Bayesian Data Analysis, Part 2

Clay Ford

Spring 2022

Agenda

- Review some basic concepts from Part 1
- Implement more complex Bayesian analyses
- model visualization
- model comparison

Review: Bayesian statistics

- The objective of a Bayesian analysis is a *posterior distribution* of our parameter(s) of interest
- To obtain a posterior distribution, we must start with a *prior distribution* and a *likelihood*
- Using *Markov chain Monte Carlo (MCMC)* we sample from the prior and likelihood to estimate the posterior distributions
- The `rstanarm` package allows us to perform common Bayesian analyses without having to learn Stan

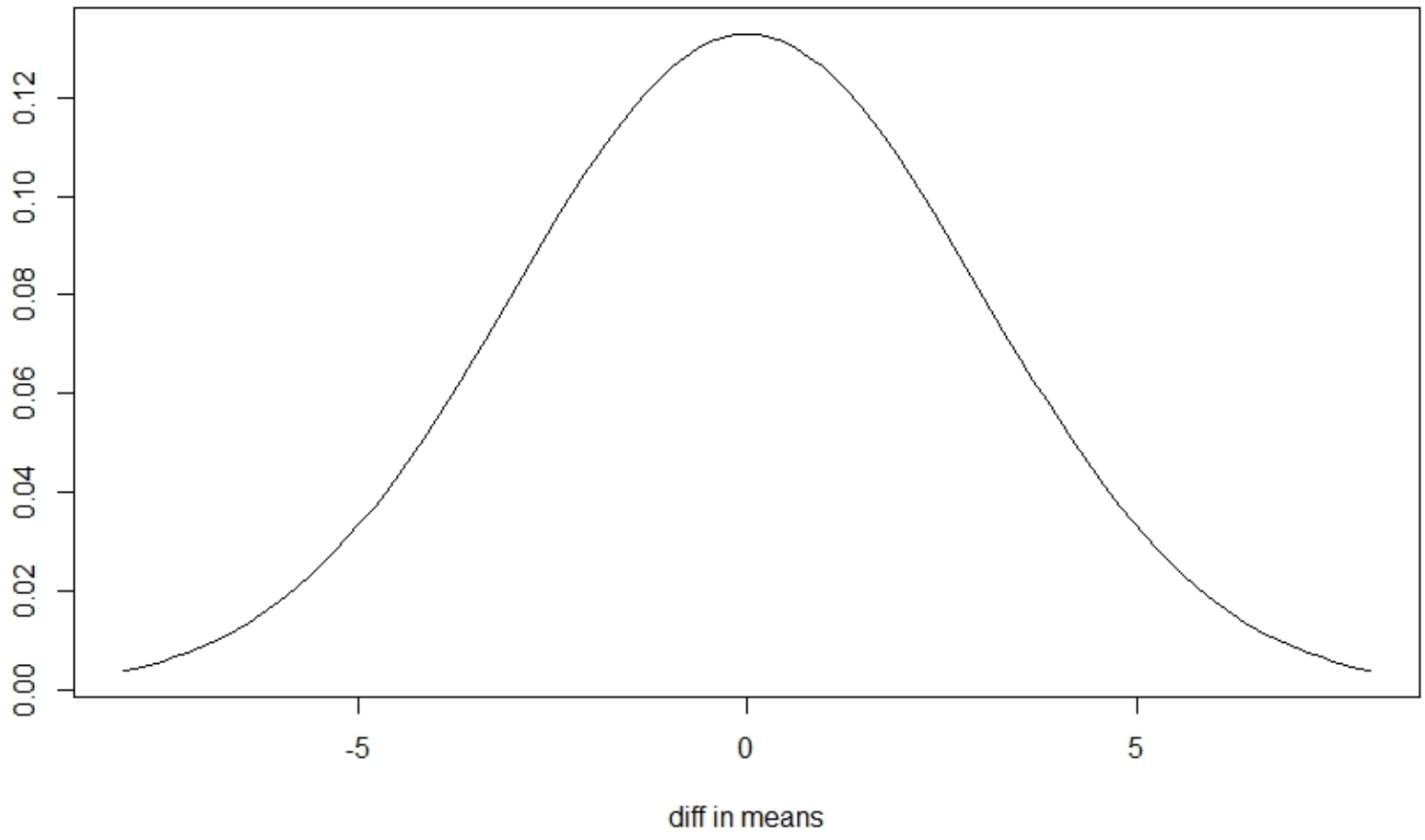
Review: Example

Examine two brands of rechargeable batteries. How long do they run (in hours) before exhausted? Take 25 samples of each brand and calculate mean time.

- begin with a *prior* distribution of how we believe the differences are distributed
- update the prior distribution using the *likelihood* of the data we collected to obtain a *posterior* distribution
- use the posterior distribution to describe the differences in brands

