15)$, 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 analysis. This is why we tend to use weakly 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 usually is a 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 is certain only to a degree. As we have seen in 6.4, 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, and the number of observations per entity plays a role, too. The more observations there are, the less is the group mean over-ruled by the population mean. Less shrinkage occurs. This is why mixed-effects 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 $\beta_0\ N(0, 10000)$. 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 $ToT_A\ N(120, 40)$, is it a 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, at any time can priors be used 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 varies so dramatically that we should see clear differences in tendency towards the mean.

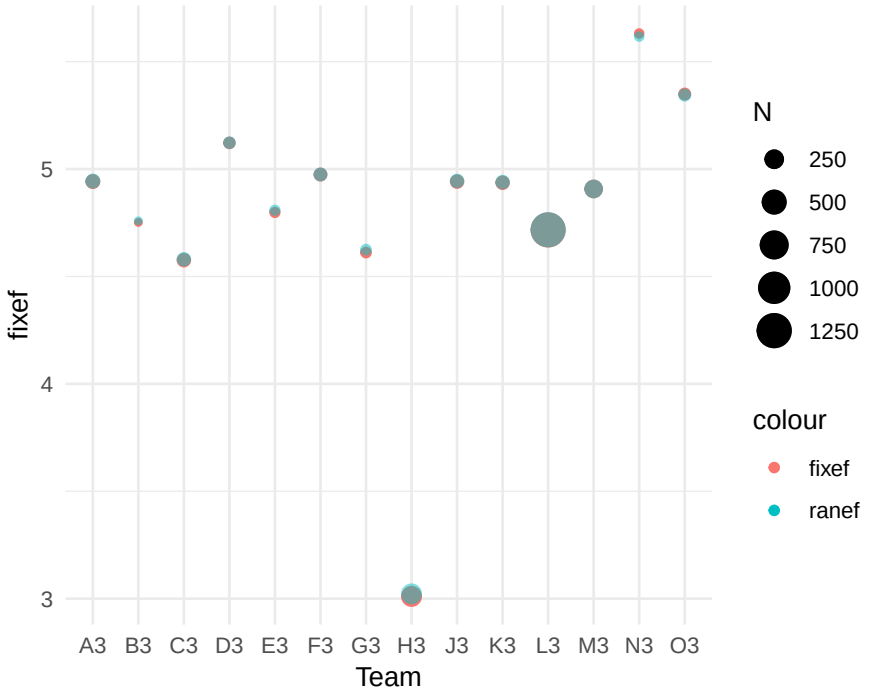
We estimate two models, a random effects model and a fixed effects model. For the RE model, the absolute group means are calculated on the posterior. Figure XY shows the comparison of FE and RE estimates.

```
attach(CUE8)
```

```
M_2 <-
  D_cue8 %>%
  stan_glm(logToT ~ Team - 1, data = .)
```

```
M_3 <-
  D_cue8 %>%
  stan_glmer(logToT ~ 1 + (1|Team), data = .)
```

```
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]

Now that random effects have been de-mystified as effects with partial pooling, where the amount of pooling is a consequence of observed data, we can

move on to applications. The variance in the population is an indicator for heterogeneity of participants (or any non-human population) and random effects variation is an accurate estimator for that. For designers, it is vital to understand how varied performance with a system is. Furthermore, as we will see in the next section, with slope random effects we can even estimate the variation of how users respond to different design conditions. When comparing two designs A and B, that enables the researcher to assess in how much a desig is uniformly better, a desireable property of universal designs. There also are a useful applications for the random effects, i.e., the individual levels. In some applications the crucial issue is to get good estimates of individual performance, which falls into the domain of psychometrics.

6.8 Variance in variation: slope random effects [TBC]

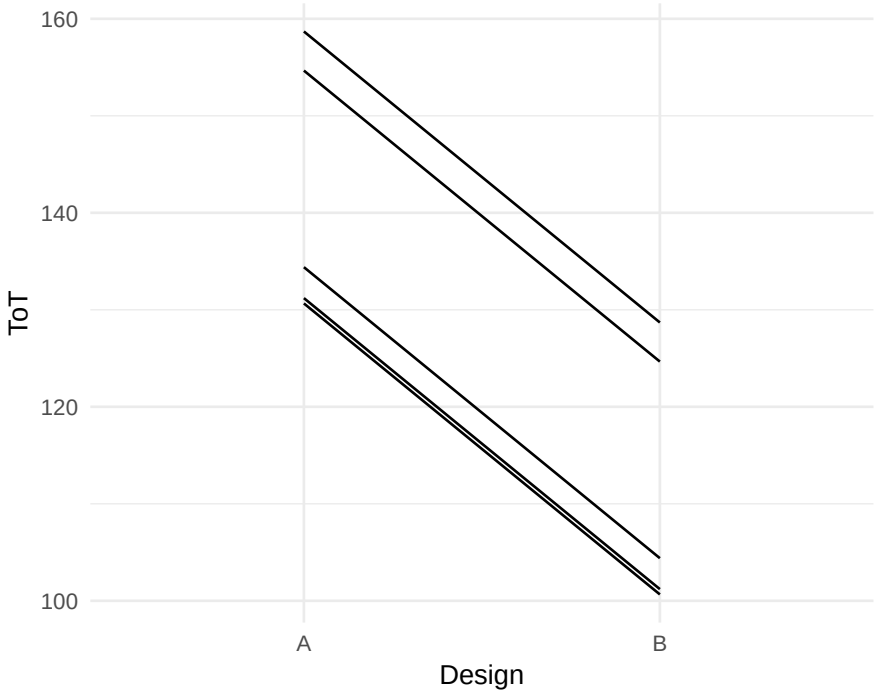
So far, we have dealt with random effects that capture the gross differences of entities: individuals differ in how quick they work, task differ in length and test items in difficulty. We introduced these random effects as conditional effects like: "the intercept depends on what person you are looking at". Whereas in the beginning of the chapter I emphasized the necessity to examine within-person processes in within-subject designs, we never went beyond conditional statements regarding the intercept effect. In contrast consider the A/B testing case. Being more efficient with design B compared to A is the primary question and that we can straight-forwardly put into a conditional statement: "by how much B is more efficient than A depends on the person".

Let's first exlore graphically what this means. Assume that we had tested only five participants with one task each. The CGM model with participant-level random effect takes the following form:

$$\mu_i = \beta_0 + \beta_{0p} + x_1\beta_1\beta_{0p} \sim N(0, \sigma_{0p})$$

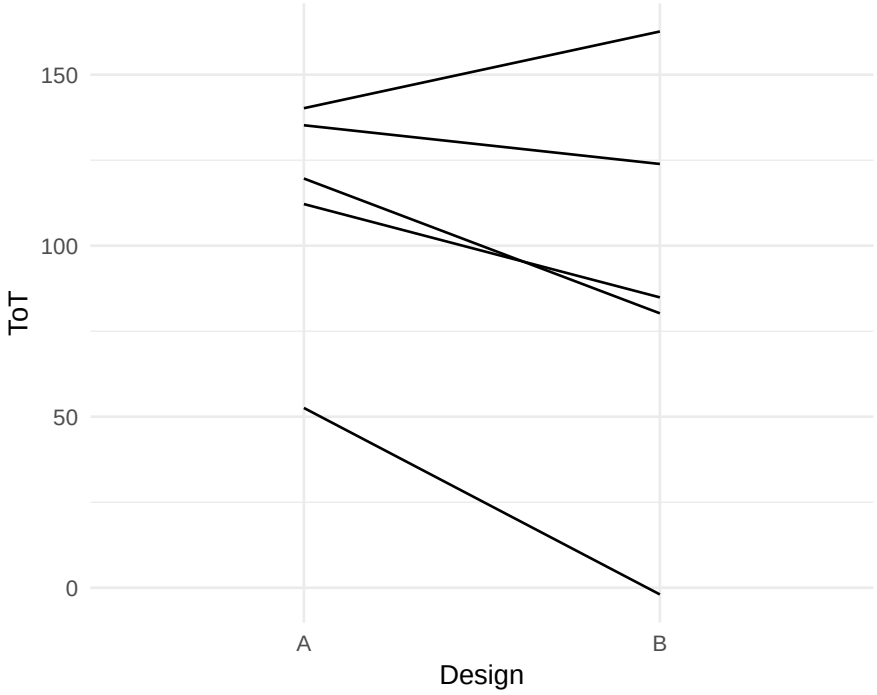
The figure below illustrates the model on a sample of five participants. As the effect is the same, we see five parallel lines. While the height where a line starts differs between users, the all benefit to the precisely same amount from design B. Is that realistic? If we expect that differential design effects exist, it is not. There can be plenty of reasons why two users differ in how much the benefit from a certain design. For example, the BrowsingAB case features a strong interaction effect on age. When thinking of random effects, it helps to imagine the situation´where an influencing factor exists, but was not recorded. The inter-individual variance will remain, and the participant identifier is the only predictor at hand.

```
sim <- function(n_Part = 5,
                beta_0P = rnorm(n_Part,0, sigma_0P),
                sigma_0P = 20,
                beta_1P = rnorm(n_Part,0, sigma_1P),
                sigma_1P = 0,
                sigma_eps = 0) {
  Part = data_frame(Part = 1:n_Part,
                    beta_0P = beta_0P,
                    beta_1P = beta_1P)
  Design = data_frame(Design = c("A", "B"),
                      beta_0 = 120,
                      beta_1 = -30)
  mascutils::expand_grid(Part = Part$Part, Design = Design$Design) %>%
    left_join(Part) %>%
    left_join(Design) %>%
    mutate(mu = beta_0 + beta_0P + (beta_1 + beta_1P) * (Design == "B"),
           ToT = rnorm(nrow(.), mu, 0)) %>%
    as_tbl_obs()
}

sim() %>%
  ggplot(aes(x = Design,
             y = ToT,
             group = Part)) +
  geom_line()
```



A more realistic figure of the effects is the spaghetti plot below. All participants improve with design B, but to rather different degrees. The lines have different starting points, and they differ in slopes. This is called a *slope random effect*. Similar to intercept random effects, slope random effects are factors that represent individual deviations from the population-level effect.

```
sim(sigma_1P = 15) %>%
  ggplot(aes(x = Design,
             y = ToT,
             group = Part)) +
  geom_line()
```



For the design effect in BrowsingAB, a model with intercept and slope random effects is formally specified as:

$$\mu_i = \beta_0 + \beta_{0p} + x_1\beta_1 + x_1\beta_{1p}\beta_{0p} \sim N(0, \sigma_{0p})\beta_{1p} \sim N(0, \sigma_{1P})$$

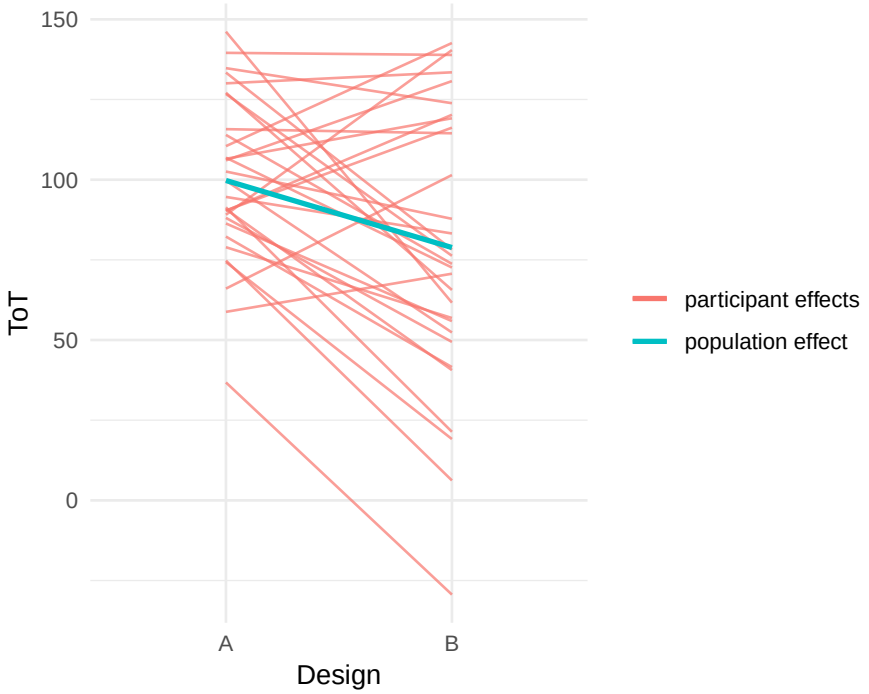
Previously, we have seen that repeated measures are the key to pull a random effect out of the residual term. The same holds for slope random effects. For estimating individual slopes, the same participant (or any non-human entity) must encounter multiple conditions. For demonstration of slope random effects, we examine another instantiation of the BrowsingAB data set `BAB5`, which differs from `BAB1` in two aspects: all participants encounter both designs (within-entity design) and ToT was measured on five given tasks (repeated measures). We use the simulation function as provided by the BrowsingAB case environment. As always, we start with some exploratory analysis.

```
attach(BrowsingAB)
```

The spaghetti plot (the pasta is dry, not cooked) is excellent for visualizing participant-level effects. For the BAB5 data set the following figure shows the individual design effects. While on average design B is favored, there is a wild bunch of slopes. That should not surprise us, as in the previous chapter we

discovered a strong interaction effect by age. Here we ignore this predictor (or pretending not to have recorded it) and exclusively work with the participant factor.

```
G_slope_RE_1 <-
  BAB5 %>%
  group_by(Part, Design) %>%
  summarize(ToT = mean(ToT)) %>%
  ggplot(aes(x = Design,
             y = ToT)) +
  geom_line(aes(color = "participant effects", group = Part), alpha = .7) +
  geom_line(data = group_by(BAB5, Design) %>% summarize(ToT = mean(ToT)),
            aes(X = Design, y = ToT, group = 1, color = "population effect"), size = 1) +
  labs(color = NULL)

G_slope_RE_1
```



The R formula for slope random effects follows the same principles as before. For the present case it reads as "the intercept and design effect are *conditional* on participants": `(1 + Design|Part)`. We estimate the model using the rstanarm engine and use the usual commands to extract population-level estimates and random effect variation.

```
M_Des_ME <- stan_glmer(ToT ~ 1 + Design + (1 + Design|Part),
                       data = BAB5)
```

```
P_Des_ME <- posterior(M_Des_ME)
```

```
detach(BrowsingAB)
```

Mind to always put complex random effects into brackets, as the + operator binds very strongly.

```
attach(BrowsingAB)

M_slope_1 <-
  stan_glmer(ToT ~ 1 + Design + (1 + Design|Part),
             data = BAB5)

P_slope_1 <- posterior(M_slope_1)

detach(BrowsingAB)
```

The random factor variation is extracted from the posterior distribution using the `grpef` command. The coefficients are standard deviations of the two random factors INtercept and DesignB. The results confirm the earlier fixed effects analysis on the data set: users experience very different advantages of Design B. The variation of the difference is almost as pronounced as the variation on Design A (Intercept), alone. The standard deviation has the same magnitude as the group level effect, hence some users are likely to experience a disadvantage, even. That matches the earlier results, where design B caused problems for elderly users. In fact, we can phrase the slope random effect in much the same way as interaction effect `age:Design` in @ref(differential_design_effects), namely as a conditional statement of the form:

> the effect of Design depends on the participant.

```
attach(BrowsingAB)

P_slope_1 %>%
  filter(fixef %in% c("Intercept", "DesignB"),
         type %in% c("grpef", "fixef")) %>%
  group_by(re_factor, fixef) %>%
  summarize(center = median(value, na.rm = T),
            lower = quantile(value, .05, na.rm = T),
```

```
            upper = quantile(value, .95, na.rm = T)) %>%
  kable()
```

re_factor	fixef	center	lower	upper
Part	DesignB	18.9	9.63	33.86
Part	Intercept	17.6	10.48	26.45
NA	DesignB	-20.6	-32.36	-8.93
NA	Intercept	99.5	91.27	108.17

```
detach(BrowsingAB)
```

Intercept random effects capture gross variations between users. Independent of designs, tasks or situation, some users are assumed to always be faster, more accurate or pleased. That is certainly better than sticking with the image of the average user, but it remains static. With slope random effects can we render *individual differences in the process*. How differently do individual users react when proceeding from design A to B? That is a completely different question than the average difference between designs. Review the spaghetti plot @ref(fig:BAB_RE_age) once again. Adding a slope random effect to the model really means that the effect is estimated as many times as there are participants in the sample (run `ranef(M_slope_1)` to see it). And these coefficients tell a story about *generalizeability*. The more the random effects are dispersed, the less typical is the average (the fixed effect). When you are doing a user study for an e-commerce company, it may be sufficient to just tell the average figure (and maybe even a good idea as to not overwhelm the mind of your manager). However, when doing design experiments for the purpose of finding more generally applicable rules for good design, generalizeability matters. In fact, with a CGM that compares to independent groups, you cannot prove that the hypothesized process (e.g., larger fonts are better to read) holds for everyone.

Proving that an effect is practically universal across participants is possible with slope random effects, but requires rendering individual trajectories. That is only possible, if the study assesses the conditions in question for every participant. This is called a within-subject design. Anyone who as trustingly made generalizing conclusions on the basis of a between-subject manipulations, should feel deeply troubled. Can we actually be sure that:

- everyone experiences the uncanny valley effect, when viewing robot faces?
- everyone's alertness when sleepy benefits from noise?
- every nurse makes fewer errors with a third millenium infusion pump design?

Even researchers doing plain user studies cannot refrain from the issue completely. When the system is safety critical, it matters that a particular design

is an universal improvement over the other. Who ever operates it, errors must occur less frequently. In the next chapter, we will encounter such a study, where a novel infusion pump interface was compared with a legacy design. Furthermore, we examined how performance with both devices changed with practice. As the experiment was fully within-subject it is open to investigate generalizeability on both issues:

- do all participants benefit from the modern design?
- do all participants learn over the course of the experiment?

As we will see in 7, most of the recorded performance measures do not comply with the assumptions of linear models and only the measure of perceived workload seems amenable. We construct a model that captures the two main factors (design and session) with a possible interaction effect. Intercept and slope random effects are added for participants and tasks:

```
attach(IPump)
```

```
M_wkld <-
  D_pumps %>%
  stan_glmer(workload ~ Design*Session + (Design*Session|Part) + (Design*Session|Task),
                      data = .)
```

```
fixef(M_wkld)
```

fixef	center	lower	upper
Intercept	21.00	14.13	28.20
DesignNovel	-8.03	-12.30	-3.78
Session2-1	-8.14	-12.80	-3.20
Session3-2	-5.69	-10.32	-1.14
DesignNovel:Session2-1	2.20	-3.64	7.96
DesignNovel:Session3-2	1.62	-4.50	7.59

```
#grpef(M_wkld) FIXME bayr
```

```
posterior(M_wkld) %>%
  filter(type == "grpef") %>%
  group_by(parameter) %>%
  summarize(median = median(value)) %>%
  filter(!is.na(median)) %>%
  kable()
```

parameter	median
Sigma[Part:DesignNovel,DesignNovel]	6.52
Sigma[Part:DesignNovel:Session2-1,DesignNovel:Session2-1]	3.41
Sigma[Part:DesignNovel:Session3-2,DesignNovel:Session3-2]	4.07
Sigma[Part:Intercept,(Intercept)]	12.38
Sigma[Part:Session2-1,Session2-1]	4.79
Sigma[Part:Session3-2,Session3-2]	3.83
Sigma[Task:DesignNovel,DesignNovel]	3.22
Sigma[Task:DesignNovel:Session2-1,DesignNovel:Session2-1]	2.37
Sigma[Task:DesignNovel:Session3-2,DesignNovel:Session3-2]	2.62
Sigma[Task:Intercept,(Intercept)]	6.91
Sigma[Task:Session2-1,Session2-1]	2.81
Sigma[Task:Session3-2,Session3-2]	2.49

