# Linear Models with the rstanarm R Package

Ben Goodrich

April 13, 2020

# Difficulty of Analytical Bayesian Inference

- Bayes Rule for an unknown parameter (vector) $\boldsymbol{\theta}$ conditional on known data (vector) $\mathbf{y}$ can be written as

$$f\left(\boldsymbol{\theta} \mid \mathbf{y}\right) = \frac{f\left(\boldsymbol{\theta}\right) f\left(\mathbf{y} \mid \boldsymbol{\theta}\right)}{f\left(\mathbf{y}\right)} = \frac{f\left(\boldsymbol{\theta}\right) f\left(\mathbf{y} \mid \boldsymbol{\theta}\right)}{\int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \dots \int_{-\infty}^{\infty} f\left(\boldsymbol{\theta}\right) f\left(\mathbf{y} \mid \boldsymbol{\theta}\right) d\theta_1 d\theta_2 \dots d\theta_K}$$

- To obtain the denominator of Bayes Rule, you would need to do an integral
- The Risch Algorithm tells you if an integral has an elementary form (rare)
- In most cases, we can't write the denominator of Bayes Rule in a useful form
- But we can draw from a distribution whose PDF is characterized by the numerator of Bayes Rule without knowing the denominator

# Four Ways to Execute Bayes Rule

1. Draw from the prior predictive distribution and keep realizations of the parameters iff the realization of the outcome matches the observed data
   - Very intuitive what is happening but is only possible with discrete outcomes and only feasible with few observations and parameters
2. Numerically integrate the numerator of Bayes Rule over the parameter(s)
   - Most similar to what we did in the discrete case but is only feasible when there are few parameters and can be inaccurate even with only one
3. Analytically integrate the numerator of Bayes Rule over the parameter(s)
   - Makes incremental Bayesian learning obvious but is only possible in simple models when the distribution of the outcome is in the exponential family
4. Use MCMC to sample from the posterior distribution
   - Stan works for any posterior PDF that is differentiable w.r.t. the parameters

# Comparing Stan to Ancient MCMC Samplers

- Like M-H, only requires user to specify numerator of Bayes Rule
- Like M-H but unlike Gibbs sampling, proposals are joint
- Unlike M-H but like Gibbs sampling, proposals always accepted
- Unlike M-H but like Gibbs sampling, tuning of proposals is (often) not required
- Unlike both M-H and Gibbs sampling, the effective sample size is typically 25% to 125% of the nominal number of draws from the posterior distribution because $\rho_1$ can be negative in $n_{eff} = \frac{S}{1+2\sum_{k=1}^{\infty} \rho_k}$
- Unlike both M-H and Gibbs sampling, Stan produces warning messages when things are not going swimmingly. Do not ignore these!
- Unlike BUGS, Stan does not permit discrete unknowns but even BUGS has difficulty drawing discrete unknowns with a sufficient amount of efficiency

# Linear Model

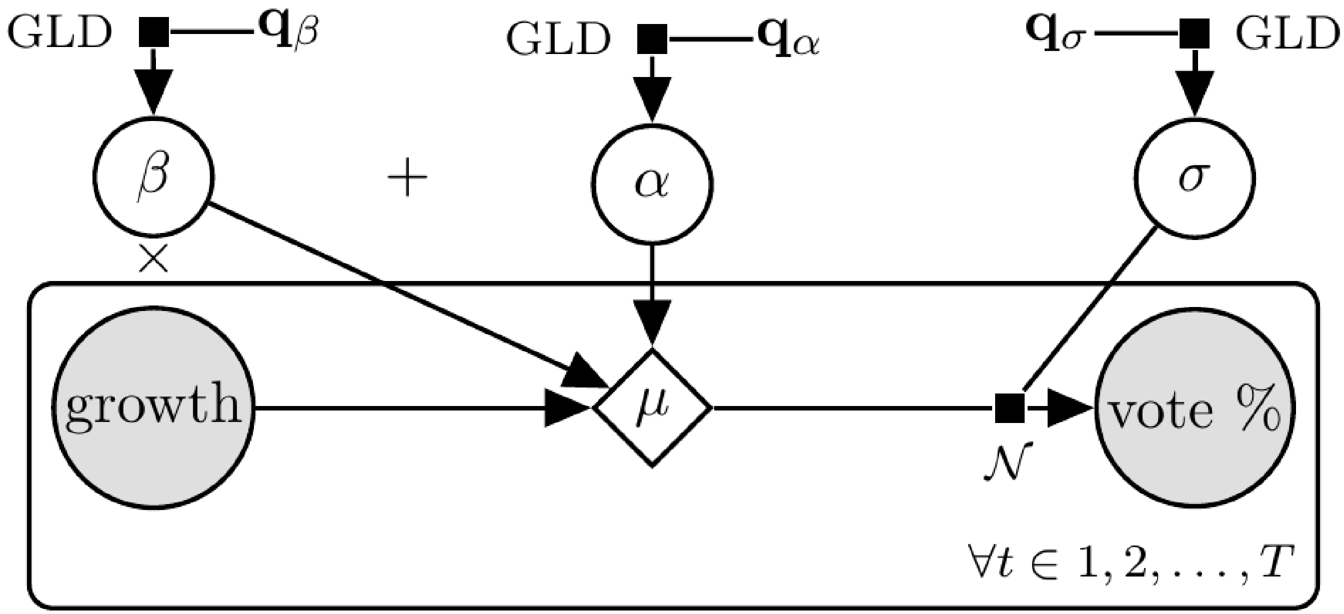
The prior predictive distribution for a linear model proceeds like