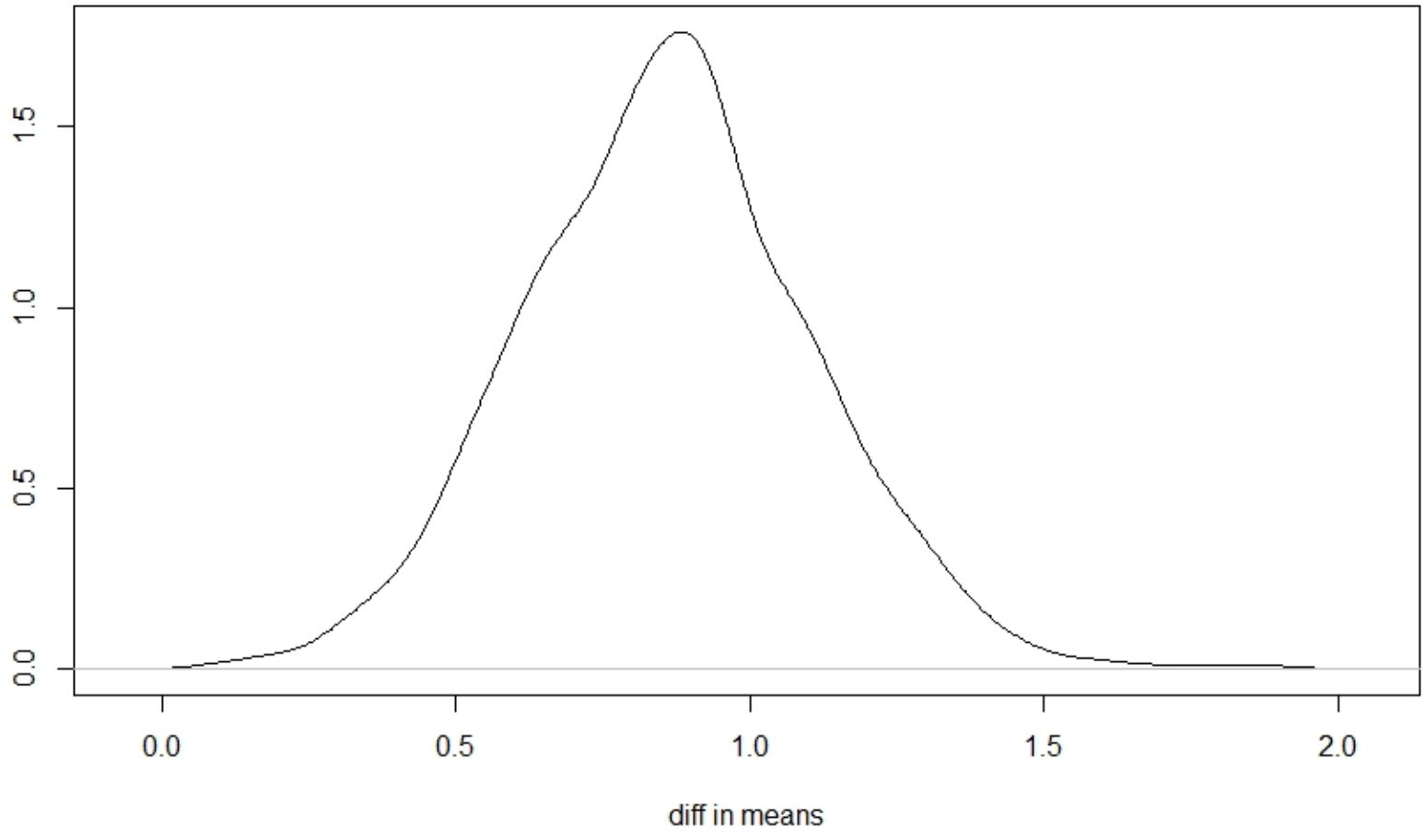
Review: a prior distribution

Perhaps we're not sure which brand is better. A possible prior distribution might look like this, a $N(0, 3)$ distribution.



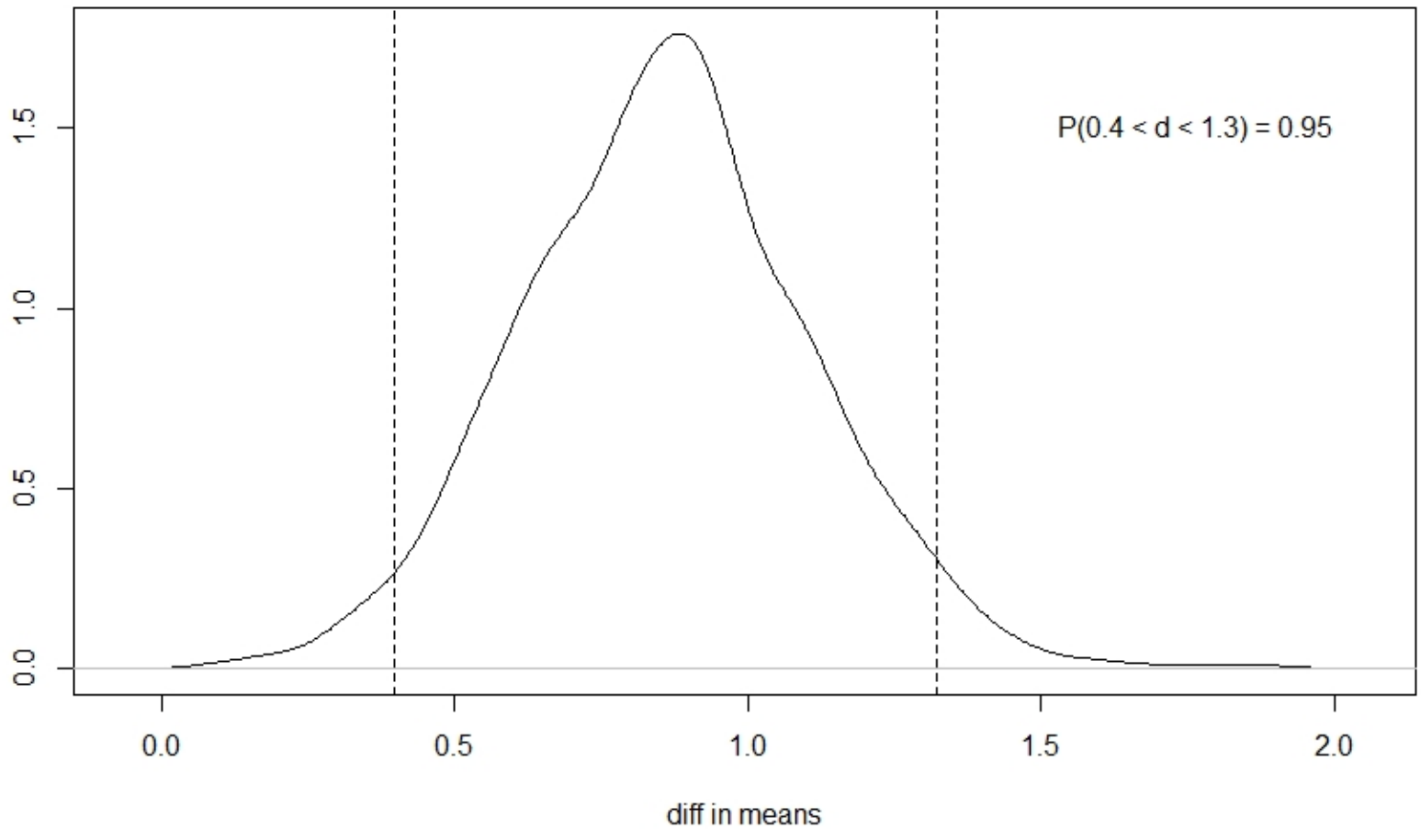
Review: a posterior distribution

After observing our data we use *Bayes' theorem* to update our prior distribution to get a posterior distribution.



Review: using posterior distribution

The probability the difference is between 0.4 and 1.3 is about 0.95.



Review: Implementation

```
bat <- read.csv("data/batteries.csv")
dplyr::sample_n(bat, 6)
```

```
##           y grp
## 1 10.887654   1
## 2 12.100025   1
## 3 10.292505   1
## 4 10.183643   0
## 5 11.595281   0
## 6  9.378759   0
```

```
aggregate(y ~ grp, data = bat, mean)
```

```
##   grp        y
## 1   0 10.16867
## 2   1 11.03223
```

Review: Implementation

```
library(rstanarm)
bm.out <- stan_glm(y ~ grp, data = bat,
                   family = gaussian,
                   prior = normal(0,3))
```