```
detach(IPump)
```

6.8.0.1

6.8.0.2 Exercises

1. Review the scientific literature in your field. Find examples of design experiments where general conclusions were drawn from between-subject predictors. If that was easy, visit one of the leading social psychology journals and repeat.

6.9 Growing with random effects correlations [TBD]

Here we finally resolve the Typing paradox and encounter another immensely useful element in random effect modelling. Let's take another good look at the Figure X [Average? Neverage. Last one].

6.10 Advanced tricks

- non-compliant test users in CUE8
- handling missing values by priors

6.11 Exercises

1. In the Egan study, cross-classified random effects capture the encounter of samples. Actually, we have not even taken this to an extreme, as the original study also measured mental workload with a scale of four items. Create and run a model that captures all samples, including the scale items. Interpret the results.

7

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 predictors term. Little space has been dedicated to the outcome variables. 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 the abundance of 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? In the next section we will review these assumptions and lead them ad absurdum. Simply put, in the real world is no such thing as Normal distribution and linearity. Checking assumptions on a model that you know upfront is inappropriate, is a futile exercise, at least when better alternatives are available, and that is the case: with *Generalized Linear Models* (GLM) we extend the regression modelling framework once again.

The GLM framework rests on two extensions that bring us a huge step closer to the data. The first one is a minor mathematical trick to establish linearity, the *link function*. The second is the informed choice about the expected *pattern of randomness*. As we will see, most of the time it is more or less obvious what statistical distribution, other than the Gaussian, matches the data.

In the following three sections I will explain the three core assumptions of linear models, recall the canonical model formulation:

$$\mu_i = \beta_0 + \beta_1 x_1 i + \ldots + \beta_k x_{ki} y_i \sim \text{Norm}(\mu_i, \sigma)$$

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 fall. The second term defines the pattern of randomness and it hosts two further assumptions: *normal distribution* and *constant error variance* of the random component.

Some classic textbooks tend 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 is used to employ a zoo of null hypothesis tests on the data. If one fails, let's say the Kolmogorov-Smirnoff test on normality, researchers often turn to non-parametric tests. Many also just continue with ANOVA, but add some shameful statements to the discussion of results or bravely cite some research paper that claims ANOVAs robustness to violations.

I have met at least one seasoned researchers who divided the world of data into two categories: parametric data, that meets ANOVA assumptions, and non-parametric that does not. All models (or families) in the present chapter, he would have regarded as non-parametric. Let me get this straight:

First of all, *data is neither parametric nor non-parametric*. Instead, data is distributed in some form and a good model aligns to this form. 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 betrween individuals. 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 are often called *semiparametric*. [CROSSREF]. As an example for a *non-parametric* test, the Mann-Withney 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 no 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 For example, 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.

Here, I will not present any tests on assumption, other than exploratory plots. Non-parametric tests are completely at odds with the philosophy of this book,

as they don't produce parameters that can be interpreted quantitatively. Instead, I will right-away debunk all three assumptions for those type of measures design researchers are routinely dealing with. Every assumption that crumbles, rebounces from two new building blocks that add to our regression framework:

1. *link functions* re-establish linearity
2. *random distributions* cover the expected pattern of randomness and the relation of mean and variance

By these two concepts, the *GLM* framework rises from the ashes of *LM*. It hosts a variety of models that leaves little reason for crude approximations, aka the *Gaussian LM*, let alone non-parametric procedures and data transformations. There almost always is a reasonable choice that largely depends on the properties of the response variable: the *Poisson LM* is the first choice for outcome variables that are counted (with no 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 members have been around for more than half a century in statistics. 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 primarily the exponentially-modified Gaussian *(ExGauss) LM* and, to some extent, *Gamma LM*. Along the path, the same chapter introduces a basic way to choose between response distributions based on model comparison. For binned rating scales, where response 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*.

7.1 Debunking the Gaussian linear model

The Gaussian linear model makes many assumptionsl, where numbers often vary between sources. In my view, the three crucial assumptions are:

1. *Linearity* of the association between predictors and outcome variable.
2. *Normal distribution of responses*
3. *constant variance of response distribution*

Researchers routinely check these assumptions by means of visual exploration or null hypothesis tests. However, on closer examination, it turns out that the first two assumptions strictly cannot be true for any real data, and the third is highly susceptible, at least.

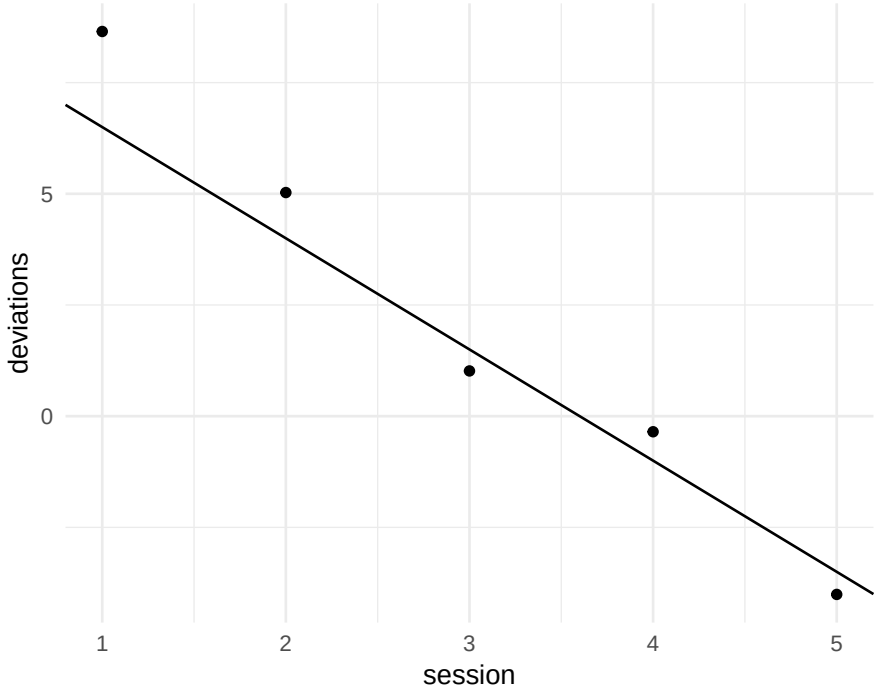
7.1.1 Assuming linearity

Recall the principle, "all-finite-in-the-endless": when two or more interventions improve the same process, e.g. visual recognition of letters, the sum is less than the summands. This results in a non-linearity when the boundary of performance is reached. With a small set of predictors this can gracefully be modelled as saturation interaction effects.

Consider a study that assesses the improvement of safe operation with continued practice. For simplicity, we regard just a single nurse whose number of errors were measured on a chain of 8 tasks. Errors in operation were measured as number of deviations from the shortest possible interaction sequence.

We simulate a linear model, assuming there is an improvement of one error less with every repetition of the sequence, with expected 7 deviations in the first session. We assume that path deviations has a normally distributed random component with $\sigma = 1$.

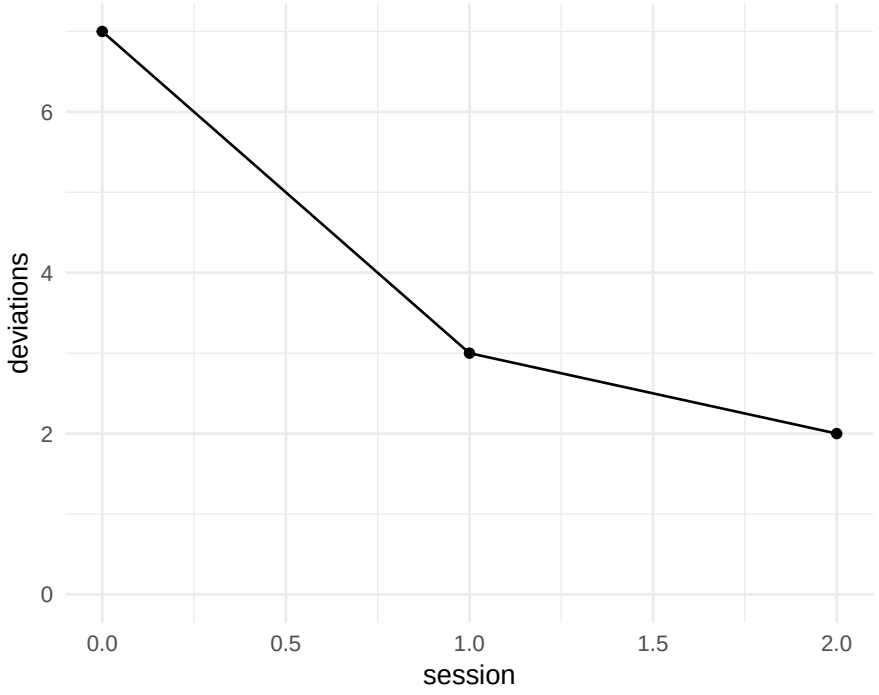
```
data_frame(session = as.integer(1:5),
           mu = 9 - session * 2.5,
           deviations = rnorm(5, mu, sd = 1)) %>%
  ggplot(aes(x = session,
             y = deviations)) +
  geom_point() +
  geom_abline(aes(intercept = 9, slope = -2.5))
```



See what happens when the linear model is naively applied to the deviation counts. In no time, negative values are produced, which is an impossibility. Similar to saturation effects, we would expect some asymptotic behaviour, rather than a straight line, when the outcome variable approaches its natural lower boundary.

Here is some real world data from the IPump study. We take a look at a small slice of it: the total number of deviations of one participant across the three sessions.

```
attach(IPump)

D_pumps %>%
  filter(Part == 5, Design == "Novel") %>%
  group_by(Part, session) %>%
  summarize(deviations = sum(deviations)) %>%
  ggplot(aes(x = session, y = deviations)) +
  geom_point() +
  geom_line() +
  ylim(0,7)
```




```
detach(IPump)
```

What really happens, when a performance measure approaches its natural limit, is an asymptotic leaning-on. Neither will the line break through the limit, nor will it stop there abruptly.

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. All outcome variables in design research suffer from the problem of impossible predictions as a consequence of their boundedness:

- Errors and other countable incidences are bound at zero
- ToT is bounded at zero and probably also bound above when users loose their patience
- Rating scales are bound at the lower and upper extreme item
- Task completion has a lower bound of zero and upper bound is the number of tasks. Or the average task completion is in the range $[0, 1]$

7.1.2 Assuming Normal distribution of randomness

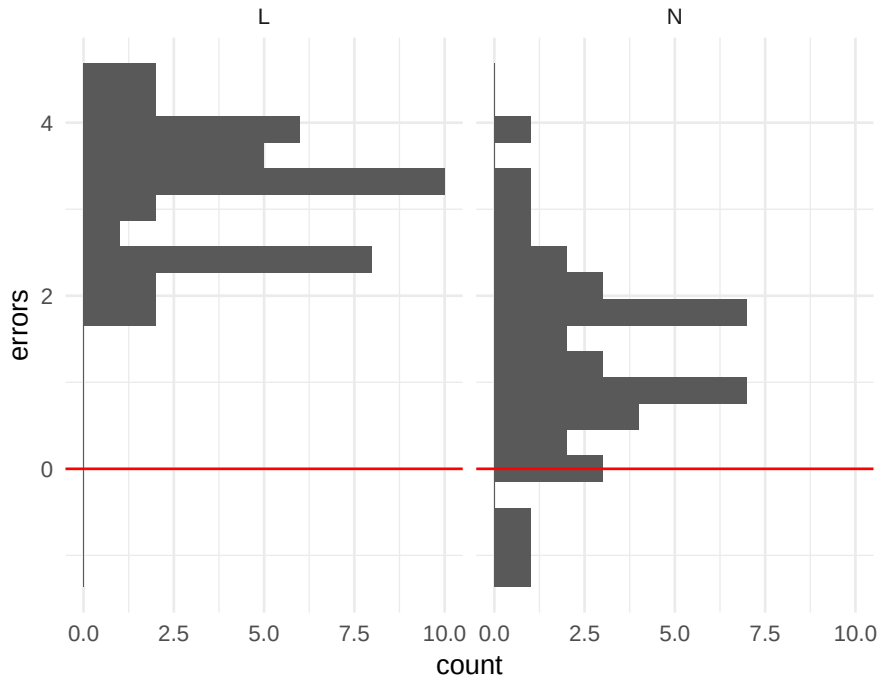
The second term of a linear model, $y_i \sim Norm(\mu_i, \sigma)$ states that the observed values are drawn from normal distributions (see @ref(resid_normality)). Two observed values y_i and y_j are only drawn from the same distribution, when they have the same expected value $\mu_i = \mu_j$. The Normal distribution has been used ubiquitously in statistics. But in fact, the normal distribution is a reasonable approximation when the measures are far off the boundaries of measures and the error is much smaller than the predicted values (@ref(normal_distributions)).

In design research studies this is frequently not the case. We begin with a simulated 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
pumps_2 <-
  data_frame(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:

```
pumps_2 %>%
  ggplot(aes(x = errors)) +
  facet_grid(~Design) +
  geom_histogram(bins = 20) +
  geom_vline(col = "red", xintercept = 0) +
  coord_flip()
```

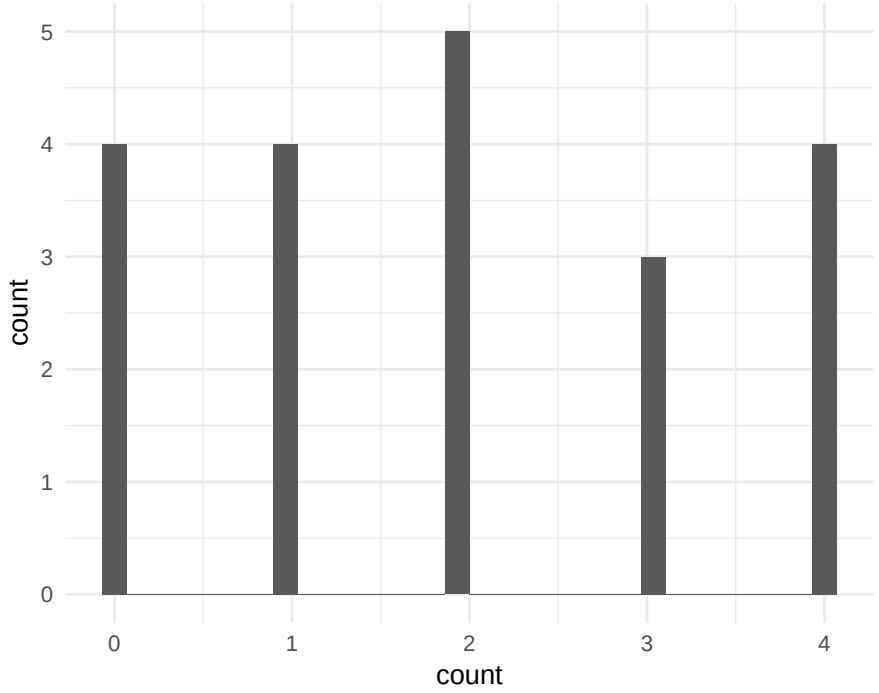


The simulated predicted values (μ_i) and errors are in a fairly realistic range. Still, we immediatly see, that simulation with Normal distributions is rather inappropriate: a substantial number of simulated observations is *negative*, which strictly makes no sense for error counts. The pragmatic and 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 for the following two reasons. First, Normal distributions support the full range of real numbers. 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), obviously is poor practice. 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 bended 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 Binomial and Poisson distribution in chapter 1 for illustrations. As a matter of fact, a lot of real data in design research is skewed that way, whereas the normal distribution eternally claims symmetry.

A common misconception is that random distributions approach the normal distribution with larger sample sizes. The only thing that happens is that increasing the number of observations renders the true distribution more fidel. Examine yourself, how the shape of Poisson distributions changes by mean count, as well as sample size. The following code simulates a Poisson distribution with a given λ and N. With every new run of the code, `rpois` generates a new random set of counts. By repeatedly running this code and watching the output, you can get a good idea of how varied the shape of the distribution can be.

```
data_frame(count = rpois(n = 20,
                         lambda = 2)) %>%
  ggplot(aes(x = count)) +
  geom_histogram()
```



7.1.3 Assuming constant variance of randomness

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 Normal distribution's dispersion is exclusively determined by σ. That is less harmless than it may sound. In most real data, the

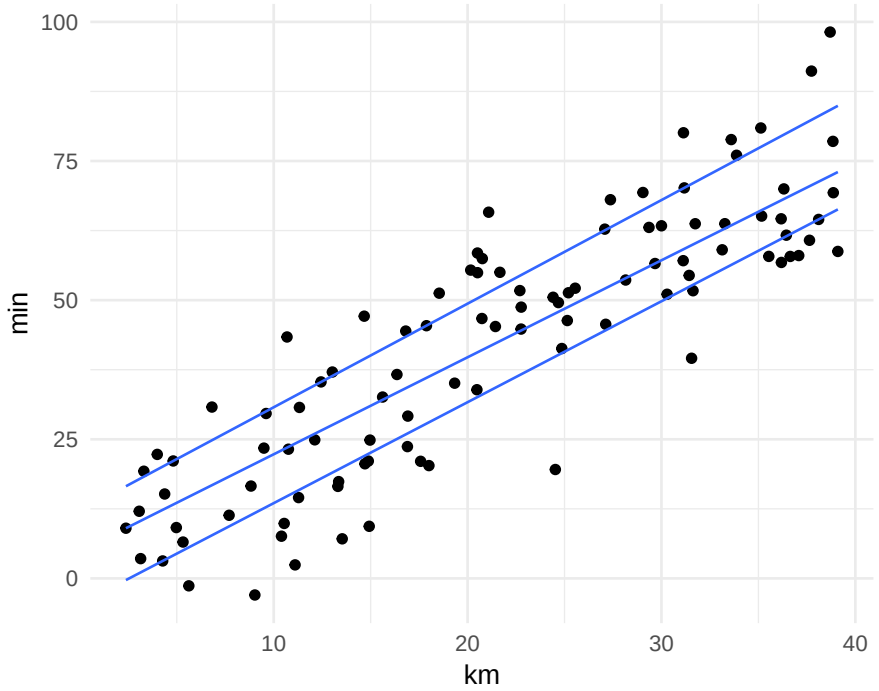
dispersion of randomness depends on the expected value, as can be illustrated by the following example.

Imagine a simple survey where commuters are asked three questions about their daily way to work:

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 inevitably look like a evenly wide band.

```
N = 100
data_frame(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))
```

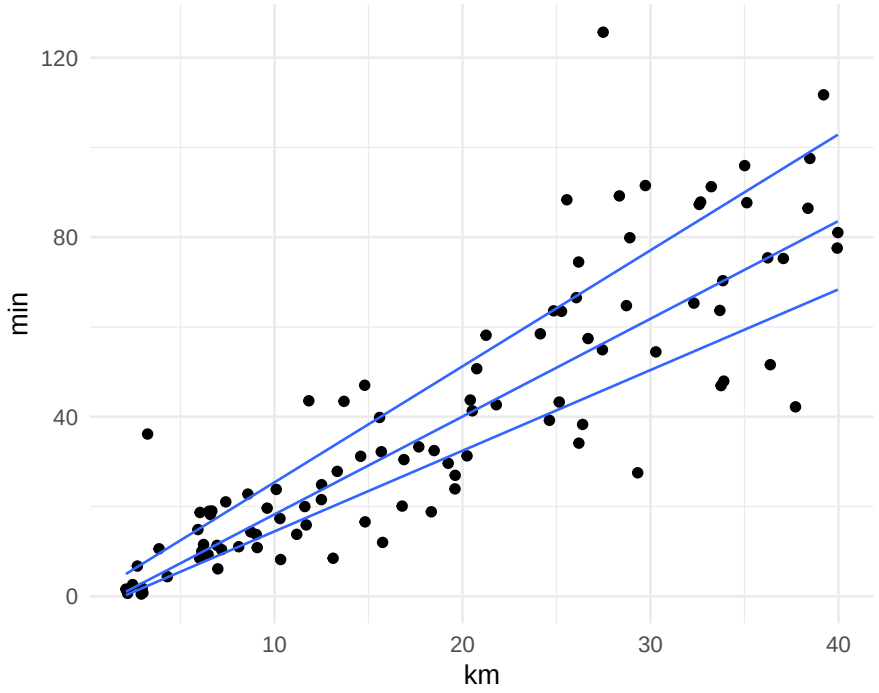


It is very unrealistic that persons who live right around the corner experience the same range of possible travel times than people who drive dozens of kilo-

meters. Most of the time, we intuit the dispersion of randomness to increase with the magnitude of the expected value. 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.

$$X \sim \text{Gamma}(\alpha, \theta) E(X) = \alpha\theta \text{Var}(X) = \alpha\theta^2 \text{Var}(X) = E(X)\theta$$

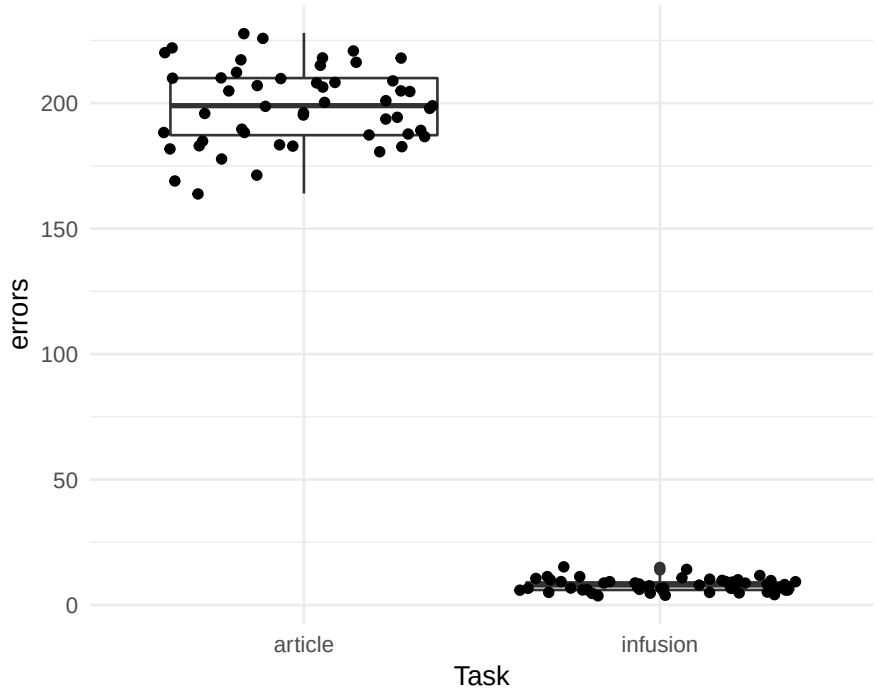
```
data_frame(Obs = as.factor(1:100),
           km  = runif(100, 2, 40),
           min = rgamma(100, shape = km * .5, scale = 4)) %>%
  ggplot(aes(x = km, y = min)) +
  geom_point() +
  geom_quantile(quantiles = c(.25, .5, .75))
```



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 good choice and for Poisson distributed variables, mean and variance are both exactly determined by the Poisson rate parameter λ, and therefore linearly connected.

$$X \sim \text{Poisson}(\alpha, \theta)\lambda = E(X) = \text{Var}(X)$$

```
data_frame(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()
```



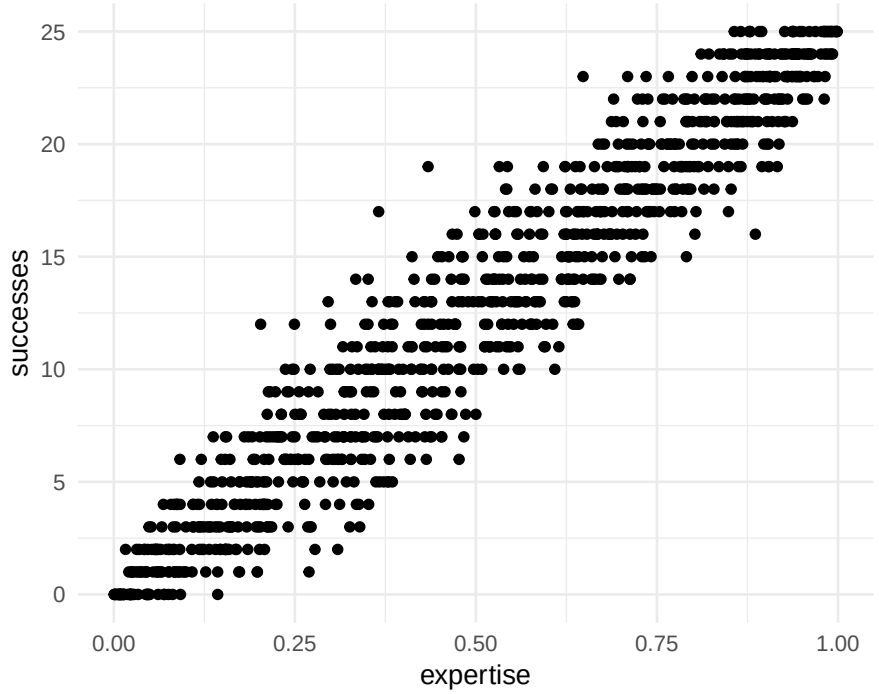
Not by coincidence, practically all distributions with a lower boundary have variance increase with the mean. Distributions that have two boundaries, like binomial or beta distributions also have a mean-variance relationship, but a different one. For binomially distributed variables, mean and variance are determined as follows:

$$X \sim \text{Binom}(p, k)E(X) = pk\text{Var}(X) = p(1-p)k\text{Var}(X) = E(X)(1-p)$$

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-normal distributions.