$$
\begin{aligned}
\alpha &\sim ???\\
\boldsymbol{\beta} &\sim ???\\
\forall n: \mu_n &= \alpha + \sum_{k=1}^{K} \beta_k x_{nk}\\
\sigma &\sim ???\\
\forall n: \epsilon_n &\sim \mathcal{N}\left(0, \sigma\right)\\
\forall n: y_n &= \mu_n + \epsilon_n
\end{aligned}
$$

where `???` indicates the parameter to the left is drawn from a distribution that is up to you.

# Hibbs Bread Model for Presidential Vote %

- What is the relationship between growth and incumbent party vote share?




Hibbs Model

# Breakout Rooms

Use R to draw $S = 10000$ times (using `replicate`) from the prior predictive distribution of the Hibbs model with reasonable GLD priors, which require

```
rstan::expose_stan_functions(file.path("..", "Week2", "quantile_functions.stan")) # GLD_icdf
source(file.path("..", "Week2", "GLD_helpers.R")) # GLD_solver and GLD_solver_bounded
ROOT <- "https://raw.githubusercontent.com/avehtari/ROS-Examples/master/"
hibbs <- readr::read_delim(paste0(ROOT, "ElectionsEconomy/data/hibbs.dat"), delim = " ")
hibbs$growth <- hibbs$growth - mean(hibbs$growth) # centering
y_ <- t(replicate(10000, {
  # fill in this part
}))
```

# Answer: Hyperparameters of GLD Priors

# Answer: Prior Predictive Distribution

# Checking the Prior Predictive Distribution

# Tthe **stan_glm** Function in the rstanarm Package

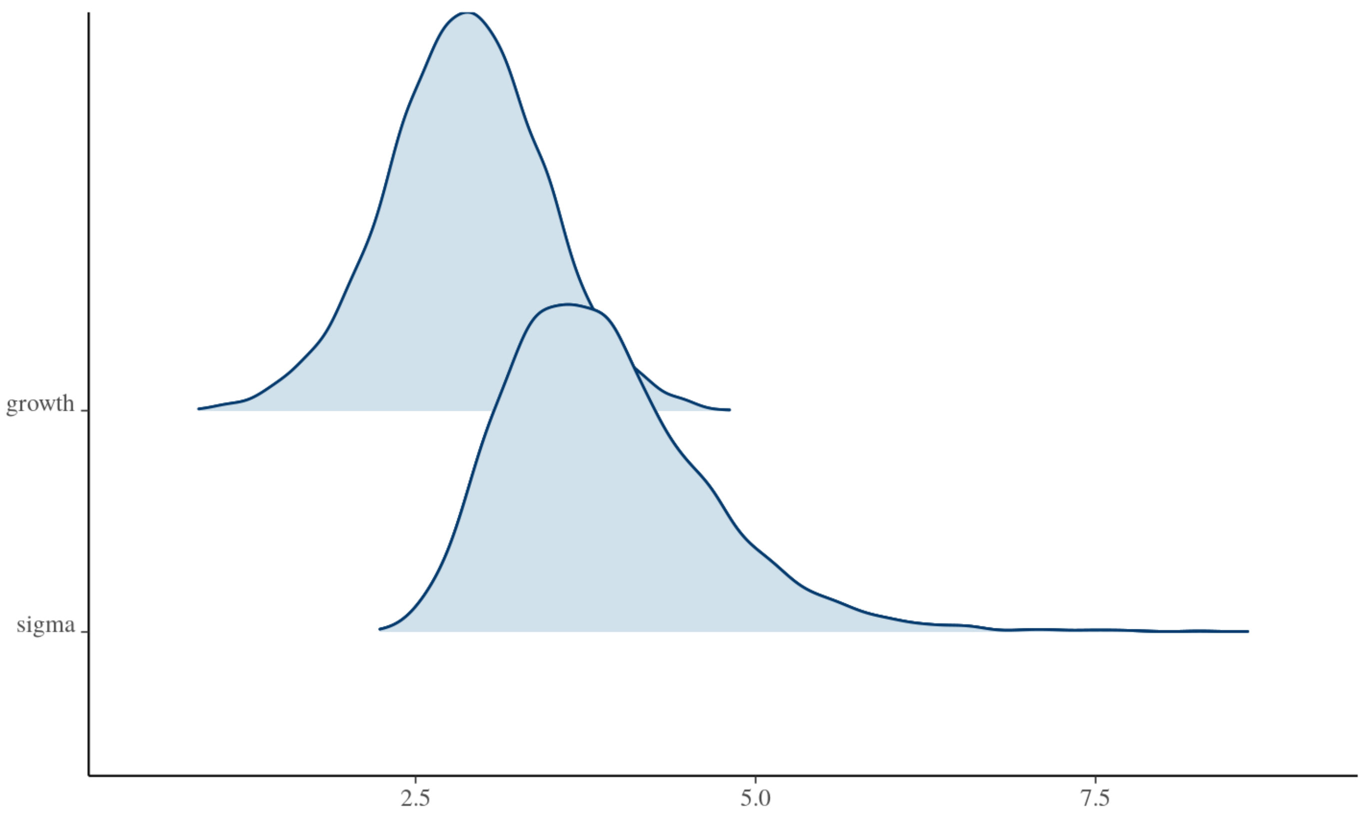
```r
options(mc.cores = parallel::detectCores()) # use all the cores on your computer
post <- stan_glm(inc.party.vote ~ growth, data = hibbs, family = gaussian,
                 prior_intercept = normal(location = 52, scale = 2, autoscale = FALSE),
                 prior = normal(location = 2.5, scale = 1, autoscale = FALSE),
                 prior_aux = exponential(rate = 0.25, autoscale = FALSE)) # expectation of 4
```

```r
post
```

```
...
## ------
##             Median MAD_SD
## (Intercept) 52.0    0.9
## growth       2.9    0.6
##
## Auxiliary parameter(s):
##       Median MAD_SD
## sigma 3.8    0.7
##
## Sample avg. posterior predictive distribution of y:
##          Median MAD_SD
## mean_PPD 52.0    1.3
##
## ------
```