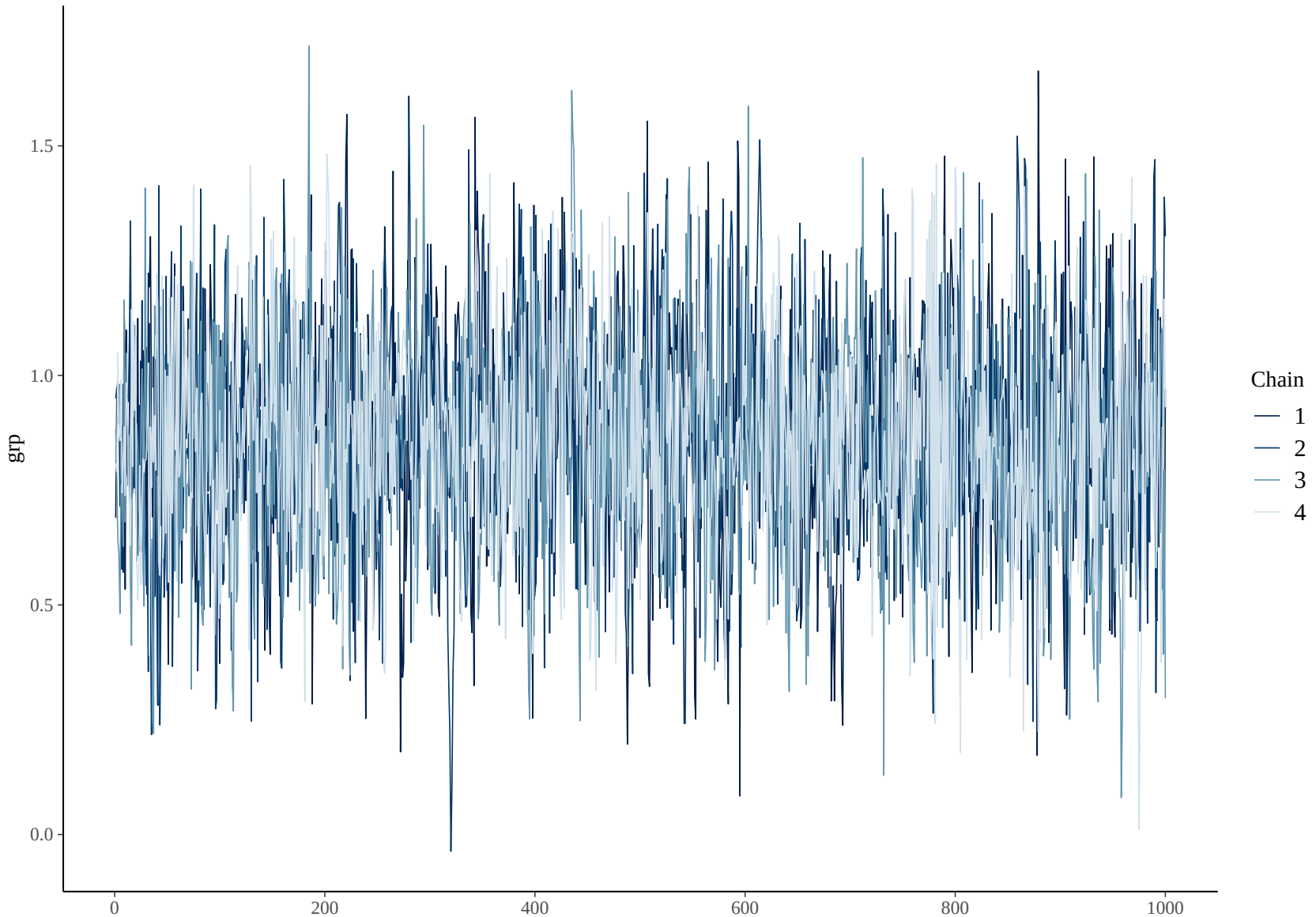
Bayesian uncertainty interval estimated from the posterior distribution:

```
round(posterior_interval(bm.out, prob = 0.95,
                         pars = "grp"),2)
```

```
##      2.5% 97.5%
## grp 0.39  1.33
```

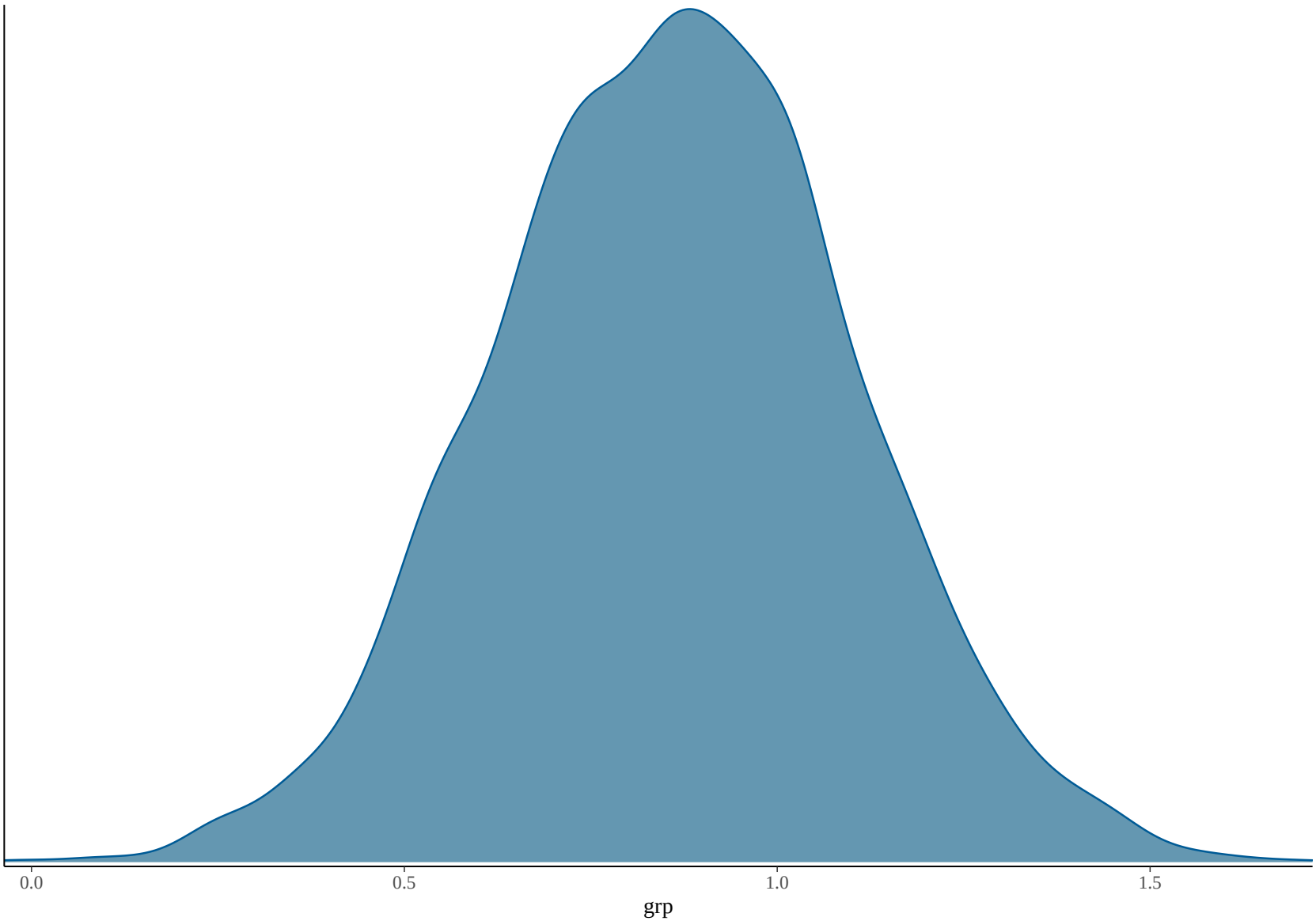
Review: trace plot

```
plot(bm.out, plotfun = "trace", pars = "grp")
```



Review: plotting posterior distribution

```
plot(bm.out, plotfun = "dens", pars = "grp")
```



Review: Working with posterior distribution

Let's say we want to estimate the probability that the difference in battery life is:

- greater than 0
- greater than 1 hour

`as.data.frame` creates a data frame of *posterior samples*.

```
post <- as.data.frame(bm.out)
mean(post$grp > 0)
```