```
data_frame(expertise  = runif(1000, 0, 1),
           successes = rbinom(1000, 25, expertise)) %>%
  ggplot(aes(x = expertise, y = successes)) +
  geom_point()
```



In conclusion, the Normal distribution assumption is flawed in two ways: 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.

7.2 Elements of Generalized Linear Models

GLM is a *framework for modeling* that produces a *family of models*. Every member of this family uses a specific *link functions* to establish linearity and

chooses a particular *random distribution*, that has an adequate shape and mean-variance relationship.

Sometimes GLM are confused as a way to relax assumptions of linear models, (or even called non-parametric). They absolutely are not! Every member of its own 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 (aks 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* GLM, as they possess some convenient mathematical properties, that made efficient estimation possible, back in the days of expensive computer time.

7.2.1 Re-linking linearity (#relinking_linearity) [TBC]

The strength of the linear term (the likelihood) is its endless versatility in specifying relations between predictor variables and outcome. Unfortunately, it represents all associations as straight lines. These lines extend from $-\infty$ to ∞ and will cross the lower or upper boundaries of every known outcome variable. All linear models predict events that cannot happen.

Generalized linear models use a simple mathematical trick to keep the linear term, but confine the expected values to the natural boundaries of the measures. In linear models, the linear term is mapped to expected values, directly, causing the before mentioned problems. In GLM, a layer is drawn between expected value μ and the linear term, *linear predictor* θ. The *link function* 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.

In arithmetics an abundance of functions exists for every possible purpose. However, link functions must fulfill two criteria,

1. they must map to the range $[-\infty; \infty]$, as that ensures linearity
2. they must be monotonically increasing

Intuitively speaking, a monotonically increasing function preserves the order in magnitude, such that the following holds for a link function.

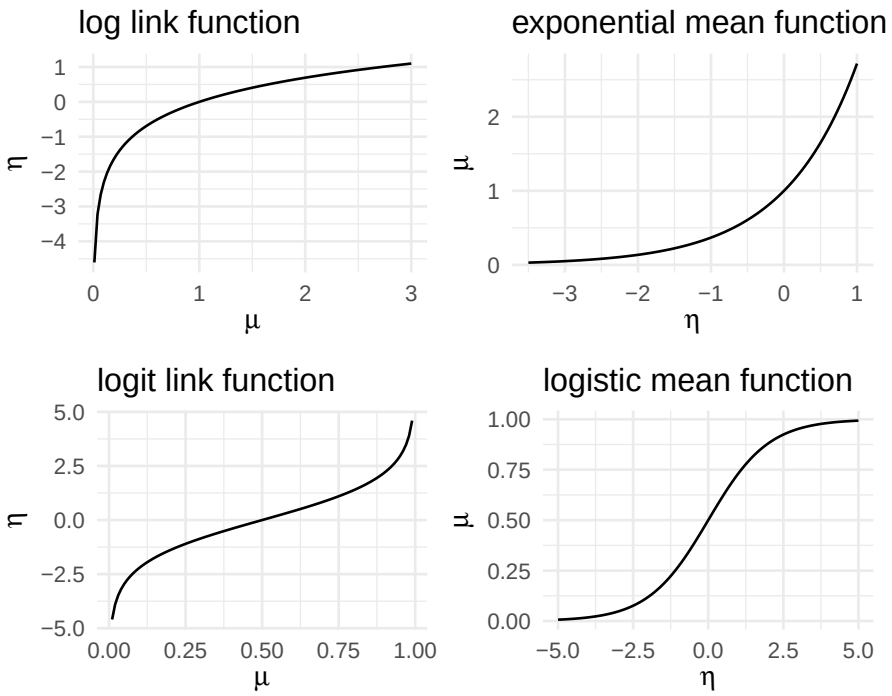
$$\mu_a > \mu_b \rightarrow \theta_a > \theta_b$$

The primary reason for this requirement is that for a link function ϕ there must exist the inverse, that is the mean function (ϕ^{-1}). A function that is *not* monotonically increasing, such as x^2 does not have an inverse function. 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, strictly.

An adequate link function for count variables would map the range of natural numbers 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. Figure XY shows them side-by-side. Note that the logarithm is *not asymptotic* as it may seem. This function truly approaches ininity, albeit at a decelerating pace (which is mind boggling). 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 <-
    data_frame(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 no longer has a natural interpretation, like "moving one unit on the predictor lets the outcome change by β_i". Later will see that logarithmic and logit scales gain an intuitive interpretation when parameters are exponentiated, $\exp(\beta_i)$ (7.4.1 and **??**

7.2.2 Choosing patterns of randomness (#choosing_randomness)

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.

In GLM, the researcher has a larger choice for modelling the random component and Table XY 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 **??** and 7.4.1.3, Poisson distribution can still be used in such cases with some additional tweaks.

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. In contrast, most continuous distributions assume that measures near zero are possible, at least. In case of response times, we will take advantage of the fact, that modern Bayesian estimation engines support a large range of distributions, by far exceeding the available choices in asymptotic methods of frequentist statistics. The `stan_glm` regression engine has been designed with downwards compatibility in mind, which is why it only includes the classic distributions. Luckily, there is a sibling engine in the package `brms`, which is more progressive and gives many more choices.

Still, using distributions that are not Gaussian sometimes carries minor complications. Normal 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 (note, however that this is not a rigorous method for model selection). In random distributions with just one parameter, the random component is either determined 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. For example, Gamma distributions have two parameters, but these do not pull location and dispersion neatly apart, as Normal distributions do. Instead, mean and variance Gamma distributions depend on both parameters.

Using distributions with entanglement of location and dispersion seems to be a step back, but frequently is necessary to render a realistic association between the expected value 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.

7.3 Case: user testing infusion pumps

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 stuff, anaesthesiologists and nurses, mostly. 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
- flimpsy 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 exetensive study of user requirements and design guidelines. As even the newest national 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. The participants were nurses and they were asked to complete a set of eight standard tasks with the devices. In order to capture learnability of the devices, every nurse completed the sequence of tasks in three consecutive sessions. 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.

As can be expected in the light of what has been said above, each one these measures violate one or more assumptions of the Gaussian linear model. In

the following chapters, proper models from teh GLM family are introduced for commonly occuring measures of types:

1. *count data*, such as the number of completed tasks and path deviations
2. *temporal data*, such as time-on-task
3. *rating scales*

7.4 Count data

Normal distributions assume that the random variable under investigation is continuous. For measures, such as time, that 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 larger 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, at least (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, logistic regression applies.

7.4.1 Poisson regression

When data can be considered successes in a fixed number of trials, logistic regression is the model type of choice. When the outcome variable is a count, but there is no apparent upper limit, Poisson regression applies.

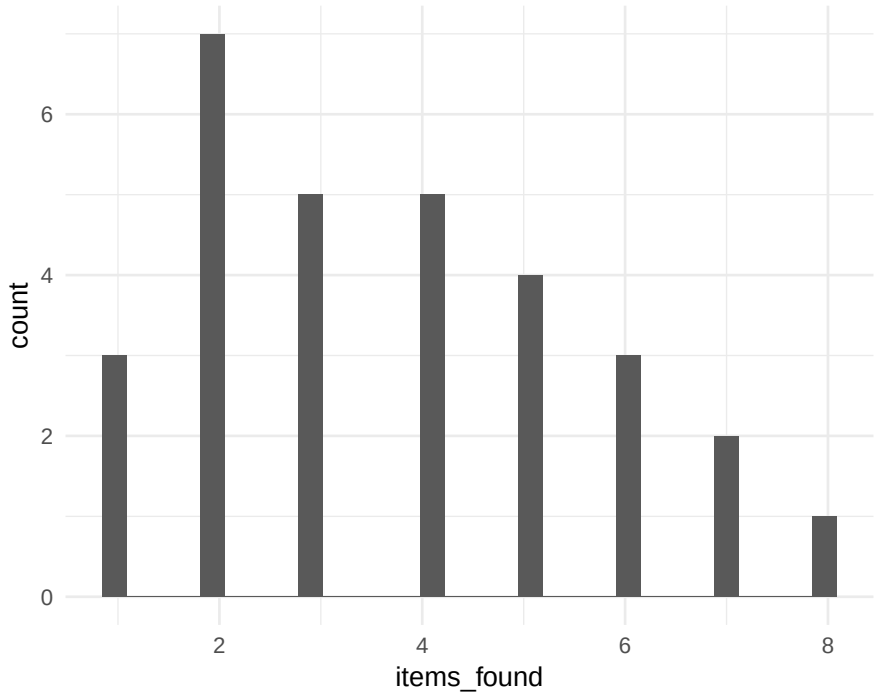
In brief, Poisson regression has the following attributes:

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 the start, we have a look at a Poisson GMM: In an advanced level of the smart smurfer game, the items are hidden from the player and therefore extremely difficult to catch. To compensate for

the increased difficulty somewhat, the level carries an abundance of items. On average, the player is not supposed to find more than three items. We simulate a data set for one player repeating the level 30 times and run a Poisson regression model:

```
set.seed(6)
D_Pois <-
  data_frame(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, iter = iter)
```

```
fixef(M_Pois)
```

model	type	fixef	center	lower	upper
object	fixef	Intercept	1.31	1.16	1.48

```
# bayr:::knit_print.tbl_coef(M_Pois)
```

The Poisson parameter λ (lambda) has a direct interpretation as it represents the expected mean (and variance) of the distribution. Instead, the regression coefficient is on a logarithmic scale, to ensure it has no boundaries. To scale it down to the scale of measurement, the exponential function is the canonical mean function in Poisson regression:

```
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.

7.4.1.1 Speaking multiplicative

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 normative path were counted to serve as a measure for safety of operation. 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.

```
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, iter = iter)

M_dev
```

```
fixef(M_dev)
```

fixef	center	lower	upper
Intercept	0.838	0.311	1.362
DesignNovel	-1.511	-2.247	-0.780
session	-0.233	-0.350	-0.139
DesignNovel:session	-0.081	-0.242	0.113

Again, the coefficients are on a logarithmic scale and cannot be interpreted right away. By using the exponential mean function, we obtain the following table:

```
fixef(M_dev, mean.func = exp)
```

fixef	center	lower	upper
Intercept	2.312	1.365	3.904
DesignNovel	0.221	0.106	0.459
session	0.792	0.705	0.870
DesignNovel:session	0.922	0.785	1.120

The intercept now has the interpretation as the expected number of deviations with the legacy design in the first session. However, it is incorrect to speak of the effects in terms of differences. i.e. summative. Fortunately, the following arithmetic law tells that what is summative on the level of the linear predictor, becomes *multiplicative* on the original scale:

$$\exp(\beta_0 + x_1\beta_1 + x_2\beta_2) = \exp(\beta_0)\exp(x_1\beta_1)\exp(x_2\beta_2)$$

Hence, the exponentiated coefficients have a *multiplicative interpretation*, like the following:

1. In the first session, the novel design produces 2.312 *times* the deviations than with the legacy design.
2. For the legacy design, every new training session reduces the number of deviations *by factor* 0.792

3. The reduction rate per training session of the novel design is *92.229% as compared to the legacy design.

With counts we usually expect the variance of randomness to rise with the mean. The Poisson distribution is very strict in the sense that the variance equals the mean

$$y_i \sim \text{Pois}(\lambda) \rightarrow \text{Var}(x_i) = \text{Mean}(x_i) = \lambda$$

.

Real count data frequently has variance that raises proportionally with the mean, but is inflated. This is called *overdispersion* and is accounted for by an observation-level random effect, which will be explained in a separate section **??**.

Hence, the novel design reduces the initial number of deviations by factor

```
detach(IPump)
```

7.4.1.2 Monotony and quantiles

The transformation of coefficients to the original scale has been applied to the point and range estimates as produced by the `fixef` command, that is *after* summarizing the posterior distribution. One may wonder if this is valid. Would we get the same estimates when applying the mean function to all draws of the MCMC chain and then summarize? The general answer is that applying the mean function after summarizing is allowed if the summary function is invariant under the exponential function.

For all GLM, the link and mean functions are monotonically increasing, with the consequence that the order of observations is preserved. Formally, for any two MCMC iterations i and j for a parameter β_i:

$$\beta_{1i} < \beta_{1j} \rightarrow \exp(\beta_{1i}) < \exp(\beta_{1j})$$

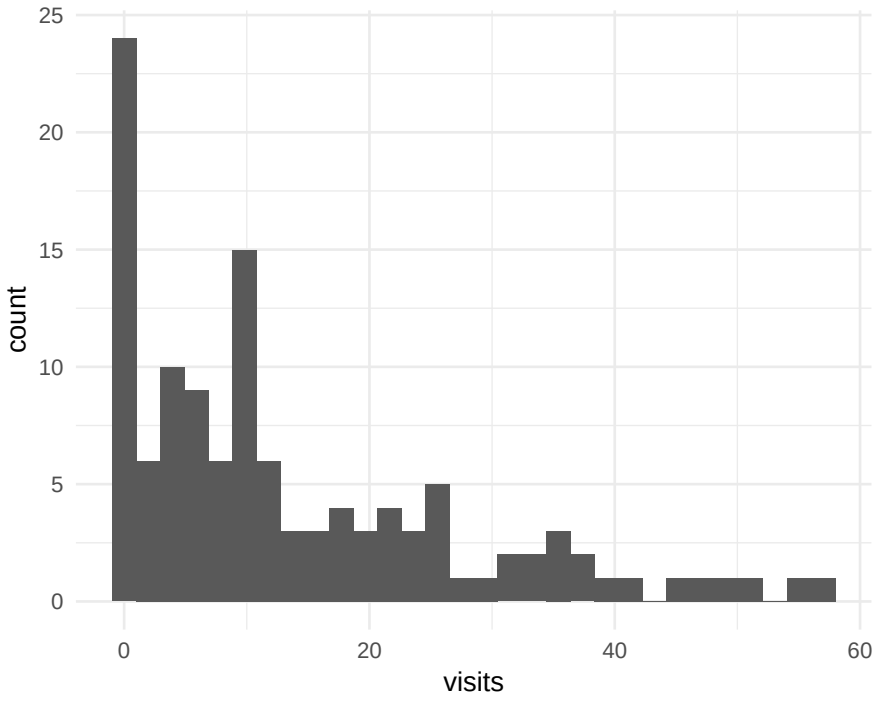
Recall that throughout this book, center and interval estimates have been obtained by simple quantiles, marking the points where 2.5%, 50% and 97.5% of all iterations are smaller. Order does not change with monotonous transformations, if 2.5% (50%, 97.5%) of draws are smaller on the linear scale, they will still be after applying the mean function. Quantiles are not affected by monotonous transformation and transformation after summary is therefore valid. Some researchers prefer the mode of the posterior to represent its center location. The mode is the point of highest density and does not rely on ranks, it therefore even invariant under all transformations that preserve identity.

This is different for higher order methods for obtaining point and interval estimates. Most notably the mean and the highest posterior density intervals are not invariant to mean functions. When using those, the mean function must be applied before summarizing the posterior, which is inconvenient and inconsistent.

7.4.1.3 Zero inflation

Imagine a study that examines the frequency of visits to the social media website Fakebook. While other researchers already set out to find predictive factors, like extrovert personality and the like, we are here interested in the frequency of daily use.

```
sim_zi <- function(
  beta_0  = 2.5,
  N_Part  = 120,
  p_zero  = .2, # proportion of non-users
  sd_Part = 0.8, # individual differences (lp scale)
  seed    = 23 # parameters passed on to simulate_1
){
  set.seed(seed)
  data_frame(Part = 1:N_Part,
             theta = rnorm(N_Part, beta_0, sd_Part),
             mu = exp(theta),
             is_user = rbinom(N_Part, 1, 1 - p_zero),
             visits = 0 + is_user *rpois(N_Part, mu))
}

D_zi <- sim_zi()
D_zi %>%
  ggplot(aes(x = visits)) +
  geom_histogram()
```




```
## [1] "sim_zi" "D_zi"
```

7.4.2 Logistic (aka Binomial) regression (#logistic_regression)

When the outcome variable can be conceived as successes in a fixed number of trials, logistic regression applies. In brief, logistic regression has the following attributes:

1. The outcome variable is bounded at zero and the number of trials k
2. The linear predictors is on a *logistic scale*, with the *logit* being the inverse. However, in 7.4.2.1 we will see, that using the logarithm results in a convenient way of speaking about the effects.
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.

The most simple form of successes-in-trials measure is when there is only one trial. This is called *dichtotomous*, with the following common examples:

- a user is successful at a task, or fails
- a visitor returnsa to a website or doesn't
- a usability problem is discovered or remains unseen
- a driver brakes just in time or crashes
- a customer would recommend a product to a friend or rather not
- a web user starts a search for information by keyword query or by following links

Most dichotomous outcome variables have a more or less clear notion of success and failure (although not necessarily as in the last example). When the outcome casts a positive light on the design, by convention it is coded as 1, otherwise 0.

In computer science jargon, every dichotomous observation accounts to a *bit*, which is the smallest amount of information ever possible. Since in inferential statistics the amount of information is tantamount with the reduction in uncertainty, with dichotomous data one usually needs an abundance of observations to reach reasonably certain conlusions. Because the information of a single observation is so sparse, large samples and repeated measures are important when dealing with dichtomous outcomes.

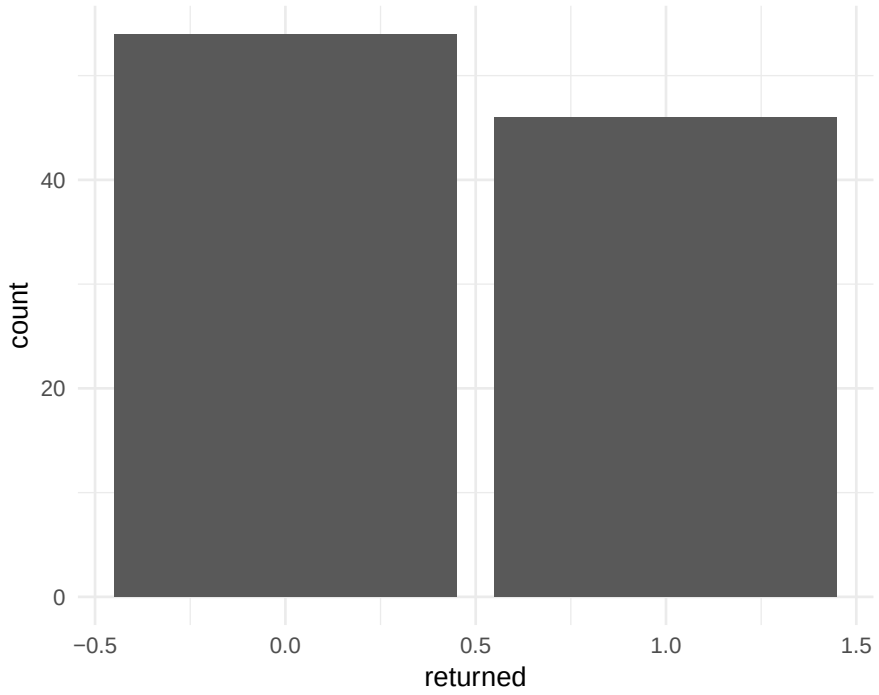
Let us see an example: early research on foraging strategies of web users revealed that they are extremely impatient companions. They scan a page for visual features, rather than reading [REF: high school students information mall]. Visitor of websites build their first judgement in a time as short as 17ms [REF: Tuch presentation time]. For e-commerce that is a highly important fact to know about their customers and practically all commercial websites shine with a pleasing visual appearance, nowadays. But, how would one measure the gratitude of a visitor who actually used the website and may have something to tell beyond visual pleasance.

A simple measure for gratitude is whether a visitor returns. This usually is a highly available measure, too, as any skilled web administrators can distill such data from the server logfiles with little effort. First, all unique visitors are extracted and if the same visitor returns within a given period of time, this is coded as a success (one) otherwise failure (zero). We simulate such a data set:

```
set.seed(42)
D_ret <- data_frame(visitor = as.factor(1:100),
                    returned = rbinom(100, 1, .4))

D_ret %>% sample_n(6) %>% kable()
```

visitor	returned
63	1
22	0
99	1
38	0
91	1
92	0

```
D_ret %>%
  ggplot(aes(x = returned)) +
  geom_bar()
```



In total, 46% visitors return. In order to estimate the return rate together with a statement on uncertainty, we run a logistic regression grand mean model and inspect the coefficient table. Note how the linear formula is completely common ground, but we explicitly pass the binomial `family` to the regression engine.

```
M_ret <- D_ret %>% stan_glm(returned ~ 1, data = .,
                            family = binomial, iter = iter) # <--
```

```
fixef(M_ret)
```

model	type	fixef	center	lower	upper
object	fixef	Intercept	-0.148	-0.542	0.211

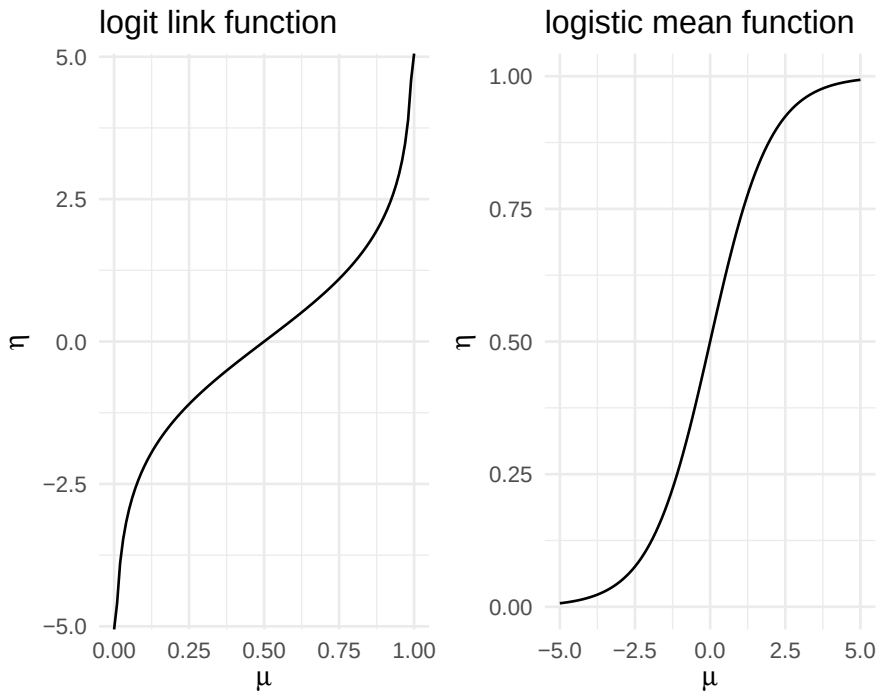
As expected from a GMM we retrieve one parameter that here reflects the average tendency to return to the site. Recall, that the linear model assumes the predictions to be unbounded. However, in case of return rates, we rather speak of *proportions* of users to return. As a side note, I would rather avoid speaking of probabilities in the context of logistic regression. While being mathematically correct, it sometimes causes confusion with certainty or, beware of this, the p-value.

Proportions are on a range from zero to one. Because the quantity of interest is bounded, a link function is needed that stretches the bounded into an unbounded range. For logistic regression, the *logit* functions maps the expected values $\mu_i \in [0; 1]$ onto the *linear predictor* scale $\eta_i \in [-\infty; \infty]$:

$$\eta_i = \text{logit}(\mu_i)$$

The inverse function, commonly called the *mean function*, of the logit is the *logistic function*. @ref(logit_logist) shows link and mean functions side-by-side.

```
grid.arrange(
  ggplot(data.frame(mu=c(0, 1)), aes(x = mu)) +
    stat_function(fun = mascutils::logit) +
    xlab(expression(mu)) + ylab(expression(eta)) +
    ggtitle("logit link function"),
  ggplot(data.frame(eta=c(-5, 5)), aes(x = eta)) +
    stat_function(fun = mascutils::inv_logit) +
    xlab(expression(mu)) + ylab(expression(eta)) +
    ggtitle("logistic mean function"),
  nrow = 1)
```



In order to obtain a statement on proportion μ (note that in a GMM, there is only one), we therefore have to perform the mean transformation:

$$\eta = \beta_0 \mu = \text{logist}(\eta)$$

The `fixef` command lets you pass on a mean function. However, ths logistic mean function only is admissable for intercepts and other absolute group means, as we will see in 7.4.2.1

```
fixef(M_ret, mean.func = inv_logit)
```

model type	fixef		center	lower	upper
object	fixef	Intercept	0.463	0.368	0.553

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. That is precisely because of the low level of information contained in dichotomous variables (one bit). For a reasonably certain estimate one would need

many more observations. These can either be obtained by a larger sample or by repeated measures.