# Plotting the Marginal Posterior Densities

```
plot(post, plotfun = "areas_ridges", pars = c("growth", "sigma")) # exclude the intercept
```



# Credible Intervals and $R^2$

```
round(posterior_interval(post, prob = 0.8), digits = 2)
```

```
##               10%   90%
## (Intercept) 50.86 53.21
## growth       2.13  3.59
## sigma        3.04  4.89
```

```
summary(bayes_R2(post))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.06705 0.46417 0.54757 0.52539 0.60778 0.70142
```

# Sampling Distribution of OLS vs. Posterior Kernel

```
functions { /* saved as OLS_rng.stan*/
  matrix OLS_rng(int S, real alpha, real beta,
                 real sigma, vector x) {
    matrix[S, 3] out; int N = rows(x);
    vector[N] x_ = x - mean(x);
    vector[N] mu = alpha + beta * x_;
    real SSX = sum(square(x_)); int Nm2 = N - 2;
    for (s in 1:S) {
      vector[N] y = to_vector(normal_rng(mu, sigma));
      real a = mean(y);
      real b = sum(y .* x_) / SSX;
      vector[N] residuals = y - (a + b * x_);
      real s2_hat = sum(square(residuals)) / Nm2;
      out[s, ] = [a, b, s2_hat];
    }
    return out;
  }
}
```

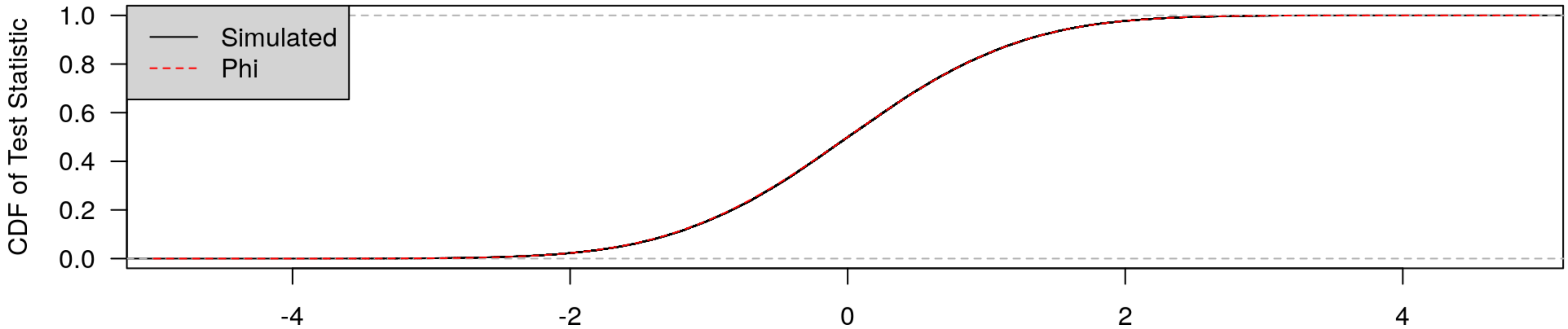
```
functions { /* saved as lm_kernel.stan*/
  real lm_kernel(real alpha, real beta, real tau,
                 vector y, vector x) {
    int N = rows(x);
    vector[N] x_ = x - mean(x);
    vector[N] mu = alpha + beta * x_;

    // alpha and beta have improper priors ...
    // ... so they add nothing to the log-kernel
    //        vvv inv_sqrt(tau) = 1 / sqrt(tau)
    real sigma = inv_sqrt(tau);
    return -log(tau) // Jeffreys prior on tau
           + normal_lpdf(y | mu, sigma);
           // ^^^ log-likelihood of parameters
  }
}
```