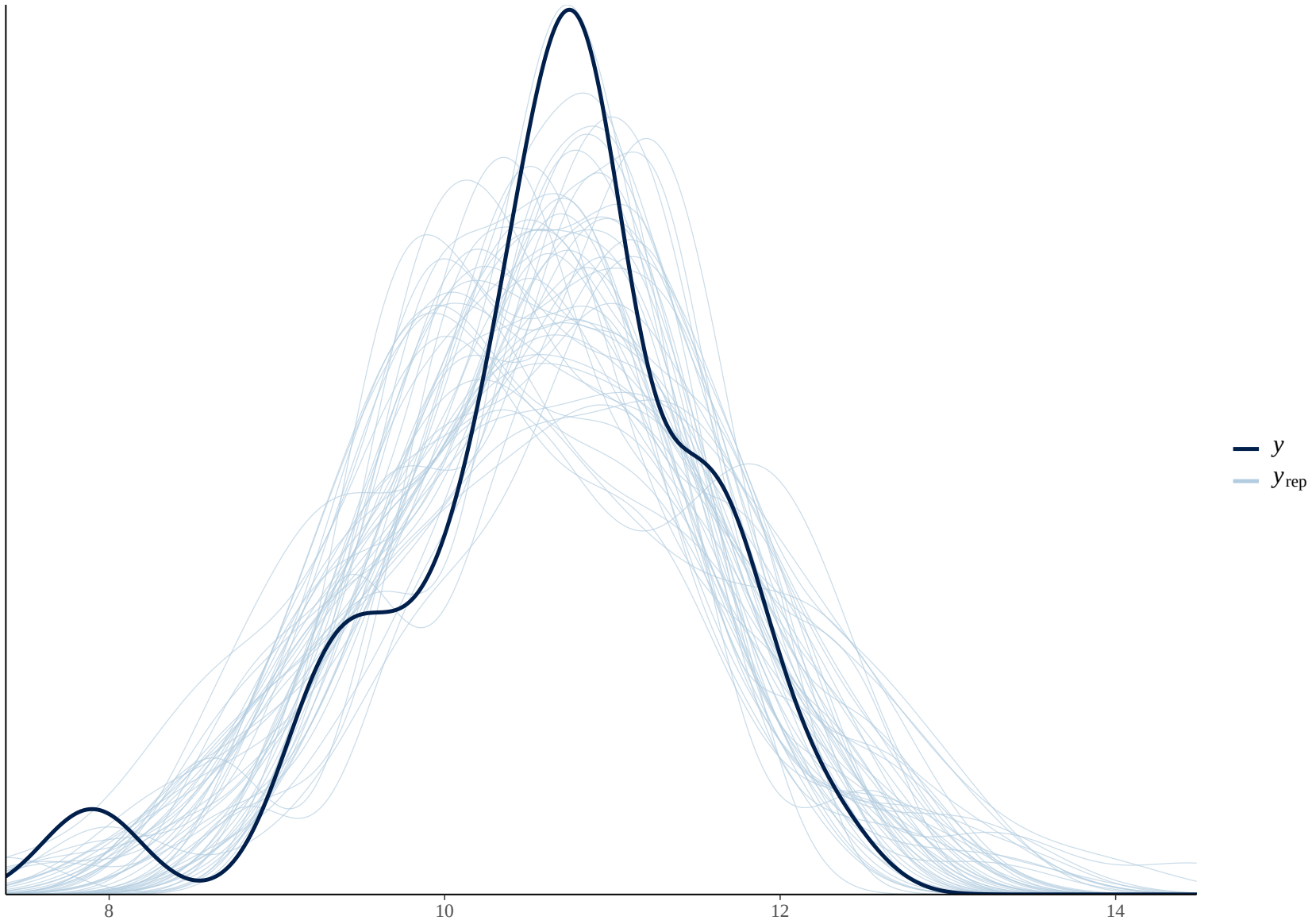
```
## [1] 0.99975
```

```
mean(post$grp > 1)
```

```
## [1] 0.28475
```

Review: graphical posterior predictive checks

```
pp_check(bm.out)
```



Moving on to more complex analyses

The previous example was a simple *model* with one predictor. It was essentially the Bayesian approach to what is traditionally analyzed as a t test.

Today's workshop will cover more complicated analyses including:

- multiple regression
- regression with interactions
- binary logistic regression

Multiple regression

Multiple regression, or linear modeling, is the idea that the variability of some numeric variable of interest can be "explained" by a sum of weighted predictors.

Example: patient satisfaction scores at a hospital. Some are high, some are low. Why? Perhaps it has do with their age, anxiety level, and illness severity.

$$satisfaction = \beta_0 + \beta_1 age + \beta_2 anxiety + \beta_3 illness$$

Where the betas represent some *weight*. Hence the term, *weighted sum*. β_0 is the intercept.

Multiple regression

$$satisfaction = \beta_0 + \beta_1 age + \beta_2 anxiety + \beta_3 illness$$

This model says, "if I take age, anxiety level and illness severity, multiply each by some weight, and add them up, I'll get an expected patient satisfaction score."

The calculated value will be off by some amount. We assume this amount, usually denonted ϵ, is a random draw from a Normal distribution with mean 0 and some unknown standard deviation, σ. This gives us

$$satisfaction = \beta_0 + \beta_1 age + \beta_2 anxiety + \beta_3 illness + \epsilon$$

Traditional multiple regression means estimating the betas and σ.

Using `lm`

The traditional approach in R uses the `lm` function.

```
ps <- read.csv("data/patient_satisfaction.csv")
m <- lm(ps ~ age + illness + anxiety, data = ps)
```

The betas (coefficients/weights) and σ can be viewed with `coef` and `sigma`

```
coef(m)
```

```
## (Intercept)         age     illness     anxiety
## 158.4912517  -1.1416118  -0.4420043 -13.4701632
```

```
sigma(m)
```

```
## [1] 10.05798
```

Using glm

We can also use glm for multiple regression. Note the family argument which allows us to specify the error distribution.

```
m2 <- glm(ps ~ age + illness + anxiety, data = ps,
          family = gaussian)
coef(m2)
```

```
## (Intercept)         age     illness     anxiety
## 158.4912517  -1.1416118  -0.4420043 -13.4701632
```

```
sigma(m2)
```

```
## [1] 10.05798
```

The Bayesian approach

As before, instead of estimating parameters, the Bayesian approach is to **estimate the distributions of parameters**.

We propose a prior distribution for the betas and σ, and update those distributions using a likelihood and the data.

Likelihood

Recall our model:

$$satisfaction = \beta_0 + \beta_1 age + \beta_2 anxiety + \beta_3 illness + \epsilon$$

where $\epsilon \sim N(0, \sigma)$

This implies

$$satisfaction \sim N(\beta_0 + \beta_1 age + \beta_2 anxiety + \beta_3 illness, \sigma)$$

This is our *likelihood*: A normal, or Gaussian, distribution.

Where traditional statistics maximizes likelihood, Bayesian statistics multiplies the likelihood by the prior to get a posterior distribution.

Using `stan_glm`

The `stan_glm` function uses the same syntax as `glm` and provides *weakly informative* default prior distributions.[1]

```
bm <- stan_glm(ps ~ age + illness + anxiety,
               data = ps,
               family = gaussian)
```

Instead of point estimates for the betas and σ, we get posterior distributions.

[1] On startup, rstanarm states "Default priors may change, so it's safest to specify priors, even if equivalent to the defaults."

Using stan_glm with explicit priors

Use the prior arguments to specify priors. Below are the default priors. The normal and exponential functions are from rstanarm.

```
bm <- stan_glm(ps ~ age + illness + anxiety,
               data = ps,
               family = gaussian,
               prior_intercept = normal(mean(ps$ps),10),
               prior = normal(c(0,0,0),c(2.5,2.5,2.5)),
               prior_aux = exponential(1))
```

The scale, or spread, of the priors is automatically rescaled to accommodate the range of the data.