Recall the fictional jump-and-run game *smart smurfer* in @ref(poisson_dist): the goal of the game is that players collect items and for the user experience it is crucial that this is neither too difficult nor too easy. Imagine, that for adjusting the difficulty level, the developers conduct a quick evaluation study, where they place a number of items (trials) in the game and the success rate of a single player is observed in a series of 15 game sessions:

```
D_smrf <-
  data_frame(
    Session = 1:15,
    trials = round(runif(15, 0, 25), 0),
    successes = rbinom(15, trials, .4),
    failures = trials - successes) %>%
  mascutils::as_tbl_obs()

D_smrf
```

Obs	Session	trials	successes	failures
1	1	18	7	11
2	2	18	8	10
7	7	19	11	8
9	9	13	4	9
11	11	0	0	0
12	12	9	6	3
13	13	15	6	9
14	14	21	9	12

Per session the player has a number of opportunities for collecting an item, which is a repeated measures situation. One might expect that we need to include random effects into the model. Later, we will see, that this is necessary when the sessions were observed on a sample of players with different abilities. However, as long as one can reasonably assume the chance of catching an item to be constant across all sessions, plain logistic regression can deal with *successes-in-multiple-trials*. In order to estimate a model with more than one trial per observation, it is necessary to add a variable for the number of failures and use a `cbind(successes, failures)` statement for the left-hand-side of the model formula. This may seem inconvenient, but it allows to have a different number of trials per observation.

```
M_smrf <- stan_glm(cbind(successes, failures) ~ 1,# <--
                   family = binomial,
                   data = D_smrf, iter = iter)
```

```
fixef(M_smrf, mean.func = inv_logit)
```

model type	fixef	center	lower	upper
object	fixef	Intercept	0.485	0.418	0.559

```
## [1] "D_smrf" "D_ret"
```

We turn now to a real case study, the comparison of two medical infusion pumps (@ref(slope_RE)). On both devices (legacy and novel), 25 nurses completed a set of eight tasks repeatedly over three session. 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 observation 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 take a compromise here: estimate the full model on group 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 = ., iter = iter)

sync_CE(IPump, M_cmpl)
```

```
T_cmpl <-
  fixef(M_cmpl)
T_cmpl
```

fixef	center	lower	upper
Intercept	1.367	0.242	2.497
DesignNovel	0.398	0.116	0.710
Session2-1	0.694	0.155	1.223
Session3-2	-0.062	-0.602	0.492
DesignNovel:Session2-1	-0.281	-1.066	0.520
DesignNovel:Session3-2	0.272	-0.543	1.006

Keep in mind that the estimates are on the scale of the linear predictor η_i. This is a boundless space, where we can freely create linear combinations of effects to obtain group means. To get a group mean prediction on the more meaningful measurement scale $\mu \in [0; 1]$, one must *first* do the linear combination, *followed* by the mean function.

- the completion rate in the first legacy session is 0.797
- in novel/session 1: `logist(Intercept + DesignNovel)` = 0.854
- in novel/session 2: `logist(Intercept + DesignNovel + Session2-1 + DesignNovel:Session2-1)` = 0.898
- in legacy/session 3: `logist(Intercept + DesignNovel + Session2-1)` = 0.881

7.4.2.1 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. The linear predictor scale has only very general intuition:

- zero marks a 50% chance
- positive values increase the chance, negative decrease
- bigger effects have larger absolute values

That ís 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, for example:

1. What is the expected frequency of failure on first use?
2. The novel design reduces failures, but sufficiently?

3. Is frequency of failures reduced to an acceptable level by two training sessions?

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)` $= 0.598$".

However, there is another transformation, that does the trick. For a better understanding, we have to first inspect, what the logit actually is. The logit is also called a *log-odds*: $\text{logit}(p) := \log(p(1-p))$. The inner part of the function, the *odds*, are the chance of success divided by the chance of failure. Odds are a rather common way to express ones chances in a game, say:

- odds are 1 against one 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`:

```
T_fixef_cmpl_odds <- fixef(M_cmpl, mean.func = exp)
T_fixef_cmpl_odds
```

fixef	center	lower	upper
Intercept	3.922	1.274	12.14
DesignNovel	1.489	1.123	2.03
Session2-1	2.001	1.167	3.40
Session3-2	0.940	0.548	1.64
DesignNovel:Session2-1	0.755	0.344	1.68
DesignNovel:Session3-2	1.312	0.581	2.73

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 €392.214 on success.
- The odds of success with the novel design increase by *factor* 1.489. Now, I would place $392.214 \times 1.489 =$ €583.99 on success.
- On success with the novel design in session 2, I would place $392.214 \times 1.489 \times 2.001 \times 0.755 =$ €882.124 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)
```

7.4.3 Modelling overdispersion

If you are in a hurry: real data is always overdispersed. If your data is real, prefer the negative binomial family over Poisson and beta-binomial over logistic regression. Or do observation-level random effects.

Poisson and binomial distributions are one-parameter distributions. As there is only one parameter, it is impossible to set location and dispersion separately. In effect, both properties are tightly entangled. For Poisson distributions they are even the same.

$x \sim Pois(\lambda) \implies \mu = \sigma^2 = \lambda$

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.

The strict variance assumptions of Poisson and binomial models are frequently violated by real data. The violation happens when impact factors have not been included in the likelihood equation. Whenever there is one or more impact factors on the outcome at question that the researcher has not regarded, overdispersion inevitably happens. That means it practically always happens in studies involving objects with complex dynamics, such as the human mind.

Two solutions exist for overdispersed count data: we can either switch to an appropriate two parameter response distribution, that disentangles mean and variance, or introduce an observation-level random effect.

7.4.3.1 Negative-binomial regression for overdispersed counts

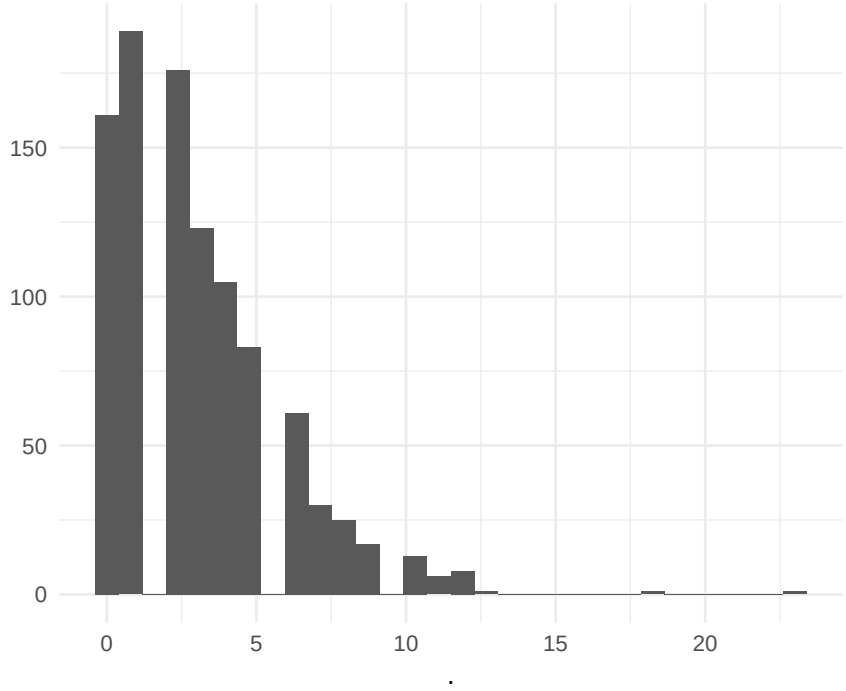
For all three one-parameter distributions, there exists a two-parameter distribution that allows to estimate the amount of random dispersion (almost) independently of the mean.

```
readxl::read_excel("Illustrations/GLM_distributions.xlsx", sheet = "plugin")
```

canonical	generalization	parameters	limiting case
binomial	betabinomial	$\alpha > 0, \beta > 0$	$\alpha, \beta \rightarrow \infty$
Poisson	negative binomial	NA	NA
exponential	gamma	rate, shape	NA

Using Poisson regression for overdispersed count data results in too favorable certainty of estimates, which is a mistake, of course. One solution to the problem is to switch to yet another response distribution that has a second parameter, thereby allowing the variance to vary (almost) freely. For the Poisson case (i.e., counts without an upper limit) that typically is the negative binomial distribution (never mind its name!), for binomial the beta-binomial applies. Both distributions are so-called *mixture distributions*. In mixture distributions, the parameter of the original distribution is not constant, but allowed to vary, typically according to some other distribution. Under this perspective, negative binomial distribution is equivalent to a Poisson distribution, when we let parameter λ follows a certain gamma distribution:

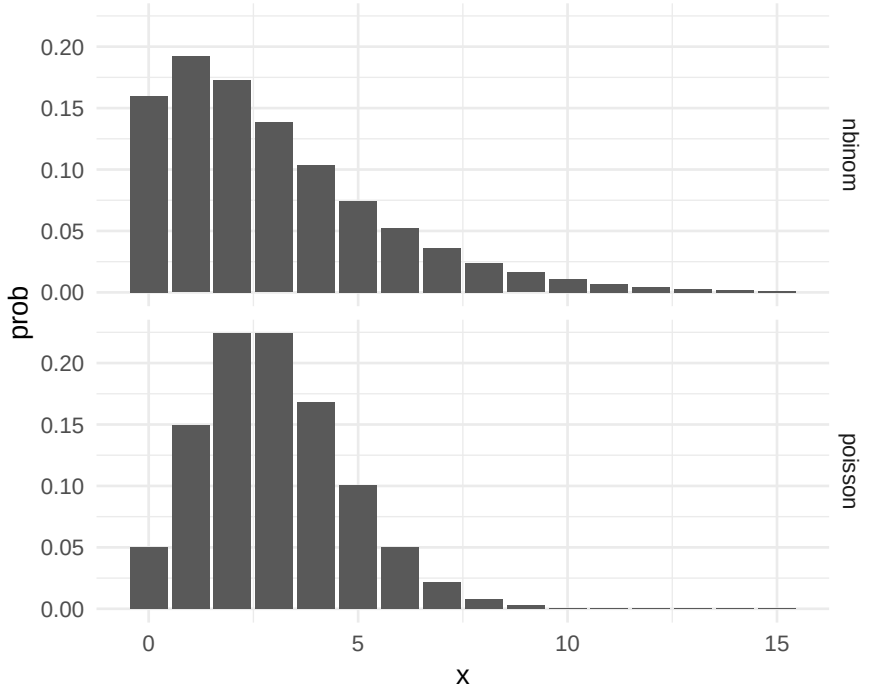
```
rnegbinom <- function(n, mu, size){
  shape = size
  scale = mu/size
  lambda = rgamma(n, shape = shape, scale = scale)
  rpois(n, lambda = lambda)
}

rnegbinom(1000, mu = 3, size = 2) %>%  qplot()
```



.

The figure below shows a negative binomial distribution and Poisson distribution with the same mean.

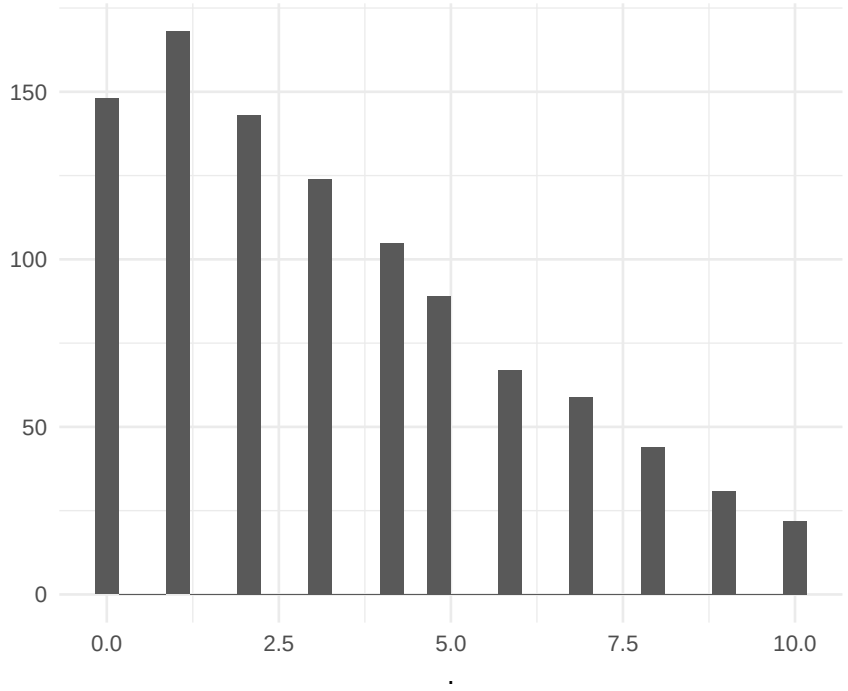
```
data_frame(x = 0:15,
           nbinom = dnbinom(x, mu = 3, size = 2),
           poisson  = dpois(x, 3)) %>%
  gather(distribution, prob, -x) %>%
  ggplot(aes(x = x, y = prob)) +
  facet_grid(distribution ~ .) +
  geom_col(position = "dodge")
```



7.4.3.2 Beta-binomial regression for successes in trials

For beta-binomial distributions, the binomial parameter p is mixed from a beta distribution, with parameters a and b:

```
rbetabinom <- function(n, size, a, b) rbinom(n, size, rbeta(n, a, b))
rbetabinom(1000, 10, 1, 2) %>% qplot()
```



Predictions and interpretation of coefficients of negative-binomial and beta-binomial models are just as with their counterparts, using the same link functions. There is just a tiny difference: a single parameter has been added to the model, which modifies the dispersion. In a standard analysis these parameters have very little meaning, even less than the standard error in a Gaussian model. No misunderstanding: these parameters are *not* the constant standard deviation of residuals. They act as scalers for the "natural" dispersion at any point.

The brms regression engine currently only implements the negative binomial, but not the beta-binomial family. That is a minor problem, because the brms regression engine can be extended. The following code is directly taken from the brms documentation and provides a betabinomial model. The canonical parametrization of the betabinomial distribution takes the parameters a and b from the inner beta distribution. Note, how the author of this code transforms the parameters to obtain an expected value μ and a dispersion parameter ϕ. Like in logistic regression, coefficients are on a logit scale.

```
# 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);
  }
"

D_betabin <- data_frame(y = rbetabinom(100, 10, 4,2), n = 10)

M_betabin <- D_betabin %>%
  brm(y | trials(n) ~ 1, family = beta_binomial2,
      stan_funs = bb_stan_funs, data = .)
```

```
coef(M_betabin)
```

parameter	type	fixef	center	lower	upper
b_Intercept	fixef	Intercept	0.876	0.676	1.08
phi	shape	NA	6.283	4.054	10.28

7.4.3.3 Using observation-level random effects

With the broad implementation of random effects models in Bayesian regression engines, there is a generic procedure to capture the extra variance even with one parameter distributions. The trick is to introduce an *observation-level random effect*. Recall how we regard variation between members of a population as normally distributed deviations from the population mean, by the example of a Poisson grand mean model with a participant-level (p) random effect:

$$\theta_{pi} = \beta_0 + x_p\beta_{0p}\mu_{pi} = \exp(\theta_{pi})\beta_{0p} \sim N(\mu_p, \sigma_p)y_p \sim \text{Pois}(\mu_{ij})$$

The OLRE is normally distributed but does not cause any bounded-range range, as it is added on the level of the linear predictor, before applying the exponential transformation. Observation-level random effects are completely

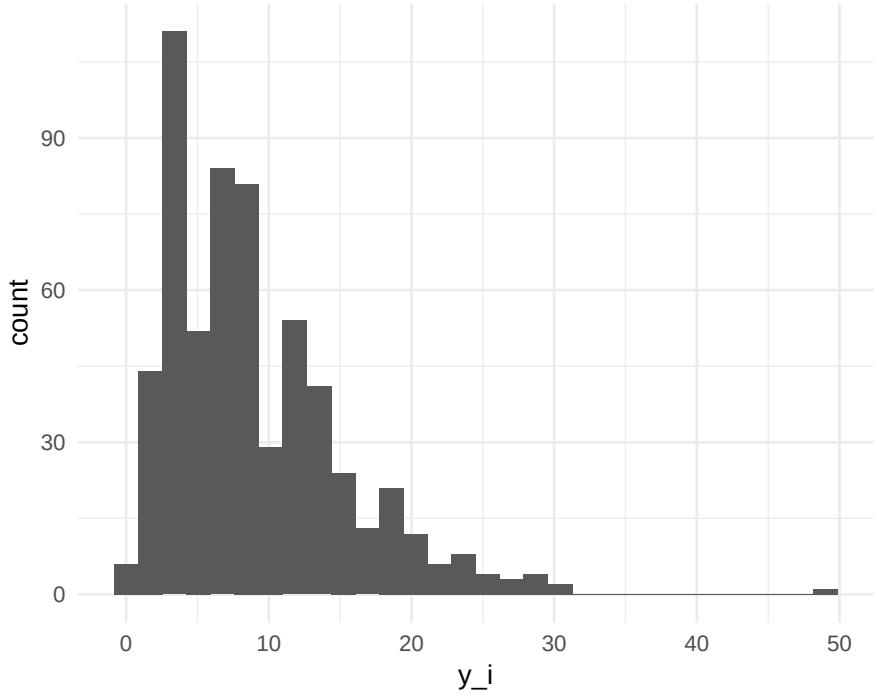
analogous, except that every observation becomes its own group, in a Poisson grand mean model with added variation:

$$\theta_i = \beta_0 + \beta_i \mu_i = \exp(\theta_i) y_{ij} \sim \text{Pois}(\mu_{ij})$$

See, how β_i is a unique deviation per observation i, and how a variance parameter σ appears in an otherwise purely Poisson model. Observation-level random effects are on the linear predictor level, and therefore additive. Compare this to the negative binomial distribution where variance is scaled up, which is multiplication. We find a resemblance with how sums on the linear predictor become multiplications on the expected values scale.

For demonstration of the concept, we simulate from an overdispersed Poisson grand mean model with participant-level random effects, and recover it via regression.

```
sim_ovdsp <- function(
  beta_0 = 2,    # mu = 8
  sd_Obs = .3,
  sd_Part   = .4,
  N_Part = 30,
  N_Rep  = 20,
  N_Obs =  N_Part * N_Rep,
  seed = 42){
  set.seed(seed)
  Part <- data_frame(Part = 1:N_Part,
                     beta_0p = rnorm(N_Part, 0, sd_Part)) ## participant-level RE
  D <- data_frame(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()
```




```
## [1] "sim_ovdsp" "D_ovdsp"
```

```
M_ovdsp <-
  D_ovdsp %>%
  stan_glmer(y_i ~ 1 + (1|Part) + (1|Obs), data = .,
             family = poisson, iter = iter)
```

Random effect variation is accurately recovered from the simulated data. The following two plots show, the participant latent scores can be recovered and even the observation levels themselves. Every observation gets an accurate measure of how much it had been pushed by unrecorded sources of variation. Practically, we obtain residuals which can be used for model criticism. For example, extreme outliers can be identified and relations between random variation and predicted values (or groups) are open to scrutiny.

```
grpef(M_ovdsp)
```

re_factor	center	lower	upper
Obs	0.305	0.264	0.350
Part	0.536	0.403	0.719

Frequently, I reminded the reader to interpret parameters quantitatively, by translating their magnitude to statements of practical relevance. For random effects variance this isn't always straight forward. One possible way is to make comparative statements on the sources of variance *"the variance due to individual differences exceeds all other sources of variation taken together"*. OLREs are on the same scale as all other random effects in the model, which makes it a good default reference. A non-default comparison of sources of variance is the one of Dennis Egan, that people cause more variance than designs do. With GLMM, the marriage of LMM and GLM this claim is testable.

From LMM we borrow a surprisingly simple and general solution, observation-level random effects. So, most of the time we won't need one of those twisted two parameter random distribution to account for overdispersion. With OLRE models we get an estimate that is very similar to a *residuals*, which have proven very useful in model criticism.

```
## [1] "sim_ovdsp" "D_ovdsp"
```

7.4.4 Exercises:

7.5 Measures of time

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 one at Physikalisch-Technischen Bundesanstalt in Braunschweig (Germany). Temporal variables often carry useful information. *Reaction time (RT)* measures are prime in experimental cognitive studies and have revealed fascinating phenomena of the human mind, such as the Stroop effect and priming. In user studies and design research *time-on-task (ToT)* is a common measure for efficiency, with quicker being the better.

Temporal variables are practically continuous (as long as one measures with sufficient precision), but have lower bounds. A large number of statistical distributions are routinely used to model temporal variables. However, those do not always apply well to ToT or RT data, as they assume a lower bound of zero, which is unrealistic. Still, we begin with two of such models, exponential and gamma regression, and present a candidate for ToT data with offset as third (exgaussian regression).

