

# **PSYC 573 Bayesian Data Analysis (2024 Fall): Course Notes**

Hok Chio (Mark) Lai

2024-11-11

# Table of contents

<b>Preface</b>	<b>3</b>
<b>I Week 1</b>	<b>4</b>
<b>1 Introduction</b>	<b>5</b>
1.1 History of Bayesian Statistics . . . . .	5
1.1.1 Thomas Bayes (1701–1762) . . . . .	5
1.1.2 Pierre-Simon Laplace (1749–1827) . . . . .	5
1.1.3 20th Century . . . . .	6
1.2 Motivations for Using Bayesian Methods . . . . .	6
1.2.1 Problem with classical (frequentist) statistics . . . . .	7
1.3 Comparing Bayesian and Frequentist Statistics . . . . .	7
1.4 Software for Bayesian Statistics . . . . .	8
<b>II Week 2</b>	<b>9</b>
<b>2 Probability</b>	<b>10</b>
2.1 History of Probability . . . . .	10
2.1.1 Games of chance . . . . .	10
2.2 Different Ways to Interpret Probability . . . . .	10
2.2.1 Classical Interpretation . . . . .	11
2.2.2 Frequentist Interpretation . . . . .	11
2.2.3 Subjectivist Interpretation . . . . .	13
2.3 Basics of Probability . . . . .	14
2.3.1 Probability Distributions . . . . .	15
2.3.2 Summarizing a Probability Distribution . . . . .	16
2.3.3 Multiple Variables . . . . .	19
2.3.4 Law of Total Probability . . . . .	23
<b>3 Bayes's Theorem</b>	<b>24</b>
3.1 Example 1: Base rate fallacy (From Wikipedia) . . . . .	24
3.1.1 Question . . . . .	24
3.1.2 Solution . . . . .	25
3.2 Bayesian Statistics . . . . .	26

3.3	Example 2: Locating a Plane . . . . .	27
3.3.1	Prior . . . . .	27
3.3.2	Likelihood . . . . .	27
3.3.3	Posterior . . . . .	29
3.3.4	Influence of Prior . . . . .	29
3.3.5	Influence of More Data . . . . .	29
3.4	Data-Order Invariance . . . . .	31
3.5	Bernoulli Likelihood . . . . .	31
3.5.1	Multiple Observations . . . . .	32
3.5.2	Setting Priors . . . . .	33
3.5.3	Summarizing the Posterior . . . . .	36
3.5.4	Influence of Sample Size . . . . .	39
3.5.5	Influence of Prior . . . . .	41
3.5.6	Remark on Grid Approximation . . . . .	44
<b>III</b>	<b>Week 3</b>	<b>45</b>
<b>4</b>	<b>Beta-Bernoulli Model</b>	<b>46</b>
4.1	Steps of Bayesian Data Analysis . . . . .	46
4.2	Beta-Bernoulli Example . . . . .	47
4.2.1	Bernoulli Model . . . . .	48
4.2.2	Exchangeability . . . . .	49
4.2.3	Check the Support . . . . .	50
4.2.4	Conjugate Prior: Beta Distribution . . . . .	50
4.2.5	Data . . . . .	53
4.2.6	Posterior . . . . .	54
4.2.7	Summarize the posterior . . . . .	55
4.2.8	Posterior Predictive Check . . . . .	55
4.2.9	Comparison to frequentist results . . . . .	59
4.2.10	Sensitivity to different priors . . . . .	60
<b>5</b>	<b>Beta-Bernoulli Model With Stan</b>	<b>61</b>
5.1	Installing Stan . . . . .	61
5.2	Fitting a Beta-Bernoulli Model in Stan . . . . .	62
5.2.1	Data Import . . . . .	62
5.2.2	Writing Stan syntax . . . . .	62
5.2.3	Compiling the Stan model from R . . . . .	63
5.2.4	Posterior Sampling . . . . .	64
5.2.5	Summarizing and plotting the posterior samples . . . . .	66
5.3	Prior Predictive Check . . . . .	68
5.4	Posterior Predictive Check . . . . .	71
5.4.1	Check 1: Sample Mean . . . . .	71

5.4.2 Check 2: Sample Mean by Age Group . . . . .	72
<b>6 Poisson Model</b>	<b>74</b>
6.1 Research Question . . . . .	74
6.2 Data Import and Pre-Processing . . . . .	74
6.3 Choosing a Prior . . . . .	75
6.3.1 Compiling . . . . .	78
6.3.2 Data for Stan . . . . .	78
6.3.3 MCMC Sampling . . . . .	78
6.3.4 Prior predictive check . . . . .	79
6.4 Posterior . . . . .	79
6.5 Posterior Predictive Check . . . . .	80
6.6 Summary of Posterior . . . . .	81
<b>IV Week 4</b>	<b>82</b>
<b>7 Hierarchical Models</b>	<b>83</b>
7.1 Hierarchical Bernoulli/Binomial . . . . .	83
7.1.1 Multiple Bernoulli = Binomial . . . . .	85
7.1.2 Multiple Binomial Observations . . . . .	85
7.1.3 Stan Code . . . . .	86
7.1.4 Prior predictive . . . . .	88
7.1.5 Calling Stan . . . . .	89
7.1.6 Table of coefficients . . . . .	91
7.1.7 Posterior Predictive Check . . . . .	92
7.1.8 Derived coefficients . . . . .	94
7.1.9 Conclusion . . . . .	94
7.1.10 Shrinkage . . . . .	94
7.1.11 Multiple Comparisons? . . . . .	96
7.2 Hierarchical Normal Model . . . . .	96
7.2.1 Eight Schools Example . . . . .	96
7.2.2 Model . . . . .	96
7.2.3 Non-Centered Parameterization . . . . .	97
7.2.4 Prediction Interval . . . . .	100
<b>V Week 5–6</b>	<b>102</b>
<b>8 Linear Models</b>	<b>103</b>
8.1 What is Regression? . . . . .	103
8.2 Linear Regression . . . . .	106
8.2.1 Example . . . . .	106

8.3	Model and Priors . . . . .	107
8.4	Model Fitting With Stan . . . . .	108
8.4.1	Prior Predictive . . . . .	109
8.4.2	Results . . . . .	111
8.4.3	Convergence . . . . .	113
8.4.4	Posterior plots . . . . .	113
8.5	Predicting a New Data Point . . . . .	117
8.6	Robust Regression . . . . .	119
<b>9</b>	<b>Multiple Predictors</b>	<b>123</b>
9.1	Stratified Analysis . . . . .	123
9.2	Introducing the <code>brms</code> package . . . . .	124
	Model formula . . . . .	125
	Setting Priors . . . . .	125
9.3	Additive Model . . . . .	128
9.4	Interaction Model: Different Slopes Across Two Groups . . . . .	130
9.4.1	Posterior predictive checks . . . . .	132
9.4.2	Conditional effects/simple slopes . . . . .	134
9.5	Interaction of Continuous Predictors . . . . .	136
9.6	Centering . . . . .	138
<b>10</b>	<b>Model Diagnostics</b>	<b>141</b>
10.1	Assumptions of Linear Models . . . . .	141
10.2	Diagnostic Tools . . . . .	142
10.2.1	Posterior Predictive Check . . . . .	142
10.2.2	Marginal model plots . . . . .	145
10.2.3	Residual plots . . . . .	148
10.2.4	Multicollinearity . . . . .	150
10.3	Other Topics . . . . .	151
<b>VI</b>	<b>Week 7</b>	<b>152</b>
<b>11</b>	<b>Model Comparison</b>	<b>153</b>
11.1	Overfitting and Underfitting . . . . .	153
11.2	Kullback-Leibler Divergence . . . . .	159
11.3	Deviance . . . . .	161
11.3.1	Experiment on Deviance . . . . .	162
11.4	Information Criteria . . . . .	164
11.4.1	Akaike Information Criteria (AIC) . . . . .	165
11.4.2	Deviance Information Criteria (DIC) . . . . .	165
11.4.3	Watanabe-Akaike Information Criteria (WAIC) . . . . .	167
11.4.4	Leave-One-Out Cross-Validation . . . . .	167

11.4.5 Example . . . . .	169
<b>12 Stacking, Regularization, and Variable Selection</b>	<b>172</b>
12.1 Stacking/Model Averaging . . . . .	172
12.1.1 Model Weights . . . . .	174
12.1.2 Stacking . . . . .	176
12.1.3 Prediction example . . . . .	176
12.2 Shrinkage Priors . . . . .	177
12.2.1 Number of parameters . . . . .	178
12.2.2 Sparsity-Inducing Priors . . . . .	179
12.2.3 Regularized Horseshoe/Hierarchical Shrinkage . . . . .	182
12.3 Variable Selection . . . . .	185
12.3.1 Projection-Based Method . . . . .	193
<b>VII Week 9</b>	<b>195</b>
<b>13 Causal Inference</b>	<b>196</b>
13.1 Potential Outcomes . . . . .	197
13.2 Directed Acyclic Graph (DAG) . . . . .	198
13.2.1 Assumptions in a DAG . . . . .	201
13.2.2 Basic Types of Junctions . . . . .	201
13.3 The Back-Door Criterion: Estimating Causal Effect in Nonexperimental Data .	202
13.4 Using Multiple Regression Model . . . . .	203
13.4.1 Table . . . . .	204
13.4.2 Posterior predictive checks . . . . .	204
13.5 Predicting an Intervention . . . . .	205
13.6 Collider . . . . .	208
13.7 Some Additional References . . . . .	214
<b>14 Mediation</b>	<b>215</b>
14.1 Summary Statistics Tables . . . . .	215
14.2 Randomization Removes Incoming Paths for Treatment . . . . .	216
14.3 Causal Chains . . . . .	220
14.3.1 Post-Treatment Bias . . . . .	220
14.4 Causal Mediation . . . . .	222
14.4.1 Using <code>brms</code> . . . . .	223
14.4.2 Using Stan . . . . .	223
14.4.3 Direct and Indirect Effects . . . . .	225
14.4.4 Controlled direct effect (CDE) . . . . .	225
14.4.5 Natural direct effect (NDE) . . . . .	226
14.4.6 Sensitivity analysis . . . . .	228

<b>VIII Week 10–11</b>	<b>233</b>
<b>15 Markov Chain Monte Carlo</b>	<b>234</b>
15.1 Monte Carlo Simulation . . . . .	234
15.2 Markov Chain Monte Carlo (MCMC): Drawing Dependent Samples . . . . .	238
15.3 The Metropolis algorithm . . . . .	238
15.4 Shiny App . . . . .	239
15.5 Example 1: Estimating the Number of People Taking the Metro . . . . .	239
15.5.1 MCMC sampling . . . . .	240
15.5.2 More on Markov Chain . . . . .	242
15.5.3 Warm-up/Burn-in . . . . .	243
15.5.4 Autocorrelation . . . . .	243
15.5.5 Acceptance Rate . . . . .	246
15.5.6 Effective sample size (ESS) . . . . .	247
15.5.7 Multiple Chains . . . . .	247
15.5.8 Poor Mixing . . . . .	248
15.5.9 Better Mixing . . . . .	249
15.5.10 Full R Code With More Iterations . . . . .	249
15.5.11 Convergence Check . . . . .	251
<b>16 Gibbs Sampling</b>	<b>255</b>
16.1 Data . . . . .	255
16.2 The Normal Model . . . . .	256
16.3 Conjugate (Actually “Semiconjugate”) Priors . . . . .	257
16.3.1 For $\mu   \sigma^2$ . . . . .	258
16.3.2 For $\sigma^2   \mu$ . . . . .	258
16.4 Prior, Model, and Posterior . . . . .	259
16.5 The Gibbs Sampler . . . . .	260
16.6 Visualizing the Jumps . . . . .	261
16.6.1 Convergence Check . . . . .	262
16.6.2 Visualizing the Marginal and Joint Posterior . . . . .	264
16.6.3 Posterior Predictive Check . . . . .	265
16.6.4 Limitations of Gibbs Sampler . . . . .	266
16.7 The Metropolis-Hastings Algorithm . . . . .	266
16.7.1 Convergence . . . . .	268
<b>17 Hamiltonian Monte Carlo</b>	<b>271</b>
17.1 Leapfrog Integrator . . . . .	279
17.2 The No-U-Turn Sampler (NUTS) . . . . .	281

<b>IX Week 12</b>	<b>282</b>
<b>18 Generalized Linear Model</b>	<b>283</b>
18.1 Overview of GLM . . . . .	283
18.2 Poisson Regression . . . . .	284
18.3 Model and Priors . . . . .	285
18.3.1 MCMC Sampling With <code>brms</code> . . . . .	285
18.3.2 Interpretations . . . . .	286
18.3.3 Rootogram . . . . .	287
18.4 Negative Binomial Model . . . . .	288
18.5 Binary Logistic Regression . . . . .	289
18.5.1 Centering . . . . .	291
18.5.2 Model and Priors . . . . .	292
18.5.3 Interpreting the results . . . . .	294
18.5.4 Posterior Predictive Check . . . . .	299
18.5.5 Linear Spline . . . . .	300
18.5.6 Posterior Prediction . . . . .	301
18.5.7 Model Comparison . . . . .	302
<b>19 Generalized Linear Model (II)</b>	<b>304</b>
19.1 Binomial Logistic Regression . . . . .	304
19.1.1 Model . . . . .	305
19.2 Ordinal Regression . . . . .	307
19.3 Model . . . . .	308
19.3.1 Posterior Predictive Check . . . . .	308
19.3.2 Plot . . . . .	309
19.4 Nominal Logistic Regression . . . . .	310
19.4.1 Model . . . . .	310
19.4.2 Model Comparison . . . . .	311
<b>References</b>	<b>313</b>

# Preface

There will be some math in this notes. Don't worry if you feel the math is challenging; for applied focused students, it is much more important to understand the concepts of Bayesian methods than to understand the mathematical symbols, as they usually can be handled by the software.

**Part I**

**Week 1**

# 1 Introduction

## 1.1 History of Bayesian Statistics

Here is a nice brief video that covers some of the 250+ years of history of Bayesian statistics:

<https://www.youtube.com/watch?v=BcvLAw-JRss>

If you are interested in learning more about the story, check out the popular science book, “[The theory that would not die](#),” by McGrawe (2011)

### 1.1.1 Thomas Bayes (1701–1762)

You may find a biography of Bayes from <https://www.britannica.com/biography/Thomas-Bayes>. There is also a nice story in the book by Lambert (2018). He was an English Presbyterian minister. The important work he wrote that founded Bayesian statistics was “An Essay Towards Solving a Problem in the Doctrine of Chances,” which he did not publish and was later discovered and edited by his friend, Richard Price, after Bayes’s death <sup>1</sup>

### 1.1.2 Pierre-Simon Laplace (1749–1827)

Laplace, a French Mathematician, was an important figure in not just Bayesian statistics but other areas of mathematics, astronomy, and physics. We know much more about the work by Laplace than by Bayes, and Laplace has worked independently on the inverse probability problem (i.e.,  $P[\text{Parameter}|\text{Data}]$ ). Indeed, he was credited for largely formalizing the Bayesian interpretation of probability and most of the machinery for Bayesian statistics, and making it a useful technique for different problems, despite the discipline being called “Bayesian.” His other contributions include the methods of least squares and the central limit theorem. See a short biography of him at <https://www.britannica.com/biography/Pierre-Simon-marquis-de-Laplace>.

---

<sup>1</sup>Price is another important figure in mathematics and philosophy, and had taken Bayes’ theorem and applied it to insurance and moral philosophy.

### 1.1.3 20th Century

Until the early 1920s, the *inverse probability* method, which is based on what is now called Bayes's Theorem, was pretty much the predominant point of view of statistics. Then a point of view later known as *frequentist* statistics arrived, and quickly became the mainstream school of thinking for statistical inferences, and is still the primary framework for quantitative research. In the early 1920s, frequentist scholars, most notably R. A. Fisher and Jerzy Neyman, criticized Bayesian inference for using subjective elements in an objective discipline. In Fisher's words,

The theory of inverse probability is founded upon an error, and must be wholly rejected—Fisher, 1925

Ironically, the term *Bayesian* was first used in one of Fisher's works. And interestingly, Fisher actually thought he “[had] been doing almost exactly what Bayes had done in the 18th century.”<sup>2</sup>

Despite criticisms from frequentist scholars, Bayesian methods have been used by scholars in the Allies in World War II, such as Alan Turing, in an algorithm to break coded messages in the Enigma machine that the German Navy used to communicate. However, because of the more complex mathematics involved in Bayesian statistics, Bayesian statistics is limited to straight-forward problems and theoretical discussions until the early 1980s, when computing speed increased tremendously and made *Markov Chain Monte Carlo*—the primary algorithm for Bayesian estimation in modern Bayesian statistics—feasible. With the help of increased computing speed, Bayesian statistics has come back and been used as an alternative way of thinking, especially given the growing dissatisfaction towards the misuse of frequentist statistics by some scholars across disciplines. Bayesian estimation methods have also been applied to many new research questions where frequentist approaches work less well, as well as in big data analytics and machine learning.

## 1.2 Motivations for Using Bayesian Methods

Based on my personal experience, Bayesian methods are used quite often in statistics and related departments, as it is consistent and coherent, in contrast to frequentist where a new and probably ad hoc procedure needed to be developed to handle a new problem. For Bayesian, as long as you can formulate a model, you just run the analysis the same way as you would for simpler problems, or in Bayesian people's words “turning the Bayesian crank,” and likely the difficulties would be more technical than theoretical, which is usually solved with better computational speed.

Social and behavioral scientists are relatively slow to adopt the Bayesian method, but things have been changing. In a recently accepted paper by Van De Schoot et al. (2017), the authors

---

<sup>2</sup>See the [paper by John Aldrich](#) on this.

reviewed papers in psychology between 1990 and 2015 and found that whereas less than 10% of the papers from 1990 to 1996 mentioned “Bayesian”, the proportion increased steadily and was found in close to 45% of the psychology papers in 2015. Among studies using Bayesian methods, more than 1/4 cited computational problems (e.g., nonconvergence) in frequentist methods as a reason, and about 13% cited the need to incorporate prior knowledge into the estimation process. The other reasons included the flexibility of Bayesian methods for complex and nonstandard problems, and the use of techniques traditionally attached to Bayesian such as missing data and model comparisons.

### 1.2.1 Problem with classical (frequentist) statistics

The rise of Bayesian methods is also related to the statistical reform movement in the past two decades. The problem is that applied researchers are obsessed with  $p < .05$  and often misinterpreted a small  $p$ -value as something that it isn’t (read Gigerenzer, 2004). Some scholars coined the term *p-hacking* to refer to the practice of obtaining statistical significance by choosing to test the data in a certain way, either consciously or subconsciously (e.g., dichotomizing using mean or median, trying the same hypothesis using different measures of the same variable, etc). This is closely related to the recent “replication crisis” in scientific research, [with psychology being in the center under close scrutiny](#).

Bayesian is no panacea to the problem. Indeed, if misused, it can give rise to the same problems as statistical significance. My goal in this class is to help you appreciate the Bayesian tradition of embracing the uncertainty in your results, and adopt rigorous model checking and comprehensive reporting rather than relying merely on a  $p$ -value. I see this as the most important mission for someone teaching statistics.

## 1.3 Comparing Bayesian and Frequentist Statistics

Attributes	Frequentist	Bayesian
Interpretation of probability	Frequentist	Subjectivist
Uncertainty	How estimates vary in repeated sampling from the same population	How much prior beliefs about parameters change in light of data
What’s relevant?	Current data set + all that might have been observed	Only the data set that is actually observed
How to proceed with analyses	MLE; ad hoc and depends on problems	“Turning the Bayesian crank”

## 1.4 Software for Bayesian Statistics

The following summarizes some of the most popular Bayesian software. Currently, JAGS and Stan are the most popular. General statistical programs like SPSS, SAS, and Stata also have some support for Bayesian analyses as well.

- [WinBUGS](#)
  - Bayesian inference Using Gibbs Sampling
  - Free, and most popular until late 2000s. Many Bayesian scholars still use WinBUGS
  - No further development
  - One can communicate from R to WinBUGS using the package `R2WinBUGS`
- [JAGS](#)
  - Just Another Gibbs Sampler
  - Very similar to WinBUGS, but written in C++, and supports user-defined functionality
  - Cross-platform compatibility
  - One can communicate from R to JAGS using the package `rjags` or `runjags`
- [Stan](#)
  - Named in honour of Stanislaw Ulam, who invented the Markov Chain Monte Carlo method
  - Uses new algorithms that are different from Gibbs sampling
  - Under very active development
  - Can interface with R through the package `rstan`, and the R packages `rstanarm` and `brms` automates the procedure for fitting models in Stan for many commonly used models

## **Part II**

## **Week 2**

# 2 Probability

## 2.1 History of Probability

### 2.1.1 Games of chance

Correspondence between French Mathematicians (Pierre de Fermat and Blaise Pascal) on gambling problem by Antoine Gombaud, Chevalier de Méré. The problem is roughly of the form<sup>1</sup>:

Imagine two people playing a multi-round game. In each round, each person has an equal chance of winning. The first person who wins six rounds will get a huge cash prize. Now, consider a scenario in which A and B have played six rounds, where A has won five and B has won one. At that time, the game had to be stopped due to a thunderstorm. Since neither A nor B have reached six wins, instead of giving the prize to either one of them, they agree to divide up the prize. What would be a fair way to do so?

The discussion led to the formalization of using mathematics to solve the problem. Basically, one way is to say if A has a 97% chance of winning the prize eventually and B has a 3% chance, then A should get 97% of the prize.

## 2.2 Different Ways to Interpret Probability

There are multiple perspectives for understanding probability.<sup>2</sup> What you've learned in your statistics training is likely based on the *frequentist* interpretation of probability (and thus frequentist statistics), whereas the foundation of what you will learn in this class is the *subjectivist* interpretation of probability. Understanding the different perspectives on probability is helpful for understanding the Bayesian framework.

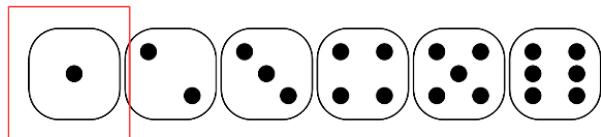
 You don't need to commit to one interpretation of probability in order to conduct Bayesian data analysis.

<sup>1</sup>see the exact form at [https://en.wikipedia.org/wiki/Problem\\_of\\_points](https://en.wikipedia.org/wiki/Problem_of_points)

<sup>2</sup>See <http://plato.stanford.edu/entries/probability-interpret/> for more information

### 2.2.1 Classical Interpretation

This is an earlier perspective and is based on counting rules. The idea is that probability is equally distributed among all “indifferent” outcomes. “Indifferent” outcomes are those where a person has no evidence to say that one outcome is more likely than another. For example, when one throws a die, one does not think that a certain number is more likely than another unless one knows that the die is biased. In this case, there are six equally likely outcomes, so the probability of each outcome is  $1 / 6$ .



### 2.2.2 Frequentist Interpretation

The frequentist interpretation states that probability is essentially the long-run relative frequency of an outcome. For example, to find the probability of getting a “1” when throwing a die, one can repeat the experiment many times, as illustrated below:

Trial	Outcome
1	2
2	3
3	1
4	3
5	1
6	1
7	5
8	6
9	3
10	3

And we can plot the relative frequency of “1”s in the trials:

As you can see, with more trials, the relative frequency approaches  $1 / 6$ . It’s the reason why in introductory statistics, many of the concepts require you to think in terms of repeated sampling (e.g., sampling distribution,  $p$ -values, standard errors, confidence intervals), because

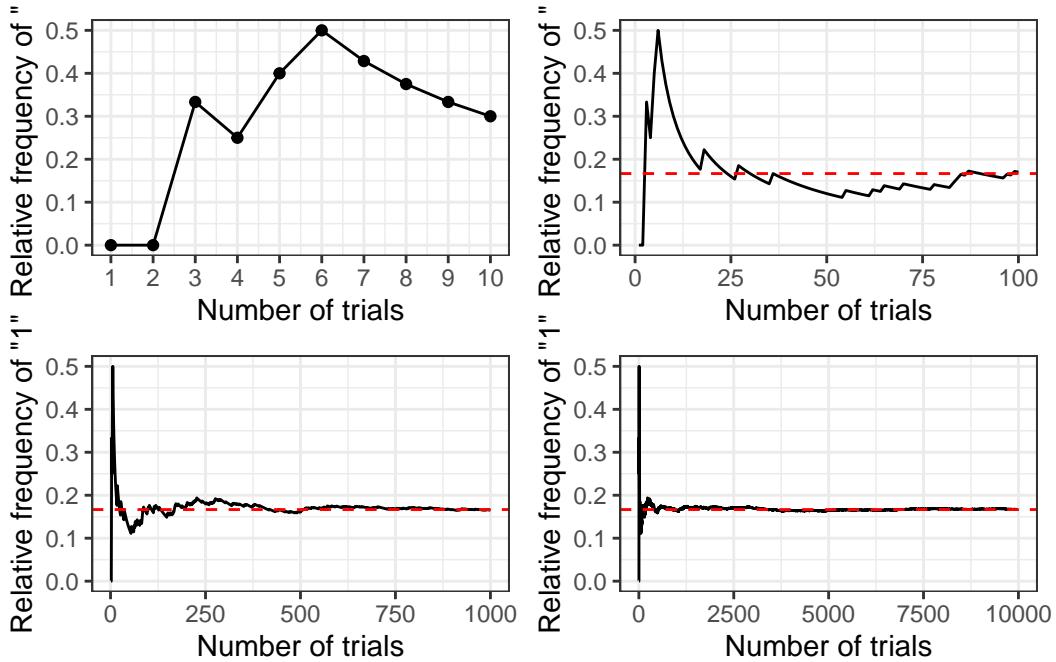


Figure 2.1: Relative frequency when repeatedly rolling a die.

probability in this framework is only possible when the outcome can be repeated. It's also the reason why we don't talk about something like:

- the probability of the null hypothesis being true, or
- the probability that the population mean is in the interval [75.5, 80.5],

because the population is fixed and cannot be repeated. Only the samples can be repeated, so probability in frequentist statistics is only about samples.

#### 2.2.2.1 Problem of the single case

Because of the frequentist's reference to long-run relative frequency, it does not make sense to talk about the probability of an event that cannot be repeated under this framework. For example, it does not make sense to talk about

- the probability that the Democrats/Republicans will win the 2028 US Presidential Election, or
- the probability that the LA Chargers winning the 2024 Super Bowl, or
- the probability that it will rain on Christmas Day in LA in 2024,

because all these are specific events that cannot be repeated. However, it is common for laypeople to talk about probabilities or chances for these events.

### 2.2.3 Subjectivist Interpretation

The frequentist interpretation is sometimes called the “objectivist view,” as the reference of probability is based on empirical evidence of long-run relative frequency (albeit hypothetical in many cases). In contrast, the *subjectivist* view of probability is based on one’s belief. For example, when I say that the probability of getting a “1” from rolling a die is  $1 / 6$ , it reflects the state of my mind about the die. My belief can arise from different sources: Maybe I make the die and know it is a fair one; maybe I saw someone throwing the die 1,000 times, and the number of “1”s was close to  $1,000 / 6$ , or maybe someone I trust and with authority says that the die has a 1-in-6 chance of showing a “1”.

The “subjective” component has been criticized a lot by frequentist scholars, sometimes unfairly. To be clear, what “subjective” here means is that probability reflects the state of one’s mind instead of the state of the world, and so it is totally fine that two people can have different beliefs about the same event. However, it does not mean that probability is arbitrary, as the beliefs are subjected to the constraints of the axioms of probability as well as the condition that the person possessing such beliefs is *rational*.<sup>3</sup> Therefore, if two persons are exposed to the same information, they should form similar, though likely not identical, beliefs about the event.

The subjective interpretation works perfectly fine with single events, as one can have a belief about whether it rains on a particular day or a belief about a particular election result.

#### 2.2.3.1 Calibrating a subjective belief

In order to represent one’s belief by probability, one needs to assign a nonzero value to every plausible outcome of an event. This has been the job of odds-makers for a long time. Indeed, a lot of the development in the field of probability has to do with coming up with a fair bet. The process of assigning probabilities to outcomes of an event according to one’s belief is called *calibration*. For example, consider three possible outcomes for tomorrow’s weather. For simplicity, consider three mutually exclusive possible outcomes: sunny, cloudy, and rainy.

To calibrate my belief, consider first if you bet \$10, and the return is (a) \$30 for sunny, (b) \$30 for cloudy, and (c) \$30 for rainy. Which one will you bet? If you’re like me in LA, I’m pretty sure I’ll bet on (a), as I think that it is more likely to have a sunny day. This means that setting  $P(\text{sunny}) = P(\text{cloudy}) = P(\text{rainy}) = 1 / 3$  is not a good reflection of my belief.

---

<sup>3</sup>In a purely subjectivist view of probability, assigning a probability  $P$  to an event does not require any justifications, as long as it follows the axioms of probability. For example, I can say that the probability of me winning the lottery and thus becoming the wealthiest person on earth tomorrow is 95%, which by definition would make the probability of me not winning the lottery 5%. Most Bayesian scholars, however, do not endorse this version of subjectivist probability and require justifications of one’s beliefs (that have some correspondence to the world).

Now consider the bet with the returns (a) \$20 for sunny, (b) \$30 for cloudy, and (c) \$60 for rainy. This would reflect the belief that there is a 50% chance of a sunny day, 33.33% chance of a cloudy day, and 16.67% chance of a rainy day. Will you take the bet? This is an improvement from the last one, but I would still say a sunny day is a good bet, which suggests that the probability of 50% is too low for a sunny day. The idea is to continue iterating until it is hard to consider (a), (b), or (c) as a clear betting favorite. For me, this would end up being something like (a) \$16.7 for sunny, (b) \$33.3 for cloudy, and (c) \$100 for rainy, which would correspond to 60% sunny, 30% cloudy, and 10% rainy.

If it's hard for you to consider the gambling analogy, an alternative way is to consider how many times is a sunny day more likely than a non-sunny day, and how many times is a cloudy day more likely than a rainy day. For example, I may consider a sunny day to be twice as likely as a non-sunny day, which would give the probability of a sunny day to be 66.67%. Then, if I also think that a cloudy day is three times as likely as a rainy day, I would assign a probability of  $33.33\% \times 3 / 4 = 25\%$  for a cloudy day, and a probability of  $33.33\% \times 1 / 4 = 8.33\%$  for a rainy day.

The process of calibrating one's belief plays a key role in Bayesian data analysis, namely in the form of formulating a *prior* probability distribution.

## 2.3 Basics of Probability

### **i** Kolmogorov axioms

For an event  $A_i$  (e.g., getting a "1" from throwing a die)

- $P(A_i) \geq 0$  [All probabilities are non-negative]
- $P(A_1 \cup A_2 \cup \dots) = 1$  [Union of all possibilities is 1]
- $P(A_1) + P(A_2) = P(A_1 \text{ or } A_2)$  [Addition rule]

Consider two events, for example, on throwing a die,

- $A$ : The number is odd
- $B$ : The number is larger than or equal to 4

Assuming that die is (believed to be) fair, you can verify that the probability of  $A$  is  $P(A) = 3 / 6 = 1 / 2$ , and the probability of  $B$  is also  $P(B) = 3 / 6 = 1 / 2$ .

### 2.3.1 Probability Distributions

- Discrete event (e.g., the outcome of throwing a die or an election): probability *mass*. The probability is nonzero, at least for some outcomes. The graph below on the left shows the probability mass of the sum of the numbers from two dice.
- Continuous event (e.g., temperature): probability *density*.<sup>4</sup> The probability is basically zero for any outcome. Instead, the probability density is approximated by  $P(A \leq a \leq A + h)/h$  for a very small  $h$ .
  - For example, to find the probability density that a person's well-being score is 80, we first find the probability that a person scores between 80 and 80.5 (or 80 and 80.0005), and divide that probability by 0.5 (or 0.0005). See the shaded area of the graph below on the right.

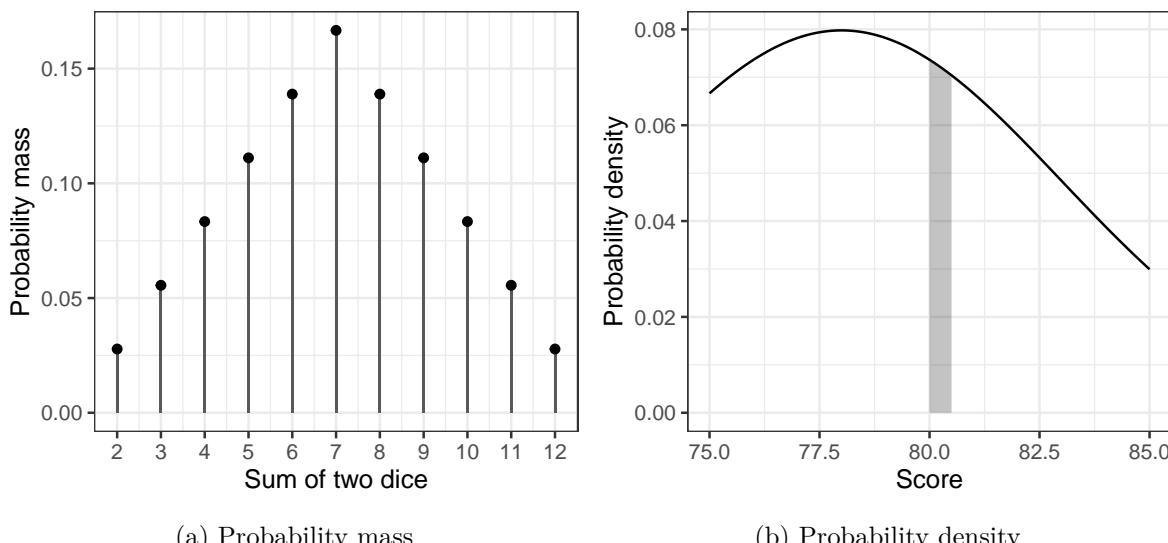


Figure 2.2: Examples of probability distributions

**i** For this course, as in the textbook, we use  $P(x)$  to mean both the probability mass for an outcome  $x$  when the event is discrete, and the probability density at an outcome  $x$  when the event is continuous.

---

<sup>4</sup>For many problems in the social and behavioral sciences, the measured variables are not truly continuous, but we still use continuous distributions to approximate them.

### 2.3.1.1 Example: Normal Distribution

$$P(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2} \left[\frac{x-\mu}{\sigma}\right]^2\right)$$

```
# Write a function to compute the density of an outcome x
# for a normal distribution
my_normal_density <- function(x, mu, sigma) {
  exp(- ((x - mu) / sigma)^2 / 2) / (sigma * sqrt(2 * pi))
}
# For example, density at x = 36 in a normal distribution
# with mu = 50 and sigma = 10
my_normal_density(36, mu = 50, sigma = 10)
```

```
#> [1] 0.01497275
```

### 2.3.2 Summarizing a Probability Distribution

While it is useful to know the probability mass/density of every possible outcome, in many situations, it is helpful to summarize a distribution by some numbers.

#### 2.3.2.1 Central Tendency

- Mean:  $E(X) = \int x \cdot P(x)dx$
- Median: 50th percentile; the median of  $X$  is  $Mdn_X$  such that  $P(X \leq Mdn_X) = 1 / 2$
- Mode: A value with maximum probability mass/density

See Figure 2.3a for examples.

#### 2.3.2.2 Dispersion

- Variance:  $V(X) = E[X - E(X)]^2$ 
  - Standard deviation:  $\sigma(X) = \sqrt{V(X)}$
- Median absolute deviation (MAD):  $1.4826 \times Mdn(|X - Mdn_X|)$

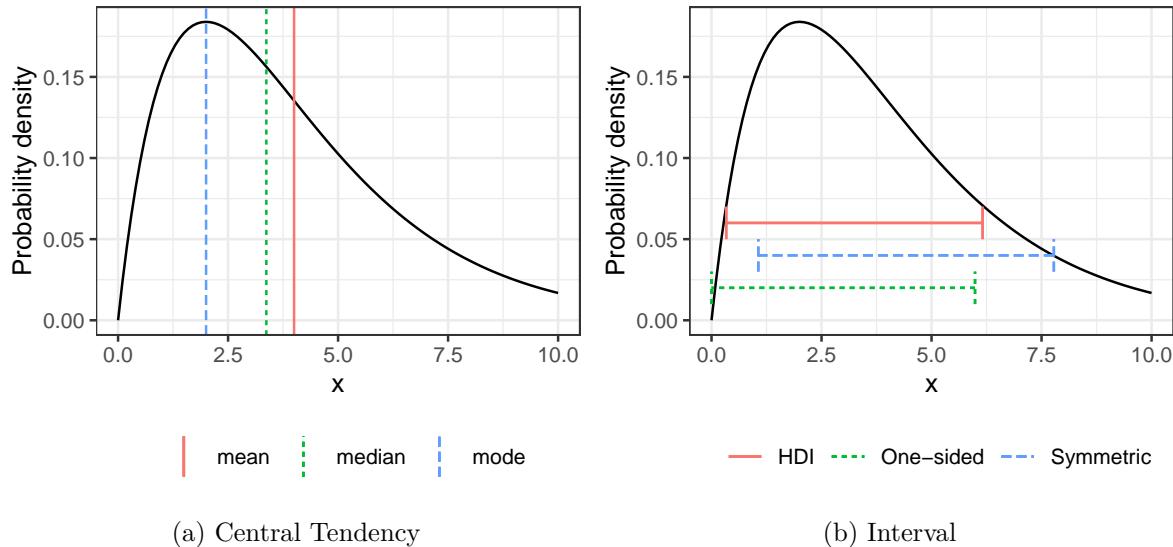


Figure 2.3: Measures of central tendency and interval

### 2.3.2.3 Interval

Use  $C(X)$  to denote an interval. A  $W\%$  interval means  $P(X \in C[X]) \approx W\%$

- One-sided interval:  $C(X)$  is half-bounded
- Symmetric  $W\%$  interval:  $C(X) = [L(X), U(X)]$  is bounded, with  $P(X < L[X]) = P(X > U[X]) \approx W\%/2$ 
  - Also called *equal-tailed* interval
- Highest density  $W\%$  interval (HDI):  $P(x_c) \geq P(x_o)$  for every  $x_c$  in  $C(X)$  and every  $x_o$  outside  $C(X)$ . In general, the HDI is the shortest  $W\%$  interval.

The plot in Figure 2.3b shows several 80% intervals.

### 2.3.2.4 Computing Summaries of Sample Distributions Using R

```
# Simulate data from a half-Student's t distribution with
# df = 4, and call it sim_s
sim_s <- rt(10000, df = 4) # can be both positive and negative
sim_s <- abs(sim_s) # take the absolute values
ggplot(data.frame(x = sim_s), aes(x = x)) +
  geom_histogram(binwidth = 0.1)
```

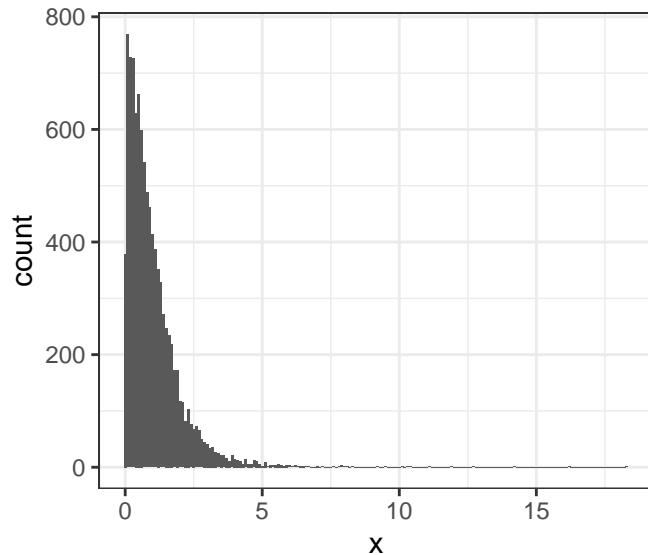


Figure 2.4: Simulated density of a half-Student's t distribution

```
# Central tendency
# (note: the mode is difficult to compute for continuous
# variables, and rarely used in this course.)
c(mean = mean(sim_s),
  median = median(sim_s),
  mode = density(sim_s, bw = "SJ")$x[
    which.max(density(sim_s, bw = "SJ")$y)
  ])
```

```
#>      mean     median      mode
#> 1.0082926 0.7426326 0.1021776
```

```
# Dispersion
c(
  variance = var(sim_s),
  sd = sd(sim_s),
  mad = mad(sim_s)
)
```

```
#> variance      sd      mad
#> 1.0231058 1.0114869 0.6996741
```

```

# 80% Interval
c(`0%` = 0, quantile(sim_s, probs = .8)) # right-sided

#>      0%     80%
#> 0.0000 1.5598

c(quantile(sim_s, probs = .2), `100%` = Inf) # left-sided

#>      20%      100%
#> 0.2680906           Inf

quantile(sim_s, probs = c(.1, .9)) # equal-tailed/symmetric

#>      10%      90%
#> 0.1299027 2.1371636

HDInterval::hdi(sim_s)

#>      lower      upper
#> 0.0003616342 2.7992620765
#> attr(,"credMass")
#> [1] 0.95

```

### 2.3.3 Multiple Variables

- Joint probability:  $P(X, Y)$
- Marginal probability:

$$P(X) = \int P(X, y) dy$$

$$P(Y) = \int P(x, Y) dx$$

- The probability that outcome  $X$  happens, regardless of what values  $Y$  take.

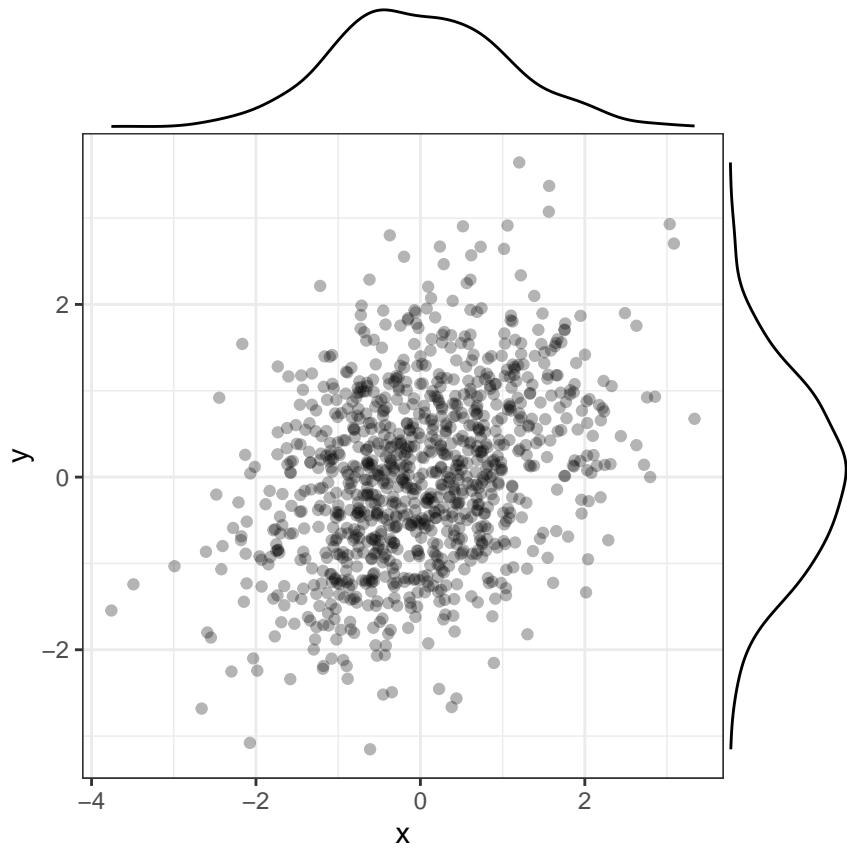


Figure 2.5: Joint and Marginal Distributions

### 2.3.3.1 Conditional Probability

Conditional probability is the probability of an event given some other information. In the real world, you can say that everything is conditional. For example, the probability of getting an odd number on throwing a die is  $1/2$  is conditional on the die being fair. We use  $P(A | B)$  to represent the the conditional probability of event  $A$  given event  $B$ .

Continuing from the previous example,  $P(A | B)$  is the conditional probability of getting an odd number, *knowing that the number is at least 4*. By definition, conditional probability is the probability that both  $A$  and  $B$  happen, divided by the probability that  $B$  happens.

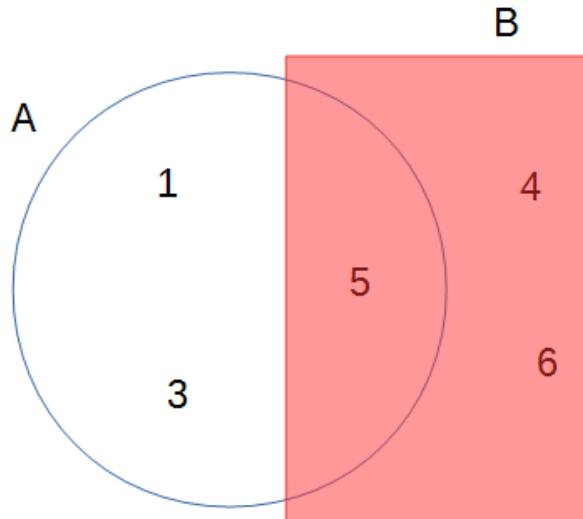
#### ! Conditional Probability

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

In the example,  $P(A, B) = 1 / 6$ , because 5 is the only even number  $\geq 4$  when throwing a die. Thus,

$$\begin{aligned} P(A | B) &= 1/3 \\ &= \frac{P(A, B)}{P(B)} \\ &= \frac{1/6}{1/2} \end{aligned}$$

This picture should make it clear:



! Please recognize that  $P(A | B) \neq P(B | A)$ . For example, when throwing a die,  $P(\text{number is six} | \text{even number}) = 1/3$ , but  $P(\text{even number} | \text{number is six})$  is 1.

### 2.3.3.2 Independence

! Two events,  $A$  and  $B$ , are independent if  $P(A | B) = P(A)$

This means that any knowledge of  $B$  does not (or should not) affect one's belief about  $A$ . Consider the example:

- $A$ : A die shows five or more
- $B$ : A die shows an odd number

Here is the joint probability

	$\geq 5$	$\leq 4$
$\text{odd}$	$1/6$	$2/6$
$\text{even}$	$1/6$	$2/6$

So the conditional probability of  $P(\geq 5 | \text{odd}) = (1/6) / (1/2) = 1/3$ , which is the same as  $P(\geq 5 | \text{even}) = (1/6) / (1/2) = 1/3$ . Similarly it can be verified that  $P(\leq 4 | \text{odd}) = P(\leq 4 | \text{even}) = 2/3$ . Therefore,  $A$  and  $B$  are independent.

On the other hand, for the example

- $A$ : A die shows four or more
- $B$ : A die shows an odd number

the joint probabilities are

	$\geq 4$	$\leq 3$
$\text{odd}$	$1/6$	$2/6$
$\text{even}$	$2/6$	$1/6$

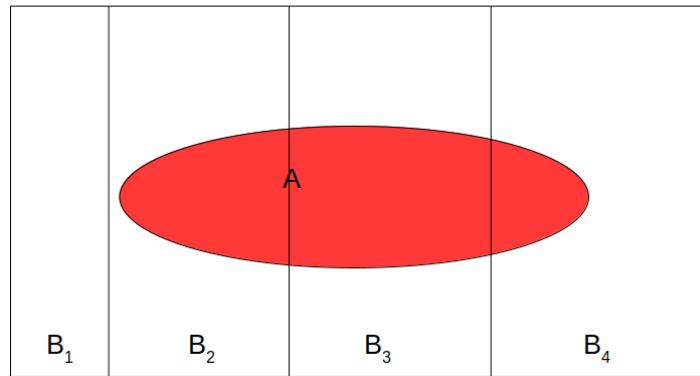
Obviously,  $A$  and  $B$  are not independent because once we know that the number is four or above, the probability of whether it is an odd number or not changes.

! Independence can also be expressed as

If A and B are independent,  $P(A, B) = P(A)P(B)$

### 2.3.4 Law of Total Probability

When we talk about conditional probability, like  $B_1 = 4$  or above and  $B_2 = 3$  or below, we can get  $P(A | B_1)$  and  $P(A | B_2)$  (see the figure below), we refer  $P(A)$  as the *marginal probability*, meaning that the probability of A without knowledge of B.



If  $B_1, B_2, \dots, B_n$  are all mutually exclusive possibilities for an event (so they add up to a probability of 1), then

! Law of Total Probability

$$\begin{aligned} P(A) &= P(A, B_1) + P(A, B_2) + \dots + P(A, B_n) \\ &= P(A | B_1)P(B_1) + P(A | B_2)P(B_2) + \dots + P(A | B_n)P(B_n) \\ &= \sum_{k=1}^n P(A | B_k)P(B_k) \end{aligned}$$

# 3 Bayes's Theorem

The Bayes's theorem is, surprisingly (or unsurprisingly), very simple:

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

More generally, we can expand it to incorporate the law of total probability to make it more applicable to data analysis. Consider  $B_i$  as one of the  $n$  many possible mutually exclusive events, then

$$\begin{aligned} P(B_i | A) &= \frac{P(A | B_i)P(B_i)}{P(A)} \\ &= \frac{P(A | B_i)P(B_i)}{P(A | B_1)P(B_1) + P(A | B_2)P(B_2) + \dots + P(A | B_n)P(B_n)} \\ &= \frac{P(A | B_i)P(B_i)}{\sum_{k=1}^n P(A | B_k)P(B_k)} \end{aligned}$$

If  $B_i$  is a continuous variable, we will replace the sum by an integral,

$$P(B_i | A) = \frac{P(A | B_i)P(B_i)}{\int_k P(A | B_k)P(B_k)}$$

The denominator is not important for practical Bayesian analysis, therefore, it is sufficient to write the above equality as

$$P(B_i | A) \propto P(A | B_i)P(B_i)$$

## 3.1 Example 1: Base rate fallacy (From Wikipedia)

### 3.1.1 Question

A police officer stops a driver *at random* and does a breathalyzer test for the driver. The breathalyzer is known to detect true drunkenness 100% of the time, but in 1% of the cases, it

gives a false positive when the driver is sober. We also know that, in general, for every 1,000 drivers passing through that spot, one is driving drunk. Suppose that the breathalyzer shows positive for the driver. What is the probability that the driver is truly drunk?

### 3.1.2 Solution

$$\begin{aligned}P(\text{positive}|\text{drunk}) &= 1 \\P(\text{positive}|\text{sober}) &= 0.01 \\P(\text{drunk}) &= 1/1000 \\P(\text{sober}) &= 999/1000\end{aligned}$$

Using Bayes' Theorem,

$$\begin{aligned}P(\text{drunk}|\text{positive}) &= \frac{P(\text{positive}|\text{drunk})P(\text{drunk})}{P(\text{positive}|\text{drunk})P(\text{drunk}) + P(\text{positive}|\text{sober})P(\text{sober})} \\&= \frac{1 \times 0.001}{1 \times 0.001 + 0.01 \times 0.999} \\&= 100/1099 \approx 0.091\end{aligned}$$

So there is less than a 10% chance that the driver is drunk even when the breathalyzer shows positive.

You can verify that with a simulation using R:

```
set.seed(4)
truly_drunk <- c(rep("drunk", 100), rep("sober", 100 * 999))
table(truly_drunk)

#> truly_drunk
#> drunk sober
#>    100 99900

breathalyzer_test <- ifelse(truly_drunk == "drunk",
  # If drunk, 100% chance of showing positive
  "positive",
  # If not drunk, 1% chance of showing positive
  sample(c("positive", "negative"), 999000,
        replace = TRUE, prob = c(.01, .99))
)
# Check the probability p(positive | sober)
table(breathalyzer_test[truly_drunk == "sober"])
```

```

#>
#> negative positive
#>    98903      997

# 997 / 99900 = 0.00997998, so the error rate is less than 1%
# Now, Check the probability p(drunk | positive)
table(truly_drunk[breathalyzer_test == "positive"])

#>
#> drunk sober
#>    100    997

# 100 / (100 + 997) = 0.0911577, which is only 9.1%!

```

## 3.2 Bayesian Statistics

**Bayesian statistics** is a way to estimate some parameter  $\theta$  (i.e., some quantities of interest, such as the population mean, regression coefficient, etc) by applying Bayes' Theorem.

$$P(\theta|D) \propto P(D|\theta)P(\theta)$$

There are three components in the above equality:

- $P(D|\theta)$ , the probability that you observe data  $D$ , given the parameter  $\theta$ ; this is called the **likelihood** (Note: It is the likelihood of  $\theta$ , but probability about  $y$ )
- $P(\theta)$ , the probability distribution  $\theta$ , without referring to the data  $D$ . This usually requires appeals to one's degree of belief, and so is called the **prior**
- $P(\theta|y)$ , the updated probability distribution of  $\theta$ , after observing the data  $D$ ; this is called the **posterior**

On the other hand, classical/frequentist statistics focuses solely on the likelihood function.<sup>1</sup> In Bayesian statistics, the goal is to update one's belief about  $\theta$  based on the observed data  $D$ .

---

<sup>1</sup>The likelihood function in classical/frequentist statistics is usually written as  $P(y; \theta)$ . You will notice that here, I write the likelihood for classical/frequentist statistics to be different from the one used in Bayesian statistics. This is intentional: In frequentist conceptualization,  $\theta$  is fixed, and it does not make sense to talk about the probability of  $\theta$ . This implies that we cannot condition on  $\theta$ , because conditional probability is defined only when  $P(\theta)$  is defined.

### 3.3 Example 2: Locating a Plane

Consider a highly simplified scenario of locating a missing plane in the sea. Assume that we know the plane, before missing, happened to be flying at the same latitude, heading west across the Pacific, so we only need to find its longitude. We want to go out to collect debris (data) so that we can narrow the location ( $\theta$ ) of the plane down.

#### 3.3.1 Prior

We start with our prior. Assume that we have some rough idea where the plane should be, so we express our belief in a probability distribution like the following:

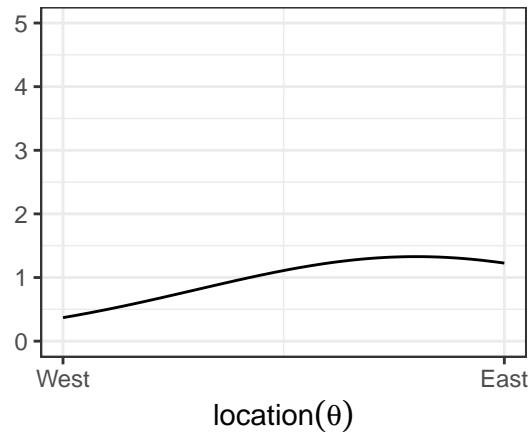


Figure 3.1: Prior distribution.

which says that our belief is that the plane is about twice more likely to be towards the east than towards the west. Below are two other options for priors (out of infinitely many), one providing virtually no information and the other encoding stronger information:

The prior is chosen to reflect the researcher's belief, so different researchers will likely formulate a different prior for the same problem, and that's okay as long as the prior is reasonable and justified. Later, we will learn that in regular Bayesian analyses, with a moderate sample size, different priors generally only make negligible differences.

#### 3.3.2 Likelihood

Now, assume that we have collected debris in the locations shown in the graph,

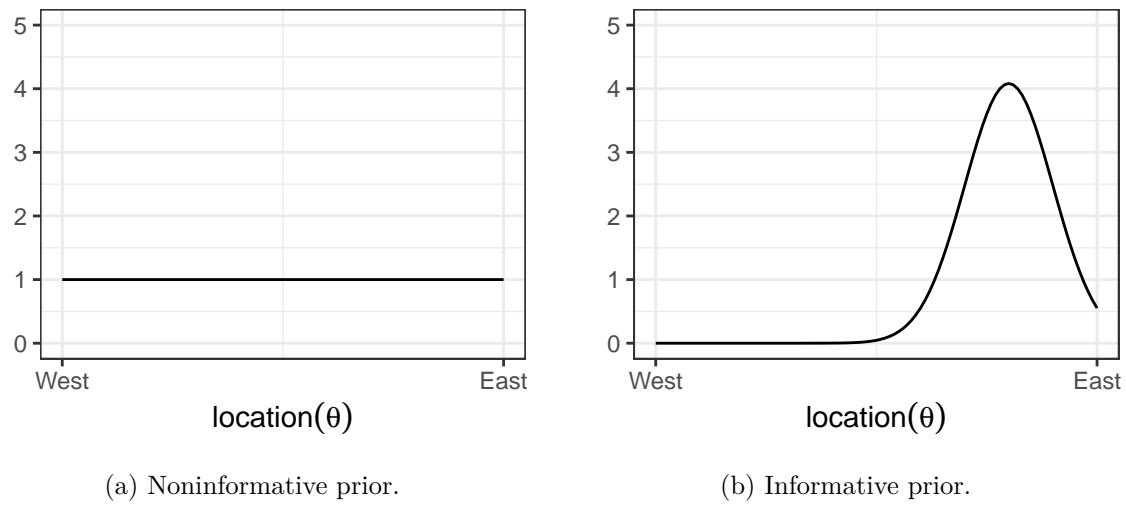


Figure 3.2: More options for prior distribution.

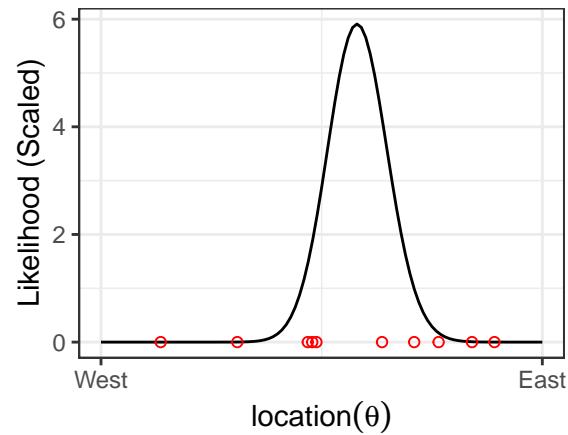


Figure 3.3

### 3.3.3 Posterior

Now, from Bayes's Theorem,

$$\text{Posterior Probability} \propto \text{Prior Probability} \times \text{Likelihood}$$

So we can simply multiply the prior probabilities and the likelihood to get the posterior probability for every location. A rescaling step is needed to ensure that the area under the curve will be 1, which is usually performed by the software.

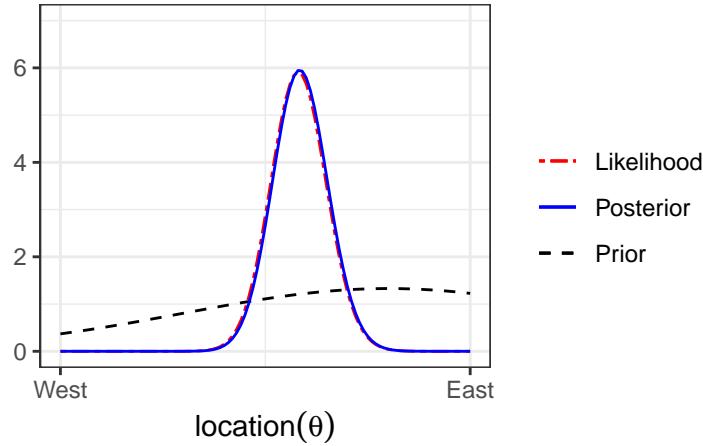


Figure 3.4

As illustrated below, the posterior distribution is a synthesis of (a) the prior and (b) the data (likelihood).

### 3.3.4 Influence of Prior

Figure 3.5 shows what happen with a stronger prior:

### 3.3.5 Influence of More Data

Figure 3.6 shows what happen with 20 more data points:

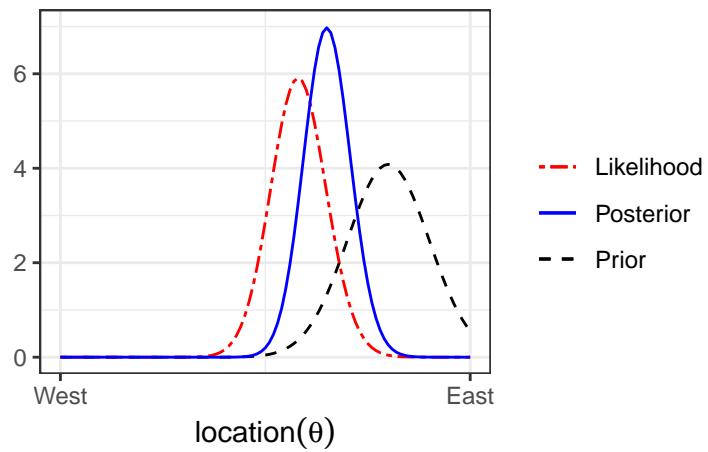


Figure 3.5

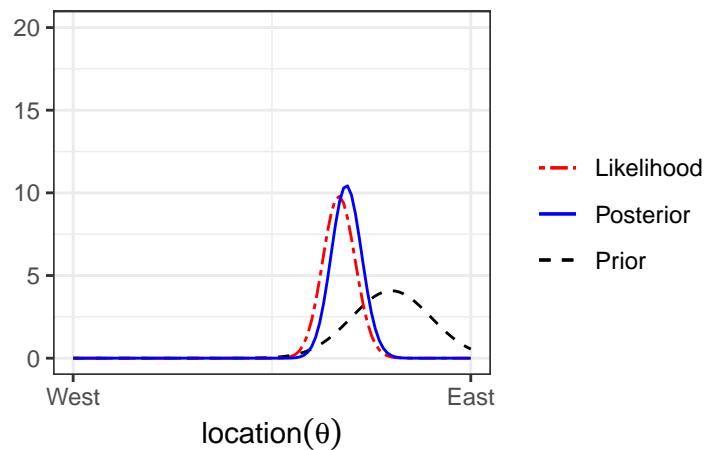


Figure 3.6

## 3.4 Data-Order Invariance

In many data analysis applications, researchers collect some data  $D_1$ , and then collect some more data  $D_2$ . An example would be researchers conducting two separate experiments to study the same research question. In Bayesian statistics, one can consider three ways to obtain the posterior:

1. Update the belief with  $D_1$ , and then with  $D_2$
2. Update the belief with  $D_2$ , and then with  $D_1$
3. Update the belief with both  $D_1$  and  $D_2$  simultaneously

Whether these three ways give the same posterior depends on whether **data-order invariance** holds. If the inference of  $D_1$  does not depend on  $D_2$ , or vice versa, then all three ways lead to the same posterior. Specifically, if we have **conditional independence** such that

$$P(D_1, D_2 | \theta) = P(D_1 | \theta)P(D_2 | \theta),$$

then one can show all three ways give the same posterior (see section 4.4 and 4.5 of Johnson et al., 2022).

### 💡 Exchangeability\*

Exchangeability is an important concept in Bayesian statistics. Data are exchangeable when the joint distribution,  $P(D_1, \dots, D_N)$ , does not depend on the ordering of the data. A simple way to think about it is if you scramble the order of your outcome variable in your data set and still can obtain the same statistical results, then the data are exchangeable. An example situation where data are not exchangeable is

- $D_1$  is from year 1990,  $D_2$  is from year 2020, and the parameter  $\theta$  changes from 1990 to 2020

When data are exchangeable, conditional independence would generally hold.<sup>2</sup>

## 3.5 Bernoulli Likelihood

For binary data  $y$  (e.g., coin flip, pass/fail, diagnosed/not), an intuitive way to analyze is to use a Bernoulli model:

$$\begin{aligned}P(y = 1 | \theta) &= \theta \\P(y = 0 | \theta) &= 1 - \theta\end{aligned}$$

<sup>2</sup>The [de Finetti's theorem](#) shows that when the data are exchangeable and can be considered an infinite sequence (i.e., not from a tiny finite population), then the data are conditionally independent given some  $\theta$ .

which is more compactly written as

$$P(y \mid \theta) = \theta^y(1 - \theta)^{(1-y)},$$

where  $\theta \in [0, 1]$  is the probability of a “1”. You can verify that the compact form is the same as the longer form.

### 3.5.1 Multiple Observations

When there are more than one  $y$ , say  $y_1, \dots, y_N$ , that are conditionally independent, we have

$$\begin{aligned} P(y_1, \dots, y_N \mid \theta) &= \prod_{i=1}^N P(y_i \mid \theta) \\ &= \theta^{\sum_{i=1}^N y_i} (1 - \theta)^{\sum_{i=1}^N (1-y_i)} \\ &= \theta^z (1 - \theta)^{N-z} \end{aligned}$$

where  $z$  is the number of “1”s (e.g., the number of heads in coin flips). Note that the likelihood only depends on  $z$ , not the individual  $y$ s. In other words, the likelihood is the same as long as there are  $z$  heads, regardless of when those heads occur.

Let’s say  $N = 4$  and  $z = 1$ . We can plot the likelihood in R:

```
# Write the likelihood as a function of theta
lik <- function(th, num_flips = 4, num_heads = 1) {
  th ^ num_heads * (1 - th) ^ (num_flips - num_heads)
}
# Likelihood of theta = 0.5
lik(0.5)
```

```
#> [1] 0.0625
```

```
# Plot the likelihood
ggplot(data.frame(th = c(0, 1)), aes(x = th)) +
  # `stat_function` for plotting a function
  stat_function(fun = lik) +
  # use `expression()` to get greek letters
  labs(x = expression(theta),
       y = "Likelihood with N = 4 and z = 1")
```

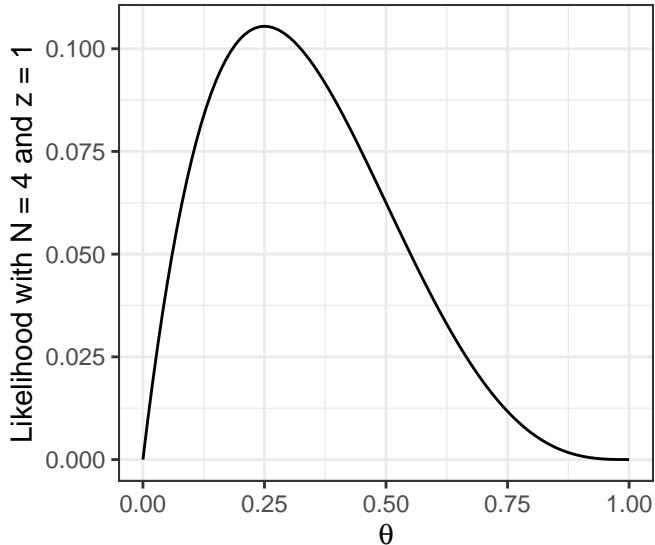


Figure 3.7: Binomial likelihood function with  $N = 4$  and  $z = 1$

### 3.5.2 Setting Priors

Remember again the relationship between the prior and the posterior:

$$P(\theta|y) \propto P(y|\theta)P(\theta)$$

The posterior distributions are mathematically determined once the priors and the likelihood are set. However, the mathematical form of the posterior is sometimes very difficult to deal with.

One straightforward, brute-force method is to discretize the parameter space into a number of points. For example, by taking  $\theta = 0, 0.05, 0.10, \dots, 0.90, 0.95, 1.00$ , one can evaluate the posterior at these 21 **grid points**.

Let's use a prior that peaks at 0.5 and linearly decreases to both sides. I assume that  $\theta = 0.5$  is twice as likely as  $\theta = 0.25$  or  $\theta = 0.75$  to reflect my belief that the coin is more likely to be fair.

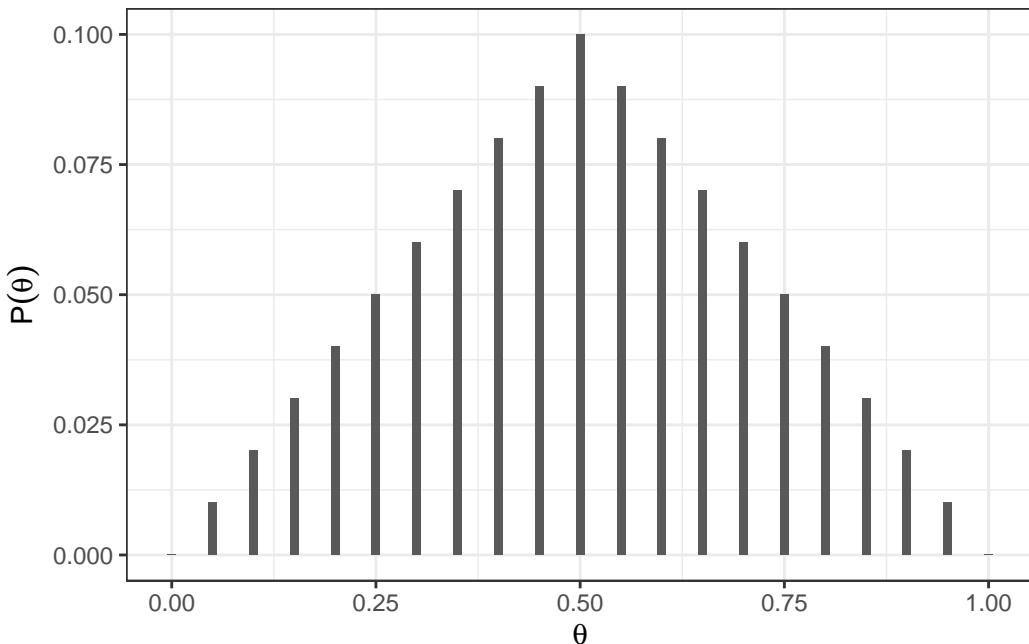
```
# Define a grid for the parameter
grid_df <- data.frame(th = seq(0, 1, by = 0.05))
# Set the prior mass for each value on the grid
grid_df$pth <- c(0:10, 9:0) # linearly increasing, then decreasing
# Convert pth to a proper distribution such that the value
# sum to one
```

```

grid_df$pth <- grid_df$pth / sum(grid_df$pth)
# Plot the prior
ggplot(grid_df, aes(x = th, y = pth)) +
  geom_col(aes(x = th, y = pth),
           width = 0.01,
  ) +
  labs(y = expression(P(theta)), x = expression(theta))

```

- ① This line ensures that the probability values sum to one. This is a trick we will use to obtain the posterior probability.



### Prior Predictive Distribution

One way to check whether the prior is appropriate is to use the **prior predictive distribution**. Bayesian models are **generative** because they can be used to simulate data. The prior predictive distribution can be obtained by first simulating some  $\theta$  values from the prior distribution and then simulating a data set for each  $\theta$ .

```

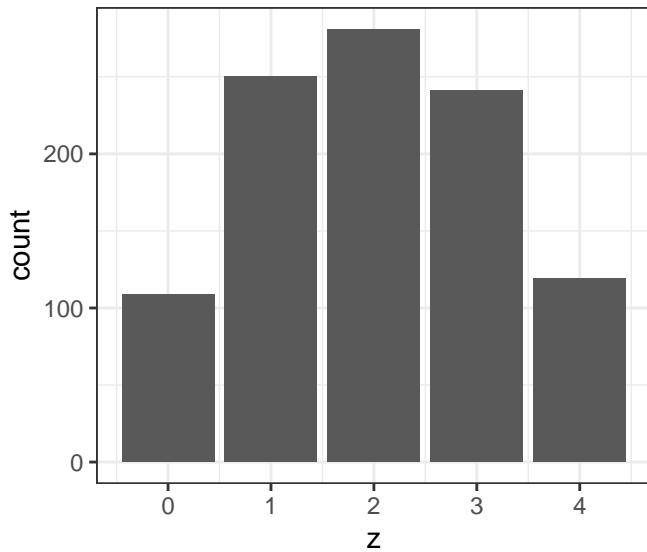
# Draw one theta
num_trials <- 4 # number of draws
sim_th1 <- sample(grid_df$th, size = 1,
                  # based on prior probability
                  prob = grid_df$pth)
# Simulate new data of four flips based on model
sim_y1 <- rbinom(num_trials, size = 1, prob = sim_th1)

# Repeat many times
# Set number of simulation draws
num_draws <- 1000
sim_th <- sample(grid_df$th, size = num_draws, replace = TRUE,
                  # based on prior probability
                  prob = grid_df$pth)
# Use a for loop
# Initialize output
sim_y <- matrix(NA, nrow = num_trials, ncol = num_draws)
for (s in seq_len(num_draws)) {
  # Store simulated data in the s-th column
  sim_y[, s] <- rbinom(num_trials, size = 1, prob = sim_th[s])
}
# Show the first 10 simulated data sets based on prior:
sim_y[, 1:10]

#>      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
#> [1,]     1    0    0    1    0    1    0    0    0    1
#> [2,]     0    0    0    1    1    0    1    0    1    1
#> [3,]     1    0    0    1    0    0    0    1    1    1
#> [4,]     1    0    1    1    1    0    0    1    1    0

# Show the distribution of number of heads
sim_heads <- colSums(sim_y)
ggplot(data.frame(z = sim_heads), aes(x = z)) +
  geom_bar()

```



The outcome seems to fit our intuition that it's more likely to be half heads and half tails, but there is a lot of uncertainty.

### 3.5.3 Summarizing the Posterior

```
grid_df <- grid_df |>
  mutate(
    # Use our previously defined lik() function
    py_th = lik(th, num_flips = 4, num_heads = 1),
    # Product of prior and likelihood
    `prior x lik` = pth * py_th,
    # Scaled the posterior
    pth_y = `prior x lik` / sum(`prior x lik`)
  )
# Print a table
knitr::kable(grid_df)
```

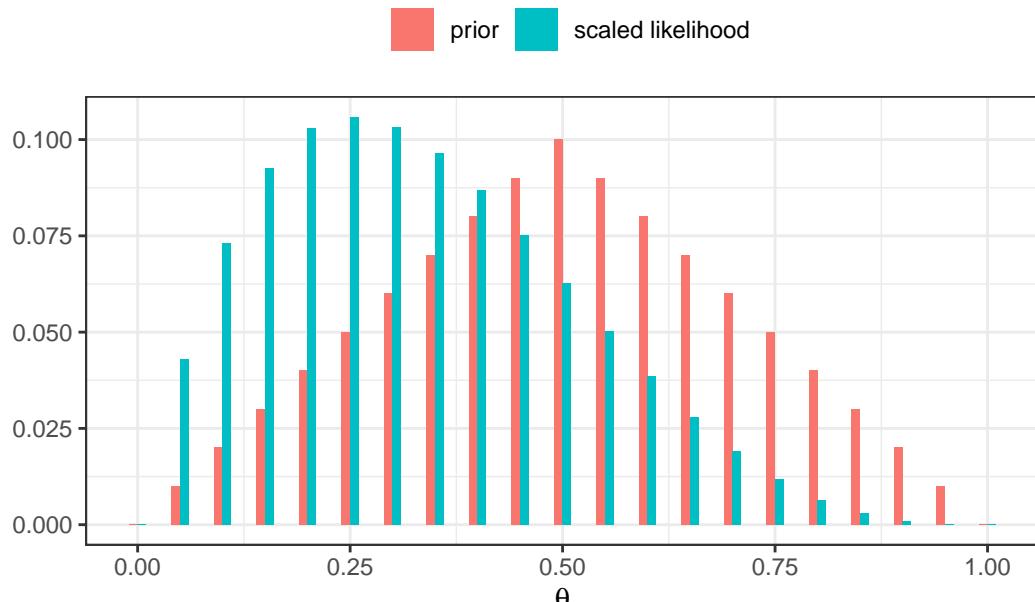
th	pth	py_th	prior x lik	pth_y
0.00	0.00	0.0000000	0.0000000	0.0000000
0.05	0.01	0.0428687	0.0004287	0.0073359
0.10	0.02	0.0729000	0.0014580	0.0249500
0.15	0.03	0.0921188	0.0027636	0.0472914

th	pth	py_th	prior x lik	pth_y
0.20	0.04	0.1024000	0.0040960	0.0700927
0.25	0.05	0.1054688	0.0052734	0.0902416
0.30	0.06	0.1029000	0.0061740	0.1056525
0.35	0.07	0.0961187	0.0067283	0.1151381
0.40	0.08	0.0864000	0.0069120	0.1182815
0.45	0.09	0.0748688	0.0067382	0.1153071
0.50	0.10	0.0625000	0.0062500	0.1069530
0.55	0.09	0.0501187	0.0045107	0.0771891
0.60	0.08	0.0384000	0.0030720	0.0525695
0.65	0.07	0.0278687	0.0019508	0.0333832
0.70	0.06	0.0189000	0.0011340	0.0194056
0.75	0.05	0.0117188	0.0005859	0.0100268
0.80	0.04	0.0064000	0.0002560	0.0043808
0.85	0.03	0.0028687	0.0000861	0.0014727
0.90	0.02	0.0009000	0.0000180	0.0003080
0.95	0.01	0.0001187	0.0000012	0.0000203
1.00	0.00	0.0000000	0.0000000	0.0000000

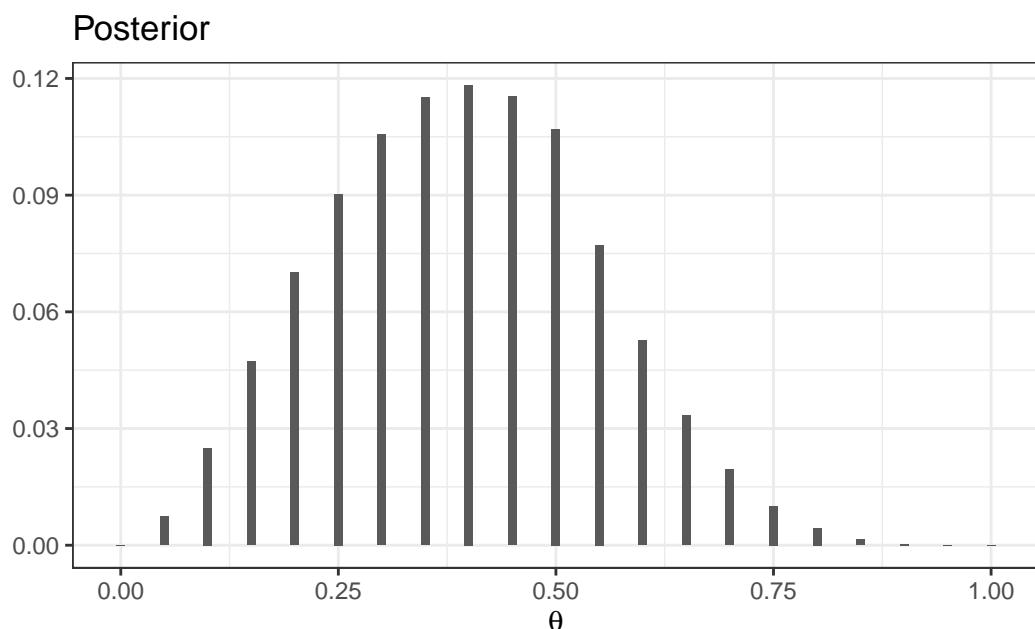
```
# Plot the prior/likelihood and the posterior
ggplot(data = grid_df, aes(x = th)) +
  geom_col(aes(x = th - 0.005, y = pth, fill = "prior"),
            width = 0.01,
  ) +
  geom_col(aes(x = th + 0.005, y = py_th / sum(py_th),
            fill = "scaled likelihood"), width = 0.01,
  ) +
  labs(fill = NULL, y = NULL, x = expression(theta)) +
  theme(legend.position = "top")
ggplot(data = grid_df, aes(x = th)) +
  geom_col(aes(x = th, y = pth_y), width = 0.01) +
  labs(
    fill = NULL, y = NULL, title = "Posterior",
    x = expression(theta)
  )
```

Figure 3.8b shows the posterior distribution, which represents our updated belief about  $\theta$ . We can summarize it by simulating  $\theta$  values from it and compute summary statistics:

```
# Define a function for computing posterior summary
summ_draw <- function(x) {
```



(a) Prior and likelihood



(b) Posterior

Figure 3.8: Bernoulli posterior distribution

```

c(
  mean = mean(x),
  median = median(x),
  sd = sd(x),
  mad = mad(x),
  `ci.1` = quantile(x, prob = .1, names = FALSE),
  `ci.9` = quantile(x, prob = .9, names = FALSE)
)
}

# Sample from the posterior
post_samples <- sample(
  grid_df$th,
  size = 1000, replace = TRUE,
  prob = grid_df$pth_y
)
summ_draw(post_samples)

#>      mean     median       sd       mad      ci.1      ci.9
#> 0.3848000 0.4000000 0.1538429 0.1482600 0.2000000 0.6000000

# Alternatively, use the `posterior` package
data.frame(theta = post_samples) |>
  posterior::summarize_draws()

#> # A tibble: 1 x 10
#>   variable  mean median     sd     mad     q5    q95  rhat ess_bulk ess_tail
#>   <chr>    <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl> <dbl>    <dbl>    <dbl>
#> 1 theta     0.385    0.4  0.154  0.148   0.15   0.65  1.00    1030.    721.