# Normal Distribution of the True Test Statistic

```r
rstan::expose_stan_functions("OLS_rng.stan"); x <- lfactorial(0:16); alpha <- 0
beta <- 0.5; sigma <- 10; OLS <- OLS_rng(S = 10 ^ 5, alpha, beta, sigma, x); colMeans(OLS)
```

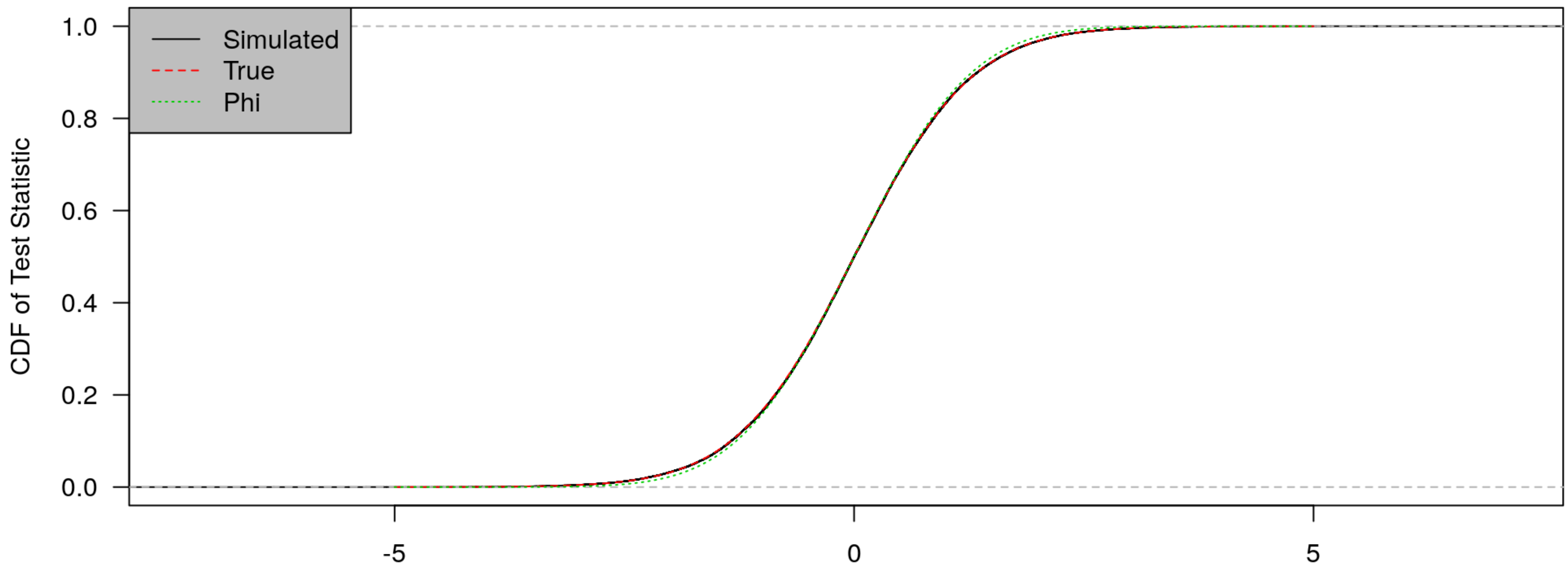
```
## [1]  -0.002718179   0.500792448 100.012222281
```

```r
se <- sqrt(sigma ^ 2 / sum( (x - mean(x)) ^ 2 )) # true standard error of estimated slope
plot(ecdf((OLS[ , 2] - beta) / se), main = "", xlab = "", ylab = "CDF of Test Statistic")
curve(pnorm(z), from = -5, to = 5, lty = 2, col = 2, add = TRUE, xname = "z")
legend("topleft", legend = c("Simulated", "Phi"), col = 1:2, lty = 1:2, bg = "lightgrey")
```



# Student t Distribution of Estimated Test Statistic

```r
se_hat <- sqrt(OLS[ , 3] / sum( (x - mean(x)) ^ 2 )) # estimated standard error of estimate
plot(ecdf((OLS[ , 2] - beta) / se_hat), main = "", xlab = "", ylab =  "CDF of Test Statistic")
curve(pt(z, df = 17 - 2), from = -5, to = 5, lty = 2, col = 2, add = TRUE, xname = "z")
curve(pnorm(z), from = -5, to = 5, lty = 3, col = 3, add = TRUE, xname = "z")
legend("topleft", legend = c("Simulated", "True", "Phi"), col = 1:3, lty = 1:3, bg = "grey")
```