Evaluating and exploring the model

We proceed the same way as before to evaluate and explore the model.

```
plot(bm, plotfun = "trace")
plot(bm, plotfun = "dens")
posterior_interval(bm)
summary(bm)
pp_check(bm)
```

Model checking

In addition, we can investigate the posterior's sensitivity to particular observations using `loo` (approximate leave-one-out cross-validation)

```
loo(bm)
```

We are notified if certain observations exceed a threshold (`pareto_k > 0.7`)

Let's go to R!

Models with interactions

In our previous model, we assumed the predictor effects were simply additive. For example, it didn't matter how ill you were, the effect of age was always the same.

$$satisfaction = \beta_0 + \beta_1 age + \beta_2 anxiety + \beta_3 illness$$

But we may have reason to believe the effects of age and illness interact. Perhaps the older you are, the effect of illness on patient satisfaction decreases.

One way to describe this interaction is to add the product of age and illness to the model:

$$satisfaction = \beta_0 + \beta_1 age + \beta_2 anxiety + \beta_3 illness + \beta_4 age \times illness$$

Specifying interactions

Use a colon to specify interactions in the model syntax.

```
bm2 <- stan_glm(ps ~ age + illness + anxiety +
                     age:illness,
                data = ps,
                family = gaussian)
```

Or use the asterisk as a shortcut: `age * illness` = `age + illness + age:illness`

The modeling result

Once again the target of the Bayesian model is the collection of posterior distributions on the model weights, or coefficients.

```
posterior_interval(bm2)
```

```
##                          5%          95%
## (Intercept)   26.20181467 251.08202193
## age           -3.42330550   2.16140201
## illness       -2.33270503   2.20467749
## anxiety      -25.84181972  -1.29440540
## age:illness   -0.06550184   0.04424613
## sigma          8.65894466  12.60598498
```

The interaction appears to be small and we're uncertain whether it's positive or negative.

The modeling result

The coef function returns the medians of the posterior distributions of the coefficients.[2]

```
as.matrix(coef(bm2))
```

```
##                          [,1]
## (Intercept) 138.552924803
## age          -0.661732680
## illness      -0.022662797
## anxiety     -13.660200408
## age:illness  -0.009611631
```

[2] Using as.matrix to force the coefficients into a column so they will fit on the slide.

Visualizing interactions

Even if we had good evidence that the coefficient for an interaction was large and in a certain direction (positive or negative), the coefficient can be hard to interpret.

To aid in interpretation we can use *effect plots* to help us visualize our models.

The basic idea is to generate predictions for various combinations of predictors and plot the result.

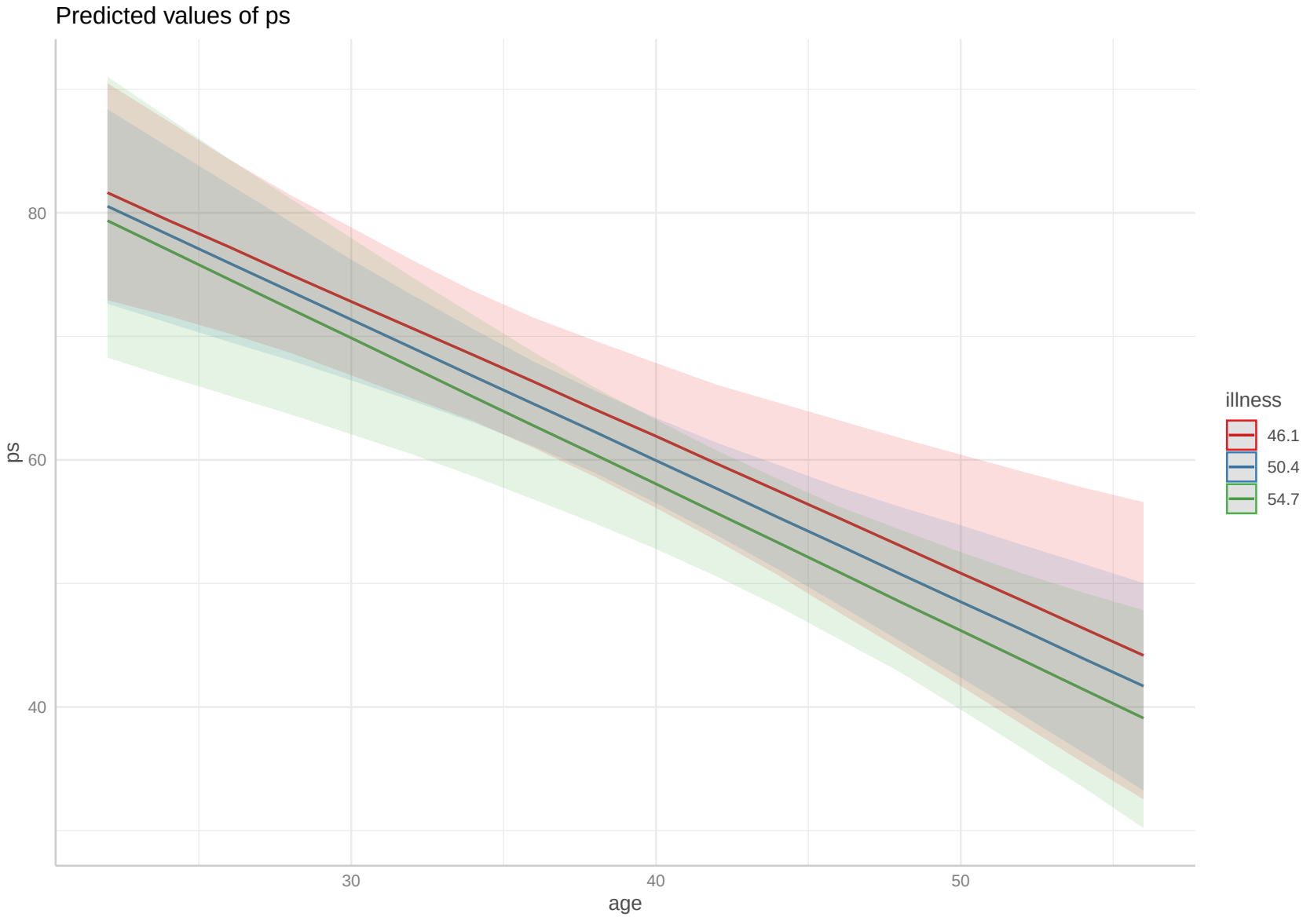
Using ggeffects to create effect plots

The ggeffects package provides methods for easily creating effect plots for models created with rstanarm.