```

### 3.5.4 Influence of Sample Size

If, instead, we have  $N = 40$  and  $z = 10$ , the posterior will be more similar to the likelihood.

```

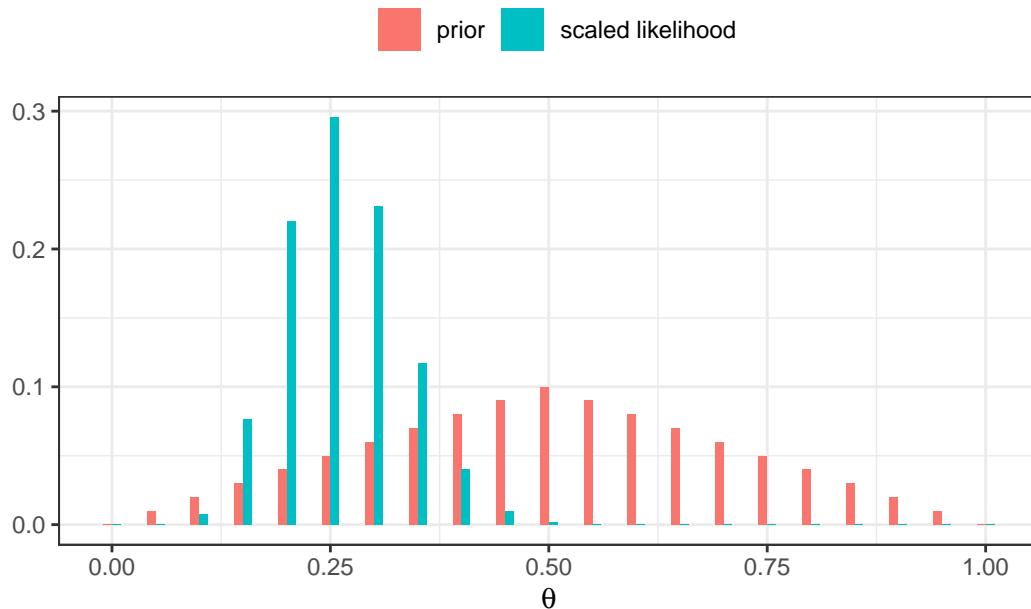
grid_df2 <- grid_df |>
  mutate(
    # Use our previously defined lik() function
    py_th = lik(th, num_flips = 40, num_heads = 10),
    # Product of prior and likelihood
    `prior x lik` = pth * py_th,
    # Scaled the posterior
  )

```

```

    pth_y = `prior x lik` / sum(`prior x lik`)
  )
# Plot the prior/likelihood and the posterior
ggplot(data = grid_df2, aes(x = th)) +
  geom_col(aes(x = th - 0.005, y = pth, fill = "prior"),
            width = 0.01,
  ) +
  geom_col(aes(x = th + 0.005, y = py_th / sum(py_th),
                fill = "scaled likelihood"), width = 0.01,
  ) +
  labs(fill = NULL, y = NULL, x = expression(theta)) +
  theme(legend.position = "top")

```

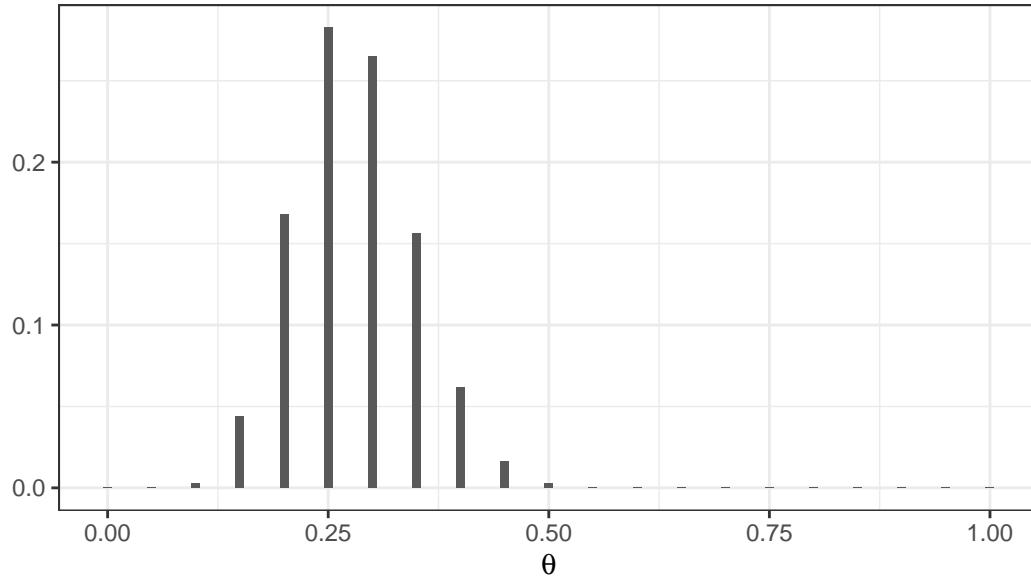


```

ggplot(data = grid_df2, aes(x = th)) +
  geom_col(aes(x = th, y = pth_y), width = 0.01) +
  labs(
    fill = NULL, y = NULL, title = "Posterior",
    x = expression(theta)
  )

```

## Posterior



```
# Sample from the posterior
post_samples <- sample(
  grid_df2$th,
  size = 1000, replace = TRUE,
  prob = grid_df2$pth_y
)
summ_draw(post_samples)
```

```
#>      mean     median       sd      mad    ci.1    ci.9
#> 0.28085000 0.30000000 0.06542215 0.07413000 0.20000000 0.35000000
```

### 3.5.5 Influence of Prior

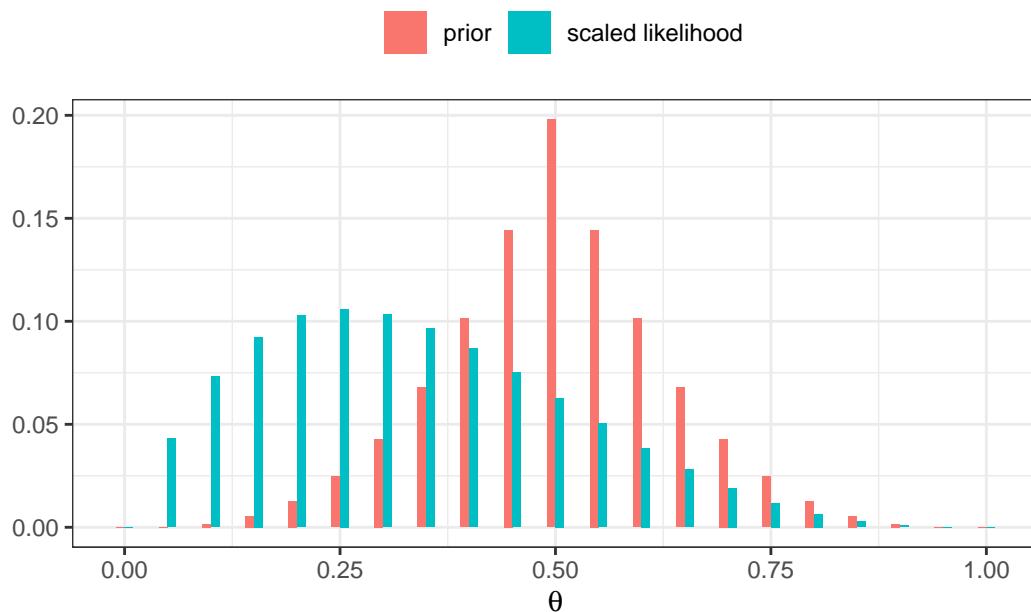
If we have a very strong prior concentrated at  $\theta = .5$ , but still with  $N = 40$  and  $z = 10$ , the posterior will be more similar to the prior.

```
grid_df3 <- grid_df |>
  mutate(
    # stronger prior
    pth = pth ^ 3,
    # scale the prior to sume to 1
    pth = pth / sum(pth),
```

```

# Use our previously defined lik() function
py_th = lik(th, num_flips = 4, num_heads = 1),
# Product of prior and likelihood
`prior x lik` = pth * py_th,
# Scaled the posterior
pth_y = `prior x lik` / sum(`prior x lik`)
)
# Plot the prior/likelihood and the posterior
ggplot(data = grid_df3, aes(x = th)) +
  geom_col(aes(x = th - 0.005, y = pth, fill = "prior"),
            width = 0.01,
  ) +
  geom_col(aes(x = th + 0.005, y = py_th / sum(py_th),
                fill = "scaled likelihood"), width = 0.01,
  ) +
  labs(fill = NULL, y = NULL, x = expression(theta)) +
  theme(legend.position = "top")

```



```

ggplot(data = grid_df3, aes(x = th)) +
  geom_col(aes(x = th, y = pth_y), width = 0.01) +
  labs(
    fill = NULL, y = NULL, title = "Posterior",

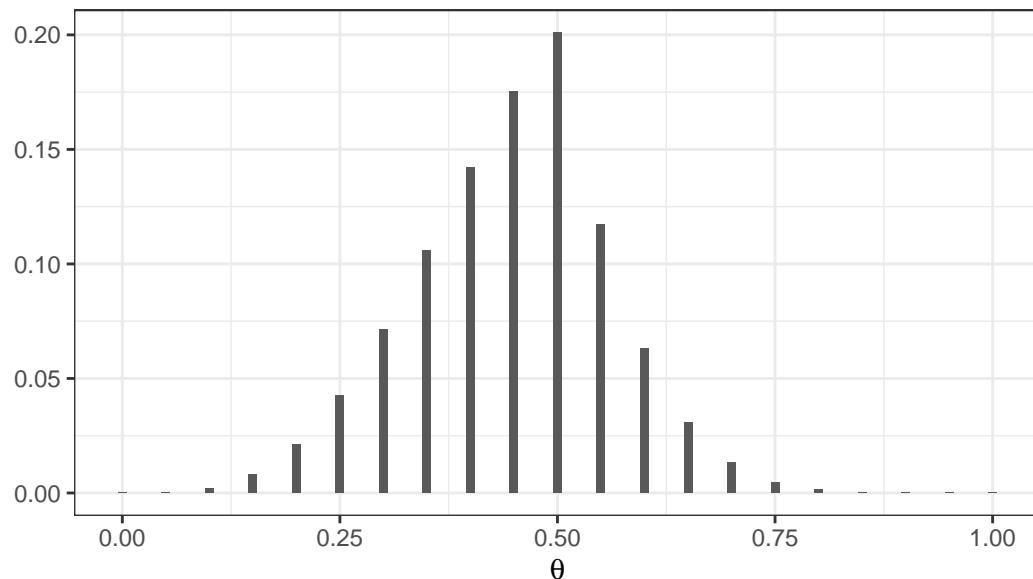
```

```

    x = expression(theta)
)

```

Posterior



```

# Sample from the posterior
post_samples <- sample(
  grid_df3$th,
  size = 1000, replace = TRUE,
  prob = grid_df3$pth_y
)
summ_draw(post_samples)

```

```

#>      mean     median       sd       mad      ci.1      ci.9
#> 0.4493000 0.4500000 0.1096656 0.0741300 0.3000000 0.6000000

```

```

# Alternatively, use the `posterior` package
data.frame(theta = post_samples) |>
  posterior::summarize_draws()

```

```

#> # A tibble: 1 x 10
#>   variable  mean median     sd     mad    q5    q95  rhat ess_bulk ess_tail
#>   <chr>    <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>    <dbl>    <dbl>
#> 1 theta     0.449    0.45  0.110  0.0741  0.25    0.6   1.00   1001.    899.

```

### 3.5.6 Remark on Grid Approximation

In this note, we discretized  $\theta$  into a finite number of grid points to compute the posterior, mainly for pedagogical purposes. A big limitation is that our posterior will have no density for values other than the chosen grid points. While increasing the number of grid points (e.g., 1,000) can give more precision, the result is still not truly continuous. A bigger issue is that the computation breaks down when there is more than one parameter; if there are  $p$  parameters, with 1,000 grid points per parameter, one needs to evaluate the posterior probability for  $1,000^p$  grid points, which is not feasible even with modern computers. So more efficient algorithms, namely Markov chain Monte Carlo (MCMC) methods, will be introduced as we progress in the course.

## **Part III**

### **Week 3**

# 4 Beta-Bernoulli Model

## 4.1 Steps of Bayesian Data Analysis

Some authors described the process as “turning the Bayesian Crank,” as the same workflow applies to a variety of research scenarios.

Adapted from [Gelman et al. \(2020\)](#), I conceptualize Bayesian data analysis as the following steps:

1. **Identify/Collect the data** required to answer the research questions.
  - As a general recommendation, it is helpful to **visualize** the data to get a sense of how they look, as well as to inspect for potential anomalies in the data collection.
2. **Choose an initial statistical model** for the data in relation to the research questions. The model should have some theoretical justification and have parameters that are meaningful for the research questions. However, it is unlikely that any chosen model will capture everything important in the data, and this initial model will be modified and expanded in later steps.
3. **Specify prior distributions** for the model parameters. Although this is a subjective endeavor, the priors chosen should be sensible to a skeptical audience.
4. **Check the prior distributions.** It is recommended you conduct a **prior predictive check**, by simulating fake data based on the chosen model and prior distributions. This is especially important for complex models as the parameters are more difficult to interpret.
5. **Obtain the posterior distributions** for the model parameters. As described below and later in the course, this can be obtained by analytical or various mathematical approximations.
  - For mathematical approximations, one should check the algorithms for **convergence** to make sure the results closely mimic the target posterior distributions.
6. Conduct a **posterior predictive check** to examine the fit between the model and the data, i.e., whether the chosen model with the estimated parameters generates predictions that deviate from the data being analyzed on important features.
7. It is unlikely that your initial model fully describes the major aspects of the data as pertaining to your research questions. Therefore, one should repeat steps 2 to 6 to **specify and compare different models**.

- If the fit between the model and the data is deemed satisfactory, one can proceed to **interpret the results** in the context of the research questions. It is also important to **visualize the results** in ways that are meaningful for the analysis.

## 4.2 Beta-Bernoulli Example

We will be using a built-in data set in R about patients diagnosed with AIDS in Australia before July 1, 1991. Here is a description of the variables (from the R documentation):

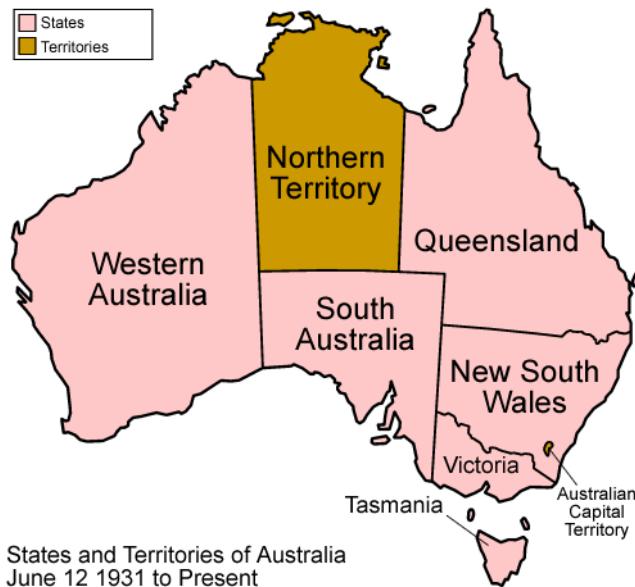


Figure 4.1: Figure from [https://commons.wikimedia.org/wiki/File:Australia\\_states\\_1931-present.png](https://commons.wikimedia.org/wiki/File:Australia_states_1931-present.png)

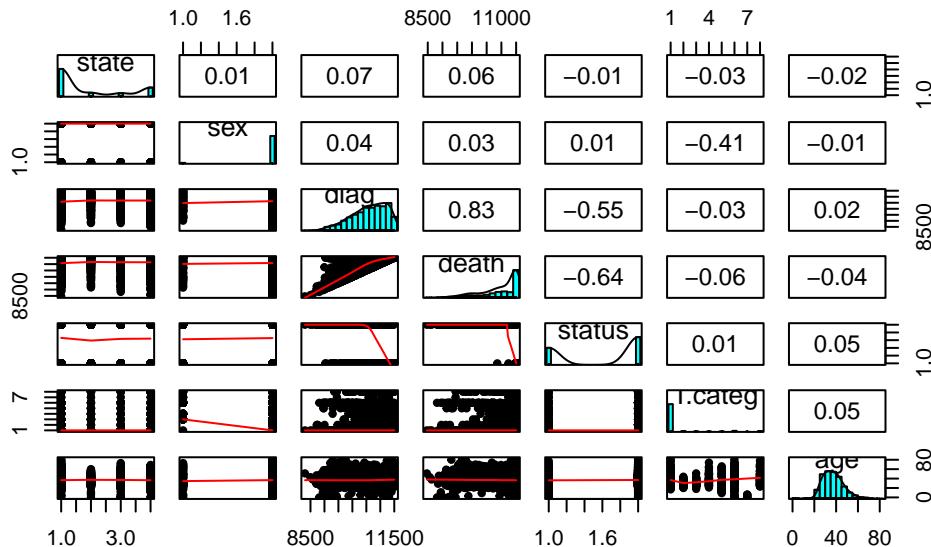
- state:** Grouped state of origin: “NSW” includes ACT and “other” is WA, SA, NT, and TAS.
- sex:** Sex of patient.
- diag:** (Julian) date of diagnosis.
- death:** (Julian) date of death or end of observation.
- status:** “A” (alive) or “D” (dead) at end of observation.
- T.categ:** Reported transmission category.
- age:** Age (years) at diagnosis.

You should always first plot your data and get some summary statistics:

```
data("Aids2", package = "MASS")
head(Aids2)
```

	state	sex	diag	death	status	T.categ	age
1	NSW	M	10905	11081	D	hs	35
2	NSW	M	11029	11096	D	hs	53
3	NSW	M	9551	9983	D	hs	42
4	NSW	M	9577	9654	D	haem	44
5	NSW	M	10015	10290	D	hs	39
6	NSW	M	9971	10344	D	hs	36

```
pairs.panels(Aids2, ellipses = FALSE)
```



We will be using the **status** variable. Our simple research question is:

What was the death rate of AIDS in Australia when the data were collected?

#### 4.2.1 Bernoulli Model

If we assume that the outcomes of the observations are exchangeable, meaning that the observations can be reordered in any way and still give the same inference, then one can choose a model:

$$y_i \sim \text{Bern}(\theta) \text{ for } i = 1, 2, \dots, N$$

- $y_i$  = status of observation  $i$  ( $0 = \text{"A"}$ ,  $1 = \text{"D"}$ )
- $N$  = number of patients in the data set
- $\theta$  = probability of “D”<sup>1</sup>

The model states that: the sample data  $y$  follows a Bernoulli distribution with  $n$  with a parameter  $\theta$

- 💡 When the data consist of binary observations, the variable is called a Bernoulli variable. It is conventional to denote one outcome as success and code it as 1, and the other as failure and code it as 0 (poor terminology, maybe, but that's by convention). Therefore, in the AIDS example, each observation is considered a “Bernoulli” outcome (Alive vs. Dead).

#### 4.2.2 Exchangeability

To illustrate exchangeability in an example, say we take 6 rows in our data set:

	state	sex	diag	death	status	T.categ	age
1	NSW	M	10905	11081	D	hs	35
31	NSW	F	10961	11504	A	id	30
1802	QLD	F	9495	9753	D	blood	66
1811	QLD	M	10770	11504	A	hsid	29
2129	VIC	M	8499	8568	D	hs	43
2137	VIC	M	9055	9394	D	hs	29

Now, when we reorder the column **status** to something like:

	state	sex	diag	death	status	T.categ	age
1	NSW	M	10905	11081	D	hs	35
31	NSW	F	10961	11504	D	id	30
1802	QLD	F	9495	9753	D	blood	66
1811	QLD	M	10770	11504	A	hsid	29
2129	VIC	M	8499	8568	A	hs	43
2137	VIC	M	9055	9394	D	hs	29

---

<sup>1</sup>An additional thing to note for the Bernoulli/binomial model is that, instead of setting the prior on  $\theta$ , sometimes we are more interested in setting the prior for a transformed parameter that has values between  $-\infty$  and  $\infty$ , such as one on the *logit* scale (as related to logistic regression).

If the results are expected to be the same, then we say that the observations are assumed exchangeable. It happens when we assume that all observations have one common mean. However, if we think that there is a mean for females and a different mean for males, we cannot reorder the outcome randomly because they are no longer exchangeable (i.e., you cannot exchange a female score for a male score and expect to get the same results).

### ! Exchangeability

A set of observations is said to be exchangeable if their joint probability distribution stays the same under all permutations. Roughly speaking, it means that the observations can be reordered and still provide the same inferences.

#### 4.2.3 Check the Support

It is important to identify the *support* of the parameter,  $\theta$ . Because  $\theta$  is a probability, its support is  $[0, 1]$ , meaning it is continuous and can take any value from 0 to 1. For a continuous parameter, there are infinitely many possible values, and it is impossible to specify our beliefs for each value. So, more commonly, we choose a probability density function with the same support as the parameter to express our prior belief.

#### 4.2.4 Conjugate Prior: Beta Distribution

A commonly used family of prior distributions for a Bernoulli/binomial model is the *Beta distribution*, which has two parameters. We can write the prior as

$$P(\theta) \sim \text{Beta}(a, b)$$

$a$  and  $b$  are the two hyperparameters. Here are a few examples:

You will notice that when  $a > b$ , there is more density closer to the right region (i.e., larger  $\theta$ ), and vice versa. Also, the variance decreases when  $a$  and  $b$  become larger.<sup>2</sup>

A nice interpretation of  $a$  and  $b$  in a Beta prior distribution is to consider

- $a - 1$  = number of prior ‘successes’ (e.g., “D”)
- $b - 1$  = number of prior ‘failures’ (e.g., “A”)

<sup>2</sup>The Beta( $1/2, 1/2$ ) distribution is called a *Jeffreys prior* ([https://en.wikipedia.org/wiki/Jeffreys\\_prior](https://en.wikipedia.org/wiki/Jeffreys_prior)), which is derived according to some statistical principles for different models. One big advantage of a Jeffreys prior is that it is **invariant**, meaning that the prior will stay the same even under reparameterization. However, like conjugate priors, Jeffreys prior limits the choice of prior even when true prior information is available.

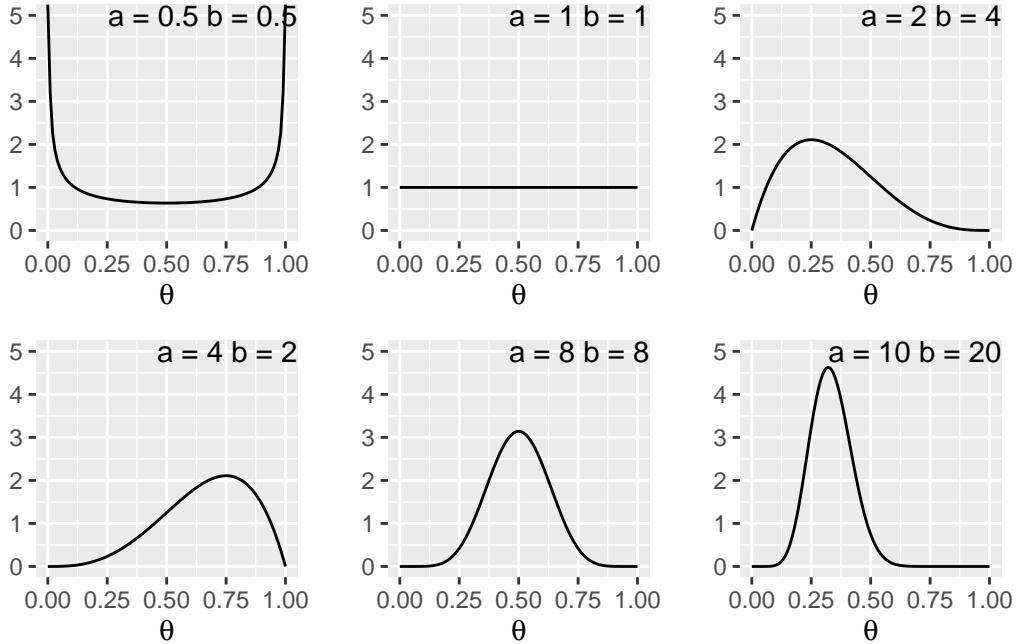


Figure 4.2: Beta distributions with different  $a$  and  $b$  values

Therefore, with  $\text{Beta}(1, 1)$ , one has seen 0 prior success and 0 failure, meaning that there is no prior information (i.e., *noninformative*). Therefore, it makes sense that all  $\theta$  values are equally likely. On the other hand, if one chooses  $\text{Beta}(10, 20)$ , one has seen 9 prior successes and 19 prior failures, so one has quite a lot of prior information (indeed more than the data with only 10 observations), so this is a *strong* prior.

The smaller the variance of the prior distribution, the stronger one's belief before looking at the data, the more prior information

So by manipulating the values of  $a$  and  $b$ , which are sometimes called *hyperparameters*, you can control the shape of the prior distribution as well as its strength, so it is quite flexible. Another advantage of using a beta prior is that it is a *conjugate prior* of the Bernoulli model, which means that the posterior distribution  $P(\theta | y)$  is also a beta distribution, the same as the prior distribution, although with different parameter values.

### ! Conjugate Prior

For a specific model, conjugate priors yield posterior distributions in the same distribution family as the priors

Conjugacy greatly simplifies the computational burden for Bayesian analyses, so conjugate priors are almost the only ones used in earlier literature. However, this limited the applica-

tions of Bayesian methods, as for many problems, no conjugate priors can provide a realistic representation of one's belief. Modern Bayesian analysis instead relies on *simulation-based* methods to approximate the posterior distribution, which can accommodate almost any kind of prior distribution. Aside from a few examples in this note, mainly for pedagogical purposes, we will be using simulation-based methods in the coming weeks.

### 💡 Proof of Conjugacy\*

To derive the form of the posterior, first recognize that the Beta distribution has the form:

$$\begin{aligned} P(\theta) &= B^{-1}(a, b)\theta^{a-1}(1-\theta)^{b-1} \\ &\propto \theta^{a-1}(1-\theta)^{b-1} \end{aligned}$$

Where  $B(\cdot)$  is the beta function which is not very important for the class. As the density function is a function of  $\theta$ , it suffices to write only the terms that involve  $\theta$ . Similarly,

$$P(\mathbf{y} | \theta) \propto \theta^z(1-\theta)^{N-z}.$$

Therefore,

$$\begin{aligned} P(\theta | \mathbf{y}) &\propto P(\mathbf{y} | \theta)P(\theta) \\ &\propto \theta^z(1-\theta)^{N-z}\theta^{a-1}(1-\theta)^{b-1} \\ &= \theta^{a+z-1}(1-\theta)^{b+N-z-1}. \end{aligned}$$

If we let  $a^* = a + z$ ,  $b^* = b + N - z$ , we can see that  $P(\theta | \mathbf{y})$  is in the same form as the prior with  $a$  and  $b$  replaced by  $a^*$  and  $b^*$ . Therefore, the posterior is also a beta distribution. So the beta distribution is a conjugate prior for the Bernoulli model.

In this example, we will choose a *weakly informative* Beta(2, 2) prior, which represents a weak belief as below:

```
ggplot(data.frame(th = c(0, 1)), aes(x = th)) +
  stat_function(fun = dbeta, args = list(shape1 = 2, shape2 = 2)) +
  ylim(0, 3) +
  labs(y = "", x = expression(theta), title = "Beta(2, 2)")
```

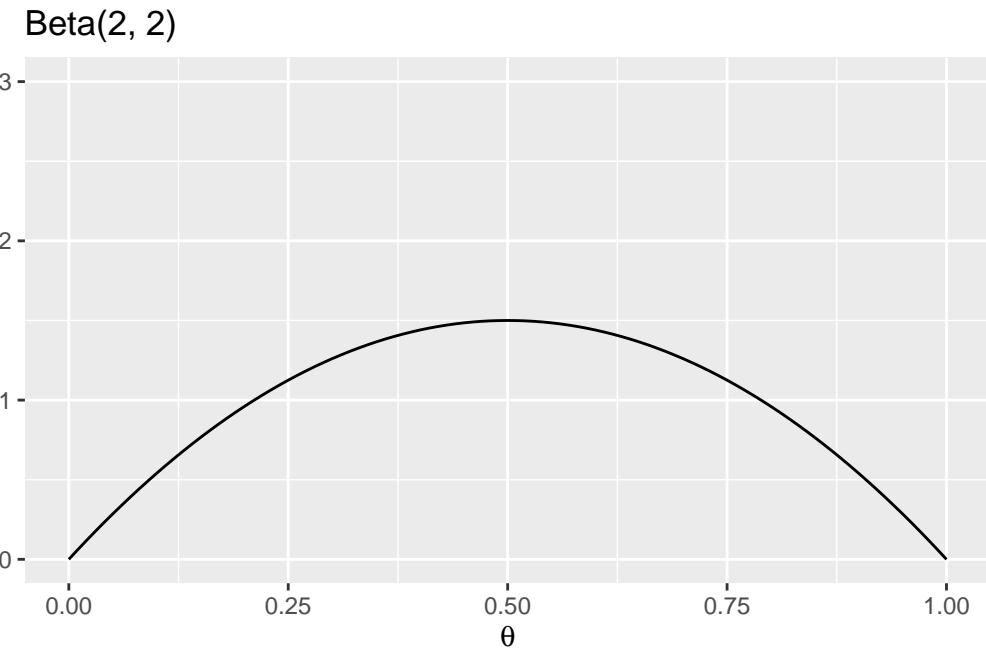


Figure 4.3: A weakly informative Beta(2, 2) prior

### ! Don't Be Stubborn

A good prior should give a non-zero probability/density for all possible values of a parameter

Otherwise, if the prior density for some parameter values is zero, the posterior density will be zero, regardless of how much the data support those parameter values

#### 4.2.5 Data

```
count(Aids2, status)
```

	status	n
1	A	1082
2	D	1761

The likelihood function is highly concentrated. I ran into some numerical issues as the computation gave zero, so I plotted the log-likelihood instead.

```

loglik <- function(th, N = 1082 + 1761, z = 1761) {
  z * log(th) + (N - z) * log(1 - th)
}
ggplot(data.frame(th = c(0.61, 0.63)), aes(x = th)) +
  stat_function(fun = loglik, n = 501) +
  labs(x = expression(theta), y = "Log-likelihood")

```

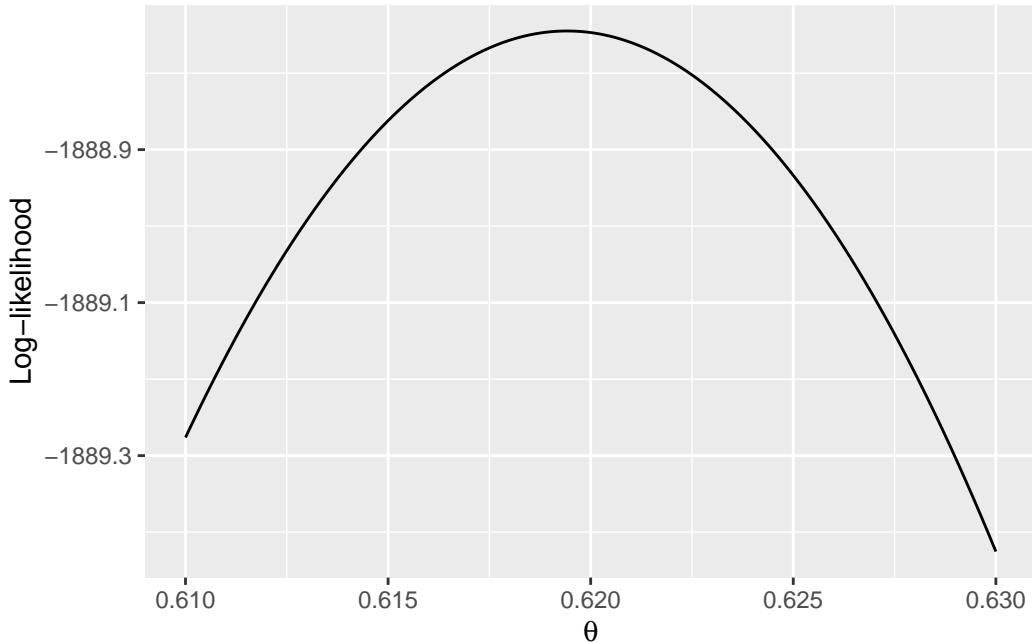


Figure 4.4: Log-likelihood function of the theta parameter

Note I only show a range of  $[0.610, 0.630]$  for the x-axis, which contains where the likelihood (thus also the log-likelihood) peaked.

#### 4.2.6 Posterior

Based on the conjugacy, the posterior of  $\theta$  is  $\text{Beta}(1,807, 1,116)$ . As we are using a conjugate prior, the posterior is also a Beta distribution:

$$P(\theta | y) \sim \text{Beta}(a + z, b + N - z),$$

which is a distribution for  $a + z - 1$  successes and  $b + N - z$  failures. This makes perfect sense as our prior information has  $a - 1$  successes and  $b - 1$  failures, and from our data, we have  $y$  successes and  $n - y$  failures, so our updated belief is based on adding up those successes and failures.

#### 4.2.7 Summarize the posterior

```
set.seed(2119)
num_draws <- 1000
sim_theta <- rbeta(num_draws, shape1 = 1807, shape2 = 1116)
c(`Bayes estimate` = mean(sim_theta),
  `Posterior median` = median(sim_theta),
  `Posterior SD` = sd(sim_theta),
  `MAD` = mad(sim_theta),
  `90% Credible interval (equal-tailed)` = quantile(sim_theta, probs = c(.1, .9)),
  `90% HDI` = HDInterval::hdi(sim_theta, credMass = .9))
```

```
Bayes estimate
0.618209694
Posterior median
0.618382945
Posterior SD
0.008829290
MAD
0.009254166
90% Credible interval (equal-tailed).10%
0.606862338
90% Credible interval (equal-tailed).90%
0.628990588
90% HDI.lower
0.604051851
90% HDI.upper
0.632766654
```

#### 4.2.8 Posterior Predictive Check

Now, we need to know whether the model fits the data well. We do not have much to check for a Bernoulli model if we only have the `status` variable. However, as there is information for other variables, we can use them to check the exchangeability assumption. For example, we can ask whether the data from different state categories are exchangeable. The death rate across the 4 state categories are

	status	
state	A	D
NSW	664	1116

Other	107	142
QLD	78	148
VIC	233	355

	status	
state	A	D
NSW	0.3730337	0.6269663
Other	0.4297189	0.5702811
QLD	0.3451327	0.6548673
VIC	0.3962585	0.6037415

We can now generate predictions from our posterior distribution and model.

```

plist <- vector("list", 12L)
plist[[1]] <- ggplot(
  Aids2,
  aes(x = state, y = mean(status == "D"), fill = state)
) +
  geom_bar(stat = "identity") +
  guides(fill = "none") +
  labs(x = "Observed data", y = "Number of Deaths") +
  theme(axis.title.x = element_text(color = "red")) +
  ylim(0, 1200)
for (i in 1:11) {
  # Get the a value from posterior samples
  theta_post <- rbeta(1, 1763, 1084)
  # For each plausible theta value, generate a status variable
  status_new <- sample(c("D", "A"), nrow(Aids2),
    replace = TRUE,
    prob = c(theta_post, 1 - theta_post)
  )
  df_new <- Aids2 |>
    mutate(status = factor(status_new))
  plist[[i + 1]] <- plist[[1]] %+%
    labs(x = paste("Simulated data", i)) +
    theme(axis.title.x = element_text(color = "black"))
}
gridExtra::grid.arrange(grobs = plist, nrow = 3)

```

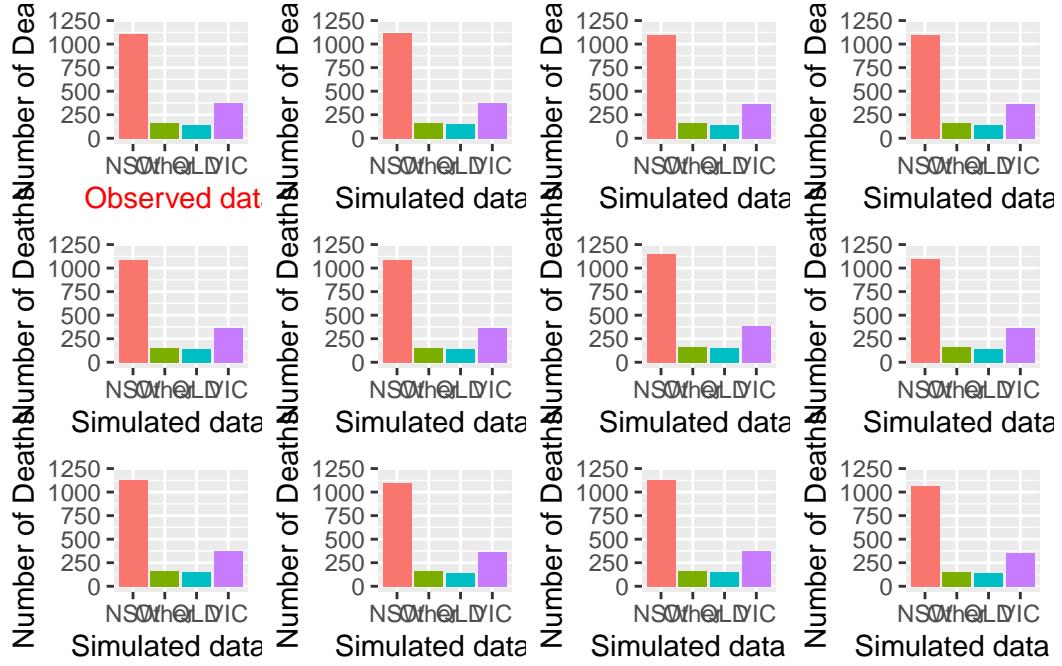


Figure 4.5: Posterior predictive check by comparing the observed data with 11 simulated data sets based on the model

So the observed data (the first subplot) look similar to the simulated data. We can also conduct a posterior predictive check by a test statistic for subgroups. Here, we will use the `bayesplot` package and look at fit across groups:

```
# Draw posterior samples of theta
post_sample <- rbeta(1e4, 1807, 1116)
# Initialize a S by N matrix to store the simulated data
y_tilde <- matrix(NA,
                  nrow = length(post_sample),
                  ncol = length(Aids2$status))

for (s in seq_along(post_sample)) {
  theta_s <- post_sample[s]
  status_new <- sample(c("D", "A"), nrow(Aids2),
                       replace = TRUE,
                       prob = c(theta_s, 1 - theta_s))
}

y_tilde[s,] <- as.numeric(status_new == "D")
}

bayesplot::ppc_stat_grouped(
  as.numeric(Aids2$status == "D"),
```

```

    yrep = y_tilde,
    group = Aids2$state
)

```

`stat\_bin()` using `bins = 30`. Pick better value with `binwidth`.

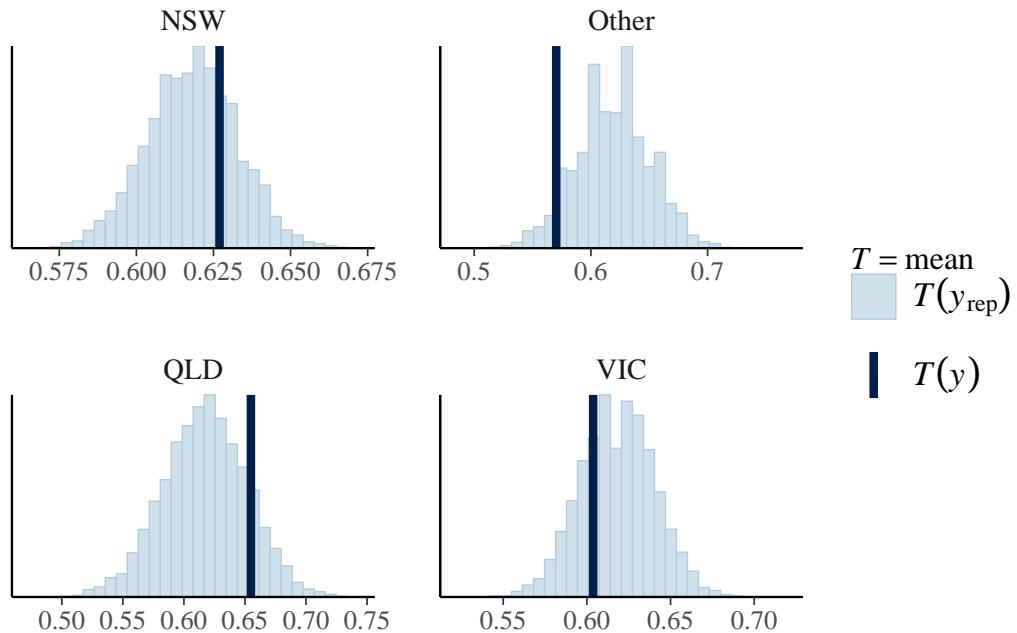


Figure 4.6: Posterior predictive check by states

If the fit is good, the mean, indicated by the darker line, should be within the simulated distribution based on the model. So the model that assumes observations are exchangeable across states is not too off, although it seems fitting less well for **Other** states.

#### 4.2.8.1 Another check on age

```

# Create an age group indicator
age50 <- factor(Aids2$age > 50, labels = c("<= 50", "> 50"))
# Draw posterior samples of theta
post_sample <- rbeta(1e4, 1807, 1116)
# Initialize a S by N matrix to store the simulated data

```

```

y_tilde <- matrix(NA,
                  nrow = length(post_sample),
                  ncol = length(Aids2$status))
for (s in seq_along(post_sample)) {
  theta_s <- post_sample[s]
  status_new <- sample(c("D", "A"), nrow(Aids2),
                        replace = TRUE,
                        prob = c(theta_s, 1 - theta_s)
  )
  y_tilde[s,] <- as.numeric(status_new == "D")
}
bayesplot::ppc_stat_grouped(
  as.numeric(Aids2$status == "D"),
  yrep = y_tilde,
  group = age50
)

```

``stat_bin()` using `bins = 30`.` Pick better value with ``binwidth``.

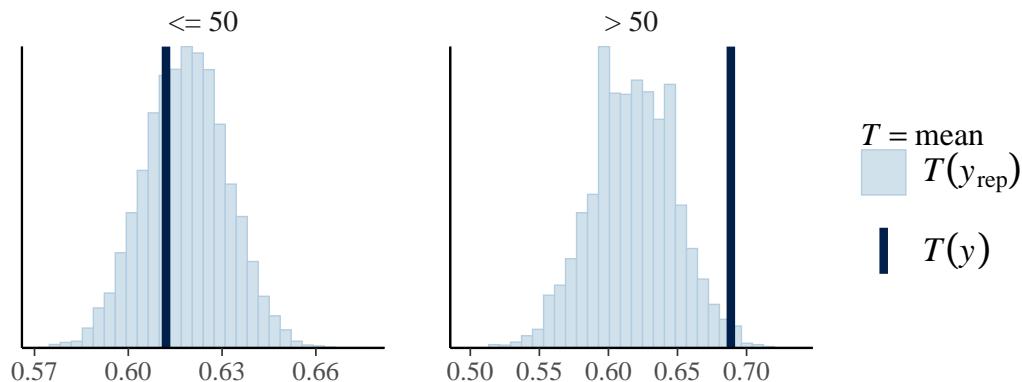


Figure 4.7: Posterior predictive check by age groups ( $\leq 50$  vs.  $> 50$ )

As can be seen, the model seems off for those aged 50+.

#### 4.2.9 Comparison to frequentist results

Using maximum likelihood, the estimated death rate would be  $\hat{\theta} = 1761/2843 = 0.62$ , with a standard error ( $SE$ ) of  $\sqrt{0.62(1 - 0.62)/n} = 0.0091$ , with a 90% confidence interval of  $[0.6, 0.63]$ , which is similar to the interval with Bayesian inference.

#### 4.2.10 Sensitivity to different priors

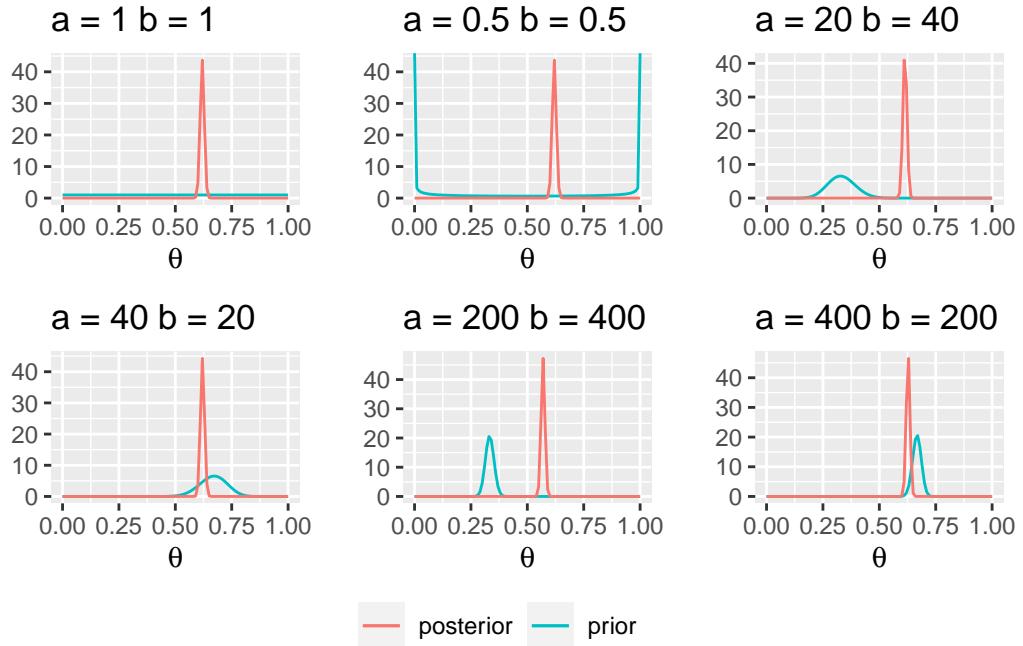


Figure 4.8: Sensitivity of posterior to different priors

You can see one needs (a) a very strong prior (equivalent to 600 data points) and (b) the prior and the data not agreeing to get a substantially different conclusion.

# 5 Beta-Bernoulli Model With Stan

In the previous lecture, we fitted a Beta-Bernoulli model using Gibbs sampling with our own R code. While this is doable in this relatively simple model (it only has one parameter), for more complex models, Gibbs sampling and other MCMC methods (to be introduced in a later class) require quite a lot of programming. Fortunately for us, we have some readily available software for doing MCMC. While it is probably overkill for the Beta-Bernoulli model, we will learn working with Stan by using it to analyze the Beta-Bernoulli model. Specifically, we will learn to:

1. Install Stan and the R package `cmdstanr` for communicating between R and Stan;
2. Write Stan code for the Beta-Bernoulli model;
3. Draw posterior samples using Stan;
4. Summarize and plot the posterior samples.

We will talk about convergence of MCMC, which is an extremely important topic, in a later class.

## 5.1 Installing Stan



1. Follow the steps in <https://mc-stan.org/cmdstanr/> to install the `cmdstanr` package.
2. Load the `cmdstanr` package, and run `install_cmdstan()`

```
library(cmdstanr)  
install_cmdstan()
```

3. If you run into an error in the previous step related to C++ toolchain, follow the directions here: <https://mc-stan.org/cmdstanr/articles/cmdstanr.html>

## 5.2 Fitting a Beta-Bernoulli Model in Stan

### 5.2.1 Data Import

From last class:

```
data("Aids2", package = "MASS")
head(Aids2)
```

	state	sex	diag	death	status	T.categ	age
1	NSW	M	10905	11081	D	hs	35
2	NSW	M	11029	11096	D	hs	53
3	NSW	M	9551	9983	D	hs	42
4	NSW	M	9577	9654	D	haem	44
5	NSW	M	10015	10290	D	hs	39
6	NSW	M	9971	10344	D	hs	36

### 5.2.2 Writing Stan syntax

Stan has its own syntax and is different from R. For example, we want to fit the following Beta-Bernoulli model:

$$y_i \sim \text{Bern}(\theta) \text{ for } i = 1, 2, \dots, N$$
$$P(\theta) \sim \text{Beta}(2, 2)$$

and the model can be written in Stan as follows:

```
data {
  int<lower=0> N; // number of observations
  array[N] int<lower=0,upper=1> y; // y
}
parameters {
  real<lower=0,upper=1> theta; // theta parameter
}
model {
  theta ~ beta(2,2); // prior: Beta(2, 2)
  y ~ bernoulli(theta); // model: Bernoulli
}
```

- 💡 Save the above model syntax in a separate file ending in `.stan`. For example, I saved the syntax in the file `beta-bernoulli.stan` in a folder named `stan_code`.

In Stan, anything after `//` denotes comments (like `#` in R) and will be ignored by the program. In each block (e.g., `data {}`), a statement should end with a semicolon (`;`). There are several blocks in the above Stan code:

- **data:** The data input for Stan is usually not only an R data frame, but a list that includes other information, such as sample size, number of predictors, and prior scales. Each type of data has an input type, such as
  - `int` = integer,
  - `real` = numbers with decimal places,
  - `matrix` = 2-dimensional data of real numbers,
  - `vector` = 1-dimensional data of real numbers, and
  - `array` = 1- to many-dimensional data. For example, we use `array[N]` for the data type of `y`, because it is a vector, but each element is an integer (and cannot take decimals).

We can set the lower and upper bounds so that Stan can check the input data. In the above, we used `<lower=0,upper=1>`.

- **parameters:** The parameters to be estimated
- **transformed parameters:** optional variables that are transformations of the model parameters. It is usually used for more advanced models to allow more efficient MCMC sampling.
- **model:** It includes expressions of prior distributions for each parameter and the likelihood for the data. There are many possible distributions that can be used in Stan.
- **generated quantities:** Any quantities that are not part of the model but can be computed from the parameters for every iteration. Examples include posterior generated samples, effect sizes, and log-likelihood (for fit computation). We will see an example later.

### 5.2.3 Compiling the Stan model from R

To compile the model, we can call the `cmdstan_model` function in `cmdstanr`:

```
bern_mod <- cmdstan_model("stan_code/beta-bernoulli.stan")
```

### 5.2.4 Posterior Sampling

We need to first prepare the data for input to Stan. In the Stan code, we have two objects in the data block:  $N$  and  $y$ , so we need to have a list of two elements in R:

```
Aids2_standata <- list(  
  N = nrow(Aids2),  
  y = as.integer(Aids2$status == "D") # integer  
)
```

Now we can draw posterior samples:

```
fit <- bern_mod$sample(Aids2_standata)
```

Running MCMC with 4 sequential chains...

```
Chain 1 Iteration: 1 / 2000 [ 0%] (Warmup)  
Chain 1 Iteration: 100 / 2000 [ 5%] (Warmup)  
Chain 1 Iteration: 200 / 2000 [ 10%] (Warmup)  
Chain 1 Iteration: 300 / 2000 [ 15%] (Warmup)  
Chain 1 Iteration: 400 / 2000 [ 20%] (Warmup)  
Chain 1 Iteration: 500 / 2000 [ 25%] (Warmup)  
Chain 1 Iteration: 600 / 2000 [ 30%] (Warmup)  
Chain 1 Iteration: 700 / 2000 [ 35%] (Warmup)  
Chain 1 Iteration: 800 / 2000 [ 40%] (Warmup)  
Chain 1 Iteration: 900 / 2000 [ 45%] (Warmup)  
Chain 1 Iteration: 1000 / 2000 [ 50%] (Warmup)  
Chain 1 Iteration: 1001 / 2000 [ 50%] (Sampling)  
Chain 1 Iteration: 1100 / 2000 [ 55%] (Sampling)  
Chain 1 Iteration: 1200 / 2000 [ 60%] (Sampling)  
Chain 1 Iteration: 1300 / 2000 [ 65%] (Sampling)  
Chain 1 Iteration: 1400 / 2000 [ 70%] (Sampling)  
Chain 1 Iteration: 1500 / 2000 [ 75%] (Sampling)  
Chain 1 Iteration: 1600 / 2000 [ 80%] (Sampling)  
Chain 1 Iteration: 1700 / 2000 [ 85%] (Sampling)  
Chain 1 Iteration: 1800 / 2000 [ 90%] (Sampling)  
Chain 1 Iteration: 1900 / 2000 [ 95%] (Sampling)  
Chain 1 Iteration: 2000 / 2000 [100%] (Sampling)  
Chain 1 finished in 0.0 seconds.  
Chain 2 Iteration: 1 / 2000 [ 0%] (Warmup)  
Chain 2 Iteration: 100 / 2000 [ 5%] (Warmup)  
Chain 2 Iteration: 200 / 2000 [ 10%] (Warmup)
```

```
Chain 2 Iteration: 300 / 2000 [ 15%] (Warmup)
Chain 2 Iteration: 400 / 2000 [ 20%] (Warmup)
Chain 2 Iteration: 500 / 2000 [ 25%] (Warmup)
Chain 2 Iteration: 600 / 2000 [ 30%] (Warmup)
Chain 2 Iteration: 700 / 2000 [ 35%] (Warmup)
Chain 2 Iteration: 800 / 2000 [ 40%] (Warmup)
Chain 2 Iteration: 900 / 2000 [ 45%] (Warmup)
Chain 2 Iteration: 1000 / 2000 [ 50%] (Warmup)
Chain 2 Iteration: 1001 / 2000 [ 50%] (Sampling)
Chain 2 Iteration: 1100 / 2000 [ 55%] (Sampling)
Chain 2 Iteration: 1200 / 2000 [ 60%] (Sampling)
Chain 2 Iteration: 1300 / 2000 [ 65%] (Sampling)
Chain 2 Iteration: 1400 / 2000 [ 70%] (Sampling)
Chain 2 Iteration: 1500 / 2000 [ 75%] (Sampling)
Chain 2 Iteration: 1600 / 2000 [ 80%] (Sampling)
Chain 2 Iteration: 1700 / 2000 [ 85%] (Sampling)
Chain 2 Iteration: 1800 / 2000 [ 90%] (Sampling)
Chain 2 Iteration: 1900 / 2000 [ 95%] (Sampling)
Chain 2 Iteration: 2000 / 2000 [100%] (Sampling)
Chain 2 finished in 0.0 seconds.

Chain 3 Iteration: 1 / 2000 [ 0%] (Warmup)
Chain 3 Iteration: 100 / 2000 [ 5%] (Warmup)
Chain 3 Iteration: 200 / 2000 [ 10%] (Warmup)
Chain 3 Iteration: 300 / 2000 [ 15%] (Warmup)
Chain 3 Iteration: 400 / 2000 [ 20%] (Warmup)
Chain 3 Iteration: 500 / 2000 [ 25%] (Warmup)
Chain 3 Iteration: 600 / 2000 [ 30%] (Warmup)
Chain 3 Iteration: 700 / 2000 [ 35%] (Warmup)
Chain 3 Iteration: 800 / 2000 [ 40%] (Warmup)
Chain 3 Iteration: 900 / 2000 [ 45%] (Warmup)
Chain 3 Iteration: 1000 / 2000 [ 50%] (Warmup)
Chain 3 Iteration: 1001 / 2000 [ 50%] (Sampling)
Chain 3 Iteration: 1100 / 2000 [ 55%] (Sampling)
Chain 3 Iteration: 1200 / 2000 [ 60%] (Sampling)
Chain 3 Iteration: 1300 / 2000 [ 65%] (Sampling)
Chain 3 Iteration: 1400 / 2000 [ 70%] (Sampling)
Chain 3 Iteration: 1500 / 2000 [ 75%] (Sampling)
Chain 3 Iteration: 1600 / 2000 [ 80%] (Sampling)
Chain 3 Iteration: 1700 / 2000 [ 85%] (Sampling)
Chain 3 Iteration: 1800 / 2000 [ 90%] (Sampling)
Chain 3 Iteration: 1900 / 2000 [ 95%] (Sampling)
Chain 3 Iteration: 2000 / 2000 [100%] (Sampling)
Chain 3 finished in 0.0 seconds.
```

```

Chain 4 Iteration: 1 / 2000 [  0%] (Warmup)
Chain 4 Iteration: 100 / 2000 [  5%] (Warmup)
Chain 4 Iteration: 200 / 2000 [ 10%] (Warmup)
Chain 4 Iteration: 300 / 2000 [ 15%] (Warmup)
Chain 4 Iteration: 400 / 2000 [ 20%] (Warmup)
Chain 4 Iteration: 500 / 2000 [ 25%] (Warmup)
Chain 4 Iteration: 600 / 2000 [ 30%] (Warmup)
Chain 4 Iteration: 700 / 2000 [ 35%] (Warmup)
Chain 4 Iteration: 800 / 2000 [ 40%] (Warmup)
Chain 4 Iteration: 900 / 2000 [ 45%] (Warmup)
Chain 4 Iteration: 1000 / 2000 [ 50%] (Warmup)
Chain 4 Iteration: 1001 / 2000 [ 50%] (Sampling)
Chain 4 Iteration: 1100 / 2000 [ 55%] (Sampling)
Chain 4 Iteration: 1200 / 2000 [ 60%] (Sampling)
Chain 4 Iteration: 1300 / 2000 [ 65%] (Sampling)
Chain 4 Iteration: 1400 / 2000 [ 70%] (Sampling)
Chain 4 Iteration: 1500 / 2000 [ 75%] (Sampling)
Chain 4 Iteration: 1600 / 2000 [ 80%] (Sampling)
Chain 4 Iteration: 1700 / 2000 [ 85%] (Sampling)
Chain 4 Iteration: 1800 / 2000 [ 90%] (Sampling)
Chain 4 Iteration: 1900 / 2000 [ 95%] (Sampling)
Chain 4 Iteration: 2000 / 2000 [100%] (Sampling)
Chain 4 finished in 0.0 seconds.

```

```

All 4 chains finished successfully.
Mean chain execution time: 0.0 seconds.
Total execution time: 0.7 seconds.

```

For this simple model, this takes less than a second.

### 5.2.5 Summarizing and plotting the posterior samples

```

# Actual posterior samples
fit$draws("theta", format = "draws_df")

# A draws_df: 1000 iterations, 4 chains, and 1 variables
  theta
1  0.63
2  0.63
3  0.62

```

```

4  0.63
5  0.62
6  0.62
7  0.62
8  0.62
9  0.64
10 0.63
# ... with 3990 more draws
# ... hidden reserved variables {'chain', 'iteration', 'draw'}
```

# Summary table

```
fit$summary()
```

# A tibble: 2 x 10

variable	mean	median	sd	mad	q5	q95	rhat	ess_bulk
<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1 lp_	-1892.	-1892.	0.698	0.326	-1894.	-1.89e+3	1.00	1598.
2 theta	0.619	0.619	0.00920	0.00930	0.604	6.34e-1	1.00	1390.

# i 1 more variable: ess\_tail <dbl>

# Histogram

```
fit$draws("theta") |>
  mcmc_hist()
```

`stat\_bin()` using `bins = 30`. Pick better value with `binwidth`.

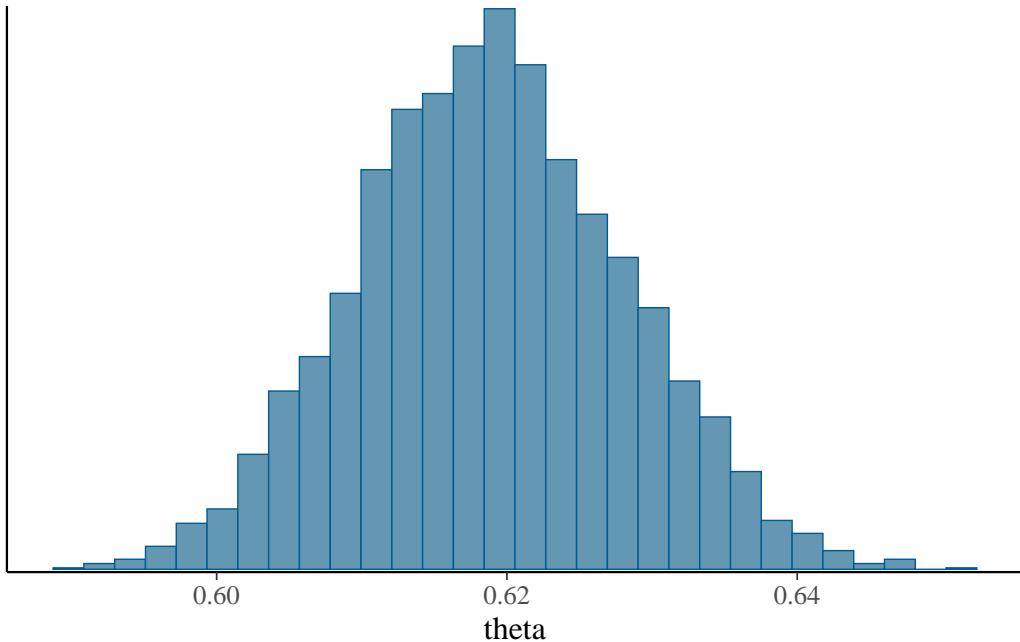


Figure 5.1: Posterior distribution of the parameter.

The results are similar to those in last class.

### 5.3 Prior Predictive Check

In Bayesian analyses, it is recommended to check both the prior and the model. This can be done by

1. Prior predictive check: Simulating data from the prior distribution, and see if the simulated data fit our prior belief.
2. Posterior predictive check: Simulating data from the posterior distribution, and see if the simulated data are comparable to the observed data.

We will use the following Stan code to do prior and posterior predictive checks, which has an additional generated quantity block to obtain

- a. `prior_theta`: simulated values of  $\theta$  based on the prior distribution
- b. `prior_ytilde`: simulated data based on the prior distribution of  $\theta$
- c. `ytilde`: simulated data based on the posterior distribution of  $\theta$

```

data {
  int<lower=0> N; // number of observations
  array[N] int<lower=0,upper=1> y; // y
}
parameters {
  real<lower=0,upper=1> theta; // theta parameter
}
model {
  theta ~ beta(2, 2); // prior: Beta(2, 2)
  y ~ bernoulli(theta); // model: Bernoulli
}
generated quantities {
  real prior_theta = beta_rng(2, 2);
  array[N] int prior_ytilde;
  array[N] int ytilde;
  for (i in 1:N) {
    ytilde[i] = bernoulli_rng(theta);
    prior_ytilde[i] = bernoulli_rng(prior_theta);
  }
}

```

```

bern_pp_mod <- cmdstan_model("stan_code/beta-bernoulli-pp.stan")
bern_pp_fit <- bern_pp_mod$sample(
  Aids2_standata,
  refresh = 500 # show progress every 500 iterations
)

```

Running MCMC with 4 sequential chains...

```

Chain 1 Iteration: 1 / 2000 [ 0%] (Warmup)
Chain 1 Iteration: 500 / 2000 [ 25%] (Warmup)
Chain 1 Iteration: 1000 / 2000 [ 50%] (Warmup)
Chain 1 Iteration: 1001 / 2000 [ 50%] (Sampling)
Chain 1 Iteration: 1500 / 2000 [ 75%] (Sampling)
Chain 1 Iteration: 2000 / 2000 [100%] (Sampling)
Chain 1 finished in 0.9 seconds.
Chain 2 Iteration: 1 / 2000 [ 0%] (Warmup)
Chain 2 Iteration: 500 / 2000 [ 25%] (Warmup)
Chain 2 Iteration: 1000 / 2000 [ 50%] (Warmup)
Chain 2 Iteration: 1001 / 2000 [ 50%] (Sampling)
Chain 2 Iteration: 1500 / 2000 [ 75%] (Sampling)
Chain 2 Iteration: 2000 / 2000 [100%] (Sampling)

```

```

Chain 2 finished in 0.9 seconds.
Chain 3 Iteration: 1 / 2000 [  0%] (Warmup)
Chain 3 Iteration: 500 / 2000 [ 25%] (Warmup)
Chain 3 Iteration: 1000 / 2000 [ 50%] (Warmup)
Chain 3 Iteration: 1001 / 2000 [ 50%] (Sampling)
Chain 3 Iteration: 1500 / 2000 [ 75%] (Sampling)
Chain 3 Iteration: 2000 / 2000 [100%] (Sampling)
Chain 3 finished in 0.8 seconds.
Chain 4 Iteration: 1 / 2000 [  0%] (Warmup)
Chain 4 Iteration: 500 / 2000 [ 25%] (Warmup)
Chain 4 Iteration: 1000 / 2000 [ 50%] (Warmup)
Chain 4 Iteration: 1001 / 2000 [ 50%] (Sampling)
Chain 4 Iteration: 1500 / 2000 [ 75%] (Sampling)
Chain 4 Iteration: 2000 / 2000 [100%] (Sampling)
Chain 4 finished in 0.9 seconds.

```

```

All 4 chains finished successfully.
Mean chain execution time: 0.9 seconds.
Total execution time: 3.8 seconds.

```

With Stan, because we obtained 4,000 prior/posterior draws (the software default) of  $\theta$ , we also obtained 4,000 simulated data sets. We can see the first one, based on only the prior distribution (i.e.,  $\theta \sim \text{Beta}(2, 2)$ ):

```

bern_pp_fit$draws("prior_ytilde", format = "draws_df")[1, ] |>
  as.numeric() |>
  table()

```

```

0     1
1109 1737

```

Note that the data set has more 1's than 0's. Our prior is weak, which means that it allows for a lot of variation in how the data would look.

The distribution of simulated *data* based on the prior distribution of the parameters is called the *prior predictive distribution*. Mathematically, we write it as

$$P(\tilde{y}) = \int P(\tilde{y}|\theta)P(\theta) d\theta$$

Because we have 4,000 data sets, it is not easy to visualize all individual data points. Instead, we can visualize some summary statistics of the simulated data. Here, we will choose the proportion of deaths (i.e., the mean of the variable) in each simulated data set:

```

bern_pp_fit$draws("prior_ytilde", format = "draws_matrix") |>
  ppd_stat(stat = "mean")

```

`stat\_bin()` using `bins = 30`. Pick better value with `binwidth`.

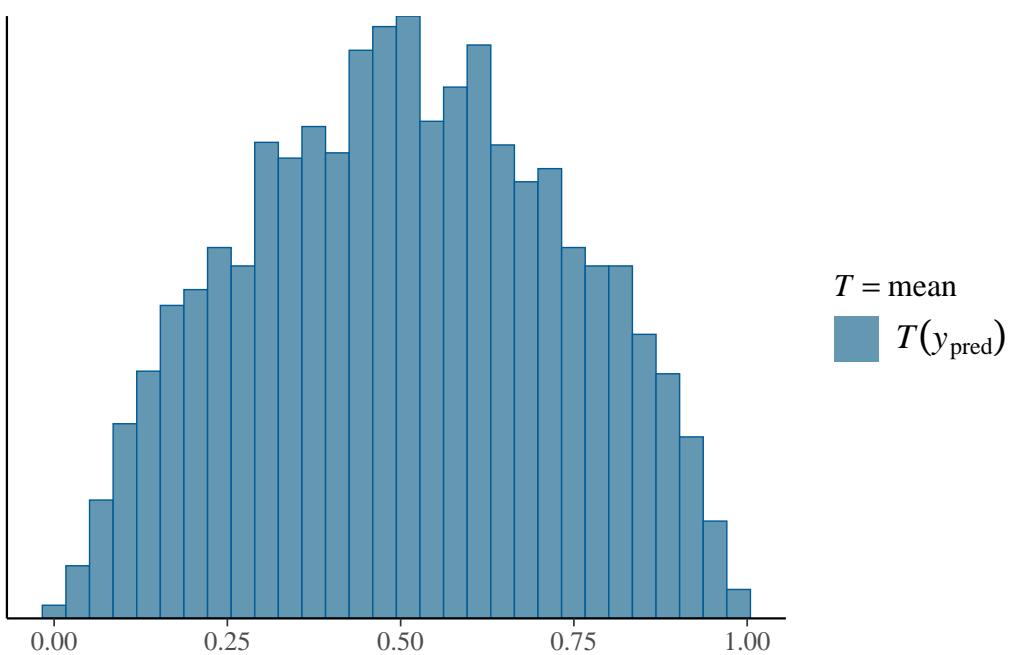


Figure 5.2: Prior predictive distribution of the sample mean.

This represents our prior belief about the proportion of deaths without looking at the actual data. If this doesn't seem to match our belief, we may want to modify our prior distribution and do the prior predictive check again, until the simulated data matches our actual prior belief.

## 5.4 Posterior Predictive Check

### 5.4.1 Check 1: Sample Mean

$$P(\tilde{y}|y) = \int P(\tilde{y}|\theta, y)P(\theta|y) d\theta$$

The difference here is that we use the posterior distribution  $P(\theta|y)$  instead of the prior distribution  $P(\theta)$ . We can obtain the posterior predictive distribution of the death rate, and indicate the actual data in the plot:

```
bern_pp_fit$draws("ytilde", format = "draws_matrix") |>
  ppc_stat(y = Aids2_standata$y, stat = "mean")
```

``stat_bin()` using `bins = 30`. Pick better value with `binwidth`.`

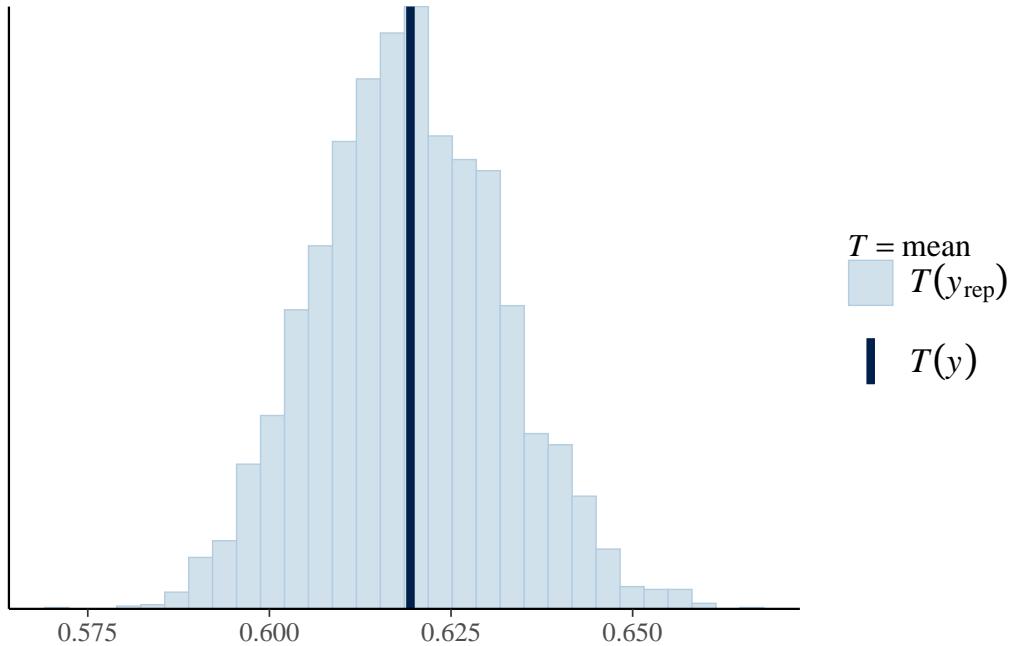


Figure 5.3: Posterior predictive distribution of the sample mean.

#### 5.4.2 Check 2: Sample Mean by Age Group

```
# Create binary indicator of two age groups
age50 <- factor(Aids2$age > 50, labels = c("<= 50", "> 50"))
bern_pp_fit$draws("ytilde", format = "draws_matrix") |>
  ppc_stat_grouped(y = Aids2_standata$y, group = age50, stat = "mean")
```

``stat_bin()` using `bins = 30`. Pick better value with `binwidth`.`

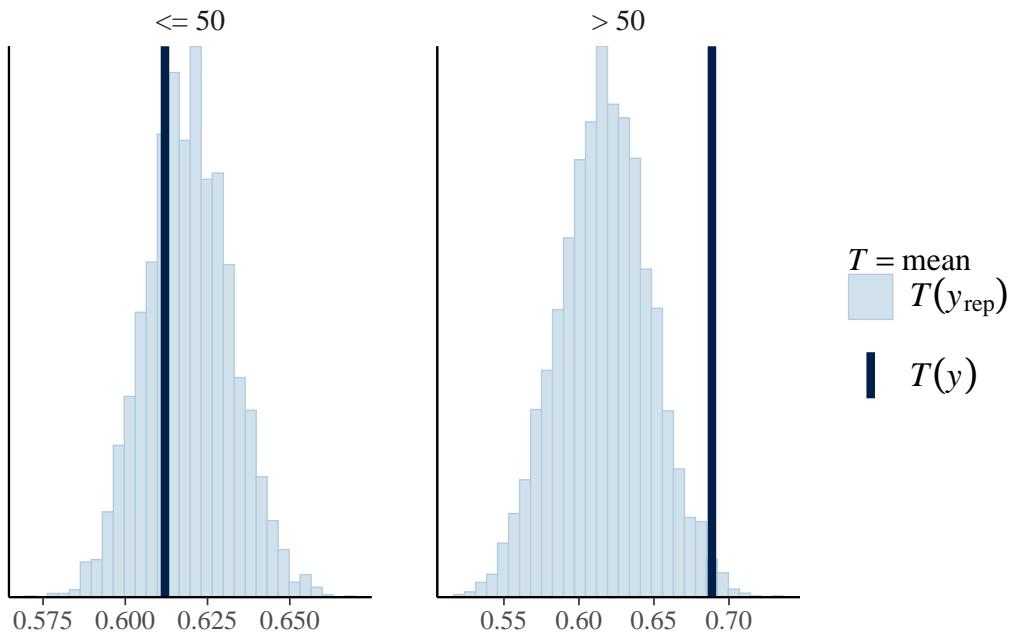


Figure 5.4: Posterior predictive distribution of the sample mean by age group.

# 6 Poisson Model

Please note: This document uses count data on fatal police shootings.

I came across this data set from <https://andrewpwheeler.com/2021/01/11/checking-a-poisson-distribution-fit-an-example-with-officer-involved-shooting-deaths-wapo-data-r-functions/>

As explained [here](#), the data are by the Washington Post in an effort to record every fatal shooting in the United States by a police officer since January 1, 2015.

## 6.1 Research Question

What's the rate of fatal police shootings in the United States per year?

## 6.2 Data Import and Pre-Processing

```
# Import data
fps_dat <- read_csv(
  "https://github.com/washingtonpost/data-police-shootings/raw/master/v2/fatal-police-shootings.csv")
```

We first count the data by year

```
# Create a year column
fps_dat <- fps_dat |>
  mutate(year = format(date, format = "%Y"))
# Filter out the latest year
fps_1523 <- filter(fps_dat, year != max(year))
count(fps_1523, year)
```

```
# A tibble: 9 x 2
  year     n
  <chr> <int>
1 2015    995
2 2016    959
3 2017    984
4 2018    992
5 2019    994
6 2020   1021
7 2021   1050
8 2022   1097
9 2023   1164
```

Our interest is the rate of occurrence of fatal police shootings per year. Denote this as  $\theta$ . The support of  $\theta$  is  $[0, \infty)$ .

A *Poisson model* is usually a starting point for analyzing count data in a fixed amount of time. It assumes that the data follow a Poisson distribution with a fixed rate parameter:

$$P(y | \theta) \propto \theta^y \exp(-\theta),$$

where the data can be any non-negative integers (no decimals).

## 6.3 Choosing a Prior

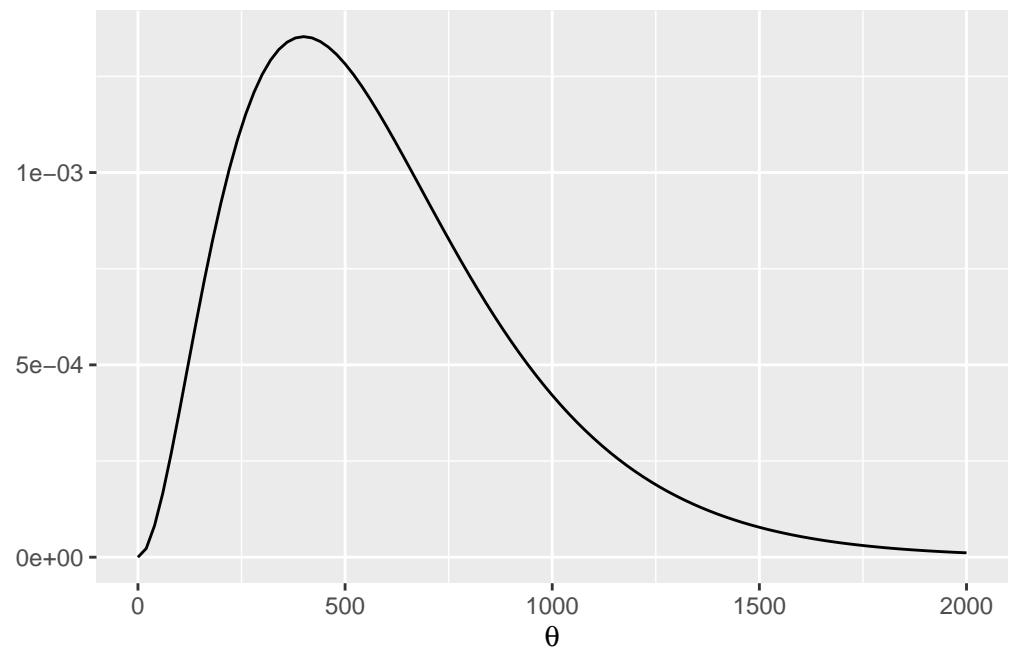
The [Gamma distribution](#) has support:  $[0, \infty)$ , and is a conjugate family to the Poisson model. The Gamma distribution has the form

$$P(\theta) \propto \theta^{a-1} \exp(-b\theta),$$

where  $a$  is the prior incidence rate, and  $b$  is the number of prior data points to control for the prior strength. Here, without much prior knowledge, I would simply guess there is one fatal shooting per state per month, so 600 shootings per year, but my belief is pretty weak, so I will assume a prior  $b$  of  $1 / 200$  (one observation is one year). The  $a$  will be  $600 * b = 3$ .

Here's a plot:

```
ggplot(data.frame(th = c(0, 2000)), aes(x = th)) +
  stat_function(fun = dgamma,
  args = list(shape = 3, rate = 1 / 200)) +
  labs(y = "", x = expression(theta))
```



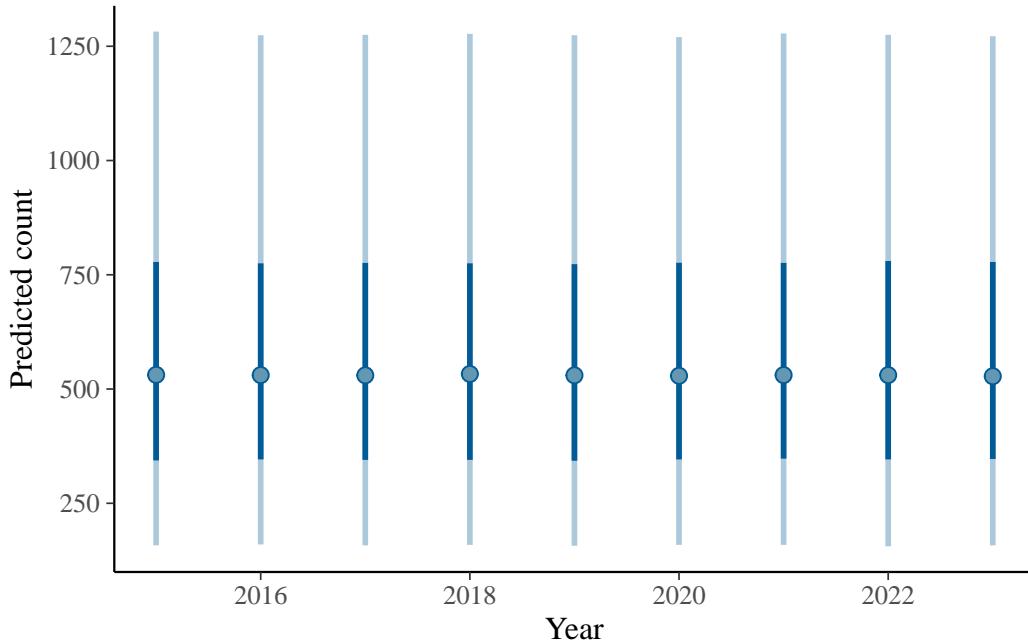




### 6.3.4 Prior predictive check

Here I plot simulated trends from the prior distribution.

```
# Extract predicted y from prior
fit$draws("prior_ytilde", format = "draws_matrix") |>
  ppd_intervals(x = 2015:2023) +
  labs(x = "Year", y = "Predicted count")
```

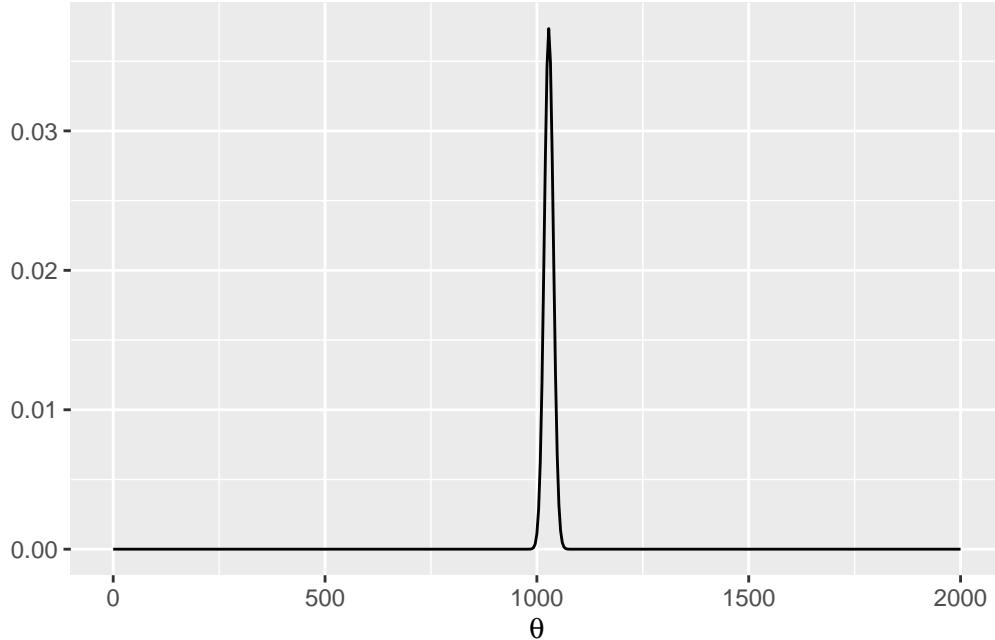


The check is on whether the numbers seem reasonably reflective of my knowledge.