# Power of the Test that $\beta = 0$ against $\beta > 0$

```
round(x, digits = 4)
```

```
##  [1]  0.0000  0.0000  0.6931  1.7918  3.1781  4.7875  6.5793  8.5252 10.6046 12.8018
## [11] 15.1044 17.5023 19.9872 22.5522 25.1912 27.8993 30.6719
```

```
mean( (OLS[ , 2] - 0) / se_hat > qt(0.95, df = 17 - 2) )
```

```
## [1] 0.62395
```

In other words, for THESE $17$ values of $x$, we EXPECT (over $Y$) to reject the false null hypothesis that $\beta = 0$ in favor of the alternative hypothesis that $\beta > 0$ at the 5% level with probability $0.624$ when the true value of $\beta$ is $\frac{1}{2}$.

- What good is this PRE-DATA (on $y_1, y_2, \dots, y_{17}$) statement?
- But in this case the posterior distribution is the same as the estimated sampling distribution of the OLS estimator across datasets

# Breakout Rooms: IQ of Three Year Olds

- All examples from the reading (plus more) are available at https://github.com/avehtari/RAOS-Examples
- At 36 months, kids were given an IQ test
- Suppose the conditional expectation is a linear function of whether its mother graduated high school and the IQ of the mother
- In breakout rooms, draw from the prior predictive distribution of the outcome using independent normal priors on the intercept and coefficients and an exponential prior on $\sigma$

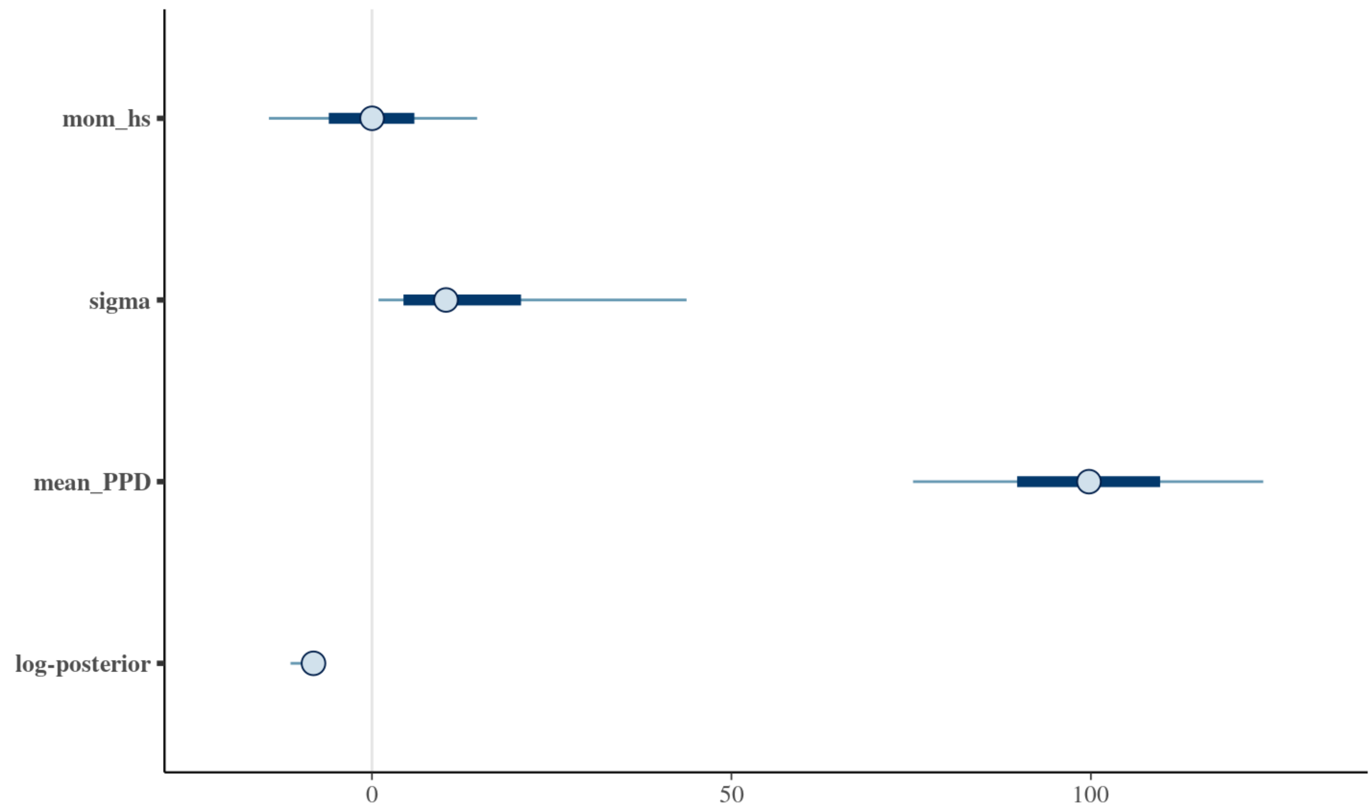
```
data(kidiq, package = "rstanarm")
colnames(kidiq) # remember to center
```

```
## [1] "kid_score" "mom_hs"    "mom_iq"    "mom_age"
```