The basic syntax to quickly generate an effect plot for our interaction:

```
library(ggeffects)
ggpredict(bm2, terms = c("age","illness")) |>
  plot()
```

An effect plot for the `age:illness` interaction



Customizing the effect plot

By default ggpredict will pick some values for the 2nd term. We can specify values if we like as follows:

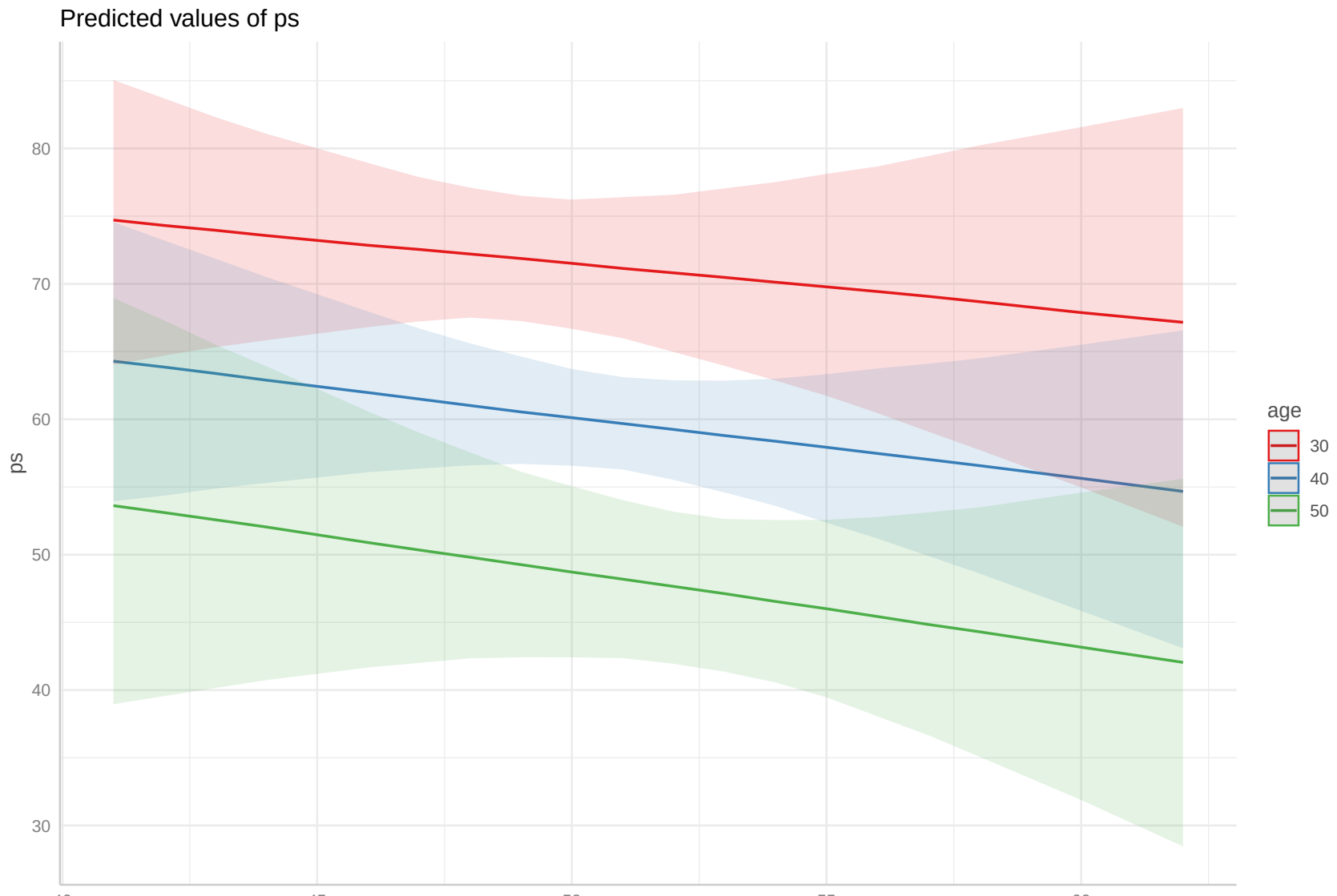
```
ggpredict(bm2, terms = c("age","illness[45,50,55]")) |>
  plot()
```

If we want illness on the x-axis:

```
ggpredict(bm2, terms = c("illness","age[30,40,50]")) |>
  plot()
```

An effect plot for the age:illness interaction

```
ggpredict(bm2, terms = c("illness","age[30,40,50]")) |>
  plot()
```



Interpreting interactions in effect plots

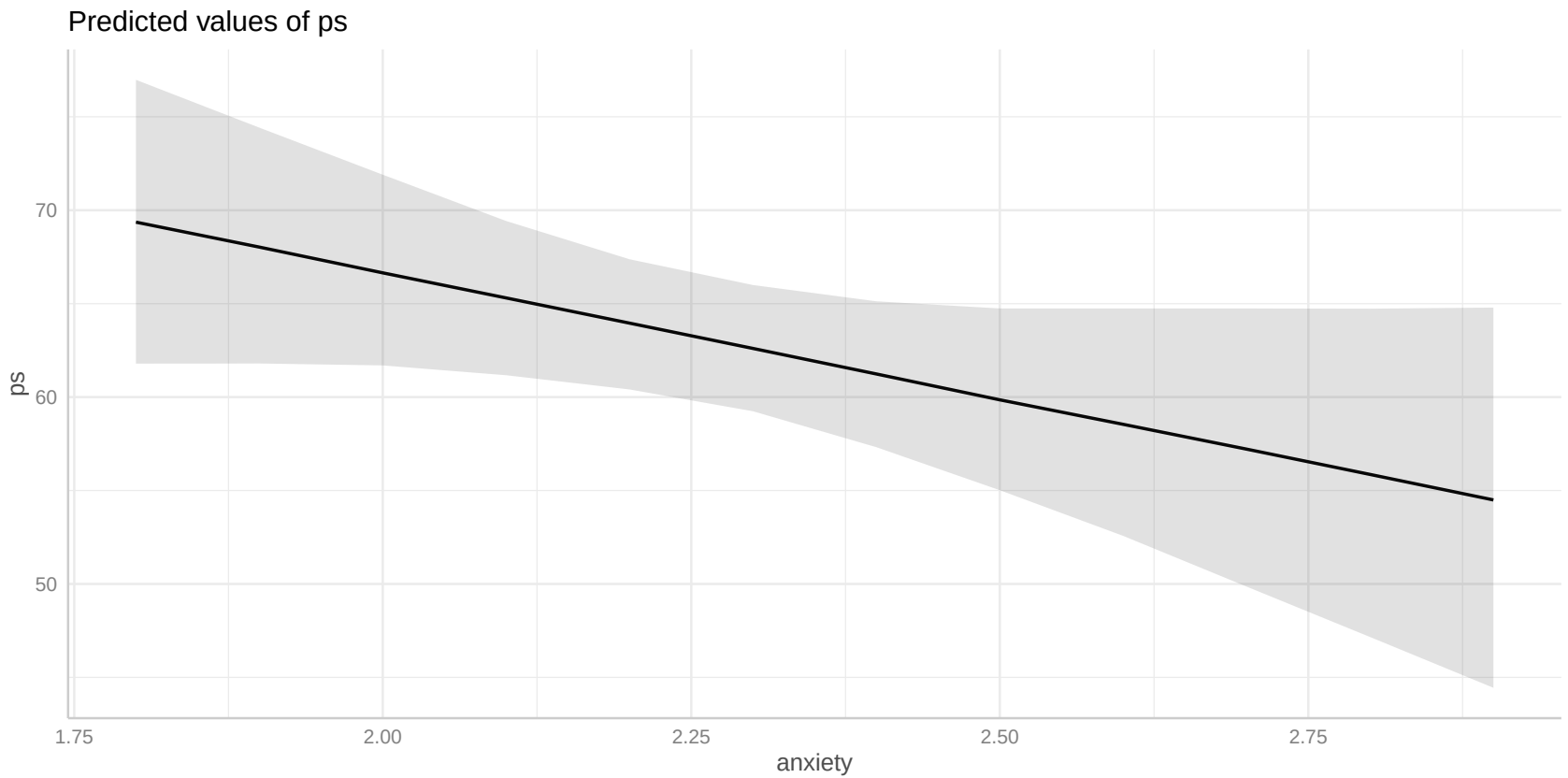
On the previous slide, the effect of illness on patient satisfaction appeared to be the same regardless of age. The plotted lines were approximately parallel. This indicates a small or negligible interaction.