## 6.4 Posterior

With a conjugate prior, the posterior distribution is  $\text{Gamma}(a + \sum_{i=1}^N y_i, b + N)$ .

```
ggplot(data.frame(th = c(0, 2000)), aes(x = th)) +
  stat_function(fun = dgamma,
  args = list(shape = 3 + nrow(fps_1523),
             rate = 1 / 200 + fps_standata$N), n = 501) +
  labs(y = "", x = expression(theta))
```



## 6.5 Posterior Predictive Check

Plot predicted data from the posterior against observed data

```
# Extract predicted y from posterior
fit$draws("ytilde", format = "draws_matrix") |>
  ppc_intervals(
    y = fps_standata$y,
    x = 2015:2023
  ) +
  labs(x = "Year", y = "Predicted count")
# We can also use `bayesplot::ppc_ribbon()`
fit$draws("ytilde", format = "draws_matrix") |>
  ppc_ribbon(
    y = fps_standata$y,
    x = 2015:2023
  ) +
  labs(x = "Year", y = "Predicted count")
```

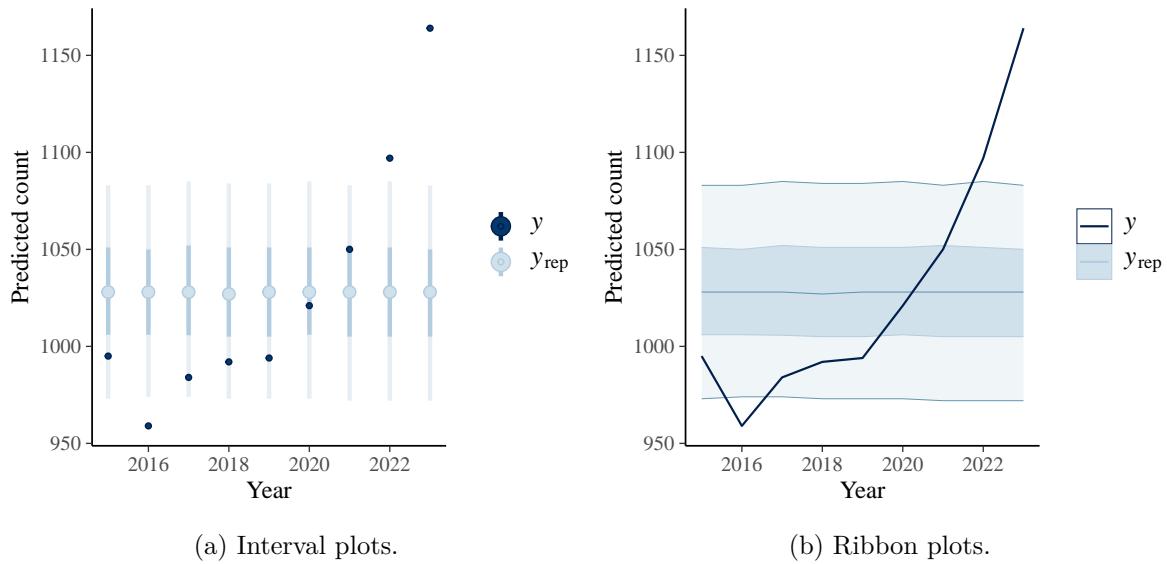


Figure 6.1: Posterior predictive check.

- 💡 From Figure 6.1, one can see that the fit is not good, as there is a large gap between the model prediction and the observed data from recent years. This suggests a need to incorporate the time trend in the model.

## 6.6 Summary of Posterior

For now, we will proceed with interpreting the posterior distribution, despite the apparent misfit.

```
(summ_theta <- fit$summary("theta"))
```

```
# A tibble: 1 x 10
  variable   mean   median     sd     mad     q5    q95   rhat ess_bulk ess_tail
  <chr>     <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>     <dbl>
1 theta      1028.  1028.  10.5   10.8  1011.  1045.  1.00    1164.   1960.
```

So the estimated rate is 1,028 per year, with a 90% CI [1,011, 1,045].

## **Part IV**

### **Week 4**

# 7 Hierarchical Models

Although many statistical models can be fitted using Bayesian or frequentist methods, some models are more naturally used in the Bayesian framework. One class of such models is the family of **hierarchical models**. Consider situations when the data contain *clusters*, such as multiple data points in each of many participants, or multiple participants in each of several treatment conditions. While it is possible to run  $J$  Bayesian analyses for the  $J$  subsets of the data, it is usually more efficient to pool the data together. In this approach, each cluster  $j$  has some parameters  $\theta_j$ , and these  $J \theta$  values themselves come from a common distribution. Figure 7.1 shows a graphical representation of the concept of pooling. This is the same idea as multilevel modeling, a topic we will discuss more later in the course.

- ! A hierarchical model is one in which some higher-level distributions govern one or more parameters, and those higher-level distributions are characterized by *hyperparameters* and can be assigned *hyperpriors*.  
Hierarchical models are commonly used to study and account for individual differences in some model parameters.

In this note, you will see two examples, one from a textbook (Kruschke, 2015) with a hierarchical Bernoulli/binomial model, and another from a classic data set with eight schools, modelled by a hierarchical normal model.

## 7.1 Hierarchical Bernoulli/Binomial

We will first consider a therapeutic touch example (Kruschke, 2015, Chapter 9). Therapeutic touch is a technique in alternative medicine to relieve pain, but scientific evidence does not support its effectiveness. The data here are from an experiment where the experimenter randomly hovered their hand over either the participant's left or right hand, and the participant had to guess which hand was being hovered without seeing. This is repeated 10 times for each participant. There are a total of 28 participants in the dataset.

Previously, we have seen the Bernoulli model for  $N$  outcomes (i.e., whether the guess is correct):

$$y_i \sim \text{Bern}(\theta), \text{ for } i = 1, \dots, N$$

We assumed exchangeability with  $\theta$  being the participant's "ability" to guess correctly.

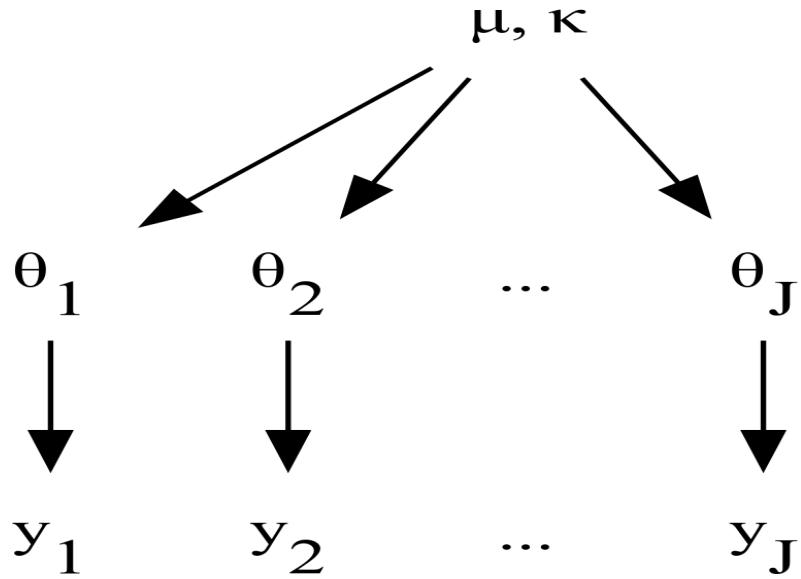


Figure 7.1: A graphical representation of a hierarchical binomial model.

### i Alternative Parameterization of Beta

In the last class, we have used the Beta( $a, b$ ) prior for a Bernoulli outcome, such that

$$P(\theta | a, b) \propto \theta^{a-1}(1-\theta)^{b-1}.$$

However, in hierarchical models to be discussed later, it is beneficial to consider another way to express the Beta distribution, in terms of the *prior mean*,  $\mu = a/(a+b)$ ,  $\mu \in [0, 1]$ , and the concentration,  $\kappa = a+b$ ,  $\kappa \in [0, \infty)$ . So, instead of the above formula, we can write

$$P(\theta | \mu, \kappa) \propto \theta^{\mu\kappa-1}(1-\theta)^{(1-\mu)\kappa-1}.$$

The two expressions represent exactly the same distribution, but just in terms of parameters of different meanings. Therefore, they are referred to as **different parameterization** of the Beta distribution.

### 7.1.1 Multiple Bernoulli = Binomial

With  $N$  exchangeable Bernoulli observations, an equivalent but more efficient way to code the model is to use the *binomial* distribution. Let  $z = \sum_{i=1}^N y_i$ , then

$$z \sim \text{Bin}(N, \theta)$$

### 7.1.2 Multiple Binomial Observations

Now, because we have multiple participants, we could study whether each participant had noticeable differences in their guessing ability. We can use a binomial model for each participant, from participant 1 to participant  $J$ :

$$\begin{aligned} z_1 &\sim \text{Bin}(N_1, \theta_1) \\ z_2 &\sim \text{Bin}(N_2, \theta_2) \\ &\vdots \\ z_J &\sim \text{Bin}(N_J, \theta_J) \end{aligned}$$

Or instead of writing  $J$  equations, we can use the subscript  $j$  to refer to each participant:

$$z_j \sim \text{Bin}(N_j, \theta_j)$$

If we believe that all participants have the same ability, then we could consider the model

$$z_j \sim \text{Bin}(N_j, \theta),$$

which still contains only one parameter,  $\theta$ . However, if we have reason to believe that the coins have different biases, then we should have

$$z_j \sim \text{Bin}(N_j, \theta_j),$$

with parameters  $\theta_1, \dots, \theta_j$ .

We can assign priors to each  $\theta_j$ . However, suppose our prior belief is that there's something common among the different participants (e.g., they're all human beings), so they come from a common distribution. In that case, we can have common parameters for the prior distributions of the  $\theta$ s:

$$\theta_j \sim \text{Beta\_Proportion}(\mu, \kappa),$$

note I use Beta\_Proportion to denote the mean parameterization. Here, we express the prior belief that the **mean ability** of the different participants is  $\mu$ , and how each participant differs from the mean depends on  $\kappa$ . Now,  $\mu$  and  $\kappa$  are hyperparameters. We can assign some fixed values to  $\mu$  and  $\kappa$ , as if we know what the average ability is. However, the power of the

hierarchical model is that we can put priors (or hyper priors) on  $\mu$  and  $\kappa$ , and obtain posterior distributions of them, based on what the data say.

What priors to use for  $\mu$  and  $\kappa$ ?  $\mu$  is relatively easy because it is the mean ability; if we put a Beta prior for each participant's ability, we can again use a Beta prior for the mean ability.  $\kappa$  is more challenging. A larger  $\kappa$  means that the participants' abilities are more similar to each other. We can perform a prior predictive check to see what the data look like. As a starting point, some textbook (e.g., chapter 9 of Kruschke, 2015) suggested using  $\text{Gamma}(0.01, 0.01)$ . So the full model in our case, with a weak Beta(1.5, 1.5) prior on  $\mu$ , is

Model:

$$\begin{aligned}z_j &\sim \text{Bin}(N_j, \theta_j) \\ \theta_j &\sim \text{Beta2}(\mu, \kappa)\end{aligned}$$

Prior:

$$\begin{aligned}\mu &\sim \text{Beta}(1.5, 1.5) \\ \kappa &\sim \text{Gamma}(0.01, 0.01)\end{aligned}$$

We will import the data and fit the model

```
# Data file from GitHub
tt_url <- paste0(
  "https://github.com/boboppie/kruschke-doing_bayesian_data_analysis/",
  "raw/master/2e/TherapeuticTouchData.csv"
)
tt_dat <- read.csv(tt_url)
# Get aggregated data by summing the counts
tt_agg <- tt_dat |>
  group_by(s) |>
  summarise(y = sum(y), # total number of correct
            n = n())
# Plot proportion correct distribution
p1 <- ggplot(tt_agg, aes(x = y / n)) +
  geom_histogram(binwidth = .1) +
  labs(x = "Proportion Correct")
```

### 7.1.3 Stan Code

```
data {
  int<lower=0> J; // number of clusters (e.g., studies, persons)
  array[J] int y; // number of "1"s in each cluster
```

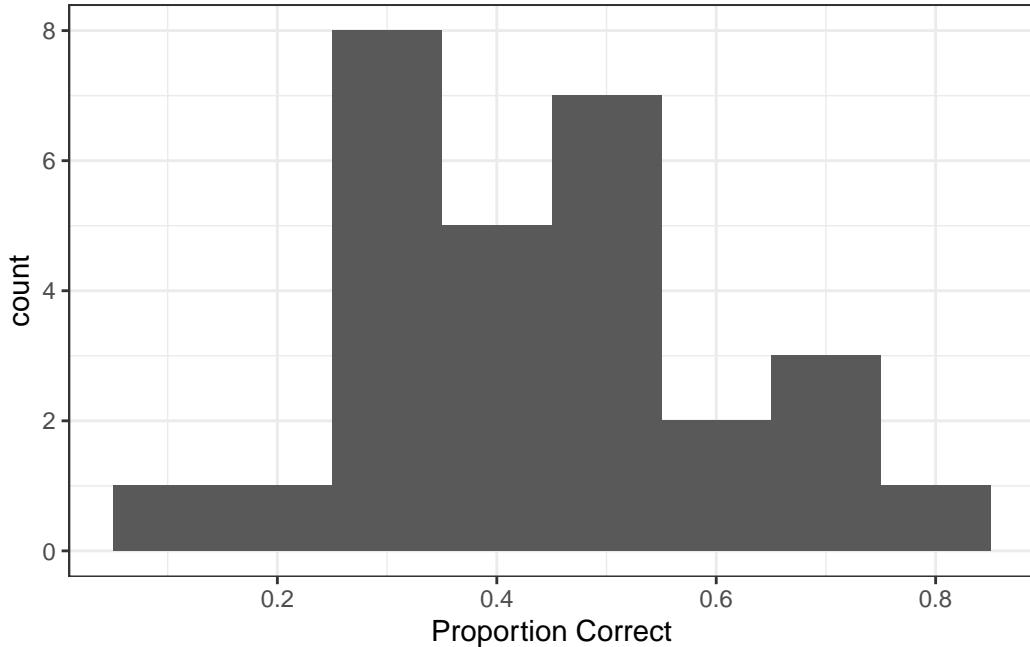


Figure 7.2: Distribution of proportions of correct responses across participants.

```

array[J] int N; // sample size for each cluster
}
parameters {
    // cluster-specific probabilities
    vector<lower=0, upper=1>[J] theta;
    real<lower=0, upper=1> mu; // overall mean probability
    real<lower=0> kappa; // overall concentration
}
model {
    y ~ binomial(N, theta); // each observation is binomial
    // Priors
    theta ~ beta_proportion(mu, kappa);
    mu ~ beta(1.5, 1.5); // weak prior
    kappa ~ gamma(.1, .1); // prior recommended by Kruschke
}
generated quantities {
    // Prior and posterior predictive
    real<lower=0, upper=1> prior_mu = beta_rng(1.5, 1.5);
    real<lower=0> prior_kappa = gamma_rng(.1, .1);
    vector<lower=0, upper=1>[J] prior_theta;
}

```

```

for (j in 1:J) {
  prior_theta[j] = beta_proportion_rng(prior_mu, prior_kappa);
}
array[J] int prior_ytilde = binomial_rng(N, prior_theta);
// Posterior predictive
array[J] int ytilde = binomial_rng(N, theta);
}

```

```
hbin_mod <- cmdstan_model("stan_code/hierarchical-binomial.stan")
```

### 7.1.4 Prior predictive

You can use Stan to sample the prior and obtain the prior predictive distribution; here, I show how to do it in R, with a  $\text{Gamma}(0.01, 0.01)$  prior on  $\kappa$ :

```

set.seed(1706)
plist <- vector("list", 12L)
plist[[1]] <- p1 +
  labs(x = "Observed data") +
  theme(axis.title.x = element_text(color = "red"))
num_subjects <- 28
for (s in 1:11) {
  # Get prior values of mu and kappa
  mu_s <- rbeta(1, shape1 = 1.5, shape2 = 1.5)
  kappa_s <- rgamma(1, shape = 0.01, rate = 0.01)
  # Generate theta
  theta <- rbeta(num_subjects,
                 shape1 = mu_s * kappa_s,
                 shape2 = (1 - mu_s) * kappa_s)
  # Generate data
  new_y <- rbinom(num_subjects, size = tt_agg$n, prob = theta)
  plist[[s + 1]] <-
    p1 %+%
    mutate(tt_agg, y = new_y) +
    labs(x = paste("Simulated data", s)) +
    theme(axis.title.x = element_text(color = "black"))
}
gridExtra::grid.arrange(grobs = plist, nrow = 3)

```

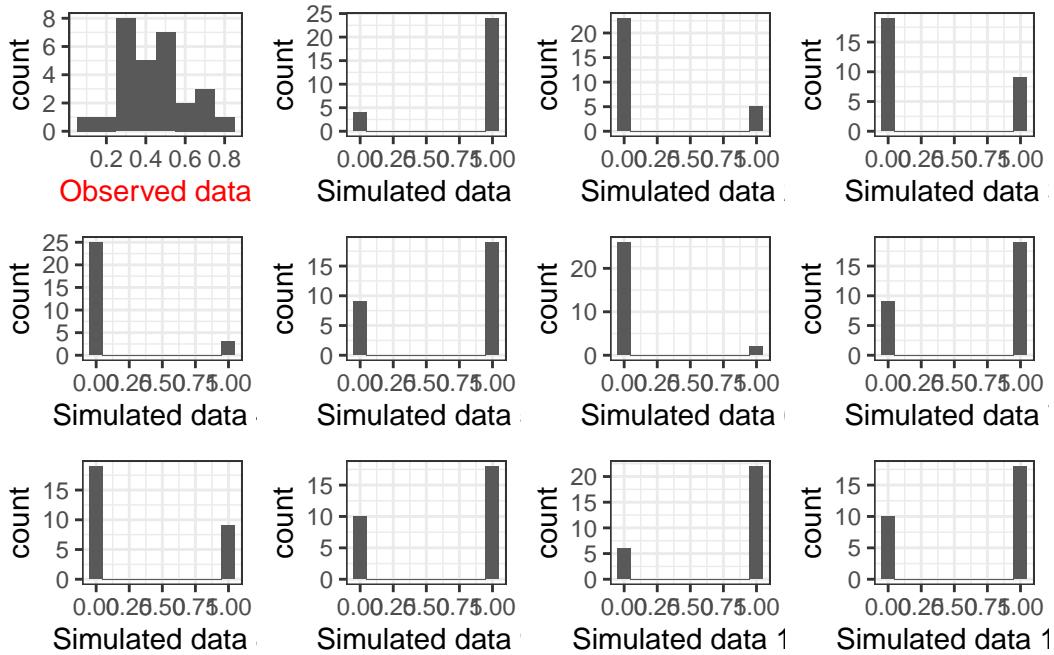


Figure 7.3: Prior predictive distribution.

The prior on  $\kappa$  is not very realistic because it pushes the bias to either 0 or 1. Using something like  $\text{Gamma}(0.1, 0.1)$  or  $\text{Gamma}(2, 0.01)$  may be more reasonable (you can try it out yourself).

### 7.1.5 Calling Stan

```
tt_fit <- hbin_mod$sample(
  data = list(J = nrow(tt_agg),
              y = tt_agg$y,
              N = tt_agg$n),
  seed = 1716, # for reproducibility
  refresh = 1000
)
```

Running MCMC with 4 sequential chains...

```
Chain 1 Iteration: 1 / 2000 [ 0%] (Warmup)
Chain 1 Iteration: 1000 / 2000 [ 50%] (Warmup)
Chain 1 Iteration: 1001 / 2000 [ 50%] (Sampling)
```

```
Chain 1 Iteration: 2000 / 2000 [100%] (Sampling)
Chain 1 finished in 0.1 seconds.
Chain 2 Iteration: 1 / 2000 [  0%] (Warmup)
Chain 2 Iteration: 1000 / 2000 [ 50%] (Warmup)
Chain 2 Iteration: 1001 / 2000 [ 50%] (Sampling)
```

Chain 2 Informational Message: The current Metropolis proposal is about to be rejected because it would violate one or more constraints.

```
Chain 2 Exception: beta_proportion_lpdf: Location parameter is 1, but must be less than 1.000000e+00
```

Chain 2 If this warning occurs sporadically, such as for highly constrained variable types like integers or ordered factors, try pre-transforming it to approximate values.

Chain 2 but if this warning occurs often then your model may be either severely ill-conditioned or misspecified.

Chain 2

```
Chain 2 Iteration: 2000 / 2000 [100%] (Sampling)
Chain 2 finished in 0.1 seconds.
Chain 3 Iteration: 1 / 2000 [  0%] (Warmup)
Chain 3 Iteration: 1000 / 2000 [ 50%] (Warmup)
Chain 3 Iteration: 1001 / 2000 [ 50%] (Sampling)
Chain 3 Iteration: 2000 / 2000 [100%] (Sampling)
Chain 3 finished in 0.1 seconds.
Chain 4 Iteration: 1 / 2000 [  0%] (Warmup)
Chain 4 Iteration: 1000 / 2000 [ 50%] (Warmup)
Chain 4 Iteration: 1001 / 2000 [ 50%] (Sampling)
```

Chain 4 Informational Message: The current Metropolis proposal is about to be rejected because it would violate one or more constraints.

```
Chain 4 Exception: beta_proportion_lpdf: Location parameter is 0, but must be positive! (in
```

Chain 4 If this warning occurs sporadically, such as for highly constrained variable types like integers or ordered factors, try pre-transforming it to approximate values.

Chain 4 but if this warning occurs often then your model may be either severely ill-conditioned or misspecified.

Chain 4

```

Chain 4 Iteration: 2000 / 2000 [100%]  (Sampling)
Chain 4 finished in 0.2 seconds.

```

```

All 4 chains finished successfully.
Mean chain execution time: 0.1 seconds.
Total execution time: 1.0 seconds.

```

You can explore the convergence and posterior distributions using the `shinystan` package

```
shinystan::launch_shinystan(tt_fit)
```

### 7.1.6 Table of coefficients

```

tt_fit$summary(c("theta", "mu", "kappa")) |>
  # Use `knitr::kable()` for tabulation
  knitr::kable(digits = 2)

```

Table 7.1: Posterior summary of hierarchical binomial model for the therapeutic touch example.

variable	mean	median	sd	mad	q5	q95	rhat	ess_bulk	ess_tail
theta[1]	0.31	0.31	0.10	0.10	0.15	0.47	1	3268.39	2404.91
theta[2]	0.35	0.35	0.10	0.10	0.19	0.51	1	3741.46	2422.69
theta[3]	0.39	0.39	0.10	0.10	0.23	0.55	1	4744.99	2794.69
theta[4]	0.39	0.39	0.10	0.10	0.23	0.55	1	4859.60	3056.80
theta[5]	0.39	0.39	0.10	0.10	0.22	0.55	1	4817.40	2485.43
theta[6]	0.39	0.38	0.10	0.10	0.23	0.55	1	5260.92	2493.09
theta[7]	0.39	0.38	0.10	0.10	0.23	0.55	1	5586.84	2968.73
theta[8]	0.39	0.39	0.10	0.10	0.23	0.55	1	4926.08	3008.76
theta[9]	0.39	0.39	0.10	0.10	0.23	0.55	1	4802.85	2558.79
theta[10]	0.39	0.39	0.10	0.10	0.22	0.55	1	4743.85	2910.25
theta[11]	0.43	0.42	0.10	0.09	0.27	0.59	1	5041.77	3066.93
theta[12]	0.42	0.42	0.09	0.10	0.27	0.58	1	6055.56	2896.89
theta[13]	0.43	0.42	0.10	0.10	0.27	0.59	1	4898.48	3112.24
theta[14]	0.43	0.42	0.10	0.10	0.27	0.59	1	5909.48	3065.15
theta[15]	0.43	0.42	0.10	0.10	0.27	0.59	1	6304.56	3049.53
theta[16]	0.46	0.46	0.10	0.10	0.31	0.62	1	5642.01	3230.67
theta[17]	0.46	0.46	0.10	0.10	0.30	0.63	1	5517.58	3108.00
theta[18]	0.46	0.46	0.10	0.10	0.30	0.63	1	5622.46	2837.76

Table 7.1: Posterior summary of hierarchical binomial model for the therapeutic touch example.

variable	mean	median	sd	mad	q5	q95	rhat	ess_bulk	ess_tail
theta[19]	0.46	0.46	0.10	0.09	0.31	0.63	1	6228.82	3062.79
theta[20]	0.46	0.46	0.10	0.10	0.30	0.63	1	4758.62	2538.88
theta[21]	0.46	0.46	0.10	0.10	0.30	0.62	1	5922.64	2921.96
theta[22]	0.46	0.46	0.10	0.10	0.31	0.62	1	5218.69	3180.07
theta[23]	0.50	0.50	0.10	0.10	0.34	0.67	1	5424.13	2936.72
theta[24]	0.50	0.50	0.10	0.10	0.34	0.67	1	5321.61	2733.91
theta[25]	0.54	0.54	0.10	0.10	0.38	0.71	1	3861.44	2248.48
theta[26]	0.54	0.54	0.10	0.10	0.37	0.72	1	3808.59	2430.69
theta[27]	0.54	0.54	0.10	0.10	0.38	0.71	1	3679.86	2556.84
theta[28]	0.58	0.57	0.10	0.11	0.41	0.75	1	3005.40	2460.66
mu	0.44	0.44	0.04	0.04	0.38	0.50	1	2482.91	2932.61
kappa	18.65	16.37	9.63	7.92	7.49	36.92	1	992.72	1793.53

### 7.1.7 Posterior Predictive Check

With hierarchical models, there are two types of posterior predictive distributions:

1. *Same participants but new observations.* We can use the model to generate new observations, but the individual parameters (e.g.,  $\theta_j$ ) remain the same. In our example, this would be the situation where we ask **the same 28 participants** to each do 10 more trials.
2. *New participants and new observations.* We can use the model to generate new observations with new parameters from the higher-order distribution (e.g., the Beta distribution for  $\theta_j$ ). In our example, this would be the situation where we ask **a new set of 28 participants** to each do 10 more trials.

In our Stan code, I used (1) to check the fit of the observed data. However, (2) can be helpful when one wants to use the model to make predictions of future data, as future data are unlikely to concern exactly the same participants.

```
# The bayesplot::ppc_bars() unfortunately contains a bug
# (https://github.com/stan-dev/bayesplot/issues/266) at
# the time when I wrote this, so I'll use my own code.
# tt_fit$draws("ytilde", format = "draws_matrix") |>
#   ppc_bars(y = tt_agg$y)
yrep <- tt_fit$draws("ytilde", format = "draws_matrix")
yrep_intervals <- apply(
```

```

yrep, MARGIN = 1, FUN = \((x) table(factor(x, levels = 0:10))
) |>
apply(MARGIN = 1, FUN = \((x) {
  c(
    lo = quantile(x, .05)[[1]],
    me = median(x),
    hi = quantile(x, .95)[[1]]
  )
})
data.frame(
  y = table(factor(tt_agg$y, levels = 0:10))
) |>
  setNames(c("x", "y")) |>
  cbind(t(yrep_intervals)) |>
  ggplot(aes(x = x, y = y)) +
  geom_col(alpha = 0.5) +
  geom_pointrange(aes(y = me, ymin = lo, ymax = hi)) +
  labs(y = "count", x = NULL)

```

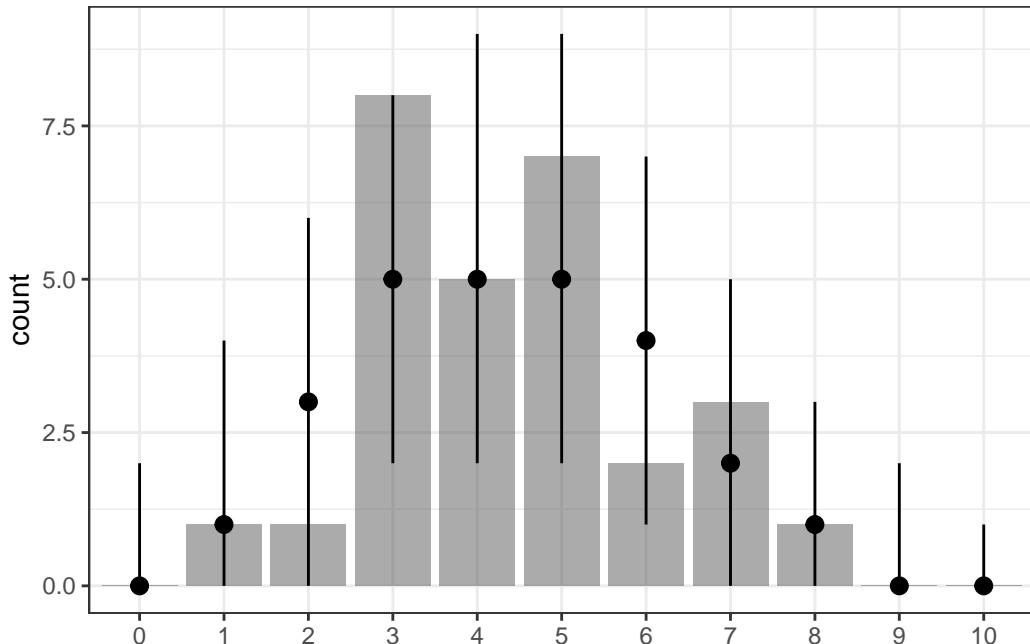


Figure 7.4: Posterior predictive check. The bar graph shows the observed data, and the error bars show the 90% posterior predictive interval in replicated data. The dots are the medians in the posterior predictive distribution.

### 7.1.8 Derived coefficients

One nice thing about MCMC is that it is straightforward to obtain posterior distributions that are functions of the parameters. For example, even though we only sampled from the posteriors of the  $\theta$ s, we can ask questions like whether there is evidence for a nonzero *difference* in  $\theta$  between person 1 and person 28.

```
tt_fit$draws() |>
  mutate_variables(
    theta1_minus14 = `theta[1]` - `theta[14]`,
    theta1_minus28 = `theta[1]` - `theta[28]`,
    theta14_minus28 = `theta[14]` - `theta[28]`
  ) |>
  subset(variable = c("theta1_minus14", "theta1_minus28",
                      "theta14_minus28")) |>
  summarise_draws() |>
  knitr::kable(digits = 2)
```

variable	mean	median	sd	mad	q5	q95	rhat	ess_bulk	ess_tail
theta1_minus14	-0.11	-0.11	0.13	0.13	-0.34	0.09	1	4950.80	2632.63
theta1_minus28	-0.27	-0.26	0.15	0.16	-0.53	-0.03	1	2312.44	3115.27
theta14_minus28	-0.15	-0.15	0.14	0.14	-0.38	0.06	1	4309.81	3013.97

### 7.1.9 Conclusion

As 0.5 is included in the 95% CI of  $\theta$  for all participants, there is insufficient evidence that people can sense “therapeutic touch.”

### 7.1.10 Shrinkage

```
mcmc_intervals(tt_fit$draws(),
  # plot only parameters matching "theta"
  regex_pars = "^\theta") +
  geom_point(
  data = tibble(
    parameter = paste0("theta[", 1:28, "]"),
    x = tt_agg$y / tt_agg$n
  ),
```

```

    aes(x = x, y = parameter),
    col = "red"
) +
xlim(0, 1)

```

Scale for x is already present.

Adding another scale for x, which will replace the existing scale.

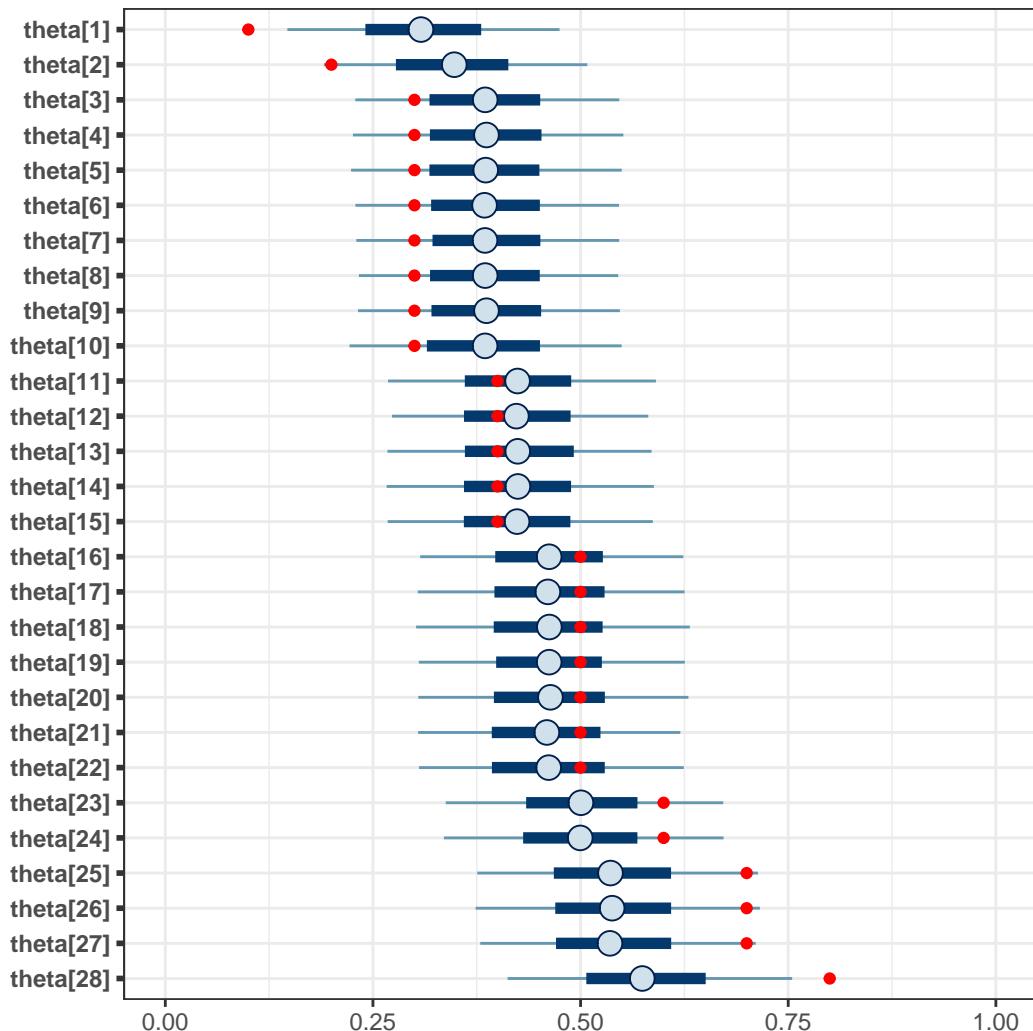


Figure 7.5: Shrinkage effect in a hierarchical model.

As can be seen, the posterior distributions are closer to the center than the data (in red). This

**pooling** results from the belief that the participants have something in common.

### 7.1.11 Multiple Comparisons?

Another benefit of a Bayesian hierarchical model is that *you don't need to worry about multiple comparisons*. There are multiple angles on why this is the case, but the basic answer is that the use of common prior distributions builds in the prior belief that the clusters/groups are likely to be equal. See discussion [here](#) and [here](#).

## 7.2 Hierarchical Normal Model

### 7.2.1 Eight Schools Example

This is a classic data set first analyzed by [Rubin \(1981\)](#). It is also the example used in the [RStan Getting Started](#) page. The data contains the effect of coaching from randomized experiments in eight schools. The numbers shown (labelled as  $y$ ) are the mean difference (i.e., effect size) in performance between the treatment and control groups on SAT-V scores.

```
schools_dat <- list(  
  J = 8,  
  y = c(28, 8, -3, 7, -1, 1, 18, 12),  
  sigma = c(15, 10, 16, 11, 9, 11, 10, 18)  
)
```

In the above data, some numbers are positive, and some are negative. Because the sample sizes are different, the data also contained the standard errors (labelled as  $\sigma$ ) of the effect sizes. Generally speaking, a larger sample size corresponds to a smaller standard error. The research question is

1. What is the average treatment effect of coaching?
2. Are the treatment effects similar across schools?

### 7.2.2 Model

Model:

$$\begin{aligned}d_j &\sim N(\theta_j, s_j) \\ \theta_j &\sim N(\mu, \tau)\end{aligned}$$

Prior:

$$\begin{aligned}\mu &\sim N(0, 100) \\ \tau &\sim t_4^+(0, 100)\end{aligned}$$

Given the SAT score range, it is unlikely that a coaching program will improve scores by 100 or so, so we use a prior of  $\mu \sim N(0, 100)$  and  $\tau \sim t_4^+(0, 100)$ .

 The model above is the same as one used in a *random-effect meta-analysis*. See [this paper](#) for an introduction.

### 7.2.3 Non-Centered Parameterization

The hierarchical model is known to create issues in MCMC sampling, such that the posterior draws tend to be highly correlated even with more advanced techniques like HMC. One way to alleviate that is to reparameterize the model using what is called the **non-centered parameterization**. The basic idea is that, instead of treating the  $\theta$ s as parameters, one uses the **standardized deviation** from the mean to be parameters. You can think about it as converting the  $\theta$ s into  $z$  scores, and then sample the  $z$  scores instead of the original  $\theta$ s.

Model:

$$\begin{aligned} d_j &\sim N(\theta_j, s_j) \\ \theta_j &= \mu + \tau \eta_j \\ \eta_j &\sim N(0, 1) \end{aligned}$$

```
data {
    int<lower=0> J;                      // number of schools
    vector[J] y;                          // estimated treatment effects
    vector<lower=0>[J] sigma;             // s.e. of effect estimates
}
parameters {
    real mu;                            // overall mean
    real<lower=0> tau;                  // between-school SD
    vector[J] eta;                     // standardized deviation (z score)
}
transformed parameters {
    vector[J] theta;
    theta = mu + tau * eta;           // non-centered parameterization
}
model {
    eta ~ std_normal();                // same as eta ~ normal(0, 1);
    y ~ normal(theta, sigma);
    // priors
    mu ~ normal(0, 100);
    tau ~ student_t(4, 0, 100);
}
```

```
hnorm_mod <- cmdstan_model("stan_code/hierarchical-normal.stan")
```

```
fit <- hnorm_mod$sample(  
  data = schools_dat,  
  seed = 1804, # for reproducibility  
  refresh = 1000,  
  adapt_delta = 0.9 # to improve convergence  
)
```

Running MCMC with 4 sequential chains...

```
Chain 1 Iteration: 1 / 2000 [ 0%] (Warmup)  
Chain 1 Iteration: 1000 / 2000 [ 50%] (Warmup)  
Chain 1 Iteration: 1001 / 2000 [ 50%] (Sampling)  
Chain 1 Iteration: 2000 / 2000 [100%] (Sampling)  
Chain 1 finished in 0.0 seconds.  
Chain 2 Iteration: 1 / 2000 [ 0%] (Warmup)  
Chain 2 Iteration: 1000 / 2000 [ 50%] (Warmup)  
Chain 2 Iteration: 1001 / 2000 [ 50%] (Sampling)  
Chain 2 Iteration: 2000 / 2000 [100%] (Sampling)  
Chain 2 finished in 0.0 seconds.  
Chain 3 Iteration: 1 / 2000 [ 0%] (Warmup)  
Chain 3 Iteration: 1000 / 2000 [ 50%] (Warmup)  
Chain 3 Iteration: 1001 / 2000 [ 50%] (Sampling)  
Chain 3 Iteration: 2000 / 2000 [100%] (Sampling)  
Chain 3 finished in 0.0 seconds.  
Chain 4 Iteration: 1 / 2000 [ 0%] (Warmup)  
Chain 4 Iteration: 1000 / 2000 [ 50%] (Warmup)  
Chain 4 Iteration: 1001 / 2000 [ 50%] (Sampling)  
Chain 4 Iteration: 2000 / 2000 [100%] (Sampling)  
Chain 4 finished in 0.0 seconds.
```

All 4 chains finished successfully.

Mean chain execution time: 0.0 seconds.

Total execution time: 0.5 seconds.

Treatment effect estimates of individual schools ( $\theta$ ), average treatment effect ( $\mu$ ), and treatment effect heterogeneity ( $\tau$ ).

```
fit$summary(c("theta", "mu", "tau")) |>  
  knitr::kable(digits = 2)
```

variable	mean	median	sd	mad	q5	q95	rhat	ess_bulk	ess_tail
theta[1]	11.70	10.60	8.69	7.38	-0.20	27.60	1.00	3475.69	3347.72
theta[2]	7.98	7.93	6.31	5.82	-2.18	18.34	1.00	5870.22	3353.39
theta[3]	6.13	6.77	7.97	6.50	-7.14	17.85	1.00	4318.80	2941.48
theta[4]	7.69	7.75	6.75	6.15	-3.43	18.53	1.00	5452.05	3268.65
theta[5]	5.22	5.53	6.22	5.82	-5.64	14.70	1.00	4118.93	3165.15
theta[6]	6.15	6.52	6.87	6.15	-5.19	16.86	1.00	4526.45	3352.39
theta[7]	10.80	10.14	6.78	6.32	0.88	23.09	1.00	4445.48	3533.95
theta[8]	8.55	8.33	7.74	6.42	-3.59	21.11	1.00	4978.08	3232.97
mu	7.95	8.00	5.26	4.97	-0.43	16.20	1.00	2917.45	1987.36
tau	6.75	5.31	5.91	4.84	0.55	17.53	1.01	1316.86	1673.79

On average, based on the 90% CI, coaching seemed to improve SAT-V by -0.43 to 16.2 points. There was substantial heterogeneity across schools.

We can also get the probability that the treatment effect was  $> 0$ :

```
# Obtain draws
draws_mu <- fit$draws("mu", format = "draws_matrix")
mean(draws_mu > 0)
```

[1] 0.93975

Here are the individual-school treatment effects:

```
mcmc_areas_ridges(fit$draws(), regex_pars = "theta")
```

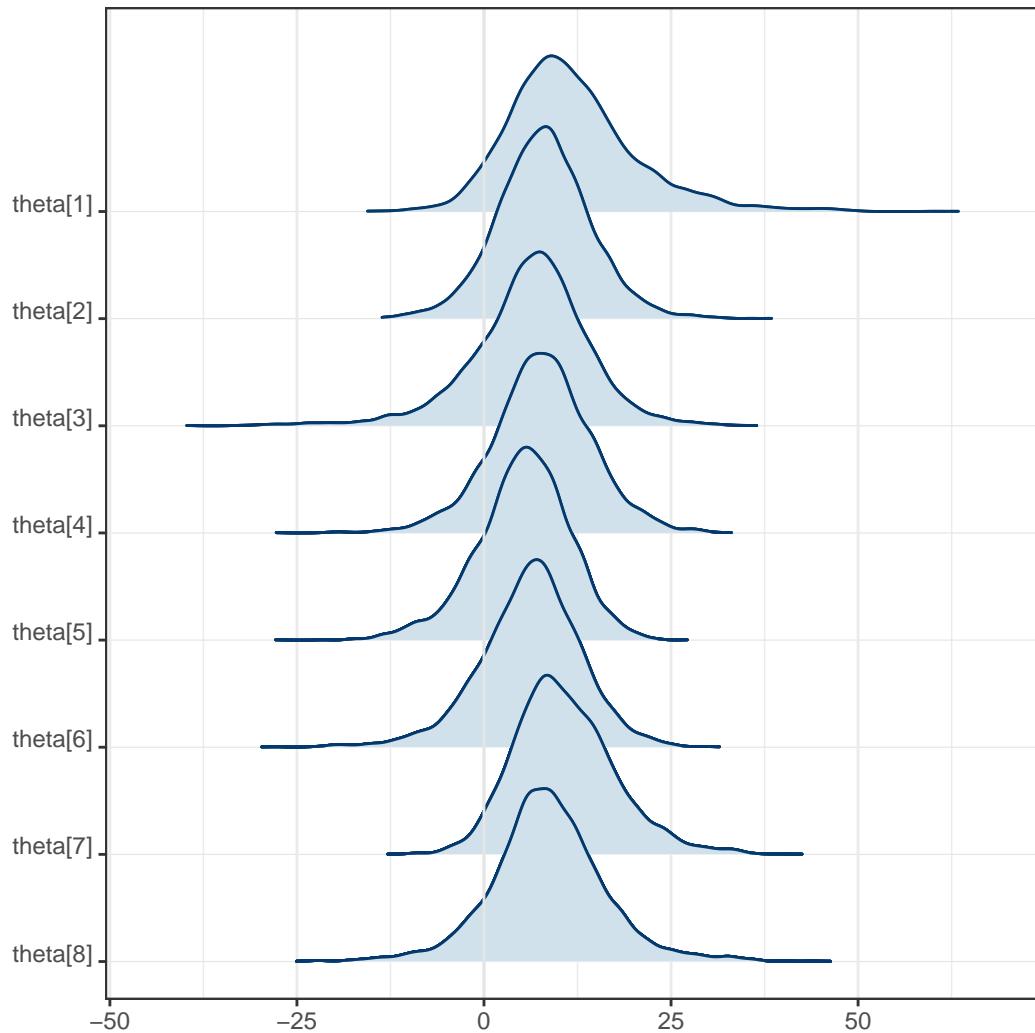


Figure 7.6: Posterior distribution of the true effect size in the eight schools example.

#### 7.2.4 Prediction Interval

Posterior distribution of the true effect size of a new study,  $\tilde{\theta}$

```
# Prediction Interval
# (can also be done in Stan, as in the previous example)
fit$draws(c("mu", "tau")) |>
```

```

  mutate_variables(
    theta_tilde = rnorm(4000, mean = mu, sd = tau)) |>
  summarise_draws() |>
  knitr::kable(digits = 2)

```

variable	mean	median	sd	mad	q5	q95	rhat	ess_bulk	ess_tail
mu	7.95	8.00	5.26	4.97	-0.43	16.20	1.00	2917.45	1987.36
tau	6.75	5.31	5.91	4.84	0.55	17.53	1.01	1316.86	1673.79
theta_tilde	8.03	7.87	10.19	7.00	-6.69	22.89	1.00	3947.08	3042.56

The posterior interval for  $\tilde{\theta}$  indicates a range of the treatment effect for a new study.

## **Part V**

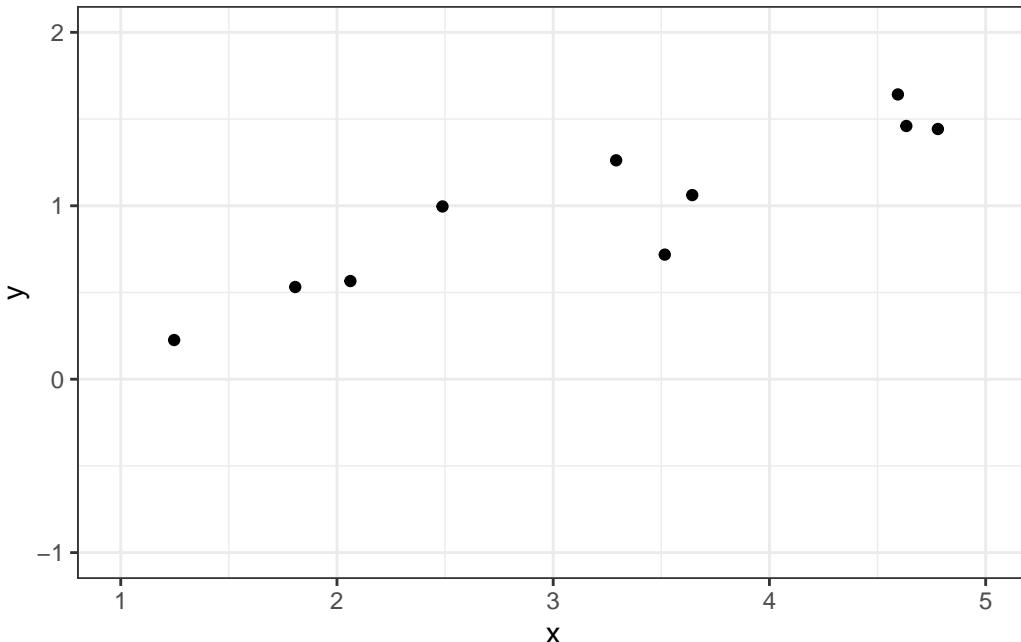
### **Week 5–6**

# 8 Linear Models

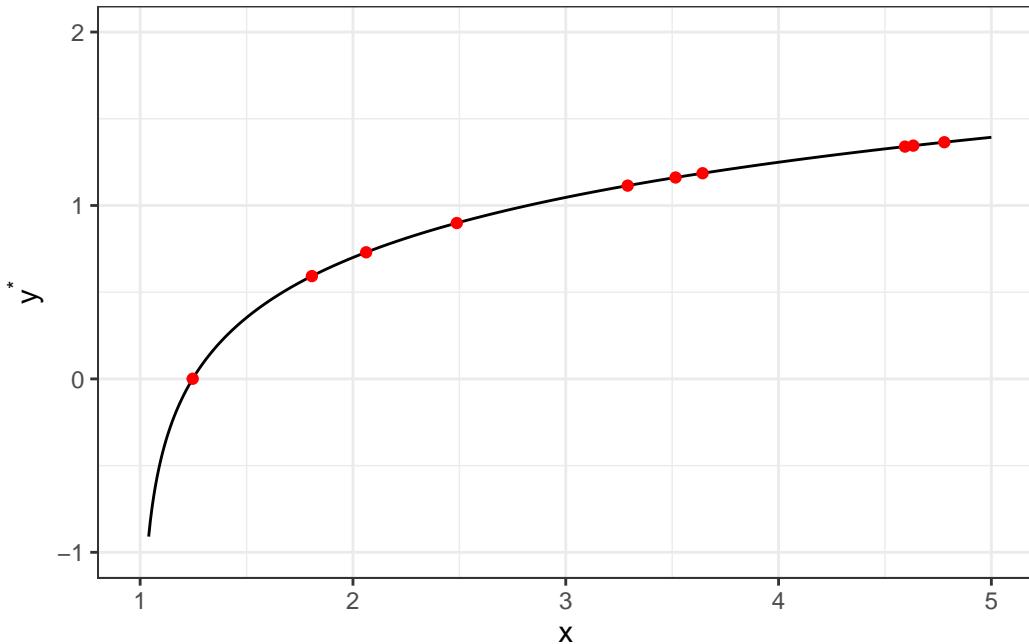
## 8.1 What is Regression?

Regression is a class of statistical techniques to understand the relationship between an outcome variable (also called a criterion/response/dependent variable) and one or more predictor variables (also called explanatory/independent variables). For example, if we have the following scatter plot between two variables ( $Y$  and  $X$ ):

```
set.seed(1)
x <- round(runif(10, 1, 5), 3)
y <- 0.7 + 0.5 * log(x - 1) + rnorm(10, sd = 0.2)
df <- data.frame(x, y)
ggplot(df, aes(x, y)) +
  geom_point() +
  xlim(1, 5) +
  ylim(-1, 2)
```



We want to find some pattern in this relationship. In conventional regression, we model the conditional distribution of  $Y$  given  $X$ ,  $P(Y | X)$ , by separating the outcome variable  $Y$  into (a) a *systematic component* that depends on the predictor, and (b) a *random/probabilistic component* that does not depend on the predictor. For example, we can start with a systematic component which only depends on the predictor:



As you can see, all the red dots fall exactly on the curve in the graph above, meaning that as long as one knows the  $X$  value, one can predict the  $Y$  value with 100% accuracy. We can thus write  $Y^* = f(X)$  (where  $Y^*$  is the systematic component of  $Y$ ).

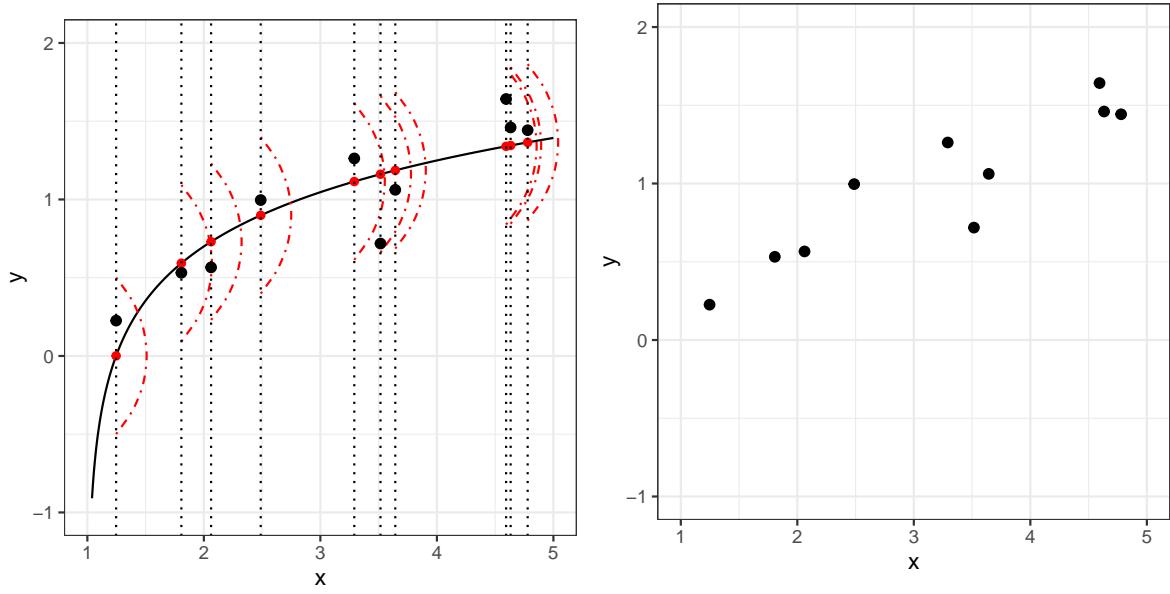
However, in almost all scientific inquiries, one can never predict with 100% certainty (e.g., there are measurement errors and college randomness in physics). The uncertainty stems from the fact that we rarely measure all the factors that determine  $Y$ , and there are genuinely random things (as in quantum physics). Therefore, we need to expand our model to incorporate this randomness by adding a probabilistic component. Therefore, instead of saying that  $Y$  depends just on  $X$ , we say  $Y$  is random, but the information about  $X$  provides information about how  $Y$  is distributed. In regression, one studies the conditional distribution  $P(Y | X)$  such that the conditional expectation,  $E(Y | X)$ , is determined by  $X$ ; on top of the conditional expectation, we assume that the observed  $Y$  values scatter around the conditional expectations, like the graph on the left below:

```
ggplot(df, aes(x, yhat)) +
  stat_function(fun = function(x) 0.7 + 0.5 * log(x - 1), n = 501) +
  geom_point(col = "red") +
```

```

xlim(1, 5) +
ylim(-1, 2) +
ylab("y") +
geom_curve(aes(x = x, y = yhat + 0.5, xend = x, yend = yhat - 0.5),
curvature = -0.4, col = "red", linetype = "dotdash")
) +
geom_vline(aes(xintercept = x), linetype = "dotted") +
geom_point(aes(x, y), size = 2)
ggplot(df, aes(x, y)) +
geom_point(size = 2) +
xlim(1, 5) +
ylim(-1, 2)

```



(a) With the true underlying relationship shown      (b) With the true underlying relationship hidden,  
as is always the case in real life

Figure 8.1: Sample regression function.

We can write the systematic part as:

$$E(Y | X) = f(X; \beta_1, \beta_2, \dots),$$

where  $\beta_1, \beta_2, \dots$  are the parameters for some arbitrary function  $f(\cdot)$ . The random part is about  $P(Y | X)$ , which can take some arbitrary distributions. In reality, even if such a model holds,

we do not know what  $f(\cdot)$  and the true distribution of  $Y | X$  are, as we only have data like those illustrated in the graph on the right above.

## 8.2 Linear Regression

The linear regression model assumes that

- (a) the function for the systematic component,  $f(\cdot)$ , is a linear function (in the  $\beta$ s),
- (b)  $Y | X$  is normally distributed, and
- (c)  $Y_i$ 's are conditionally exchangeable given  $X$  with equal variance  $\sigma^2$  (which can be relaxed).

Under these conditions, we have a model

$$\begin{aligned} Y_i &\sim N(\mu_i, \sigma) \\ \mu_i &= \eta_i \\ \eta_i &= \beta_0 + \beta_1 X_i \end{aligned}$$

With parameters:

- $\beta_0$ : regression intercept; predicted  $Y$  value for observations with  $X = 0$
- $\beta_1$ : regression slope/coefficient; predicted difference in  $Y$  for one unit difference in  $X$
- $\sigma$ : standard deviation of prediction error; roughly speaking, the margin of error in prediction

### 8.2.1 Example

We will use an example based on the “bread and peace” model by political scientist Douglas Hibbs, which can be used to forecast the U.S. presidential election outcome based on some weighted metric of personal income growth. The example is taken from chapter 7 of the book [Regression and Other Stories](#).<sup>1</sup>

Figure 8.2 is a graph showing the data from 1952 to 2012, with one variable being personal income growth in the four years prior to an election, and the other being the vote share (in %) of the incumbent's party.

---

<sup>1</sup>See <https://douglas-hibbs.com/background-information-on-bread-and-peace-voting-in-us-presidential-elections/> for more information on the “bread and peace” model.

```

# Economy and elections data
if (!file.exists("data/hibbs.dat")) {
  download.file(
    "https://github.com/avehtari/ROS-Examples/raw/master/ElectionsEconomy/data/hibbs.dat",
    "data/hibbs.dat"
  )
}
hibbs <- read.table("data/hibbs.dat", header = TRUE)

ggplot(hibbs, aes(x = growth, y = vote, label = year)) +
  geom_point() +
  ggrepel::geom_text_repel() +
  labs(x = "Average recent growth in personal income",
       y = "Incumbent party's vote share (%)")

```

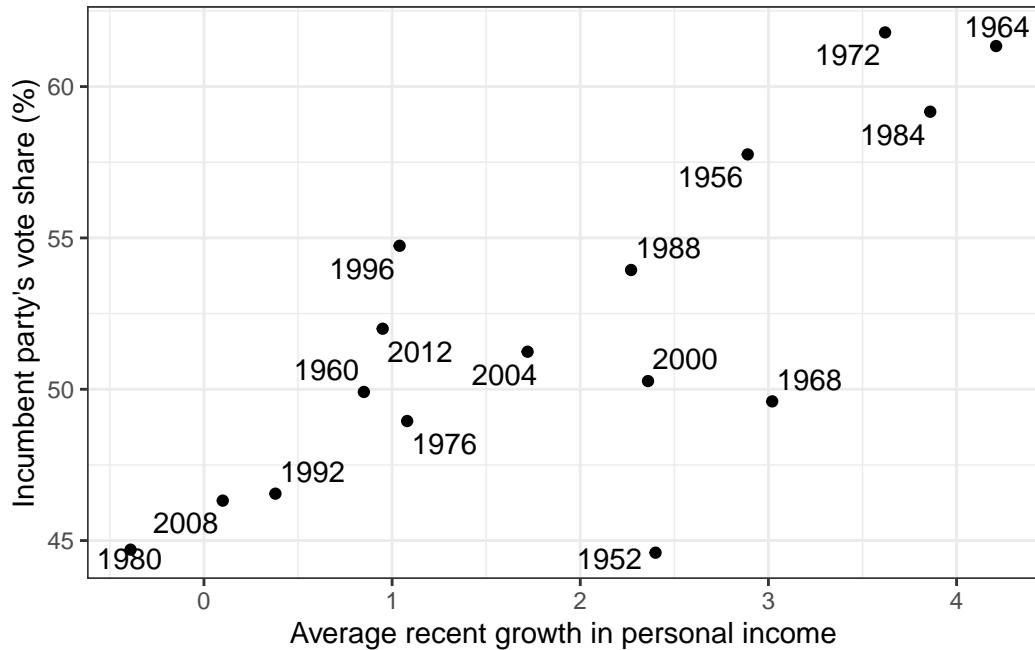


Figure 8.2: Scatter plot of personal income growth and vote share.

### 8.3 Model and Priors

Model:

$$\text{vote}_i \sim N(\mu_i, \sigma)$$

$$\mu_i = \beta_0 + \beta_1 \text{growth}_i$$

$\sigma$ : SD (margin) of prediction error

Priors:

$$\beta_0 \sim N(45, 10)$$

$$\beta_1 \sim N(0, 10)$$

$$\sigma \sim t_4^+(0, 5)$$

## 8.4 Model Fitting With Stan

Once you've written the model, it's straightforward to code the model in Stan to perform MCMC sampling, like the code below.

```
data {
  int<lower=0> N; // number of observations
  vector[N] y; // outcome;
  vector[N] x; // predictor;
  int<lower=0,upper=1> prior_only; // whether to sample prior only
}
parameters {
  real beta0; // regression intercept
  real beta1; // regression coefficient
  real<lower=0> sigma; // SD of prediction error
}
model {
  // model
  if (!prior_only) {
    y ~ normal(beta0 + beta1 * x, sigma);
  }
  // prior
  beta0 ~ normal(45, 10);
  beta1 ~ normal(0, 10);
  sigma ~ student_t(4, 0, 5);
}
generated quantities {
  // Prior/posterior predictive
```

```
    array[N] real ytilde = normal_rng(beta0 + beta1 * x, sigma);  
}
```

```
linear_reg <- cmdstan_model("stan_code/linear_reg.stan")
```

#### 8.4.1 Prior Predictive

```
m1_prior <- linear_reg$sample(  
  data = list(N = nrow(hibbs),  
             y = hibbs$vote,  
             x = hibbs$growth,  
             prior_only = TRUE),  
  seed = 1227, # for reproducibility  
  refresh = 1000  
)
```

Running MCMC with 4 sequential chains...

```
Chain 1 Iteration: 1 / 2000 [ 0%] (Warmup)  
Chain 1 Iteration: 1000 / 2000 [ 50%] (Warmup)  
Chain 1 Iteration: 1001 / 2000 [ 50%] (Sampling)  
Chain 1 Iteration: 2000 / 2000 [100%] (Sampling)  
Chain 1 finished in 0.0 seconds.  
Chain 2 Iteration: 1 / 2000 [ 0%] (Warmup)  
Chain 2 Iteration: 1000 / 2000 [ 50%] (Warmup)  
Chain 2 Iteration: 1001 / 2000 [ 50%] (Sampling)  
Chain 2 Iteration: 2000 / 2000 [100%] (Sampling)  
Chain 2 finished in 0.0 seconds.  
Chain 3 Iteration: 1 / 2000 [ 0%] (Warmup)  
Chain 3 Iteration: 1000 / 2000 [ 50%] (Warmup)  
Chain 3 Iteration: 1001 / 2000 [ 50%] (Sampling)  
Chain 3 Iteration: 2000 / 2000 [100%] (Sampling)  
Chain 3 finished in 0.0 seconds.  
Chain 4 Iteration: 1 / 2000 [ 0%] (Warmup)  
Chain 4 Iteration: 1000 / 2000 [ 50%] (Warmup)  
Chain 4 Iteration: 1001 / 2000 [ 50%] (Sampling)  
Chain 4 Iteration: 2000 / 2000 [100%] (Sampling)  
Chain 4 finished in 0.0 seconds.
```

```
All 4 chains finished successfully.
Mean chain execution time: 0.0 seconds.
Total execution time: 0.5 seconds.
```

```
m1_prior$draws("ytilde", format = "matrix") |>
  ppc_ribbon(y = hibbs$vote, x = hibbs$growth,
             y_draw = "points") +
  labs(x = "Average recent growth in personal income",
       y = "Incumbent party's vote share (%)") +
  coord_cartesian(ylim = c(0, 100))
```

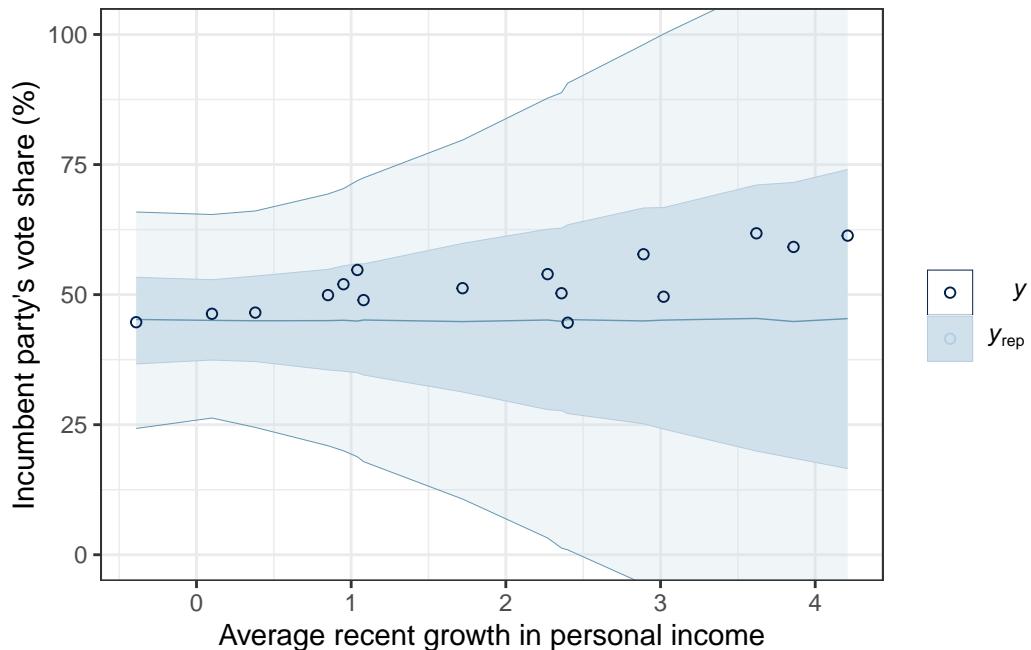


Figure 8.3: Prior predictive distribution of the linear regression model for vote share against personal income.

We can also visualize the prior regression lines based on the prior distributions:

```
prior_draws_beta <- m1_prior$draws(c("beta0", "beta1"), format = "data.frame")
ggplot(hibbs, aes(x = growth, y = vote, label = year)) +
  geom_point() +
  geom_abline(data = prior_draws_beta, aes(intercept = beta0, slope = beta1),
              linewidth = 0.1, alpha = 0.1) +
  ggrepel::geom_text_repel() +
```

```

  labs(x = "Average recent growth in personal income",
       y = "Incumbent party's vote share (%)") +
  coord_cartesian(ylim = c(0, 100))

```

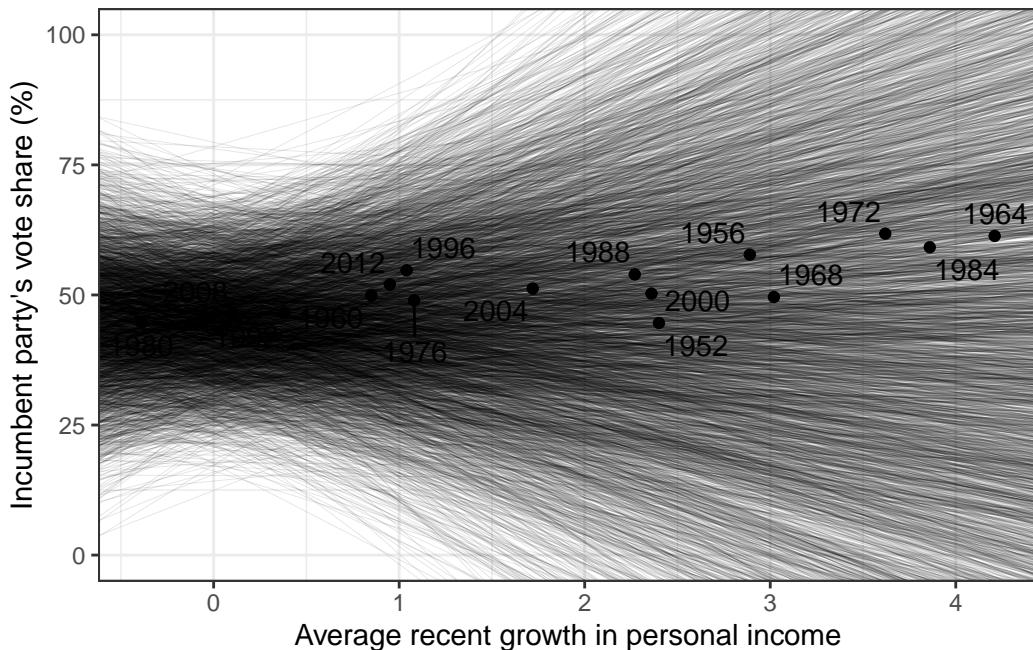


Figure 8.4: Prior predictive distribution of the regression lines.

#### 8.4.2 Results

We'll now fit the model (without `prior_only = TRUE`).

```

m1_post <- linear_reg$sample(
  data = list(N = nrow(hibbs),
              y = hibbs$vote,
              x = hibbs$growth,
              prior_only = FALSE),
  seed = 1227, # for reproducibility
  refresh = 1000
)

```

Running MCMC with 4 sequential chains...

```

Chain 1 Iteration: 1 / 2000 [  0%] (Warmup)
Chain 1 Iteration: 1000 / 2000 [ 50%] (Warmup)
Chain 1 Iteration: 1001 / 2000 [ 50%] (Sampling)
Chain 1 Iteration: 2000 / 2000 [100%] (Sampling)
Chain 1 finished in 0.0 seconds.
Chain 2 Iteration: 1 / 2000 [  0%] (Warmup)
Chain 2 Iteration: 1000 / 2000 [ 50%] (Warmup)
Chain 2 Iteration: 1001 / 2000 [ 50%] (Sampling)
Chain 2 Iteration: 2000 / 2000 [100%] (Sampling)
Chain 2 finished in 0.0 seconds.
Chain 3 Iteration: 1 / 2000 [  0%] (Warmup)
Chain 3 Iteration: 1000 / 2000 [ 50%] (Warmup)
Chain 3 Iteration: 1001 / 2000 [ 50%] (Sampling)
Chain 3 Iteration: 2000 / 2000 [100%] (Sampling)
Chain 3 finished in 0.0 seconds.
Chain 4 Iteration: 1 / 2000 [  0%] (Warmup)
Chain 4 Iteration: 1000 / 2000 [ 50%] (Warmup)
Chain 4 Iteration: 1001 / 2000 [ 50%] (Sampling)
Chain 4 Iteration: 2000 / 2000 [100%] (Sampling)
Chain 4 finished in 0.0 seconds.

```

All 4 chains finished successfully.

Mean chain execution time: 0.0 seconds.

Total execution time: 0.5 seconds.

```

m1_summ <- m1_post$summary(c("beta0", "beta1", "sigma"))
# Use `knitr::kable()` for tabulation
knitr::kable(m1_summ, digits = 2)

```

Table 8.1: Posterior summary of linear regression model.

variable	mean	median	sd	mad	q5	q95	rhat	ess_bulk	ess_tail
beta0	46.25	46.26	1.73	1.66	43.42	49.06	1	1582.97	1590.35
beta1	3.05	3.06	0.75	0.71	1.83	4.26	1	1605.18	1745.88
sigma	4.03	3.91	0.78	0.70	2.97	5.48	1	2047.68	2056.33

The parameter estimates are shown in Table 13.3. Here's a paragraph for the results:

💡 The model predicts that when personal income growth is 0, the vote share for the incumbent party is 46%, 90% CI [43%, 49%]. A 1-unit difference in personal income growth corresponds to a difference in vote share by 3 percentage points, 90% CI [1.8, 4.3].

### 8.4.3 Convergence

We will talk about convergence more in a future week. For now, you want to see that

- The chains mix well
- the rank histograms are close to uniform distributions

```
m1_post_draws <- m1_post$draws(c("beta0", "beta1", "sigma"))
mcmc_trace(m1_post_draws)
mcmc_rank_hist(m1_post_draws)
```

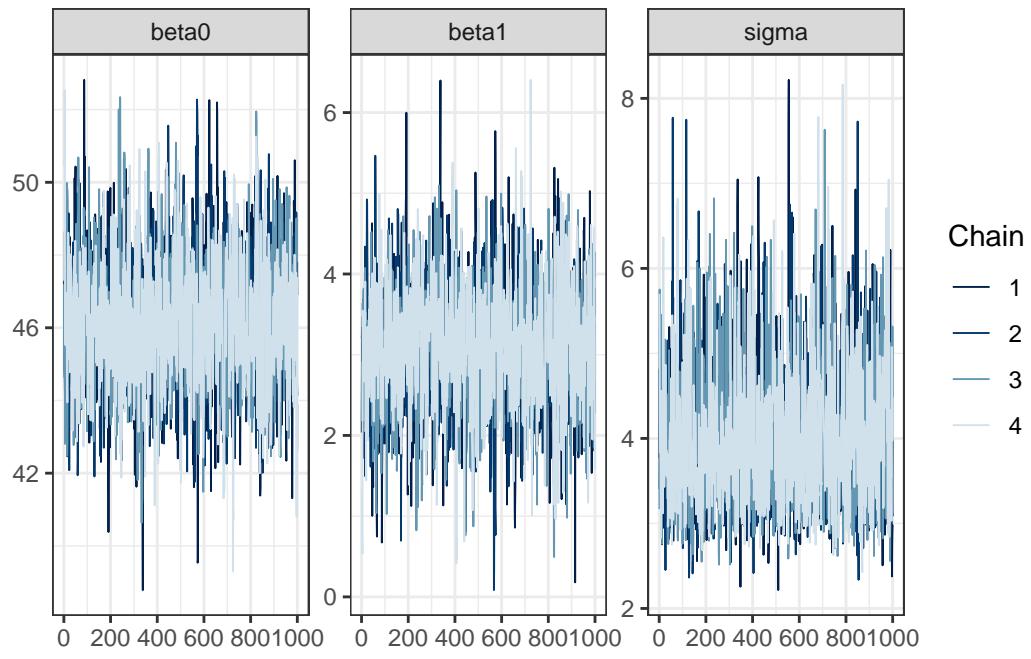
### 8.4.4 Posterior plots

There are many ways to visualize the results of a regression model. **The most important thing is to be as familiar with what the results mean as possible.** Statistics is not magic that gives you numbers that either support or do not support your theory or hypothesis. It is a way to describe your data. **If you do not want to do the work to understand what your data tell you, why bother to collect the data in the first place?**

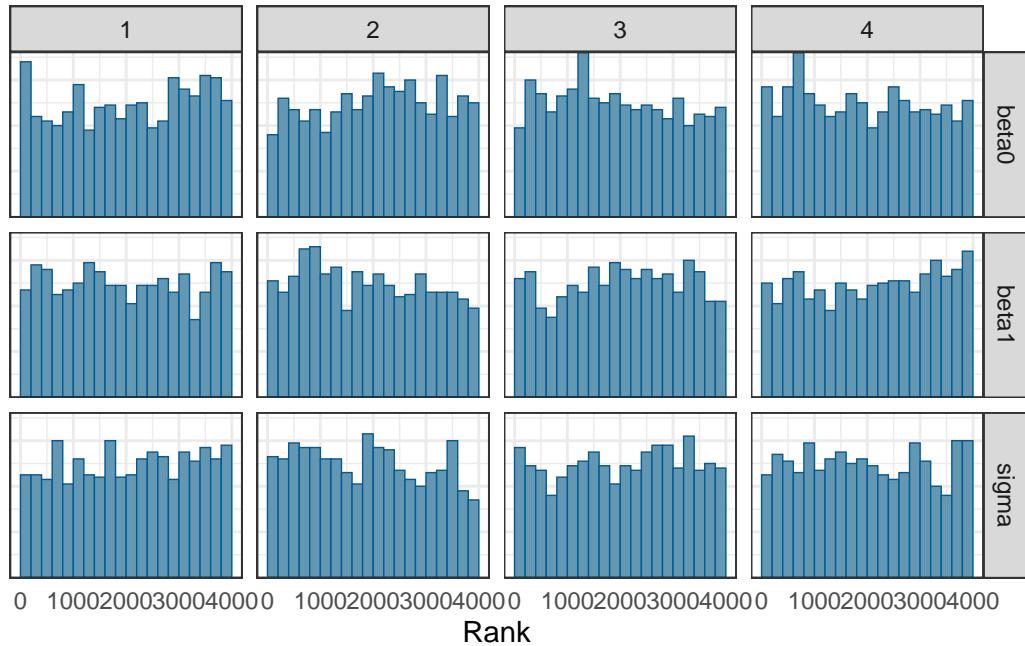
#### 8.4.4.1 Posterior density

Posterior distributions of the three parameters

```
mcmc_dens(m1_post_draws)
```



(a) Trace plot



(b) Rank histogram

Figure 8.5: Convergence diagnostics for the linear regression model.

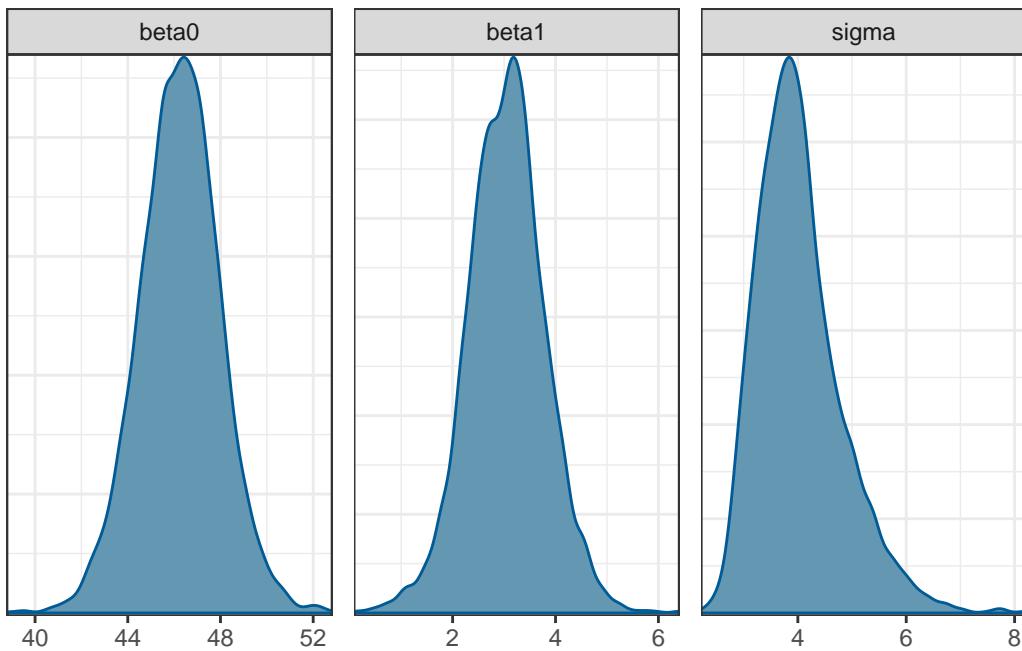


Figure 8.6: Posterior density plots for  $\beta_0$ ,  $\beta_1$ , and  $\sigma$  in the linear regression model.

You can also combine the density plot and the trace plot

```
mcmc_combo(m1_post_draws)
```

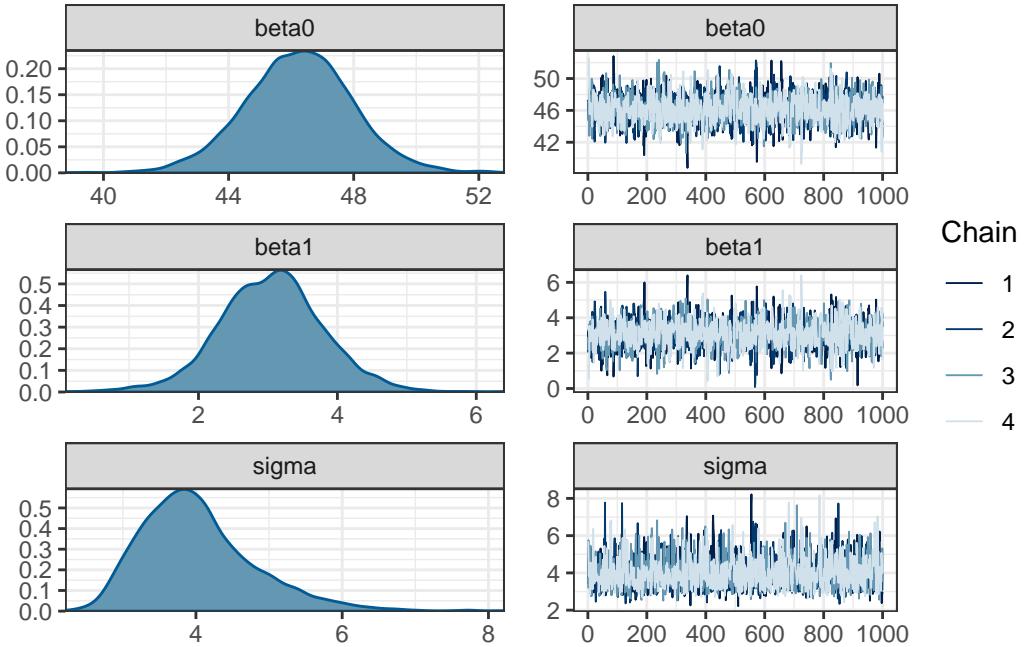


Figure 8.7: Diagnostic plots for MCMC sampling.