# Answer

```
prior_PD <- posterior_predict(priors)
dim(prior_PD); plot(priors, regex_pars = "^[^(Intercept)]") # exclude intercept
```

```
## [1] 4000  434
```



# What Does `QR = TRUE` Do?

- Let the vector of linear predictions in a GLM be $\boldsymbol{\eta} = \mathbf{X}\beta$
- If we apply the QR decomposition to the linear predictor,

$$\boldsymbol{\eta} = \overbrace{\mathbf{QR}}^{\mathbf{X}}\beta = \mathbf{Q}\overbrace{\boldsymbol{\theta}}^{\mathbf{R}\beta}$$

- When you specify `QR = TRUE` in `stan_glm` (or use `stan_lm` or `stan_polr`), **rstanarm** internally does a GLM using $\mathbf{Q}$ as the matrix of predictors instead of $\mathbf{X}$ to get the posterior distribution of $\boldsymbol{\theta}$ and then pre-multiplies each posterior draw of $\boldsymbol{\theta}$ by $\mathbf{R}^{-1}$ to get a posterior draw of $\boldsymbol{\beta} = \mathbf{R}^{-1}\boldsymbol{\theta}$
- Doing so makes it easier for NUTS to sample from the posterior distribution (of $\boldsymbol{\theta}$) efficiently because the columns of $\mathbf{Q}$ are orthogonal, whereas the columns of $\mathbf{X}$ are not

# Drawing from the Posterior Distribution

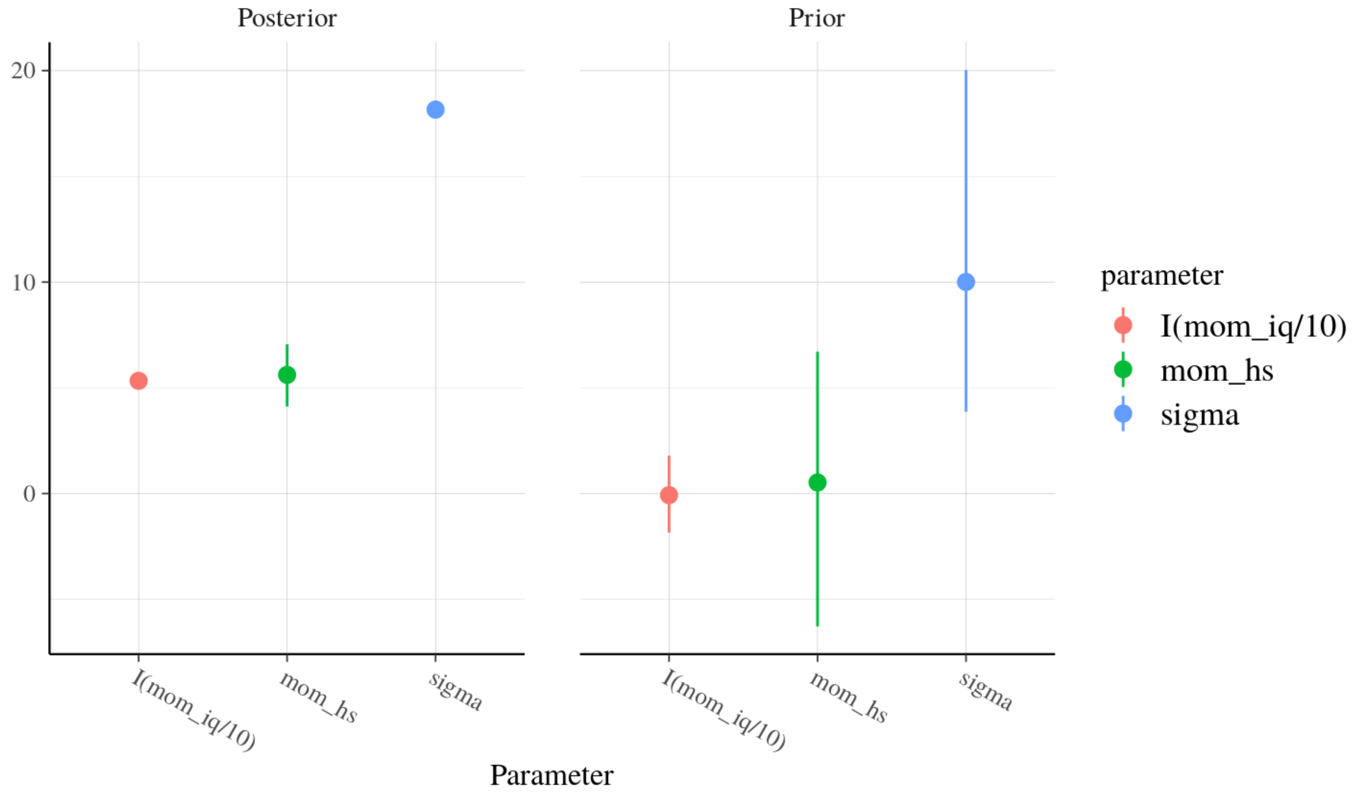
```r
post <- update(priors, prior_PD = FALSE)
```

```r
summary(post)
```

```
...
##                 mean    sd      2.5%    25%     50%     75%     97.5%
## (Intercept)       29.1     5.8    17.9    25.2    29.1    33.1    40.2
## mom_hs             5.6     2.2     1.4     4.1     5.6     7.1     9.8
## I(mom_iq/10)       5.3     0.6     4.2     4.9     5.3     5.7     6.5
## sigma             18.2     0.6    17.0    17.8    18.2    18.6    19.4
## mean_PPD          86.9     1.3    84.4    86.0    86.9    87.7    89.3
## log-posterior -1884.2     1.4 -1887.7 -1884.9 -1883.9 -1883.1 -1882.4
##
## Diagnostics:
##               mcse Rhat n_eff
## (Intercept)   0.1  1.0  5211
## mom_hs        0.0  1.0  4799
## I(mom_iq/10)  0.0  1.0  5009
## sigma         0.0  1.0  5212
## mean_PPD      0.0  1.0  4142
## log-posterior 0.0  1.0  1726
##
```