7.5.1 Exponential regression

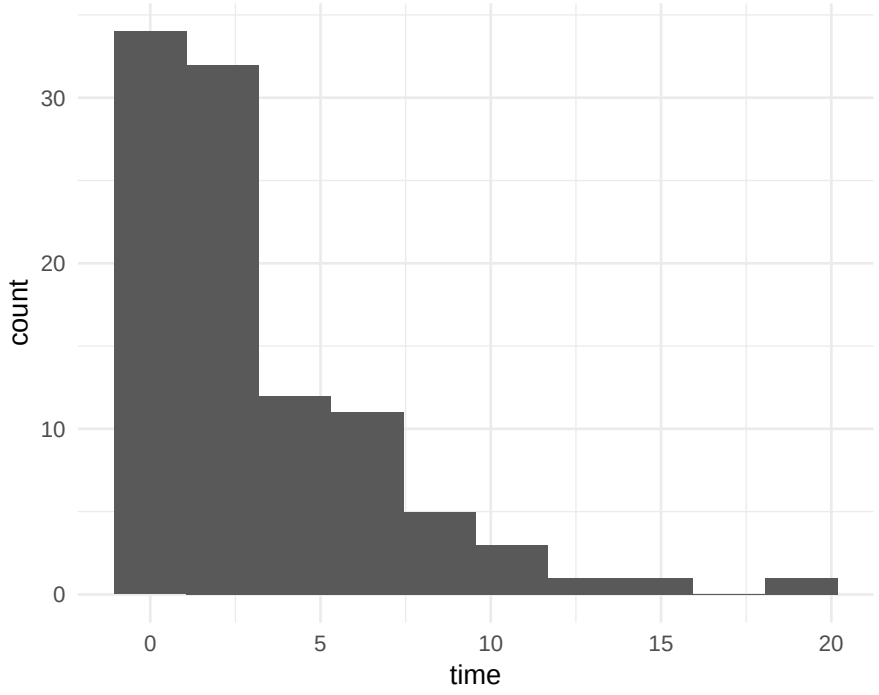
Exponentially distributions arise from random processes under some very idealized conditions. First, the lower boundary must be zero and second, the

waiting time of any event in a series is completely independent, i.e. not predictable, from how long one has been waiting so far. This is also called the property of *being memoryless*. In physics memoryless processes are observed only in the simplest systems, such as the time between two nuclei of a radioactive isotope to decay. Examples of memoryful processes are plenty, such as:

- earthquakes are essentially relaxations in the earth crust. Once it happened, it is less likely to reoccur in near future.
- participants learn with every trial or task, making shorter times more likely

Reconsider the subway smurfer example, 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 inverse $\mu = 1/\lambda$ and the variance is $\text{Var} = 1/\lambda^2$

```
set.seed(20)
D_exp <-
  data_frame(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)
```

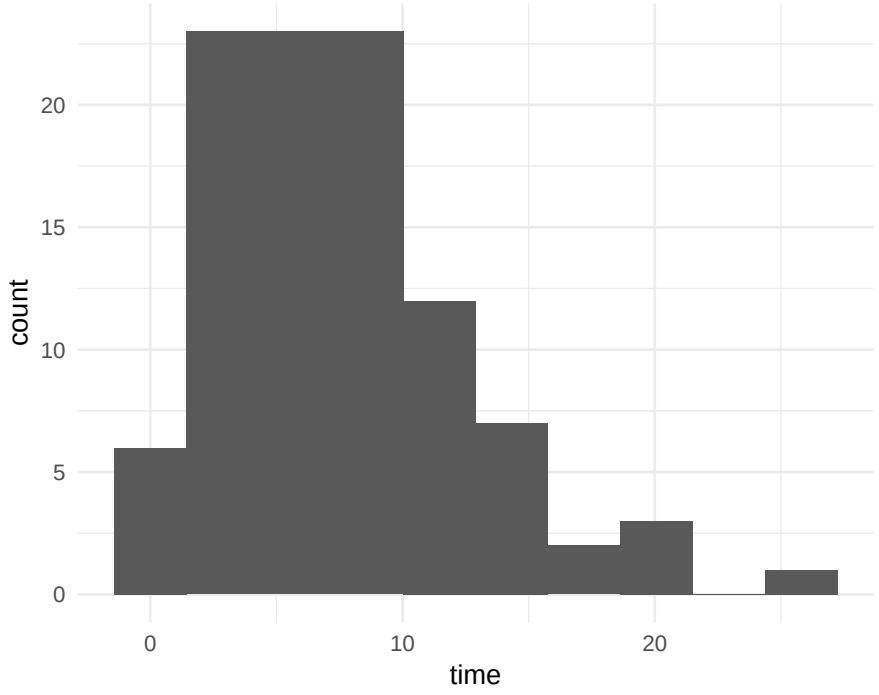
```
fixef(M_exp, mean.func = exp)
```

type	fixef	center	lower	upper
fixef	Intercept	3.29	2.73	4

7.5.2 Gamma regression

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 probability to occur, will an exponential distribution arise, which for behavioural research means: never. Again, one could solve this by introducing an observation-level random effect. Instead, here we will tackle the problem by using a continuous, zero-bounded distribution 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 variances rises with the mean, but as we have argued in 7.1.3, this is rather a desired feature than a problem. In the following, we simulate gamma distributed observations.

```
set.seed(20)
D_gam <-
  data_frame(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 resort to the common log link.

```
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, the exponential and the 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. For the exponential distribution that even goes to the extreme, that zero always is the point of highest density, i.e. the most likely outcome. As we will see in the following section, that assumption is usually violated with RT and ToT data, and exponential and gamma regression should be avoided. 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 emit a neutron. If you measure the interval between two emissions the time interval is exponentially distributed. (And if you count the neutrons per time interval, the result is a Poisson distribution). An analogue situation arises for customer support systems. Customers are like atoms in that their decision to file a request is independent from each other, usually. By chance it can happen that the whole center is idle for 20 minutes, but it is equally possible that 20 customers call within a minute and some of them practically at the same moment. Overwhelmed hotline queues does not make people nhappy, when 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. Whatever predictors such a model would have, the response distribution would probably be very much a gamma distribution. In contrast, when the events are the moments of completion in successive tasks performed by one user, the triggers are not independent and the measure cannot be arbitrarily small (unless, perhaps, you kill seven flies with a single strike).

7.5.3 ExGaussian regression

The problem with the two temporal models discussed so far is that they assume a RT or ToT of zero to be possible. In the case of exponential regression, this even is the most likely region. This assumption is not realistic in most cases, as any task uses up a minimum time to complete. For example, the table below shows the minimum reaction times for finding the academic calendar on ten university websites (case Egan). This varies a lot between designs, but is never 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)
```

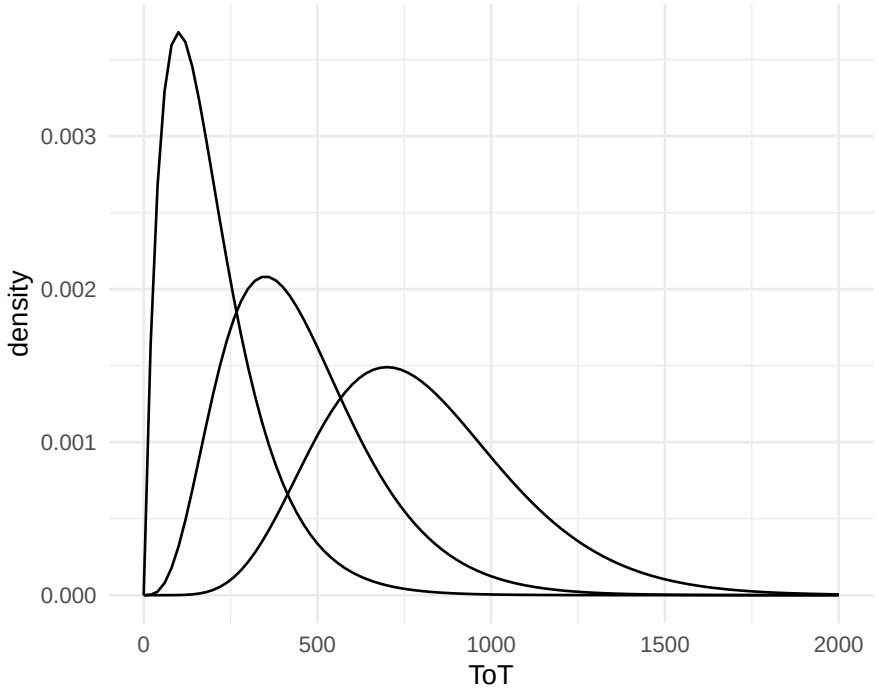
```
D_egan %>%
  filter(success,
         Task == "academic calendar") %>%
  group_by(Task, Design) %>%
  summarize(min_time = min(ToT),
            range = max(ToT) - min_time,
            min_time/range) %>%
  kable()
```

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

On the first glance, that does not seem to pose a major problem for gamma regression, as the mode can move along the x-axis. However, when a theoretical gamma distribution moves to the right, it inevitably becomes more symmetric, i.e. the skewness is reduced (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 that with a bunch of gamma distributions that move from left to right (`M`), keeping the variance constant (`V`):

```
M = c(200, 300, 400)
V = 20000

## gamma
rate = M/V
shape = rate^2 * V

ggplot(data.frame(x = c(0, 2000)), aes(x = x)) +
```

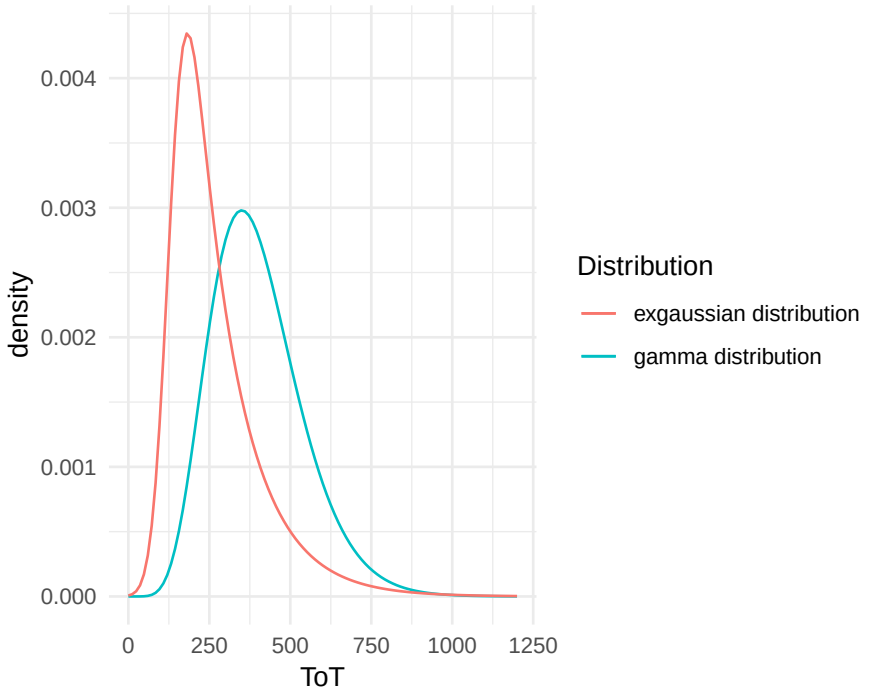
```
  stat_function(fun = dgamma,
                args = list(rate = rate[1], shape = shape[1])) +
  stat_function(fun = dgamma,
                args = list(rate = rate[1], shape = shape[2])) +
  stat_function(fun = dgamma,
                args = list(rate = rate[1], shape = shape[3])) +
  labs(x = "ToT", 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 (Gamma, Gaussian). It seems logical that only three-parameter distributions can do the trick of setting skewness separately. The exponentially modified Gaussian (exgaussian) distribution is a convolution of a normal distribution and an exponential distribution and has three parameters, μ, σ and rate β. Very roughly, the Gaussian component controls location and dispersion whereas the exponential part introduces the skewness. When β is large in comparison to μ, the distribution is more left skewed. By this additional degree of freedom it becomes possible to simulate (and estimate) distributions that are far to the right have strong dispersion *and* strong skewness. The fol-

lowing plot shows the density of an exgaussian and gamma distribution with the same mean and variance.

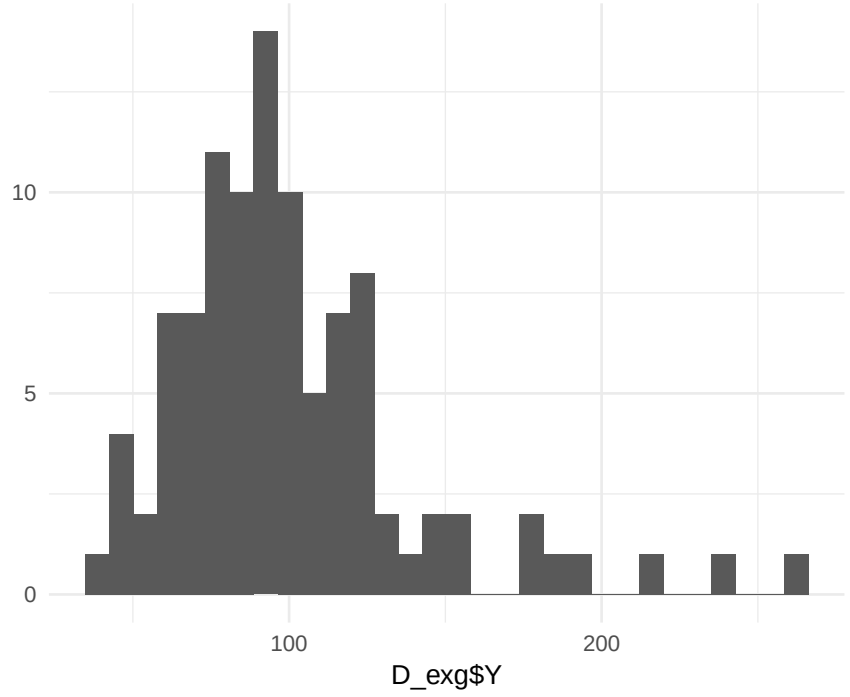
```
M = 400
V = 20000

## exgauss
beta = 135
mu = M - beta
sigma = sqrt(V - beta^2)

## gamma
rate = M/V
shape = rate^2 * V

ggplot(data.frame(x = c(0, 1200)), aes(x = x)) +
  stat_function(fun = dgamma,
                args = list(rate = rate, shape = shape),
                mapping = aes(colour = "gamma distribution")) +
  stat_function(fun = brms::dexgaussian,
                args = list(mu = mu,
                            sigma = sigma,
                            beta = beta),
                mapping = aes(colour = "exgaussian distribution")) +
  labs(colour="Distribution", x = "ToT", y = "density")
```



The gamma distribution in this example starts approaching a bell curved form, almost resembling a 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 fabulous `brm` engine:

```
attach(Chapter_GLM)
```

```
D_exg <- data_frame(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

Noteworthy, 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.

7.5.3.1 Modelling reaction times

In experimental studies, the *inertia of the nervous system* sets a limit larger than zero for reaction times. This is due to some hard electrochemical and biomechanical facts of the peripheral systems. Nerve cells and muscle fibers are slow working horses. The same goes for our minds. Arguably, they are blazingly fast at complex tasks, such as recognizing written words in the faintest colors and the bizarrest advertisements. Still, there is a minimum time to retrieve an idea from the memories and activate the surrounding nodes, too.

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 don't 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 geek. The hypothetical geek personality feels attracted to the inner workings of a computer system itself, rather than the applications and media 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, than 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. As a response we had choden the Stroop task, and in order to make this more likely, participants were primed by a picture shown before the Stroop task. These pictures were from two conditions: either showing computers in a geekish way (for example, an open case or programming code on the screen) or as regular product images. Furthermore, we theorized that the least one can say is that geeks like to think hard and therefore used the need-for-cognition scale as a 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.

What would be a suitable response distribution for reaction times in the semantic Stroop task? In the following, we run three models with the same predictor term, but with exgaussian, gamma or Gaussian random components.

```
attach(Hugme)
```

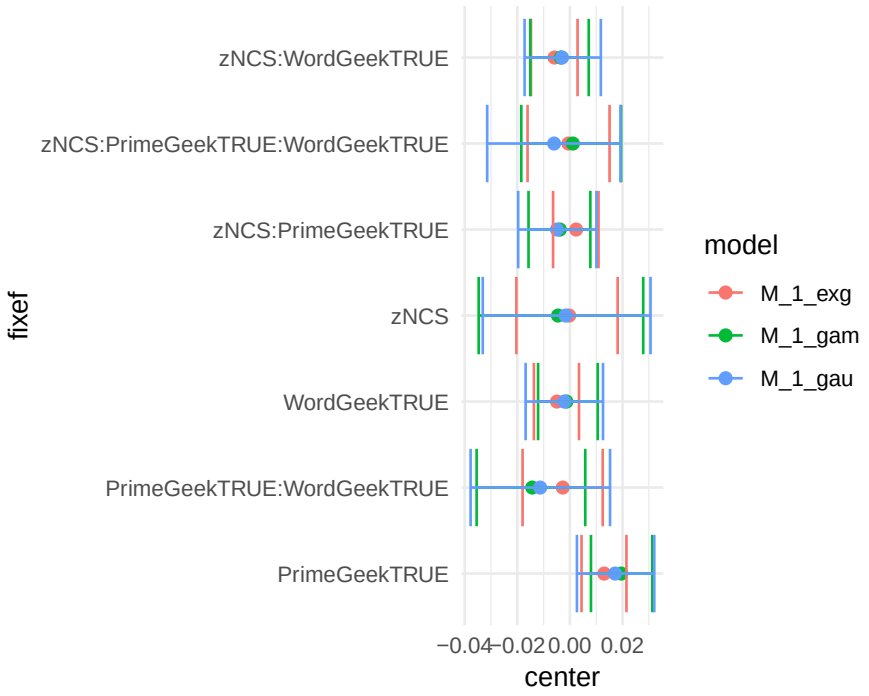
```
D_hugme <-
  filter(D_hugme, correct) %>%
  select(Obs, Part, zNCS, WordGeek, PrimeGeek, RT) %>%
  as_tbl_obs()
D_hugme
```

Obs	Part	zNCS	WordGeek	PrimeGeek	RT
3	1	0.318	FALSE	FALSE	0.656
1771	26	0.703	FALSE	FALSE	1.066
1885	28	0.382	FALSE	FALSE	0.798
2194	32	1.281	TRUE	FALSE	0.500
2798	54	-1.801	FALSE	FALSE	0.475
3595	66	-0.260	TRUE	FALSE	0.813
3608	66	-0.260	FALSE	FALSE	0.688
3996	78	-0.324	FALSE	TRUE	0.645

```
F_1 <- formula(RT ~ zNCS*PrimeGeek*WordGeek + (1|Part))

M_1_gau <- D_hugme %>%
  brm(F_1,
      family = gaussian,
      data = .)

M_1_exg <- D_hugme %>%
  brm(F_1,
      family = exgaussian,
      data = .)

M_1_gam <- D_hugme %>%
  brm(F_1,
      family = Gamma(link = identity),
      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()
```

The following plot shows the fixed effects for the three random components:

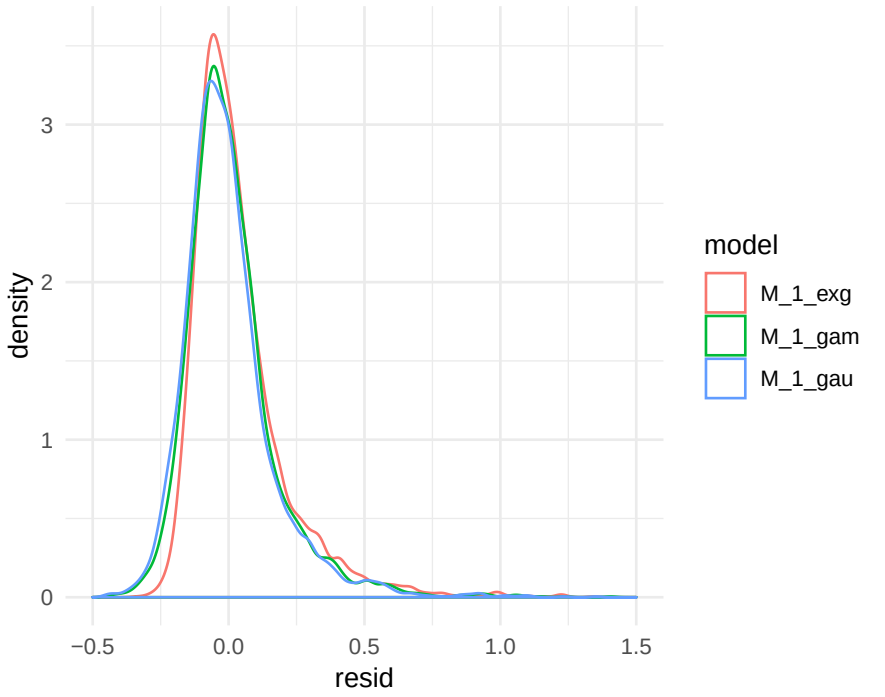
```
T_1_fixef <- fixef(P_1)

T_1_fixef %>%
  filter(fixef != "Intercept") %>%
  ggplot(aes(y = fixef, x = center, xmin = lower, xmax = upper, color = model, group = model
  geom_point(size = 2) +
  geom_errorbarh()
```



None of the effects have considerable impact. Nevertheless, it seems that the Exgaussian model produces much tighter credibility limits, i.e. better degrees of certainty. In any case, if there is a reason to prefer the exgaussian model, we should primarily see that in how the residuals are shaped. In the following plot it can be seen, that the extra flexibility of the exgaussian is employed, indeed. Both, Gaussian and gamma are unable to accomodate the steep left climb and are producing a visible drag to the left.

```
D_hugme <- D_hugme %>%
  left_join(T_1_predict) %>%
  mutate(resid = RT - center)

D_hugme %>%
```

```
ggplot(aes(x = resid, color = model)) +
geom_density() +
xlim(-.5,1.5)
```



7.5.3.2 Modelling Time-on-task

Although experimental psychologists call the Stroop task a complex one, the minimal processing time is rather short in comparison to tasks usually administered in usability studies, such as finding information on websites or renting cars online. 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 we will see in a later section, the remote condition is contaminated with cheaters. Therefore, we only use the moderated sessions. In addition, it is interesting to compare the impact of the chosen distribution on effects estimates. For this reason only, we include the factor Task as a fixed effect (with treatment contrasts), despite this not being very meaningful.

The comparison draws upon the effects, as well as residual distributions. For the latter, the predictive posterior distribution is required. With several hundred observations, this results in a very large object. To prevent running into

the memory limit, we crank it up (`memory.limit`) and thin out the number of posterior predictive samples by factor 5. Finally, we combine the three posterior predictive distributions and examine the differences.

```
attach(CUE8)
```

```
D_cue8_mod <- D_cue8 %>%
  filter(Condition == "moderated", !is.na(ToT)) %>%
  as_tbl_obs()
```

```
memory.limit(16000)

F_4 <- formula(ToT ~ 1 + Task + (1|Part) + (1|Team))

M_4_gau <- D_cue8_mod %>%
  brm(F_4,
      family = gaussian,
      data = .)

M_4_exg <- D_cue8_mod %>%
  brm(F_4,
      family = exgaussian,
      data = .)

M_4_gam <- D_cue8_mod %>%
  brm(F_4,
      family = Gamma(link = identity),
      data = .)