#### 8.4.4.2 Plot regression prediction

Figure 13.4 shows the 50% and 90% prediction intervals based on the posterior samples and the model. For example, with the 90% intervals, one expects 90% of the data should be within those intervals. If many data points lie outside the intervals, the linear model is not a good fit.

```
m1_post$draws("ytilde", format = "matrix") |>
  ppc_intervals(y = hibbs$vote, x = hibbs$growth) +
  labs(x = "Average recent growth in personal income",
       y = "Predicted incumbent party's vote share (%)") +
  ggrepel::geom_label_repel(
    aes(y = hibbs$vote, label = hibbs$year)
  )
```

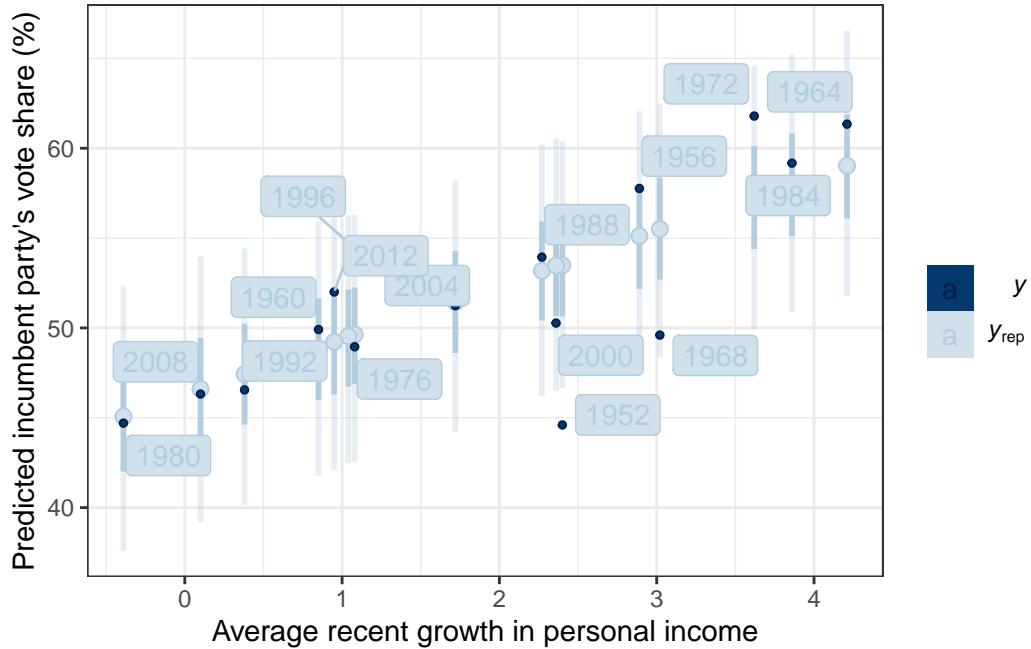


Figure 8.8: Posterior predictive intervals of vote share against personal income growth.

## 8.5 Predicting a New Data Point

In the four years before 2016, the weighted average personal income growth was 2.0 (based on Hibbs' calculation). So, based on the model, we can obtain a posterior distribution for the predicted vote share of the Democratic Party, which is the incumbent party prior to the 2016 presidential election.

```
linear_pred <- cmdstan_model("stan_code/linear_reg_pred.stan")
```

```
m1_pred <- linear_pred$sample(
  data = list(N = nrow(hibbs),
             y = hibbs$vote,
             x = hibbs$growth,
             prior_only = FALSE,
             xpred = 2),
  seed = 1227, # for reproducibility
  refresh = 1000
)
```

```
Running MCMC with 4 sequential chains...

Chain 1 Iteration: 1 / 2000 [ 0%] (Warmup)
Chain 1 Iteration: 1000 / 2000 [ 50%] (Warmup)
Chain 1 Iteration: 1001 / 2000 [ 50%] (Sampling)
Chain 1 Iteration: 2000 / 2000 [100%] (Sampling)
Chain 1 finished in 0.0 seconds.
Chain 2 Iteration: 1 / 2000 [ 0%] (Warmup)
Chain 2 Iteration: 1000 / 2000 [ 50%] (Warmup)
Chain 2 Iteration: 1001 / 2000 [ 50%] (Sampling)
Chain 2 Iteration: 2000 / 2000 [100%] (Sampling)
Chain 2 finished in 0.0 seconds.
Chain 3 Iteration: 1 / 2000 [ 0%] (Warmup)
Chain 3 Iteration: 1000 / 2000 [ 50%] (Warmup)
Chain 3 Iteration: 1001 / 2000 [ 50%] (Sampling)
Chain 3 Iteration: 2000 / 2000 [100%] (Sampling)
Chain 3 finished in 0.0 seconds.
Chain 4 Iteration: 1 / 2000 [ 0%] (Warmup)
Chain 4 Iteration: 1000 / 2000 [ 50%] (Warmup)
Chain 4 Iteration: 1001 / 2000 [ 50%] (Sampling)
Chain 4 Iteration: 2000 / 2000 [100%] (Sampling)
Chain 4 finished in 0.0 seconds.

All 4 chains finished successfully.
Mean chain execution time: 0.0 seconds.
Total execution time: 0.5 seconds.
```

```
m1_pred$draws("ypred") |>
  mcmc_dens()
```

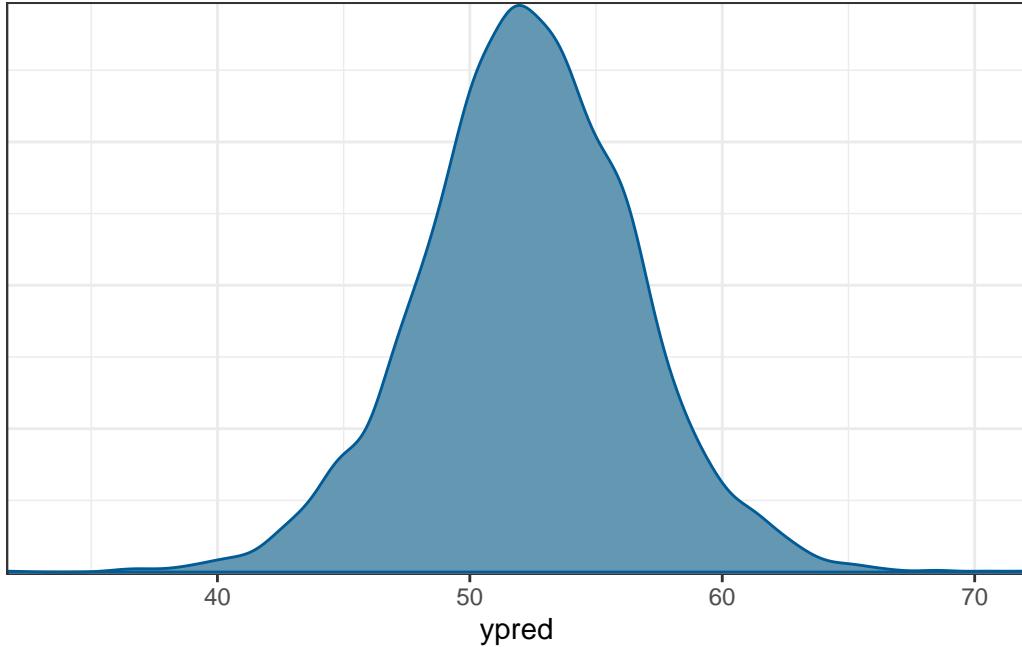


Figure 8.9: Posterior predictive distribution of predicted vote share when growth = 2.

The mean of the incumbent's predicted vote share with average income growth = 2 is 52.3006575. The actual outcome of the election was that the Democratic Party received about 51.1% of the votes among the two parties, so it was below the posterior predictive mean but within the range of possible outcomes. Of course, we know that the actual election outcome is based on the Electoral College, not the majority vote.

## 8.6 Robust Regression

The linear model assuming normally distributed errors is not robust to outliers or influential observations. It can be robustified by assuming a more heavy-tailed distribution, such as the  $t$  distribution:

$$\text{vote}_i \sim t_\nu(\mu_i, \sigma),$$

where  $\nu$  is an additional degrees of freedom parameter controlling for the “heaviness” of the tails. When  $\nu$  is close to 0 (e.g., 3 or 4), the outliers/influential cases are weighed less; when  $\nu$  is close to infinity, the  $t$  distribution becomes a normal distribution.

With Bayesian methods, we can let the model learn from the data about the  $\nu$  parameter. In this case, we assume a *hyperprior* for  $\nu$ ;  $\text{Gamma}(2, 0.1)$  is something recommended in the literature.

```

data {
  int<lower=0> N; // number of observations
  vector[N] y; // outcome;
  vector[N] x; // predictor;
  int<lower=0,upper=1> prior_only; // whether to sample prior only
}
parameters {
  real beta0; // regression intercept
  real beta1; // regression coefficient
  real<lower=0> sigma; // SD of prediction error
  real<lower=1> nu; // df parameter
}
model {
  // model
  if (!prior_only) {
    y ~ student_t(nu, beta0 + beta1 * x, sigma);
  }
  // prior
  beta0 ~ normal(45, 10);
  beta1 ~ normal(0, 10);
  sigma ~ student_t(4, 0, 5);
  nu ~ gamma(2, 0.1); // prior for df parameter
}
generated quantities {
  vector[N] ytilde; // place holder
  for (i in 1:N)
    ytilde[i] = normal_rng(beta0 + beta1 * x[i], sigma);
}
```

```
robust_reg <- cmdstan_model("stan_code/robust_reg.stan")
```

```

m1_robust <- robust_reg$sample(
  data = list(N = nrow(hibbs),
             y = hibbs$vote,
             x = hibbs$growth,
             prior_only = FALSE),
  seed = 1227, # for reproducibility
  refresh = 0
```

```
)
```

```
Running MCMC with 4 sequential chains...
```

```
Chain 1 finished in 0.0 seconds.  
Chain 2 finished in 0.0 seconds.  
Chain 3 finished in 0.0 seconds.  
Chain 4 finished in 0.0 seconds.
```

```
All 4 chains finished successfully.  
Mean chain execution time: 0.0 seconds.  
Total execution time: 0.5 seconds.
```

```
m1r_summ <- m1_robust$summary(c("beta0", "beta1", "sigma", "nu"))  
# Use `knitr::kable()` for tabulation  
knitr::kable(m1r_summ, digits = 2)
```

Table 8.2: Posterior summary of Student t regression model.

variable	mean	median	sd	mad	q5	q95	rhat	ess_bulk	ess_tail
beta0	46.17	46.18	1.55	1.45	43.55	48.64	1	1628.35	1673.90
beta1	3.20	3.22	0.69	0.66	2.01	4.30	1	1598.75	1914.04
sigma	3.56	3.49	0.92	0.83	2.20	5.10	1	1044.24	586.21
nu	18.26	14.64	13.86	11.60	2.97	45.62	1	1234.64	638.30

```
m1_robust$draws("ytilde", format = "matrix") |>  
  ppc_intervals(y = hibbs$vote, x = hibbs$growth) +  
  labs(x = "Average recent growth in personal income",  
        y = "Predicted incumbent party's vote share (%)") +  
  ggrepel::geom_label_repel(  
    aes(y = hibbs$vote, label = hibbs$year)  
)
```

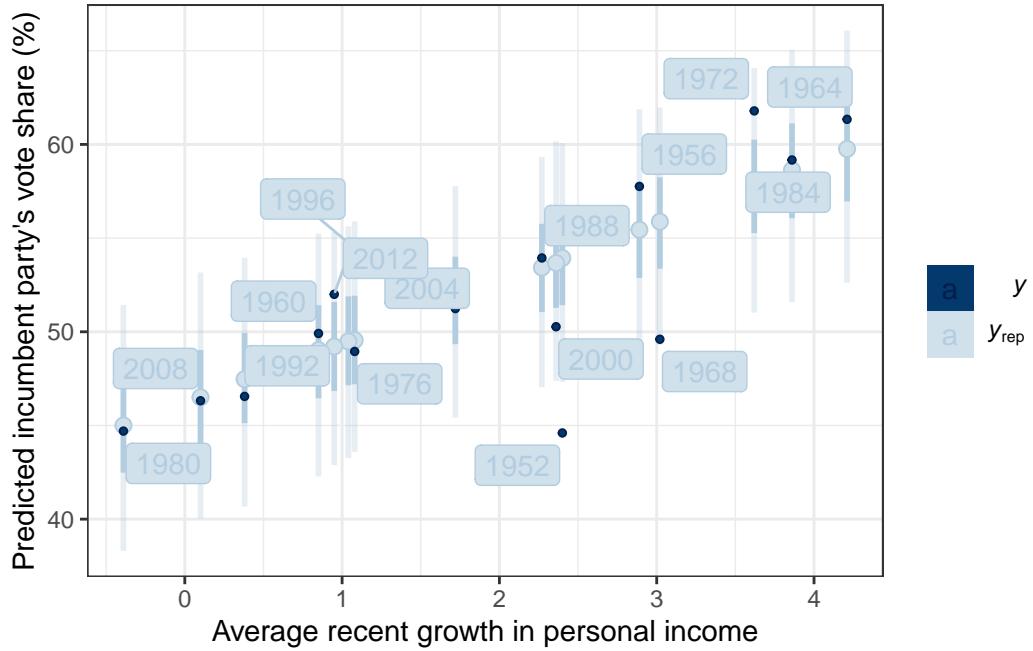


Figure 8.10: Posterior predictive intervals of vote share against personal income growth based on Student t regression.

# 9 Multiple Predictors

Here, we'll use an example from McElreath (2020), which contains some marriage and demographic statistics for the individual states in the United States. See <https://rdrr.io/github/rmcelreath/rethinking/man/WaffleDivorce.html> for more details.

```
if (!file.exists("data/WaffleDivorce.csv")) {  
  download.file(  
    "https://raw.githubusercontent.com/rmcelreath/rethinking/master/data/WaffleDivorce.csv",  
    "data/WaffleDivorce.csv"  
  )  
}  
  
waffle_divorce <- read_delim( # read delimited files  
  "data/WaffleDivorce.csv",  
  delim = ";"  
)  
  
# Rescale Marriage and Divorce by dividing by 10  
waffle_divorce$Marriage <- waffle_divorce$Marriage / 10  
waffle_divorce$Divorce <- waffle_divorce$Divorce / 10  
waffle_divorce$MedianAgeMarriage <- waffle_divorce$MedianAgeMarriage / 10  
  
# Recode `South` to a factor variable  
waffle_divorce$South <- factor(waffle_divorce$South,  
  levels = c(0, 1),  
  labels = c("non-south", "south"))  
}  
  
# See data description at https://rdrr.io/github/rmcelreath/rethinking/man/WaffleDivorce.html
```

## 9.1 Stratified Analysis

Let's consider whether the association between `MedianAgeMarriage` and `Divorce` differs between Southern and non-Southern states. Because (and **only because**) the groups are **independent**, we can fit a linear regression for each subset of states.

```

ggplot(waffle_divorce,
       aes(x = MedianAgeMarriage, y = Divorce, col = South)) +
  geom_point() +
  geom_smooth() +
  labs(x = "Median age marriage (10 years)",
       y = "Divorce rate (per 10 adults)") +
  ggrepel::geom_text_repel(aes(label = Loc), max.overlaps = 15)

```

`geom\_smooth()` using method = 'loess' and formula = 'y ~ x'

Warning: ggrepel: 4 unlabeled data points (too many overlaps). Consider increasing max.overlaps

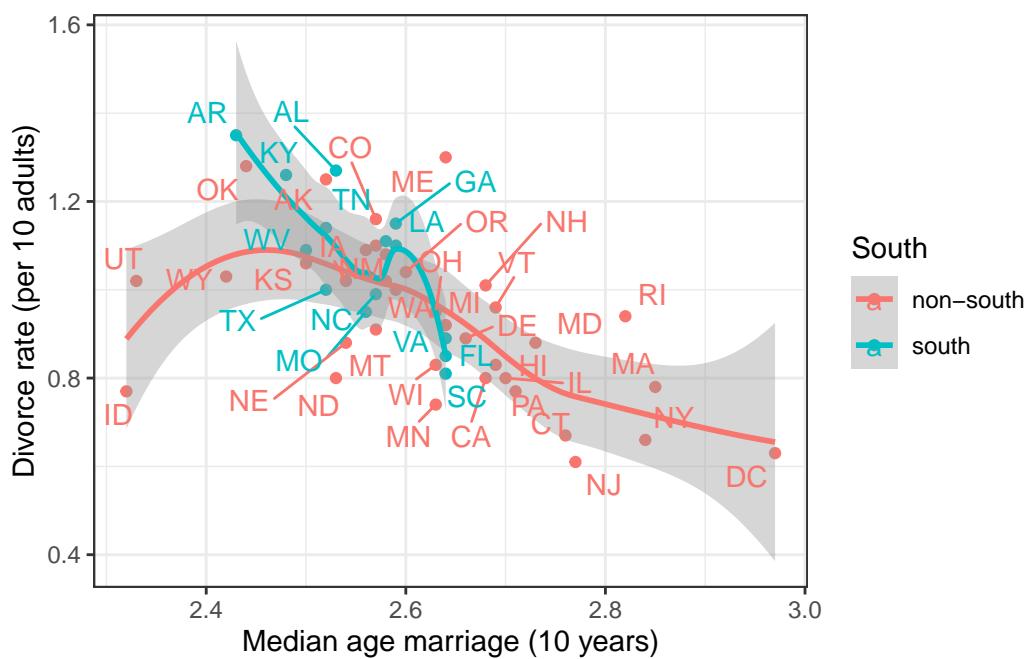


Figure 9.1: State-level association between median age of marriage and divorce rate.

## 9.2 Introducing the `brms` package

While Stan is very flexible, because the linear model and some related models are so widely used, some authors have created packages that would further simplify the fitting of such models. One of those packages is `brms`, which I believe stands for “Bayesian regression models with

Stan.” It allows one to do MCMC sampling using Stan, but with syntax similar to that in R functions `lm()`, `glm()`, and `lme4::lmer()`. `brms` probably supports more models than any R packages that statisticians routinely used; there are models, like factor analysis, that are not directly supported. If you came across some fancy regression models, chances are you can do something similar in `brms`. You can find some resources for learning `brms` on this page: <https://paul-buerkner.github.io/brms/>.

## Model formula

`brms` comes with some default prior options, but I recommend you always check what priors are used and think about whether they make sense for your data. You can use `get_priors()` to show the default priors used in `brms`.

To do MCMC sampling, we use the `brm()` function. The first argument is a formula in R. The variable before `~` is the outcome, whereas the ones after `~` are the predictors. For example,

```
vote ~ 1 + growth
```

means the model

$$E(\text{vote}_i) = \beta_0(1) + \beta_1(\text{growth}_i).$$

Usually, I write `vote ~ 1 + growth` as `vote ~ growth`, as the `1 +` part is automatically added.

## Setting Priors

`brms` comes with some default prior options, but over the years, the package maintainers have changed those priors, so the default today may be different from the one next year. Therefore, you should always check what priors are used and think about whether they make sense for your data. You can use `get_priors()` to show the default priors used in `brms`. For example,

```
get_prior(Divorce ~ MedianAgeMarriage,
           data = waffle_divorce)
```

prior	class	coef	group	resp	dpar	npar	lb	ub
(flat)	b							
(flat)	b	MedianAgeMarriage						
student_t(3, 1, 2.5)	Intercept							
student_t(3, 0, 2.5)	sigma						0	

```

source
default
(vectorized)
default
default

m_nonsouth <-
  brm(Divorce ~ MedianAgeMarriage,
    # Filter `waffle_divorce` to only include non-southern states
    data = filter(waffle_divorce, South == "non-south"),
    # Use N(0, 2) and N(0, 10) as priors for beta1 and beta0,
    # and use t_4(0, 3) as a prior for sigma
    prior = prior(normal(0, 2), class = "b") +
      prior(normal(0, 10), class = "Intercept") +
      prior(student_t(4, 0, 3), class = "sigma"),
    seed = 941,
    iter = 4000,
    file = "m_nonsouth"
  )

```

```

m_south <-
  brm(Divorce ~ MedianAgeMarriage,
    data = filter(waffle_divorce, South == "south"),
    prior = prior(normal(0, 2), class = "b") +
      prior(normal(0, 10), class = "Intercept") +
      prior(student_t(4, 0, 3), class = "sigma"),
    seed = 2157, # use a different seed
    iter = 4000,
    file = "m_south"
  )

```

We can make a table like Table 9.1 for `brms` results using the `modelsummary::msummary()` function.

```

msummary(list(South = m_south, `Non-South` = m_nonsouth),
  estimate = "{estimate} [{conf.low}, {conf.high}]",
  statistic = NULL, fmt = 2,
  gofomit = "^(?!Num)" # only include number of observations
)

```

**Warning:**

`modelsummary` uses the `performance` package to extract goodness-of-fit

Table 9.1: Intercepts and slopes for south and non-south states.

	South	Non-South
b_Intercept	6.09 [3.60, 8.45]	2.74 [1.75, 3.74]
b_MedianAgeMarriage	-1.96 [-2.89, -0.98]	-0.69 [-1.07, -0.31]
sigma	0.11 [0.07, 0.17]	0.15 [0.12, 0.20]
Num.Obs.	14	36

statistics from models of this class. You can specify the statistics you wish to compute by supplying a `metrics` argument to `modelsummary`, which will then push it forward to `performance`. Acceptable values are: "all", "common", "none", or a character vector of metrics names. For example: `modelsummary(mod, metrics = c("RMSE", "R2"))` Note that some metrics are computationally expensive. See `?performance::performance` for details.

This warning appears once per session.

We can now ask two questions:

- Is the intercept different across southern and non-southern states?
- Is the slope different across southern and non-southern states?

The correct way to answer the above questions is to obtain the posterior distribution of the **difference** in the coefficients. Repeat: obtain the posterior distribution of the **difference**. The incorrect way is to compare whether the CIs overlap.

Here are the posteriors of the differences ( $\beta_0^{\text{south}} - \beta_0^{\text{nonsouth}}$  and  $\beta_1^{\text{south}} - \beta_1^{\text{nonsouth}}$ ):

```
# Extract draws
draws_south <- as_draws_matrix(m_south,
  variable = c("b_Intercept", "b_MedianAgeMarriage")
)
draws_nonsouth <- as_draws_matrix(m_nonsouth,
  variable = c("b_Intercept", "b_MedianAgeMarriage")
)
# Difference in coefficients
draws_diff <- draws_south - draws_nonsouth
# Rename the columns
colnames(draws_diff) <- paste0("d", colnames(draws_diff))
# Summarize
summarize_draws(draws_diff) |>
  knitr::kable(digits = 2)
```

Table 9.2: Difference in intercept and slope for south and non-south states.

variable	mean	median	sd	mad	q5	q95	rhat	ess_bulk	ess_tail
db_Intercept	3.32	3.34	1.34	1.29	1.14	5.45	1	6115.12	5017.01
db_MedianAgeMarriage26	-1.27	0.52	0.50	-2.09	-0.41	1	6112.20	5034.56	

As you can see, the southern states have a higher intercept and a lower slope.

```
plot(
  conditional_effects(m_nonsouth),
  points = TRUE, plot = FALSE
)[[1]] + ggtitle("Non-South") + lims(x = c(2.3, 3), y = c(0.6, 1.4))
plot(
  conditional_effects(m_south),
  points = TRUE, plot = FALSE
)[[1]] + ggtitle("South") + lims(x = c(2.3, 3), y = c(0.6, 1.4))
```

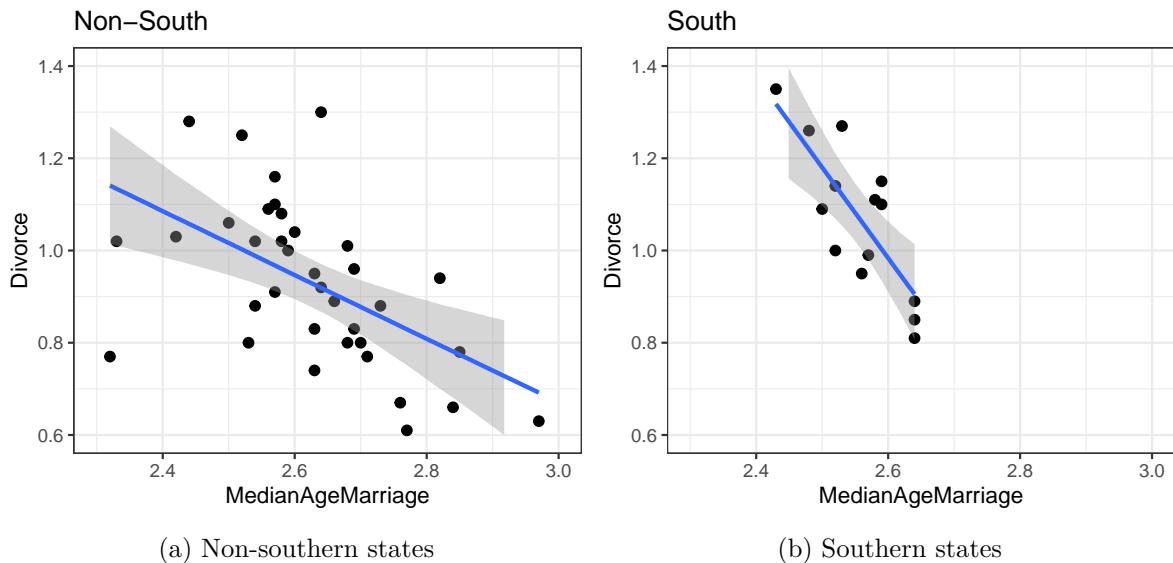


Figure 9.2: Model-implied regression lines

### 9.3 Additive Model

An additive model assumes that the difference between predicted  $Y$  for two levels of  $X_1$  is the same regardless of the level of  $X_2$ . In our example, we assume that the predicted difference

in divorce rate associated with the median age of marriage does not depend on whether the state is southern or not. Equivalently, we assume that the predicted difference in Southern and non-Southern states does not depend on the median age of marriage.

$$\begin{aligned} D_i &\sim N(\mu_i, \sigma) \\ \mu_i &= \beta_0 + \beta_1 S_i + \beta_2 A_i \\ \beta_0 &\sim N(0, 10) \\ \beta_1 &\sim N(0, 10) \\ \beta_2 &\sim N(0, 1) \\ \sigma &\sim t_4^+(0, 3) \end{aligned}$$

- $\beta_1$ : Expected difference in divorce rate between southern and non-southern states with the same median age of marriage.
- $\beta_2$ : Expected difference in divorce rate for one unit difference in median age of marriage, when both states are southern (or non-southern).

In the model, the variable  $S$ , southern state, is a dummy variable with 0 = non-southern and 1 = southern. Therefore,

### Dummy Coding

- For non-southern states,  $\mu = (\beta_0) + (\beta_2)A$ ;
- For southern states,  $\mu = (\beta_0 + \beta_1) + \beta_2 A$

```
m_additive <- brm(
  Divorce ~ South + MedianAgeMarriage,
  data = waffle_divorce,
  prior = prior(normal(0, 2), class = "b") +
    prior(normal(0, 10), class = "b", coef = "Southsouth") +
    prior(normal(0, 10), class = "Intercept") +
    prior(student_t(4, 0, 3), class = "sigma"),
  seed = 941,
  iter = 4000,
  file = "m_additive"
)
```

```
m_additive
```

```
Family: gaussian
Links: mu = identity; sigma = identity
```

```

Formula: Divorce ~ South + MedianAgeMarriage
Data: waffle_divorce (Number of observations: 50)
Draws: 4 chains, each with iter = 4000; warmup = 2000; thin = 1;
      total post-warmup draws = 8000

```

Regression Coefficients:

	Estimate	Est.Error	1-95% CI	u-95% CI	Rhat	Bulk_ESS	Tail_ESS
Intercept	3.00	0.46	2.10	3.89	1.00	7435	5360
Southsouth	0.08	0.05	-0.01	0.18	1.00	7847	5519
MedianAgeMarriage	-0.79	0.18	-1.13	-0.45	1.00	7406	5423

Further Distributional Parameters:

	Estimate	Est.Error	1-95% CI	u-95% CI	Rhat	Bulk_ESS	Tail_ESS
sigma	0.15	0.02	0.12	0.18	1.00	7217	5426

Draws were sampled using `sample(hmc)`. For each parameter, Bulk\_ESS and Tail\_ESS are effective sample size measures, and Rhat is the potential scale reduction factor on split chains (at convergence, Rhat = 1).

## 9.4 Interaction Model: Different Slopes Across Two Groups

An alternative is to include an interaction term

$$\begin{aligned}
D_i &\sim N(\mu_i, \sigma) \\
\mu_i &= \beta_0 + \beta_1 S_i + \beta_2 A_i + \beta_3 S_i \times A_i \\
\beta_0 &\sim N(0, 10) \\
\beta_1 &\sim N(0, 10) \\
\beta_2 &\sim N(0, 1) \\
\beta_3 &\sim N(0, 2) \\
\sigma &\sim t_4^+(0, 3)
\end{aligned}$$

- $\beta_1$ : Difference in intercept between southern and non-southern states.
- $\beta_3$ : Difference in the coefficient for  $A \rightarrow D$  between southern and non-southern states

### 💡 Dummy Coding in Interaction Model

- For non-southern states,  $\mu = (\beta_0) + (\beta_2)A$ ;
- For southern states,  $\mu = (\beta_0 + \beta_1) + (\beta_2 + \beta_3)A$

```

m_inter <- brm(
  Divorce ~ South * MedianAgeMarriage,
  data = waffle_divorce,
  prior = prior(normal(0, 2), class = "b") +
    prior(normal(0, 10), class = "b", coef = "Southsouth") +
    prior(normal(0, 10), class = "Intercept") +
    prior(student_t(4, 0, 3), class = "sigma"),
  seed = 941,
  iter = 4000,
  file = "m_inter"
)

```

The formula `Divorce ~ South * MedianAgeMarriage` is the same as

```
Divorce ~ South + MedianAgeMarriage + South:MedianAgeMarriage
```

where : is the symbol in R for a product term.

```

# Print summary of the model
m_inter

Family: gaussian
Links: mu = identity; sigma = identity
Formula: Divorce ~ South * MedianAgeMarriage
Data: waffle_divorce (Number of observations: 50)
Draws: 4 chains, each with iter = 4000; warmup = 2000; thin = 1;
      total post-warmup draws = 8000

Regression Coefficients:
Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS
Intercept          2.78     0.46    1.87    3.68 1.00    4716
Southsouth         3.20     1.60    0.19    6.30 1.00    2863
MedianAgeMarriage -0.70     0.18   -1.05   -0.35 1.00    4714
Southsouth:MedianAgeMarriage -1.22     0.62   -2.42   -0.03 1.00    2876
Tail_ESS
Intercept          4042
Southsouth         3608
MedianAgeMarriage 4085
Southsouth:MedianAgeMarriage 3584

```

Further Distributional Parameters:

```
Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
sigma     0.14      0.02      0.12      0.18 1.00      4244      4998
```

Draws were sampled using `sample(hmc)`. For each parameter, `Bulk_ESS` and `Tail_ESS` are effective sample size measures, and `Rhat` is the potential scale reduction factor on split chains (at convergence, `Rhat = 1`).

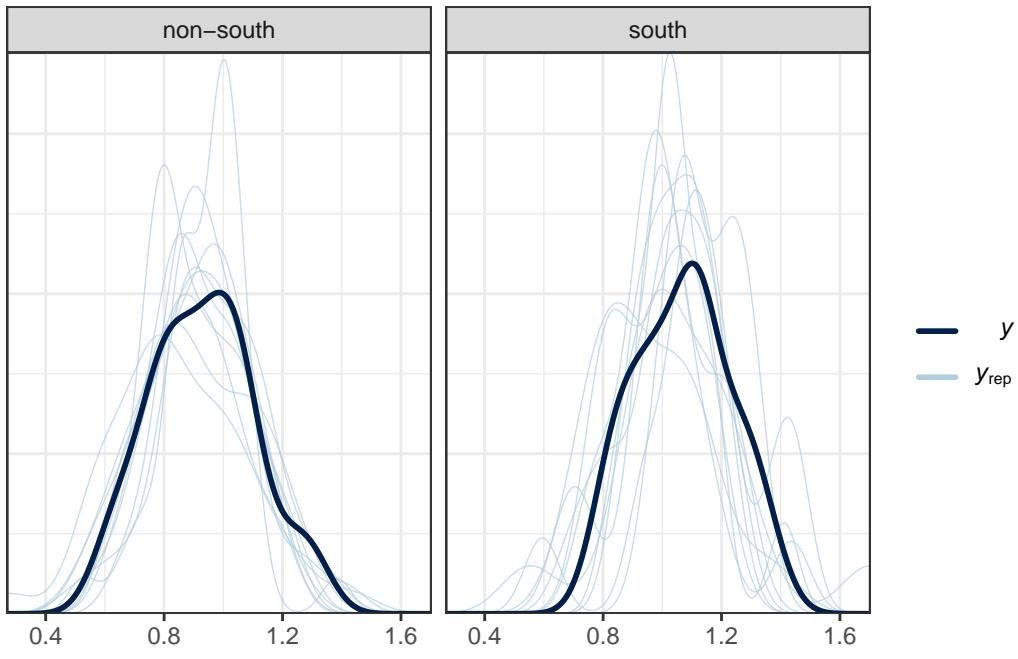
#### 9.4.1 Posterior predictive checks

```
# Check density (normality)
pp_check(m_inter, type = "dens_overlay_grouped", group = "South")
```

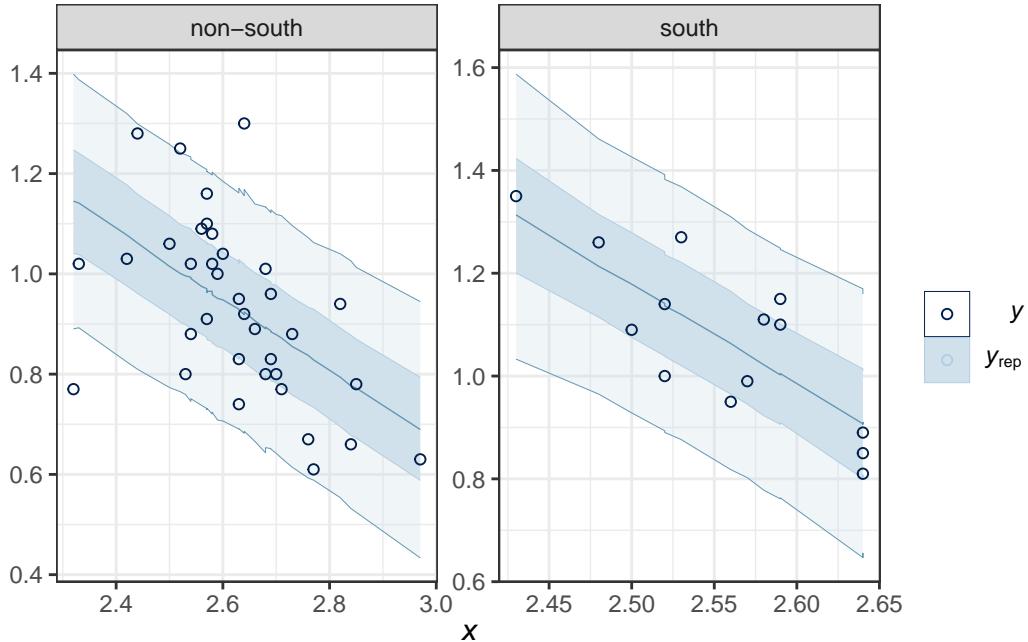
Using 10 posterior draws for ppc type 'dens\_overlay\_grouped' by default.

```
# Check prediction (a few outliers)
pp_check(m_inter,
  type = "ribbon_grouped", x = "MedianAgeMarriage",
  group = "South",
  y_draw = "points"
)
```

Using all posterior draws for ppc type 'ribbon\_grouped' by default.



(a) Density overlayed plots



(b) Prediction intervals

Figure 9.3: Posterior predictive checks for the interaction model.

```

# Check errors (no clear pattern)
pp_check(m_inter,
  type = "error_scatter_avg_vs_x", x = "MedianAgeMarriage"
)

```

Using all posterior draws for ppc type 'error\_scatter\_avg\_vs\_x' by default.

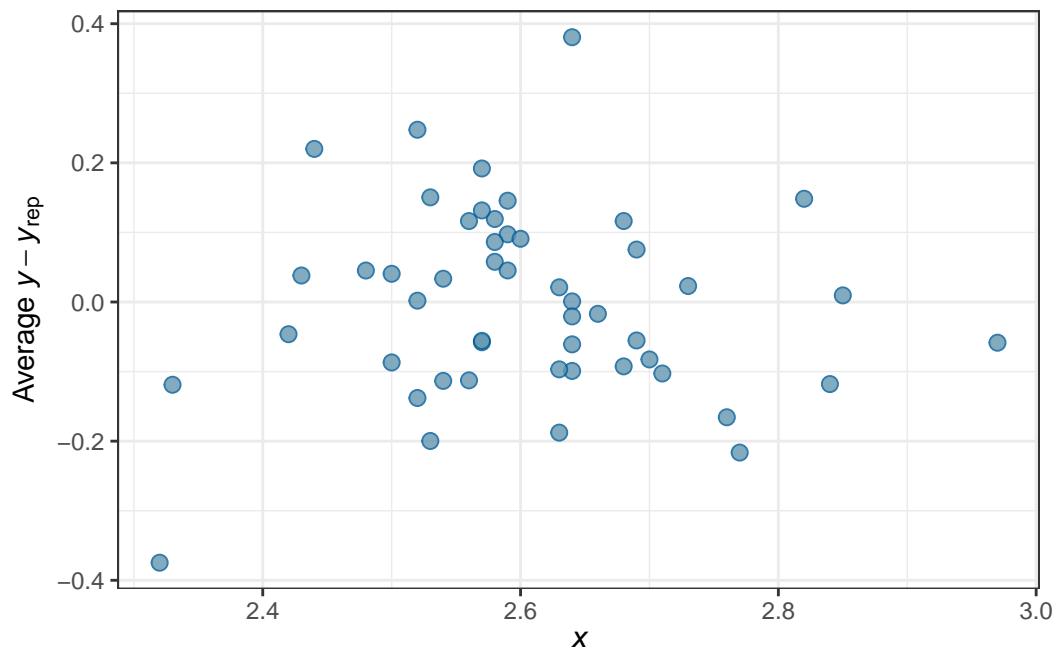


Figure 9.4: Average prediction error against the predictor. No clear pattern should be observed for a correctly specified model.

#### 9.4.2 Conditional effects/simple slopes

Slope of MedianAgeMarriage when South = 0:  $\beta_1$

Slope of MedianAgeMarriage when South = 1:  $\beta_1 + \beta_3$

```

as_draws(m_inter) |>
  mutate_variables(
    b_nonsouth = b_MedianAgeMarriage,
    b_south = b_MedianAgeMarriage + `b_Southsouth:MedianAgeMarriage` )
  ) |>

```

```

posterior::subset_draws(
  variable = c("b_nonsouth", "b_south")
) |>
summarize_draws() |>
knitr::kable(digits = 2)

```

variable	mean	median	sd	mad	q5	q95	rhat	ess_bulk	ess_tail
b_nonsouth	-0.70	-0.71	0.18	0.18	-1.00	-0.41	1	4713.61	4085.16
b_south	-1.92	-1.91	0.60	0.60	-2.91	-0.93	1	3168.15	3659.43

```

plot(
  conditional_effects(m_inter,
    effects = "MedianAgeMarriage",
    conditions = data.frame(South = c("south", "non-south"),
      cond__ = c("South", "Non-South"))
  ),
  points = TRUE
)

```

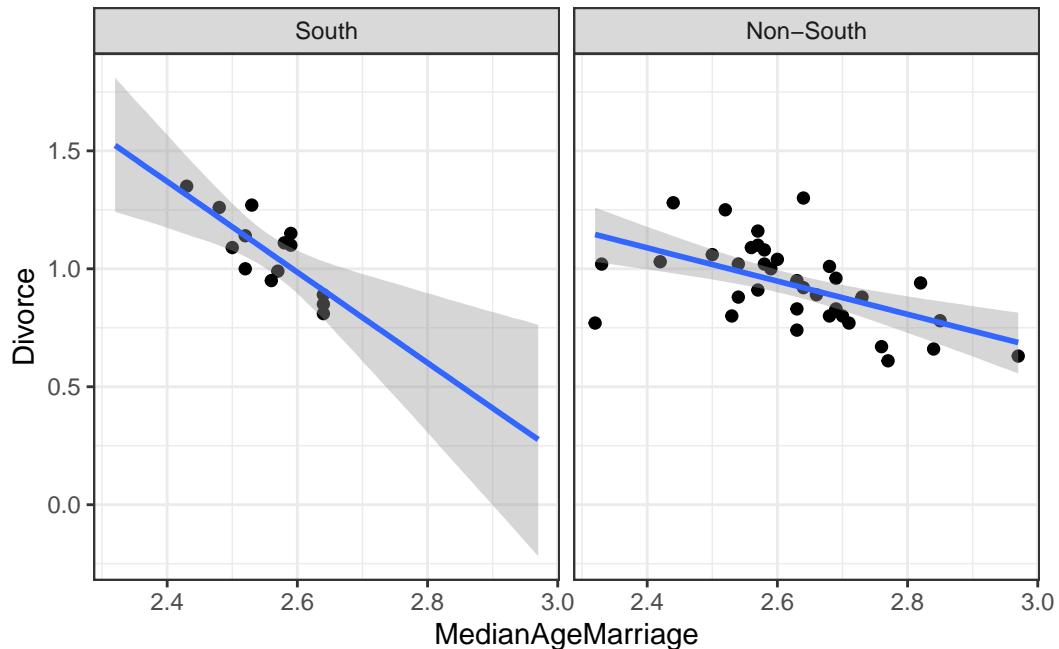


Figure 9.5: Model-implied simple slopes based on the interaction model.

## 9.5 Interaction of Continuous Predictors

```
plotly::plot_ly(waffle_divorce,
                 x = ~Marriage,
                 y = ~MedianAgeMarriage,
                 z = ~Divorce)
```

No trace type specified:

Based on info supplied, a 'scatter3d' trace seems appropriate.

Read more about this trace type -> <https://plotly.com/r/reference/#scatter3d>

No scatter3d mode specified:

Setting the mode to markers

Read more about this attribute -> <https://plotly.com/r/reference/#scatter-mode>

Figure 9.6: 3-D plot visualizing one outcome and two predictors.

$$D_i \sim N(\mu_i, \sigma)$$

$$\mu_i = \beta_0 + \beta_1 M_i + \beta_2 A_i + \beta_3 M_i \times A_i$$

```
# Use default priors (just for convenience here)
m_inter2 <- brm(Divorce ~ Marriage * MedianAgeMarriage,
  data = waffle_divorce,
  seed = 941,
  iter = 4000,
  file = "m_inter2"
)
```

## 9.6 Centering

In the previous model,  $\beta_1$  is the slope of  $M \rightarrow D$  when  $A$  is 0 (i.e., median marriage age = 0), and  $\beta_2$  is the slope of  $A \rightarrow D$  when  $M$  is 0 (i.e., marriage rate is 0). These two are not very meaningful. Therefore, it is common to make the zero values more meaningful by doing *centering*.

Here, I use  $M - 2$  as the predictor, so the zero point means a marriage rate of 2 per 10 adults; I use  $A - 2.5$  as the other predictor, so the zero point means a median marriage rate of 25 years old.

$$\mu_i = \beta_0 + \beta_1(M_i - 2) + \beta_2(A_i - 2.5) + \beta_3(M_i - 2) \times (A_i - 2.5)$$

```
# Use default priors (just for convenience here)
m_inter2c <- brm(Divorce ~ I(Marriage - 2) * I(MedianAgeMarriage - 2.5),
  data = waffle_divorce,
  seed = 941,
  iter = 4000,
  file = "m_inter2c"
)
```

```
msummary(list(`No centering` = m_inter2, `centered` = m_inter2c),
  estimate = "[estimate] [{conf.low}, {conf.high}]",
  statistic = NULL, fmt = 2)
```

As shown in the table above, while the two models are equivalent in fit and give the same posterior distribution for  $\beta_3$ , they differ in  $\beta_0$ ,  $\beta_1$ , and  $\beta_2$ .

	No centering	centered
b_Intercept	7.38 [3.02, 11.58]	1.09 [1.02, 1.16]
b_Marriage	-1.93 [-4.00, 0.16]	
b_MedianAgeMarriage	-2.45 [-4.07, -0.78]	
b_Marriage × MedianAgeMarriage	0.74 [-0.07, 1.56]	
sigma	0.15 [0.12, 0.18]	0.15 [0.12, 0.18]
b_IMarriageM2		-0.08 [-0.24, 0.09]
b_IMedianAgeMarriageM2.5		-0.94 [-1.44, -0.44]
b_IMarriageM2 × IMedianAgeMarriageM2.5		0.75 [-0.08, 1.59]
Num.Obs.	50	50
R2	0.414	0.413
R2 Adj.	0.280	0.273
ELPD	21.5	21.2
ELPD s.e.	6.1	6.2
LOOIC	-42.9	-42.5
LOOIC s.e.	12.2	12.5
WAIC	-43.5	-43.1
RMSE	0.14	0.14

```

plot(
  conditional_effects(m_inter2c,
    effects = "Marriage:MedianAgeMarriage",
    int_conditions = list(MedianAgeMarriage = c(2.3, 2.5, 2.7)),
  ),
  points = TRUE
)

```

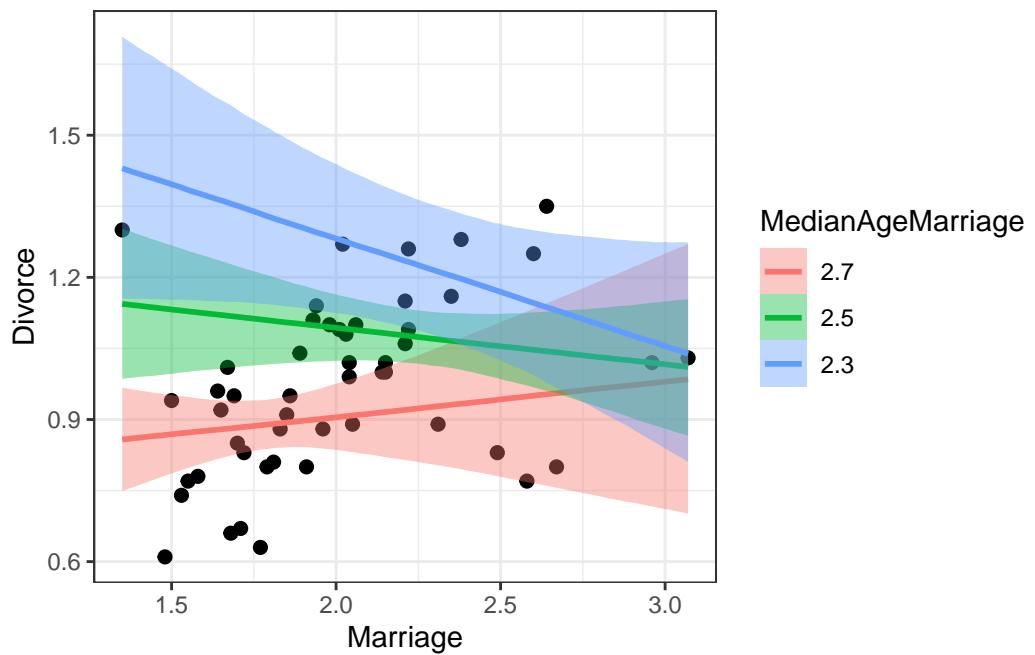


Figure 9.7: Model-implied simple slopes of marriage for different levels of median age of marriage.

# 10 Model Diagnostics

All statistical models are sets of assumptions about the data-generating process, and estimation will be meaningless or misleading if these assumptions do not hold for the data. As we have discussed, choosing a good model is generally more important than choosing a good prior. In this note, we will learn some tools to check the validity of linear models. Most of them are similar to what you have learned in frequentist regression.

## 10.1 Assumptions of Linear Models

The assumptions of the linear model is encoded in the model. The model is

$$Y_i \sim N(\mu_i, \sigma) \\ \mu_i = \beta_0 + \beta_1 X_{1i} + \beta_2 X_{2i} + \dots$$

From the model, we have the following assumptions, in the order of the most important one to the least important one:

1. *Correct specification of the model.* This means that all relevant predictors for  $Y$  have been included in the model. This is probably an assumption that is never satisfied in real data analysis, as one can never include all relevant factors that can have an impact on  $Y$ , be it small or large. However, it is important to be thoughtful to include major predictors that have been shown to relate to  $Y$ . Leaving out key predictors can bias the coefficients  $\beta$ .
2. *Linearity.* This is about the conditional mean,  $\mu = E(Y|X_1, X_2, \dots)$ , being a linear function. If you have a function like  $\mu = \exp[\beta_1 X_1 \sin(\beta_2 X_2)]$ , the conditional mean is not a linear function. Note that linearity does not require  $\mu$  to be a linear function of predictors; quadratic and exponential relationships, interaction, and polynomials can all be handled by linear models. (And technically, linearity requires  $\mu$  to be a linear function of the coefficients.)
3. *Independent observations.* This assumption is not directly encoded in the model equation above, mainly because I omit that part when writing out the model. This assumption requires that the value of one observation is independent of the value of another observation after taking into account the conditional mean,  $\mu$ . This will be discussed more in multilevel models.

4. *Equal variance of errors.* This means that  $\sigma^2$  has to be constant for each observation. In general, Violating this assumption is generally a minor issue, although it can affect the posterior standard deviation (analogous to standard errors).
5. *Normality.* This requires that the conditional distribution of  $Y$  is normal. Violating of the normality assumption generally does not affect the estimation of the coefficients, and will be a minor issue when the sample size is large enough ( $> 30$ ) and when the degree of nonnormality is small to moderate.

## 10.2 Diagnostic Tools

Now let's review some tools for regression diagnostics for Bayesian regression. There are hundreds of plots available that I will not cover here, and you can treat what is discussed in this note as a minimal requirement for regression diagnostics. The first one is about a correct specification of the model, which can be partly assessed with posterior predictive check.

### 10.2.1 Posterior Predictive Check

We've already seen a few examples of posterior predictive checks in the previous chapters. Continuing with the example of predicting the divorce rate, here are a few more plots.

Below is the posterior predictive graphical check for the interaction model we fit for the state-level divorce rate data:

Based on the graphical check, we do not see any major systematic discrepancies between the distribution of our data and what can be predicted from our model. The ribbon plot shows that some points are outside the 90% predictive intervals but not too far off.

We should do the posterior predictive check with test statistics too. The following functions show the mean, maximum value, and minimum value of the outcome.

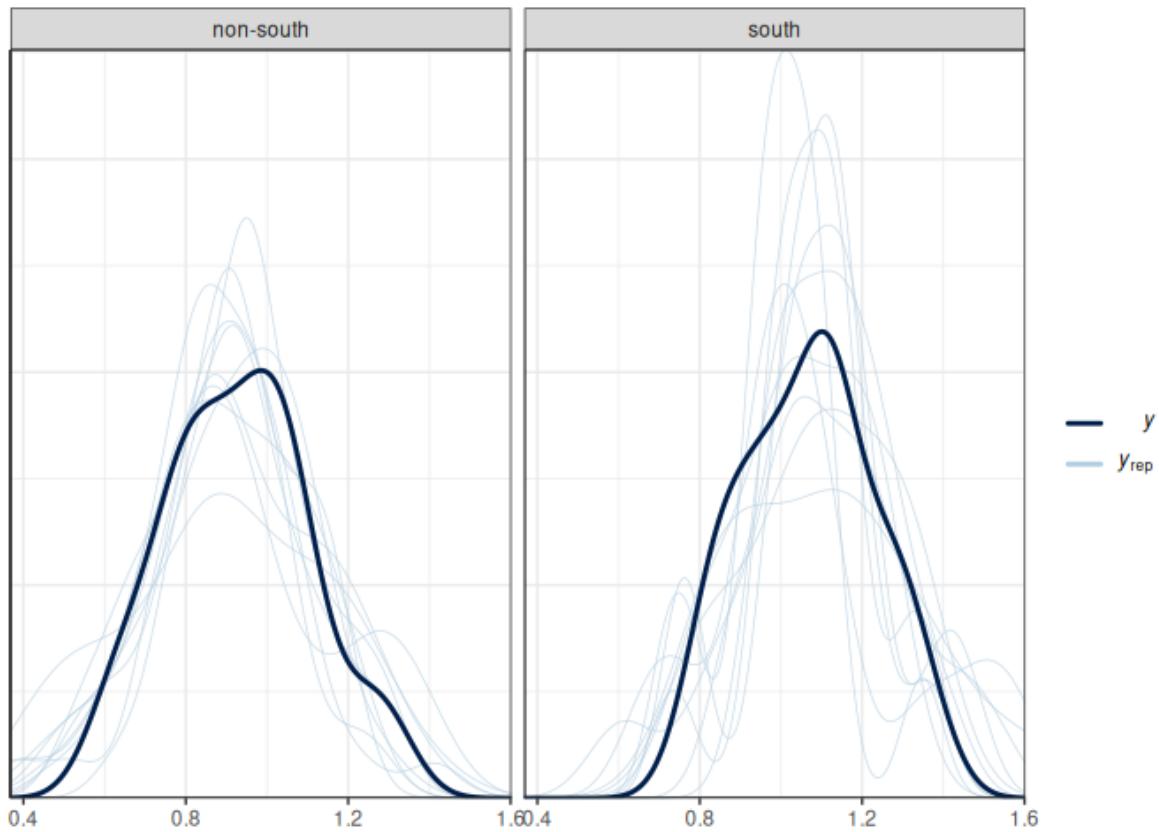
```
# PPC for the mean (it should always fit)
pp_check(m_inter, type = "stat_grouped", stat = "mean", group = "South")
```

Using all posterior draws for ppc type 'stat\_grouped' by default.

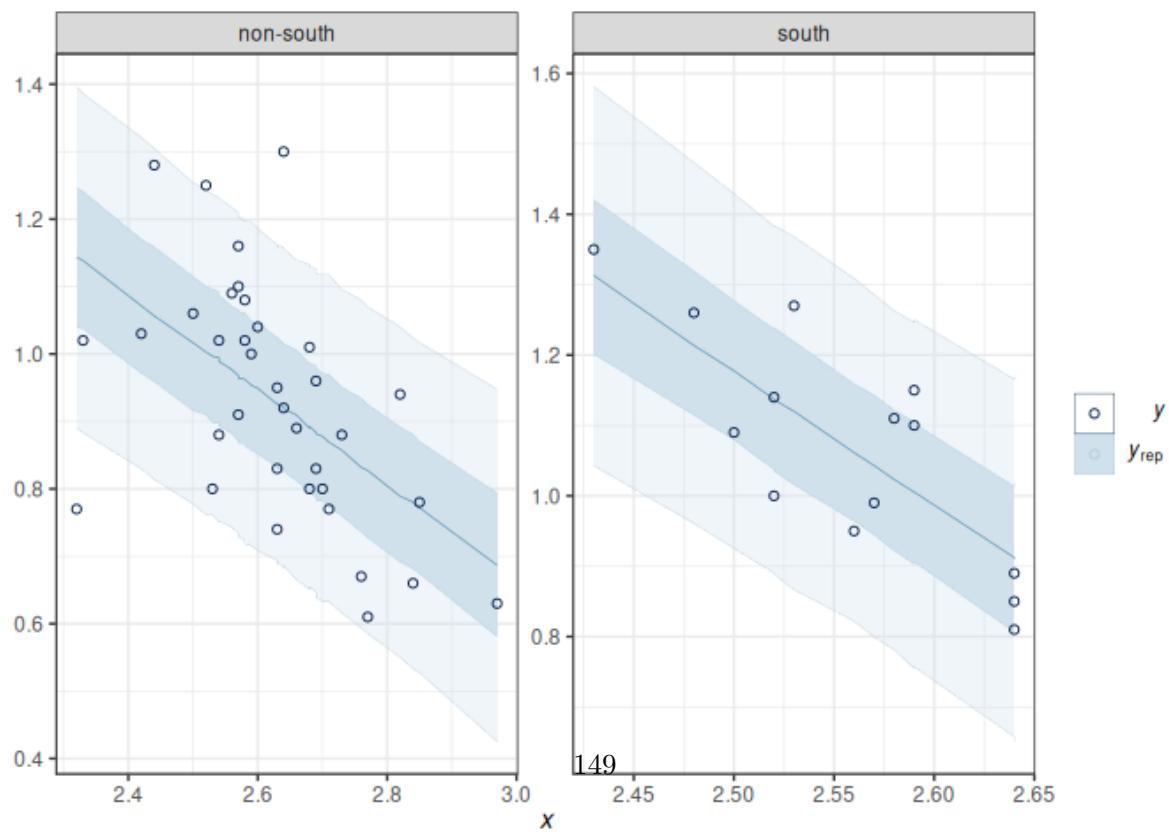
Loading required package: rstan

Loading required package: StanHeaders

rstan version 2.32.5 (Stan version 2.32.2)



(a) Density overlayed plots



(b) Prediction intervals

Figure 10.1: Posterior predictive checks for the interaction model.

```

For execution on a local, multicore CPU with excess RAM we recommend calling
options(mc.cores = parallel::detectCores()).
To avoid recompilation of unchanged Stan programs, we recommend calling
rstan_options(auto_write = TRUE)
For within-chain threading using `reduce_sum()` or `map_rect()` Stan functions,
change `threads_per_chain` option:
rstan_options(threads_per_chain = 1)

```

Attaching package: 'rstan'

The following object is masked from 'package:tidyর':

```

extract

`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.

```

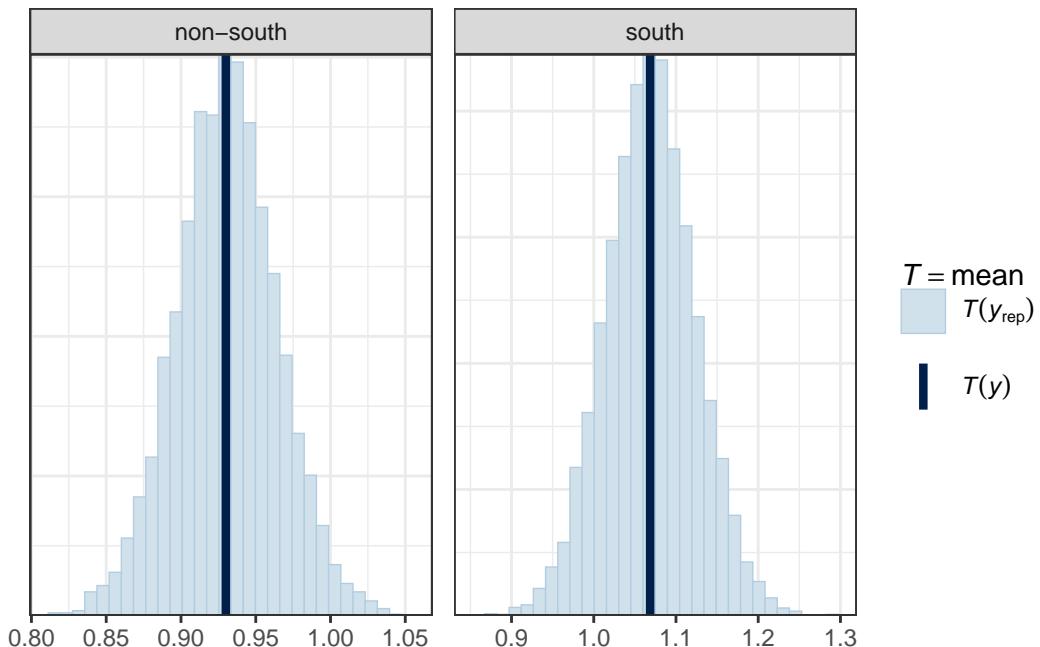


Figure 10.2: Posterior predictive check for the sample mean.

```

# PPC for the mean (it should always fit)
# PPC for the maximum and minimum values
pp_check(m_inter, type = "stat_2d", stat = c("max", "min"))

```

Using all posterior draws for ppc type 'stat\_2d' by default.

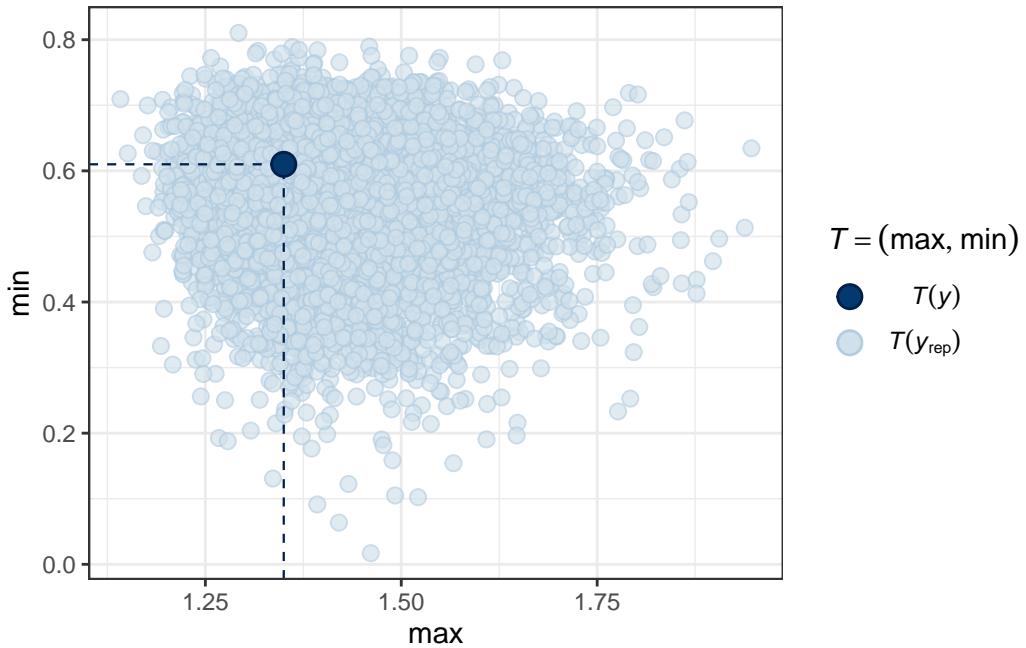


Figure 10.3: Posterior predictive check for the sample maximum and minimum.

### 10.2.2 Marginal model plots

To check the linearity assumption, we need to ensure that the conditional mean of  $Y$  fits according to the model. A marginal model plot compares the model predicted relationship between the outcome and each predictor, and the relationship obtained using nonparametric methods with smoothing.

```
pp_check(m_inter, type = "intervals_grouped",
          x = "MedianAgeMarriage", group = "South") +
  geom_smooth(se = FALSE, col = "blue") +
  geom_smooth(aes(y = y_obs), se = FALSE, col = "red", linetype = "dashed")
```

Using all posterior draws for ppc type 'intervals\_grouped' by default.

```
`geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```

```

Warning: The following aesthetics were dropped during statistical transformation: ymin,
ymax
i This can happen when ggplot fails to infer the correct grouping structure in
the data.
i Did you forget to specify a `group` aesthetic or to convert a numerical
variable into a factor?
The following aesthetics were dropped during statistical transformation: ymin,
ymax
i This can happen when ggplot fails to infer the correct grouping structure in
the data.
i Did you forget to specify a `group` aesthetic or to convert a numerical
variable into a factor?

`geom_smooth()` using method = 'loess' and formula = 'y ~ x'

Warning: The following aesthetics were dropped during statistical transformation: ymin,
ymax
i This can happen when ggplot fails to infer the correct grouping structure in
the data.
i Did you forget to specify a `group` aesthetic or to convert a numerical
variable into a factor?
The following aesthetics were dropped during statistical transformation: ymin,
ymax
i This can happen when ggplot fails to infer the correct grouping structure in
the data.
i Did you forget to specify a `group` aesthetic or to convert a numerical
variable into a factor?

# Alternative code
# plot(
#   conditional_effects(m_inter,
#     effects = "MedianAgeMarriage",
#     conditions = data.frame(South = c("south", "non-south"),
#                             cond__ = c("South", "Non-South"))
#   ),
#   plot = FALSE
# )[[1]] +
#   # Add data points
#   geom_point(
#     data = m_inter$data,
#     aes(x = MedianAgeMarriage, y = Divorce),
#     inherit.aes = FALSE

```

```

#      ) +
#      # Add smoother
#      geom_smooth(
#          data = m_inter$data,
#          aes(x = MedianAgeMarriage, y = Divorce),
#          col = "red", linetype = "dashed",
#          inherit.aes = FALSE,
#          se = FALSE) +
#      facet_wrap(~ South)

```

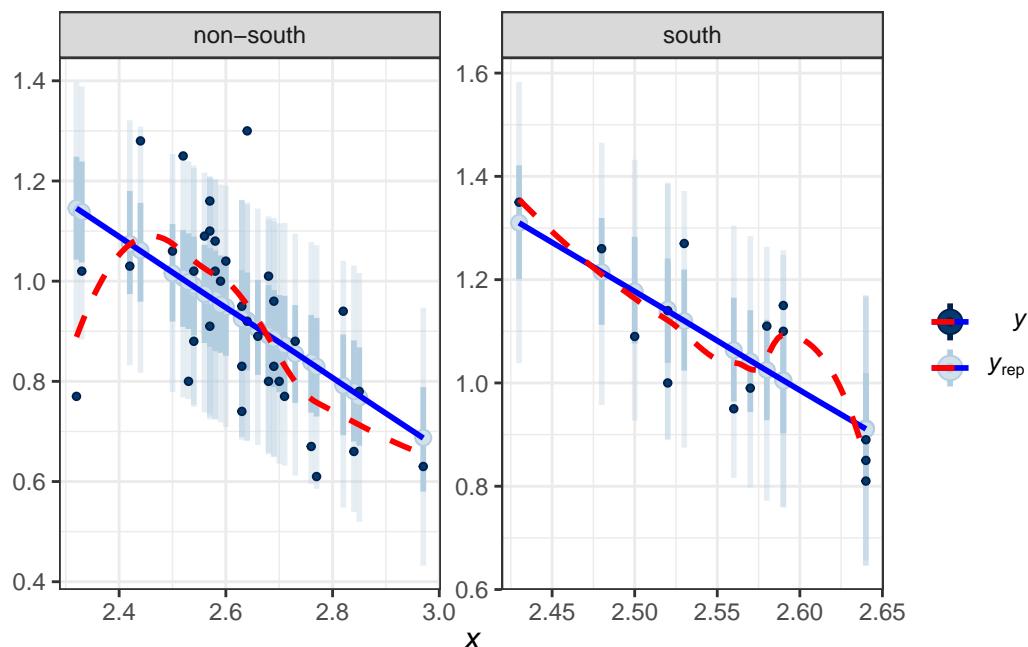


Figure 10.4: Marginal model plots with the model-implied (blue solid) and the nonparametric (red dashed) smoother.

Marginal model plots are more appropriate for ordinal or continuous predictors. As you can see above, the red line (for nonparametric fit) and the blue line (from the linear model) fit the data well, but not for the left tail area, where some of the non-Southern states have lower divorce rates than predicted. If the linearity assumption holds, these two lines should be very similar. Generally speaking, deviations in the middle indicate a strong misspecification that needs to be fixed.

Also, we want to check outliers that lie way outside the predictive interval. With a 95% predictive interval, we generally expect 5% to lie outside of the predictive interval band. In this example, we don't see a great problem with outliers.

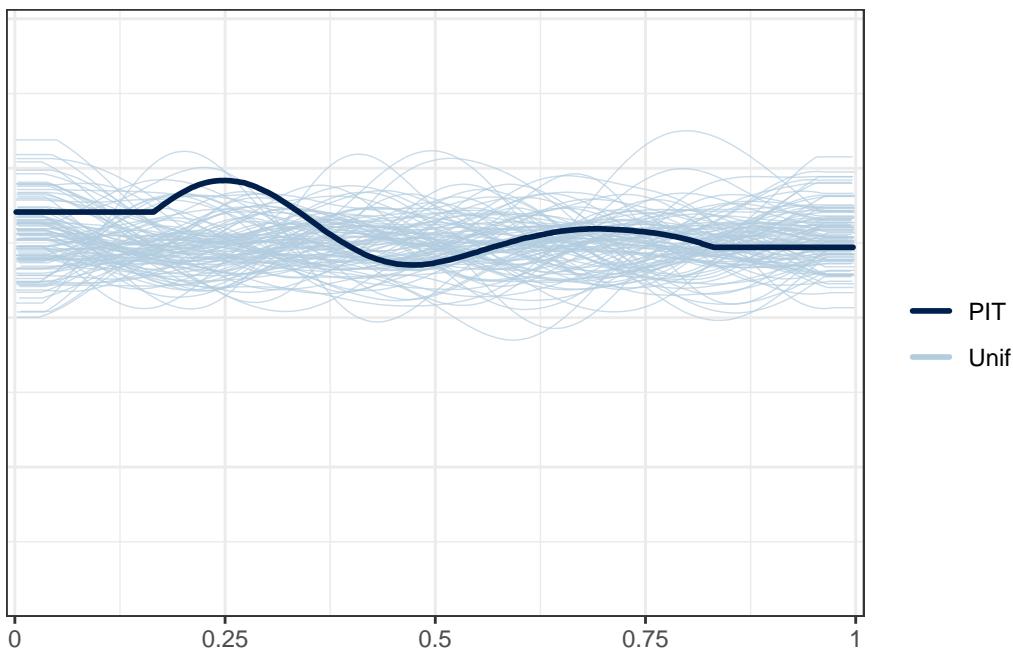
### 10.2.2.1 LOO PIT plots

Here's a check using [probability integral transform](#). Roughly speaking, you can think of it as a flattened version of the marginal model plots, where you'd like to see the darkened line be relatively flat and within the range of the model prediction.

```
pp_check(m_inter, type = "loo_pit_overlay")
```

Using all posterior draws for ppc type 'loo\_pit\_overlay' by default.

NOTE: The kernel density estimate assumes continuous observations and is not optimal for discrete data.



### 10.2.3 Residual plots

For regression analyses, one can learn a lot about model fit from the residuals, which is  $y_i - \tilde{y}_i|\theta$ , i.e., subtracting the observed  $y_i$  values by the posterior predictions. Because in Bayesian, there is not just one predicted value, but a whole predictive distribution, one also has an entire posterior distribution for each residual. Figure 10.5 is a check of the average residual  $\bar{Y}$  (i.e.,  $\bar{Y} - \hat{Y}$ ) and the true value of  $\bar{Y}$ . If the model fit the data well, the points should be scattered with no specific pattern, like in Figure 10.5.

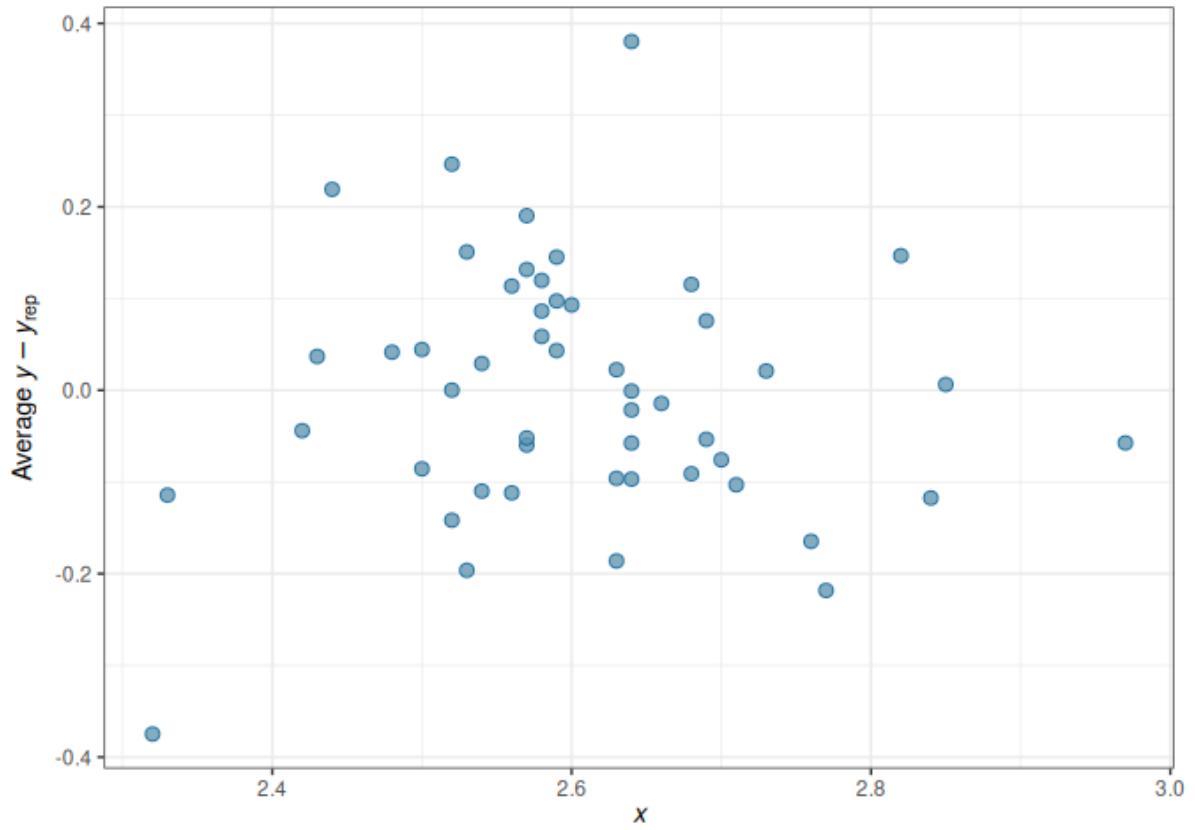


Figure 10.5: Average prediction error against the predictor. No clear pattern should be observed for a correctly specified model.

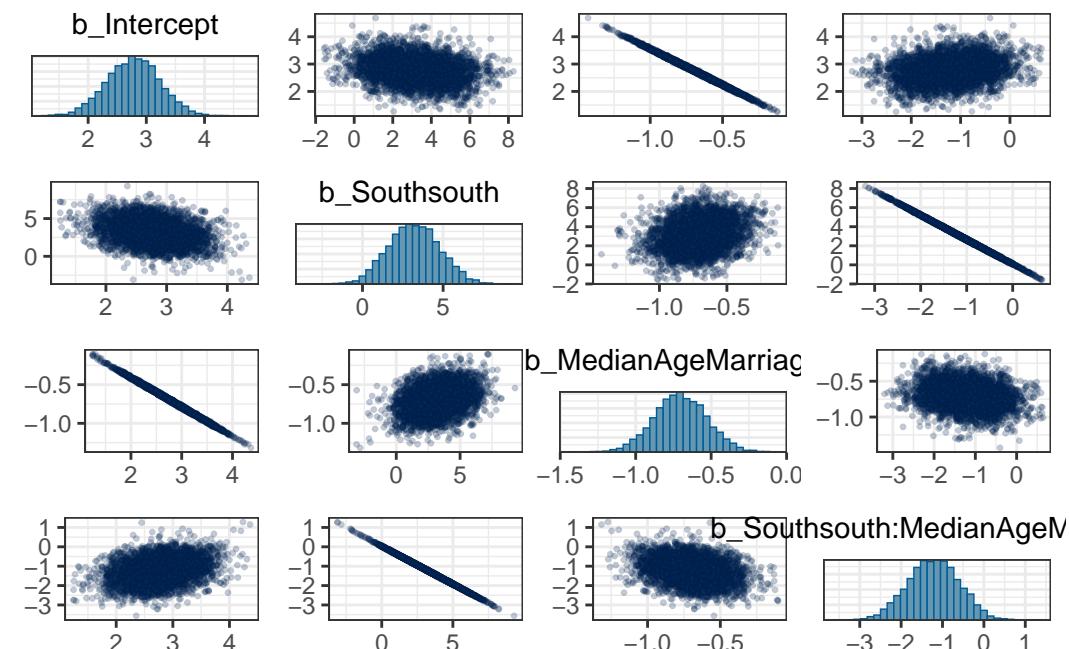
No big problem was found in the residuals. If you see that the *SD* of the residuals is not uniform, or the residuals have some non-linear relationships with the predictor, there can be some problems.

#### 10.2.4 Multicollinearity

Strictly speaking, multicollinearity is not an assumption of regression. However, especially in frequentist analysis, having predictors that are strongly correlated can increase the uncertainty of the posterior distributions of the regression coefficients. On the other hand, the use of the prior distribution in Bayesian analyses can somewhat come to the rescue, as it makes it less likely for the posteriors to have extremely large posterior mean and standard deviation

You can look at the posterior density of the coefficients to see how correlated they are:

```
pairs(m_inter,
      variable = "^b", # for all variables starting with b
      regex = TRUE,
      off_diag_args = # arguments of the scatterplots
      list(
        size = 0.5, # point size
        alpha = 0.25 # transparency
      )
)
```



If some coefficients are particularly strongly correlated, you may need to think about using a stronger prior or combining some predictors. In this case, the collinearity is a result of the interactions. Principal component and factor analysis are some approaches for that.

## 10.3 Other Topics

There are other topics we have yet to discuss here for diagnostics of multiple regression, but are just as important, including:

- Transformation (e.g., logarithm transformation with skewed outcomes and predictors, like income);
- Leverage points and influential observations (e.g., hat values, Cook's  $D$ )
- Measurement error of predictors

**Part VI**

**Week 7**

# 11 Model Comparison

## 11.1 Overfitting and Underfitting

In statistical modeling, a more complex model almost always results in a better fit to the data. Roughly speaking, a more complex model means a model with more parameters. However, as you will see later, determining the number of parameters in Bayesian analyses is not straightforward. On the extreme side, if one has 10 observations, a model with 10 parameters will perfectly predict every single data point (by just having a parameter to predict each data point). However, there are two problems with too complex a model. First, an increasingly complex model makes it increasingly hard to extract useful information from the data. Instead of describing the relationship between two variables, like `Marriage` and `Divorce`, by a straight line, one ends up with a crazy model that is difficult to make sense of. Second, as you will also see, the more complex a model, the more is the risk that it *overfits* the current data, such that it does not work for future observations.

For example, let's randomly sample 10 states in the `waffle_divorce` data set and build some models.

```
waffle_divorce <- read_delim( # read delimited files
  "data/WaffleDivorce.csv",
  delim = ";"
)

Rows: 50 Columns: 13
-- Column specification -----
Delimiter: ";"
chr (2): Location, Loc
dbl (11): Population, MedianAgeMarriage, Marriage, Marriage SE, Divorce, Div...
  i Use `spec()` to retrieve the full column specification for this data.
  i Specify the column types or set `show_col_types = FALSE` to quiet this message.

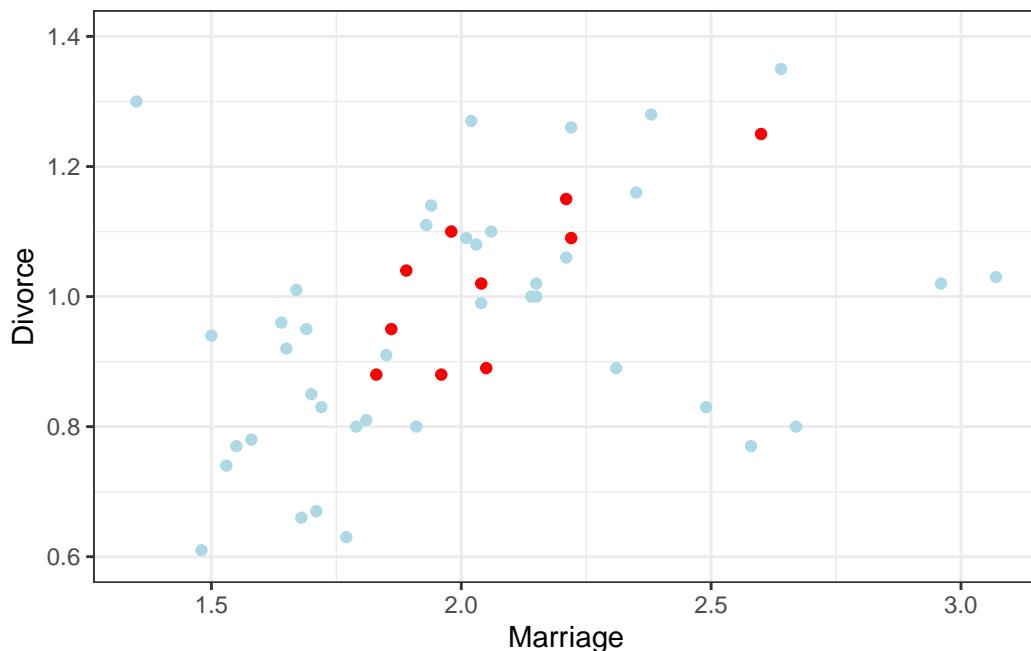
# Rescale Marriage and Divorce by dividing by 10
waffle_divorce$Marriage <- waffle_divorce$Marriage / 10
waffle_divorce$Divorce <- waffle_divorce$Divorce / 10
```

```

waffle_divorce$MedianAgeMarriage <- waffle_divorce$MedianAgeMarriage / 10
# Recode `South` to a factor variable
waffle_divorce$South <- factor(waffle_divorce$South,
  levels = c(0, 1),
  labels = c("non-south", "south")
)

set.seed(1547) # set the seed for reproducibility
# Sample 10 observations
train <- sample.int(nrow(waffle_divorce), 10L)
wd_sub <- waffle_divorce[train, ]
base <- ggplot(aes(x = Marriage, y = Divorce),
  data = wd_sub) +
  geom_point() +
  coord_cartesian(ylim = c(0.6, 1.4)) +
  xlim(range(waffle_divorce$Marriage))
ggplot(waffle_divorce,
  aes(x = Marriage, y = Divorce)) +
  geom_point(col = "lightblue") +
  geom_point(size = 1.5, data = wd_sub, col = "red") +
  coord_cartesian(ylim = c(0.6, 1.4)) +
  xlim(range(waffle_divorce$Marriage))

```



When using `Marriage` to predict `Divorce`, we can use beyond a linear regression line by using higher-order *polynomials*. For example, a second-order polynomial represents a quadratic effect (with one turning point); it goes to cubic, quartic, and more. The figure below shows the fit from a linear effect of `Marriage`, a quadratic effect, and increasingly complex models up to a sixth-degree polynomial. As you can see, as the model gets more complex, the fitted line tries to capture all the 10 points really well, with an increasing  $R^2$ . However, the standard error around the fitted line also gets larger and bizarre, meaning more uncertainty in the model parameters.