# Posterior vs. Prior

```r
posterior_vs_prior(post, prob = 0.5, regex_pars = "^[^(]") # excludes (Intercept)
```



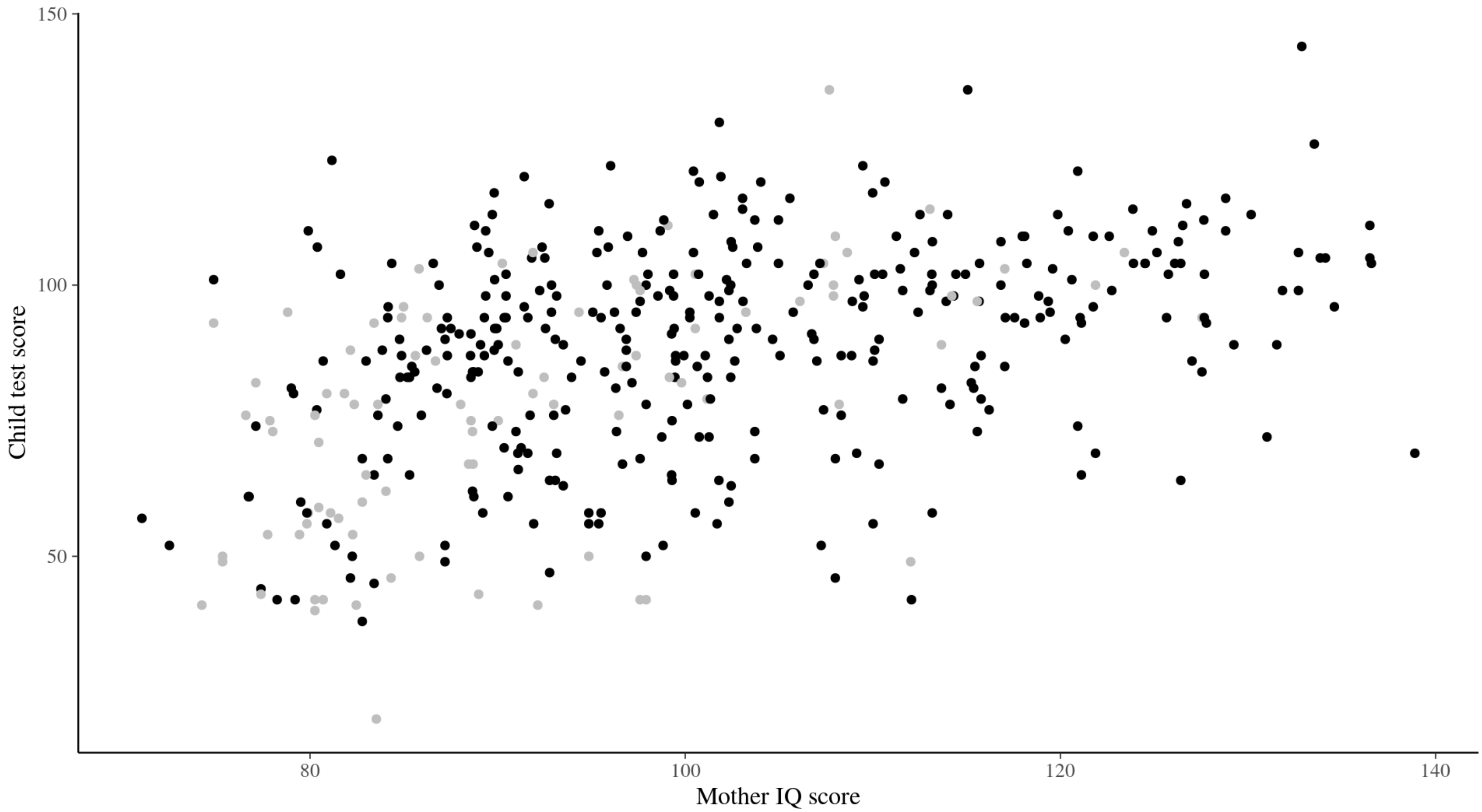
# ShinyStan

- ShinyStan can be launched on an object produced by rstanarm via

```
launch_shinystan(post)
```