P_4 <- bind_rows(
  posterior(M_4_gau),
  posterior(M_4_gam),
  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()
```

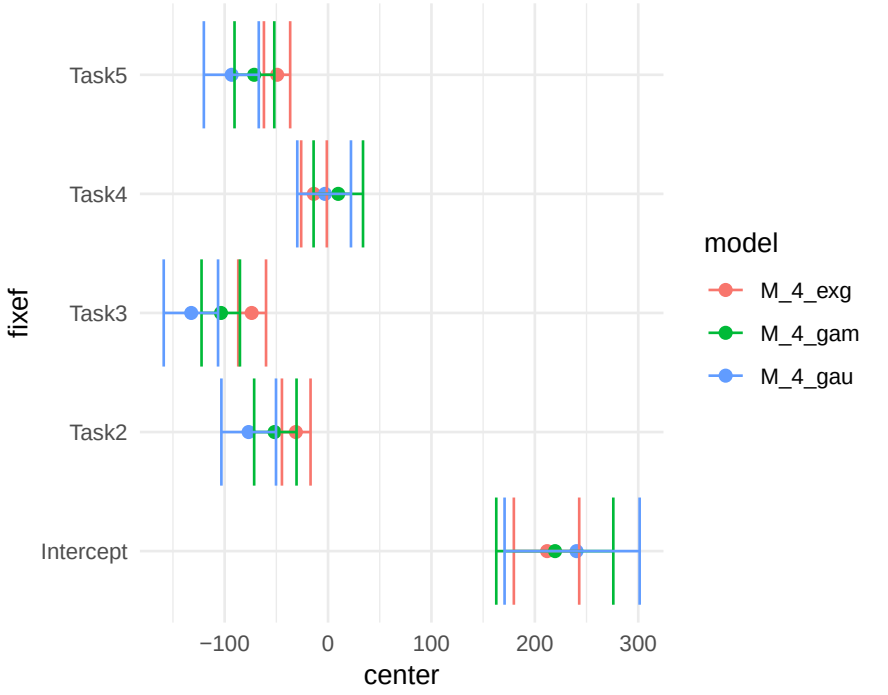
```
fixef(P_4)
```

model	fixef	center	lower	upper
M_4_exg	Intercept	211.87	179.7	242.93
M_4_exg	Task2	-31.15	-44.6	-16.94
M_4_exg	Task3	-73.86	-86.9	-59.97
M_4_exg	Task4	-13.82	-26.0	-1.18
M_4_exg	Task5	-49.23	-62.0	-36.65
M_4_gam	Intercept	219.58	162.7	275.89
M_4_gam	Task2	-51.72	-71.5	-30.49
M_4_gam	Task3	-103.55	-122.4	-85.23
M_4_gam	Task4	9.84	-14.0	33.81
M_4_gam	Task5	-71.43	-90.5	-52.00
M_4_gau	Intercept	240.33	170.7	301.38
M_4_gau	Task2	-76.84	-103.2	-50.34
M_4_gau	Task3	-132.31	-158.9	-106.38
M_4_gau	Task4	-2.94	-29.7	22.16
M_4_gau	Task5	-93.21	-120.1	-67.02

```
grpef(P_4)
```

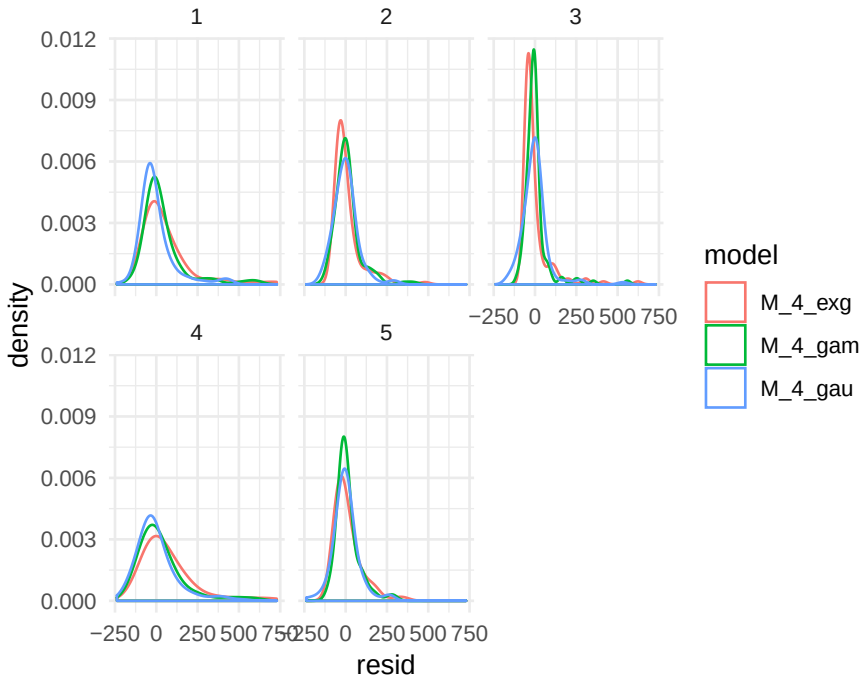
model	re_factor	center	lower	upper
M_4_exg	Part	23.4	16.2	31.5
M_4_exg	Team	37.3	21.4	76.5
M_4_gam	Part	40.2	29.3	52.0
M_4_gam	Team	71.6	42.0	141.5
M_4_gau	Part	65.2	52.4	80.2
M_4_gau	Team	82.2	47.3	157.0

```
fixef(P_4) %>%
  ggplot(aes(y = fixef, x = center, xmin = lower, xmax = upper, color = model)) +
  geom_point(size = 2) +
  geom_errorbarh()
```



As with reaction times, the three models produce rather different fixed effects estimates and with the tendency that the Gaussian, followed by the gamma model, exaggerates the effects. Again, the exgaussian model produces estimates of superior certainty.

```
D_cue8_mod %>%
  left_join(T_4_predict) %>%
  mutate(resid = ToT - center) %>%
  ggplot(aes(x = resid, color = model)) +
  facet_wrap(~Task) +
  geom_density(adjust = 2)
```



Inspecting the residual distributions yields a different pattern as with RT: generally, the left skewness is much less pronounced and Gaussian and gamma even tend to be right skewed. Strikingly, the residuals of the exgaussian models sit much tighter around the center, which corresponds with the narrower credibility intervals for the fixed effects.

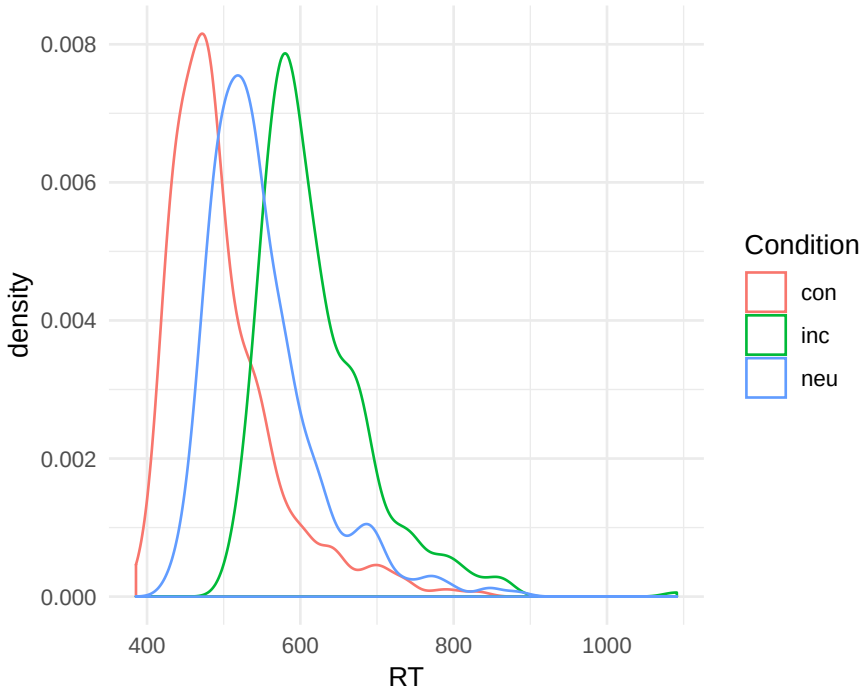
In general, it seems that the exgaussian model for RT and ToT accomodates left skewness better and produces estimates that are more conservative and certain at the same time. Could it be true, that Gaussian and gamma models overestimate group mean differences for left skewed RT and ToT responses? We examine this possibility by simulating a well-known experiment using an exgaussian distribution.

```
sim_Stroop <- function(beta_0 = 500, # congruent
                       beta_1 = 50,  # neutral
                       beta_2 = 120, # incongr
                       sigma = 20,
                       beta = 70,
                       N = 400,      # obs per condition
                       seed = 42){
  set.seed(seed)
  data_frame(Condition = rep(c("con", "neu", "inc"), N),
```

```
            mu = beta_0 + beta_1 * (Condition == "neu") + beta_2 * (Condition == "inc"),
            RT = rexgaussian(N * 3, mu, sigma, beta)) %>%
    as_tbl_obs()
  }

D_stroop <- sim_Stroop()

D_stroop %>%
  group_by(Condition) %>%
  summarize(mean(RT)) %>%
  kable()
```

Condition	mean(RT)
con	497
inc	619
neu	549

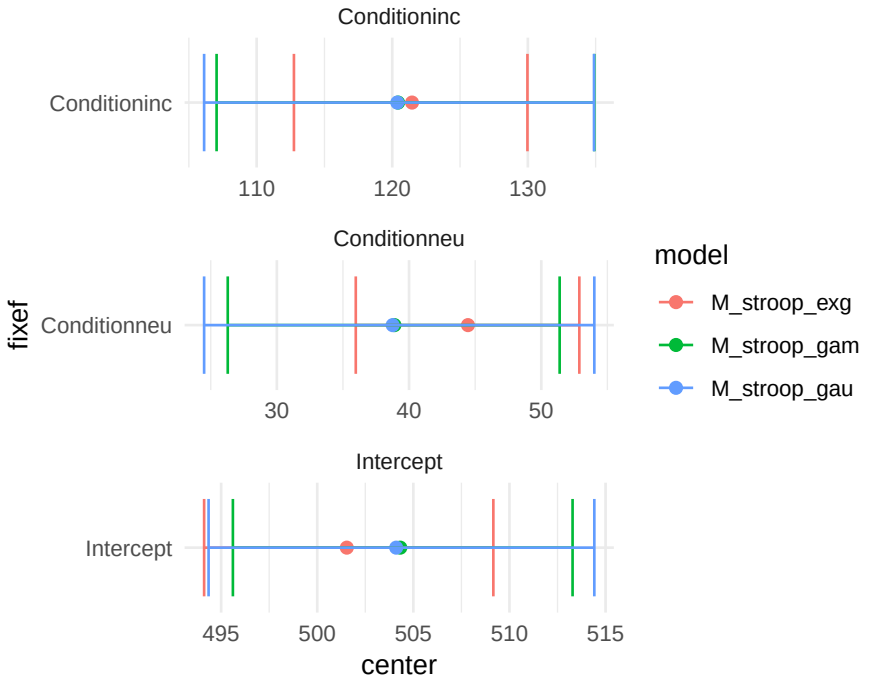
```
D_stroop %>%
  ggplot(aes(x = RT, col = Condition)) +
  geom_density()
```



In the same way as above, we recover the effects by three models that have the same likelihood, but differ in their response distribution.

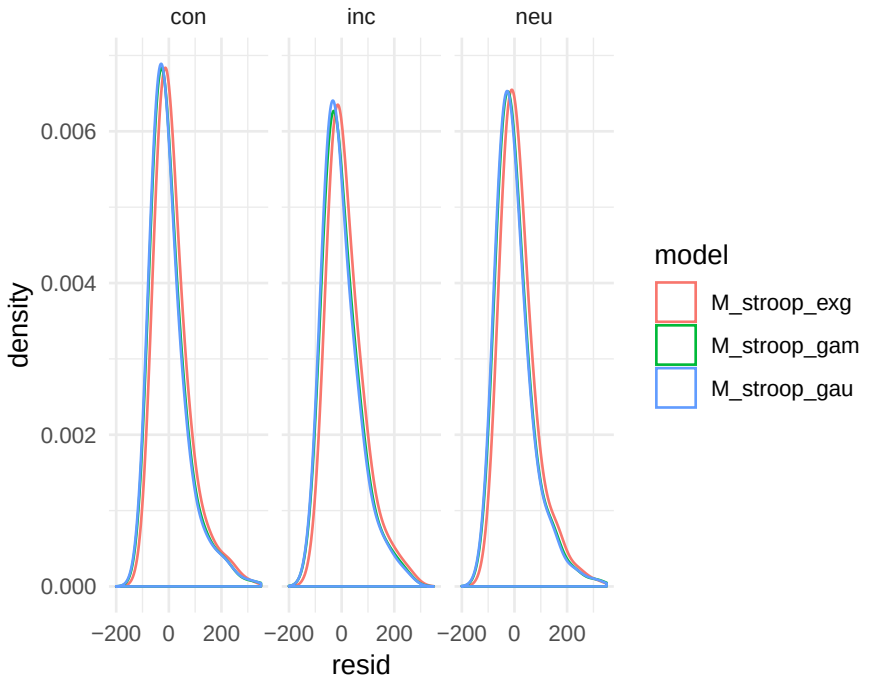
```
F_stroop <- formula(RT ~ Condition)

M_stroop_gau <- D_stroop %>%
  brm(F_stroop,
      family = gaussian,
      data = .)

M_stroop_exg <- D_stroop %>%
  brm(F_stroop,
      family = exgaussian,
      data = .)

M_stroop_gam <- D_stroop %>%
  brm(F_stroop,
      family = Gamma(link = identity),
      data = .)

P <- bind_rows(
  posterior(M_stroop_gau),
  posterior(M_stroop_gam),
  posterior(M_stroop_exg)
)


T_stroop_predict <- bind_rows(
  post_pred(M_stroop_gau, thin = 4),
  post_pred(M_stroop_gam, thin = 4),
  post_pred(M_stroop_exg, thin = 4)
) %>%
  predict()
```

```
fixef(P_stroop) %>%
  ggplot(aes(y = fixef, x = center, xmin = lower, xmax = upper, color = model)) +
  facet_wrap(~fixef, ncol = 1, scales = "free") +
  geom_point(size = 2) +
  geom_errorbarh()
```



What is confirmed by the simulation is that the exgaussian model, when it is the true one, produces more certain estimates.

```
D_stroop  %>%
  left_join(T_stroop_predict) %>%
  mutate(resid = RT - center) %>%
  ggplot(aes(x = resid, color = model)) +
  facet_wrap(~Condition) +
  geom_density(adjust = 2) +
  xlim(-200, 350)
```



Different to what has been observed above, the shape of residual distributions do not differ much in shape, except for a shift offset to the right. What could be accountable for that is that the simulation only contained three homogenous groups, rather than the many groups in the previous multi-level data sets (random factors). It remains to be clarified what precisely the biases and drags are caused by ill-specified response distribution for RT and ToT in complex research designs. Despite these question marks, it has been confirmed that the superior certainty of estimates is not just an artifact of the exgaussian model, but is real and likely to make quantitative inference from RT and ToT data more efficient.

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. But, what risk is there to get impossible, i.e. negative, predictions? We check this on the posterior predictive distributions of both studies, CUE8 and Hugme. The following table shows the proportion 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	0.029
M_4_gam	0.000
M_4_gau	0.714

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 is 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. Possibly, exgaussian models are more efficient for carving out delicate cognitive effects in comparison to Gaussian models (not to mention non-parametric tests).

However, as the discussion has not even fully started, to declare it settled would be premature. In contrast, the aim of this chapter 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.

That being said, the brms engine offers even more opportunities. First, it supports two more distributions with an offset component: the shifted log-

normal and the Wiener distribution. Interestingly, the latter grounds on one of the few formally specified cognitive process models, the diffusion model for simple choice tasks. All four parameters of the Wiener distribution are directly linked to individual elements of the cognitive process. This brings us to the second relevant extension of brms, which I will not fully cover, but is worth mentioning: *distributional models*. The vast majority of statistical analysis capitalizes on the location parameters. We ask whether an increase in a continuous predictor causes an increase in the outcome or if one group has a higher average than the other.

Only in the analysis of random effects have we drawn conclusions from dispersion parameters, such as to test Egans claim. In design research, the variance in performance is a crucial issue. To give another example: reportedly, several areas of cognitive functioning deteriorate with age, on average, but variance typically increases. It was the very idea of Dennis Egan that designs should not just improve performance on average, but also keep variance at a minimum. Hence, linking variance to predictors, such as design and age, can be a fruitful endeavour under the paradigm of robust designs. For RT the beforementioned Wiener distribution matches RTs in simple choice tasks and every parameter corresponds with an element in the so called diffusion model. In design research, such ideas are almost unexplored. Perhaps, one day a researcher finds that the gaussian and the exponential component are influenced by different design features.

7.5.4 Literature

7.5.5 Exercises:

1. Review the literature on reaction times. What is the shortest time length you can find?
2. The user can only do things one after the other and therefore, RT and ToT will never come close to zero. Conceive examples of independently triggered events that *happen* to users and effect user satisfaction.

7.6 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 that this is happening 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 guts 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), they may end up with a lower anchor than today's kids. If too few cues are given upfront, participant 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 the website of Mapple 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):

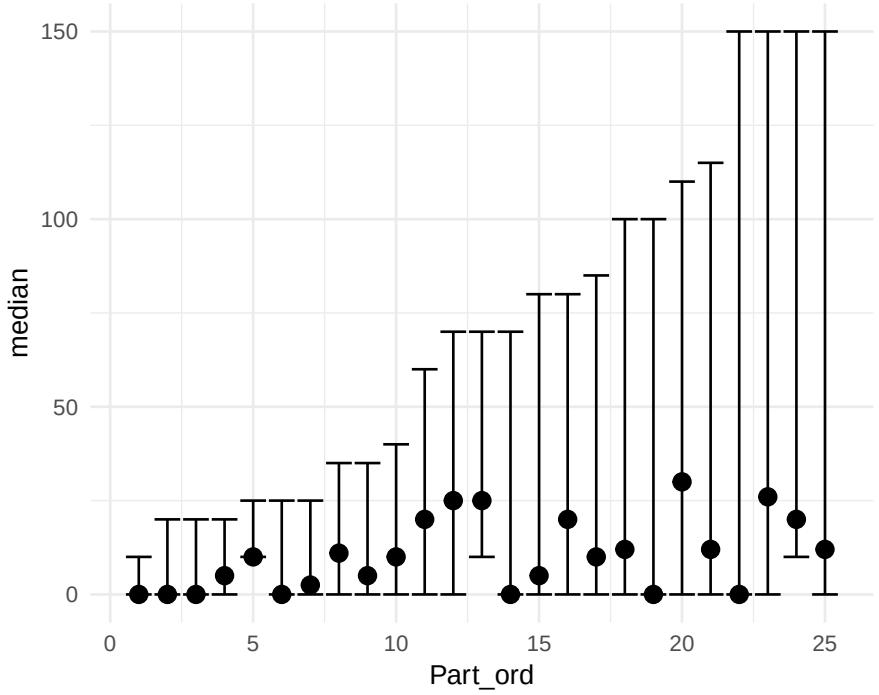
1. A is undecisive and stays in the center region
2. B has a crush on the experimenter and responds slightly more positive
3. C politely avoids the negative extremes
4. D is a human rights activist and habitually treats the seven bins equally.
5. E is annoyed by the experiment (rightly so) and falls 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 cases 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 idiosyncracies rating scales 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 frokm 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 The sum score is still binned, but more finely grained so. 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.

7.6.1 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 category can often be thought of as in an order: success is better than failure, presence more than absence and a return better than bailing out. 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 responses 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

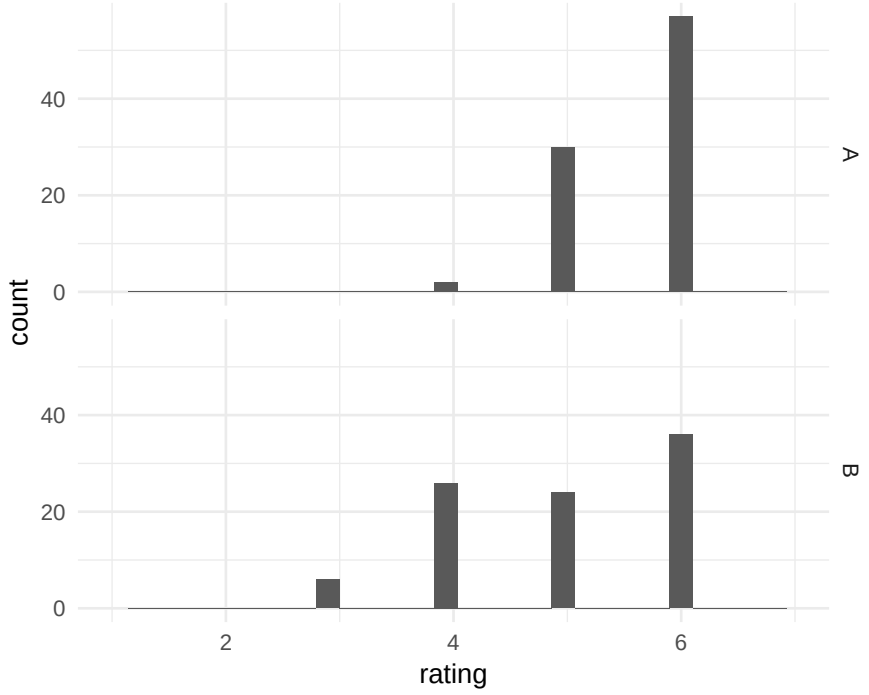
thresholds and generally, with c categories, there are $c-1$ thresholds. Ordinal regression deals with the problem by estimating $c-1$ intercept estimates $\beta_{0[k]}$. Each threshold intercept $\beta_{0[k]}$ represents the probability (on logit scale) that the response falls into category k *or lower*, 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	-12.753	-47.631	-5.870
NA	-12.979	-53.096	-5.817
NA	-4.087	-4.999	-3.260
NA	-2.263	-2.825	-1.802
NA	-1.118	-1.566	-0.744
NA	1.322	0.847	1.924
DesignB	-0.865	-1.339	-0.443

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:

$$\pi(y_i \leq 2|A) = \text{logit}^{-1}(\beta_{0[2]}) = \text{logit}^{-1}(-13) = 2.26 \times 10^{-6}$$

The expected proportion of responses equal to or smaller than 2 for design B we get by the usual linear combination:

$$\pi(y_i \leq 2|B) = \text{logit}^{-1}(\beta_{0[2]} + \beta_1) = \text{logit}^{-1}(-13.9) = 9.19 \times 10^{-7}$$

All coefficients are shifting all threshold 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 in sync. 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 sevenfolded, 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)
```

7.6.2 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 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 [CROSSREF], 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. Not just so, because 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.

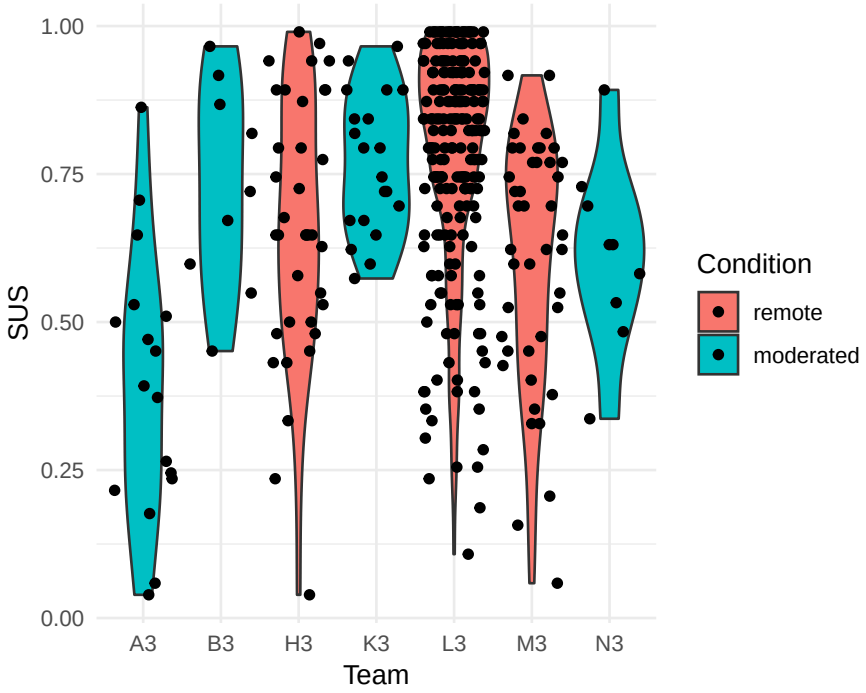
A practical solution is to scale the responses in such a way as to avoid the two boundaries, which is what the following hack does.