```
r2 <- function(object, newresp, newdata) {
  # Function for computing R^2
  ypred <- predict(object, newdata = newdata)
  cor(ypred, newresp)^2
}

rmse <- function(object, newresp, newdata) {
  # Function for RMSE
  ypred <- predict(object, newdata = newdata)
  sqrt(mean((ypred - newresp)^2))
}

# Create six plots through a loop
p_list <- map(1:6, function(i) {
  # Use frequentist analyses for speed
  mod <- lm(Divorce ~ poly(Marriage, degree = i), data = wd_sub)
  base +
    geom_smooth(method = "lm", formula = y ~ poly(x, i), level = .80,
                fullrange = TRUE) +
    annotate("text", x = 1.7, y = 1.4,
             label = paste0("italic(R)^2 == ",
                           round(r2(mod, wd_sub$Divorce), 2)),
             parse = TRUE) +
    annotate("text", x = 1.7, y = 1.2,
             label = paste0("RMSE == ",
                           round(rmse(mod, wd_sub$Divorce), 2)),
             parse = TRUE)
})
do.call(grid.arrange, c(p_list, nrow = 2))
```

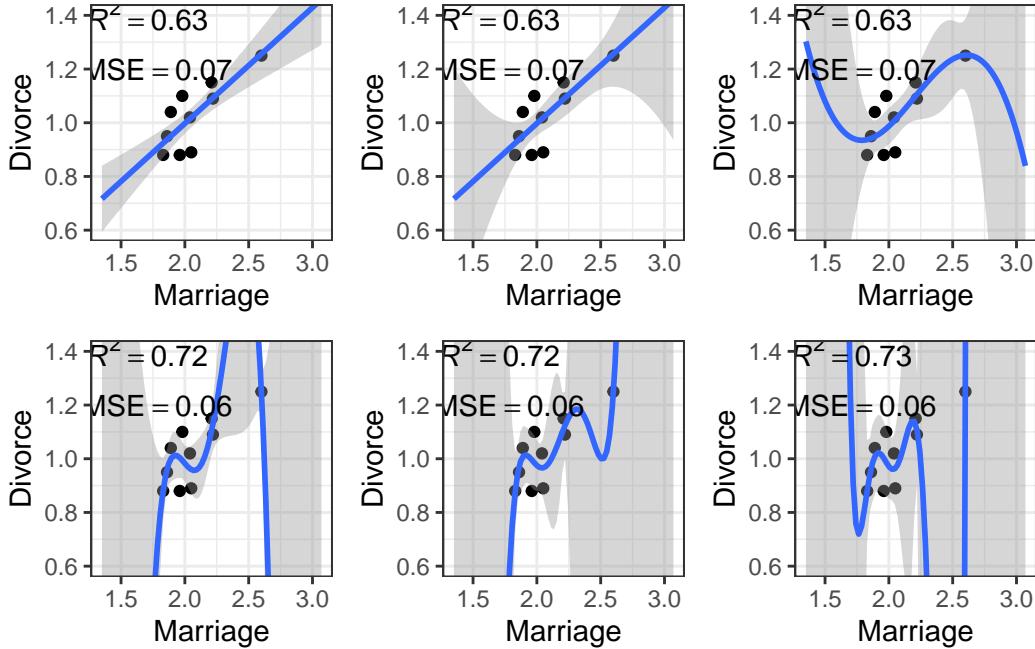


Figure 11.1: Fit of models on the 10 random cases. Top panel: linear, quadratic, and cubic; bottom panel: 4th, 5th, and 6th degree polynomials

Another way to look at model accuracy is the *Root Mean Squared Error* (RMSE), defined as the square root of the average squared prediction error. RMSE is a measure of prediction error. The smaller the RMSE, the better the prediction is. As you can see in the above figure, more complex models always reduce the RMSE in the data we use to fit the model (also called training data).

However, if I take the estimated regression line/curve based on the subsample of 10 observations, and predict the remaining cases in the data set, things will be different. As you can see in the figure below, whereas prediction error is comparable for the linear and the quadratic model, polynomials of higher degrees predict the data really badly. When you use a complex model in a data set, it tailors the coefficients to any sampling errors and noise in the data such that it will not generalize to new observations. Therefore, our goal in model comparison is to choose a model complex enough to capture the essence of the data generation process (and thus avoid *underfitting*), but not too complex such that it suffers from *overfitting*.

```
base2 <- ggplot(aes(x = Marriage, y = Divorce),
                 data = waffle_divorce[-train, ]) +
  geom_point() +
  coord_cartesian(ylim = c(0.6, 1.4)) +
  xlim(range(waffle_divorce$Marriage))
```

```

# Create six plots through a loop
p_list2 <- map(1:6, function(i) {
  # Use frequentist analyses for speed
  mod <- lm(Divorce ~ poly(Marriage, degree = i), data = wd_sub)
  # New data and response
  test_dat <- waffle_divorce[-train, ]
  ynew <- test_dat$Divorce
  base2 +
    geom_smooth(data = wd_sub, method = "lm", formula = y ~ poly(x, i),
                level = .80, fullrange = TRUE) +
    annotate("text", x = 1.7, y = 1.4,
             label = paste0("italic(R)^2 == ",
                           round(r2(mod, ynew, test_dat), 2)),
             parse = TRUE) +
    annotate("text", x = 1.7, y = 1.2,
             label = paste0("RMSE == ",
                           round(rmse(mod, ynew, test_dat), 2)),
             parse = TRUE)
})
do.call(grid.arrange, c(p_list2, nrow = 2))

```

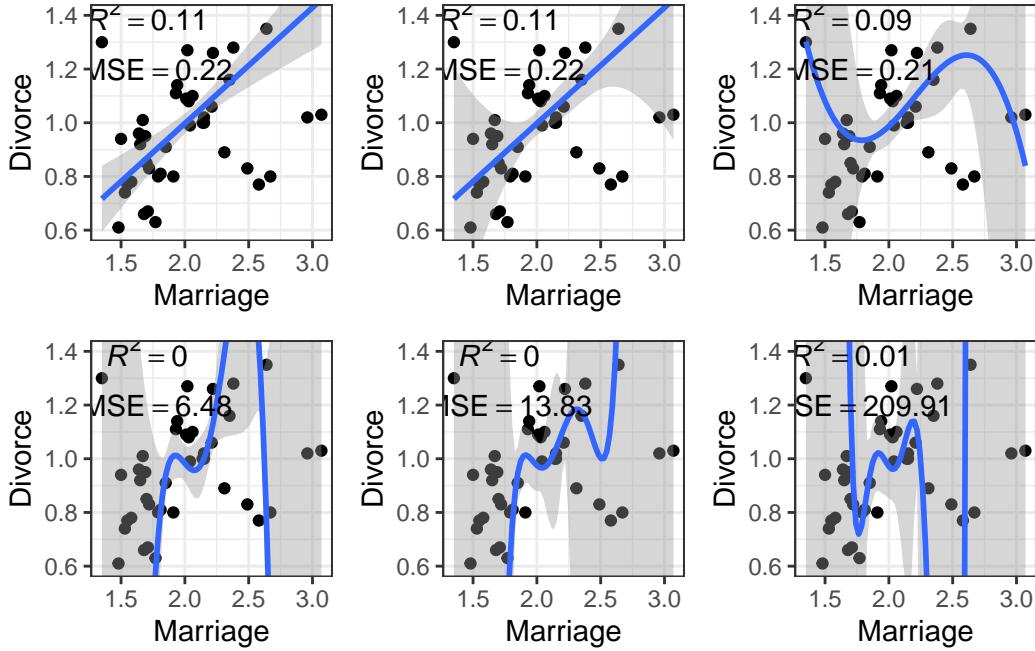


Figure 11.2: Using the regression lines based on 10 random cases to predict the remaining 40 cases. Top panel: linear, quadratic, and cubic; bottom panel: 4th, 5th, and 6th degree polynomials

The goal of statistical modeling is to choose an optimal model between the overfitting/underfitting dichotomy. In machine learning, this is also commonly referred to as the bias-variance trade-off, as a model that is too simple tends to produce biased predictions because it does not capture the essence of the data generating process. In contrast, an overly complex model is unbiased but results in a lot of uncertainty in the prediction because there are too many unnecessary components that can affect predictions, as indicated in the confidence bands around the 6th-degree polynomial line.

Polynomials of varying degrees are merely one example of comparing simple to complex models. You can think about:

- models with and without interactions,
- models with a few predictors versus hundreds of predictors,
- regression analyses versus multilevel models, etc.

Whereas one can always avoid underfitting by fitting a more and more complex model, we need tools to keep us from overfitting. This lecture is about finding an optimal model that avoids overfitting and avoids underfitting. You will learn to perform model comparisons with information criteria to find a model that has a better balance between overfitting and underfitting.

## 11.2 Kullback-Leibler Divergence

When comparing models (e.g., linear vs. quadratic), we prefer models closer to the “true” data-generating process. To do so, we need some ways to quantify the degree of “closeness” to the true model. In this context, models comprise both the distributional family *and* the parameter values. For example, the model  $y_i \sim N(5, 2)$  is a different model than  $y_i \sim N(3, 2)$ , which is a different model than  $y_i \sim \text{Gamma}(2, 2)$ . The first two have the same family but different parameter values (different means, same *SD*). In contrast, the last two have different distributional families (Normal vs. Gamma).

To measure the degree of “closeness” between two models,  $M_0$  and  $M_1$ , by far the most popular metric in statistics is the *Kullback-Liebler Divergence* (or Kullback-Liebler discrepancy;  $D_{\text{KL}}$ ). By definition,

$$\begin{aligned} D_{\text{KL}}(M_0 \mid M_1) &= \int_{-\infty}^{\infty} p_{M_0}(\mathbf{y}) \log \frac{p_{M_0}(\mathbf{y})}{p_{M_1}(\mathbf{y})} d\mathbf{y} \\ &= \int_{-\infty}^{\infty} p_{M_0}(\mathbf{y}) \log p_{M_0}(\mathbf{y}) d\mathbf{y} - \int_{-\infty}^{\infty} p_{M_0}(\mathbf{y}) \log p_{M_1}(\mathbf{y}) d\mathbf{y}. \end{aligned}$$

Note that strictly speaking,  $D_{\text{KL}}$  cannot be called a “distance” between two models because in general,  $D_{\text{KL}}(M_0 \mid M_1) \neq D_{\text{KL}}(M_1 \mid M_0)$ . As an example, assume that the data are generated by a true model  $M_0$ , and we have two candidate models  $M_1$  and  $M_2$ , where

- $M_0 : y \sim N(3, 2)$
- $M_1 : y \sim N(3.5, 2.5)$
- $M_2 : y \sim \text{Cauchy}(3, 2)$

```
ggplot(data.frame(x = c(-3, 9)), aes(x = x)) +
  stat_function(fun = dnorm, args = list(mean = 3, sd = 2),
                aes(col = "M0"), linewidth = 1) +
  stat_function(fun = dnorm, args = list(mean = 3.5, sd = 2.5),
                aes(col = "M1"), linewidth = 2) +
  stat_function(fun = dcauchy, args = list(location = 3, scale = 2),
                aes(col = "M2"), linewidth = 2) +
  scale_color_manual(values = c("black", "red", "blue"),
                     labels = c("M0", "M1", "M2")) +
  labs(x = "y", y = "density", col = NULL)
```

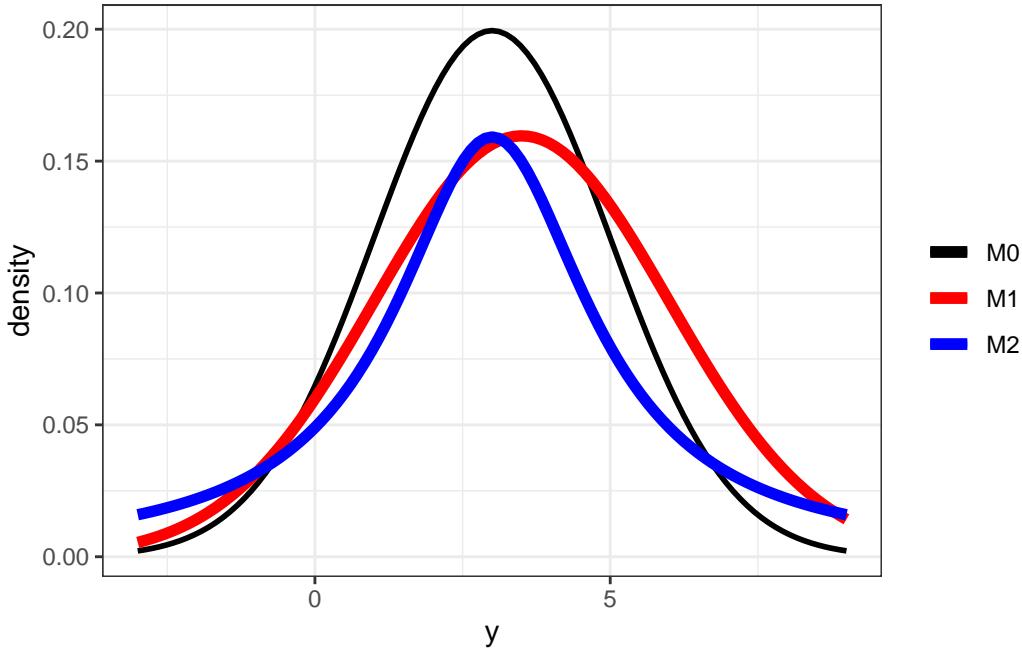


Figure 11.3: Density for  $M_0$ ,  $M_1$ , and  $M_2$

```
f1 <- function(x) {
  dnorm(x, 3, 2) * (dnorm(x, 3, 2, log = TRUE) -
    dnorm(x, 3.5, 2.5, log = TRUE))
}

f2 <- function(x) {
  dnorm(x, 3, 2) * (dnorm(x, 3, 2, log = TRUE) -
    dcauchy(x, 3, 2, log = TRUE))
}
```

One can compute that  $D_{\text{KL}}(M_0 \mid M_1) = 0.0631436$  and  $D_{\text{KL}}(M_0 \mid M_2) = 0.2592445$ , and so  $M_1$  is a better model than  $M_2$ .

Note that in the expression of  $D_{\text{KL}}$ , when talking about the same target model, the first term is always the same and describes the “true” model,  $M_0$ . Therefore, it is sufficient to compare models on the second term,  $\int_{-\infty}^{\infty} p_{M_0}(\mathbf{y}) \log p_{M_1}(\mathbf{y}) d\mathbf{y}$ , which can also be written as  $E = [\log p_{M_1}(\mathbf{y})]$ , i.e., the *expected log predictive density (elpd)*. In other words, a model with a larger elpd is preferred over a model with a smaller elpd.

However, we don’t know what  $M_0$  is in real data analysis. If we knew, then we would just need to choose  $M_0$  as our model, and there will be no need for model comparisons. In addition, even if we know that the true model is, e.g., a normal model (which never happens in real data analysis), we still need to estimate the parameter values, and the estimates will not be exactly

the same as the true parameter values. However, elpd is defined as the expected value over the true predictive distribution,  $p_{M_0}(y)$ , which cannot be obtained without knowing what  $M_0$  is.

So instead, we need to estimate the elpd. A naive way to estimate it is to use the data distribution in place of the true model, but that will lead to an overly optimistic estimate as the sample data are noisy. Computing elpd this way will always favor a more complex model. The ideal way is to collect data on a new, independent sample that share the same data generating process as the current sample, and estimate elpd on the new sample. This is called *out-of-sample validation*. The problem, of course, is that we usually do not have the resources to collect a new sample.

Therefore, statisticians have worked hard to find ways to estimate elpd from the current sample, and there are two broad approaches:

- Information criteria: AIC, DIC, and WAIC, which estimate the elpd in the current sample, minus a correction factor
- Cross-validation, which splits the current sample into  $K$  parts, estimates the parameters in  $K - 1$  parts, and estimates the elpd in the remaining part. A special case is when  $K = N$ , each time one uses  $N - 1$  data points to estimate the model parameters, and estimate the elpd for the observation that was left out. This is called *leave-one-out* cross-validation (LOO-CV).

### 11.3 Deviance

Without going too deep into the underlying math, it can be shown that a good estimate of elpd is

$$\sum_{i=1}^n \log p_{M_1}(y_i) - p,$$

where  $p$  is some measure of the number of parameters in  $M_1$ . The first term is the likelihood of the model in the current sample. The second term is an adjustment factor so that the quantity above represents the average likelihood of the model *in a new sample*. It is more common to work with *deviance* by multiplying the log-likelihood by  $-2$ , i.e.,

$$D = -2 \sum_{i=1}^n \log p_{M_1}(y_i).$$

### 11.3.1 Experiment on Deviance

Now, let's check the in-sample deviance and out-of-sample deviance of our `waffle_divorce` data with different polynomial functions. Here is a sample function for computing elpd (with frequentist, just for speed) for polynomials of different degrees:

```
# Function for computing deviance with different polynomial
deviance_divorce <- function(degree = 1,
                                train = 10,
                                y = waffle_divorce$Divorce,
                                x = waffle_divorce$Marriage) {
  N <- length(y)
  # get training sample
  if (length(train) == 1) {
    train <- sample.int(N, train)
  }
  ntrain <- length(train)
  # Obtain design matrix
  X <- cbind(1, poly(x, degree, simple = TRUE))
  # Get elpd for training sample
  Xtrain <- X[train, ]
  ytrain <- y[train]
  betahat <- qr.solve(Xtrain, ytrain) # estimated betas
  res_train <- ytrain - Xtrain %*% betahat
  sigmahat <- sqrt(sum(res_train^2) /
    (ntrain - 1 - degree)) # estimated sigma
  deviance_train <- -2 * sum(dnorm(res_train, sd = sigmahat, log = TRUE))
  res_test <- y[-train] - X[-train, ] %*% betahat
  deviance_test <- -2 * sum(dnorm(res_test, sd = sigmahat, log = TRUE))
  data.frame(
    degree = degree,
    sample = c("in-sample", "out-of-sample"),
    deviance = c(deviance_train / ntrain,
                deviance_test / (N - ntrain)))
}
```

Below shows the in-sample and out-of-sample elpd for the linear model:

```
deviance_divorce(degree = 1, train = train)
```

degree	sample	deviance
--------	--------	----------

```

1      1    in-sample -2.37580
2      1 out-of-sample  3.78599

```

And for quadratic:

```
deviance_divorce(degree = 2, train = train)
```

	degree	sample	deviance
1	2	in-sample	-2.342268
2	2	out-of-sample	3.048412

In general, as you can see, the deviance is smaller for the current data than for the hold-out data. Note that because the training and testing data sets have different sizes, I divided the deviance by the sample size so that they can be compared.

Now let's run an experiment to check the elpd with different degrees polynomial, with a training sample size of 25:

```

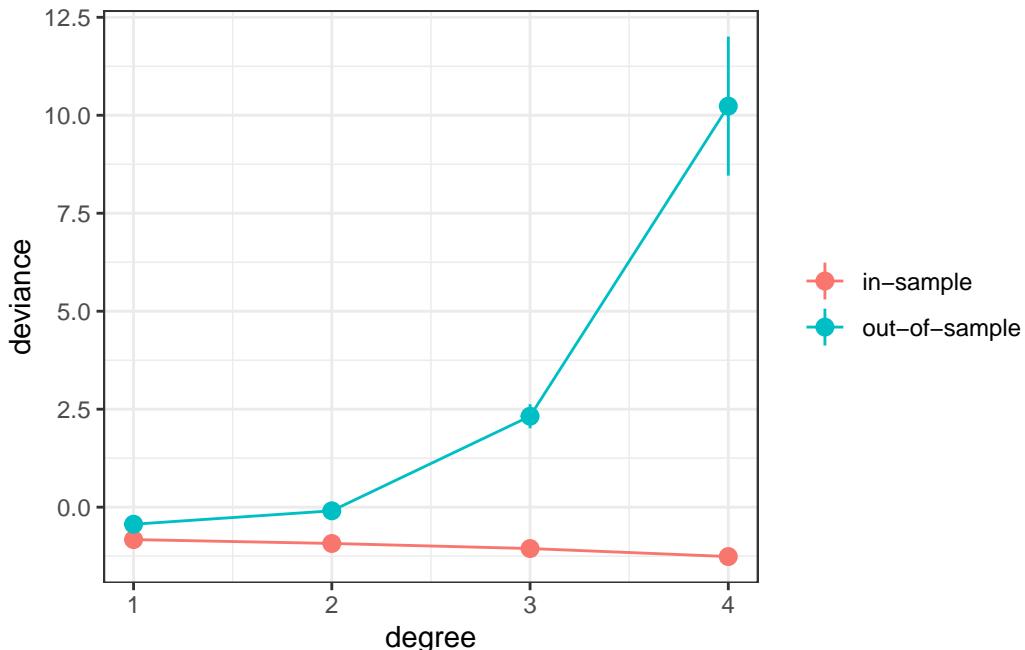
set.seed(1733)
# Use the `map` function to run different polynomials, and use the `rerun`
# function run the deviance 100 times. The code below runs `deviance_divorce` by
# randomly sampling 25 training samples 100 times, and compute the in-sample
# and out-of-sample deviance for each.
# rerun(100, deviance_divorce(degree = 1, train = 25L)) |>
#     bind_rows()
# Now run 1 to 4 degree polynomial, each 1000 times:
dev_list <- lapply(1:4, FUN = function(p) {
  results <- replicate(1000, deviance_divorce(degree = p, train = 25L), simplify = FALSE)
  do.call(rbind, results)
})
dev_df <- do.call(rbind, dev_list)
# Plot the results
dev_df |>
  ggplot(aes(x = degree, y = deviance, col = sample)) +
  stat_summary() +
  stat_summary(geom = "line") +
  labs(col = NULL)

```

```

No summary function supplied, defaulting to `mean_se()`
No summary function supplied, defaulting to `mean_se()`

```



As you can see, the in-sample deviance (red line) keeps decreasing, indicating that a more complex model fits the data better, which is always the case. So if one were to use deviance to determine what model is optimal, one would always choose the most complex model, just like using  $R^2$  (indeed, for linear models, deviance is basically the same as  $R^2$ ).

Now, look at the blue line, which represents the deviance computed using the coefficients obtained from the training set but applied to the remaining data. As you can see, the deviance achieves its minimum around the linear and the quadratic model, and starts to increase, meaning that the more complex models do not fit the hold-out data.

A statistical model is used to learn something from a data set that can generalize to other observations. Therefore, we should care about the blue line, instead of the red one. The indices you will see in the remaining of this note are all attempts to approximate the blue line.

More complex models always fit the current data better, but may not generalize to other data. In other words, models that are too complex are not generalizable.

## 11.4 Information Criteria

We will illustrate the computation of information criteria with `Marriage` predicting `Divorce`:

```

m1 <- brm(Divorce ~ Marriage, data = waffle_divorce,
            prior = c(prior(student_t(4, 0, 5), class = "Intercept"),
                      prior(normal(0, 2), class = "b"),
                      prior(student_t(4, 0, 1), class = "sigma")),
            iter = 4000,
            seed = 2302,
            file = "07_m1"
)

```

### 11.4.1 Akaike Information Criteria (AIC)

Multiplying the quantity of  $\text{elpd} - p$  by  $-2$ , or deviance  $+ 2p$ , with the deviance obtained using the maximum likelihood estimates (MLEs) for the parameters, gives you the formula for AIC:

$$\text{AIC} = D(\hat{\theta}) + 2p,$$

and  $p$  in AIC is just the number of parameters. As we have multiplied by a negative number, maximizing the estimate of elpd is equivalent to minimizing the AIC, so one would prefer a model with the smallest AIC.

The AIC is not Bayesian because it only uses point estimates (MLEs) of parameters rather than their posterior distributions. Also, it does not take into account any prior information.

```

# Frequentist model
m1_freq <- lm(m1$formula, data = m1$data)
AIC(m1_freq)

```

[1] -30.96869

### 11.4.2 Deviance Information Criteria (DIC)

The definition of AIC assumes that the parameter estimates are known or are maximum likelihood estimates. The DIC, instead, replaces those with the posterior distribution of the parameters. The general formula for DIC is

$$\text{DIC} = \text{E}(D | \mathbf{y}) + 2p_D,$$

where  $p_D$  is the effective number of parameters estimated in the Markov chain. Although DIC does take into account the prior distributions, it does not consider the full posterior distributions of the parameters.

```
# Function to compute DIC
dic_brmsfit <- function(object) {
  Dbar <- -2 * mean(rowSums(log_lik(object)))
  res <- residuals(object)[ , "Estimate"]
  sigma <- posterior_summary(object, variable = "sigma")[ , "Estimate"]
  Dhat <- -2 * sum(dnorm(res, sd = sigma, log = TRUE))
  p <- Dbar - Dhat
  elpd <- Dhat / -2 - p
  data.frame(elpd_dic = elpd, p_dic = p, dic = Dhat + 2 * p,
             row.names = "Estimate")
}
dic_brmsfit(m1)
```

Loading required package: rstan

Loading required package: StanHeaders

rstan version 2.32.5 (Stan version 2.32.2)

For execution on a local, multicore CPU with excess RAM we recommend calling  
options(mc.cores = parallel::detectCores()).

To avoid recompilation of unchanged Stan programs, we recommend calling

rstan\_options(auto\_write = TRUE)

For within-chain threading using `reduce\_sum()` or `map\_rect()` Stan functions,  
change `threads\_per\_chain` option:

rstan\_options(threads\_per\_chain = 1)

Attaching package: 'rstan'

The following objects are masked from 'package:posterior':

ess\_bulk, ess\_tail

```
The following object is masked from 'package:tidyr':
```

```
extract
```

```
elpd_dic    p_dic      dic  
Estimate 15.36384 3.109187 -30.72769
```

### 11.4.3 Watanabe-Akaike Information Criteria (WAIC)

A further modification is to use the *log pointwise posterior predictive density*, with the effective number of parameters computed using the posterior variance of the likelihood.

$$\text{WAIC} = -2 \sum_{i=1}^n \log E[p(y_i | , \mathbf{y})] + 2p_{\text{WAIC}},$$

where  $E[p(y_i | , \mathbf{y})]$  is the posterior mean of the likelihood of the  $i$ th observation. The WAIC incorporates prior information, and the use of pointwise likelihood makes it more robust when the posterior distributions deviate from normality. In general, WAIC is a better estimate of the out-of-sample deviance than AIC and DIC.

```
waic(m1) # built-in function in brms
```

```
Computed from 8000 by 50 log-likelihood matrix.
```

	Estimate	SE
elpd_waic	15.2	4.9
p_waic	3.2	0.9
waic	-30.3	9.9

```
1 (2.0%) p_waic estimates greater than 0.4. We recommend trying loo instead.
```

### 11.4.4 Leave-One-Out Cross-Validation

The idea of cross-validation is to split the sample so that it imitates the scenario of estimating the parameters in part of the data and predicting the remaining part. The part used for estimation is called the *training set*, and the part used for prediction is called the *validation set*. Leave-one-out information criteria (LOO-IC) means that one uses  $N - 1$  observations as the training set and 1 observation as the validation sample and repeat the process  $N$  times so that a different observation is being predicted each time. Adding up the prediction results

will give an estimate of elpd that closely approximates the results that would be obtained by collecting new data and doing the validation. To make it more concrete, we can go back to the `waffle_divorce` data with `Marriage` predicting `Divorce`. We can do this for case #1 (Alabama), as an example:

```
# Estimate the model without case #1
m1_no1 <- update(m1, newdata = waffle_divorce[-1, ])
```

```
# The log predictive density for case #1
mean(log_lik(m1_no1, newdata = waffle_divorce[1, ]))
```

```
[1] -0.7965575
```

Because LOO-IC requires fitting the model  $N$  times, it is generally very computationally intensive. There are, however, shortcuts for some models to make the computation faster. WAIC can also be treated as a fast approximation of LOO-IC, although LOO-IC is more robust and will be a better estimate of out-of-sample deviance. The `loo` package uses the so-called Pareto smoothed importance sampling (PSIS) to approximate LOO-IC without repeating the process  $N$  times.

Here is the LOO-IC for the model:

```
loo(m1)
```

```
Computed from 8000 by 50 log-likelihood matrix.
```

	Estimate	SE
<code>elpd_loo</code>	15.1	5.0
<code>p_loo</code>	3.3	1.0
<code>looic</code>	-30.2	9.9
<hr/>		
MCSE of <code>elpd_loo</code> is 0.0.		
MCSE and ESS estimates assume MCMC draws ( <code>r_eff</code> in [0.7, 1.0]).		
All Pareto k estimates are good (k < 0.7).		
See <code>help('pareto-k-diagnostic')</code> for details.		

You can save the WAIC and the LOO-IC information to the fitted result:

```
m1 <- add_criterion(m1, criterion = c("loo", "waic"))
```

See Vehtari et al. (2017) for more discussions on WAIC and LOO-IC.

---

### 11.4.5 Example

Consider four potential models in predicting Divorce:

$$\text{Divorce}_i \sim N(\mu_i, \sigma)$$

- M1: Marriage
- M2: Marriage, South, Marriage  $\times$  South
- M3: South, smoothing spline of Marriage by South
- M4: Marriage, South, MedianAgeMarriage, Marriage  $\times$  South, Marriage  $\times$  MedianAgeMarriage, South  $\times$  MedianAgeMarriage, Marriage  $\times$  South  $\times$  MedianAgeMarriage

```
# Note, m1 has been fit before; the `update()` function
# can be used to simply change the formula, and brms will
# determine whether it needs re-compiling.
# M2: Add South and interaction
m2 <- update(m1, formula = Divorce ~ Marriage * South,
             newdata = waffle_divorce)
m2 <- add_criterion(m2, c("loo", "waic"))
```

Warning:

1 (2.0%) p\_waic estimates greater than 0.4. We recommend trying loo instead.

```
# M3: Spline function for Marriage
m3 <- update(m1, formula = Divorce ~ South + s(Marriage, by = South),
             newdata = waffle_divorce,
             control = list(adapt_delta = .999))
m3 <- add_criterion(m3, c("loo", "waic"))
```

Warning: Found 3 observations with a pareto\_k > 0.7 in model 'm3'. We recommend to set 'moment\_match = TRUE' in order to perform moment matching for problematic observations.

```
Warning:  
4 (8.0%) p_waic estimates greater than 0.4. We recommend trying loo instead.
```

```
# M4: Three-way interactions  
m4 <- update(m1, formula = Divorce ~ Marriage * MedianAgeMarriage * South,  
             newdata = waffle_divorce,  
             control = list(max_treedepth = 12)) # increase due to warning  
m4 <- add_criterion(m4, c("loo", "waic"))
```

```
Warning: Found 1 observations with a pareto_k > 0.7 in model 'm4'. We recommend  
to set 'moment_match = TRUE' in order to perform moment matching for  
problematic observations.
```

```
Warning:  
3 (6.0%) p_waic estimates greater than 0.4. We recommend trying loo instead.
```

The first model only has `Marriage` as a predictor, which means that the coefficients for `South` and `MedianAgeMarriage` are assumed to be zero. The second model added `South` and its interaction with `Marriage` as a predictor. The third model includes a smoothing spline term (a flexible non-linear function, within the class of linear models), whereas the fourth model also includes `MedianAgeMarriage` and all two-way and three-way interactions. Now, we can compare the four models:

```
loo_compare(m1, m2, m3, m4)
```

	elpd_diff	se_diff
m4	0.0	0.0
m2	-5.5	4.0
m3	-6.1	4.0
m1	-8.6	4.2

```
# m4 is the best
```

```
msummary(list(M1 = m1, M2 = m2, M3 = m3, M4 = m4),  
         estimate = "{estimate} [{conf.low}, {conf.high}]",  
         statistic = NULL, fmt = 2)
```

```
Warning:  
'modelsummary` uses the `performance` package to extract goodness-of-fit  
statistics from models of this class. You can specify the statistics you wish
```

	M1	M2	M3
b_Intercept	0.61 [0.35, 0.87]	0.66 [0.41, 0.93]	0.94 [0.89, 0.99]
b_Marriage	0.18 [0.05, 0.31]	0.13 [0.01, 0.26]	
sigma	0.17 [0.14, 0.21]	0.16 [0.13, 0.20]	0.15 [0.12, 0.19]
b_Southsouth		-0.62 [-1.43, 0.21]	0.10 [-0.02, 0.22]
b_Marriage × Southsouth		0.37 [-0.04, 0.76]	
bs_sMarriage × SouthnonMsouth_1			-0.45 [-3.32, 1.44]
bs_sMarriage × Southsouth_1			1.27 [-2.09, 3.59]
sds_sMarriageSouthnonMsouth_1			0.86 [0.05, 2.64]
sds_sMarriageSouthsouth_1			0.49 [0.02, 2.65]
b_MedianAgeMarriage			
b_Marriage × MedianAgeMarriage			
b_MedianAgeMarriage × Southsouth			
b_Marriage × MedianAgeMarriage × Southsouth			
Num.Obs.	50	50	50
R2	0.139	0.308	0.390
R2 Adj.	0.069	0.207	0.160
ELPD	15.1	18.2	17.6
ELPD s.e.	5.0	5.5	5.9
LOOIC	-30.2	-36.5	-35.2
LOOIC s.e.	9.9	11.1	11.7
WAIC	-30.3	-36.7	-36.7
RMSE	0.17	0.15	0.14

to compute by supplying a `metrics` argument to `modelsummary`, which will then push it forward to `performance`. Acceptable values are: "all", "common", "none", or a character vector of metrics names. For example: `modelsummary(mod, metrics = c("RMSE", "R2"))` Note that some metrics are computationally expensive. See `?performance::performance` for details.

This warning appears once per session.

Model 4 has the lowest LOO-IC, so one may conclude that Model 4 is the best model among the four, **for prediction purposes**.

# 12 Stacking, Regularization, and Variable Selection

## 12.1 Stacking/Model Averaging

Sometimes it may not be a good practice to only choose one model with low WAIC or LOO-IC, especially when several models have very similar WAIC/LOO-IC, but they make somewhat different predictions. Instead, we can perform *stacking* or *model averaging* by weighting the *predictions* from multiple models, using weights that are based on their information criteria performance. Stacking approaches this by optimizing the leave-one-out mean squared error in the resulting prediction, whereas model averaging preserves the uncertainty and was not optimized for that task. The technical details can be found in Yao et al. (2018).

Note that the conventional Bayesian model averaging used the posterior model probability (Hoeting et al., 1999), which are approximated by the BIC. The discussion in this note is based on more recent discussion in, e.g., Yao et al. (2018).

We'll use a data set `kidiq` that is used in the textbook by Gelman et al. (2021), which can be downloaded and imported with the direct link:

```
kidiq <- haven::read_dta("http://www.stat.columbia.edu/~gelman/arm/examples/child.iq/kidiq.dta")
head(kidiq)
```

```
# A tibble: 6 x 5
  kid_score mom_hs mom_iq mom_work mom_age
  <dbl>    <dbl>   <dbl>     <dbl>    <dbl>
1       65      1   121.       4     27
2       98      1   89.4      4     25
3       85      1   115.       4     27
4       83      1   99.4      3     25
5      115      1   92.7      4     27
6       98      0   108.       1     18
```

Let's run four models. First rescale some of the variables:

```

kidiq100 <- kidiq |>
  mutate(mom_iq = mom_iq / 100, # divid mom_iq by 100
         kid_score = kid_score / 100, # divide kid_score by 100
         mom_iq_c = mom_iq - 1,
         mom_hs = factor(mom_hs, labels = c("no", "yes")),
         mom_age_c = (mom_age - 18) / 10)

```

I will run four models, which is from the last note

$$\text{kidscore}_i \sim N(\mu_i, \sigma)$$

$$\begin{aligned}\mu_i &= \beta_0 + \beta_1(\text{mom\_iq}_i) \\ \mu_i &= \beta_0 + \beta_1(\text{mom\_iq}_i) + \beta_2(\text{mom\_hs}_i) \\ \mu_i &= \beta_0 + \beta_1(\text{mom\_iq}_i) + \beta_2(\text{mom\_hs}_i) + \beta_3(\text{mom\_iq}_i \times \text{mom\_hs}_i) \\ \mu_i &= \beta_0 + \beta_1(\text{mom\_iq}_i) + \beta_2(\text{mom\_hs}_i) + \beta_3(\text{mom\_iq}_i \times \text{mom\_hs}_i) + \beta_4(\text{mom\_age}_i)\end{aligned}$$

```

m1 <- brm(kid_score ~ mom_iq_c,
            data = kidiq100,
            prior = c(
              prior(normal(0, 1), class = "Intercept"),
              prior(normal(0, 1), class = "b"),
              prior(student_t(4, 0, 1), class = "sigma")
            ),
            seed = 2302,
            file = "07b_m1"
)
m1 <- add_criterion(m1, c("loo", "waic"))
# Use `update` will sometimes avoid recompiling
m2 <- update(m1, kid_score ~ mom_iq_c + mom_hs,
             newdata = kidiq100,
             file = "07b_m2"
)

```

The desired updates require recompiling the model

```

m2 <- add_criterion(m2, c("loo", "waic"))
m3 <- update(m2, kid_score ~ mom_iq_c * mom_hs,
             prior = c(prior(normal(0, 0.5),
                           class = "b",

```

```

        coef = "mom_iq_c:mom_hsyes"
    )),
    file = "07b_m3"
)

```

The desired updates require recompiling the model

```

m3 <- add_criterion(m3, c("loo", "waic"))
m4 <- update(m3, kid_score ~ mom_iq_c * mom_hs + mom_age_c,
  newdata = kidiq100,
  file = "07b_m4"
)

```

The desired updates require recompiling the model

```
m4 <- add_criterion(m4, c("loo", "waic"))
```

### 12.1.1 Model Weights

We can see that m3 and m4 gave the best LOO-IC and WAIC:

```
loo_compare(m1, m2, m3, m4)
```

	elpd_diff	se_diff
m3	0.0	0.0
m4	-0.6	1.1
m2	-3.5	2.5
m1	-6.0	3.9

So it makes sense that if we're to assign weights, m4 should get most weights. Let's check the following:

```

# Weights based on WAIC
waic_wts <- model_weights(m1, m2, m3, m4, weights = "waic")
# Weights based on Stacking (based on the posterior predictive distribution)
stack_wts <- loo_model_weights(m1, m2, m3, m4)

```

Loading required package: rstan

```
Loading required package: StanHeaders
```

```
rstan version 2.32.5 (Stan version 2.32.2)
```

```
For execution on a local, multicore CPU with excess RAM we recommend calling
options(mc.cores = parallel::detectCores()).
To avoid recompilation of unchanged Stan programs, we recommend calling
rstan_options(auto_write = TRUE)
For within-chain threading using `reduce_sum()` or `map_rect()` Stan functions,
change `threads_per_chain` option:
rstan_options(threads_per_chain = 1)
```

```
Attaching package: 'rstan'
```

```
The following object is masked from 'package:tidy়':
```

```
extract
```

```
# Print out the weights
round(cbind(waic_wts, stack_wts), 3)
```

	waic_wts	stack_wts
m1	0.002	0.100
m2	0.019	0.000
m3	0.635	0.882
m4	0.344	0.018

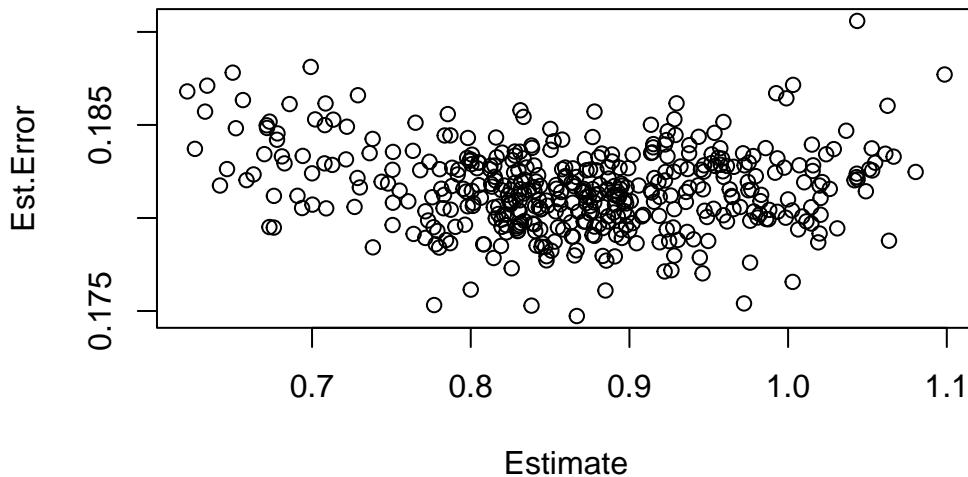
You can see m3 would get the highest weight, but it's only 0.6352557 and thus less than half of the weights when all four models are considered together.

In Bayesian, we want to preserve all the uncertainty in our analyses. Therefore, if we're not certain which models to use and have tried multiple ones, it would make sense to use all of them to get the best information. So unlike what is commonly done in practice where a researcher would test multiple models and present the best model *as if* they intended only to test this model, Bayesian analysts should do the honest thing and use all models. The reward is usually better prediction!

### 12.1.2 Stacking

Stacking is one way to combine the predictions of different models. The technical details can be found in Yao et al. (2018), but you can obtain the predictions using the `pp_average` function:

```
# Prediction from stacking by Yao et al. (2018)
pred_stacking <- pp_average(m1, m2, m3, m4)
# Compare the weights
plot(pred_stacking)
```



### 12.1.3 Prediction example

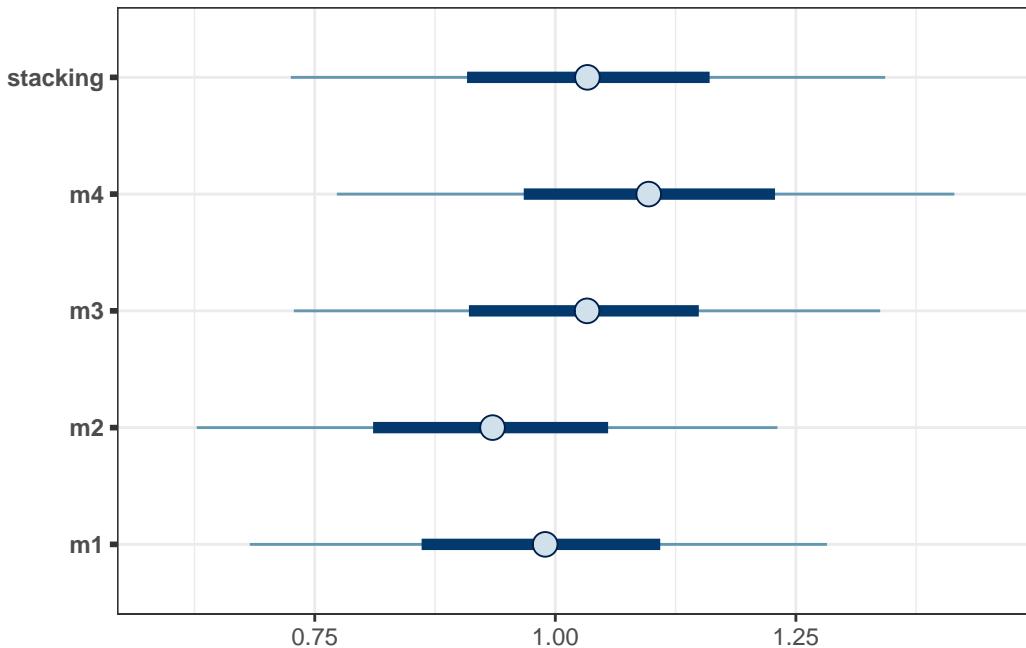
Consider a kid whose mother's IQ is 120 (`mom_iq_c = .2`), mother's age is 40, (`mom_age_c = 2.2`), mother does not have a high school degree, and mother did not work in first three years of child's life (`mom_work = 1`). Then the prediction based on the various models are:

```
newkid <- data.frame(mom_iq_c = .2,
                      mom_age_c = 2.2,
                      mom_work = 1,
                      mom_hs = "no")
# Visualize the prediction by different models
```

```

data.frame(stacking = pp_average(m1, m2, m3, m4, newdata = newkid,
                                 summary = FALSE),
           m4 = posterior_predict(m4, newdata = newkid),
           m3 = posterior_predict(m3, newdata = newkid),
           m2 = posterior_predict(m2, newdata = newkid),
           m1 = posterior_predict(m1, newdata = newkid)) |>
mcmc_intervals()

```



Check out this blog post <https://mc-stan.org/loo/articles/loo2-weights.html> for more information on stacking and Averaging.

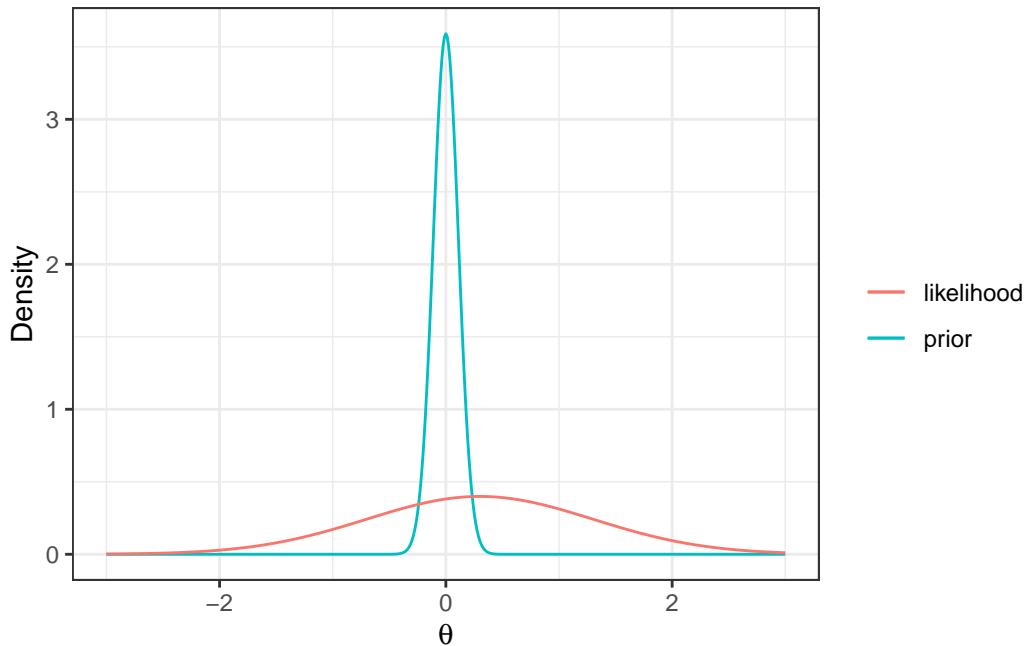
## 12.2 Shrinkage Priors

When the number of parameters to be estimated is large relative to the amount of data available, ordinary least square (in frequentist) and estimation using non-informative or weakly informative priors tend to overfit. For example, fitting a 6th degree polynomial (with 8 parameters) on a data set with only 10 observations will severely overfit the data, making the results not generalizable. One way to avoid overfitting is to perform *regularization*, that is, to shrink some of the parameters to closer to zero. This makes the model fit less well to the existing data, but will be much more generalizable to an independent data set.

### 12.2.1 Number of parameters

In Bayesian analyses, the concept of number of parameters is a little vague. This is because the posterior distribution is a function of both the prior and the data. For non-informative priors, it would make sense to simply count the number of parameters. However, say one put a very strong prior on one of the regression coefficients, which has about 9 times the weights of the information contributed by the data:

```
ggplot(data.frame(th = c(-3, 3)), aes(x = th)) +  
  stat_function(  
    fun = dnorm, args = list(mean = 0, sd = 1 / 9),  
    aes(col = "prior"), n = 501  
  ) +  
  stat_function(  
    fun = dnorm, args = list(mean = 0.3, sd = 1),  
    aes(col = "likelihood"), n = 501  
  ) +  
  labs(y = "Density", x = expression(theta), col = "")
```



Then the posterior for the parameter only uses 1/10 of the information from the data! Therefore, it would make more sense to count this as 0.1 parameter, instead of 1 full parameter.

The concept of regularization is essentially to introduce a stronger prior so that the posterior is less likely to overfit the data, and the resulting model will have lower *effective number*

of parameters, which, when done appropriately, would find a model that is more likely to generalize to external data sets.

In Bayesian methods, regularization can be done by choosing a prior on the coefficient that has a sharp peak at 0, but also has a heavy tail. One such prior is what is called the *horseshoe* prior. The discussion here is based on the blog post by Michael Betancourt: [https://betanalpha.github.io/assets/case\\_studies/bayes\\_sparse\\_regression.html](https://betanalpha.github.io/assets/case_studies/bayes_sparse_regression.html)

It should first be pointed out that these priors were based on the assumption that the predictors and the outcome has been scaled to have a standard deviation of one. So we will do this here:

```
# For variable selection, scale the predictor and outcome to have unit variance
kidiq_std <- scale(kidiq)
head(kidiq_std)
```

	kid_score	mom_hs	mom_iq	mom_work	mom_age
[1,]	-1.06793237	0.521631	1.4078352	0.93422435	1.5602285
[2,]	0.54886757	0.521631	-0.7092079	0.93422435	0.8197811
[3,]	-0.08805362	0.521631	1.0295443	0.93422435	1.5602285
[4,]	-0.18604150	0.521631	-0.0366907	0.08776638	0.8197811
[5,]	1.38176451	0.521631	-0.4836193	0.93422435	1.5602285
[6,]	0.54886757	-1.912647	0.5267892	-1.60514956	-1.7717849

## 12.2.2 Sparsity-Inducing Priors

The *horseshoe* prior (Carvalho et al., 2010) is a type of hierarchical prior for regression models by introducing a global scale,  $\tau$ , and local scale,  $\lambda_m$ , parameters on the priors for the regression coefficients (Piironen & Vehtari, 2017). Specifically, with  $p$  predictors,

$$\begin{aligned} Y_i &\sim N(\mu_i, \sigma^2) \\ \mu_i &= \beta_0 + \sum_{m=1}^p \beta_m X_m \\ \beta_0 &\sim N(0, 1) \\ \beta_m &\sim N(0, \tau \lambda_m) \\ \lambda_m &\sim \text{Cauchy}^+(0, 1) \\ \tau &\sim \text{Cauchy}^+(0, \tau_0) \end{aligned}$$

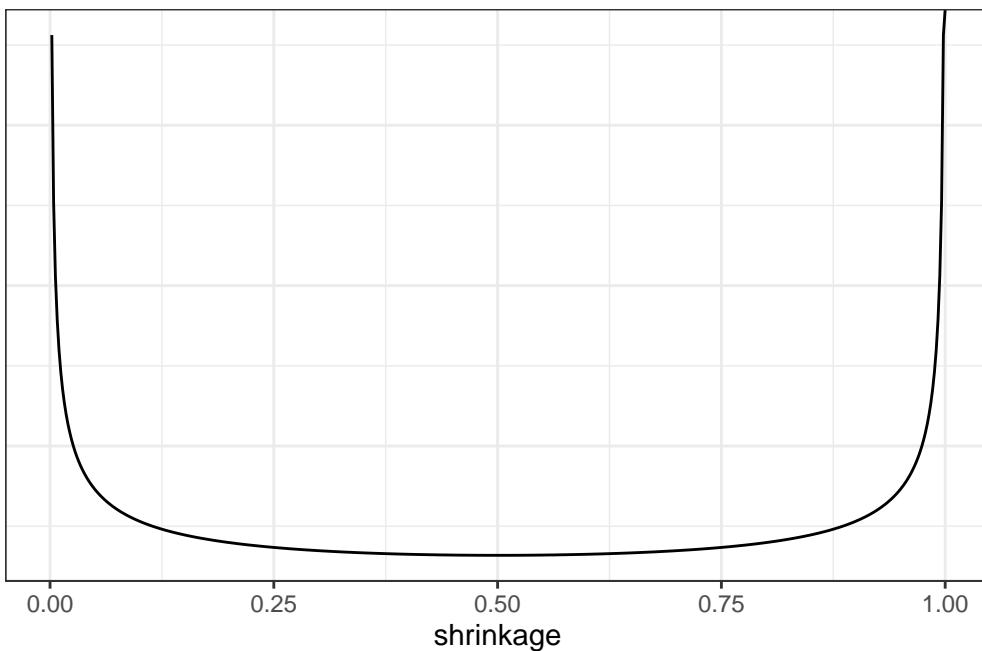
The local scale,  $\lambda_m$ , can flexibly shrink the coefficient to close to zero. Below is the implication of the prior on the shrinkage of  $\beta$ :

```

ggplot(data.frame(shrinkage = c(0, 1)), aes(x = shrinkage)) +
  stat_function(fun = function(x) {
    dcauchy(sqrt(1 / x - 1)) * 2 * (1 / x - 1)^(-1 / 2) * x^(-2) / 2
  }, n = 501) +
  theme(
    axis.text.y = element_blank(),
    axis.ticks.y = element_blank()
  ) +
  labs(y = "")

```

Warning: Removed 1 row containing missing values (`geom\_function()`).



The U-shape here means that, for coefficients that are weakly supported by the data, the horseshoe will shrink it to very close to zero, whereas for coefficients that are more strongly supported by the data, the horseshoe will not shrink it much.

The red curve in the following is one example for the resulting prior distribution on  $\beta$ :

```

dhs <- Vectorize(
  function(y, df = 1) {
    ff <- function(lam) dnorm(y, 0, sd = lam) * dt(lam, df) * 2
    if (y != 0) {

```

```

            integrate(ff, lower = 0, upper = Inf)$value
} else {
  Inf
}
}
)
ggplot(data.frame(x = c(-6, 6)), aes(x = x)) +
  stat_function(
    fun = dhs, args = list(df = 3), n = 501,
    aes(col = "HS"), linetype = 1
) +
  stat_function(
    fun = dnorm, n = 501,
    aes(col = "norm"), linetype = 2
) +
  scale_color_manual("", 
    values = c("red", "black"),
    labels = c("horseshoe(3)", "N(0, 1)")
) +
  xlab("y") +
  ylab("density") +
  ylim(0, 0.75)

```

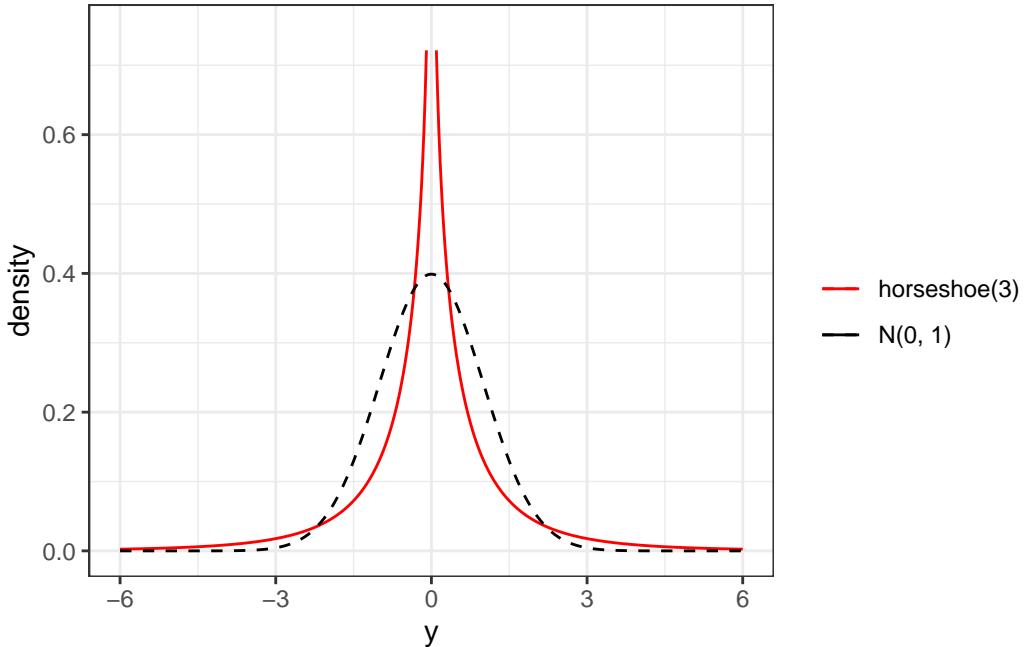


Figure 12.1: Density for the regularized horseshoe prior with 3 degrees of freedom

Such a prior has more density at 0, but also more density for extreme values, as compared to a normal distribution. Thus, for coefficients with very weak evidence, the regularizing prior will shrink it to zero, whereas for coefficients with strong evidence, the shrinkage will be very small. This is called a horseshoe prior. In `brms`, one can specify it with `horseshoe()`, which is a stabilized version of the original horseshoe prior (Carvalho et al., 2010).

### 12.2.3 Regularized Horseshoe/Hierarchical Shrinkage

The regularized horseshoe (<https://projecteuclid.org/euclid.ejs/1513306866>) prior is

$$\begin{aligned}\beta_m &\sim N(0, \tau \tilde{\lambda}_m) \\ \tilde{\lambda}_m &= \frac{c \lambda_m}{\sqrt{c^2 + \tau^2 \lambda_m^2}} \\ \lambda_m &\sim \text{Cauchy}^+(0, 1) \\ c^2 &\sim \text{Inv-Gamma}(\nu/2, nu/2s^2) \\ \tau &\sim \text{Cauchy}^+(0, \tau_0)\end{aligned}$$

The additional parameters are chosen in the code below. First, fit a model without shrinkage:

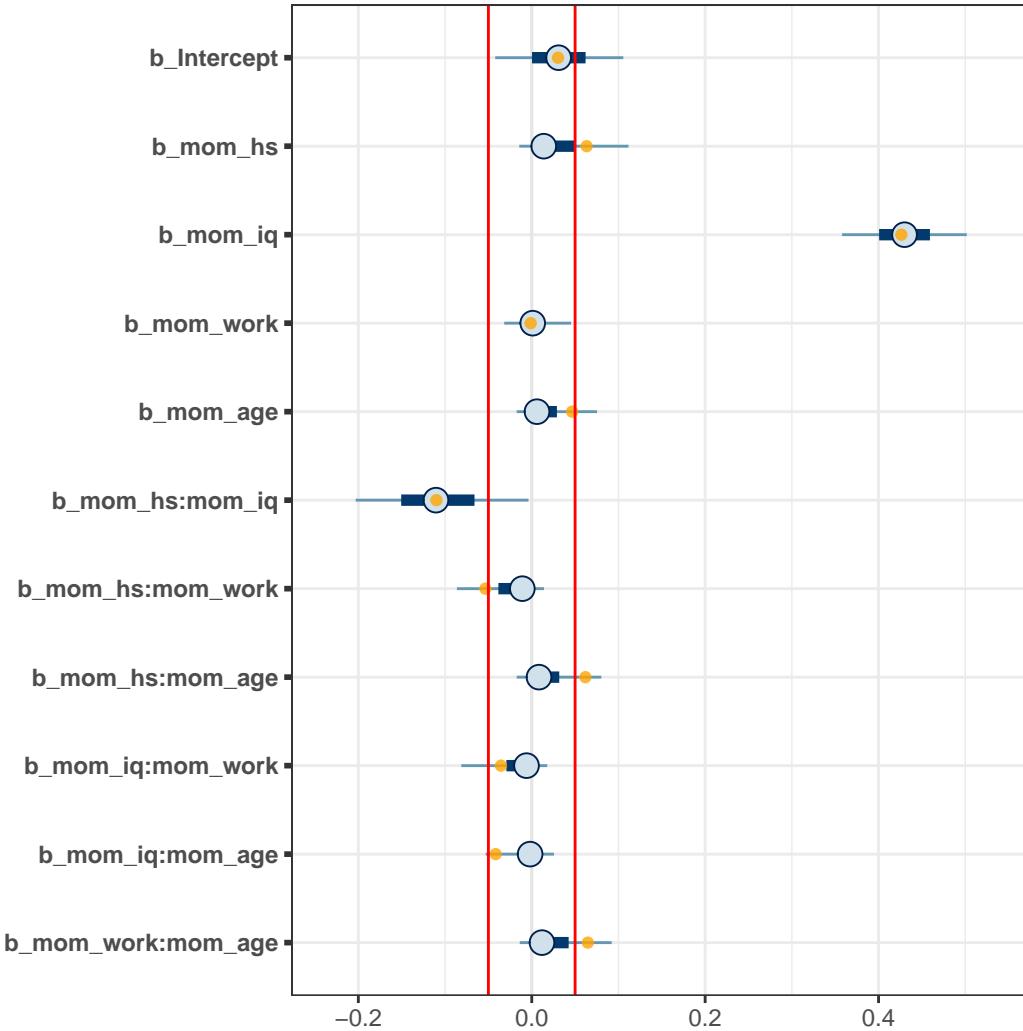
```
# A model with all main and interaction effects
m5 <- brm(kid_score ~ .)^2,
  data = kidiq_std,
  prior = c(
    prior(normal(0, 1), class = "Intercept"),
    prior(normal(0, 1), class = "b"),
    prior(student_t(4, 0, 1), class = "sigma")
  ),
  seed = 2217,
  file = "07b_m5"
)
```

```
# A model with all main and interaction effects, and regularization
m_hs <- brm(kid_score ~ .)^2,
  data = kidiq_std,
  prior = c(
    prior(normal(0, 1), class = "Intercept"),
    # Prior guess of 20% of the terms are non-zero
    prior(horseshoe(par_ratio = 2 / 8), class = "b"),
    prior(student_t(4, 0, 1), class = "sigma")
  ),
  # Need higher adapt_delta
  control = list(adapt_delta = .995, max_treedepth = 12),
  seed = 2217,
  file = "07b_m_hs"
)
```

We can plot the coefficients:

```
mcmc_plot(m_hs, variable = "b",
           regex = TRUE) +
  # Show the shrinkage as orange, transparent dots
  geom_point(
    data = posterior_summary(m5) |>
      as_tibble(rownames = "parameter") |>
      filter(parameter != "lp__"),
    aes(x = Estimate, y = parameter), alpha = 0.8,
    col = "orange"
  ) +
  geom_vline(xintercept = c(-.05, .05), col = "red")
```

Warning: Removed 3 rows containing missing values (`geom\_point()`).



An arbitrary cutoff is to select only coefficients with posterior means larger than .05, in which case only `mom_iq` and `mom_hs` and their interaction were supported by the data.

You can also double check that the regularized version has better LOO-IC:

```
loo(m5, m_hs)
```

Output of model 'm5':

Computed from 4000 by 434 log-likelihood matrix.

	Estimate	SE
elpd_loo	-567.3	14.4

```
p_loo      12.8  1.4
looic     1134.6 28.8
-----
MCSE of elpd_loo is 0.1.
MCSE and ESS estimates assume MCMC draws (r_eff in [0.6, 2.0]).
```

All Pareto k estimates are good (k < 0.7).  
See help('pareto-k-diagnostic') for details.

Output of model 'm\_hs':

Computed from 4000 by 434 log-likelihood matrix.

	Estimate	SE
elpd_loo	-565.2	14.4
p_loo	8.5	0.9
looic	1130.5	28.9

-----

```
MCSE of elpd_loo is 0.1.
MCSE and ESS estimates assume MCMC draws (r_eff in [0.5, 1.3]).
```

All Pareto k estimates are good (k < 0.7).  
See help('pareto-k-diagnostic') for details.

Model comparisons:

	elpd_diff	se_diff
m_hs	0.0	0.0
m5	-2.1	2.3

And also that the effective number of parameters was smaller in m\_hs.

## 12.3 Variable Selection

One way to identify variables that are relevant to predict a certain outcome is to use the projection-based method, as discussed in <https://mc-stan.org/projpred/articles/projpred.html> and in Piironen et al. (2020).

Building from the full model with shrinkage priors, we first do a trial run to identify the importance of various variables in terms of their importance for prediction:

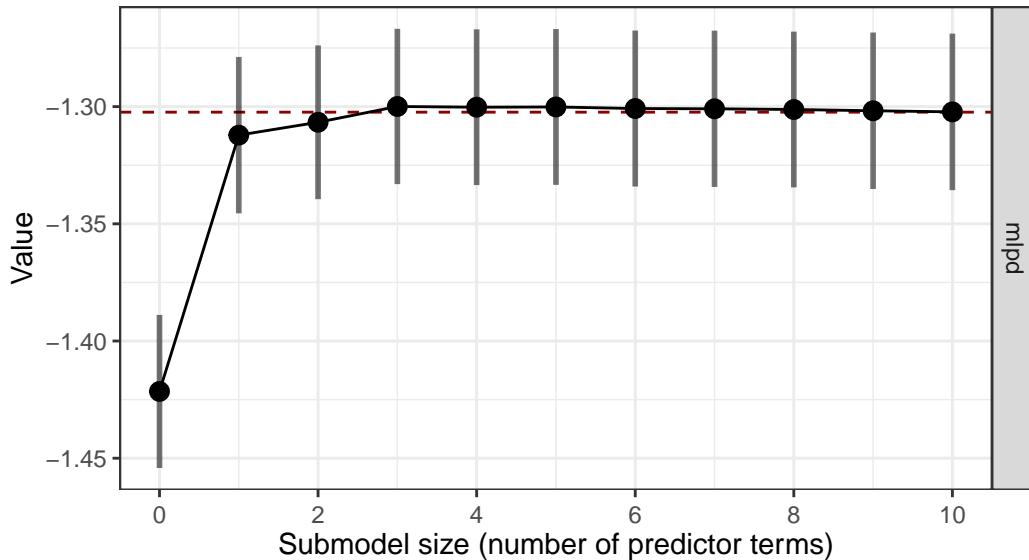
```
# Get reference model
refm_obj <- get_refmodel(m_hs)
# Preliminary run to find `nterms_max`
cvvs_fast <- cv_varsel(
  refm_obj,
  validate_search = FALSE
)

-----
Running the search ...
10% of terms selected
20% of terms selected
30% of terms selected
40% of terms selected
50% of terms selected
60% of terms selected
70% of terms selected
80% of terms selected
90% of terms selected
100% of terms selected
-----
-----
Running the performance evaluation with `refit_prj = TRUE` ...
-----
```

```
# mlpd = mean log predictive density
plot(cvvs_fast, stats = "mlpd", ranking_nterms_max = NA)
```

## Predictive performance

Vertical bars indicate 68.3% normal–approximation intervals



From the plot, we find when the predictive performance starts to level off when we keep adding more terms to the model. In this case, it seems to be 4. Given that this is a trial run, we'll set `nterms_max` to 5, which is slightly higher.

We then set `validate_search = FALSE` for a final run. For computational feasibility, we'll use 10-fold validation.

```
# With 10-fold cross-validation
cvvs <- cv_varsel(
  refm_obj,
  validate_search = TRUE,
  cv_method = "kfold",
  K = 10,
  nterms_max = 5
)
```

```
-----
Running the search using the full dataset ...
20% of terms selected
40% of terms selected
60% of terms selected
80% of terms selected
100% of terms selected
```

-----

-----

Refitting the reference model K = 10 times (using the fold-wise training data) ...  
Running MCMC with 4 sequential chains...

Chain 1 finished in 2.7 seconds.  
Chain 2 finished in 2.0 seconds.  
Chain 3 finished in 2.8 seconds.  
Chain 4 finished in 2.5 seconds.

All 4 chains finished successfully.  
Mean chain execution time: 2.5 seconds.  
Total execution time: 10.3 seconds.

Running MCMC with 4 sequential chains...

Chain 1 finished in 2.6 seconds.  
Chain 2 finished in 2.1 seconds.  
Chain 3 finished in 2.5 seconds.  
Chain 4 finished in 2.1 seconds.

All 4 chains finished successfully.  
Mean chain execution time: 2.3 seconds.  
Total execution time: 9.5 seconds.

Running MCMC with 4 sequential chains...

Chain 1 finished in 2.4 seconds.  
Chain 2 finished in 2.1 seconds.  
Chain 3 finished in 2.1 seconds.  
Chain 4 finished in 2.1 seconds.

All 4 chains finished successfully.  
Mean chain execution time: 2.2 seconds.  
Total execution time: 8.9 seconds.

Running MCMC with 4 sequential chains...

Chain 1 finished in 2.5 seconds.  
Chain 2 finished in 2.0 seconds.  
Chain 3 finished in 2.2 seconds.  
Chain 4 finished in 2.2 seconds.

```
All 4 chains finished successfully.  
Mean chain execution time: 2.2 seconds.  
Total execution time: 9.2 seconds.
```

```
Running MCMC with 4 sequential chains...
```

```
Chain 1 finished in 2.2 seconds.  
Chain 2 finished in 1.8 seconds.  
Chain 3 finished in 1.9 seconds.  
Chain 4 finished in 1.8 seconds.
```

```
All 4 chains finished successfully.  
Mean chain execution time: 1.9 seconds.  
Total execution time: 8.1 seconds.
```

```
Running MCMC with 4 sequential chains...
```

```
Chain 1 finished in 1.5 seconds.  
Chain 2 finished in 2.2 seconds.  
Chain 3 finished in 1.6 seconds.  
Chain 4 finished in 1.7 seconds.
```

```
All 4 chains finished successfully.  
Mean chain execution time: 1.8 seconds.  
Total execution time: 7.5 seconds.
```

```
Running MCMC with 4 sequential chains...
```

```
Chain 1 finished in 1.8 seconds.  
Chain 2 finished in 1.9 seconds.  
Chain 3 finished in 1.4 seconds.  
Chain 4 finished in 2.5 seconds.
```

```
All 4 chains finished successfully.  
Mean chain execution time: 1.9 seconds.  
Total execution time: 8.1 seconds.
```

```
Running MCMC with 4 sequential chains...
```

```
Chain 1 finished in 2.0 seconds.  
Chain 2 finished in 1.9 seconds.  
Chain 3 finished in 2.1 seconds.  
Chain 4 finished in 1.8 seconds.
```

```
All 4 chains finished successfully.  
Mean chain execution time: 1.9 seconds.  
Total execution time: 8.0 seconds.
```

```
Running MCMC with 4 sequential chains...
```

```
Chain 1 finished in 2.2 seconds.  
Chain 2 finished in 2.0 seconds.  
Chain 3 finished in 2.3 seconds.  
Chain 4 finished in 1.8 seconds.
```

```
All 4 chains finished successfully.  
Mean chain execution time: 2.1 seconds.  
Total execution time: 8.8 seconds.
```

```
Running MCMC with 4 sequential chains...
```

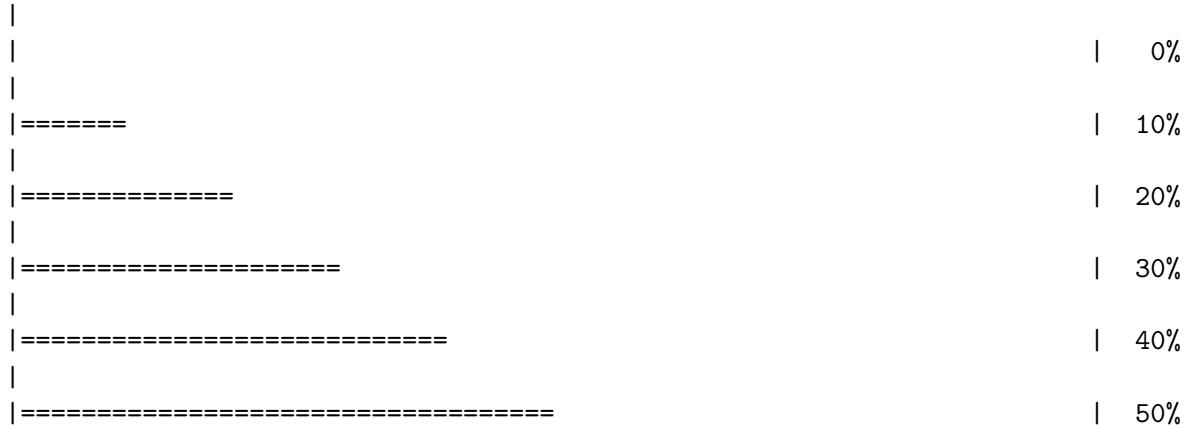
```
Chain 1 finished in 1.6 seconds.  
Chain 2 finished in 1.9 seconds.  
Chain 3 finished in 1.6 seconds.  
Chain 4 finished in 1.8 seconds.
```

```
All 4 chains finished successfully.  
Mean chain execution time: 1.7 seconds.  
Total execution time: 7.3 seconds.
```

```
-----
```

```
-----
```

```
Running the search and the performance evaluation with `refit_prj = TRUE` for each of the K =
```

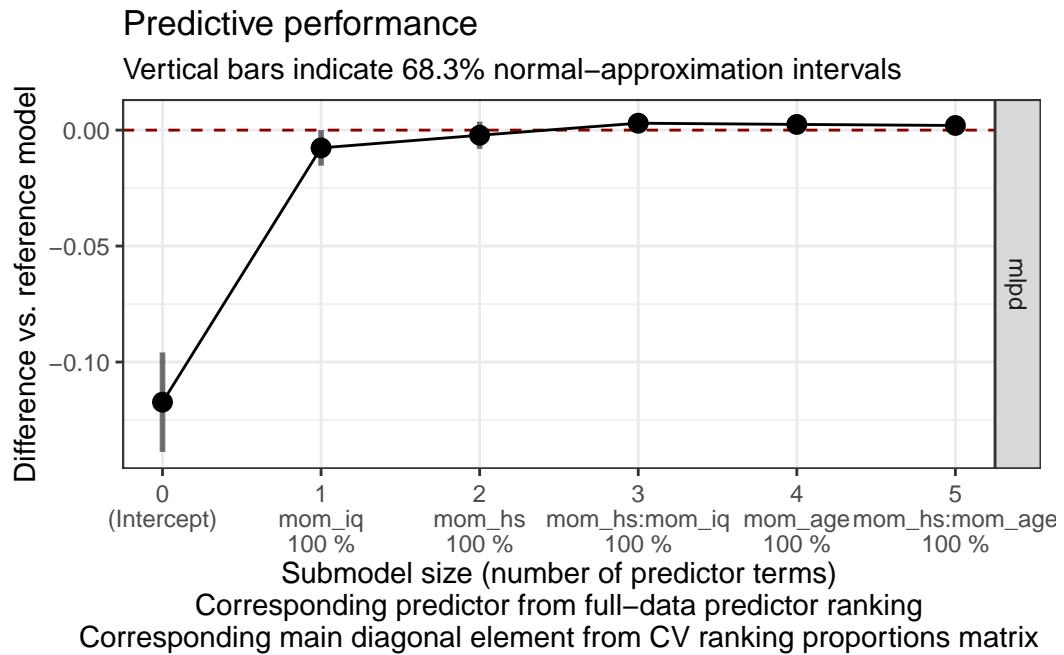


```

|=====
|===== | 60%
|===== | 70%
|===== | 80%
|===== | 90%
|===== | 100%
-----

```

```
plot(cvvs, stats = "mlpd", deltas = TRUE)
```



```
# model size suggested by the program
suggest_size(cvvs, stat = "mlpd")
```

```
[1] 1
```

```
# Same with RMSE
suggest_size(cvvs, stat = "rmse")
```

```
[1] 1
```

```
# Summary of the variable selection results
summary(cvvs,
  stats = "mlpd", type = c("mean", "lower", "upper"),
  deltas = TRUE
)
```

Family: gaussian

Link function: identity

Formula: kid\_score ~ mom\_hs + mom\_iq + mom\_work + mom\_age + mom\_hs:mom\_iq +
mom\_hs:mom\_work + mom\_hs:mom\_age + mom\_iq:mom\_work + mom\_iq:mom\_age +
mom\_work:mom\_age

Observations: 434

Projection method: traditional

CV method: K-fold CV with K = 10 and search included (i.e., fold-wise searches)

Search method: forward

Maximum submodel size for the search: 5

Number of projected draws in the search: 20 (from clustered projection)

Number of projected draws in the performance evaluation: 400

Argument `refit\_prj`: TRUE

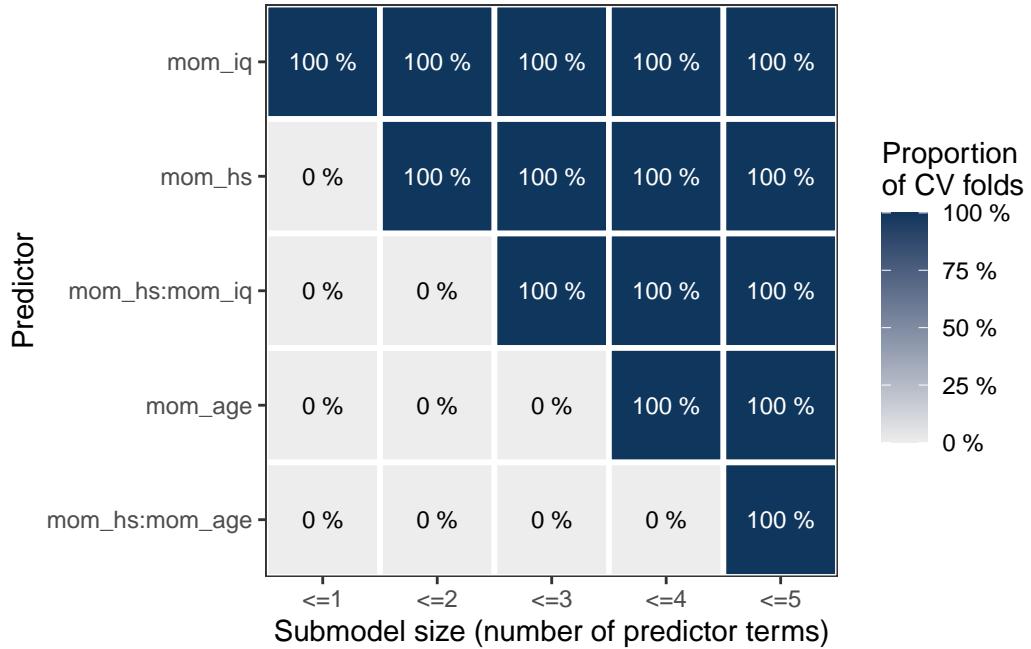
Submodel performance evaluation summary with `deltas = TRUE` and `cumulate = FALSE`:

	size ranking_fulldata	cv_proportions_diag	mlpd	mlpd.lower	mlpd.upper
0	(Intercept)		NA	-0.1173	-1.4e-01 -9.6e-02
1	mom_iq		1	-0.0077	-1.5e-02 1.7e-05
2	mom_hs		1	-0.0022	-8.1e-03 3.6e-03
3	mom_hs:mom_iq		1	0.0030	6.4e-05 5.9e-03
4	mom_age		1	0.0025	-3.8e-04 5.3e-03
5	mom_hs:mom_age		1	0.0020	-4.0e-04 4.4e-03

Reference model performance evaluation summary with `deltas = TRUE`:

```
mlpd mlpd.lower mlpd.upper
0 0 0
```

```
# Predictor ranking(s)
rk <- ranking(cvvs)
plot(cv_proportions(rk, cumulate = TRUE))
```



Here it suggests to either include only `mom_iq` and `mom_hs`, or to also include their interactions.

### 12.3.1 Projection-Based Method

The projection-based method will obtain the posterior distributions based on a projection from the full model on the simplified model. In other words, we're asking the question:

If we want a model with only `mom_iq` and `mom_hs`, what coefficients should be obtained so that the resulting prediction accuracy is as close to the full model as possible?

Note that the coefficients will be different from if you were to directly estimate the model using the two predictors (i.e., `m2`). In this case, simulation results showed that the projection-based method will yield a model with better predictive performance.