If the plotted lines had vastly different trajectories such that they crossed or grew further apart, then we would have evidence of an interaction.

Effect plots for main effects

Effect plots are useful for main effects as well (ie, predictors not involved in an interaction)

```
ggpredict(bm2, terms = "anxiety") |> plot()
```

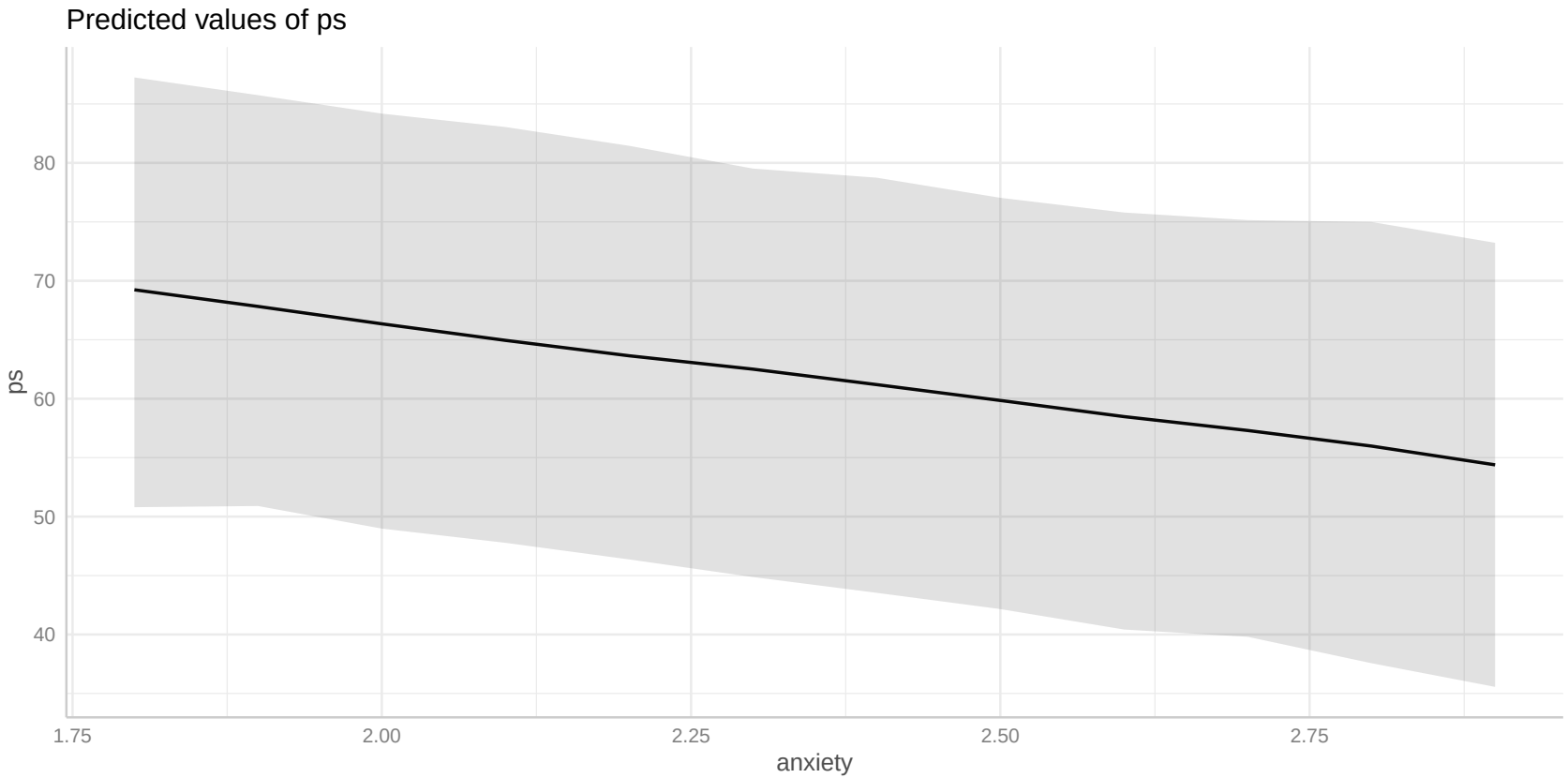


The 95% credibility ribbon is for the *mean* response value.

Effect plots for main effects

Set `ppd = TRUE` to get a prediction interval.

```
ggpredict(bm2, terms = "anxiety", ppd = TRUE) |> plot()
```



The 95% credibility ribbon is for the *predicted* response value.

Let's go to R!

Logistic regression

Logistic regression models the probability of a binary outcome. The dependent variable is often of the form 0/1, failure/success or no/yes.

Like multiple regression, we model probability as a weighted sum of predictors.

Unlike multiple regression, there is no σ and the weighted sum of predictors are embedded in the *logistic* function:

$$P(Y = 1) = \frac{1}{1 + \exp(-X\beta)}$$

where $X\beta = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_k X_k$

This ensures our model always returns values between 0 and 1.

The logit transformation

We can express the logistic regression model as a simple weighted sum of predictors by using the *logit* transformation:

$$\log\left(\frac{P(Y=1)}{1-P(Y=1)}\right) = \beta_0 + \beta_1 X_1 + \ldots + \beta_k X_k$$

In this transformation, the response and coefficients are on the *log odds* scale.

This is the form logistic regression takes when performed in R (or any other program).

Likelihood

Since we're modeling the probability of an event happening, our response variable has a Bernoulli distribution, or binomial distribution with $n = 1$:

$$Y \sim B\left(n = 1, p = \frac{1}{1 + \exp(-X\beta)}\right)$$

While traditional statistics maximizes this likelihood, Bayesian statistics multiplies the likelihood by the prior to get a posterior distribution.

Logistic regression example

A clinical trial investigates a new treatment for rheumatoid arthritis. Model probability of seeing improvement in condition based on treatment, sex, and age.

```
dplyr::sample_n(arthritis, 6)
```

```
##    Treatment    Sex Age Better
## 52   Placebo   Male  63      1
## 21   Treated Female  48      1
## 16   Treated Female  32      0
## 19   Treated Female  41      1
## 49   Placebo   Male  59      0
## 9    Treated   Male  63      0
```

$$\log\left(\frac{P(Better = 1)}{1 - P(Better = 1)}\right) = \beta_0 + \beta_1 Trt + \beta_2 Sex + \beta_3 Age$$

Fitting the model

The traditional method uses `glm`. Notice we set `family = binomial` since our response is binary.

```
glm1 <- glm(Better ~ Treatmnt + Sex + Age,
            data = arthritis,
            family = binomial)
```

The `rstanarm` specification is virtually identical, except we use `stan_glm`

```
bglm1 <- stan_glm(Better ~ Treatment + Sex + Age,
                  data = arthritis,
                  family = binomial)
```

The default coefficient priors are the same as those used when performing multiple regression. The intercept has location $= 0$.

Interpreting the coefficients

Recall Bayesian modeling does not return point estimates for the coefficients (the betas) but rather distributions. To get a coefficient value, we typically take the median or the mean of the posterior distribution.