1. add a tiny value to all responses
2. create a divisor by adding double 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 Eff.Sample Rhat
## sd(Intercept)     0.81      0.39     0.34     1.95        297 1.02
##
## Population-Level Effects:
##                    Estimate Est.Error l-95% CI u-95% CI Eff.Sample Rhat
## Intercept              0.79      0.52    -0.51     1.78        385 1.00
## Conditionmoderated    -0.34      0.70    -1.70     1.11        967 1.00
##
## Family Specific Parameters:
##     Estimate Est.Error l-95% CI u-95% CI Eff.Sample Rhat
## phi     4.13      0.30     3.56     4.74       3362 1.00
##
## Samples were drawn using sampling(NUTS). For each parameter, Eff.Sample
## is a crude measure of effective sample size, 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, an allegedly validated rating scale.

Is that a realistic model for the SUS responses? Another glance at the violin plot suggests another pathology: the teams seem to differ in variation. In the closing section I will briefly demonstrate how to deal with differences in variance (rather than mean) by a *distributional model*.

7.6.3 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. one would think that multi-level models deal with variance to a good extent, and they do. But, they only give us the variance of a certain effect on a set of objects. That is absolutely not the same as asking: 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: GLM greatly enhanced the scope of modelling by giving us the choice of response distributions and linearizing functions. Still, all models introduced so far establish an association between predictors and the expected value μ, only. However, all but the one-parameter distributions come with additional parameters that capture a family of distributions in its whole variety of shapes. Strong variance can become a real problem, because this implies that extremely poor performance becomes more likely. Under the perspectives of safety in hazardous environments and universal usability, variance is a crucial parameter by itself. We should track carefully, that an improvement on average is not accompanied by more variance (other than the mean-variance relationship prescribed by the error distribution).

As a first illustration, imagine two versions of a continuous rating scale for visual beauty that differ in how their extreme levels are labelled:

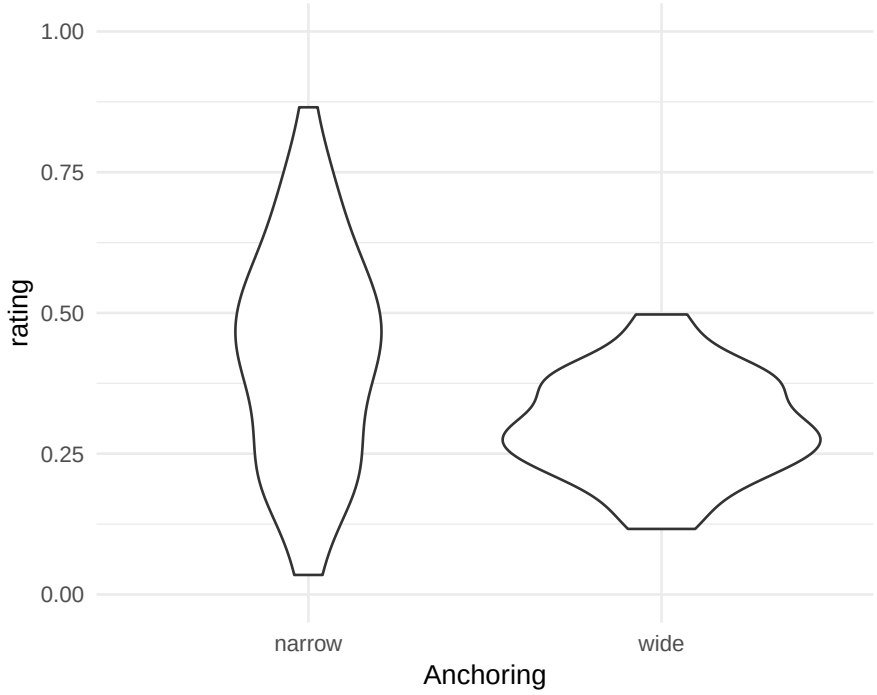
1. like the ugliest website I have ever seen: 0 ——- 1 like the most beautiful website
2. distgusting as a screenshot from a splatter movie: 0 ——- 1 like the most beautiful sunset I have seen

The second scale has the lower and upper anchors moved to more extremes. Two questions arise:

1. Is the overall location of the scale untouched, that is, have both anchors been moved by the same distance outwards?
2. What range do websites cover as compared to all thinkable visual impressions?

For the sake of simplicity (not for a strong research design), let's assume that one participant has rated a randomized sample of 100 websites in two conditions: narrow anchoring and wide anchoring. The following simulates data as if there were just a minor positive shift of the wider condition, accompanied by an immense up-scaling.

```
set.seed(42)

N = 100

Conditions <-
  frame_data(~Anchoring, ~mu, ~phi,
             "wide", .3, 18,
             "narrow", .4, 6) %>%
  mutate(a = mu * phi,
         b = phi - mu * phi)

D_Anchor <-
  data_frame(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)
```



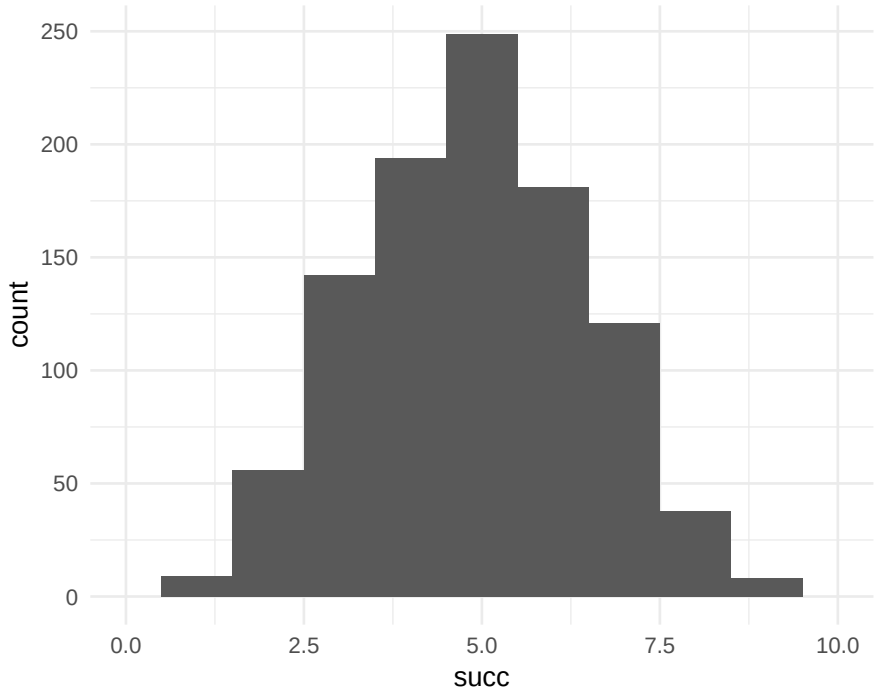
The two conditions have about the same location, but a more narrow anchoring produces a wider dispersion of responses. How would we confirm this statistically?

Models that contain predictors linked to any distribution parameter other than μ are called *distributional models* and have only recently been implemented in the brms engine. They have an immense potential as they relax another assumption of GLM, namely that all parameters, but μ, are constant across observations or follow the mean-variance relationship. Here we aim to estimate the difference in variance by experimental condition.

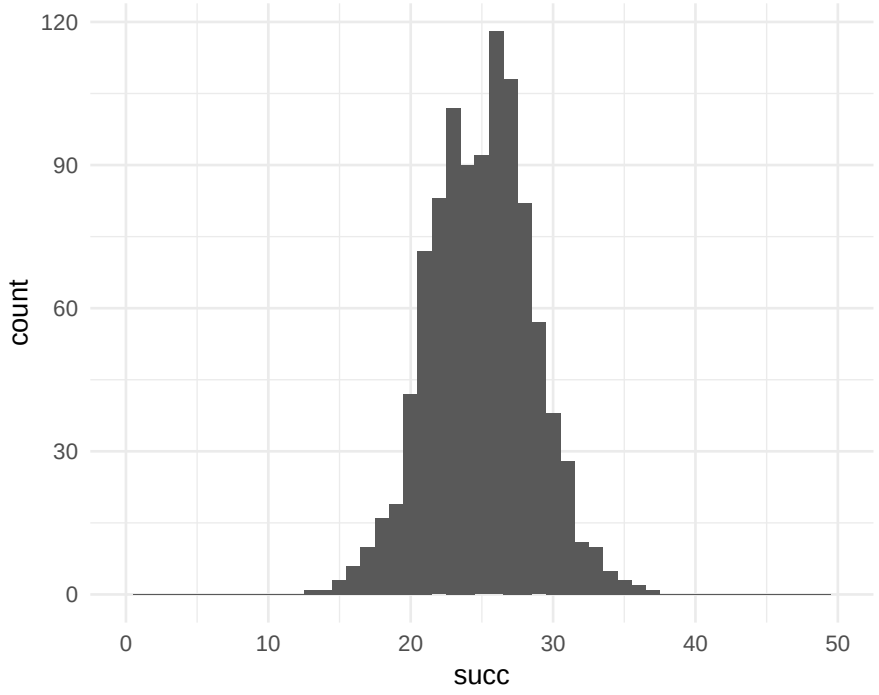
The brms engine uses a parametrization of the beta distribution (other than the more common a, b parametrization), where μ is the location and *phi* is a *scale* parameter. Take great care: "scale" does not refer to dispersion itself. Rather, it is comparable to the number of trials in a binomial process. The more trials there are, the tighter the distribution becomes, *relative* to the boundaries:

```
plot_rbinom <- function(N, size, p) {
  data_frame(succ = rbinom(N, size, p)) %>%
  ggplot(aes(x = succ)) +
  geom_histogram(bins = size + 1) +
  xlim(0, size)
```

```
}

plot_rbinom(1000, 10, .5)
```




```
plot_rbinom(1000, 50, .5)
```



In effect, increasing ϕ results in reduced dispersion and accordingly, when used in a distributional model, positive efefcts decrease variance and negative increase. This all works on top of to the mean-variance relationship of the beta distribution.

When estimating dispersion or scale parameters, we have to regard that these are positive, strictly. When linking a linear term to such a parameter, predictions are generated for the variance and these should never become negative. The brm engine simply extends the principle of link functions to parameters other the μ and sets a default log link for ϕ. In order to estimate the changes in μ and ϕ simultanously, the brm engine receives two regression formulas, which is done by the command `bf()`.

```
M_beta <- brm(bf(rating ~ 1 + Anchoring,
                 phi ~ 1 + Anchoring),
              family = Beta(),
              data = D_Anchor)

M_beta <- brm(fit = M_beta, data = D_Anchor)
```

The coefficient table shows the effect of anchoring on both parameters of the response distribution: the widely anchored item produces slightly less favorable ratings, but dramatically reduces on dispersion.

```
fixef(M_beta)
```

fixef	center	lower	upper
Intercept	-0.318	-0.544	-0.095
NA	1.630	1.247	1.986
Anchoringwide	-0.529	-0.788	-0.274
NA	1.504	0.968	2.023

Back to the CUE8 study. Do teams differ in how much variance arises, as the violin plot suggests? Before we test this by a distributional model, recall one again the principle of mean-variance relation. Beta distributions have variance tied to the mean in much the same way as binomial distribution. When moving along the range between 0 and 1, variance is largest in the center and decreases towards the boundaries. That does not seem to be the case in CUE8. Compare the distributions of Teams B and K to H and L. They all are close to the upper boundary, but show inflated variance. It could be the case, that somehow some teams trigger more extreme responses, inflating variance. The following model tests the effect of testing condition and teams on the location and dispersion of responses simultaneously.

```
F_unequal_var <- bf(SUS ~ 1 + Condition + (1 | Team),
                    phi ~ 1 + Condition + (1 | Team))

M_6_bet <-
  D_cue8_SUS %>%
  brm(F_unequal_var,
      family = Beta(), iter = 1, chains = 1,
      data = .)

M_6_bet <-
  D_cue8_SUS %>%
  brm(fit = M_6_bet, data = .,
      chains = 6,
      iter = 4000,
      warmup = 2000)

sync_CE(M_6_bet, Env = CUE8)
```

```
T_fixef <-
  brms::fixef(M_6_bet)[,c(1,3,4)] %>%
  as_data_frame(rownames = "fixef") %>%
  dplyr::rename(center = Estimate)
T_fixef
```

fixef	center	Q2.5	Q97.5
Intercept	0.751	-0.247	1.86
phi_Intercept	1.087	-0.632	1.88
Conditionmoderated	-0.211	-1.723	1.15
phi_Conditionmoderated	0.701	-0.277	2.82

The default behaviour of brm is that the scale parameter ϕ is on a log scale. As usual, lifting it to original scale by exponantiation makes it a multiplier. By $\exp(0.701) = 2.015$, we see that there is just a small difference between the two conditions. But, are there other differences between teams? The following table shows the scale multipliers for individual teams. All multipliers are closely arranged around 1, which means there is little differences. The SUS scale can safely be adminstsred in remote and moderated usability testing.

```
T_ranef <-
  ranef(M_6_bet) %>%
  filter(nonlin == "phi") %>%
  mutate_if(is.numeric, exp) %>%
  discard_redundant()
T_ranef
```

re_entity	center	lower	upper
A3	0.830	0.323	1.32
B3	0.965	0.384	1.71
H3	0.967	0.485	6.75
K3	1.151	0.720	2.95
L3	1.029	0.608	7.78
M3	1.112	0.669	9.87
N3	1.015	0.573	2.41

```
detach(CUE8)
```

7.6.4 Exercises

1. The IPump study took repeated measure of workload on a one-item quasi-continuous rating scale. Examine the effect of training and Design on workload. Then take a closer look at how participants used the scale. Build a distributional model.

Part III

Workflow

8

Data management

The process of **data modelling** starts with analyzing a domain of concepts (of interest), by identifying the properties of concepts as well as the relationships between concepts. The results of the analysis is then formalized using one of several possible standards. Most of the time, the formalization is computer-readable or computer executable. Here, I first introduce the formalization approach of Entity-Relationship Modelling and demonstrate how this approach is handled in R, using data frames and data transformations.

In statistical analysis, a good data model has three qualities:

1. efficient and accurate data entry
2. all information is preserved
3. efficient reshaping of data for subsequent analyses

Let us first consider the task of data entry and further describe what "efficiently and accurately" means here: first of all, data entry is more efficient if you have to type less. In consequence, every datum should be entered only once. That saves key presses and also reduces the risk of erroneous input, hence better accuracy. Furthermore, during the process of data entry, as much information as possible should be preserved. That regards two aspects, one should not do any aggregation during data entry. For example, the individual responses to several items of a scale should be kept and not just summed up. Second, all known, even secondary aspects of the data should be kept, such as the order of responses in an experiment. Also, relations in the data must be kept. Obviously, the identity of a participant needs to be kept for examining associations. A less obvious example is one should keep detailed information about stimuli and items of rating scales.

Now, consider the following situation: you want to validate a new personality scale using real behavioural measures. By *real* I mean that you are not just correlating the scores with scores from another rating scale, but you predict

that the trait at question has impact on how people behave in an experimental task. For example, Schmettow, Mundt & Noordzij (2013) tested whether persons with high need for cognition (NFC) have stronger associations with pictures showing computers with open cases (treatment), as compared to control pictures. This study uses data from three sources:

1. the experimental data consists of dozens or hundreds of trials per participant with a response (correct/incorrect or RT).
2. the questionnaire data consists of every participants ratings on a numnber of given items
3. demographic data captures the usual variables such as gender, age or level of education

Are you a former SPSS user, perhaps? Then, please, take a moment, a sheet and a pencil, and think about how you would structure the data from the hybrid questionnaire/experimental study. I bet, that you attempted to create *one* table that has one row per participant and is very wide, because you have assigned a column for every item and every trial of the experiment. This is the data format dictated by many of the legacy dialogues in SPSS and other statistical programs. Unfortunately, this format has severe limitations: you can add as many variables that "belong" to participants, e.g. their field of study, but you cannot properly record properties of other entities, such as stimuli or items. To give an almost trivial example: the depicted experiment used a randomized order for the pictures to occur. There might be training effects, which you may want to control (i.e. add as a covariate to your model). But, in the wide scheme you can either use the order of columns to represent the order of stimulus presentations or the target words, but not both.

Now, I will briefly present a framework for structuring (and using) data that is called *entity relationship modelling (ERM)*. ERM comes from computer science, more specifically the field of databases. In modern software systems, databases contain huge amount of data with often extremely complex relationships. As the name says, the ERM approach handles complex data by decomposition into *entities* and *relationships*. Take as an example, the data base of a company selling products to customers. The data base will contain, at least, two entities: customers and products. In the very moment the customer enters the checkout, there must be a data structure represent the shopping cart. Formally, shopping cart structure relates customers to products, in plain words: "customer x has selected products a and b". Now, returning to the wide format, this would still be possible: you simply had to create a (huge) table with all customers in rows and all (ten thousands) products in columns. To indicate that a customer has selected products a and b, mark the respective cells with 1, and leave all other cells at 0.

```
data.frame(Customer = c("...", "X", "Y", "Z"),
           Prod_a = c("...", "1", "0", "0"),
           Prod_b = c("...", "1", "0", "1"),
           Prod_c = c("...", "0", "0", "0"),
           "Prod_..." = c("...", "0", "0", "0")) %>%
  gtable()
```

Customer	Prod_a	Prod_b	Prod_c	Prod_...
...	...	...	...	...
X	1	1	0	0
Y	0	0	0	0
Z	0	1	0	0

However, in this structure it would not be possible to reasonably store the price of the product, or any other product information. And the problem becomes completely intractable when we consider all the other entities that play a role in the process, for example the fact that the same customer can have different addresses.

So, what is done in data modelling, is that *multiple tables* are being used. Áll classes of entities, customer and product, or participant and stimulus, get their own table. And, what you may find surprising at first, most relationships get their own tables, too. In the shopping cart example, we have two entities, *Customer* and *Product*, and the relationship *has_selected*. As you can see in tab_shopping_cart_wide_table, properties can be attached to entities, for example, customers have names and products have prices. These properties are called *attributes*. However, there is a twist. Could one not consider the selected products as a property, and use such an attribute? There are theoretical and

practical reasons, not to do so. Here, it should suffice to state the following rule, for when a property is an attribute, and when it is a relationship: first, the property must be expressable as a singular value, such as a number, a piece of text, or a factor level. Second, the value must be literal, in the sense that it does not represent another complex entity. If a property is atomic and literal, it can be added as an attribute to the entity and will later become a column in the respective data table.

We will return to representation of complex properties, soon. But, first we need to introduce a concept of ERM, that literally ties everything together: *keys* are unique identifier of entities. Because natural names of persons and things are often, but not always unique (think of "John Smith"), most of the time, one chooses a numerical counter as a key, like a customer ID or product number. In ERM, the attribute that uniquely identifies an entity is called the *primary key (PK)*. These keys are used for identification, but also for marking relations between entities. In Figure XY, the relation `has_selected` identifies the customer, as well as the product. These references to entities are called *foreign keys (FK)*.

By using entities with PK and establishing relations by FK, one avoids any redundancy as much as possible. As a general rule, when instantiating an entity-relationship model, one must create one table per entity and one table per relationship. There is only one exception from this rule, which has to do with the *cardinality* of a relation. The `has_selected` relationship as we have modelled it, allows that several customers select the same product, and that one customer selects multiple products. This is called an *n:m* relationship, which is the most general case. However, other cardinalities exist: consider the entity relationship model of a family tree. This model, will for certain have a entity `person`. But, do we need a seperate table for the relationship `is_child_of`? The answer is: no, because the property is singular, so we could just add it as an attribute, if it were also literal. It is not, which in the context here is just a formal issue. Still, in case of *n:1* relationships, we can add the foreign key directly to the entity. It is just not called an attribute, but a FK. There would be no harm done in terms of adding redundancy.

8.1 Modelling questionnaire responses

Returning to the original problem of a hybrid experimental/survey study, what are the entities? Clearly, participants are entities. The second set of entities are the items of personality questionnaires, and the data we can conceive as the *encounter of items and participants* that results in participants-by-items responses.

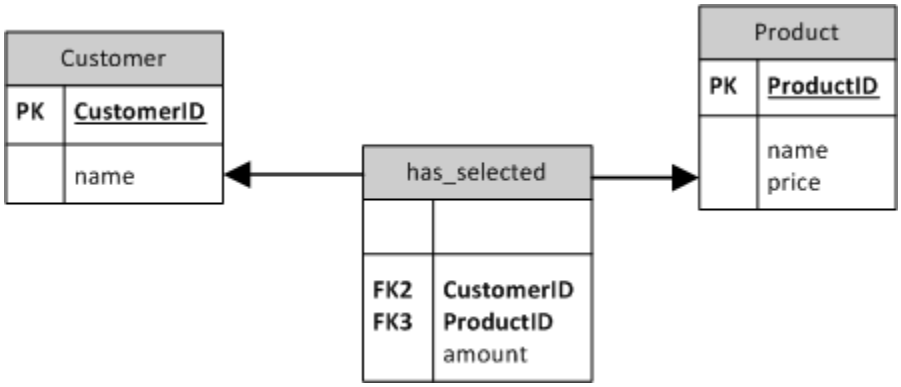


Fig. 8.1. Entity relationship model of a shopping cart

```
attach(Hugme)

T_some_items <-
  HUG3_resp %>%
  select(Item, Scale) %>%
  distinct() %>%
  filter(Item %in% c("Geek01", "Geek02", "NCS01"))

T_some_part <-
  HUG_part %>%
  slice(1:3)

T_some_resp <-
  expand.grid(Part = T_some_part$Part,
              Item = T_some_items$Item) %>%
  left_join(T_some_part) %>%
  left_join(T_some_items) %>%
  mutate(rating = round(runif(9, 1,7), 0)) %>%
  select(Part, Item, Scale, rating)

grid.arrange(
  T_some_part %>% gtable(plot = F),
  T_some_items %>% gtable(plot = F),
  T_some_resp %>% gtable(plot = F),
  layout_matrix = rbind(c(1,3), c(2,3))
)
```

Obs	Part	gender	age
1	1	female	23
2	2	male	27
3	3	female	24

Part	Item	Scale	rating
1	Geek01	Geek	5
2	Geek01	Geek	7
3	Geek01	Geek	4
1	Geek02	Geek	6
2	Geek02	Geek	4
3	Geek02	Geek	5
1	NCS01	NCS	5
2	NCS01	NCS	1
3	NCS01	NCS	6

Item	Scale
Geek01	Geek
Geek02	Geek
NCS01	NCS

So, according to the ERM framework, we are going to have two separate entity tables, at least. The *participants table* has one row per participant, identified by a unique participant ID, and all singular attributes of participants, like gender, age etc. The *items table* carries the questionnaires. Items are uniquely identified by an ID. When several scales are used, it is recommended to add a separate factor variable assigning an item to a scale. This can creates some redundancy as the item name might already identify the scale. As we will later see, this Scale variable is very useful for calculating the aggregated scores. fig_questionnaire_entity gives the smallest possible item table, but further useful information about items can be added as variables, such as:

- item text in all used language
- item labels
- if an item is reversed or not
- the position of an item in the questionnaire
- any sub scale (or sub sub scale) the item belongs to

Finally, the *questionnaire table* stores the "encounter" between participants and items in the study. In ERM speak it is a *relationship table*, as it borrows entities and their keys from participants and items. The so-called *foreign key* is the combination of participants and items. Like with the entity tables, it

is important that every observation (that is a response of one participant to one item) is uniquely identified by the foreign key.