```
# Fit m2 with the standardized data
m2_std <- brm(kid_score ~ mom_hs + mom_iq,
  data = kidiq_std,
  prior = c(
    prior(normal(0, 1), class = "Intercept"),
    prior(normal(0, 1), class = "b"),
    prior(student_t(4, 0, 1), class = "sigma")
```

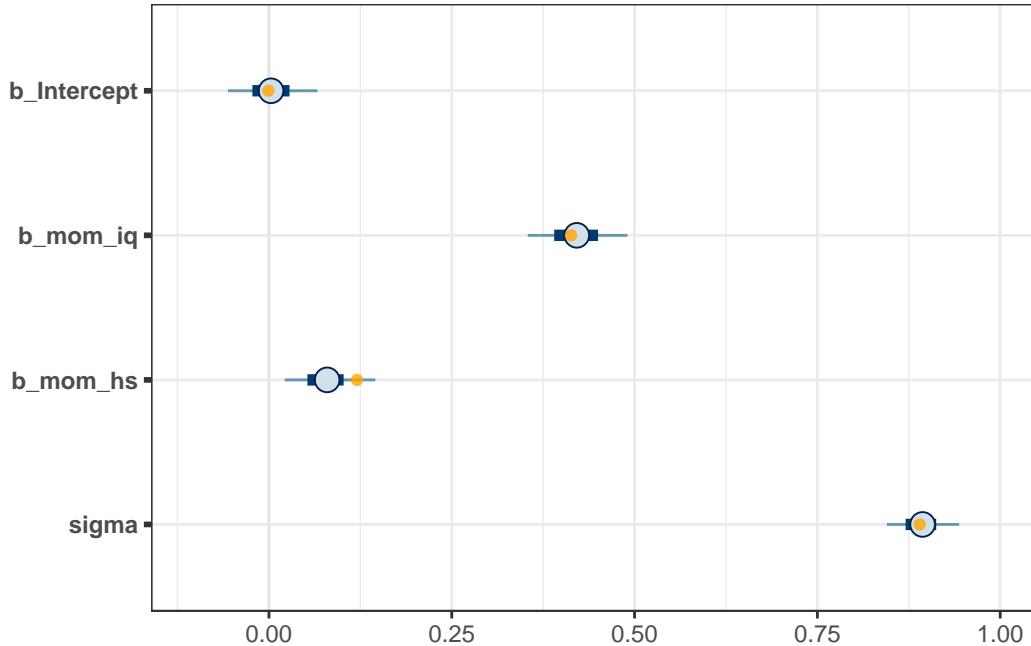
```

),
seed = 2302,
file = "07b_m2_std"
)

# Visualise the projected three most relevant variables
prj <- project(refm_obj,
  predictor_terms = c("mom_iq", "mom_hs"),
  verbose = FALSE
)
mcmc_intervals(as.matrix(prj)) +
  # Show the non-projection version as black, transparent dots
  geom_point(
    data =
      posterior_summary(m2_std) |>
        as_tibble(rownames = "parameter"),
    aes(x = Estimate, y = parameter), alpha = 0.8,
    col = "orange"
  )

```

Warning: Removed 3 rows containing missing values (`geom\_point()`).



**Part VII**

**Week 9**

# 13 Causal Inference

Causal inference is an important topic in statistics, but it also has a complicated history with statistics. Early-day statisticians saw causality as a taboo such that researchers were discouraged from drawing any causal conclusions in nonexperimental research. In the past few decades, much progress has been made on causal inference in epidemiology, computer science, and statistics, with several frameworks proposed and many tools developed for experimental and nonexperimental data.<sup>1</sup> This note will introduce the causal diagram framework for encoding causal assumptions, which can be used to guide analysis for drawing causal conclusions. The goal is to give you some basic ideas so that you can learn more from other sources, such as the book *Causality* by Pearl and the book *Causal Inference for Statistics, Social, and Biomedical Sciences: An Introduction* by Imbens and Rubin.

💡 Two prominent frameworks for causal inference are

1. Structural causal model, which uses tools such as DAGs and  $d$ -separation, is thoroughly described in the book *Causality* by Pearl, and
2. Potential outcome model, which is thoroughly described in the book *Causal Inference for Statistics, Social, and Biomedical Sciences: An Introduction* by Imbens and Rubin.

In this note, we will draw on both frameworks.

## ❗ What is Causal Inference?

Causal inference is the process of determining the causal effect of one variable on another, based on data and causal assumptions. Two common ways of interpreting a causal effect are (a) *intervention*: how  $Y$  will change if  $X$  is manipulated to be  $x$ ; (b) *counterfactual*: what would  $Y$  be if  $X$  were set to  $x$ .

<sup>1</sup>See this paper: <https://doi.org/10.1177/1745691620921521> for an argument of how the taboo against causal inference has impeded the progress of psychology research.

## 13.1 Potential Outcomes

Consider a binary treatment variable. For each individual, there are two *potential outcomes*: one if they receive the treatment ( $T = 1$ ) and one if they do not ( $T = 0$ ). Here is a hypothetical data set of a new drug for improving statistics knowledge, with the treatment condition receiving the drug and the control condition receiving a placebo:

```
po_dat <- data.frame(
  person = c(1, 2, 3, 4, 5, 6, 7, 8),
  attitude = c(4, 7, 3, 9, 5, 6, 8, 2),
  treat = c(75, 80, 70, 90, 85, 82, 95, 78),
  control = c(70, 88, 75, 92, 82, 85, 90, 78)
)
knitr::kable(po_dat,
  col.names = c("Person", "Math Attitude", "Y (if T = 1)", "Y (if T = 0)"))
```

Person	Math Attitude	Y (if T = 1)	Y (if T = 0)
1	4	75	70
2	7	80	88
3	3	70	75
4	9	90	92
5	5	85	82
6	6	82	85
7	8	95	90
8	2	78	78

If one could observe the two potential outcomes for each person (which is impossible in the real world), one could compute the treatment effect for each person:

```
(te <- po_dat$treat - po_dat$control)
```

```
[1]  5 -8 -5 -2  3 -3  5  0
```

and the **average treatment effect (ATE)** (here we just compute it on the sample, so the sample ATE):

```
(ate <- mean(te))
```

```
[1] -0.625
```

However, in practice, we can only observe one of the two potential outcomes. For example, if persons 2, 4, 6, 7 are in the treatment condition, we have

```
po_dat$t <- ifelse(po_dat$person %in% c(2, 4, 6, 7), 1, 0)
po_dat$y <- ifelse(po_dat$t, po_dat$treat, po_dat$control)
po_dat[c("t", "y")] |>
  knitr::kable(col.names = c("Treatment", "Observed Y"))
```

Treatment	Observed Y
0	70
1	80
0	75
1	90
0	82
1	82
1	95
0	78

and the naive comparison of those in the treatment condition and those in the control condition will have a mean difference of

```
mean(po_dat$y[po_dat$t == 1]) - mean(po_dat$y[po_dat$t == 0])
```

```
[1] 10.5
```

which gives a misleading estimate of the ATE.

## 13.2 Directed Acyclic Graph (DAG)

DAG is a tool for encoding causal assumptions. It contains nodes and paths. A node is usually a variable that can be measured in the data or unmeasured. Generally, paths in a DAG are directional (as implied by *directed*), which indicates the directions of causal relations. Acyclic means the causal chain does not close in a loop.

We will use an example described in McElreath (2020), chapter 5, about data from 50 U.S. states from the 2009 American Community Survey (ACS), which we have already seen in Chapter 9 and Figure 9.1.

```
waffle_divorce <- read_delim( # read delimited files
  "https://raw.githubusercontent.com/rmcelreath/rethinking/master/data/WaffleDivorce.csv",
  delim = ";"
)

Rows: 50 Columns: 13
-- Column specification -----
Delimiter: ;"
chr (2): Location, Loc
dbl (11): Population, MedianAgeMarriage, Marriage, Marriage SE, Divorce, Div...
i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
# Rescale Marriage and Divorce by dividing by 10
waffle_divorce$Marriage <- waffle_divorce$Marriage / 10
waffle_divorce$Divorce <- waffle_divorce$Divorce / 10
waffle_divorce$MedianAgeMarriage <- waffle_divorce$MedianAgeMarriage / 10
# See data description at https://rdrr.io/github/rmcelreath/rethinking/man/WaffleDivorce.html
```

The outcome of interest is the divorce rate. Remember the plot shows how marriage rate is related to divorce rate at the state level. It looks like marriage rate can predict divorce rate. A causal interpretation would be a stronger assertion, such that we expect a higher divorce rate if policymakers encourage more people to get married. In order to make a causal claim, we need to remove potential confounders.

A potential confounder is when people get married. Suppose people are forced to get married later in life. In that case, fewer people in the population will be married, and people may have less time, opportunity, and motivation to get divorced (as it may be harder to find a new partner). Therefore, we can use the following DAG:

```
dag1 <- dagitty("dag{ A -> D; A -> M; M -> D }")
coordinates(dag1) <- list(x = c(M = 0, A = 1, D = 2),
                           y = c(M = 0, A = 1, D = 0))
# Plot
ggdag(dag1) + theme_dag()
```

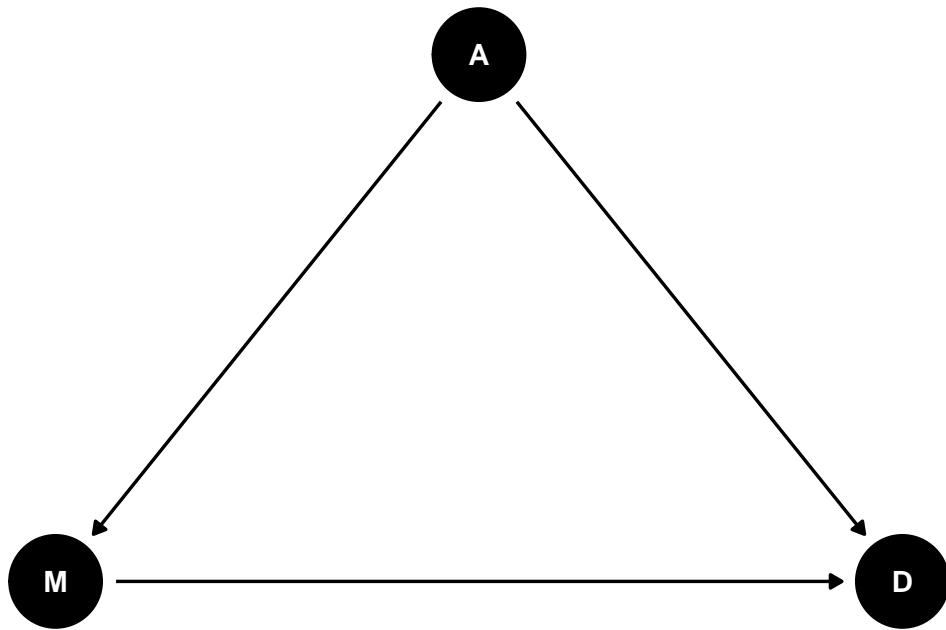


Figure 13.1: DAG for the relationship between marriage rate (M) and divorce rate (D) example, with median age of marriage (A) as a potential confounder.

We can look at how the median age people get married in different states relates to the divorce rate:

```

ggplot(waffle_divorce,
       aes(x = MedianAgeMarriage, y = Divorce)) +
  geom_point() +
  geom_smooth() +
  labs(x = "Median age marriage (10 years)",
       y = "Divorce rate (per 10 adults)") +
  ggrepel::geom_text_repel(aes(label = Loc))

```

```
`geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```

```
Warning: ggrepel: 9 unlabeled data points (too many overlaps). Consider
increasing max.overlaps
```

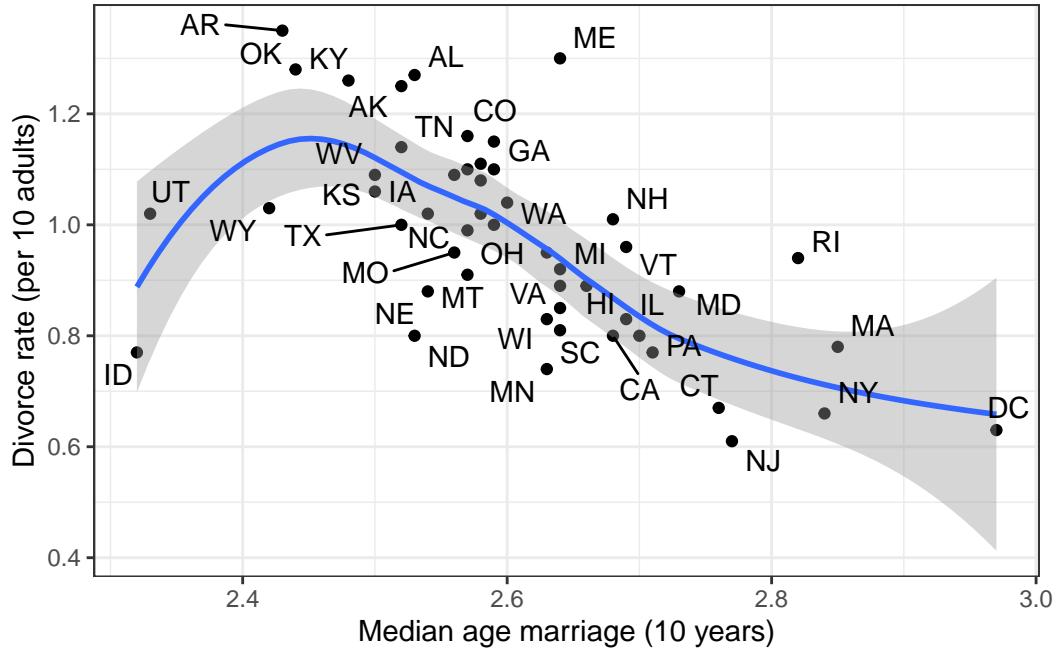


Figure 13.2: Scatter plot of divorce rate (y) vs median age of marriage (x).

### 13.2.1 Assumptions in a DAG

In a graphical model, there are usually two kinds of assumptions conveyed. Perhaps counterintuitively, the weaker assumptions are the ones you can see, whereas the stronger assumptions are ones you **do not see**.

For example, a weak assumption is:

- Marriage age *may* directly influence the number of people who are married

On the other hand, examples of a strong assumption are:

- Marriage rate does not directly influence the age people get married
- there is no other variable in the causal pathway  $M \rightarrow D$

The last assumption will not hold if there is a common cause of M and D other than A.

### 13.2.2 Basic Types of Junctions

- Fork:  $A \leftarrow B \rightarrow C$ 
  - A and C are correlated due to a common cause B

- Chain/Pipe:  $A \rightarrow B \rightarrow C$ 
  - B is on the causal pathway of A to C
- Collider:  $A \rightarrow B \leftarrow C$ 
  - B is a *descendant* of both A and C

Going back to our example, there is a fork relation:  $M \rightarrow A \rightarrow D$ . In this case, A is a confounder when estimating the causal relation between M and D. In this case, the causal effect of M to D can be obtained by adjusting for A, which means looking at the subsets of data where A is constant. In regression, adjustment is obtained by including both M and A as predictors of D.

### 13.3 The Back-Door Criterion: Estimating Causal Effect in Nonexperimental Data

Randomization is one—but not the only—way to rule out confounding variables. Much progress has been made in clarifying that causal effects **can** be estimated in nonexperimental data. After all, researchers have not randomly assigned individuals to smoke  $x$  cigarettes per day, but we are confident that smoking causes cancer; we never manipulated the moon’s gravitational force, but we are pretty sure that the moon is a cause of high and low tides.

In a DAG, the back-door criterion can be used to identify variables that we should adjust for to obtain a causal effect. The set of variables to be adjusted should (a) blocks every path between X and Y that contains an arrow entering X and (b) does not contain variables that are descendant of X.

```
dag4 <- dagitty(
  "dag{
    X -> Y; W1 -> X; U -> W2; W2 -> X; W1 -> Y; U -> Y
  }"
)
latents(dag4) <- "U"
coordinates(dag4) <- list(x = c(X = 0, W1 = 0.66, U = 1.32, W2 = 0.66, Y = 2),
                           y = c(X = 0, W1 = 1, U = 1, W2 = 0.5, Y = 0))
ggdag(dag4) + theme_dag()
```

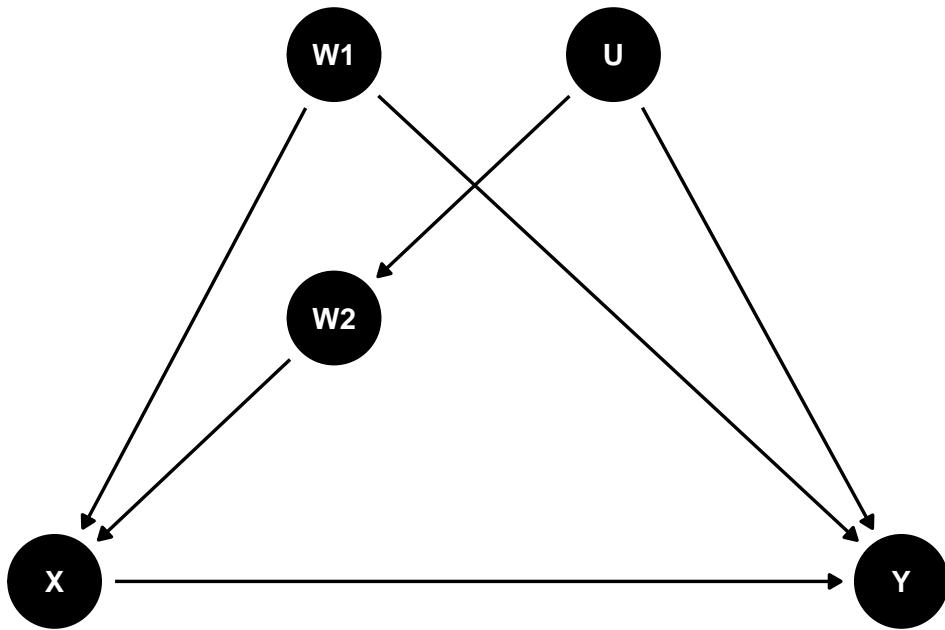


Figure 13.3: DAG for a hypothetical model with two measured and one unobserved confounder.

We can use a function in the `daggity` package to identify the set of variables that satisfy the back-door criterion. In this case, we say that  $X$  and  $Y$  are  $d$ -separated by this set of adjusted variables:

```
adjustmentSets(dag4, exposure = "X", outcome = "Y",
                effect = "direct")
```

```
{ W1, W2 }
```

## 13.4 Using Multiple Regression Model

$$\begin{aligned}
 D_i &\sim N(\mu_i, \sigma) \\
 \mu_i &= \beta_0 + \beta_1 A_i + \beta_2 M_i \\
 \beta_0 &\sim N(0, 1) \\
 \beta_1 &\sim N(0, 1) \\
 \beta_2 &\sim N(0, 1)
 \end{aligned}$$

Assuming a correctly specified DAG, all all assumptions of a linear model are met,  $\beta_2$  is the causal effect of  $M$  on  $D$ . Here is the `brms` code:

```

m1 <- brm(Divorce ~ MedianAgeMarriage + Marriage,
            data = waffle_divorce,
            prior = prior(std_normal(), class = "b") +
              prior(normal(0, 5), class = "Intercept") +
              prior(student_t(4, 0, 3), class = "sigma"),
            seed = 941,
            iter = 4000,
            file = "08_m1"
)

```

### 13.4.1 Table

```

msummary(m1,
          estimate = "{estimate} [{conf.low}, {conf.high}]",
          statistic = NULL, fmt = 2
)

```

Warning:

`modelsummary` uses the `performance` package to extract goodness-of-fit statistics from models of this class. You can specify the statistics you wish to compute by supplying a `metrics` argument to `modelsummary`, which will then push it forward to `performance`. Acceptable values are: "all", "common", "none", or a character vector of metrics names. For example: `modelsummary(mod, metrics = c("RMSE", "R2"))` Note that some metrics are computationally expensive. See `?performance::performance` for details.

This warning appears once per session.

As shown in the table, when holding constant the median marriage age of states, marriage rate has only a small effect on divorce rate.

### 13.4.2 Posterior predictive checks

```

pp_check(m1, ndraws = 100) # density
pp_check(m1, type = "intervals", x = "Marriage") +
  labs(x = "Marriage", y = "Divorce")

```

Using all posterior draws for ppc type 'intervals' by default.

Table 13.3: Model results for the divorce rate example.

	(1)
b_Intercept	3.49 [1.98, 5.02]
b_MedianAgeMarriage	-0.93 [-1.43, -0.45]
b_Marriage	-0.04 [-0.20, 0.12]
sigma	0.15 [0.12, 0.19]
Num.Obs.	50
R2	0.349
R2 Adj.	0.282
ELPD	21.0
ELPD s.e.	6.4
LOOIC	-42.0
LOOIC s.e.	12.9
WAIC	-42.2
RMSE	0.14

```
pp_check(m1, type = "intervals", x = "MedianAgeMarriage") +
  labs(x = "MedianAgeMarriage", y = "Divorce")
```

Using all posterior draws for ppc type 'intervals' by default.

The model does not fit every state (e.g., UT, ME, ID).

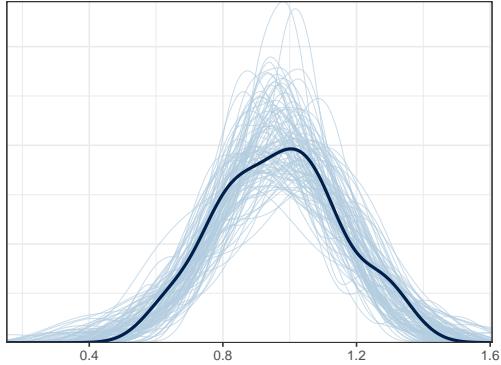
## 13.5 Predicting an Intervention

If the causal assumption in our DAG is reasonable, we have obtained an estimate of the causal effect of marriage rate on divorce rate at the state level. This allows us to answer questions like

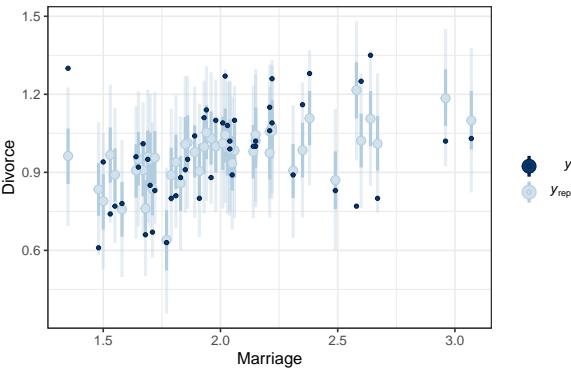
What would happen to the divorce rate if we encouraged more people to get married (so the marriage rate would increase by 10 percentage points)?

We can use the Bayesian model, which is generative, to make such predictions. Consider a state with a marriage rate of 2 (per 10 adults). The predicted divorce rate is

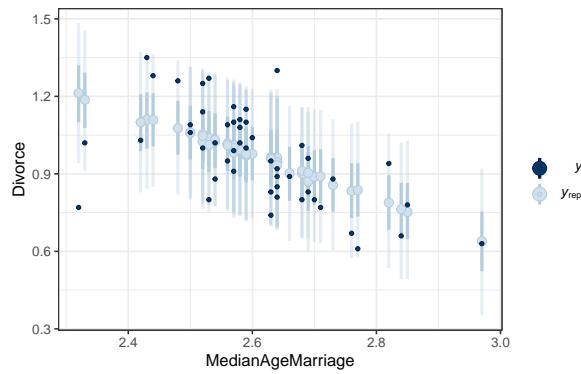
$$\beta_0 + \beta_1 A + \beta_2(2)$$



(a) Posterior predictive density of divorce rate.



(b) Posterior predictive intervals of marriage rate predicting divorce rate.



(c) Posterior predictive intervals of median marriage age predicting marriage rate.

Figure 13.4: Posterior predictive checks for the multiple regression model.

Consider an intervention that increases the marriage rate from 2 to 3. The predicted divorce rate will be

$$\beta_0 + \beta_1 A + \beta_2(3)$$

The change in divorce rate is

$$\beta_0 + \beta_1 A + \beta_2(3) - \beta_0 + \beta_1 A + \beta_2(2) = \beta_2$$

Here is what the model would predict in terms of the change in the divorce rate (holding median marriage age to 25 years old), using R:

```
pred_df <- data.frame(
  Marriage = c(2, 3),
  MedianAgeMarriage = c(2.5, 2.5)
)
cbind(pred_df, fitted(m1, newdata = pred_df)) %>%
  knitr::kable(digits = 3)
```

Marriage	MedianAgeMarriage	Estimate	Est.Error	Q2.5	Q97.5
2	2.5	1.068	0.035	1.002	1.138
3	2.5	1.026	0.067	0.894	1.160

The difference should be just the estimate of  $\beta_2$ , which has the following posterior distribution:

```
mcmc_dens(m1, pars = "b_Marriage")
```

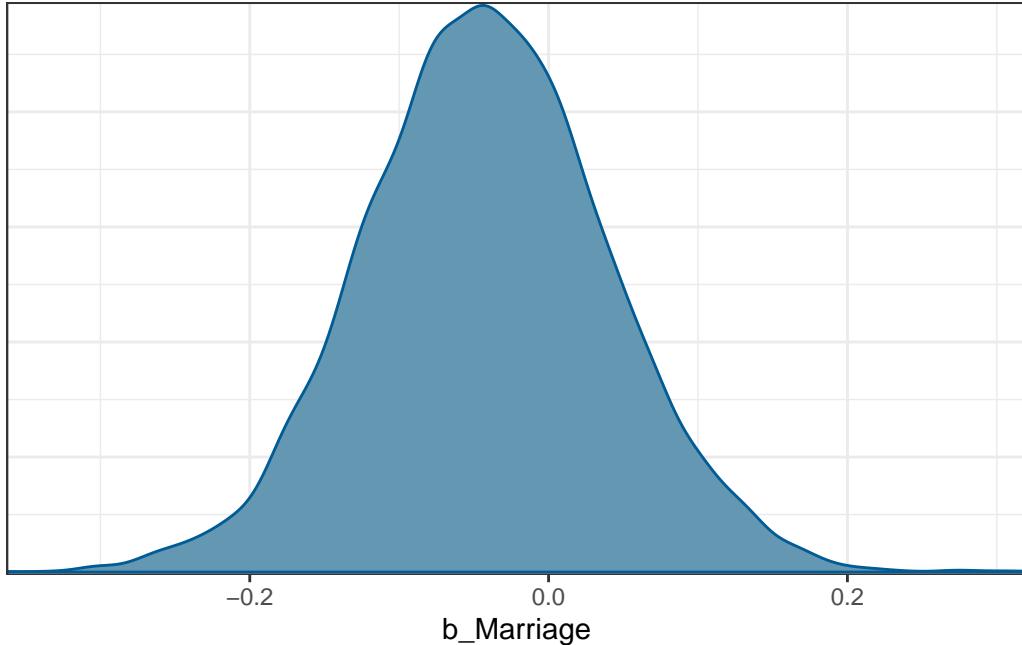


Figure 13.5: Estimated “causal” effect of marriage rate on divorce rate, based on the specified DAG and a linear model.

## 13.6 Collider

A collider is a descendant of two parent nodes. When one holds the collider constant, it induces a spurious association between the parents. This leads to many unexpected results in data analysis and daily life. For example, consider people who are nice and who are good-looking. Is there an association between a person being nice and a person being good-looking? Maybe. But this association can be induced when we condition on a collider. Let’s say a friend of yours only dates people who are either nice or good-looking. So we have this colliding relation:

$$\text{Nice} \rightarrow \text{Dating} \leftarrow \text{Good looking}$$

Suppose you look at just the people that your friend dated. In that case, they are either nice or good-looking, but the fact that you select to look at (condition on) only the people that your friend dated would automatically eliminate people who are neither nice nor good-looking. This induces a spurious negative association between nice and good-looking, so your friend may believe that nice people would be less likely to have appearances that fit their taste, and vice versa.

As another example, is research that is newsworthy less trustworthy? There could be a negative association due to collider bias, because people who select research to be reported, funded, or

published, consider both newsworthiness and trustworthiness. Assume that trustworthiness has no causal relation with newsworthiness, below shows what happens when we only focus on papers that are either high on newsworthiness or trustworthiness:

```
set.seed(2221) # different seed from the text
num_proposals <- 200 # number of grant proposals
prop_selected <- 0.1 # proportion to select
# Simulate independent newsworthiness and trustworthiness
plot_dat <- data.frame(
  nw = rnorm(num_proposals),
  tw = rnorm(num_proposals)
)
plot_dat <- plot_dat |>
  mutate(total = nw + tw)
sel_dat <- plot_dat |>
  # select top 10% of combined scores
  slice_max(order_by = total, prop = prop_selected)
plot_dat |>
  ggplot(aes(x = nw, y = tw)) +
  geom_point() +
  geom_point(data = sel_dat, shape = 1, size = 3,
             color = "red") +
  geom_smooth(method = "lm", se = FALSE) +
  geom_smooth(data = sel_dat, method = "lm", se = FALSE,
              col = "purple") +
  labs(x = "newsworthiness", y = "trustworthiness")

`geom_smooth()` using formula = 'y ~ x'
`geom_smooth()` using formula = 'y ~ x'
```

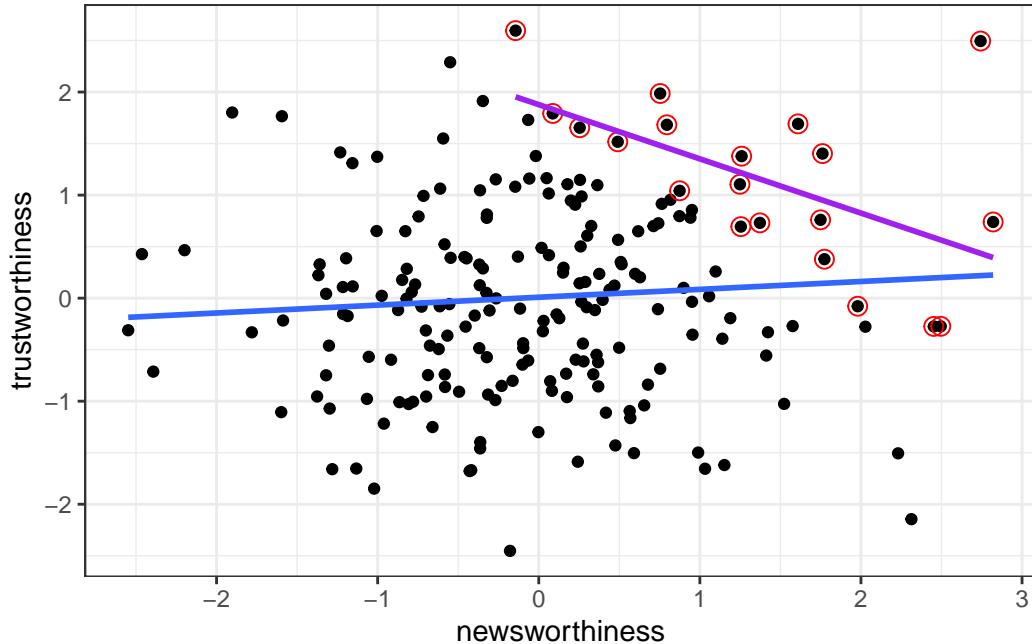


Figure 13.6: Simulation of newsworthiness and trustworthiness.

As you can see, a negative correlation happens. This is collider bias: spurious correlation due to conditioning on the common descendant. This also goes by the name [Berkson's Paradox](#).

In the DAG below, X and Y have no causal association. Conditioning on S, however, induces a negative correlation between X and Y. With the DAG below,

Generally speaking, do not condition on a collider, unless you're going to de-confound the spurious association using other variables.

```
dag7 <- dagitty(
  "dag{
    X -> Y; X -> S; Y -> S
  }"
)
coordinates(dag7) <- list(x = c(X = 0, S = 1, Y = 2),
                           y = c(X = 0, S = -1, Y = 0))
ggdag(dag7) + theme_dag()
```

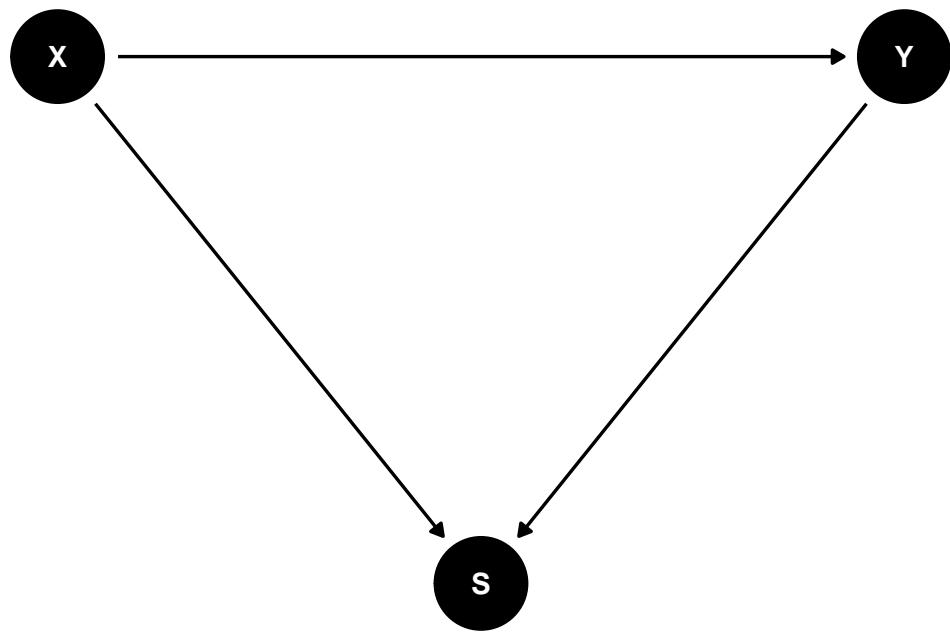


Figure 13.7: DAG with collider bias.

**Age and Happiness, conditioned on Marital Status (Example from text)**

In this example, we assume that age and happiness are not related, but as people age, they are more likely to get married.

```

# Code adapted from the rethinking package (https://github.com/rmcelreath/rethinking/blob/master/R/sim_happiness.R)
sim_happiness <- function(seed = 1977, N_years = 100,
                           max_age = 65, N_births = 50,
                           aom = 18) {
  set.seed(seed)
  H <- M <- A <- c()
  for (t in seq_len(N_years)) {
    A <- A + 1 # age existing individuals
    A <- c(A, rep(1, N_births)) # newborns
    H <- c(H, seq(from = 1, to = 5, length.out = N_births)) # sim happiness trait - new
    M <- c(M, rep(0, N_births)) # not yet married
    # for each person over 17, chance get married
    M[A >= aom & M == 0] <-
      rbinom(A[A >= aom & M == 0], size = 1,
             prob = plogis(H[A >= aom & M == 0] - 7.9))
    # mortality
    deaths <- which(A > max_age)
    if (length(deaths) > 0) {
      A <- A[-deaths]
      H <- H[-deaths]
      M <- M[-deaths]
    }
  }
  d <- data.frame(age = A, married = M, happiness = H)
  return(d)
}
dd <- sim_happiness(2024, N_years = 100)

ggplot(dd, aes(x = age, y = happiness)) +
  geom_point(alpha = .1, size = .5)

```

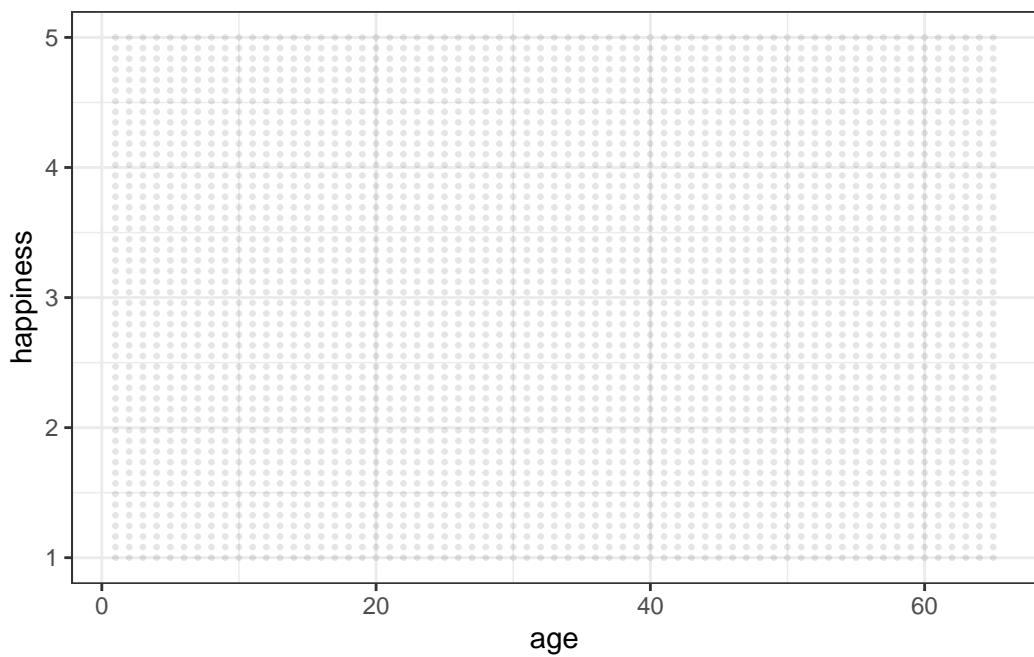


Figure 13.8: Age and happiness.

```
ggplot(dd[dd$married == 1, ], aes(x = age, y = happiness)) +  
  geom_point(alpha = .5, size = .5)
```

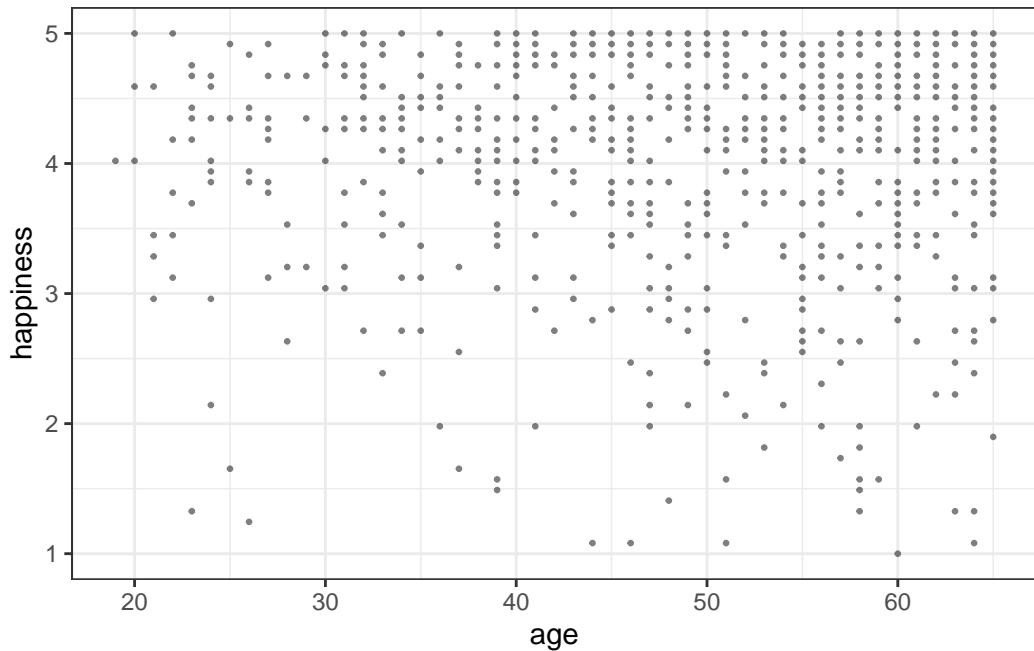


Figure 13.9: Age and happiness, conditioned on married couples.

Some other collider examples:

- impulsivity → high-risk youth ← delinquency
- healthcare worker → COVID-19 testing ← COVID-19 severity
- standardized test → admission ← research skills
- maternal smoking → birth weight → birth defect ← mortality

## 13.7 Some Additional References

- <https://doi.org/10.1177/25152459221095823>
- <https://doi.org/10.1038/s41562-024-01939-z>

# 14 Mediation

We'll use the `framing` data set from the `mediation` package, so you'll need to have the package installed first. The data were analyzed in a [report in the American Journal of Political Science](#). The study examined whether “news about the costs of immigration boosts white opposition” (p. 959).

## 14.1 Summary Statistics Tables

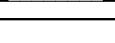
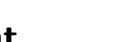
I'll take a quick detour to introduce you to some useful functions for tabulating your data. The functions will be from the `modelsummary` package.

```
data(framing, package = "mediation")
# Subtract the `emo` variable by 3 so that it starts from 0-9
framing$emo <- framing$emo - 3
# Quick summary of selected variables
datasummary_skim(framing)

# Summary by treatment condition (`tone`)
datasummary_balance(~ tone, data = framing)

# More tailor-made table with selected variables by treatment condition
datasummary(emo + p_harm + cong_mesg ~ Factor(tone) * (Mean + SD + Histogram),
            data = framing)

# Correlation table
framing |>
  select(tone, emo, cong_mesg) |>
  datasummary_correlation(method = "pearson")
```

	Unique (#)	Missing (%)	Mean	SD	Min	Median	Max	
age	63	0	47.8	16.0	18.0	47.0	85.0	
income	19	0	10.8	3.9	1.0	11.0	19.0	
emo	10	0	4.0	2.8	0.0	4.0	9.0	
p_harm	7	0	5.9	1.8	2.0	6.0	8.0	
tone	2	0	0.5	0.5	0.0	1.0	1.0	
eth	2	0	0.5	0.5	0.0	1.0	1.0	
treat	2	0	0.3	0.4	0.0	0.0	1.0	
immigr	4	0	3.0	1.0	1.0	3.0	4.0	
anti_info	2	0	0.1	0.3	0.0	0.0	1.0	
cong_mesg	2	0	0.3	0.5	0.0	0.0	1.0	

## 14.2 Randomization Removes Incoming Paths for Treatment

Let's consider a DAG without randomization, for the following two variables:

- X: Exposure to a negatively framed news story about immigrants
- Y: Anti-immigration political action

There are many possible confounders when we observe these two variables in data (e.g., an individual's political affiliation and the state in which a person resides). We can represent these unobserved confounders as U, so the DAG will be something like

```
dag2 <- dagitty(
  "dag{
    X -> Y; U -> X; U -> Y
    U [unobserved]
  }"
)
coordinates(dag2) <- list(x = c(X = 0, U = 1, Y = 2),
                           y = c(X = 0, U = 1, Y = 0))
# Plot
ggdag(dag2) + theme_dag()
```

		0			1			Diff. in Means	Std. Errc
		Mean	Std. Dev.		Mean	Std. Dev.			
age		48.4	16.0	47.1	16.0		-1.3	2.2	
income		11.0	3.9	10.6	3.9		-0.4	0.6	
emo		3.4	2.6	4.5	2.8		1.1	0.6	
p_harm		5.5	1.8	6.2	1.7		0.7	0.6	
eth		0.5	0.5	0.5	0.5		0.0	0.6	
treat		0.0	0.0	0.5	0.5		0.5	0.6	
immigr		2.8	1.0	3.2	0.9		0.4	0.6	
anti_info		0.1	0.3	0.1	0.3		0.0	0.6	
cong_mesg		0.3	0.5	0.4	0.5		0.0	0.6	
		N		Pct.	N		Pct.		
cond	not asked	0		0.0	0		0.0		
	refused	0		0.0	0		0.0		
	control	0		0.0	0		0.0		
	1	0		0.0	68		50.4		
	2	0		0.0	67		49.6		
	3	67		51.5	0		0.0		
	4	63		48.5	0		0.0		
anx	not asked	0		0.0	0		0.0		
	refused	0		0.0	0		0.0		
	very anxious	34		26.2	26		19.3		
	somewhat anxious	48		36.9	38		28.1		
	a little anxious	31		23.8	43		31.9		
	not anxious at all	17		13.1	28		20.7		
educ	not asked	0		0.0	0		0.0		
	refused	0		0.0	0		0.0		
	less than high school	6		4.6	14		10.4		
	high school	48		36.9	44		32.6		
	some college	31		23.8	39		28.9		
	bachelor's degree or higher	45		34.6	38		28.1		
gender	not asked	0		0.0	0		0.0		
	refused	0		0.0	0		0.0		
	male	64		49.2	62		45.9		
	female	66		50.8	73		54.1		
english	Strongly Favor	1		0.8	6		4.4		
	Favor	15		11.5	10		7.4		
	Oppose	44		33.8	38		28.1		
	Strongly Oppose	70		53.8	81		60.0		

	0			1		
	Mean	SD	Histogram	Mean	SD	Histogram
emo	3.39	2.63		4.53	2.81	
p_harm	5.53	1.77		6.23	1.69	
cong_mesg	0.31	0.46		0.36	0.48	

	tone	emo	cong_mesg
tone	1	.	.
emo	0.21	1	.
cong_mesg	0.05	0.37	1

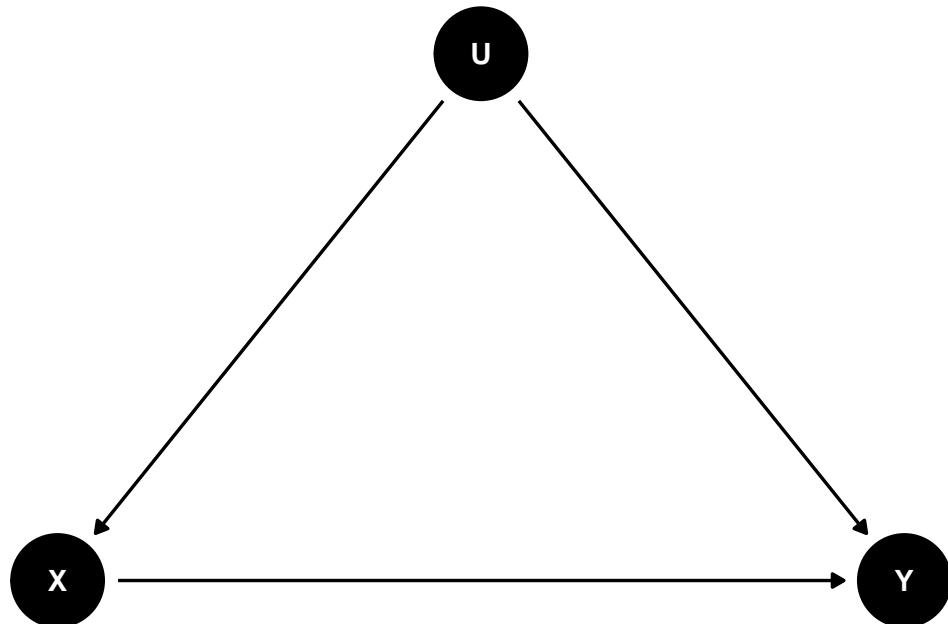


Figure 14.1: DAG with an unobserved confounder

The magic of randomization—randomly assigning individuals to a manipulated level of X—is that it blocks the path  $U \rightarrow X$ . Therefore, with randomization, the reason a person sees a negatively-framed news story about immigrants is not related to reasons that the person may have intentions for anti-immigration actions. The DAG becomes

```

dag3 <- dagitty(
  "dag{"
  
```

```

    X -> Y; U -> Y
    U [unobserved]
}""
)
coordinates(dag3) <- list(x = c(X = 0, U = 1, Y = 2),
                           y = c(X = 0, U = 1, Y = 0))
# Plot
ggdag(dag3) + theme_dag()

```

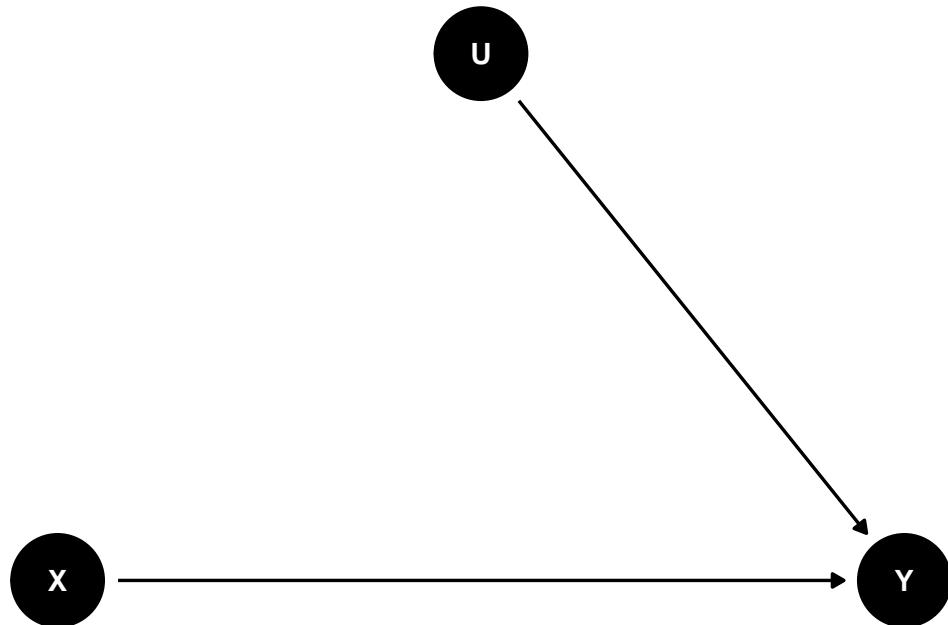


Figure 14.2: DAG with randomization

Therefore, when randomization is successful, the path coefficient of  $X \rightarrow Y$  is the causal effect of  $X$  on  $Y$ . However, randomized experiments do not always rule out all confounding. For example, suppose participants are randomly assigned to different experimental conditions, but those who disagree with the presented news story drop out. In that case, such attrition can induce a non-zero correlation between  $X$  and  $Y$  in the remaining sample.

## 14.3 Causal Chains

### 14.3.1 Post-Treatment Bias

Here are some key variables from the `framing` data set:

- `cong_mesg`: binary variable indicating whether or not the participant agreed to send a letter about immigration policy to his or her member of Congress
- `emo`: posttest anxiety about increased immigration (0-9)
- `tone`: framing of news story (0 = positive, 1 = negative)

We can compare the results of two models:

1. `tone → cong_mesg`
2. `tone → cong_mesg`, adjusting for `emo`

Which model gives us the causal effect estimate for `tone`? Let's run the two models.

```
m_no_adjust <- brm(cong_mesg ~ tone,
                      data = framing,
                      family = bernoulli(link = "logit"),
                      file = "08b_m_no_adjust")
m_adjust <- brm(cong_mesg ~ tone + emo,
                  data = framing,
                  family = bernoulli(link = "logit"),
                  file = "08b_m_adjust")
```

We can combine the results in a table:

```
msummary(list(`No adjustment` = m_no_adjust,
              `Adjusting for feeling` = m_adjust),
            estimate = "{estimate} [{conf.low}, {conf.high}]",
            statistic = NULL, fmt = 2
      )
```

Warning:

`modelsummary` uses the `performance` package to extract goodness-of-fit statistics from models of this class. You can specify the statistics you wish to compute by supplying a `metrics` argument to `modelsummary`, which will then push it forward to `performance`. Acceptable values are: "all", "common", "none", or a character vector of metrics names. For example: `modelsummary(mod, metrics = c("RMSE", "R2"))` Note that some metrics are computationally expensive. See `?performance::performance` for details.

This warning appears once per session.

	No adjustment	Adjusting for feeling
b_Intercept	-0.81 [-1.19, -0.46]	-2.00 [-2.63, -1.44]
b_tone	0.21 [-0.28, 0.71]	-0.13 [-0.71, 0.43]
b_emo		0.32 [0.21, 0.43]
Num.Obs.	265	265
R2	0.003	0.142
ELPD	-170.1	-152.6
ELPD s.e.	5.5	7.4
LOOIC	340.2	305.2
LOOIC s.e.	11.0	14.8
WAIC	340.2	305.2
RMSE	0.47	0.44

We can see that the Bayes estimate of the coefficient for `tone` was positive without `emo`, but was negative with `emo`. Which one should we believe? The information criteria (LOOIC and WAIC) suggested that the model with `emo` was better for prediction, but just because something helps predict the outcome does not make it a causal variable. To repeat,

Coefficients in a predictive model are not causal effects.

And to emphasize again,

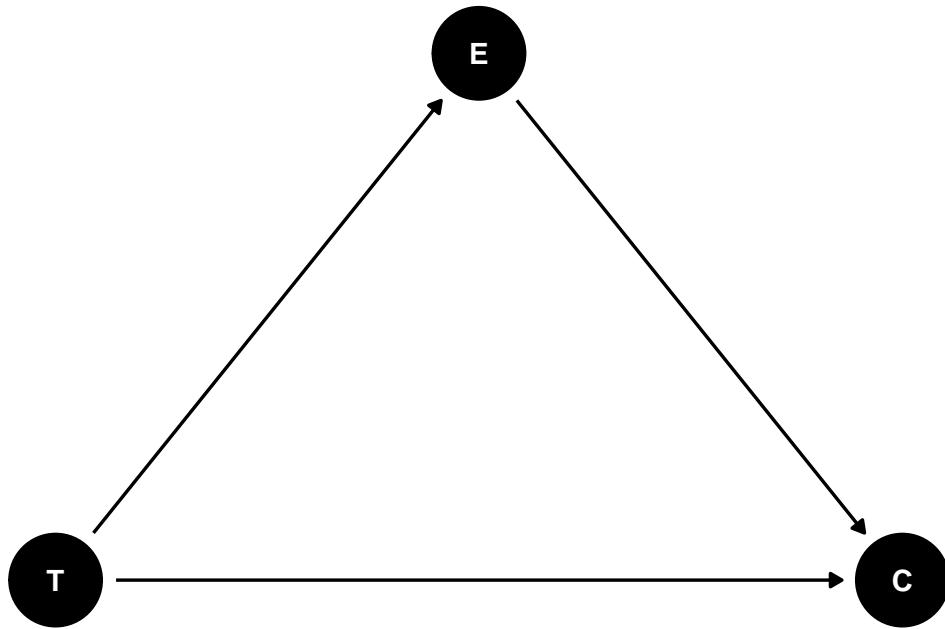
Causal inference requires causal assumptions, and these assumptions are not in the data.

So instead, we need a DAG. Given that we know `emo` is measured **after** the intervention, it seems reasonable to think that a negatively framed story about immigrants would elicit some negative emotions about increased immigration, and that emotion may prompt people to take anti-immigrant actions. Therefore, we have the following DAG:

```
dag5 <- dagitty(
  "dag{
    T -> C; T -> E; E -> C
  }"
)
coordinates(dag5) <- list(x = c(T = 0, E = 1, C = 2),
                           y = c(T = 0, E = 1, C = 0))
# Plot
ggdag(dag5) + theme_dag()
```

This is an example of a pipe/chain. It is a causal chain going from  $T(\text{one}) \rightarrow E(\text{motion}) \rightarrow C(\text{ongress message})$ . If we are interested in the causal effect of  $T$ , we should not condition

Table 14.1: DAG for a mediation model with emotion as a mediator.



on E; conditioning on E would mean comparing those who saw the negatively-framed story and *those who saw the positively-framed story but had the same negative emotion towards immigrants*. In other words, adjusting for E would mean taking out part of the effect of T on C through E.

As another example, think about a drug that is supposed to lower the risk of a heart attack. Imagine someone conducting a study comparing drug/no drug conditions on their probability of getting heart attacks. Should we adjust for participants' blood pressure after the intervention? If we want to know the drug's effect, and that the drug works by lowering blood pressure, we should not adjust for posttest blood pressure. Otherwise, we would be asking the question: Does the drug help prevent heart attacks through something other than lowering one's blood pressure?

The latter question can be answered in mediation analysis.

## 14.4 Causal Mediation

In the DAG above, E is a post-treatment variable potentially influenced by T, which we call a **mediator**. Mediator is an important topic in causal inference, as it informs the mechanism of how a variable has a causal effect on an outcome.

One thing to be careful of is that, statistically speaking, a mediator behaves very much like a confounder, and their difference is based on causal assumptions.

Let's analyze the mediation model in Figure X. There are two variables that are on the receiving end of some causal effects: `emo` and `cong_mesg`. Whereas the generalized linear model handles one outcome, with `brms`, we can have a system of equations—one for each outcome—estimated simultaneously, as shown in the code below.

#### 14.4.1 Using `brms`

```
m_med <- brm(  
  # Two equations for two outcomes  
  bf(cong_mesg ~ tone + emo) +  
    bf(emo ~ tone) +  
    set_rescor(FALSE),  
  # A list of two family arguments for two outcomes  
  family = list(bernoulli("logit"), gaussian("identity")),  
  data = framing,  
  prior = prior(normal(0, 2), class = "b", resp = "emo") +  
    prior(student_t(4, 0, 5), class = "sigma", resp = "emo") +  
    prior(student_t(4, 0, 2.5), class = "b", resp = "congmesg"),  
  seed = 1338,  
  file = "08b_m_med"  
)
```

#### 14.4.2 Using Stan

Here's some Stan code for running the same mediation model

```
data {  
  int<lower=0> N0; // number of observations (control)  
  int<lower=0> N1; // number of observations (treatment)  
  array[N0] int y0; // outcome (control);  
  array[N1] int y1; // outcome (treatment);  
  vector[N0] m0; // mediator (control);  
  vector[N1] m1; // mediator (treatment);  
}  
parameters {  
  real alpham; // regression intercept for M  
  real alphay; // regression intercept for M
```

```

real beta1;    // X -> M
real beta2;    // X -> Y
real beta3;    // M -> Y
real<lower=0> sigmam; // SD of prediction error for M
}
model {
    // model
    m0 ~ normal(alpham, sigmam);
    m1 ~ normal(alpham + beta1, sigmam);
    y0 ~ bernoulli_logit(alphay + beta3 * m0);
    y1 ~ bernoulli_logit(alphay + beta2 + beta3 * m1);
    // prior
    alpham ~ normal(4.5, 4.5);
    alphay ~ normal(0, 5);
    beta1 ~ std_normal();
    beta2 ~ std_normal();
    beta3 ~ std_normal();
    sigmam ~ student_t(4, 0, 5);
}

```

```
med_mod <- cmdstan_model("stan_code/mediation_logit_normal.stan")
```

```

# 1. Form the data list for Stan
stan_dat <- with(
  framing,
  list(
    N0 = sum(tone == 0),
    N1 = sum(tone == 1),
    m0 = emo[which(tone == 0)],
    m1 = emo[which(tone == 1)],
    y0 = cong_mesg[which(tone == 0)],
    y1 = cong_mesg[which(tone == 1)]
  )
)
# 2. Run Stan
m_med_stan <- med_mod$sample(
  data = stan_dat,
  seed = 1338,
  refresh = 1000
)

```

### 14.4.3 Direct and Indirect Effects

In a mediation model, the effect of X on Y has two mechanisms:

- Indirect effect: X causes Y because X causes M, and M causes Y
- Direct effect: X causes Y without involving M

More specifically, the direct effect is the change in Y for one unit change in X, *holding M constant*. In our example, it means comparing subsets of the treatment and the control groups, under the condition that both subsets have the same level of negative emotion about increased immigration. The indirect effect takes more effort to understand: it is the change in Y for the control group (or the treatment group) if their mediator value is set to the same level as the treatment group. In our example, it would mean comparing the control group and the *counterfactual* where the control group had their negative emotion changed to the same level as the treatment group.

The *causal mediation* literature has more distinctions on the different types of direct and indirect effects. Below, I give codes for obtaining these effects without going into detail.<sup>1</sup>

### 14.4.4 Controlled direct effect (CDE)

CDE is the direct effect when the mediator is set to a specific level. Below is the CDE for `emo = 0` and `emo = 9`, respectively.

```
cond_df <- data.frame(tone = c(0, 1, 0, 1),
                      emo = c(0, 0, 9, 9))
cond_df |>
  bind_cols(
    fitted(m_med, newdata = cond_df)[ , , "congmesg"])
) |>
  knitr::kable()
```

Table 14.2: Estimated direct effect of `tone` on `cong_mesg` at two different levels of `emo`.

tone	emo	Estimate	Est.Error	Q2.5	Q97.5
0	0	0.1212844	0.0320367	0.0662354	0.1907005
1	0	0.1084567	0.0330696	0.0538386	0.1830547
0	9	0.6982051	0.0698306	0.5559154	0.8234034
1	9	0.6702609	0.0631406	0.5418592	0.7855347

<sup>1</sup>You can find more information about causal mediation in this paper: [https://ftp.cs.ucla.edu/pub/stat\\_ser/r389.pdf](https://ftp.cs.ucla.edu/pub/stat_ser/r389.pdf)

#### 14.4.5 Natural direct effect (NDE)

The “natural” effects are quantities for some kind of population averages. NDE is the direct effect when the mediator is held constant at the level of the control group. As a first step, we need to obtain the potential outcome of `emo` when `tone = 0`. We call this potential outcome variable  $M_0$  (potential outcome of  $M$  if  $X = 0$ ). This has already been observed for the control group but will be a counterfactual for the treatment group.

We will use a general approach by Imai et al. (2010) to simulate potential outcomes for each observation, before computing NDE (and NIE). For each observation, we will first simulate the potential outcome of `emo` when `tone = 0` ( $M_0$ ), and then simulate the potential outcome of `emo` when `tone = 1` ( $M_1$ ).

```
# Simulate potential outcomes for mediator when T = 0
dat0 <- m_med$data
dat0$tone <- 0
# Predicted emo when T = 0
po_m0 <- posterior_predict(m_med, newdata = dat0, resp = "emo")
# Simulate potential outcomes for mediator when T = 1
dat1 <- m_med$data
dat1$tone <- 1
# Predicted emo when T = 1
po_m1 <- posterior_predict(m_med, newdata = dat1, resp = "emo")
```

Next, we will simulate four potential outcomes for `cong_mesg` (Y):

- $Y(T = 0, M = M_0)$
- $Y(T = 1, M = M_0)$
- $Y(T = 0, M = M_1)$
- $Y(T = 1, M = M_1)$

```
m_med_draws <- as_draws_df(m_med)
# Predicted logit of congmesg when T = 0 and emo = po_m0
po_y0_m0 <- m_med_draws$b_congmesg_Intercept + m_med_draws$b_congmesg_emo * po_m0
# Predicted logit of congmesg when T = 1 and emo = po_m0
po_y1_m0 <- m_med_draws$b_congmesg_Intercept +
  m_med_draws$b_congmesg_tone +
  m_med_draws$b_congmesg_emo * po_m0
# Predicted logit of congmesg when T = 0 and emo = po_m1
po_y0_m1 <- m_med_draws$b_congmesg_Intercept + m_med_draws$b_congmesg_emo * po_m1
# Predicted logit of congmesg when T = 1 and emo = po_m1
po_y1_m1 <- m_med_draws$b_congmesg_Intercept +
```

Table 14.3: Estimated natural direct and indirect effects for the control group.

```
draws_nde_nie <- as_draws(list(NDE = nde, NIE = nie))
posterior::summarise_draws(draws_nde_nie)

# A tibble: 2 x 10
  variable    mean   median     sd     mad      q5     q95   rhat ess_bulk ess_tail
  <chr>      <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
1 NDE       -0.0246 -0.0249 0.0518 0.0523 -0.110  0.0615  1.00    5171.   3113.
2 NIE        0.0681  0.0668 0.0276 0.0267  0.0246  0.116   1.00    3916.   2977.
```

```
m_med_draws$b_congmesg_tone +
m_med_draws$b_congmesg_emo * po_m1
```

The NDE is defined as  $Y(T = 1, M = M_0) - Y(T = 0, M = M_0)$ .

```
# NDE = Y(1, M(0)) - Y(0, M(0))
nde <- rowMeans(plogis(po_y1_m0)) - rowMeans(plogis(po_y0_m0))
```

#### 14.4.5.1 Natural indirect effect (NIE)

NIE is the difference in the outcome between the actual control group and a counterfactual control group. The counterfactual here is a control group that did not receive the treatment, but had their `emo` value set to be equal to the treatment group. The NIE is defined as  $Y(T = 0, M = M_1) - Y(T = 0, M = M_0)$ .

```
# NIE = Y(0, M(1)) - Y(0, M(0))
nie <- rowMeans(plogis(po_y0_m1)) - rowMeans(plogis(po_y0_m0))
```

```
mcmc_areas(draws_nde_nie, bw = "SJ")
```

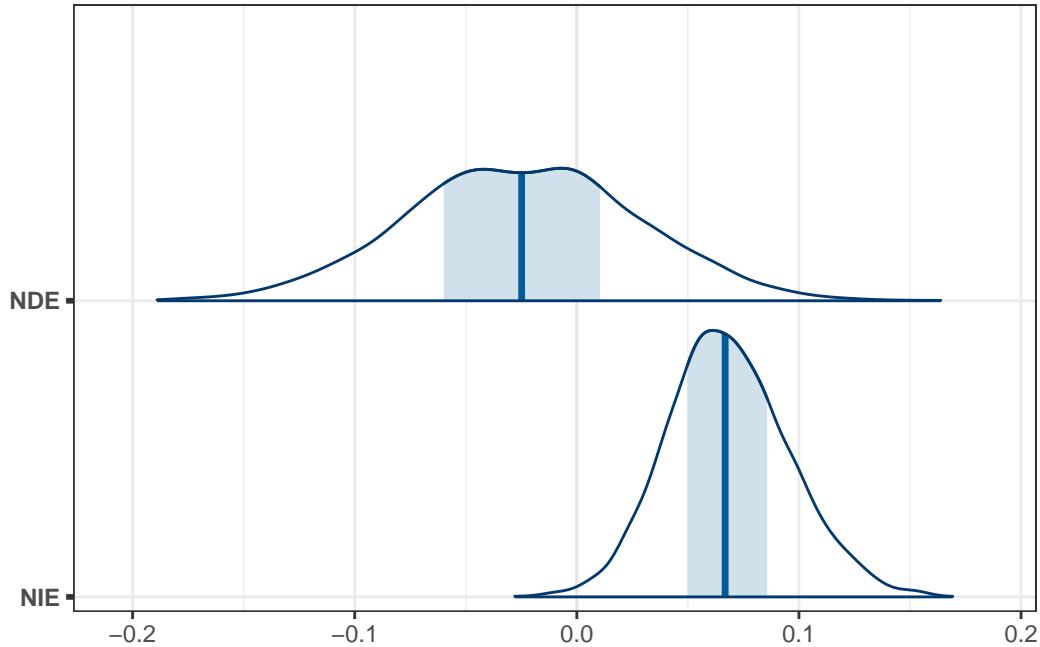


Figure 14.3: Estimated natural direct and indirect effects for the control group.

#### 14.4.6 Sensitivity analysis

```

dag6 <- dagitty(
  "dag{
    T -> C; T -> E; E -> C; U -> E; U -> C
  }"
)
coordinates(dag6) <- list(x = c(T = 0, E = 1, C = 2, U = 2),
                           y = c(T = 0, E = 1, C = 0, U = 1))
ggdag(dag6) + theme_dag()

```

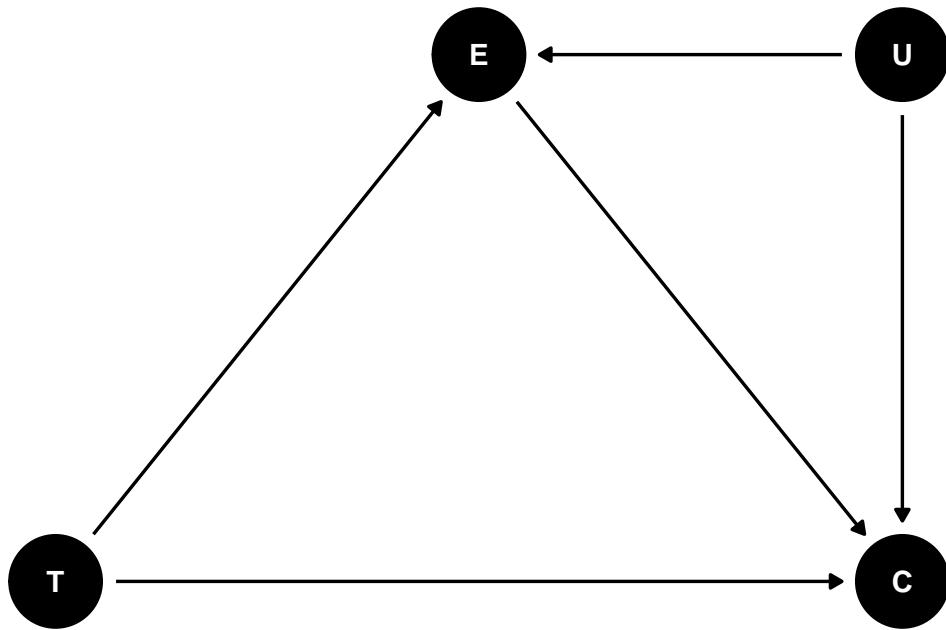


Figure 14.4: DAG for a mediation model with an unobserved mediator-outcome confounder.

### ! Assumptions of Causal Mediation

Mediation effects can be estimated properly when the causal diagram and the corresponding model are specified correctly. In our model, we assume

- No unmeasured treatment-outcome confounding
- No unmeasured mediator-outcome confounding
- No unmeasured treatment-mediator confounding
- The mediator-outcome path is not moderated by the treatment

Note that randomization of the treatment does not rule out confounding for the mediator-outcome path.

One important assumption in mediation is that there is no unobserved confounding variable between the mediator and the outcome. This assumption requires researchers' input, as usually we don't have studies that randomly assign both the treatment variable and the mediator. An additional technique is to ask: what would the effect be if there were unobserved confounding variables of a certain magnitude? With Bayesian, we can represent the strength of the confounding effect by a prior distribution (see McCandless & Somers, 2019, p. 10.1177/0962280217729844). The following shows the mediation estimates assuming the effect of the confounding variable to the mediator is 1, and to the outcome has a prior of  $N(0.5,$

0.2).

```
data {
    int<lower=0> N0; // number of observations (control)
    int<lower=0> N1; // number of observations (treatment)
    array[N0] int y0; // outcome (control);
    array[N1] int y1; // outcome (treatment);
    vector[N0] m0; // mediator (control);
    vector[N1] m1; // mediator (treatment);
}
parameters {
    real alpham; // regression intercept for M
    real alphay; // regression intercept for M
    real beta1; // X -> M
    real beta2; // X -> Y
    real beta3; // M -> Y
    vector[N0] u0; // confounding variable
    vector[N1] u1; // confounding variable
    real beta4; // U -> Y
    real<lower=0> sigmam; // SD of prediction error for M
}
model {
    // model
    u0 ~ std_normal();
    u1 ~ std_normal();
    m0 ~ normal(alpham + u0, sigmam);
    m1 ~ normal(alpham + beta1 + u1, sigmam);
    y0 ~ bernoulli_logit(alphay + beta3 * m0 + beta4 * u0);
    y1 ~ bernoulli_logit(alphay + beta2 + beta3 * m1 + beta4 * u1);
    // prior
    alpham ~ normal(4.5, 4.5);
    alphay ~ normal(0, 5);
    beta1 ~ std_normal();
    beta2 ~ std_normal();
    beta3 ~ std_normal();
    beta4 ~ normal(0.5, 0.2);
    sigmam ~ student_t(4, 0, 5);
}
```

```
med_mod_sens <- cmdstan_model("stan_code/mediation_logit_normal_sensitivity.stan")
```

```

# 1. form the data list for Stan
stan_dat <- with(
  framing,
  list(
    N0 = sum(tone == 0),
    N1 = sum(tone == 1),
    m0 = emo[which(tone == 0)],
    m1 = emo[which(tone == 1)],
    y0 = cong_mesg[which(tone == 0)],
    y1 = cong_mesg[which(tone == 1)]
  )
)
# 2. Run Stan
m_med_sens <- med_mod_sens$sample(
  data = stan_dat,
  seed = 1338,
  refresh = 1000
)

```

Running MCMC with 4 sequential chains...

```

Chain 1 Iteration: 1 / 2000 [ 0%] (Warmup)
Chain 1 Iteration: 1000 / 2000 [ 50%] (Warmup)
Chain 1 Iteration: 1001 / 2000 [ 50%] (Sampling)
Chain 1 Iteration: 2000 / 2000 [100%] (Sampling)
Chain 1 finished in 1.4 seconds.
Chain 2 Iteration: 1 / 2000 [ 0%] (Warmup)
Chain 2 Iteration: 1000 / 2000 [ 50%] (Warmup)
Chain 2 Iteration: 1001 / 2000 [ 50%] (Sampling)
Chain 2 Iteration: 2000 / 2000 [100%] (Sampling)
Chain 2 finished in 1.1 seconds.
Chain 3 Iteration: 1 / 2000 [ 0%] (Warmup)
Chain 3 Iteration: 1000 / 2000 [ 50%] (Warmup)
Chain 3 Iteration: 1001 / 2000 [ 50%] (Sampling)
Chain 3 Iteration: 2000 / 2000 [100%] (Sampling)
Chain 3 finished in 1.0 seconds.
Chain 4 Iteration: 1 / 2000 [ 0%] (Warmup)
Chain 4 Iteration: 1000 / 2000 [ 50%] (Warmup)
Chain 4 Iteration: 1001 / 2000 [ 50%] (Sampling)
Chain 4 Iteration: 2000 / 2000 [100%] (Sampling)
Chain 4 finished in 1.0 seconds.

```

```
All 4 chains finished successfully.
Mean chain execution time: 1.1 seconds.
Total execution time: 4.9 seconds.
```

`beta1 = tone -> emo; beta2 = tone -> cong_mesg; beta3 = emo -> cong_mesg`

```
m_med_sens$draws(
  c("alpham", "alphay",
    "beta1", "beta2", "beta3")
) |>
  mcmc_intervals()
```

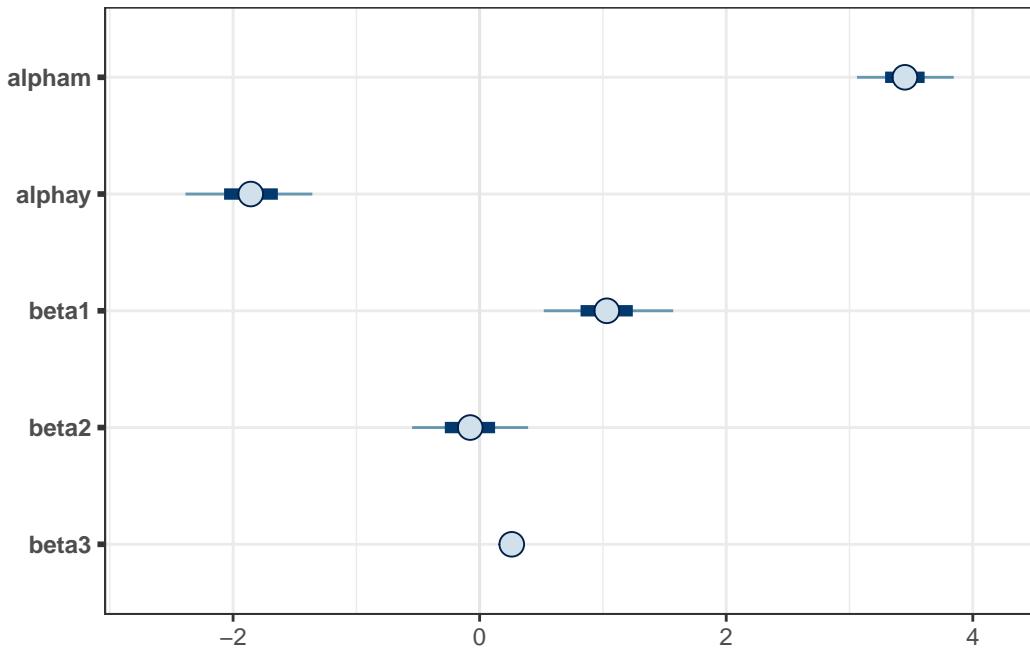


Figure 14.5: Estimated coefficients for the mediation model with the sensitivity analysis.

As you can see, the prior of the confounding effect attenuates the coefficients from the mediator to the outcome, but the path from the mediator to the outcome remains pretty much above zero.

You may also check out the [BayesGmed package](#), which is also based on Stan. More descriptions are in the paper Yimer et al. (2023).

## **Part VIII**

# **Week 10–11**

# 15 Markov Chain Monte Carlo

Previously, we have seen a few examples with Bayesian inferences where the posterior distribution concerns only one parameter, like the Bernoulli and the Poisson model. We have also discussed the grid approximation and the conjugate prior approaches to obtain/approximate the posterior. In this note, we will discuss the simulation method and explain why we need a special class of methods called *Markov Chain Monte Carlo*. This note will consider mainly the **Metropolis algorithm**, which subsumes many other commonly used MCMC algorithms. Therefore, it is beneficial to build a solid foundation on what the basic version of the Metropolis algorithm is. You will also write your own Metropolis sampler to understand how it works.

But first, let's talk about the Monte Carlo method.

## 15.1 Monte Carlo Simulation

In a previous example, we see that with a conjugate prior (e.g., Beta), the posterior distribution is from the same distributional family (Beta). Thus, we can easily draw simulation samples from the posterior distribution using R. The more samples we draw, the better we can approximate the posterior distribution based on the simulation samples. It is the same logic to get a large sample to describe our population precisely; here, the posterior distribution, determined using mathematics, is considered the population, and the simulation draws are, well, a sample from that population. With 10,000 or 100,000 samples (draws), we can accurately describe our population (posterior).

For example, if we know that the posterior is a Beta(15, 10) distribution, consider drawing 10, 100, 10,000, and 10,000 samples from it using the R function `rbeta`, and contrast the density estimated from the samples (in the histogram) with that of the actual Beta distribution (in red).

```
# Set the `seed` (initial point) for pseudo-random number generation algorithm
set.seed(2)
num_draws <- c(10, 100, 1000, 10000)
beta_draws <- data.frame(
  th = rbeta(sum(num_draws), shape1 = 15, shape2 = 10),
  sam = rep(paste(num_draws, "samples"), num_draws)
)
```

```

ggplot(beta_draws, aes(x = th)) +
  geom_histogram(aes(y = after_stat(density))) +
  stat_function(
    fun = dbeta, args = list(shape1 = 15, shape2 = 10),
    col = "red"
  ) +
  labs(x = expression(theta)) +
  facet_wrap(~sam)

```

``stat_bin()` using `bins = 30`.` Pick better value with ``binwidth``.

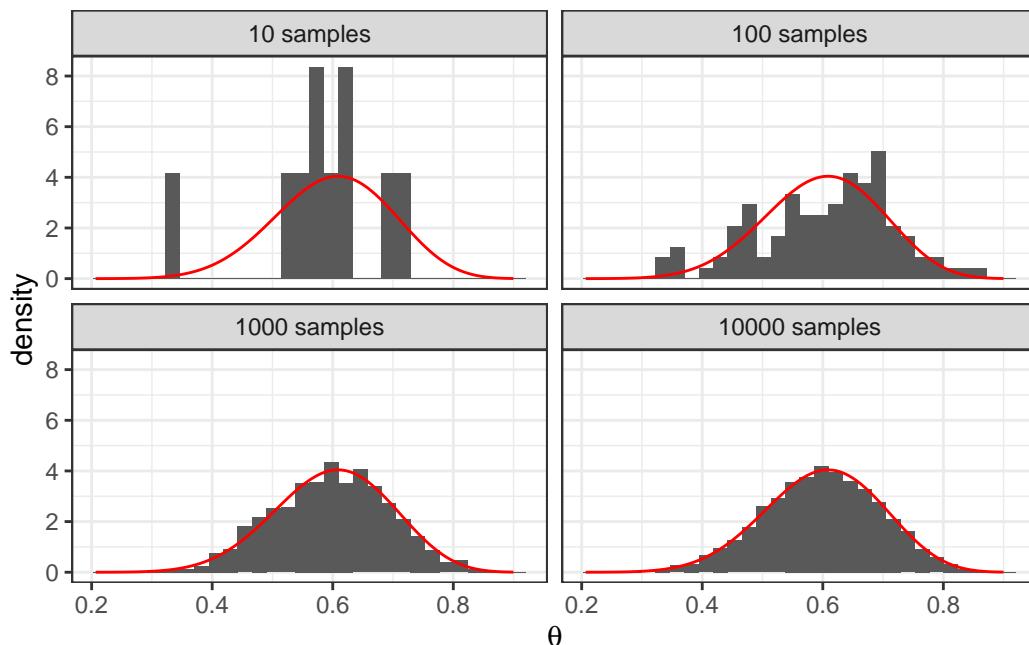


Figure 15.1: Monte Carlo simulation samples from the  $\text{Beta}(15, 10)$  distribution

The figure below shows the values when drawing 100 samples in time order:

```

beta_draws |>
  filter(sam == "100 samples") |>
  rowid_to_column("iter") |>
  ggplot(aes(y = th, x = iter)) +
  geom_line() +
  labs(y = expression(theta))

```

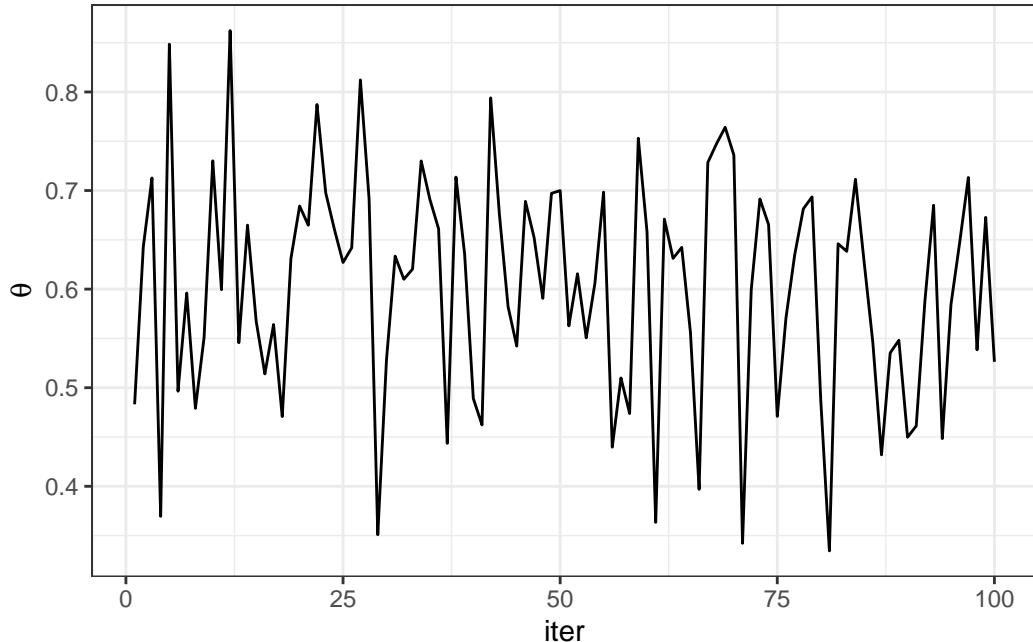


Figure 15.2: Trace plots of Monte Carlo samples.