The `coef` function returns the median values.

```
coef(bglm1)
```

```
##       (Intercept) TreatmentTreated           SexMale
##       -3.11556378        1.74090082       -1.45083163
```

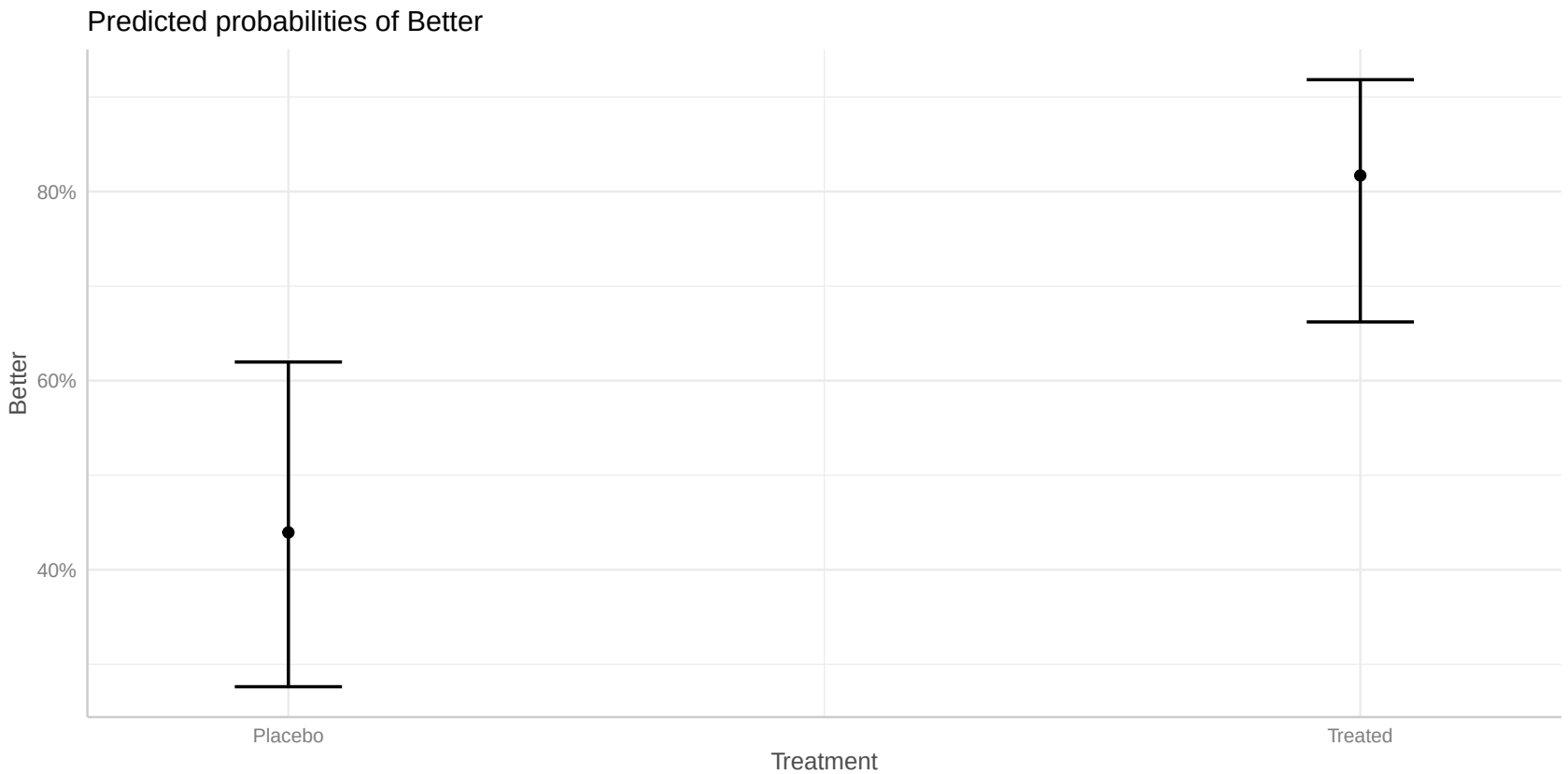
Exponentiating the coefficient value returns an odds ratio.

The odds that `Better = 1` about 6 times higher for the Treated group: $\exp(1.74) \approx 5.7$

Using effect plots with logistic regression models

An effect plot can help communicate a model in terms of probability.

```
ggpredict(bglm1, terms = "Treatment") |> plot()
```



Let's go to R!

Model comparison

Let's say we have the following model:

$$y = \beta_0 + \beta_1 x1 + \beta_2 x2$$

Do we need $x2$? Maybe the following model is just as "good"?

$$y = \beta_0 + \beta_1 x1$$

Or maybe a model with just $x2$ is better than a model with just $x1$.

$$y = \beta_0 + \beta_2 x2$$

Model comparison

In traditional statistics, models are often compared using hypothesis tests or information criteria, such as AIC.

In Bayesian statistics, the Pareto Smoothed Importance-Sampling Leave-One-Out cross-validation (PSIS-LOO) is often used.

It is relatively easy to implement with the `loo()` and `loo_compare()` functions in the `rstanarm` package.

loo_compare output

The output reports the difference in expected log predictive density (ELPD) along with the standard error of the difference.

```
loo_compare(loo(m1), loo(m2))
```

```
##    elpd_diff se_diff
## m1  0.0       0.0
## m2 -1.7       3.0
```

The first model listed has the largest ELPD. Each subsequent model is compared to the first model.

The standard error of the difference, `se_diff`, gives us some idea of how certain that difference is. If `se_diff` is bigger than `elpd_diff`, then we shouldn't be so sure the first model is necessarily "better".

ELPD

The expected log predictive density (ELPD) is basically a scoring rule to assess how well a model predicts new data.

When comparing models, models with higher ELPD are closer to the "true" data generating process.

Remember, "expected" is the key word. These are just estimates. Pay attention to the standard error of the difference (`se_diff`)

Let's go to R!

Some journal articles using `rstanarm`

Espe, M. et al. (2016) Yield gap analysis of US rice production systems shows opportunities for improvement. *Field Crops Research*, 196:276-283.

Kubrak, O. et al. (2017) Adaptation to fluctuating environments in a selection experiment with Drosophila melanogaster. *Ecology and Evolution*, 7:3796-3807.

Herzog, S. et al. (2017) Sun Protection Factor Communication of Sunscreen Effectiveness. *JAMA Dermatology*, 153(3):348-350.

Kovic, M. and Hänsli, N. (2017) The impact of political cleavages, religiosity, and values on attitudes towards nonprofit organizations. *Social Sciences*, 7(1), 2.

References

McElreath, M. (2016). *Statistical Rethinking*. CRC Press. Boca Raton.

Muth, C., Oravecz, Z., & Gabry, J. (2018). User-friendly Bayesian regression modeling: A tutorial with rstanarm and shinystan. *The Quantitative Methods for Psychology*, 14(2), 99-119.

Vehtari, A., Gelman, A., and Gabry, J. (2017). Practical Bayesian model evaluation using leave-one-out cross-validation and WAIC. *Statistics and Computing*. 27(5), 1413–1432.

Vehtari, A., Gelman, A., and Hwang, J. (2014) Understanding predictive information criteria for Bayesian models. *Statistics and Computing*. 24(6), 997-106.

Nicenboim, B., Schad, D. , and Vasishth, S. (2021) An Introduction to Bayesian Data Analysis for Cognitive Science. https://vasishth.github.io/bayescogsci/book/

`rstanarm` web site: http://mc-stan.org/rstanarm/

Thanks for coming

- For statistical consulting: statlab@virginia.edu
- Sign up for more workshops or see past workshops: http://data.library.virginia.edu/training/
- Register for the Research Data Services newsletter to be notified of new workshops: http://data.library.virginia.edu/newsletters/