In more complicated cases, the combination of participant and item might not identify every observation uniquely. For example, a preferred method to assess a questionnaires reliability is to repeat the measurement in a second session. If we would use the item table as in fig_questionnaire_entity, the observations were no longer uniquely identified, as all participants encounter every item twice. As a solution one would simply extend the key of the table by a variable called `session`.

8.2 Modelling experimental responses

Storing experimental data follows the same principles as for questionnaire data. In terms of ERM an experiment is a relationship, the encounter, between participants and experimental stimuli. Consequently, the complete data model will comprise two entity tables, participants and stimuli, and one relationship table for the experiment. However, depending on the design of the experiment and the nature of stimuli, we can distinguish several variations. These variations regard

1. the identity versus exchangeability of stimuli
2. the research design

```
T_exchangeable_between_subj_part <-
  data_frame(Part = as.factor(c(1:3)),
             Gender = c("f","m","m"),
             age = c(23,22,26),
             Group = as.factor(c("gamer", "gamer", "no_gamer")))

T_exchangeable_between_subj_exp <-
  expand.grid(Participant = as.factor(c(1:3)),
              position = 1:3) %>%
  mutate(waiting_time = runif(9, 10, 60),
         RT = brms::rexgaussian(9, 200, 20, 100))

grid.arrange(T_exchangeable_between_subj_part %>% gtable(plot = F),
             T_exchangeable_between_subj_exp %>% gtable(plot = F),
             ncol = 2)
```

Part	Gender	age	Gr
1	f	23	ga
2	m	22	ga
3	m	26	no_g

Participant	position	waiting_time
1	1	59.1
2	1	26.1
3	1	58.2
1	2	24.0
2	2	41.7
3	2	57.5
1	3	14.6
2	3	20.7
3	3	16.8

```
detach(Hugme)
```

The model gets just slightly more complex when the experiment comprises a condition. Assume an experiment that examines the difference in vigilance between gamers and non-gamers. As this is a between-subject design, the condition is assigned to the participant table, as is shown in fig_experiment_exchangeable_data_model. Note that the experiment table only contains the absolute necessary attributes, namely all information that characterizes the encounter itself. Of course, for the data analysis we need a table that contains information on participants as well. A later section will show how to combine and manipulate data from separate tables.

```
T_Stroop_part <-
  data_frame(Part = as.factor(c(1:3)),
             Gender = c("f","m","m"),
             age = c(23,22,26))

T_Stroop_stim <-
  data_frame(Stim = as.factor(1:4),
             Word = c("red", "blue", "wood", "dog"),
             Color = c("red","green","blue", "green"),
```

```
            Condition = ifelse(Word == Color, "congruent",
                               ifelse(Word %in% c("red", "green", "blue"), "congruent",
                                      "neutral")))
T_Stroop_exp <-
  expand.grid(Part = as.factor(c(1:3)),
              Stim = as.factor(c(1:4))) %>%
  mutate(RT = brms::rexgaussian(n(), 200, 20, 100 )) %>%
  arrange(Part) %>%
  slice(1:9)

grid.arrange(
  T_Stroop_part %>% gtable(plot = F),
  T_Stroop_stim %>% gtable(plot = F),
  T_Stroop_exp %>% gtable(plot = F),
  layout_matrix = rbind(c(1,3), c(2,3))
)
```

Part	Gender	age
1	f	23
2	m	22
3	m	26

Part	Stim	RT
1	1	164
1	2	144
1	3	167
1	4	144
2	1	205
2	2	132
2	3	106
2	4	144
3	1	267

Stim	Word	Color	Condition
1	red	red	congruent
2	blue	green	congruent
3	wood	blue	neutral
4	dog	green	neutral

In other experiments stimuli are not exchangeable at all, but have a clear identity. Typically this is the case when stimuli are not strictly designed, but taken from a natural population of stimuli, such as words or statements (from a language) and pictures (of faces, websites, objects etc.). Take as an example,

the classic Stroop experiment, where participants respond to the color a word is shown in. A typical finding is that when the word denotes an incongruent color (like the word "yellow" in red letters), then response times are delayed as when the word is congruent or just neutral. Most experimenters would probably just record the condition of presentation (congruent, incongruent or neutral). However, it could be worth the effort to be able to distinguish between words. Imagine in a Stroop experiment the word "wood" were used in the neutral condition. One could then well argue whether this word was neutral, indeed, as trees are associated with green color. Only if the identity of word stimuli is preserved in the data, is this a testable issue, and only then would the experimenter be able to remove the critical trials from the data set.

In the HUG3 experiment, the semantic Stroop task was used to detect whether a person feels attarcted by computers as objects of playful exploration and intellectual challenge. The stimuli were rather complex: first, a picture was shown, then a word was displayed and the participant had to respond to the color. The idea behind the semantic Stroop task is that a strong semnatic association between picture and word would cause a moment of distraction. Delayed responses on pictures of computers followed by words such as "playful" or "challenging" is therefore taken as an indicator for a person's computer enthusiasm (or: geekism).

```
T_SemStroop_part <-
  data_frame(Part = as.factor(c(1:3)),
             Gender = c("f","m","m"),
             age = c(23,22,26),
             Study = c("psy", "cs", "psy"))

T_SemStroop_word <-
  data_frame(Word_EN = c("mastering", "chic", "useful", "adapting"),
             Word_NL = c("behappen", "vlot", "nuttig", "aanpassen"),
             word_cat = c("geek", "hedonist", "utility", "geek"))


T_SemStroop_prime <-
  data_frame(Prime = c("img001.png", "img002.png", "img003.png", "img004.png"),
             Prime_cat = c("geek", "control", "geek", "neutral"))

T_SemStroop_stim <-
expand.grid(Prime = unique(T_SemStroop_prime$Prime),
            Word_EN = unique(T_SemStroop_word$Word_EN),
            stringsAsFactors = F) %>%
  slice(c(1,6,9,13)) %>%
#   left_join(T_SemStroop_prime) %>%
#   left_join(T_SemStroop_word) %>%
```

```
  mutate(Stim = as.factor(row_number())) %>%
  select(3,1,2)

T_SemStroop_exp <-
  expand.grid(Part = as.factor(c(1:3)),
              Stim = as.factor(1:4)) %>%
  arrange(Part) %>%
  group_by(Part) %>%
  mutate(position = sample(1:4,4)) %>%
  ungroup() %>%
  mutate(RT = brms::rexgaussian(n(), 700, 20, 100 )) %>%
  arrange(Part, position) %>%
  slice(1:9)

grid.arrange(
  T_SemStroop_part %>% gtable(plot = F),
  T_SemStroop_word %>% gtable(plot = F),
  T_SemStroop_prime %>% gtable(plot = F),
  T_SemStroop_stim %>% gtable(plot = F),
  T_SemStroop_exp %>% gtable(plot = F),
  layout_matrix = rbind(c(1,2,5), c(3,4,5))
)
```

rt	Gender	ag
	f	2
	m	2
	m	2

Word_EN	Word_NL	word_cat
mastering	behappen	
chic	vlot	
useful	nuttig	
adapting	aanpassen	

Part	Stim	position	P
1	2	1	6
1	3	2	6
1	1	3	6
1	4	4	6
2	2	1	6
2	4	2	7
2	3	3	6
2	1	4	6
3	1	1	8

Prime	Prim	Stim	Prime	W
ng001.png	g	1	img001.png	m
ng002.png	co	2	img002.png	
ng003.png	g	3	img001.png	
ng004.png	ne	4	img001.png	adapting

Both, prime pictures and target words were collected upfront and classified. As shown in fig_HUG3_experiment_data_model, the data model for this experiment consists of three entity tables (participants, pictures and words), one intermediate relationship table for stimuli and finaqlly the experiment table. As color only plays a minor role (the classic Stroop conditions no longer apply), it is not recorded in the target word table, but is generated during the experiment and could be recorded in the experiment table. Still, we could think of other properties of words to be added, such as word length an familiarity (as potential control variables), or as shown, translations for another languages.

8.3 Modelling design studies (tbd)

Design studies assess how users respond to designs. Typically, in such a test, users encounter a set of tasks. The situation is similar to that of questionnaires: a participant encounters several tasks, which may differ in relevant respects, like difficulty, length or how much it triggers frustration.

8.4 Modelling organizational structures (tbd)

8.5 Modelling longitudinal data (tbd)

8.6 Doing it in R

Now that we have seen how to structure data in various research situations, we will see how this works out in R. I will demonstrate the full workflow by example of a hybrid questionnaire/experimental study that explored in how much the *uncanny valley* phenomenon depends on attitude and personality. Participants were asked to fill out five personality scales and completed an experiment, where they rated morphing sequences of faces using items of the eeriness scale.

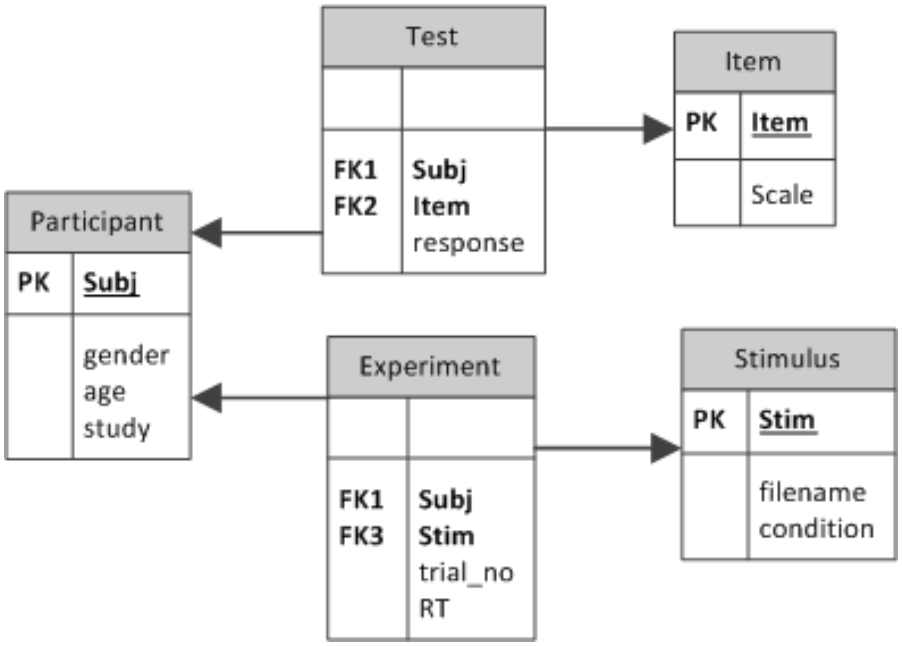


Fig. 8.2. Entity relationship model of a hybrid survey/experimental study

8.6.1 Creating and reading data

We start by creating an item table in Excel, using the following structure:

- *Scale* identifies the scale an item belongs to
- *Item* is the item name
- *Label* is a short phrase representing the item content
- *reverse* is set to -1, when an item is reversed, otherwise 1
- *minValue* and *maxValue* give the range of possible responses

Then we read the item table into R, create an empty data frame for the planned 42 participants and write it to an Excel file. In the process, we add a few demographic variables, like *Gender* and *Age*.

```
setwd("external_data/UC2/")
## reading in the items table
UC_items <-
  read.xlsx(xlsxFile = "UC_items.xlsx")

UC_items %>%
    slice(c(1:4, 13:16, 23))


## generating the questionnaire table with all participant-item combinations
## and an empty response column
UC_resp <-
  expand.grid(Participant = 1:42,
              Item = UC_items$Item) %>%
  arrange(Participant, Item) %>%
  mutate(response = "")

## printing a random selection of rows for control
UC_resp %>%
  sample_n(10)

## preparing an Excel file
UC_resp %>%
  write.xlsx(file = "UC_resp_empty.xlsx")

UC_resp %>%
  mascutils:::as_tbl_obs.tbl_df()
```

The above code used a few commands for data creation and manipulation that I will now briefly explain. First, `expand.grid(Participant = ..., Item = ...)` creates a long data frame with all possible combinations of the given levels of the given variables (here: Participant, Item). The resulting data frame is then chained into another data transformation command: `arrange` orders a data frame by the given variables (here: Participant, then Item), and outputs the re-arranged data frame. Then, `mutate` adds new variables to the data

frame. Often `mutate` is used to alter variables or compute new ones. When creating the empty response table, it is trivially used to newly create the variable for responses, which is manually entered. The full chain, from making the grid to creating the empty response variable is then stored in the variable `UC_resp`

The next command, `sample_n()`, randomly selects a number of rows from a data frame and outputs it as a data frame. Here I used it to illustrate the structure by a few examples observations. The advantage above the more common use of `summary` is that, seeing individual observations hints you at the grouping structure. Even the small sample shown here, its is easily conveyed that observations are "cross-classified" by participants and items, as we wanted it to be. In interactive programming with R, it is highly recommended to frequently do such intermediate checks on the results of computations. Finally, teh data frame is written to an Excel file, using `write.xlsx()` (openxlsx).

The general principle of the above (and all following) code is the *chaining* of commands that read a data frame, do a transformation and output an altered data frame. The chaining of commands is done with magritte operator `%>%`. Most transformatiomn commands used in this book come from the dplyr/tidyr package bundle. The main idea of these libraries is to provide a comprehensive set of comparably intuitive transformations. Complex tasks can be done by chaining these commands together. In terms of flexibility and productivity this approach is a great improvement over the classic R way of doing things. To give just one example, the following "classic" code would order our data frame by Participants:

```
UC_resp[order(UC_resp$Participant),]
```

With dplyr, this just becomes:

```
UC_resp %>% arrange(Participant)
```

Before a chain can be established, data frames have to come from somewhere. They are either produced by reading from a file, or creating it to certain rules, such as with `expand.grid`. In chapter [SIM], we will use a combination of transformation and random number generators to simulate complete data sets. However, the regular case is that data has been stored in a file. `read.xlsx(file, sheet)` is a command from the openxlsx package, that reads a sheet from an Excel file and puts it into a data frame. For a smooth workflow, the researcher should now rename the file to *UC_resp.xlsx* (in order to not accidently overwrite it) and fill in the responses as they arrive. Then the questionnaire is read back into R with:

```
setwd("external_data/UC2/")
## reading EXcel
UC_resp <-
  read.xlsx("UC_resp.xlsx")
```

While R does not have an own data editor, it can read many different formats. The packages *foreign* and *haven* provide an abundance of methods for reading in other data formats. One commonly used format is comma-separated-value (CSV), which is shown in the code below.

```
setwd("external_data/UC2/")
## reading CSV
UC_resp <-
  read.csv("UC_resp.csv")
```

As described in the data model, the properties of participants are kept with their entity, in a separate table. Assumed, this table (`UC_part`) has been created in SPSS format (.sav), the following code will read it in:

```
setwd("external_data/UC2/")
## reading SPSS
UC_demo <-
  read.spss("UC_part.sav") %>%
  mutate(Part = as_factor(Participant))
```

Unfortunately, there are minor incompatibilities between SPSS tables and R data frames that require some extra data cleaning with `mutate()` on the data frame returned by `read_sav()` (from package haven). SPSS knows a data type *labelled vector* which is almost like a factor in R, but not exactly. While a data frame can store such a variable, other functions in R cannot deal with it. The call of `as_factor()` inside `mutate()` converts labelled vectors into real factors.

Experimental data often arrives as CSV files and often already is in the long form, with one row per observation. In the uncanny valley study, the experiment was run in PsychoPy, which generates one file per participant. In R, the following chain of transformation reads in all CSV file to be found in the given directory and combine them into one large data frame. As experimental responses are uniquely identified by the stimulus (a morphed robot face), it is required that the CSV files contain a variable for the Participant identity. Here this is *participant*, which we rename to *Participant* to comply to the rule, that identifier variables are starting with capital letters.

```
setwd("external_data/UC2/")
UC_exp <-
  dir(pattern = "csv$", recursive = T) %>>% ## piping a list
    ldply(read.csv) %>% ## appending data frames
    select(Participant  = participant,
                  Item,
                  Stimulus = trialStimulus,
                  presentationTime,
           response = rating.response) %>%
  filter(!is.na(response)) %>%
  mutate(Stimulus = str_replace(Stimulus, ".png", "")) %>%
  separate(Stimulus, c("Face", "morphLevel"))

UC_exp %>% sample_n(10)
```

The code for loading the bunch of CSV files introduces a few new functions that deserve attention. Two commands operate on the filesystem of the computer: `setwd()` sets a new working directory. Here it is used to enter the subdirectory containing the CSV files. Note how `setwd("/..")` is used to go back to the upper directory. The `dir(...)` command returns a list of all files and subdirectories in the working directory. Here we add two arguments `pattern = "csv$` filters the output, such that only files ending with "csv" are regarded. `recursive = T` searches for files in all sub and sub-sub (etc) directories. This is useful as researchers and experimental programs often organize their data in one directory per participant. Note that the results of `dir()` is not stored as a variable, but piped into the next command: the pipeline operator `%>>%` (from package pipeR) is similar to the magritte operator, but works with lists instead of data frames.

The next command `ldply(read.csv)` picks up the list of csv files from the pipeline. The special thing about `ldply` is that is takes another function, `read.csv()`, as an argument, which does the actual work. `ldply()` is a so-called higher-level function that organizes the call of another function on some data. It is a member of a whole family of –plyr functions (package plyr). The two first letters of the –plyr functions indicates the format of data input and output. Here, this is a list (of file names) as input and a data frame as output. Internally, `ldply` performs the function argument on every element of the input list, expecting that the function returns a data frame (which `read.csv` does) and stacking all data frames into one.

`ldply` returns a data frame and the chain proceeds with the `select` command (dplyr) that selects the needed columns from the input data frame. It is possible to rename columns on the spot by using assignments, such as `Stimulus = trialStimulus`. The chain proceeds with `filter`, which is the row-wise counterpart, in that it selects observations. Here it is used to filter out miss-

ing observations, which in R are coded as `NA`. Finally, we are doing some data manipulation: the stimuli stem from 20 morphing sequences with four morphing levels each. This is indirectly coded in variable Stimulus, which actually is the filename of the stimulus, with the form `<face>.<morphlevel>.png`. By calling `mutate` with the string manipulation command `str_replace` (stringr) we chop off the ".png" file ending. Then `separate` (tidyr) splits the remainders into two new variables, Face and morphLevel. After this final step we have every observation uniquely identified by participant, item, face and morphing level.

8.6.2 Reshaping data

After all data has been imported, we have two data files, that we need to merge and transform for the final analysis. Before we come to that, some more processing is required on the questionnaire data for turning the raw responses into workable scores. All the needed information is stored in the *items table*, which we have to join with the questionnaire data first. Then, we proceed in three steps:

1. reversely coded items are handled
2. scores are rescaled to the interval $[-1; 1]$
3. scores are created by averaging over items

The result of the processing chain is a data frame `UC_scores` that contains exactly one score per participant and scale.

Note that I am here making the assumption that personality scores are predictors. This is may not be the case if associations between personality factors are studied, or in longitudinal studies.

```
UC_scores <-
  UC_resp %>%
  left_join(UC_items, by = "Item") %>%
  mutate(response = ((response - minValue) * 2/(maxValue - minValue) - 1) * reverse) %>%
  group_by(Participant, Scale) %>%
  summarize(score = mean(response, na.rm = T)) %>%
  ungroup()

UC_scores %>% sample_n(10)
```

The first operation in the above command chain is `left_join`, which has the effect, that all columns of *UC_items* (Scale, nimValue, maxValue, reverse) are added to *UC_resp*, where they match by Item. `left_join()` is one of several functions for joining tables. A join usually operates on the identifiers that

the two input tables have in common, here *Item*. The various join commands differ in how they deal with rows that are present in the one table, but not in the the other. `left_join()` preserves all identities from the left table (first argument or "piped in"), even if they are not present in the right table. Missing values (`NA`) are used to fill the gaps. Identities that are only present in the right table are being dropped. A `right_join()` does the exact opposite and `full_join()` preserves all identities from both tables. `inner_join()` only uses identities that are present in both tables. `anti_join()` is a useful function for diagnosting errors, because it returns the identities that are not present in both tables.

The second element in the chain is another call to `mutate()`. It uses *minValue* and *maxValue* and *reversed* to normalize all responses to consistently be in the interval $[0; 1]$ and have the correct direction. Often, this transformation is more cosmetic, but here it is necessary, as one of the scales (arem) has items with differing ranges.

The subsequent `group_by()` declares a grouping structure on the observations. Here we want to compute the scores as the average on all items per participant, per scale. The `group_by()` is followed by `summarize()`, which computes the mean. The final command `ungroup()` turns the result into a regular data frame, which now has two key variables (Participant, Scale) and the average score. Note the argument `na.rm = T` used with `mean()`. Many commands that summarize a vector of numbers, such as `mean()`, `sum()`, `var()` and `sd()`, play it extremely safe in returning `NA`, when there is a single missing value. That is in so far good behaviour, as it reminds the researcher that missing values are present. The `na.rm = T` argument makes the functions ignore missing values.

After preparing the questionnaire data, we are now left with two data tables *UC_exp* and *UC_resp* that are both organized by one-row-per-observation principle and have the key Participant in common. In general, the further shaping of data depends on the research question and on the choice of a dependent variable. The purpose of the uncanny valley study is to predict the eeriness ratings of morphed faces by the personality and attitude scores. Accordingly, the eeriness ratings are the dependent variable and we leave the structure of the table *UC_exp* intact, while joining in the personality scores.

```
UC_1 <-
  UC_exp %>%
  join(UC_scores, by = "Participant")
```

8.6.3 Tipps and tricks

8.6.3.1 As few variables as possible, but no more

Classic R data transforming goes like below and we can make two observations on it:

```
data_frame_2 <- transformation_1(data_frame_1)
data_frame_3 <- transformation_1(data_frame_2)
data_frame_i <- transformation_i+1(data_frame_i+1)
```

The first observation is that the syntax is counter-intuitive as the order of terms is output > transformation > input. The second observation is that, with five transformation steps we would create six data frames, that are hanging around in your R environment. You could, of course, re-assign the result of every transformation the same data frame (and then have a lot of fun with debugging your program). Chaining transformations of lists and data frames (pipeR, plyr, dplyr and tidyr) doesn't have these problems:

```
data_frame_2 <-
  data.frame_1 %>%
  transformation_1() %>%
  transformation_2() %>%
  transformation_3()
```

This is almost as intuitive as programming ScratchJR, and one could be tempted to just make the whole data analysis *one chain*. Still, there are good reasons to create *intermediate variables*. As a general rule: a transformation sequence that involves costly operations should stop right after this operation, plus a few trivial data cleaning steps.

Below you see the workflow we have used for un-canning the uncanny valley data.

8.6.3.2 Transforming between wide and long format

Before we turn to merging the questionnaire and Let's first begin with an issue that of the questionnaire data: it currently is in its long form, but many statistical routines still require the broad format, like exploratory factor analysis does. The tidyr package provides the command `spread()` to accomplish this.

It can also be the other way round: you have some legacy questionnaire data, which is coded in the wide format. In this case, there is an antagonist to `spread`, the command `gather`.

8.6.3.3 Grouping variables: factors or strings?