So we can say that, when the number of posterior samples is very large, the sample distribution *converges* to the population density. The Monte Carlo method will work for many situations. Note, of course, the number of simulation samples,  $S$ , is controlled by the analysts; it is different from the sample size of the data, which is fixed and is a property of the observed data.

In addition, most descriptive statistics (e.g., mean,  $SD$ ) of the simulation draws will converge to the corresponding values of the true posterior distribution. The graphs below show how the mean, median,  $SD$ , and skewness converge to the true values (red dashed lines) when the number of simulation samples increases.

```
beta_draws |>
  filter(sam == "1000 samples") |>
  rowid_to_column("iter") |>
  mutate(
    mean = cumsum(th) / row_number(),
    median = map_dbl(row_number(), ~ median(th[1:.x])),
    SD = map_dbl(row_number(), ~ sd(th[1:.x])),
    skewness = map_dbl(row_number(), ~ e1071::skewness(th[1:.x])))
  ) |>
  ungroup() |>
```

```

gather("stat", "val", mean:skewness) |>
ggplot(aes(x = iter, y = val)) +
geom_line() +
geom_hline(
  data = data.frame(
    stat = c("mean", "median", "SD", "skewness"),
    val = c(
      15 / 25,
      qbeta(.50, 15, 10),
      sqrt(15 * 10 / (15 + 10)^2 / (15 + 10 + 1)),
      2 * (10 - 15) * sqrt(15 + 10 + 1) /
        (15 + 10 + 2) / sqrt(15 * 10)
    )
  ),
  aes(yintercept = val), col = "red", linetype = "dashed"
) +
facet_wrap(~stat, scales = "free") +
labs(y = "")

```

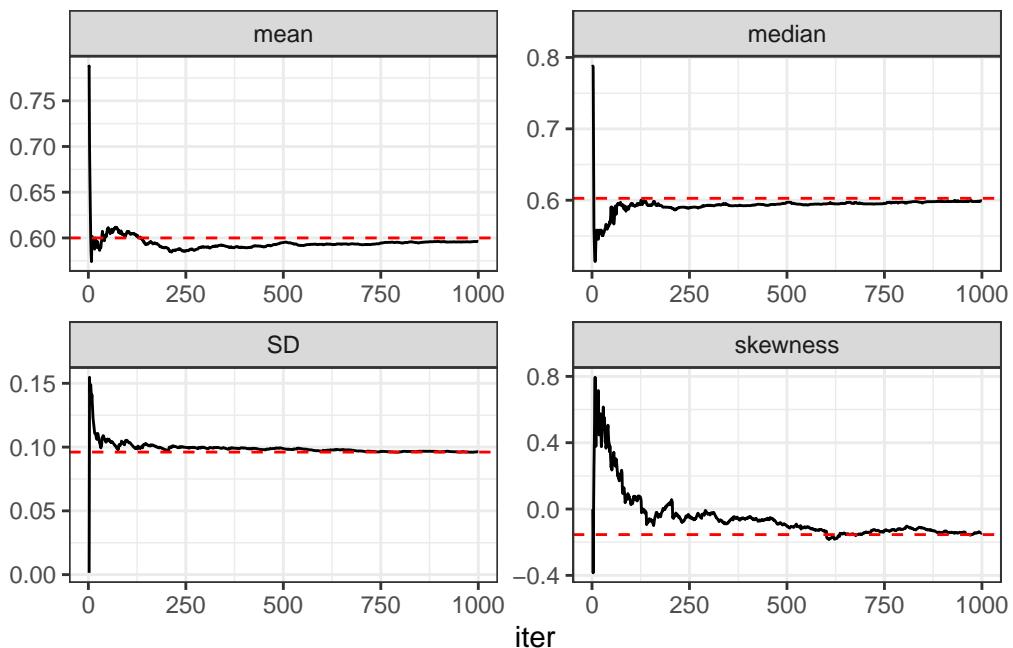


Figure 15.3: Sample statistics of the Monte Carlo samples.

## 15.2 Markov Chain Monte Carlo (MCMC): Drawing Dependent Samples

The above Monte Carlo simulation requires that (a) we know that the posterior distribution is exactly a beta distribution, and (b) R knows how to draw simulation samples from a beta distribution (with `rbeta`). As we progress through the class, it is more of an exception than we can use conjugate prior distribution, so in general, neither (a) nor (b) would hold. For example, if we instead use a normal distribution for the prior of  $\theta$ , we may get something like

$$P(\theta | y) = \frac{e^{-(\theta-1/2)^2} \theta^y (1-\theta)^{n-y}}{\int_0^1 e^{-(\theta^*-1/2)^2} \theta^{*y} (1-\theta^*)^{n-y} d\theta^*}$$

and it would be very hard, if possible, to draw simulation samples directly from the posterior. Luckily, MCMC provides a way to draw samples from the posterior distribution without the need to know everything about the posterior distribution. For example, the basic version of the Metropolis algorithm only requires that we know the *density ratio* of every two possible values  $\theta_1$  and  $\theta_2$ . Thus, we don't need to deal with the integral in the denominator, as the integral does not depend on  $\theta$  and will get canceled out when taking the ratio.

## 15.3 The Metropolis algorithm

The Metropolis algorithm can generally be used to draw samples from a distribution as long as the density ratio of any two points can be computed. Remember, in Bayesian inference, for two values in the posterior distribution, the ratio of the posterior densities at  $\theta_1$  and  $\theta_2$  is

$$\begin{aligned} \frac{P(\theta = \theta_2 | y)}{P(\theta = \theta_1 | y)} &= \frac{P(y | \theta = \theta_2) P(\theta = \theta_2) / P(y)}{P(y | \theta = \theta_1) P(\theta = \theta_1) / P(y)} \\ &= \frac{P(y | \theta = \theta_2) P(\theta = \theta_2)}{P(y | \theta = \theta_1) P(\theta = \theta_1)}. \end{aligned}$$

Therefore, even though we may not know  $P(\theta = \theta_1 | y)$  as it involves  $P(y)$  as the denominator, we can still compute the density ratio.

In addition, the Metropolis algorithm requires the use of a **proposal distribution**, which can be any symmetric distribution. Common choices are normal distribution or uniform distribution. For example, let's assume we will use a  $N(0, 0.1)$  proposal distribution, with 0.1 being the standard deviation.

The steps of a Metropolis algorithm are:

1. Randomly start from a certain point in the parameter space, and call that point  $\theta_0$
2. Randomly generate a sampled value from a  $N(\theta_0, 0.11)$  distribution. Call this proposed value  $\theta^{\text{prop}}$

3. Compute the density ratio  $[P(\theta = \theta^{\text{prop}} | y)]/[P(\theta = \theta_0 | y)]$
4. If the ratio is larger than 1, accept  $\theta^{\text{prop}}$  and include this value in the sample
5. If the ratio is smaller than 1, accept  $\theta^{\text{prop}}$  with probability equal to the density ratio.  
For example, if the ratio is 0.7, one first generates a simulated value,  $u$ , from a uniform distribution between 0 and 1 (i.e.,  $U(0, 1)$ ). If  $u$  is smaller than the ratio, accept  $\theta^{\text{prop}}$  and include it in the sample. Otherwise, reject the proposed value, and include  $\theta_0$  (again) in the sample
6. After accepting  $\theta^{\text{prop}}$  or  $\theta_0$  in the sample, denote the accepted value as  $\theta_0$ , and repeat steps 2 to 6.

Compared to the Monte Carlo method, which directly samples from a Beta distribution, the Metropolis algorithm does not require an R function to draw samples from the target distribution. The cost, however, is that the sampling process is not as *efficient* because the sampled values are dependent. We'll discuss this point later after seeing an example of the algorithm.

---

## 15.4 Shiny App

To see a visual demonstration, you may run the shiny app I created by typing in R

```
shiny::runGitHub("metropolis_demo", "marklhc")
```

---

## 15.5 Example 1: Estimating the Number of People Taking the Metro

This example uses data from the [LA Barometer](#) survey conducted by the USC Dornsife Center for Economic and Social Research. Specifically, I'm interested in the proportion of participants who took the Metro in the previous year among first-generation immigrants in LA county. You can see a press release on the data at <https://dornsife.usc.edu/news/stories/3164/labarometer-mobility-in-los-angeles-survey/>

```
uas_dat <- read_dta(here("data", "uas219_psyc573.dta"))
uas_dat |>
  filter(immigrant_status == 0) |>
  count(tr002s2)
```

```
# A tibble: 3 x 2
  tr002s2          n
  <dbl+lbl>    <int>
1     0 [0 No]    26
2     1 [1 Yes]   111
3 NA(a)           192
```

So in 338 participants who are first-generation immigrants, 86 said they had used the Metro.

Let's use a weakly informative prior of Beta(1.5, 2), which has a weight of 1.5 prior data points, with a weak belief that less than half of the people had used the Metro.

Model:

$$\text{usemetro}_i \sim \text{Bern}(\theta)$$

Prior:

$$\theta \sim \text{Beta}(1.5, 2)$$

```
prior_a <- 1.5
prior_b <- 2
```

Based on conjugacy, we know the posterior is Beta(87.5, 254). For pedagogical purposes, we will instead use a Metropolis sampler, which only requires the ratio of prior  $\times$  likelihood for any two  $\theta$  values.

### 15.5.1 MCMC sampling

```
num_yes <- 86
num_obs <- 86 + 252
# Define a function to compute values proportional to p(y | th) * p(th)
prior_times_lik <- function(th) {
  # Return 0 if th is out of range
  if (th < 0 || th > 1) return(0)
  pth <- dbeta(th, shape1 = prior_a, shape2 = prior_b)
  py_given_th <- th ^ num_yes * (1 - th) ^ (num_obs - num_yes)
  pth * py_given_th
}
# Define a function for generating data from the proposal distribution
generate_proposal <- function(th, sd = 0.1) {
  rnorm(1, mean = th, sd = sd)
}
# Initialize the Metropolis algorithm
```

```

set.seed(2037) # set the seed for reproducibility
num_draws <- 1000
num_warmup <- num_draws / 2
th_all_draws <- rep(NA, num_draws)
# Step 1: starting value
th_all_draws[1] <- 0.1
# counter for tracking acceptance rate
num_accepted <- 0
for (s in seq_len(num_draws - 1)) {
  current_th <- th_all_draws[s]
  # Step 2: Generate proposal
  proposed_th <- generate_proposal(current_th)
  # Step 3: Compute acceptance probability
  prob_accept <- min(
    1,
    prior_times_lik(proposed_th) /
    prior_times_lik(current_th)
  )
  # Steps 4 & 5: determine whether to make the jump
  if (runif(1) < prob_accept) {
    th_all_draws[s + 1] <- proposed_th
    if (s + 1 >= num_warmup) {
      num_accepted <- num_accepted + 1
    }
  } else {
    th_all_draws[s + 1] <- current_th
  }
}

```

We can visualize the MCMC chain by plotting the iteration index on the x-axis and the sampled value on the y-axis:

```

ggplot(
  data.frame(th = th_all_draws, iter = seq_along(th_all_draws)),
  aes(x = iter, y = th)
) +
  geom_line()

```

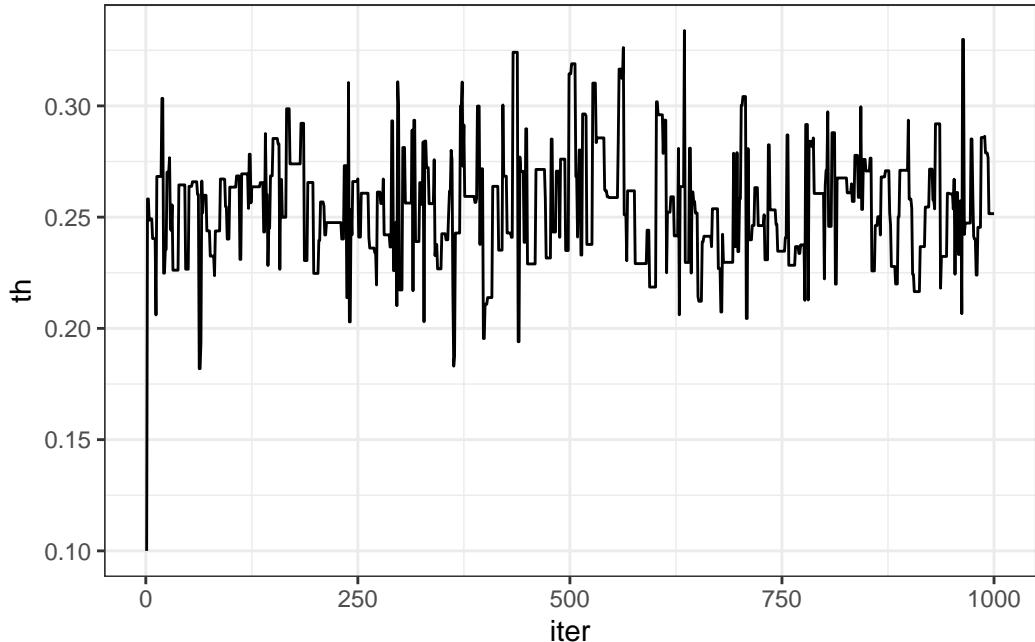


Figure 15.4: Trace plots of MCMC samples for the Metro example.

Each step in MCMC is called an *iteration*. The sampled values are *dependent*, meaning that the value at iteration  $s$  depends on the value at iteration  $s - 1$ . This is a major difference from functions that simulate independent random samples, like `rbeta()` or `rnorm()`. The resulting sampled values will form a *Markov chain*, meaning that each sampled value is correlated with the previous value (e.g., if  $\theta^{(s)}$  is large,  $\theta^{(s+1)}$  is also large).

As shown above, the chain starts at 0.10 then quickly moves to the region around 0.25, the area with high posterior density. It then oscillates around that for the remaining iterations.

### 15.5.2 More on Markov Chain

A Markov chain describes how a variable transitions from one “state” to another. The current state depends on the previous state. A well-behaved Markov Chain is said to be *ergodic*, which means that it is (see Hoff, 2009, chapter 10):

- *irreducible*: at any state  $\theta^{(s)}$ , it can go to any value  $\theta^*$  eventually. A reducible chain is one where some states cannot get to some other states; an example is when, at some point, the chain stays as a positive value forever and never gets back to the negative side.
- *aperiodic*: the chain does not have any periodic states. If it is a periodic chain, some values can only be visited every  $k$ th iteration.

- *recurrent*: after the chain visits a certain state  $\theta^*$ , if the chain runs long enough, it eventually returns to the same state  $\theta^*$ .

Under the above conditions, a Markov chain will converge to a *stationary distribution*. Thus, after a certain large amount of iterations, the draws from the chain can be considered a random (but correlated) sample of the stationary distribution. Moreover, one can prove that, with the Metropolis algorithm, the converging stationary distribution is the posterior distribution (see the discussion in Kruschke, 2015, chapter 7).

### 15.5.3 Warm-up/Burn-in

A Markov chain needs some iterations to get to the stationary distribution. Those iterations are usually called *warm-up* or *burn-in* (depending on the algorithm and the software) iterations and are usually discarded. In many software programs, the first half of the iterations are considered warm-ups, so even though we got 1,000 iterations, only 500 will be used:

```
num_warmup <- num_draws / 2
th_draws <- th_all_draws[-(1:num_warmup)]
```

### 15.5.4 Autocorrelation

The degree to which the value at iteration  $s$  is correlated with the value at  $s - 1$  (and at  $s - 2$ , etc) can be measured by the autocorrelation. For example, below shows the lag-1 correlation:

```
ggplot(data.frame(current = th_draws,
                  previous = lag(th_draws)),
       aes(x = previous, y = current)) +
  geom_point() +
  geom_smooth()

`geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```

Warning: Removed 1 rows containing non-finite values (`stat\_smooth()`).

Warning: Removed 1 rows containing missing values (`geom\_point()`).

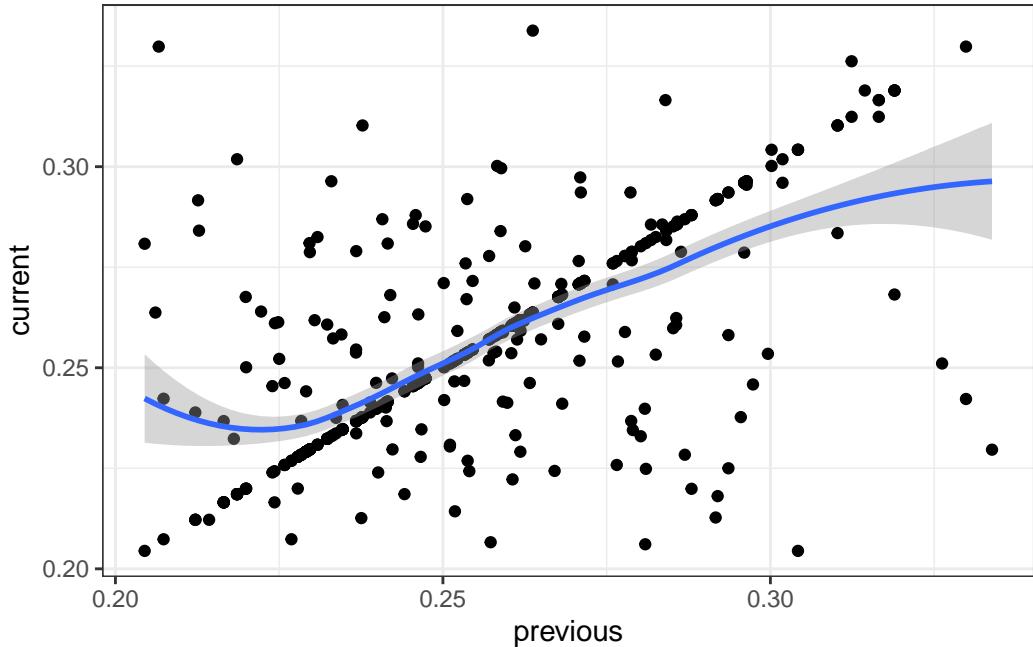


Figure 15.5: Scatterplot of observations at iteration s against observations at iteration s - 1, with MCMC samples.

Compared to samples from `rbeta()`

```
rbeta_draws <- rbeta(num_draws, shape1 = 87.5, shape2 = 254)
ggplot(data.frame(current = rbeta_draws,
                  previous = lag(rbeta_draws)),
       aes(x = previous, y = current)) +
  geom_point() +
  geom_smooth()

`geom_smooth()` using method = 'gam' and formula = 'y ~ s(x, bs = "cs")'

Warning: Removed 1 rows containing non-finite values (`stat_smooth()`).

Warning: Removed 1 rows containing missing values (`geom_point()`).
```

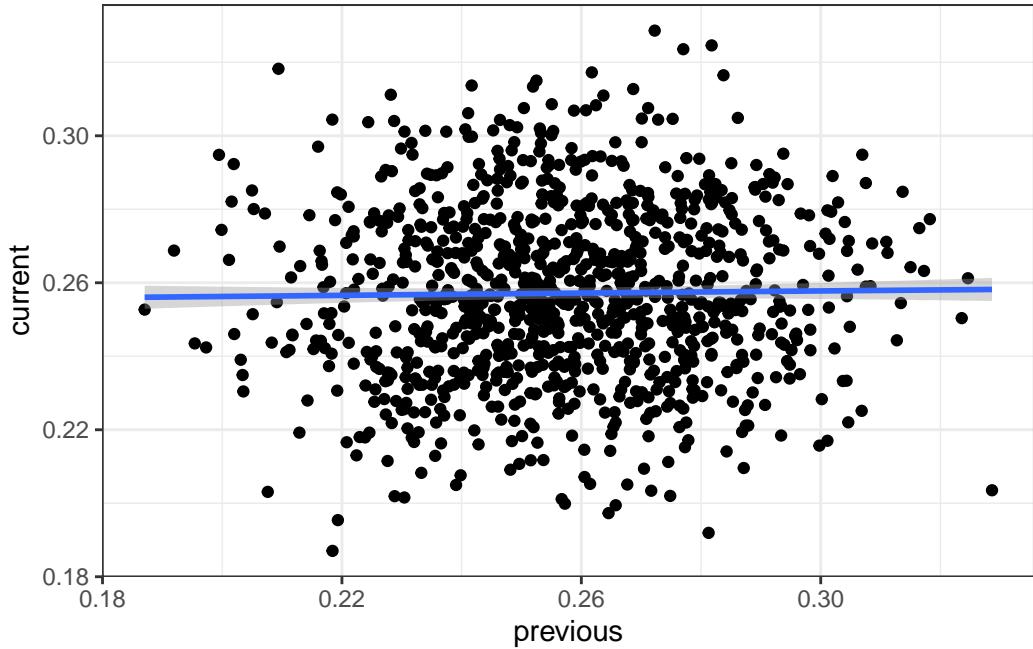


Figure 15.6: Scatterplot of observations at iteration  $s$  against observations at iteration  $s - 1$ , with independent Monte Carlo samples.

You can get the autocorrelation function plot (autocorrelation plot on the right):

```
mcmc_combo(data.frame(theta = th_draws),
            combo = c("trace", "acf")
)
```

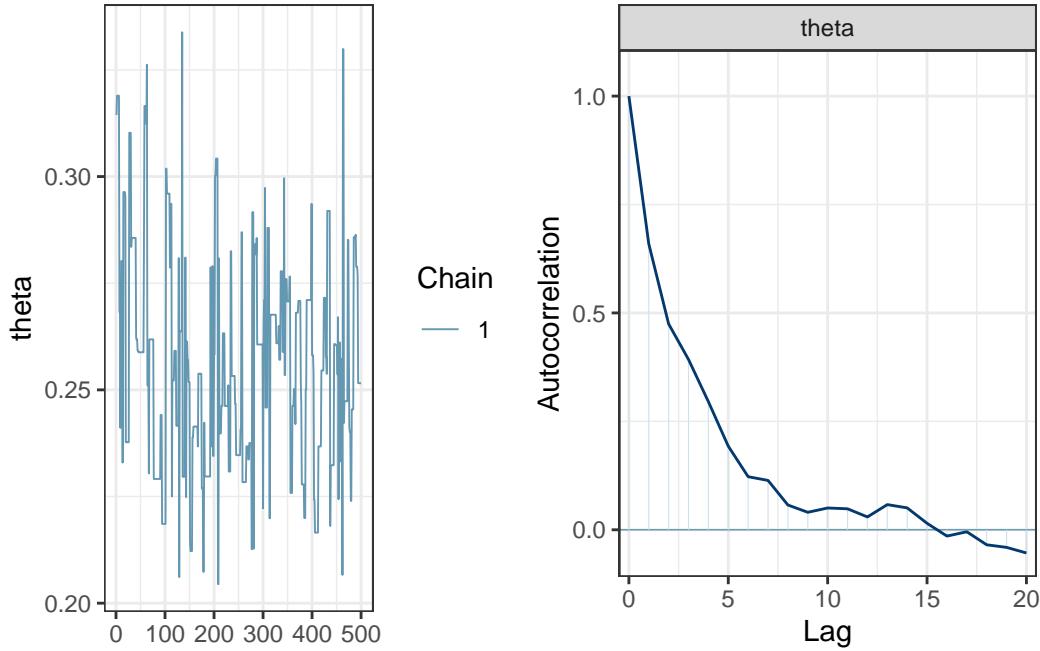


Figure 15.7: Trace plots and autocorrelation plots with MCMC samples.

which shows substantial autocorrelations until about nine iterations apart.

### 15.5.5 Acceptance Rate

When using the Metropolis algorithm, you want to monitor the acceptance rate and ensure it is within the optimal range. If you accept almost every time, it likely means that, in each iteration, the chain only jumps a tiny step (so that the acceptance ratio is close to 1 every time). As such, the chain will take many iterations to reach other regions of the stationary distribution, and consecutive draws are very strongly correlated. On the other hand, if the acceptance rate is very low, the chain gets stuck in the same location for many iterations before moving to a different state. For the basic Metropolis algorithm with one parameter, an optimal acceptance rate would be something between 40% to 50%.

 For Hamiltonian Monte Carlo to be discussed later, the optimal acceptance rate would be much higher, from 80% to 99%, or even higher.

```
# Acceptance rate
sum(num_accepted) / length(th_draws)
```

```
[1] 0.306
```

### 15.5.6 Effective sample size (ESS)

When iterations are dependent, each iteration contains overlapping information with the previous iterations. In other words, when one gets 500 dependent draws from the posterior, it only contains information equivalent to  $< 500$  independent draws. The ESS quantifies the actual amount of information, so a chain with  $\text{ESS} = n$  will contain roughly the same information as in  $n$  independent draws. In general, we want ESS to be at least 400 for the general purpose of summarizing the posterior. You can obtain ESS using the `posterior::ess_basic()` function:

```
ess_basic(th_draws)
```

```
[1] 84.42518
```

which is not sufficient for summarizing the posterior.

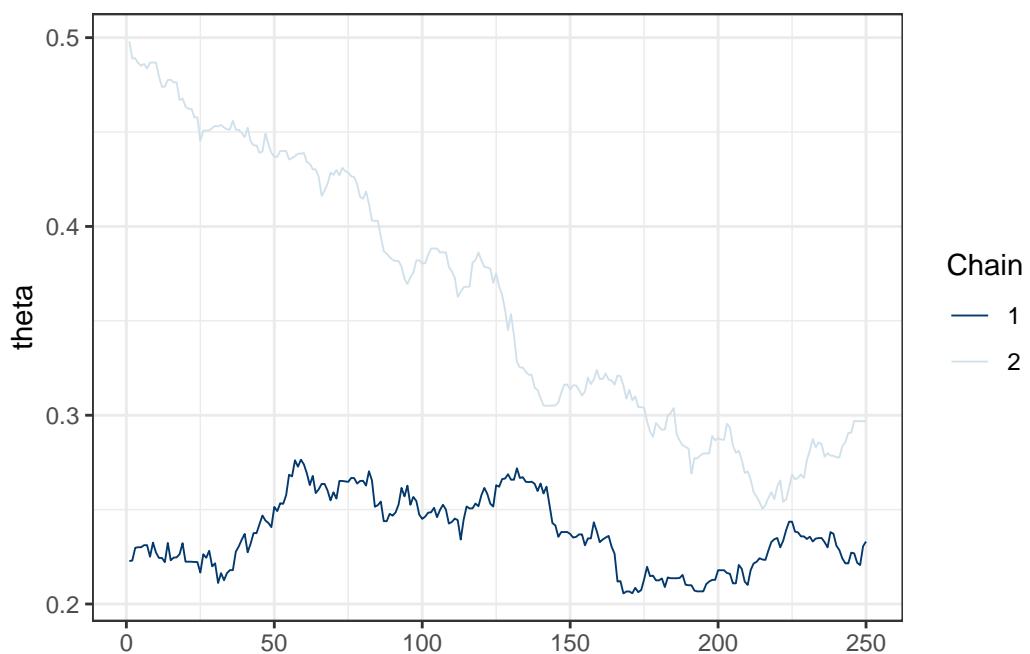
### 15.5.7 Multiple Chains

Because each state in a Markov chain depends on the previous states, the starting value(s) can influence the sampled values. Remember, in complex problems, one does not know how the posterior distributions would look. One solution to check the sensitivity to the starting value(s) is to use multiple chains, each with different starting values.

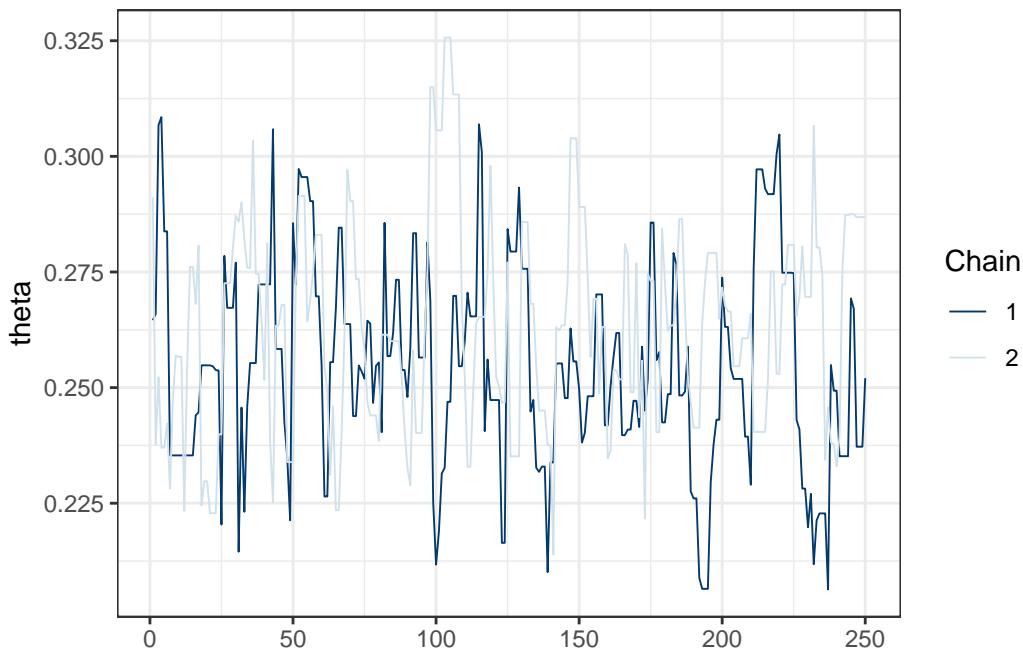
If multiple chains sample the same target distribution, they should be *mixing well*, meaning they cross each other in a trace plot.

Below are examples of two chains with good/poor mixing.

### 15.5.8 Poor Mixing



### 15.5.9 Better Mixing



### 15.5.10 Full R Code With More Iterations

```
num_yes <- 86
num_obs <- 86 + 252
# Define a function to compute values proportional to p(y | th) * p(th)
prior_times_lik <- function(th) {
    # Return 0 if th is out of range
    if (th < 0 || th > 1) return(0)
    pth <- dbeta(th, shape1 = prior_a, shape2 = prior_b)
    py_given_th <- th ^ num_yes * (1 - th) ^ (num_obs - num_yes)
    pth * py_given_th
}
# Define a function for generating data from the proposal distribution
generate_proposal <- function(th, sd = 0.1) {
    rnorm(1, mean = th, sd = sd)
}
# Initialize the Metropolis algorithm
set.seed(2037) # set the seed for reproducibility
num_chains <- 2
```

```

num_draws <- 10000
num_warmup <- num_draws / 2
th_all_draws <- matrix(NA, nrow = num_draws, ncol = num_chains)
th_all_draws[1, ] <- c(0.1, 0.9) # starting value
# counter for tracking acceptance rate
num_accepted <- rep(0, num_chains)
for (s in seq_len(num_draws - 1)) {
  for (j in seq_len(num_chains)) {
    current_th <- th_all_draws[s, j]
    # Generate proposal
    proposed_th <- generate_proposal(current_th, sd = 0.05)
    # Compute acceptance probability
    prob_accept <- min(
      1,
      prior_times_lik(proposed_th) /
      prior_times_lik(current_th)
    )
    # Determine whether to make the jump
    if (runif(1) < prob_accept) {
      th_all_draws[s + 1, j] <- proposed_th
      if (s + 1 >= num_warmup) {
        num_accepted[j] <- num_accepted[j] + 1
      }
    } else {
      th_all_draws[s + 1, j] <- current_th
    }
  }
}
# Save the draws after warm-up
th_draws <- th_all_draws[-(1:num_warmup), ]

```

As shown below, the MCMC draws approximate the posterior distribution reasonably well.

```

ggplot(data.frame(th = c(th_draws)), aes(x = th)) +
  # Plot histogram, with density on y axis
  geom_histogram(binwidth = 0.0025, aes(y = after_stat(density))) +
  # Overlay the theoretical Beta distribution
  stat_function(
    fun = dbeta,
    args = list(
      shape1 = 86 + prior_a,
      shape2 = 252 + prior_b

```

```

),
  col = "red"
) +
  labs(x = expression(theta))
# Acceptance rate
sum(num_accepted) / length(th_draws)

```

[1] 0.4889

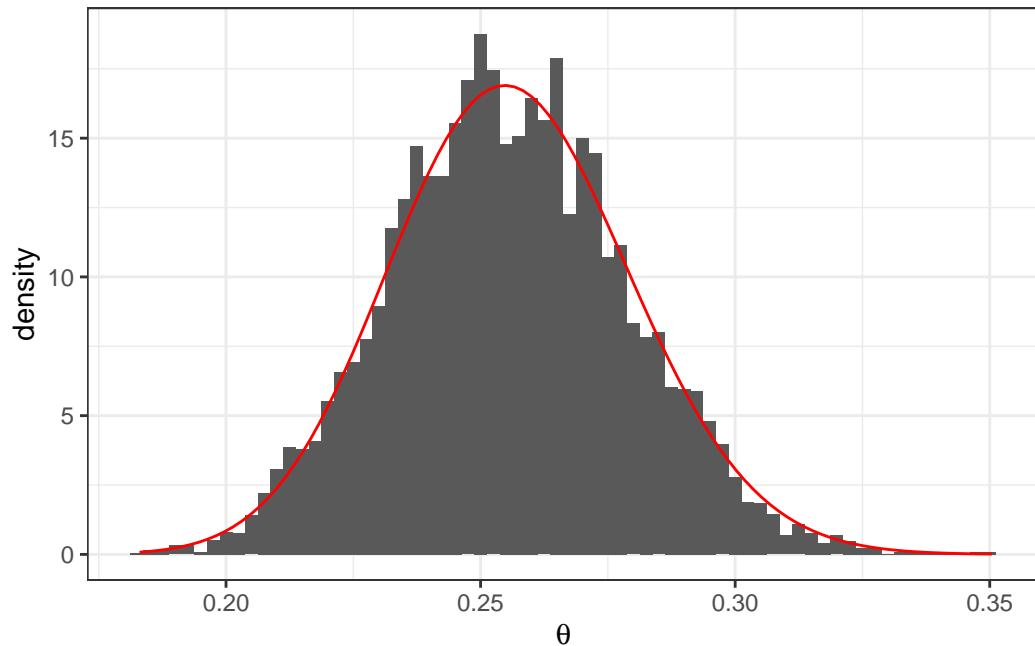


Figure 15.8: Comparing MCMC samples to the theoretical posterior distribution for the Metro example.

### 15.5.11 Convergence Check

The goal of checking for convergence of MCMC samples is to ensure that the draws are *representative* samples of the posterior distribution, and that they contain *sufficient information* to describe it. Convergence can be considered in two aspects:

- *Mixing*
- *Stationarity*

We'll see some tools in R for diagnosing convergence.

### 15.5.11.1 Trace Plot

Multiple chains have good *mixing* if they frequently cross each other. If two chains iterate in different regions of the posterior distribution and never cross each other, they don't mix well. You can check mixing using the trace plot. The below graph shows the two chains mixing well.

```
# Convert to `draws_array` object to use the following functions
th_draws_array <- draws_array(
  theta = th_draws,
  .nchains = num_chains
)
# Trace plot
mcmc_trace(th_draws_array)
```

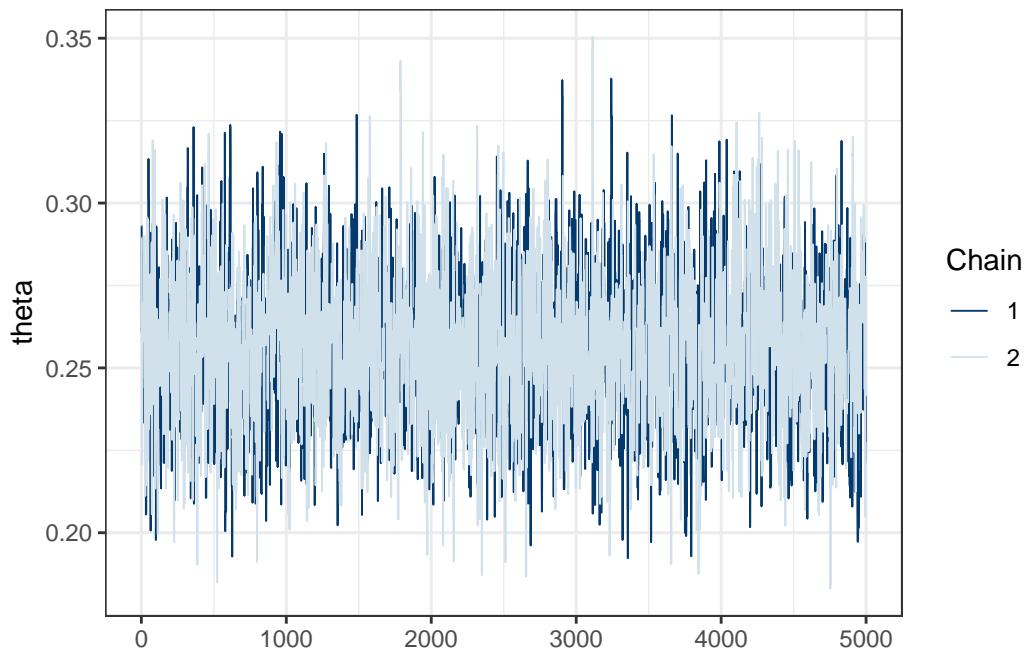


Figure 15.9: Trace plots of multiple chains.

### 15.5.11.2 Rank Histograms

Another useful tool is the rank histogram, recently proposed by [Vehtari et al. \(2021\)](#). The idea is first to convert the posterior values into ranks and then look at the distributions of the ranks across iterations. If the chains mix well and all explore the same target distribution, the

average rank in an interval of iterations should be similar for every chain. Therefore, the rank histogram should show something close to a uniform distribution, as shown below:

```
# Rank plot
mcmc_rank_hist(th_draws_array)
```

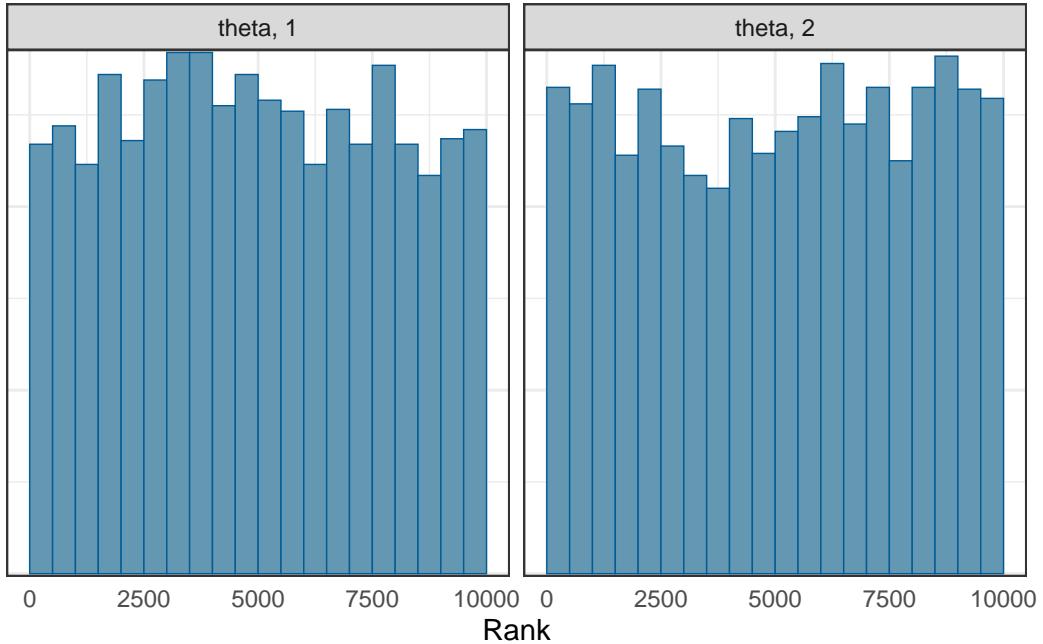


Figure 15.10: Rank histograms of two chains.

### 15.5.11.3 $\hat{R}$

A commonly used index in diagnosing convergence is  $\hat{R}$ , also called the potential scale reduction factor, proposed by Gelman & Rubin (1992) and later extended for multivariate distributions by Brooks & Gelman (1998). Vehtari et al. (2021) further improved it to account for unequal variances across chains using rank normalization and folding.  $\hat{R}$  measures the ratio of the total variability combining multiple chains to the within-chain variability. When the Markov chains converge, each chain is based on the same posterior distribution, and they should have the same variance. Therefore, if the chains converge, there should be no between-chain variability, so  $\hat{R}$  should be very close to 1.0.<sup>1</sup>

---

<sup>1</sup>Note that in Stan,  $\hat{R}$  is computed by splitting each chain into half. So if you have two chains,  $\hat{R}$  will be based on four groups.

While in older literature, the cutoff of  $\hat{R} < 1.1$  was usually used, **Vehtari et al. (2021)** recommended a safer criterion of  $\hat{R} < 1.01$ .

You can use the `summarize_draws()` function from the `posterior` package, which provides summary statistics of the posterior and some convergence diagnostics.

```
summarize_draws(th_draws_array)
```

```
# A tibble: 1 x 10
  variable   mean median     sd    mad     q5    q95 rhat ess_bulk ess_tail
  <chr>     <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>    <dbl>    <dbl>
1 theta      0.256  0.255  0.0233 0.0236 0.218  0.295  1.00    2368.   2465.
```

The numbers `ess_bulk` and `ess_tail` are two ways to compute ESS. `ess_bulk` says more about the center region of the posterior and is useful when assessing how accurate the mean of the posterior draws approximates the true posterior mean (or the posterior median). `ess_tail` is most useful for assessing how accurate the sample quantiles in the tail areas (e.g., 1st, 95th) approximate the true posterior quantiles. For example, the accuracy of the credible intervals would require a large `ess_tail`.

As recommended by Vehtari et al. (2021),

ESS should be at least 400 for  $\hat{R}$  to be useful in diagnosing convergence.

# 16 Gibbs Sampling

In the previous note, we learn about the Metropolis algorithm. While the algorithm is very general and easy to implement, it is inefficient because the effective sample size is only about 1/4 of the actual number of draws for a one-parameter model. The Metropolis algorithm usually gets stuck when there are multiple parameters. Therefore, until about 2010, many Bayesian analyses relied on a more efficient algorithm—the Gibbs sampler. The Gibbs sampler uses conjugate priors on the *conditional posterior* distributions to get proposal values with high acceptance probability (it's 100%). We'll also see the *normal* model with two parameters: mean and variance.

## 16.1 Data

The data set was from one of the studies reported in [a paper published in \*Psychological Science\* in 2014](#). The authors compared participants' performance on conceptual questions after taking notes either using laptops or notebooks ("longhand"). Let's import the data directly from the Open Science Framework (<https://osf.io/qrs5y/>):

```
# Use haven::read_sav() to import SPSS data
nt_dat <- read_sav("https://osf.io/qrs5y/download")
```

One outcome variable the authors studied is the number of words in participants' notes. Here are the distributions of the variable `wordcount` for the two conditions (0 = laptop, 1 = long-hand).

```
ggplot(nt_dat, aes(x = wordcount)) +
  geom_histogram(bins = 10) +
  facet_wrap(~ condition)
```

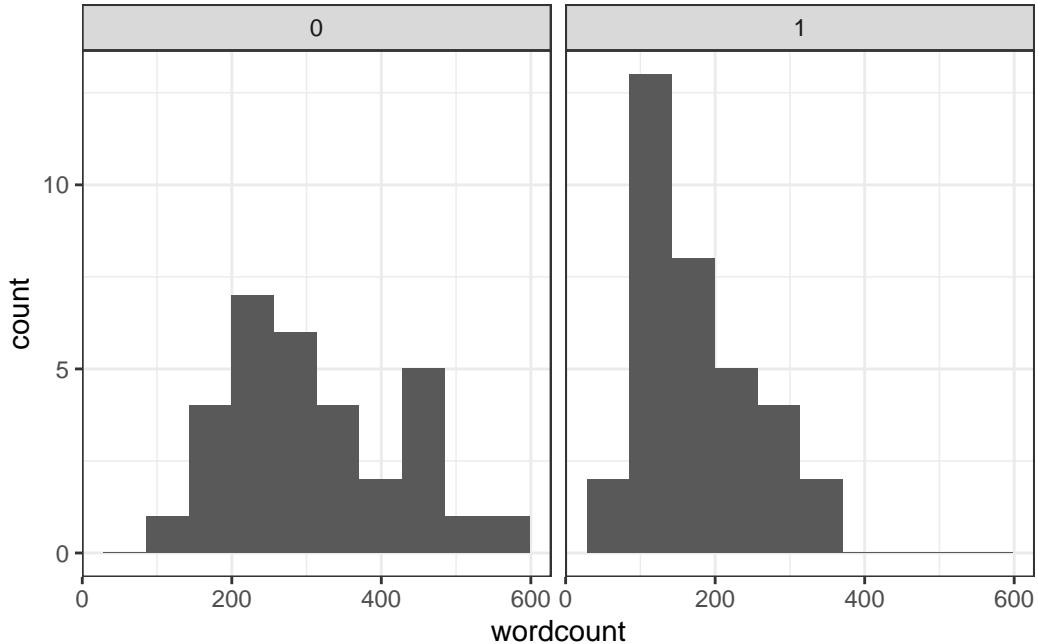


Figure 16.1: Number of words by experimental conditions.

The variable is somewhat skewed. We'll focus on the control group (`laptop`) first, so let's create a variable for that data. We'll also divide the count by 100.

- 💡 Often, in Bayesian analyses (and in frequentist ones as well), computer algorithms work best when the variables do not contain too many digits. In my experience, scaling variables to a range of -10 to 10 usually helps the computation.

```
(wc_laptop <- nt_dat$wordcount[nt_dat$condition == 0] / 100)
```

```
[1] 4.20 4.61 5.72 4.47 3.34 1.27 2.65 3.40 2.43 2.55 2.73 2.26 3.16 2.47 3.25
[16] 1.67 4.49 4.77 1.67 5.19 3.00 2.98 1.59 2.23 4.39 2.29 1.52 2.13 3.11 3.82
[31] 2.62
```

## 16.2 The Normal Model

While only an approximation, the normal distribution is a popular choice for modeling variables that are relatively symmetric and have more than a few discrete values. The normal distribution has two parameters; in some situations, there are advantages to choosing a specific

parameterization. With the Gibbs sampler, we will parameterize it using the mean parameter,  $\mu$ , and the variance parameter,  $\sigma^2$ . The model is

$$\text{wc\_laptop}_i \sim N(\mu, \sigma^2)$$

Note again that there are no subscripts to  $\mu$  and  $\sigma^2$ , indicating exchangeability assumption.

### 16.3 Conjugate (Actually “Semiconjugate”) Priors

It can be shown that when  $\sigma^2$  is known, a conjugate prior to  $\mu$  is a normal prior, meaning that the posterior is also normal. However, when  $\sigma^2$  is unknown, the prior distribution needs to be a joint distribution of two parameters.

What we can do is consider the conditional distribution of a parameter. So instead of drawing a posterior sample from the joint posterior,  $P(\mu, \sigma^2 | y)$ , we consider each parameter separately, by drawing from the conditional of  $P(\sigma^2 | \mu^{(s-1)}, y)$  first, and then from  $P(\mu | \sigma^2 = \sigma^{2(s)}, y)$  with  $\sigma^{2(s)}$  being the previous draw, and then continuing with  $P(\sigma^2 | \mu^{(s)}, y)$ . The advantage of doing so is that we can use a conjugate normal prior for  $\mu$  when conditioning on  $\sigma^2$ . We also have a conjugate prior for  $\sigma^2$  when conditioning on  $\mu$ , which will be discussed later. Here is how the algorithm works:

1. Set an initial value for  $\sigma^2(1)$
2. At iteration  $s$ , given sample  $\sigma^{2(s-1)}$ , sample  $\mu^{(s)}$  from the conditional posterior,  $P(\mu | \sigma^{2(s)}, y)$
3. Given  $\mu^{(s)}$ , sample  $\sigma^{2(s)}$  from the conditional posterior,  $P(\sigma^2 | \mu^{(s-1)}, y)$
4. Repeat steps 2 and 3

It can be shown that with this *Gibbs* algorithm, the posterior draws will be from the joint posterior of  $P(\mu, \sigma^2 | y)$ .

Below I provide more details on conjugacy. The actual detail is not the most important for this class because, in practice, the software will likely handle the computation. You may want to, however, pay attention to the suggested meanings of the hyperparameters:

- $\mu_0$ : Prior mean
- $\tau_0^2$ : Prior variance (i.e., uncertainty) of the mean
- $\nu_0$ : Prior sample size for the variance
- $\sigma_0^2$ : Prior expectation of the variance

### 16.3.1 For $\mu | \sigma^2$

A conjugate prior for  $\mu | \sigma^2$  is  $\mu \sim N(\mu_0, \tau_0^2)$ , which gives the posterior conditional

$$\mu | \sigma^2, y \sim N(\mu_n, \tau_n^2),$$

where

$$\begin{aligned}\tau_n^2 &= \left( \frac{1}{\tau_0^2} + \frac{n}{\sigma^2} \right)^{-1}, \\ \mu_n &= \tau_n^2 \left( \frac{\mu_0}{\tau_0^2} + \frac{n\bar{y}}{\sigma^2} \right),\end{aligned}$$

with  $n$  being the number of observations in the data and  $\bar{y}$  being the sample mean of the data. The hyperparameters,  $\mu_0$  and  $\tau_0^2$ , can be considered the prior mean and the prior variance of the mean, with a smaller prior variance indicating a stronger prior.

 Note that  $n$  (instead of  $N$ ) is used here for the sample size to avoid confusion with the normal distribution.

### 16.3.2 For $\sigma^2 | \mu$

A conjugate prior for  $\mu | \sigma^2$  is from the Inverse-Gamma family:  $\sigma^2 \sim \text{Inv-Gamma}(\nu_0/2, \nu_0\sigma_0^2/2)$ . You usually only hear about the Inverse-Gamma distribution in Gibbs sampling, mainly because of conjugacy. As the name suggested, the Inverse-Gamma distribution is the distribution of the *inverse* of a variable that follows a Gamma distribution. So we also write  $1/\sigma^2 \sim \text{Gamma}(\nu_0/2, \nu_0\sigma_0^2/2)$ .

 The Gamma distribution used here is also called a scaled  $\chi^2$  distribution by some authors, with  $\text{Gamma}(\nu_0/2, \nu_0\sigma_0^2/2)$  being the same as  $\chi^2(\nu_0, \sigma_0^2)$ .

The posterior is also Inverse-Gamma, with

$$1/\sigma^2 | \mu^2, y \sim \text{Gamma}(\nu_n/2, \nu_n\sigma_n^2[\mu]/2),$$

where

$$\begin{aligned}\nu_n &= \nu_0 + n \\ \sigma_n^2(\mu) &= \frac{1}{\nu_n} \left[ \nu_0\sigma_0^2 + (n-1)s_y^2 + \sum(\bar{y} - \mu)^2 \right],\end{aligned}$$

with  $s_y^2 = \sum(y_i - \bar{y})^2/(n-1)$  being the sample variance of  $y$ . The hyperparameters,  $\nu_0$  and  $\sigma_0^2$ , can be considered the prior degrees of freedom and prior expected value of variance;  $\nu_0$  can be roughly considered as the prior sample size.

## 16.4 Prior, Model, and Posterior

We will use some weakly informative priors, written in the following equations:

Model:

$$wc\_laptop_i \sim N(\mu, \sigma^2)$$

Prior:

$$\begin{aligned}\mu &\sim N(5, 10^2) \\ 1/\sigma^2 &\sim \text{Gamma}(1/2, [1][1]/2)\end{aligned}$$

The priors are very weak. We also assume that the priors are independent, which is commonly the case. Here are some simulated data from these priors:

```
set.seed(2259)
num_draws <- 100
mu <- rnorm(num_draws, mean = 5, sd = 10)
inv_sigma2 <- rgamma(num_draws,
                      shape = 1 / 2, rate = 1 / 2)
num_obs <- length(wc_laptop)
# Initialize an S by N matrix to store the simulated data
y_tilde <- matrix(NA,
                  nrow = num_draws,
                  ncol = num_obs)
for (s in seq_len(num_draws)) {
  mu_s <- mu[s]
  sigma2_s <- 1 / inv_sigma2[s]
  y_tilde[s, ] <- rnorm(num_obs, mean = mu_s, sd = sqrt(sigma2_s))
}
# Plot the simulated data based on priors
ppd_dens_overlay(y_tilde)
```

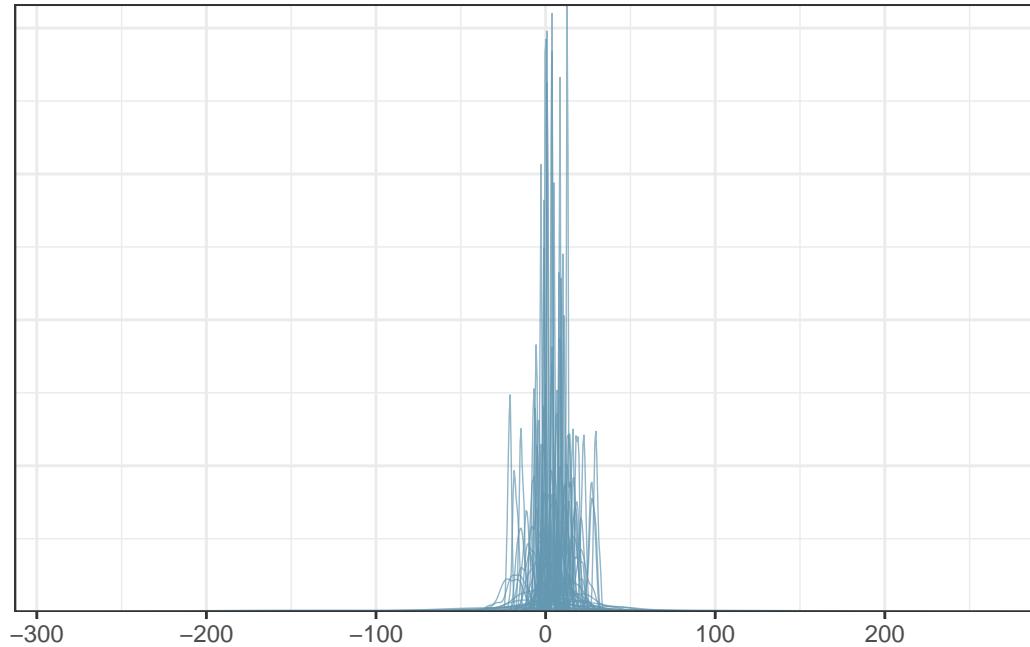


Figure 16.2: Prior predictive distributions.

The plot shows the types of data one can get. One can do better by avoiding the negative values if desired.

## 16.5 The Gibbs Sampler

The following is the full Gibbs sampler for the above normal model.

```
# Sufficient statistics from data
ybar <- mean(wc_laptop) # sample mean
s2y <- var(wc_laptop) # sample variance
n <- length(wc_laptop) # sample size
# Hyperparameters
mu_0 <- 5
sigma2_0 <- 1
tau2_0 <- 10^2
nu_0 <- 1
# Initialize the Gibbs sampler
set.seed(2120)
num_draws <- 10000
```

```

num_warmup <- num_draws / 2
num_chains <- 2
# Initialize a 3-D array (S x # chains x 2 parameters)
post_all_draws <- array(
  dim = c(num_draws, num_chains, 2),
  dimnames = list(NULL, NULL, c("mu", "sigma2"))
)
# Step 1: starting values for sigma2
post_all_draws[1, 1, "sigma2"] <- 1 # for chain 1
post_all_draws[1, 2, "sigma2"] <- 3 # for chain 2
for (s in seq_len(num_draws - 1)) {
  for (j in seq_len(num_chains)) {
    sigma2_s <- post_all_draws[s, j, "sigma2"]
    # Step 2: Sample mu from the conditional posterior
    tau2_n <- 1 / (1 / tau2_0 + n / sigma2_s)
    mu_n <- tau2_n * (mu_0 / tau2_0 + n * ybar / sigma2_s)
    mu_new <- rnorm(1, mean = mu_n, sd = sqrt(tau2_n))
    post_all_draws[s + 1, j, "mu"] <- mu_new
    # Step 3: Sample sigma2 from the conditional posterior
    nu_n <- nu_0 + n # you could put this line outside the loop
    sigma2_n <- 1 / nu_n *
      (nu_0 * sigma2_0 + (n - 1) * s2y + (ybar - mu_new)^2)
    sigma2_new <- 1 / rgamma(1,
      shape = nu_n / 2,
      rate = nu_n * sigma2_n / 2
    )
    post_all_draws[s + 1, j, "sigma2"] <- sigma2_new
  }
}
# Draws after warm-up
post_draws <- post_all_draws[-(1:num_warmup), , ]

```

## 16.6 Visualizing the Jumps

The plot below shows the jumps for 20 iterations in one chain, with the intermediate steps.

```

data.frame(
  mu = post_draws[c(1, rep(2:20, each = 2)), 1, "mu"],
  sigma2 = post_draws[c(rep(1:19, each = 2), 20), 1, "sigma2"]
) |>

```

```
ggplot(aes(x = mu, y = sigma2)) +
  geom_path() +
  geom_point() +
  labs(x = expression(mu), y = expression(sigma^2))
```

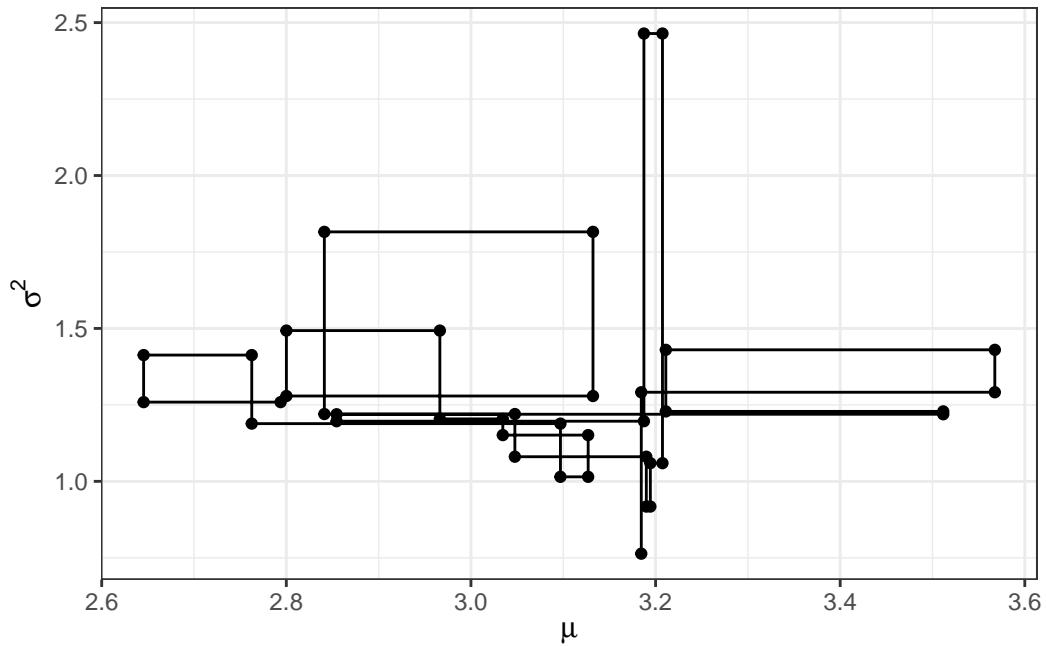
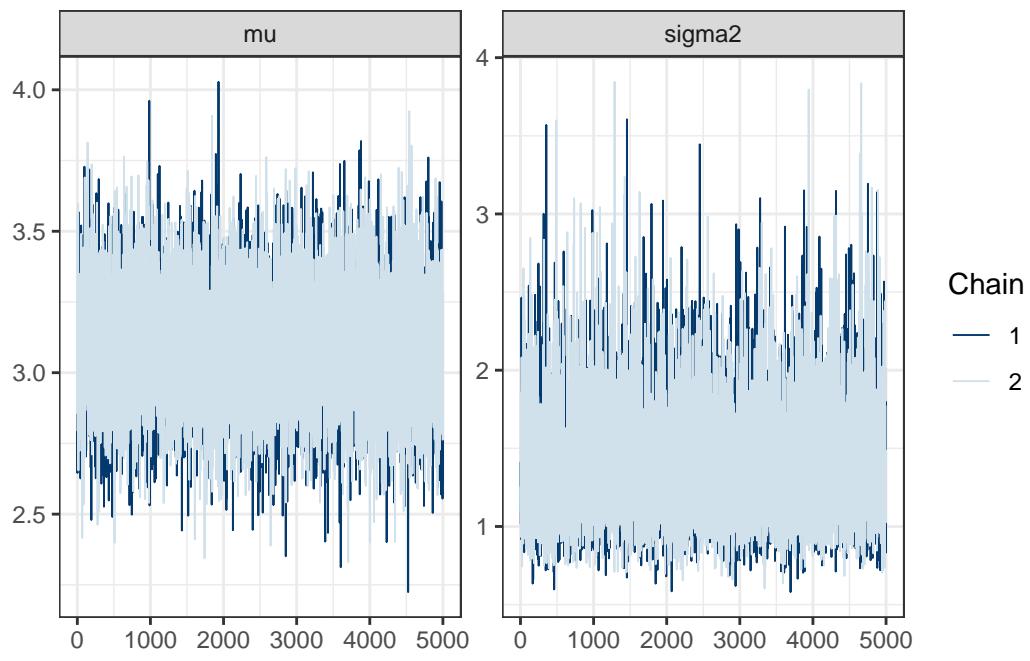


Figure 16.3: Jumps in the Gibbs sampler across iterations.

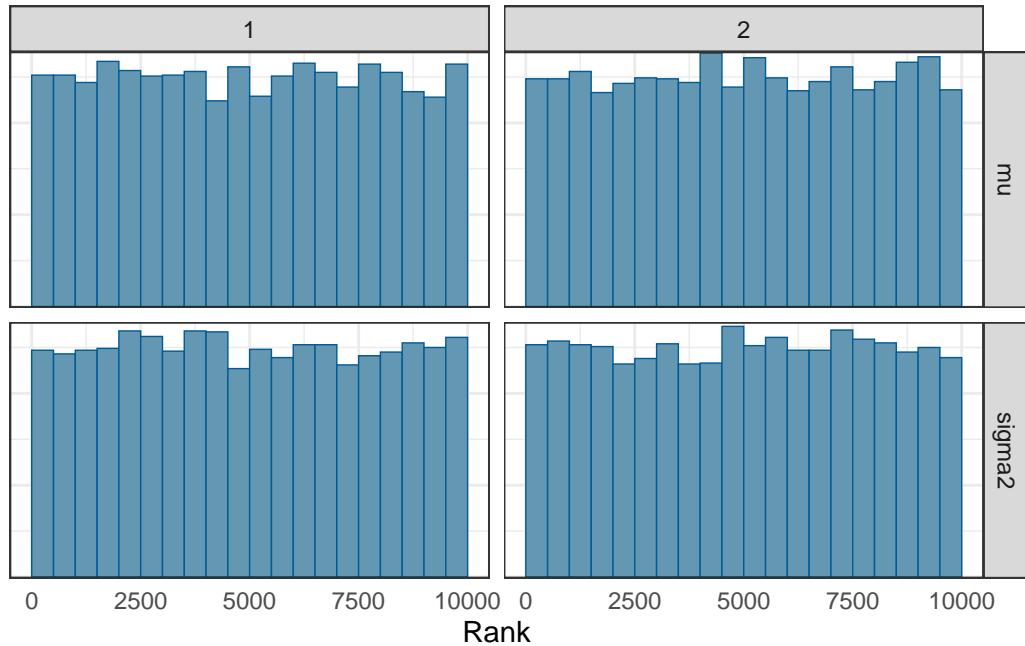
### 16.6.1 Convergence Check

```
# Convert to `draws_array` object to use the following functions
post_draws_array <- as_draws_array(post_draws)
# Trace plots
mcmc_trace(post_draws_array) # good mixing
# Rank histograms
mcmc_rank_hist(post_draws_array) # good mixing
```

```
# Summary (with rhat and ESS)
summarize_draws(post_draws_array) |>
  knitr::kable(digits = 2)
```



(a) Trace plots.



(b) Rank histograms.

Figure 16.4: Diagnostic plots for the Gibbs sampler.

variable	mean	median	sd	mad	q5	q95	rhat	ess_bulk	ess_tail
mu	3.1	3.10	0.21	0.21	2.75	3.45	1	9928.49	9935.87
sigma2	1.4	1.33	0.38	0.34	0.90	2.11	1	10188.76	10135.62

As can be seen, the ESS is very high, indeed much higher than that obtained with the Metropolis algorithm.

### 16.6.2 Visualizing the Marginal and Joint Posterior

```
mcmc_areas(post_draws_array) # marginal
```

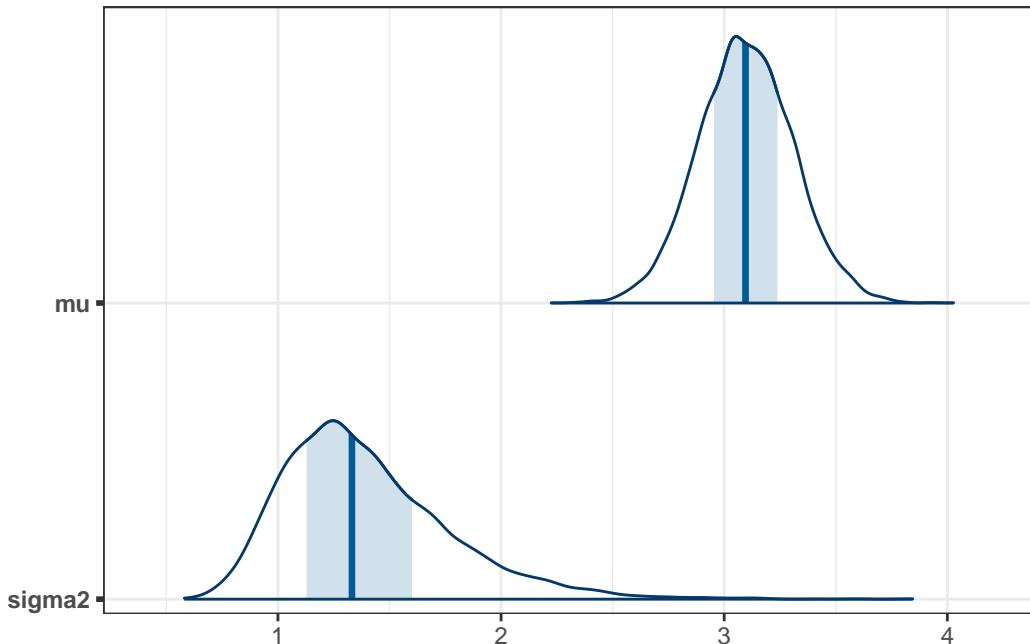


Figure 16.5: Marginal posterior distributions of model parameters.

```
mcmc_scatter(post_draws_array, # joint
              size = 1, alpha = 0.3) # make points smaller
```

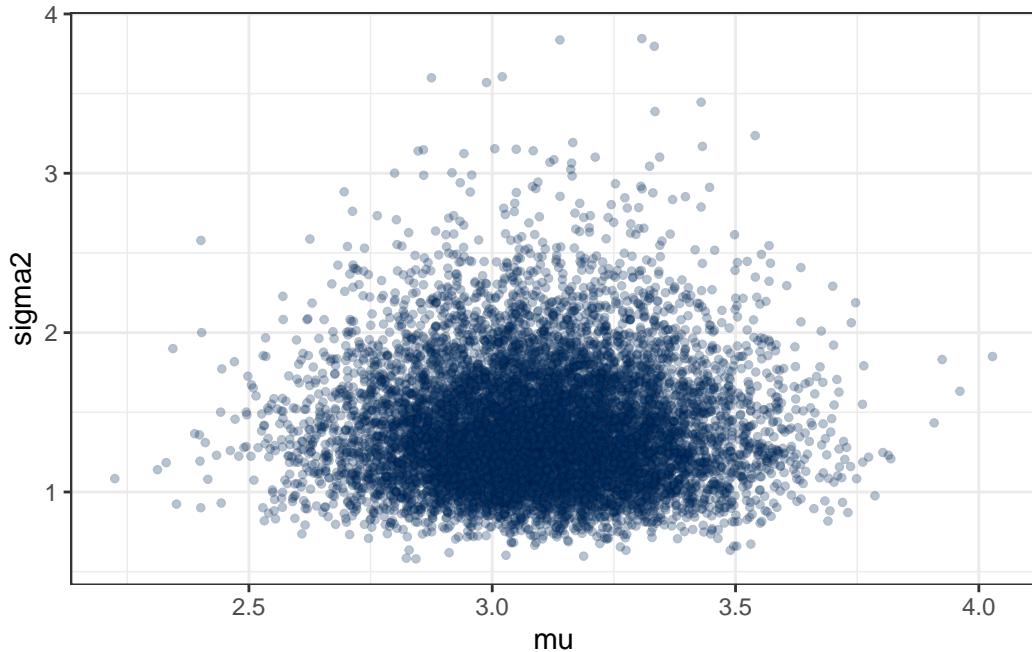


Figure 16.6: Joint posterior distributions of model parameters.

### 16.6.3 Posterior Predictive Check

We can check whether the simulated data based on the posterior look like the observed data.

```

num_draws <- 100
num_obs <- length(wc_laptop)
# Initialize an S by N matrix to store the simulated data
y_tilde <- matrix(NA,
                  nrow = num_draws,
                  ncol = num_obs)
for (s in seq_len(num_draws)) {
  mu_s <- post_draws[s, 1, "mu"]
  sigma2_s <- post_draws[s, 1, "sigma2"]
  y_tilde[s, ] <- rnorm(num_obs, mean = mu_s, sd = sqrt(sigma2_s))
}
# Plot the simulated data (in lighter lines)
# and the current data (in the darker line)
ppc_dens_overlay(wc_laptop, yrep = y_tilde)

```

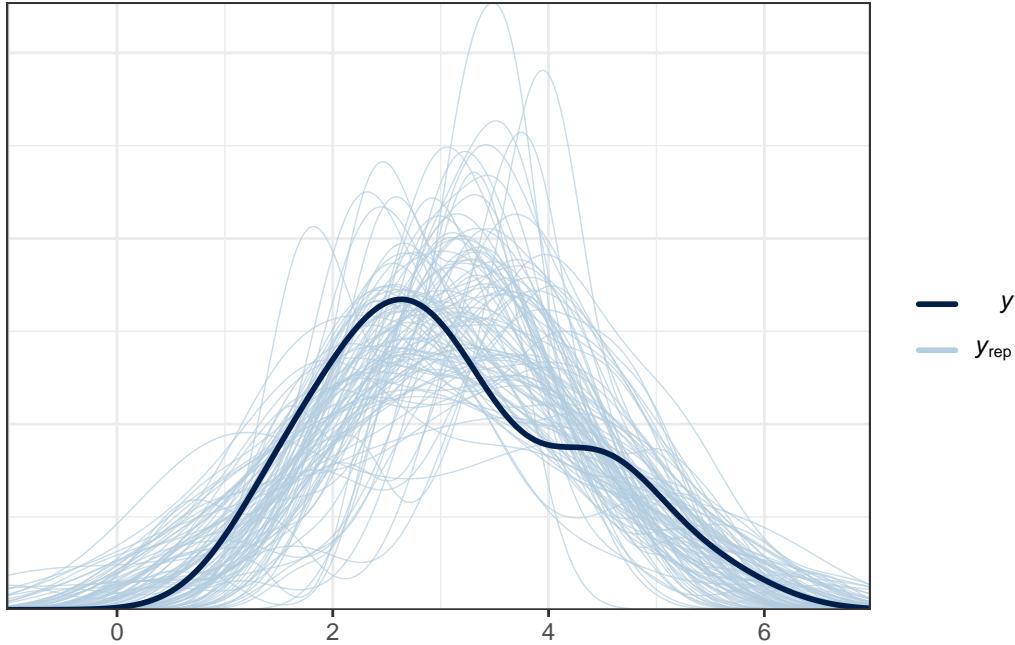


Figure 16.7: Posterior predictive check.

The simulated data are not too far off from the observed data. Just that the simulated data can also get negative values.

#### 16.6.4 Limitations of Gibbs Sampler

The Gibbs sampler has been popular due to its computational efficiency for many problems. However, there are situations in which it works less well, such as when some parameters are highly correlated in the posterior, in which case a Gibbs sampler may get stuck. Another limitation is that Gibbs samplers require the use of conjugate priors. On the one hand, sometimes researcher beliefs may be better expressed in distributions other than the conjugate family. On the other hand, some conjugate families, such as the inverse Gamma distribution, are hard to work with and may yield suboptimal results in small sample situations.

## 16.7 The Metropolis-Hastings Algorithm

The Metropolis-Hastings (MH) algorithm is a generalization of the Metropolis algorithm, where it allows for more than one parameter and can use a proposal distribution that is not symmetric. It is less efficient than the Gibbs sampler but can accommodate non-conjugate prior distributions. Note that the proposal distribution needs to have the same dimension as the

posterior so that the proposal is a vector in the parameter space. For example, we need a bivariate proposal density with  $\mu$  and  $\sigma$ .

Below is an example using a bivariate normal distribution for the proposal. While we can choose how correlated the variables are in the proposal, for simplicity, I just consider zero correlation and equal  $SD$  for both dimensions. I also parameterize the normal distribution by the mean,  $\mu$ , and the standard deviation (instead of the variance),  $\sigma$ . The priors are:

$$\begin{aligned}\mu &\sim N(5, 10^2) \\ \sigma &\sim N^+(0, 3)\end{aligned}$$

where  $N^+$  is a half-normal distribution because  $\sigma$  is non-negative. We'll revisit this for some later models.

Here is the code for the MH algorithm:

```
# Define a function to compute values proportional to p(y | th) * p(th)
prior_times_lik <- function(mu_sigma, y = wc_laptop) {
  mu <- mu_sigma[1]
  sigma <- mu_sigma[2]
  # Return 0 if sigma is out of range
  if (sigma < 0) return(0)
  # Joint prior = product of marginal priors
  pth <- dnorm(mu, mean = 5, sd = 10) *
    # half-normal is proportional to normal in [0, infinity)
    dnorm(sigma, sd = 3)
  # Likelihood
  py_given_th <- prod(dnorm(y, mean = mu, sd = sigma))
  pth * py_given_th
}

# Define a function for generating data from the proposal distribution
generate_proposal <- function(mu_sigma, sd = 0.1) {
  rnorm(length(mu_sigma), mean = mu_sigma, sd = sd)
}

# Initialize the Metropolis algorithm
set.seed(1051) # set the seed for reproducibility
num_draws <- 10000
num_warmup <- num_draws / 2
num_chains <- 2
# Initialize a 3-D array (S x # chains x 2 parameters)
post_all_draws <- array(
  dim = c(num_draws, num_chains, 2),
  dimnames = list(NULL, NULL, c("mu", "sigma"))
)
```

```

# Step 1: starting value
post_all_draws[1, 1, ] <- c(1, 1) # for chain 1
post_all_draws[1, 2, ] <- c(8, 3) # for chain 2
# counter for tracking acceptance rate
num_accepted <- rep(0, num_chains)
for (s in seq_len(num_draws - 1)) {
  for (j in seq_len(num_chains)) {
    current_par <- post_all_draws[s, j, ]
    # Generate proposal vector
    proposed_par <- generate_proposal(current_par, sd = 0.1)
    # Compute acceptance probability
    prob_accept <- min(
      1,
      prior_times_lik(proposed_par) /
      prior_times_lik(current_par)
    )
    # Determine whether to make the jump
    if (runif(1) < prob_accept) {
      post_all_draws[s + 1, j, ] <- proposed_par
      if (s + 1 >= num_warmup) {
        num_accepted[j] <- num_accepted[j] + 1
      }
    } else {
      post_all_draws[s + 1, j, ] <- current_par
    }
  }
}
# Draws after warm-up
post_draws <- post_all_draws[-(1:num_warmup), , ]
# Acceptance rate
sum(num_accepted) / length(post_draws)

```

[1] 0.3638

### 16.7.1 Convergence

```

# Convert to `draws_array` object to use the following functions
post_draws_array <- as_draws_array(post_draws)
# Trace plots
mcmc_trace(post_draws_array) # good mixing

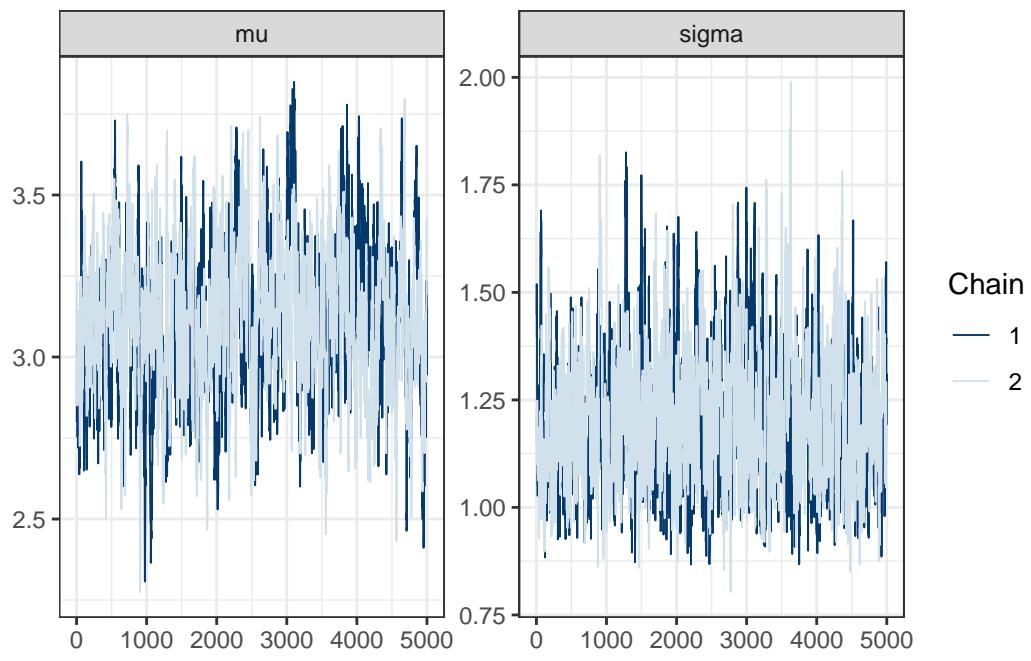
```

```
# Rank histograms
mcmc_rank_hist(post_draws_array) # good mixing
```

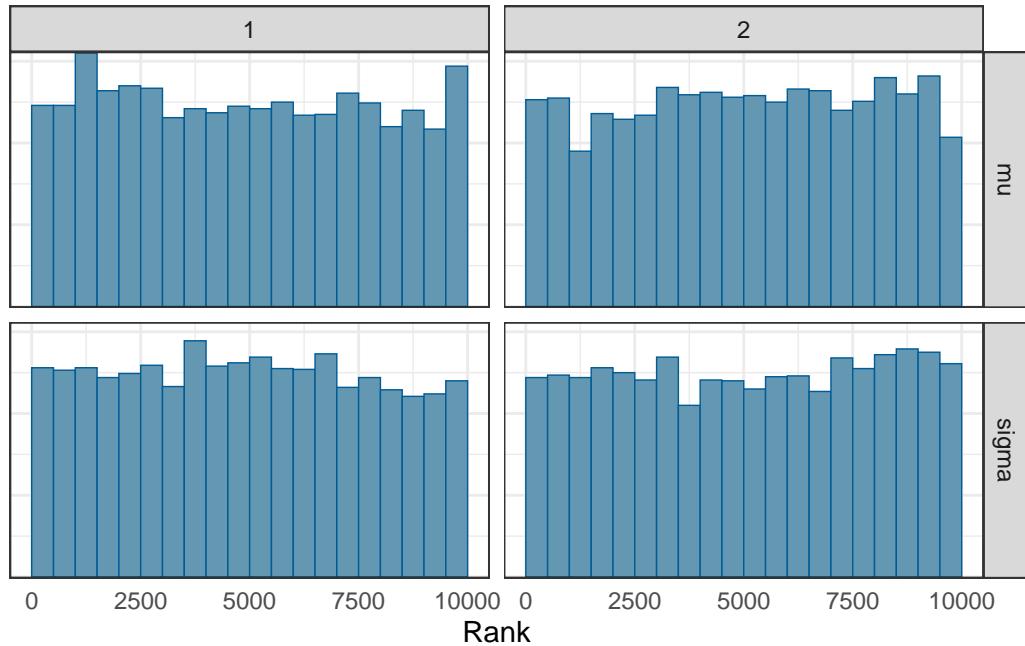
```
# Summary (with rhat and ESS)
summarize_draws(post_draws_array) |>
  knitr::kable(digits = 2)
```

variable	mean	median	sd	mad	q5	q95	rhat	ess_bulk	ess_tail
mu	3.11	3.11	0.23	0.23	2.72	3.49	1.01	304.66	426.00
sigma	1.20	1.19	0.15	0.15	0.98	1.48	1.00	694.50	733.81

As can be seen, ESS is much lower, so more iterations will be needed with MH.



(a) Trace plots.



(b) Rank histograms.

Figure 16.8: Diagnostic plots for the MH sampler.

# 17 Hamiltonian Monte Carlo

A more recent development in MCMC is the development of algorithms under the umbrella of *Hamiltonian Monte Carlo* or *hybrid Monte Carlo* (HMC). The algorithm is a bit complex, and my goal is to help you develop some intuition of the algorithm, hopefully, enough to help you diagnose the Markov chains.

1. It produces simulation samples that are much less correlated, meaning that if you get 10,000 samples from Metropolis and 10,000 samples from HMC, HMC generally provides a much higher effective sample size (ESS).
2. It does not require conjugate priors
3. When there is a problem in convergence, HMC tends to raise a clear red flag, meaning it goes really, really wrong.

So now, how does HMC work? First, consider a two-dimensional posterior below, which is the posterior of  $(\mu, \sigma^2)$  from a normal model.

```
ybar <- 3.096129
s2y <- 1.356451
n <- 31
# Hyperparameters
mu_0 <- 5
sigma2_0 <- 1
kappa_0 <- 1 / 10^2
nu_0 <- 1
# Joint log density
lp_mu_sigma <- function(mu, sigma2) {
  kappa_n <- kappa_0 + n
  mu_n <- (kappa_0 * mu_0 + n * ybar) / kappa_n
  nu_n <- nu_0 + n
  sigma2_n <- (nu_0 * sigma2_0 + (n - 1) * s2y +
    kappa_0 * n / kappa_n * (ybar - mu_0)^2) / nu_n
  log_dens <- -(mu - mu_n)^2 / 2 / sigma2 * kappa_n +
    (-nu_n / 2 - 1) * log(sigma2) +
    (-nu_n * sigma2_n / 2 / sigma2)
  log_dens[is.nan(log_dens)] <- -Inf
  log_dens}
```

```
}

num_gridpoints <- 51
mu_grid <- seq(2, to = 4, length.out = num_gridpoints)
sigma2_grid <- seq(0.5, to = 4.5, length.out = num_gridpoints)
grid_density <- exp(outer(mu_grid,
  Y = sigma2_grid,
  FUN = lp_mu_sigma
))
plot_ly(
  x = ~sigma2_grid, y = ~mu_grid, z = ~grid_density,
  type = "surface"
)
```

Figure 17.1: Joint posterior of  $(\mu, \sigma^2)$  from a normal model.

The HMC algorithm improves from the Metropolis algorithm by simulating smart proposal values. It does so by using the *gradient* of the logarithm of the posterior density. Let's first take the log of the joint density:

```
grid_logdensity <- outer(mu_grid,
  Y = sigma2_grid,
  FUN = lp_mu_sigma
)
plot_ly(
  x = ~sigma2_grid, y = ~mu_grid, z = ~grid_logdensity,
  type = "surface"
)
```

Figure 17.2: Gradient of  $(\mu, \sigma^2)$  from a normal model.

Now, flip it upside-down.

```
grid_minuslogdensity <- - outer(mu_grid,
  Y = sigma2_grid,
  FUN = lp_mu_sigma
)
plot_ly(
  x = ~sigma2_grid, y = ~mu_grid, z = ~grid_minuslogdensity,
  type = "surface"
)
```

Figure 17.3: "Flipped" gradient of  $(\mu, \sigma^2)$  from a normal model.

Imagine placing a marble on the above surface. With gravity, it is natural that the marble will have a tendency to move to the bottom of the surface. If you place the marble in a higher location, it will have a higher *potential energy* and tends to move faster towards the bottom, because that energy converts to a larger *kinetic energy*. If you place it close to the bottom, it will tend to move slower.

For simplicity, let's consider just one dimension with the following picture:

```
lp_sigma <- function(sigma2) {
  kappa_n <- kappa_0 + n
  nu_n <- nu_0 + n
  sigma2_n <- (nu_0 * sigma2_0 + (n - 1) * s2y +
    kappa_0 * n / kappa_n * (ybar - mu_0)^2) / nu_n
  log_dens <- (-nu_n / 2 - 1) * log(sigma2) +
    (-nu_n * sigma2_n / 2 / sigma2)
  log_dens[is.nan(log_dens)] <- -Inf
  log_dens
}
p2 <- ggplot(data = data.frame(x = c(0.5, 4.5)), aes(x = x)) +
  stat_function(fun = function(x) -lp_sigma(x)) +
  labs(x = expression(sigma^2),
       y = "-log(density)")
p2 +
  geom_point(
    x = 0.6, y = -lp_sigma(0.6),
    size = 2
  )
p2 +
  geom_point(
    x = 1.4, y = -lp_sigma(1.4),
    size = 2
  )
```

For the graph on the left, the marble (represented as a dot) has high potential energy and should quickly go down to the bottom; on the right, the marble has low potential energy and should move less fast.

- i If the sampler is near the bottom (i.e., the point with high posterior density), it tends to stay there.

But we don't want the marble to stay at the bottom without moving; otherwise, all our posterior draws will be just the posterior mode. In HMC, it moves the marble with a push, called *momentum*. The direction and the magnitude of the momentum will be randomly

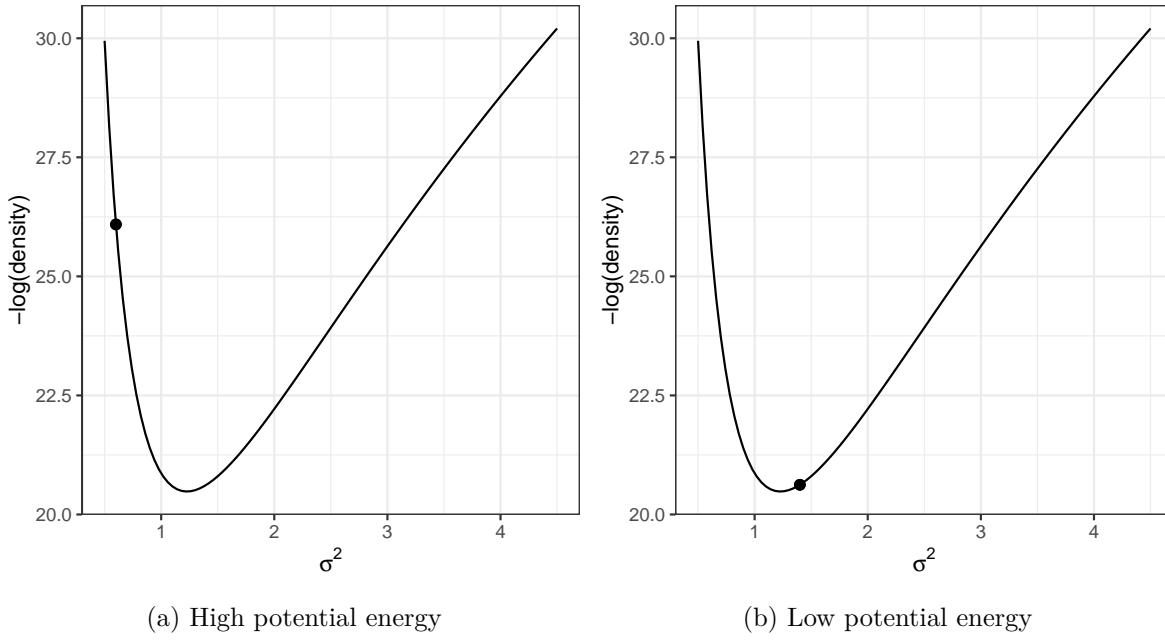


Figure 17.4: Marginal log posterior distribution of  $\sigma^2$ .

simulated, usually from a normal distribution. If the momentum is large, it should travel farther away; however, it also depends on the *gradient*, which is the slope in a one-dimension case.

- i** HMC generates a proposed value by simulating the motion of an object on a surface, based on an initial momentum with a random magnitude and direction, and locating the object after a fixed amount of time.

At the beginning of a trajectory, the marble has a certain amount of kinetic and potential energy, and the sum is the total amount of energy, called the *Hamiltonian*. Based on the conservation of energy, at any point in the motion, the Hamiltonian should remain constant.

## 17.1 Leapfrog Integrator

To simulate the marble's motion, one needs to solve a system of differential equations, which is not an easy task. A standard method is the so-called *leapfrog* integrator, which discretizes a unit of time into  $L$  steps. The following shows how it approximates the motion using  $L = 3$  and  $L = 10$  steps. The red dot is the target. As can be seen, larger  $L$  simulates the motion more accurately.

```

num_steps <- 3
leapfrog_df <- cbind(
  leapfrog_step(3, rho = -0.5, num_steps = num_steps),
  step = 0:num_steps
)
final_x <- leapfrog_step(3, rho = -0.5, num_steps = 1000)[1001, 1]
p2 + geom_point(
  data = leapfrog_df,
  aes(x = x, y = -lp_sigma(x)), col = "green"
) +
  geom_point(
    x = final_x, y = -lp_sigma(final_x),
    col = "red", size = 2, shape = 21
) +
  geom_path(
    data = leapfrog_df,
    aes(x = x, y = -lp_sigma(x)), col = "green"
  )
num_steps <- 10
leapfrog_df <- cbind(
  leapfrog_step(3, rho = -0.5, num_steps = num_steps),
  step = 0:num_steps
)
p2 + geom_point(
  data = leapfrog_df,
  aes(x = x, y = -lp_sigma(x)), col = "green"
) +
  geom_point(
    x = final_x, y = -lp_sigma(final_x),
    col = "red", size = 2, shape = 21
) +
  geom_path(
    data = leapfrog_df,
    aes(x = x, y = -lp_sigma(x)), col = "green"
  )

```

Suppose the simulated trajectory differs from the true trajectory by a certain threshold, like the one on the left. In that case, it is called a *divergent transition*, in which case the proposed value should not be trusted. In software like STAN, it will print out a warning about that. When the number of divergent transitions is large, one thing to do is increase  $L$ . If that does not help, it may indicate difficulty in a high-dimensional model, and reparameterization may be needed.

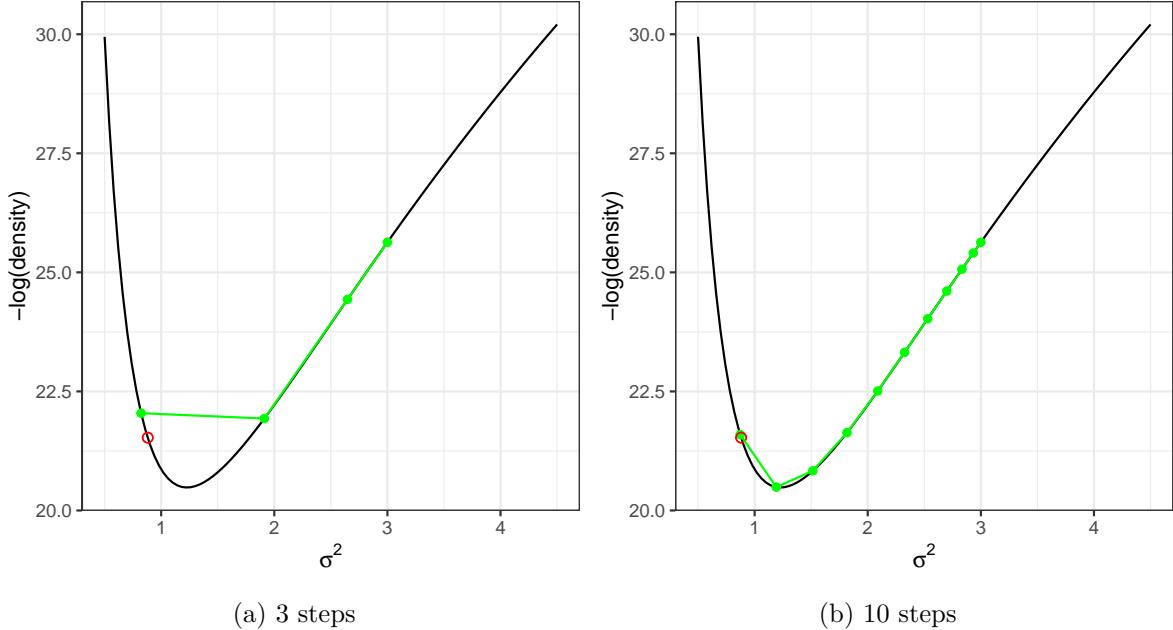


Figure 17.5: Simulated motion using the leapfrog integrator.

For more information on HMC, go to <https://mc-stan.org/docs/reference-manual/mcmc.html>. You can also find some sample R code at <http://www.stat.columbia.edu/~gelman/book/software.pdf>.

## 17.2 The No-U-Turn Sampler (NUTS)

Although HMC is more efficient by generating smarter proposal values, one performance bottleneck is that the motion sometimes takes a U-turn, like in some graphs above, in which the marble goes up (right) and then goes down (left), and eventually may stay in a similar location. The improvement of NUTS is that it uses a binary search tree that simulates both the forward and backward trajectory. The algorithm is complex, but to my understanding, it achieves two purposes: (a) finding the path that avoids a U-turn and (b) selecting an appropriate number of leapfrog steps,  $L$ . When each leapfrog step moves slowly, the search process may take a long time. With NUTS, one can control the maximum *depth* of the search tree.

See <https://mc-stan.org/docs/reference-manual/mcmc.html#hmc-algorithm-parameters> for more information on NUTS.

## **Part IX**

### **Week 12**

# 18 Generalized Linear Model

GLM (generalized linear model) is a general class of statistical models for predicting an outcome variable,  $Y$ . It accommodates  $Y$  in different types of measurement, such as continuous, counts, ordinal, or categorical. GLM is a generalization of the usual regression model that assumes linear associations and normally distributed errors.

Just to be careful, some scholars also use the abbreviation GLM to mean the *general* linear model, and the latter is actually the same as the linear model, which is a special case of the GLM we will discuss here. Here we discuss GLM (the generalized one) that is especially popular for modeling binary and count outcomes, but we will start with the linear regression case.

## 18.1 Overview of GLM

Under the GLM framework, with one predictor, we have models in the form

$$\begin{aligned} Y_i &\sim \text{Dist}(\mu_i, \tau) \\ g(\mu_i) &= \eta_i \\ \eta_i &= \beta_0 + \beta_1 X_i \end{aligned}$$

A GLM model has three components:

- Conditional distribution of  $Y$  (e.g.,  $\text{Dist} = \text{Normal}, \text{Poisson}, \text{Binomial}, \text{Bernoulli}$ )<sup>1</sup>
- Linear predictor  $\eta$ , which is a linear combination of the predictor
- Link function  $g(\cdot)$ , which maps  $\mu$  to  $\eta$

In a GLM, one selects distributions like  $\text{Dist} = \text{Normal}, \text{Poisson}, \text{Binomial}, \text{Bernoulli}$ , etc. The distribution has a mean parameter  $\mu_i$  and may have a dispersion parameter,  $\tau$ . An intermediate step in GLM is transforming  $\mu_i$  to  $\eta_i$ .  $\eta_i$  is called the *linear predictor*, which is the linear function of the predictors. In linear models, we directly model the conditional mean,

<sup>1</sup>Strictly speaking, GLM requires distributions that are in the *exponential family*, which will not include distributions like the  $t$  distribution, but here we will use GLM also to include models that use distributions similar to those in the exponential family, like the Student's  $t$  distribution.

$\mu_i$ , as the same as  $\eta_i$ . However, to allow for the possibility of  $\mu_i$  being a nonlinear function of the predictors, in GLM we transform  $\mu_i$  by applying a *link function*,  $g(\cdot)$ , so that, even though we  $\eta_i$  to be linear in the coefficients,  $\mu_i = g^{-1}(\eta_i)$  will be a nonlinear function of the coefficients as long as the link function is not linear. This step is needed to ensure that the predicted values will not be out of range.

Table 18.1 includes some commonly used GLMs.

Table 18.1: Commonly used GLMs

outcome type	Support	Distributions	Link
continuous	$[-\infty, \infty]$	Normal	Identity
count (fixed duration)	$\{0, 1, \dots\}$	Poisson	Log
count (known # of trials)	$\{0, 1, \dots, N\}$	Binomial	Logit
binary	$\{0, 1\}$	Bernoulli	Logit
ordinal	$\{0, 1, \dots, K\}$	categorical	Logit
nominal	$K$ -vector of $\{0, 1\}$	categorical	Logit
multinomial	$K$ -vector of $\{0, 1, \dots, K\}$	categorical	Logit

## 18.2 Poisson Regression

The Poisson GLM is used to model count outcomes. Count outcomes are non-negative discrete integers. Remember, with GLM, we are modeling the mean of the outcome,  $\mu$ . Therefore, we need to make sure  $\mu$  is non-negative, so we need a link function that can map  $\eta$  from the whole real line to non-negative numbers; by far, the most commonly used link function is the logarithmic transformation,  $g(\mu) = \log(\mu)$ .

Here's an example data set recording effect of anticonvulsant therapy in epilepsy. The outcome variable is the number of seizures in the two-week window prior to the last of the four visits.

First check the distribution of the counts in Figure 18.1.

```
epilepsy4 <- dplyr::filter(epilepsy, visit == 4)
epilepsy4$Trt <- factor(epilepsy4$Trt)
ggplot(epilepsy4, aes(x = count)) +
  geom_bar(width = 0.5)
ggplot(epilepsy4, aes(x = Trt, y = count)) +
  geom_boxplot() +
  geom_jitter(width = 0.05)
```

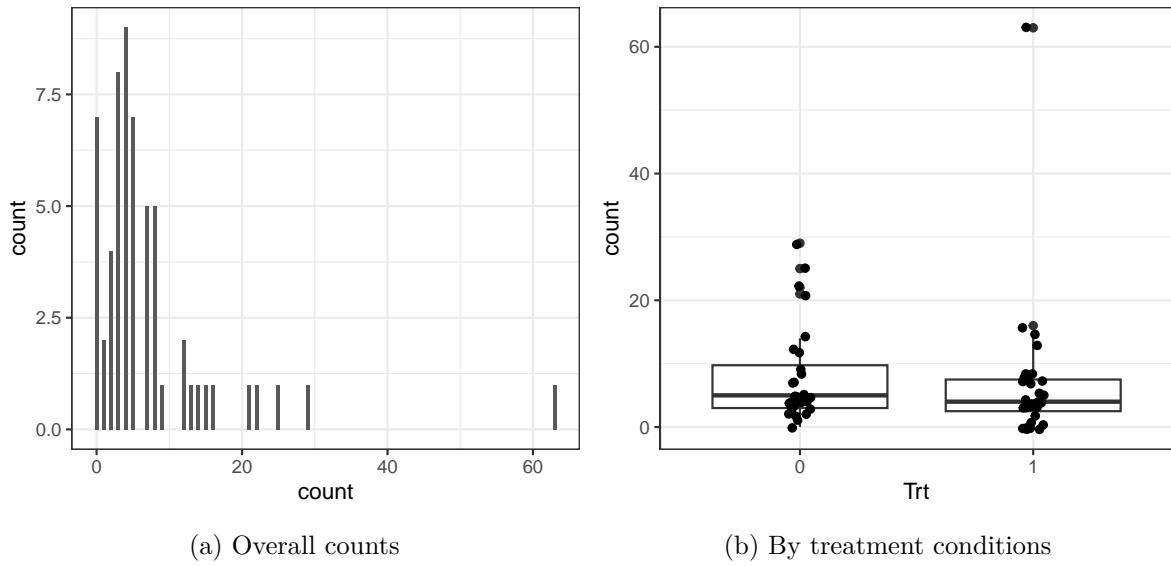


Figure 18.1: Distribution of number of seizures in the epilepsy data set

## 18.3 Model and Priors

Model:

$$\begin{aligned} \text{count}_i &\sim \text{Pois}(\mu_i) \\ \log(\mu_i) &= \eta_i \\ \eta_i &= \beta_0 + \beta_1 \text{Trt}_i \end{aligned}$$

Priors:

$$\begin{aligned} \beta_0 &\sim N(3, 2.5) \\ \beta_1 &\sim N(0, 1) \end{aligned}$$

Predicted seizure rate =  $\exp(\beta_0 + \beta_1) = \exp(\beta_0) \exp(\beta_1)$  for  $\text{Trt} = 1$ ;  $\exp(\beta_0)$  for  $\text{Trt} = 0$

$\beta_1$  = mean difference in **log** rate of seizure;  $\exp(\beta_1)$  = ratio in rate of seizure

### 18.3.1 MCMC Sampling With brms

```

m2 <- brm(count ~ Trt,
  data = epilepsy4,
  family = poisson(link = "log"),
  prior = c(
    prior(normal(1, 3), class = "Intercept"),
    prior(normal(0, 1), class = "b")
  ),
  seed = 31143,
  file = "10_m2"
)

```

### 18.3.2 Interpretations

Because of the nonlinear link function, one needs to be careful in interpreting the coefficients. With the log link, it is common to obtain the exponentiated coefficient. With the exponentiated coefficient, for every unit difference in  $X$ , the predicted rate of seizure occurrence is **multiplied** by  $\exp(\beta_1)$  times. Here is the usual step for obtaining the posterior distributions of the transformed parameters:

```

m2_summary <- as_draws(m2) |>
  mutate_variables(
    exp_beta0 = exp(b_Intercept),
    exp_beta1 = exp(b_Trт1)
  ) |>
  summarize_draws()

```

Loading required package: rstan

Loading required package: StanHeaders

rstan version 2.32.5 (Stan version 2.32.2)

For execution on a local, multicore CPU with excess RAM we recommend calling  
`options(mc.cores = parallel::detectCores())`.  
To avoid recompilation of unchanged Stan programs, we recommend calling  
`rstan_options(auto_write = TRUE)`  
For within-chain threading using `reduce\_sum()` or `map\_rect()` Stan functions,  
change `threads\_per\_chain` option:  
`rstan_options(threads_per_chain = 1)`

```

Attaching package: 'rstan'

The following objects are masked from 'package:posterior':

  ess_bulk, ess_tail

```

The following object is masked from 'package:tidy়':

extract

```
knitr::kable(m2_summary, digits = 2)
```

Table 18.2: Posterior summary of the Poisson regression model

variable	mean	median	sd	mad	q5	q95	rhat	ess_bulk	ess_tail
b_Intercept	2.07	2.07	0.07	0.07	1.96	2.18	1	3472.02	2643.16
b_Trt1	-0.17	-0.17	0.10	0.10	-0.33	-0.01	1	3139.53	2637.42
Intercept	1.98	1.98	0.05	0.05	1.91	2.06	1	2930.65	2699.11
lprior	-3.01	-3.00	0.02	0.01	-3.04	-2.99	1	3326.32	2899.07
lp__	-334.41	-334.12	0.99	0.70	-336.27	-333.48	1	1838.59	2379.17
exp_beta0	7.95	7.93	0.52	0.53	7.13	8.84	1	3472.02	2643.16
exp_beta1	0.85	0.85	0.08	0.08	0.72	0.99	1	3139.53	2637.42

So here is a paragraph for the example:

 The model predicts that the mean seizure rate in two weeks for the control condition is 8, 90% CI [7.1, 8.8]; the exponentiated coefficient for the treatment indicator is 0.85, 90% CI [0.72, 0.99], meaning that on average, the treatment reduces seizure rate by 0.94% to 27.83%.

### 18.3.3 Rootogram

There is also a useful graphical tool, the rootogram (Figure 18.2), for diagnosing count models.

```
pp_check(m2, type = "rootogram", style = "hanging")
```

Using all posterior draws for ppc type 'rootogram' by default.

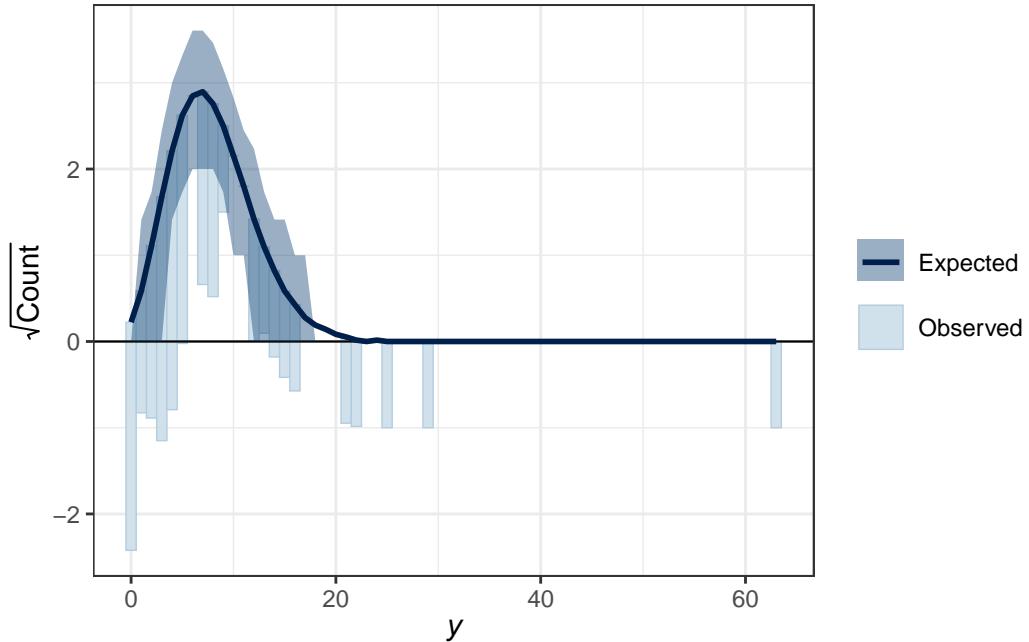


Figure 18.2: Rootogram for the Poisson regression model

If the fit is good, the bars should be close to touching the horizontal x-axis. Thus, the fit was not good in the above figure. Models that accommodate overdispersion and excessive zeros may be needed.

## 18.4 Negative Binomial Model

The negative binomial distribution is usually used to handle overdispersion in count data. The idea is that even with the same predictor value, each individual has a different rate of occurrence for an event, which is highly plausible for the epilepsy data. This class will not get into the details of the negative binomial, but you can check out [this paper](#) or [this book](#).

```
m2_nb <- brm(count ~ Trt,
  data = epilepsy4,
  family = negbinomial(link = "log"),
  prior = c(
    prior(normal(1, 3), class = "Intercept"),
    prior(normal(0, 1), class = "b")
  ),
  seed = 31143,
```

```

    file = "10_m2_nb"
)

pp_check(m2_nb, type = "rootogram", style = "hanging")

```

Using all posterior draws for ppc type 'rootogram' by default.

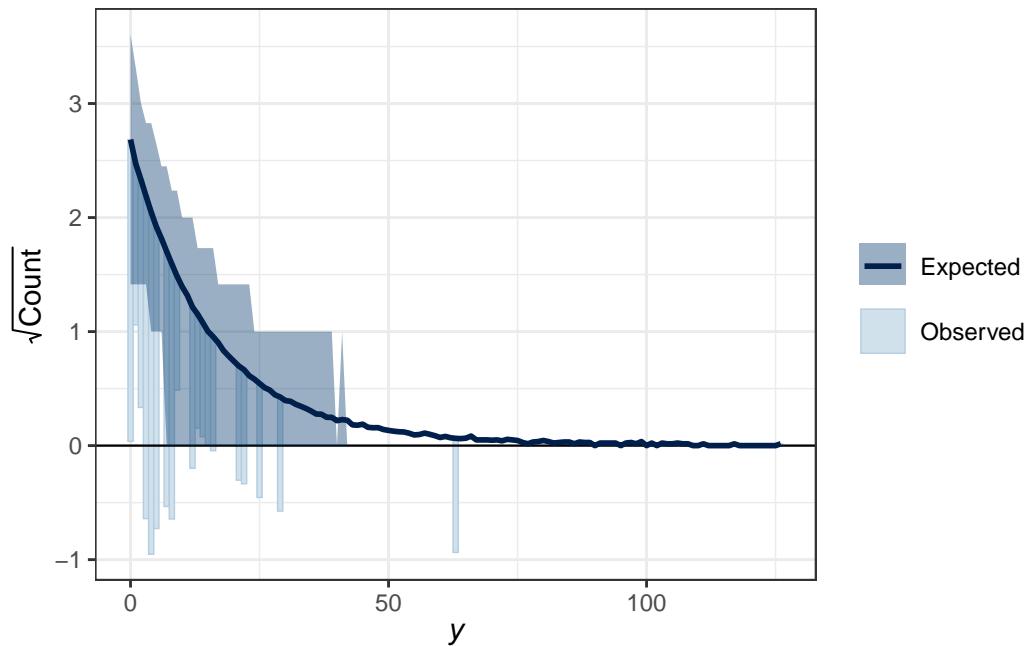


Figure 18.3: Rootogram for the Negative binomial model

## 18.5 Binary Logistic Regression

We will use an example from this paper: <https://journals.sagepub.com/doi/abs/10.1177/0956797616645672>, which examines how common it was for researchers to report marginal  $p$  values (i.e.,  $.05 < p \leq .10$ ). The outcome is whether a study reported one or more marginal  $p$  values (1 = Yes, 0 = No). The researchers also categorized the studies into three subfields (Cognitive Psychology, Developmental Psychology, Social Psychology), and there were a total of 1,535 studies examined from 1970 to 2010.

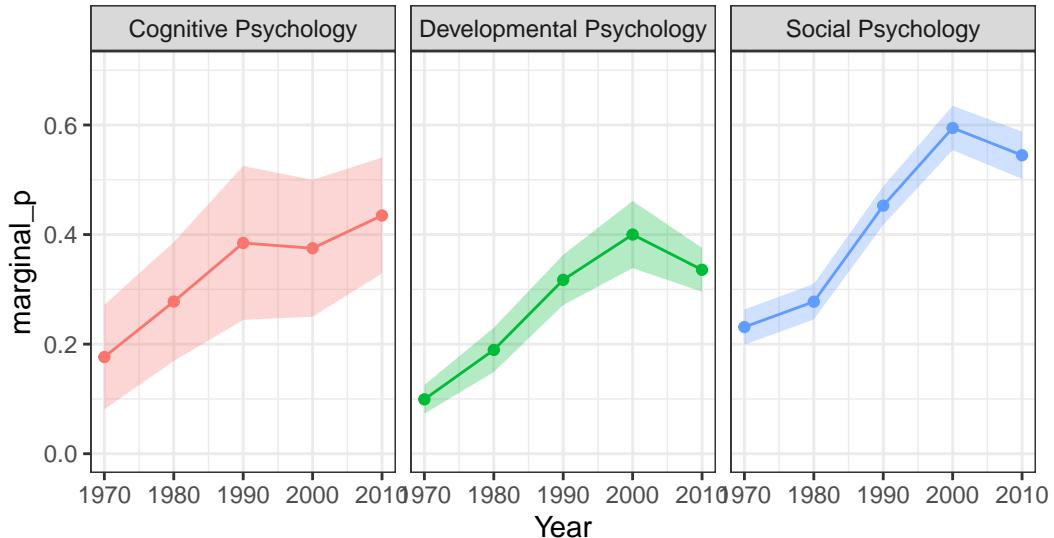
```

# readxl package can be used to import excel files
datfile <- here("data", "marginalp.xlsx")
if (!file.exists(datfile)) {
  dir.create(here("data"))
  download.file("https://osf.io/r4njf/download",
    destfile = datfile
  )
}
marginalp <- readxl::read_excel(datfile)
# Recode `Field` into a factor
marginalp <- marginalp |>
  # Filter out studies without any experiments
  filter(`Number of Experiments` >= 1) |>
  mutate(Field = factor(Field,
    labels = c(
      "Cognitive Psychology",
      "Developmental Psychology",
      "Social Psychology"
    )
  )) |>
  # Rename the outcome
  rename(marginal_p = `Marginals Yes/No`)
# Proportion of marginal p in each subfield across Years
marginalp |>
  ggplot(aes(x = Year, y = marginal_p)) +
  stat_summary(aes(fill = Field), geom = "ribbon", alpha = 0.3) +
  stat_summary(aes(col = Field), geom = "line") +
  stat_summary(aes(col = Field), geom = "point") +
  coord_cartesian(ylim = c(0, 0.7)) +
  facet_wrap(~ Field) +
  theme(legend.position = "top")

```

No summary function supplied, defaulting to `mean\_se()`  
 No summary function supplied, defaulting to `mean\_se()`

Field ● Cognitive Psychology ● Developmental Psychology ● Social Psycholog



### 18.5.1 Centering

The issue of centering applies for most regression models, including GLMs. Because the predictor `Year` has values 1970, 1980, etc, if we use this variable as a predictor, the intercept  $\beta_0$  will be the predicted value of  $\eta$  when `Year` = 0, which is like 2,000 years before the data were collected. So instead, we usually want the value 0 to be something meaningful and/or a possible value in the data. So we will center the variable year by making 0 the year 1970. In addition, we will divide the values by 10 so that one unit in the new predictor means 10 years.

In other words, we will use a transformed predictor,  $\text{Year10} = (\text{Year} - 1970) / 10$ . So when `Year` = 1970, `Year10` = 10; when `Year` = 1990, `Year10` = 2.

```
marginalp <- marginalp |>
  mutate(Year10 = (Year - 1970) / 10)
# Check the recode
distinct(marginalp, Year, Year10)
```

```
# A tibble: 5 x 2
  Year Year10
  <dbl>   <dbl>
1 1980     1
```

```

2 1970      0
3 2000      3
4 1990      2
5 2010      4

```

For this example, I'll look at the subset for developmental psychology. You may want to try with the other subsets of the data for practice.

```

marginalp_dev <- filter(marginalp,
                         Field == "Developmental Psychology")
head(marginalp_dev)

```

```

# A tibble: 6 x 9
  Field          `Paper Title` Authors  Year Number of Experiments~1 marginal_p
  <fct>         <chr>       <chr>    <dbl>                <dbl>      <dbl>
1 Developmental P~ A New Look a~ Christ~  2010                  1          1
2 Developmental P~ A Tale to Tw~ daniel~  2010                  1          1
3 Developmental P~ Associations~ Rachel~  2010                  1          1
4 Developmental P~ Attachment~b~ kathy~r~ 2000                  1          1
5 Developmental P~ Effects of A~ Timoth~ 1990                  1          1
6 Developmental P~ Effortful co~ grazyn~  2000                  1          1
# i abbreviated name: 1: `Number of Experiments`
# i 3 more variables: `First or Only Marginal` <dbl>,
#   `Conventional Threshold` <dbl>, Year10 <dbl>

```

### 18.5.2 Model and Priors

As the name “logistic regression” suggested, the logistic function is used somewhere in the model. The logistic function, or the inverse logit function, is

$$\mu = g^{-1}(\eta) = \frac{\exp(\eta)}{1 + \exp(\eta)},$$

which is a transformation from  $\eta$  to  $\mu$ . The link function, which is a transformation from  $\mu$  to  $\eta$ , is the logit function,

$$\eta = g(\mu) = \log \frac{\mu}{1 - \mu},$$

which converts  $\mu$ , which is in probability metric, to  $\eta$ , which is in log odds metric (odds = probability / [1 - probability]).

Model:

$$\begin{aligned} \text{marginal\_p}_i &\sim \text{Bern}(\mu_i) \\ \log(\mu_i) &= \eta_i \\ \eta_i &= \beta_0 + \beta_1 \text{Year10}_i \end{aligned}$$

Priors:

$$\begin{aligned} \beta_0 &\sim t_4(0, 2.5) \\ \beta_1 &\sim t_4(0, 1) \end{aligned}$$

There has been previous literature on what choices of prior on the  $\beta$  coefficients for logistic regressions would be appropriate (see [this paper](#)).  $\beta$  coefficients in logistic regression can be relatively large, unlike in normal regression. Therefore, it's pointed out that a heavy tail distribution, like Cauchy and  $t$ , would be more appropriate. Recent discussions have settled on priors such as  $t$  distributions with a small degrees of freedom as a good balance between heavy tails and efficiency for MCMC sampling. I use  $t_4(4, 0, 2.5)$  for  $\beta_0$ , which puts more density to be in [-2.5, 2.5] on the log odds unit. This means, in the year 1970, our belief is that the probability of marginally significant results would be somewhere between 0.08 to 0.92, which seems to cover a reasonable range. For  $\beta_1$ , I use  $t_4(4, 0, 1)$ , which suggests that a 10 year difference like corresponds to a difference in log odds between -1 and 1. While it's hard to interpret difference in log odds, a quick rule of thumb is to divide  $\beta_1$  by 4, which roughly corresponds to the maximum difference in probability for unit difference in the predictor. So if  $\beta_1 = 1$ , the maximum difference in probability will be no more than 25 percentage points. So my prior says that it's unlikely that the probability of reporting marginal  $p$  values would increase or decrease by 25 percentage points every 10 years, which again is a weak prior. If you're uncertain, you can consider something weaker.

The priors are chosen to be weakly informative.

```
m4 <- brm(marginal_p ~ Year10,
  data = marginalp_dev,
  family = bernoulli(link = "logit"),
  prior = c(
    prior(student_t(4, 0, 1), class = "b"),
    prior(student_t(4, 0, 2.5), class = "Intercept")
  ),
  # Note: no sigma
  seed = 1340,
  file = "10_m4"
)
```

Note that in `brms` we used `family = bernoulli()`. In other R functions, such as `glm`, they do not distinguish between `bernoulli` and `Binomial` and only recognize `family = binomial()`, as a Bernoulli variable is a binomial variable with  $n = 1$ .

### 18.5.3 Interpreting the results

Any non-linear relationships will involve more work in interpretations, and the coefficients in logistic regressions are no exceptions.

#### 18.5.3.1 Posterior predictions

A good way to start is to plot the model-implied association on the original unit of the data. The `conditional_effects()` function in `brms` comes in very handy, and I recommend you always start with that:

```
plot(
  conditional_effects(m4, prob = .90),
  points = TRUE,
  point_args = list(height = 0.01, width = 0.05, alpha = 0.05),
  plot = FALSE
)[[1]] +
  scale_x_continuous(
    breaks = 0:4,
    labels = c("1970", "1980", "1990", "2000", "2010")
) +
  xlab("Year")
```

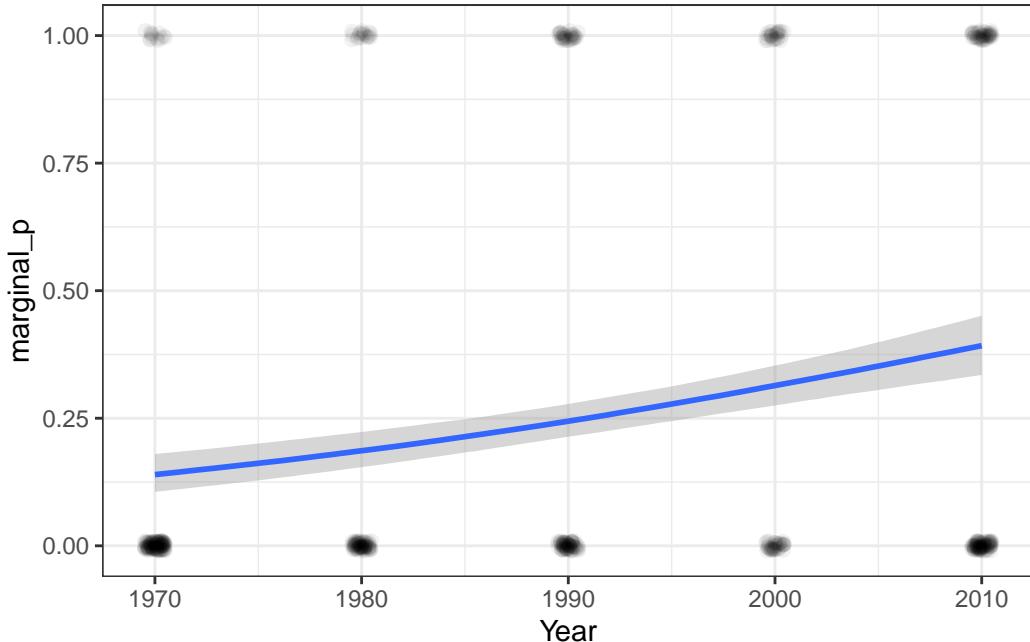


Figure 18.4: Predicted probabilities of marginal significant results over time.

As you can see, the logistic model implies a somewhat non-linear association between `Year10` and the outcome. From the graph, when `Year10` = 2, which is the Year 1990, the predicted probability of marginal significant results is about 25%, with a 90% credible interval of about 21 to 28%. We can get that using

```
posterior_epred(m4, newdata = list(Year10 = 2)) |>
  summarise_draws()

# A tibble: 1 x 10
  variable  mean   median      sd     mad     q5     q95    rhat  ess_bulk  ess_tail
  <chr>    <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>    <dbl>    <dbl>
1 ...1     0.245  0.244  0.0195  0.0191  0.214  0.278  1.00    2460.   2529.
```

### 18.5.3.2 Intercept

From the equation, when all predictors are zero, we have

$$\text{logit}(\mu_i) = \beta_0.$$

Therefore, the intercept is the log odds that a study reported a marginally significant  $p$  value when `Year10` = 0 (i.e., in 1970), which was estimated to be -1.8, 90% CI [-2.1, -1.5]. As log

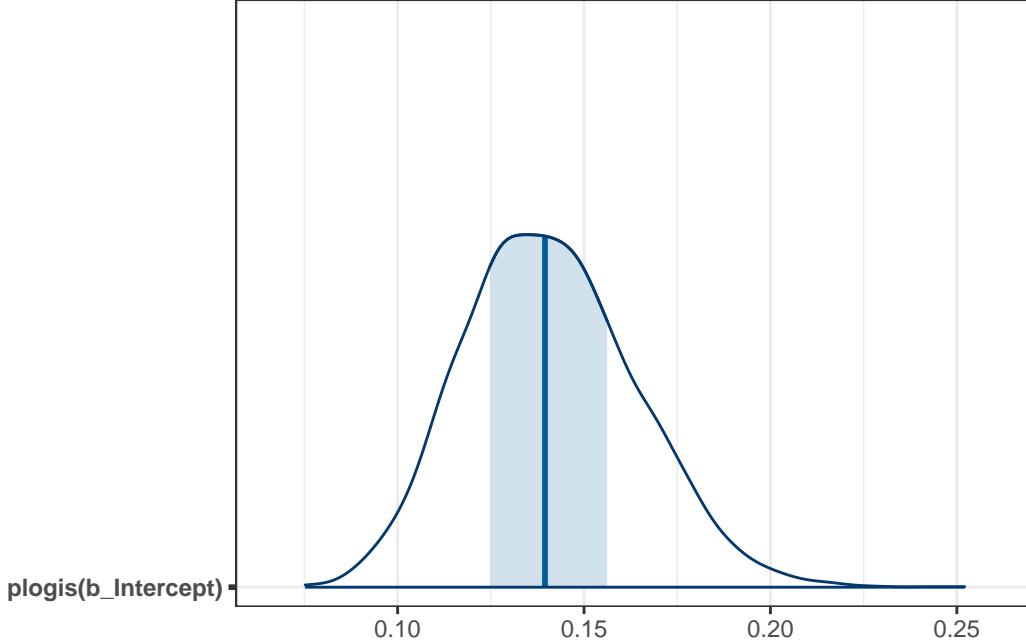
odds are not as intuitive as probability, it is common to instead interpret  $\hat{\mu} = \text{logistic}(\beta_0)$ , which is the conditional probability of being `marginal_p` = 1 in 1970. For Bayesian, that means obtaining the posterior distribution of  $\text{logistic}(\beta_0)$ , which can be done by

```
m4_draws <- as_draws(m4)
m4_draws <- m4_draws |>
  mutate_variables(logistic_beta0 = plogis(b_Intercept))
m4_draws |>
  subset(variable = "logistic_beta0") |>
  summary()

# A tibble: 1 x 10
  variable      mean   median     sd    mad    q5    q95  rhat ess_bulk ess_tail
  <chr>        <dbl>   <dbl>   <dbl>  <dbl>  <dbl>  <dbl> <dbl>    <dbl>    <dbl>
1 logistic_beta0 0.141   0.140  0.0230  0.0230  0.106  0.180  1.00    2100.   2030.
```

The `bayesplot` package allow you to plot transformed parameters quickly:

```
mcmc_areas(m4, pars = "b_Intercept",
            transformations = list("b_Intercept" = "plogis"),
            bw = "SJ")
```



### 18.5.3.3 Interpreting $\exp(\beta_1)$ as odds ratio

The slope,  $\beta_1$ , represents the difference in the predicted log odds between two observations with 1 unit difference in the predictor. For example, for two individuals with 1 unit difference in `Year10` (i.e., 10 years), we have

$$\text{logit}(\mu_{\text{marginal\_p}=1}) - \text{logit}(\mu_{\text{marginal\_p}=0}) = \beta_1.$$

Again, difference in log odds are hard to interpret, and so we will exponentiate to get

$$\frac{\text{odds}_{\text{marginal\_p}=1}}{\text{odds}_{\text{marginal\_p}=0}} = \exp(\beta_1).$$

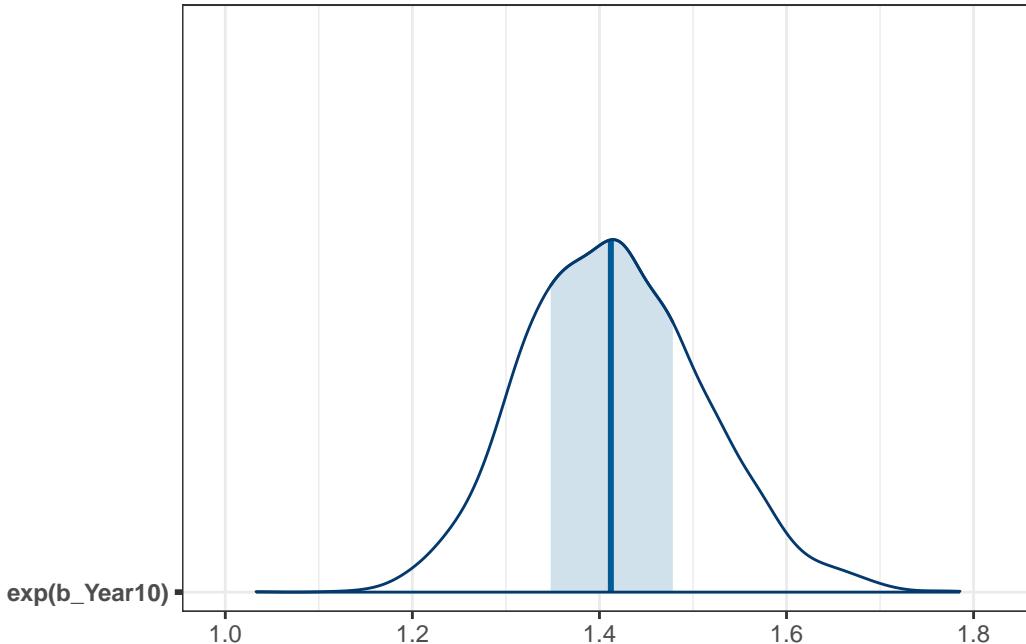
The fraction on the left hand side is the *odds ratio* of reporting a marginal  $p$  value associated with a one unit difference in `Year10` (i.e., 10 years). An odds of 1.0 means that the probability of success and failure is equal; an odds  $> 1$  means success is more likely than failures; and an odds  $< 1$  means success is less likely than failures. Again, for Bayesian, we can obtain the posterior distribution of  $\exp(\beta_1)$  by

```
m4_draws <- m4_draws |>
  mutate_variables(exp_beta1 = exp(b_Year10))
m4_draws |>
  subset(variable = "exp_beta1") |>
  summary()

# A tibble: 1 x 10
  variable   mean median    sd    mad    q5    q95   rhat ess_bulk ess_tail
  <chr>     <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>    <dbl>    <dbl>
1 exp_beta1 1.42   1.41  0.0954 0.0968  1.27   1.58   1.00    2688.    2698.
```

Using the posterior mean, we predict that the odds of reporting a marginal  $p$  value for a study that is 10 years later is multiplied by 1.4 times, 90% CI [1.3, 1.6]; in other words, every 10 years, the odds increased by 41.5852521%. Here's the posterior density plot for the odds ratio:

```
mcmc_areas(m4, pars = "b_Year10",
            transformations = list("b_Year10" = "exp"),
            bw = "SJ")
```



Odds ratio (OR) is popular as the multiplicative effect is constant, thus making interpretations easier. Also, in medical research and some other research areas, OR can be an excellent approximation of the relative risk, which is the probability ratio of two groups, of some rare disease or events. However, odds and odds ratio are never intuitive metrics for people, and in many situations a large odds ratio may be misleading as it corresponds to a very small effect. Therefore, in general I would recommend you to interpret coefficients in probability unit, even though that means more work.

#### 18.5.3.4 Interpreting Coefficients in Probability Units

Another way to interpret the results of logistic regression coefficient is to examine the change in probability. Because the predicted probability is a non-linear function of the predictors, a one unit difference in the predictor has different meanings depending on the values on  $X$  you chose for interpretations. You can see that in the `conditional_effects()` plot earlier.

Consider the change in the predicted probability of reporting a marginal  $p$  value with  $\text{Year10} = 0$  (1970),  $\text{Year10} = 1$  (1980), to  $\text{Year10} = 4$ , respectively:

```
m4_pred <- posterior_epred(m4, list(Year10 = 0:4))
colnames(m4_pred) <- paste0("Year10=", 0:4)
summarise_draws(m4_pred)
```

```
# A tibble: 5 x 10
```

	variable	mean	median	sd	mad	q5	q95	rhat	ess_bulk	ess_tail
	<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	Year10=0	0.141	0.140	0.0230	0.0230	0.106	0.180	1.00	2062.	2024.
2	Year10=1	0.187	0.186	0.0211	0.0212	0.154	0.223	1.00	2072.	2150.
3	Year10=2	0.245	0.244	0.0195	0.0191	0.214	0.278	1.00	2460.	2529.
4	Year10=3	0.314	0.314	0.0235	0.0233	0.275	0.353	1.00	3601.	2610.
5	Year10=4	0.393	0.392	0.0351	0.0349	0.335	0.451	1.00	3948.	2653.

As you can see, the predicted difference in probability is smaller when comparing Year10 = 0 and 1, but is larger when comparing Year10 = 3 and 4.

#### 💡 The “divide by 4 rule”

A quick approximation is to divide the coefficient by 4 to get an upper bound on the change in probability associated with a one unit change in the predictor. In our example, this corresponds to  $0.3454722 / 4 = 0.086368$ , which is very close to the predicted difference in probability from Year10 = 3 to Year10 = 4.

#### 18.5.4 Posterior Predictive Check

```
pp_check(
  m4,
  type = "error_binned"
)
```

Using 10 posterior draws for ppc type 'error\_binned' by default.

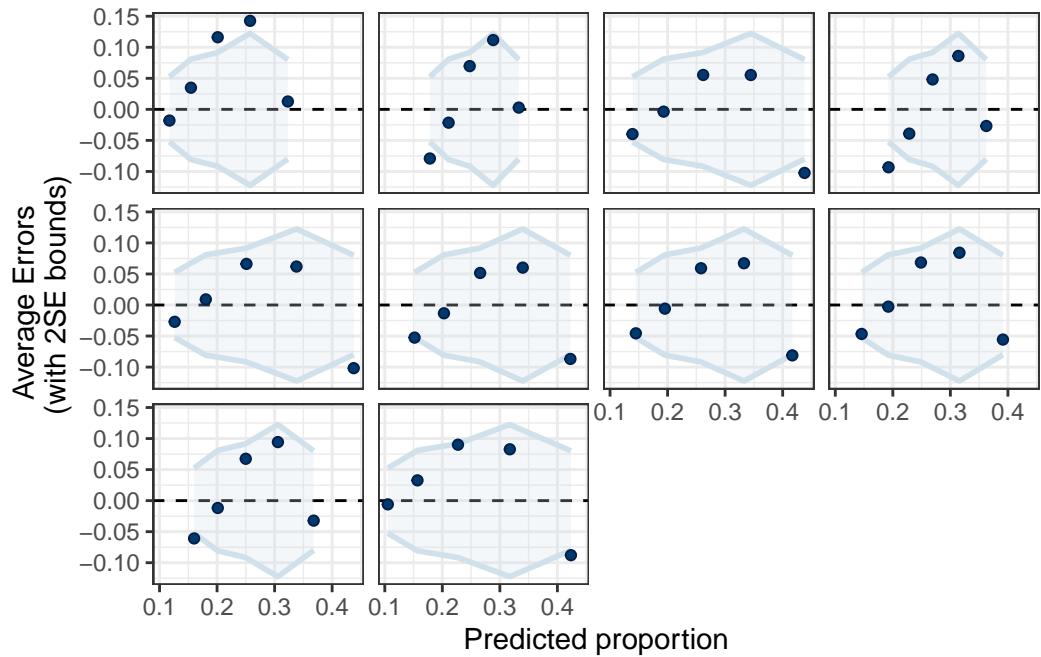


Figure 18.5: Posterior Predictive Check for the binary logistic model.

The linear model is not the best fit.

### 18.5.5 Linear Spline

Use `bs()` to specify a linear spline (degree = 1) with one turning point (knots = 3).

```
library(splines)
m5 <- brm(marginal_p ~ bs(Year10, degree = 1, knots = 3),
  data = marginalp_dev,
  family = bernoulli(link = "logit"),
  prior = prior(student_t(4, 0, .875), class = "b"),
  # Note: no sigma
  seed = 1340,
  file = "10_m5"
)
```

### 18.5.6 Posterior Prediction

```
plot(  
  conditional_effects(m5),  
  points = TRUE,  
  point_args = list(height = 0.01, width = 0.05, alpha = 0.05)  
)
```

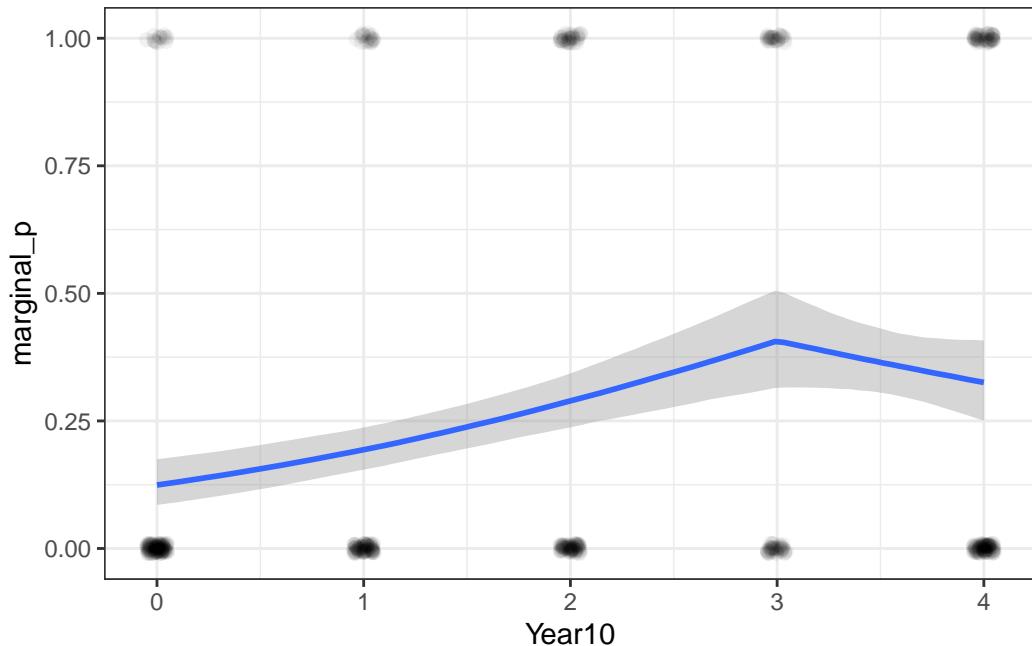


Figure 18.6: Predicted probabilities of marginal significant results over time with the spline model.

```
pp_check(  
  m5,  
  type = "error_binned",  
  x = "Year10"  
)
```

Using 10 posterior draws for ppc type 'error\_binned' by default.

Warning: The following arguments were unrecognized and ignored: x

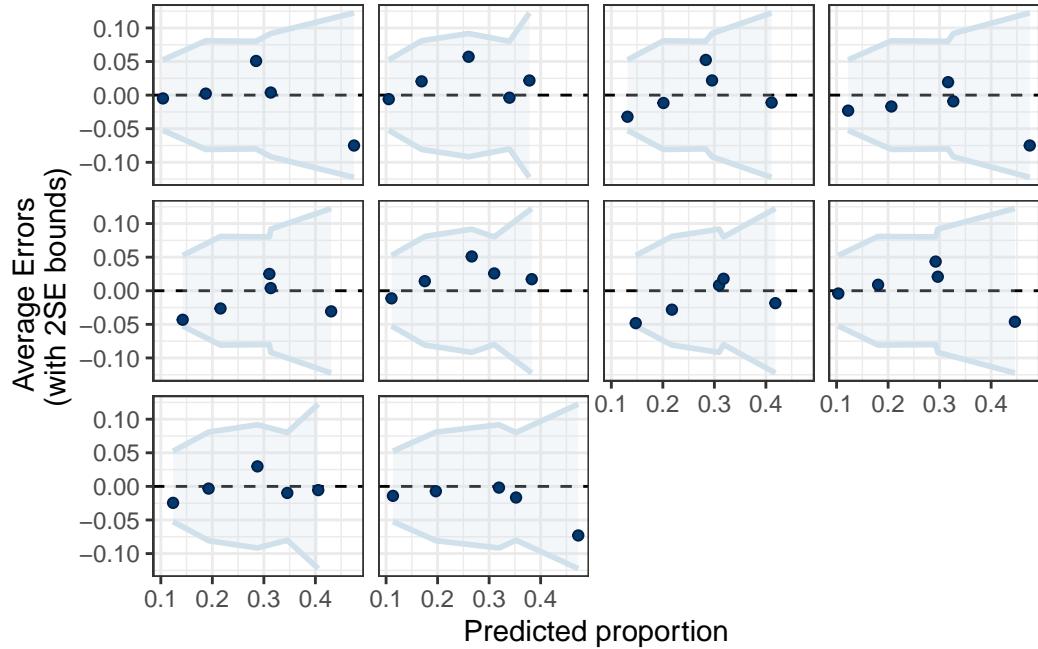


Figure 18.7: Posterior Predictive Check for the binary logistic model with a linear spline.

### 18.5.7 Model Comparison

```
msummary(list(linear = m4, `linear spline` = m5),
         estimate = "{estimate} [{conf.low}, {conf.high}]",
         statistic = NULL, fmt = 2)
```

Warning:

`modelsummary` uses the `performance` package to extract goodness-of-fit statistics from models of this class. You can specify the statistics you wish to compute by supplying a `metrics` argument to `modelsummary`, which will then push it forward to `performance`. Acceptable values are: "all", "common", "none", or a character vector of metrics names. For example: `modelsummary(mod, metrics = c("RMSE", "R2"))` Note that some metrics are computationally expensive. See `?performance::performance` for details.

This warning appears once per session.

The linear spline is better according to information criteria.

	linear	linear spline
b_Intercept	-1.82 [-2.20, -1.46]	-1.95 [-2.37, -1.55]
b_Year10	0.35 [0.22, 0.48]	
b_bsYear10degreeEQ1knotsEQ31		1.57 [0.93, 2.27]
b_bsYear10degreeEQ1knotsEQ32		1.22 [0.69, 1.77]
Num.Obs.	535	535
R2	0.049	0.050
ELPD	-292.9	-290.1
ELPD s.e.	11.2	11.1
LOOIC	585.8	580.2
LOOIC s.e.	22.5	22.2
WAIC	585.8	580.2
RMSE	0.43	0.42

# 19 Generalized Linear Model (II)

## 19.1 Binomial Logistic Regression

```
datfile <- here("data", "marginalp.xlsx")
marginalp <- readxl::read_excel(datfile)
# Recode `Field` into a factor
marginalp <- marginalp |>
  # Filter out studies without any experiments
  filter(`Number of Experiments` >= 1) |>
  mutate(Field = factor(Field,
    labels = c(
      "Cognitive Psychology",
      "Developmental Psychology",
      "Social Psychology"
    )
  )) |>
  # Rename the outcome
  rename(marginal_p = `Marginals Yes/No`)
marginalp <- marginalp |>
  mutate(Year10 = (Year - 1970) / 10)
marginalp_dev <- filter(marginalp,
  Field == "Developmental Psychology")
```

### 💡 Two Equivalent Models

When the probability is assumed equal across trials, the following are equivalent:

- Individual data: Bernoulli
- Grouped data: Binomial

### 19.1.1 Model

$$\begin{aligned}\text{marginal\_p}_j &\sim \text{Bin}(N_j, \mu_j) \\ \log(\mu_j) &= \eta_j \\ \eta_j &= \beta_0 + \beta_1 \text{Year10}_j\end{aligned}$$

Priors:

$$\begin{aligned}\beta_0 &\sim t_3(0, 2.5) \\ \beta_1 &\sim t_4(0, 1)\end{aligned}$$

```
# The group should only be done for observations with
# the same predicted probabilities
marginalp_dev_grouped <-
  marginalp_dev |>
  group_by(Year10) |>
  summarize(
    marginal_p = sum(marginal_p), # number of "successes"
    n = n() # number of trials
  )

m5_bin <- brm(
  marginal_p | trials(n) ~ bs(Year10, degree = 1, knots = 3),
  data = marginalp_dev_grouped,
  family = binomial(link = "logit"),
  prior = prior(student_t(4, 0, 1), class = "b"),
  # Note: no sigma
  seed = 1340,
  file = "10_m5_bin"
)

pp_check(m5_bin, type = "intervals", x = "Year10")
```

Using all posterior draws for ppc type 'intervals' by default.

Loading required package: rstan

Loading required package: StanHeaders

```
rstan version 2.32.5 (Stan version 2.32.2)
```

```
For execution on a local, multicore CPU with excess RAM we recommend calling  
options(mc.cores = parallel::detectCores()).  
To avoid recompilation of unchanged Stan programs, we recommend calling  
rstan_options(auto_write = TRUE)  
For within-chain threading using `reduce_sum()` or `map_rect()` Stan functions,  
change `threads_per_chain` option:  
rstan_options(threads_per_chain = 1)
```

```
Attaching package: 'rstan'
```

```
The following object is masked from 'package:tidy঱':
```

```
extract
```

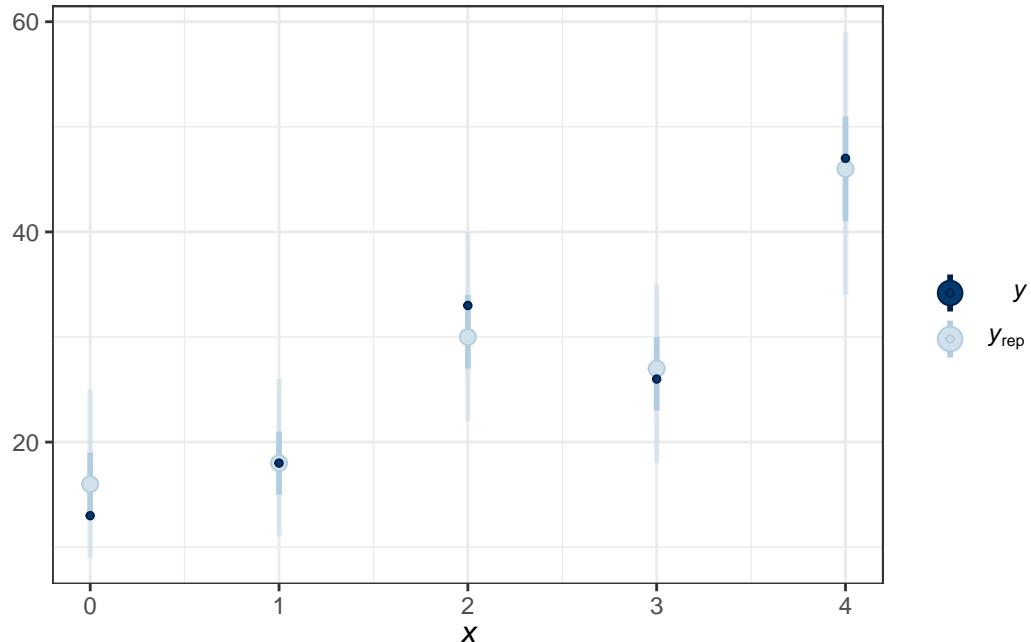


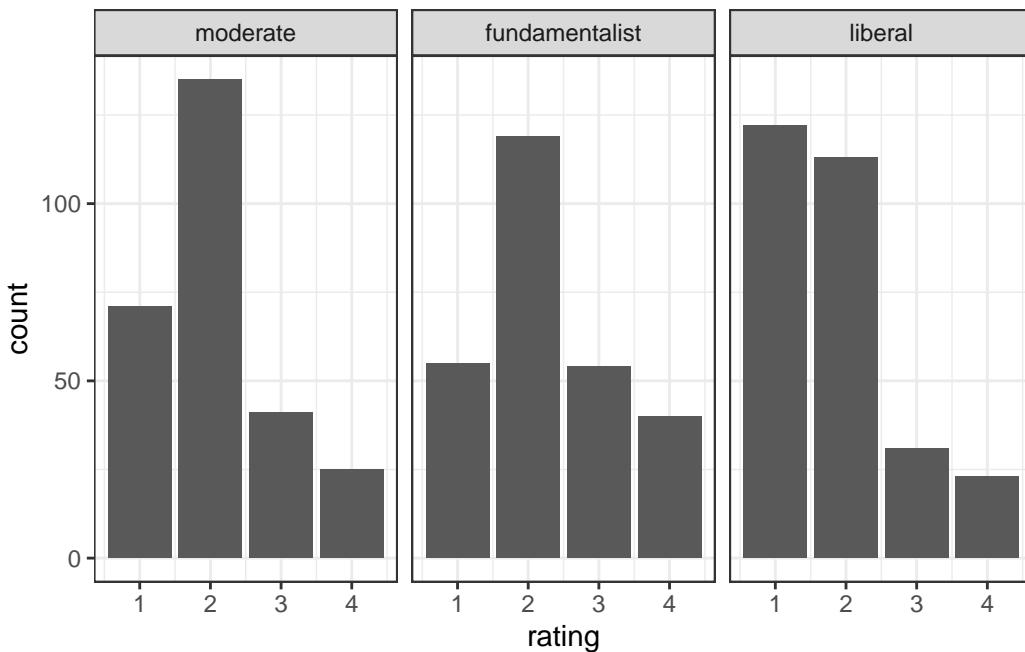
Figure 19.1: Posterior predictive check using the predicted and observed counts.

## 19.2 Ordinal Regression

Check out this paper: <https://journals.sagepub.com/doi/full/10.1177/2515245918823199>

```
stemcell <- read.csv("https://osf.io/vxw73/download")
stemcell <- stemcell |>
  mutate(
    belief = factor(belief,
                    levels = c("moderate", "fundamentalist", "liberal"))
  )
)

stemcell |>
  ggplot(aes(x = rating)) +
  geom_bar() +
  facet_wrap(~ belief)
```



[https://www.thearda.com/archive/files/Codebooks/GSS2006\\_CB.asp](https://www.thearda.com/archive/files/Codebooks/GSS2006_CB.asp)

The outcome is attitude towards stem cells research, and the predictor is religious belief.

Recently, there has been controversy over whether the government should provide any funds at all for scientific research that uses stem cells taken from human embryos. Would you say the government . . .

- 1 = Definitely, should fund such research
- 2 = Probably should fund such research
- 3 = Probably should not fund such research
- 4 = Definitely should not fund such research

## 19.3 Model

$$\begin{aligned}
 \text{rating}_i &\sim \text{Categorical}(\pi_i^1, \pi_i^2, \pi_i^3, \pi_i^4) \\
 \pi_i^1 &= \text{logit}^{-1}(\tau^1 - \eta_i) \\
 \pi_i^2 &= \text{logit}^{-1}(\tau^2 - \eta_i) - \text{logit}^{-1}(\tau^1 - \eta_i) \\
 \pi_i^3 &= \text{logit}^{-1}(\tau^3 - \eta_i) - \text{logit}^{-1}(\tau^2 - \eta_i) \\
 \pi_i^4 &= 1 - \text{logit}^{-1}(\tau^3 - \eta_i) \\
 \eta_i &= \beta_1 \text{fundamentalist}_i + \beta_2 \text{liberal}_i
 \end{aligned}$$

Priors:

$$\begin{aligned}
 \tau^1, \tau^2, \tau^3 &\sim t_3(0, 2.5) \\
 \beta_1 &\sim N(0, 1)
 \end{aligned}$$

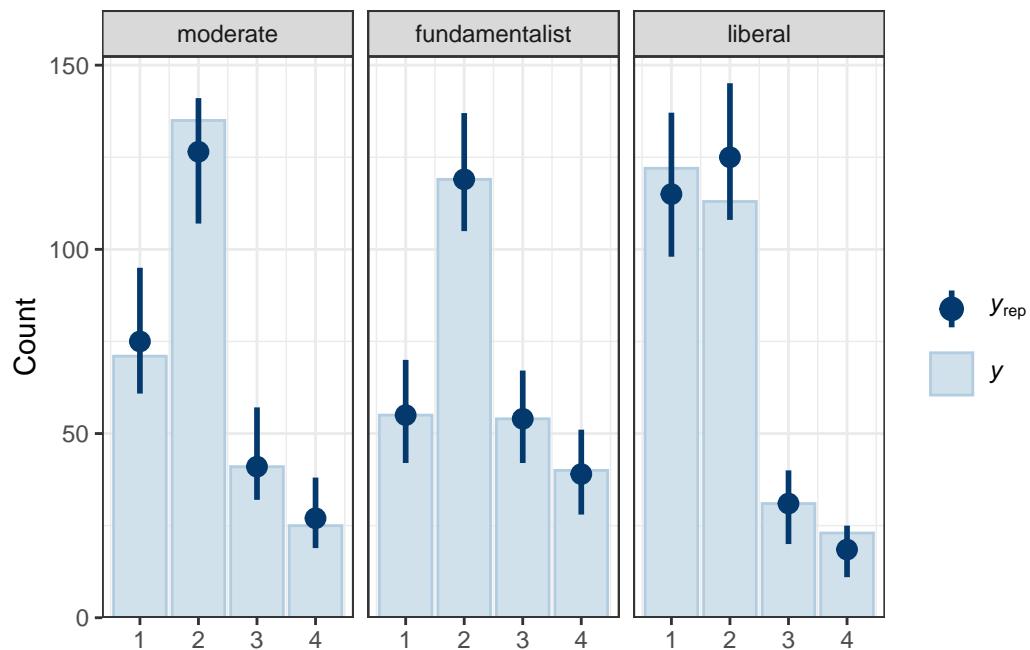
```

m6 <- brm(
  rating ~ belief,
  data = stemcell,
  family = cumulative(link = "logit"),
  prior = prior(std_normal(), class = "b"),
  seed = 1340,
  file = "10_m6"
)
  
```

### 19.3.1 Posterior Predictive Check

```

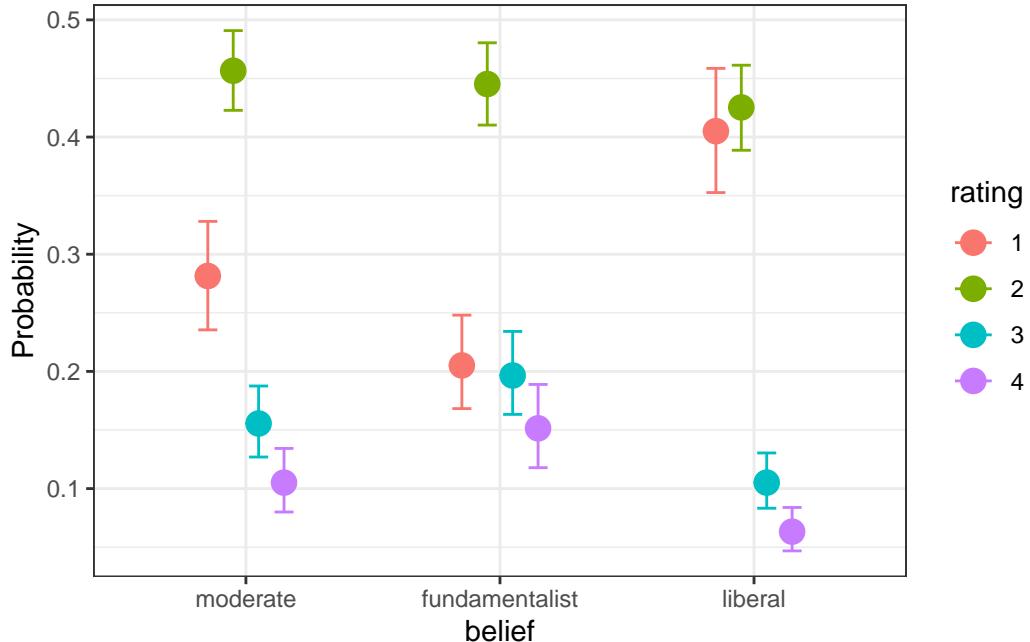
pp_check(m6, type = "bars_grouped", group = "belief",
         ndraws = 100)
  
```



The fit was reasonable.

### 19.3.2 Plot

```
conditional_effects(m6, categorical = TRUE)
```



## 19.4 Nominal Logistic Regression

Ordinal regression is a special case of nominal regression with the proportional odds assumption.

### 19.4.1 Model

$$\begin{aligned}
 \text{rating}_i &\sim \text{Categorical}(\pi_i^1, \pi_i^2, \pi_i^3, \pi_i^4) \\
 \pi_i^1 &= \frac{1}{\exp(\eta_i^2) + \exp(\eta_i^3) + \exp(\eta_i^4) + 1} \\
 \pi_i^2 &= \frac{\exp(\eta_i^2)}{\exp(\eta_i^2) + \exp(\eta_i^3) + \exp(\eta_i^4) + 1} \\
 \pi_i^3 &= \frac{\exp(\eta_i^3)}{\exp(\eta_i^2) + \exp(\eta_i^3) + \exp(\eta_i^4) + 1} \\
 \pi_i^4 &= \frac{\exp(\eta_i^4)}{\exp(\eta_i^2) + \exp(\eta_i^3) + \exp(\eta_i^4) + 1} \\
 \eta_i^2 &= \beta_0^2 + \beta_1^2 \text{fundamentalist}_i + \beta_2^2 \text{liberal}_i \\
 \eta_i^3 &= \beta_0^3 + \beta_1^3 \text{belief}_i + \beta_2^3 \text{liberal}_i \\
 \eta_i^4 &= \beta_0^4 + \beta_1^4 \text{belief}_i + \beta_2^4 \text{liberal}_i
 \end{aligned}$$

As you can see, it has two additional parameters for each predictor column.

```
m7 <- brm(  
  rating ~ belief,  
  data = stemcell,  
  family = categorical(link = "logit"),  
  prior = prior(std_normal(), class = "b", dpar = "mu2") +  
    prior(std_normal(), class = "b", dpar = "mu3") +  
    prior(std_normal(), class = "b", dpar = "mu4"),  
  seed = 1340,  
  file = "10_m7"  
)
```

## 19.4.2 Model Comparison

```
msummary(list(`ordinal (proportional odds)` = m6, `norminal` = m7),  
  estimate = "{estimate} [{conf.low}, {conf.high}]",  
  statistic = NULL, fmt = 2)
```

Warning:

`modelsummary` uses the `performance` package to extract goodness-of-fit statistics from models of this class. You can specify the statistics you wish to compute by supplying a `metrics` argument to `modelsummary`, which will then push it forward to `performance`. Acceptable values are: "all", "common", "none", or a character vector of metrics names. For example: `modelsummary(mod, metrics = c("RMSE", "R2"))` Note that some metrics are computationally expensive. See `?performance::performance` for details.

This warning appears once per session.

	ordinal (proportional odds)	norminal
b_Intercept[1]	-0.94 [-1.18, -0.72]	
b_Intercept[2]	1.04 [0.81, 1.28]	
b_Intercept[3]	2.14 [1.86, 2.44]	
b_belieffundamentalist	0.41 [0.12, 0.72]	
b_beliefliberal	-0.55 [-0.85, -0.25]	
b_mu2_Intercept		0.63 [0.38, 0.90]
b_mu3_Intercept		-0.57 [-0.94, -0.20]
b_mu4_Intercept		-1.05 [-1.48, -0.65]
b_mu2_belieffundamentalist		0.12 [-0.29, 0.54]
b_mu2_beliefliberal		-0.69 [-1.06, -0.33]
b_mu3_belieffundamentalist		0.52 [0.00, 1.04]
b_mu3_beliefliberal		-0.77 [-1.32, -0.24]
b_mu4_belieffundamentalist		0.70 [0.13, 1.26]
b_mu4_beliefliberal		-0.58 [-1.18, 0.01]
Num.Obs.	829	829
R2	0.043	
ELPD	-1019.2	-1020.9
ELPD s.e.	15.5	15.6
LOOIC	2038.5	2041.8
LOOIC s.e.	31.1	31.2
WAIC	2038.5	2041.8

# References

- Brooks, S. P., & Gelman, A. (1998). General methods for monitoring convergence of iterative simulations. *Journal of Computational and Graphical Statistics*, 7(4), 434–455. <https://doi.org/10.1080/10618600.1998.10474787>
- Carvalho, C. M., Polson, N. G., & Scott, J. G. (2010). The horseshoe estimator for sparse signals. *Biometrika*, 97(2), 465–480. <https://doi.org/10.1093/biomet/asq017>
- Gelman, A., Hill, J., & Vehtari, A. (2021). *Regression and other stories*. Cambridge University Press.
- Gelman, A., & Rubin, D. B. (1992). Inference from iterative simulation using multiple sequences. *Statistical Science*, 7(4). <https://doi.org/10.1214/ss/1177011136>
- Gigerenzer, G. (2004). Mindless statistics. *The Journal of Socio-Economics*, 33(5), 587–606. <https://doi.org/10.1016/j.socec.2004.09.033>
- Hoeting, J. A., Madigan, D., Raftery, A. E., & Volinsky, C. T. (1999). Bayesian model averaging: A tutorial. *Statistical Science*, 14(4). <https://doi.org/10.1214/ss/1009212519>
- Imai, K., Keele, L., & Tingley, D. (2010). A general approach to causal mediation analysis. *Psychological Methods*, 15(4), 309–334. <https://doi.org/10.1037/a0020761>
- Johnson, A. A., Ott, M. Q., & Dogucu, M. (2022). *Bayes rules! An introduction to Bayesian modeling with R*. CRC Press.
- Lambert, B. (2018). *A student's guide to Bayesian statistics*. SAGE.
- McCandless, L. C., & Somers, J. M. (2019). Bayesian sensitivity analysis for unmeasured confounding in causal mediation analysis. *Statistical Methods in Medical Research*, 28(2), 515–531. <https://doi.org/10.1177/0962280217729844>
- McElreath, R. (2020). *Statistical rethinking: a Bayesian course with examples in R and Stan* (Second edition). CRC Press.
- McGrayne, S. B. (2011). *The theory that would not die: How Bayes' rule cracked the enigma code, hunted down Russian submarines, & emerged triumphant from two centuries of controversy*. Yale university press.
- Piironen, J., Paasiniemi, M., & Vehtari, A. (2020). Projective inference in high-dimensional problems: Prediction and feature selection. *Electronic Journal of Statistics*, 14(1). <https://doi.org/10.1214/20-EJS1711>
- Piironen, J., & Vehtari, A. (2017). Sparsity information and regularization in the horseshoe and other shrinkage priors. *Electronic Journal of Statistics*, 11(2). <https://doi.org/10.1214/17-EJS1337SI>
- Van De Schoot, R., Winter, S. D., Ryan, O., Zondervan-Zwijnenburg, M., & Depaoli, S. (2017). A systematic review of Bayesian articles in psychology: The last 25 years. *Psychological Methods*, 22(2), 217–239. <https://doi.org/10.1037/met0000100>

- Vehtari, A., Gelman, A., & Gabry, J. (2017). Practical Bayesian model evaluation using leave-one-out cross-validation and WAIC. *Statistics and Computing*, 27(5), 1413–1432. <https://doi.org/10.1007/s11222-016-9696-4>
- Vehtari, A., Gelman, A., Simpson, D., Carpenter, B., & Bürkner, P.-C. (2021). Rank-normalization, folding, and localization: An improved R<sup>̂</sup> for assessing convergence of MCMC (with discussion). *Bayesian Analysis*, 16(2). <https://doi.org/10.1214/20-BA1221>
- Yao, Y., Vehtari, A., Simpson, D., & Gelman, A. (2018). Using stacking to average Bayesian predictive distributions (with discussion). *Bayesian Analysis*, 13(3). <https://doi.org/10.1214/17-BA1091>
- Yimer, B. B., Lunt, M., Beasley, M., Macfarlane, G. J., & McBeth, J. (2023). BayesGmed: An R-package for Bayesian causal mediation analysis. *PLOS ONE*, 18(6), e0287037. <https://doi.org/10.1371/journal.pone.0287037>