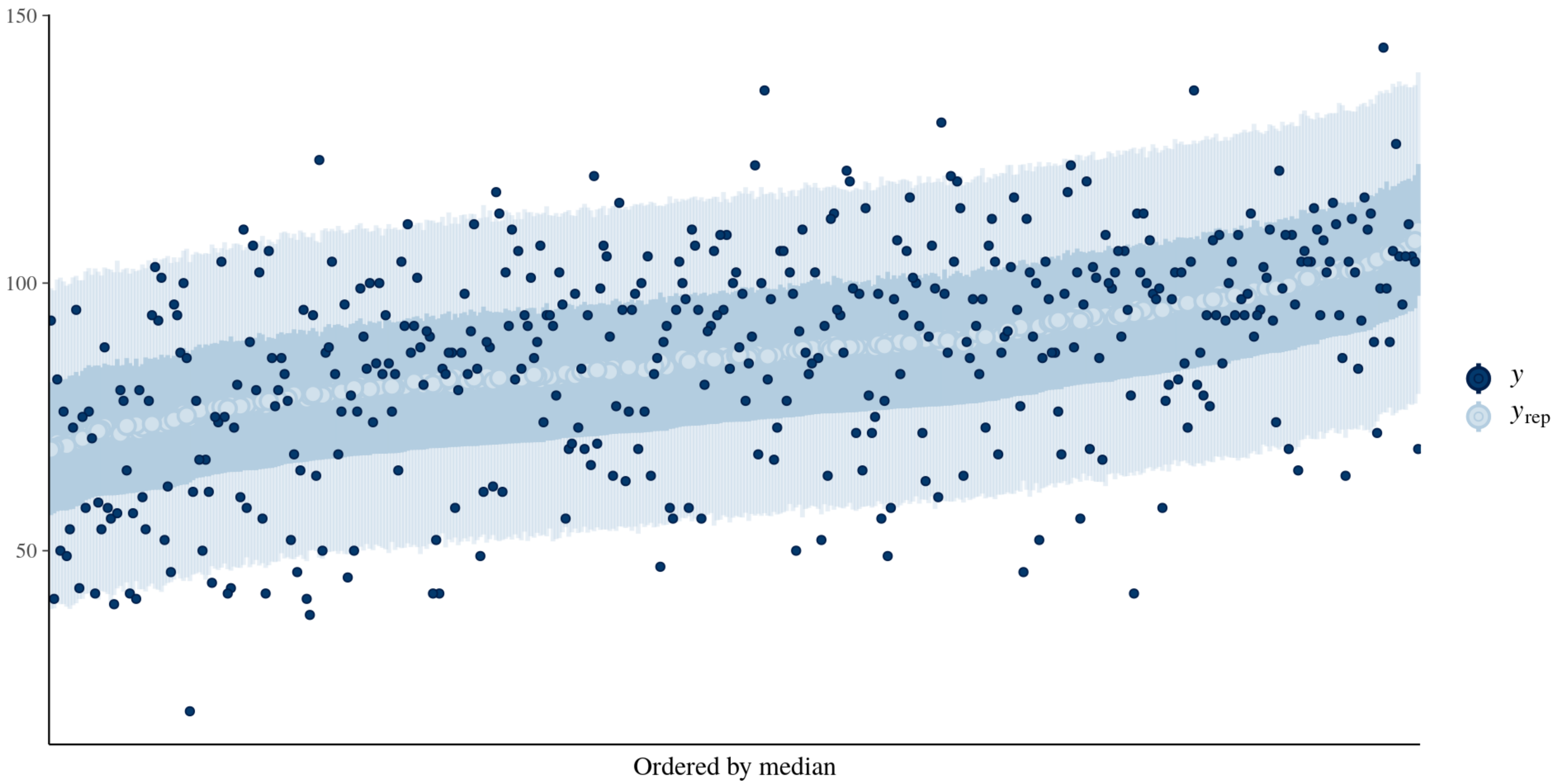
- A webapp will open in your default web browser that helps you visualize the posterior distribution and diagnose problems

# Plot at the Posterior Median Estimates



# Correct Plot

```
pp_check(post, plotfun = "loo_intervals", order = "median")
```



# Utility Function for Predictions of Future Data

- For Bayesians, the log predictive PDF is the most appropriate utility function
- Choose the model that maximizes the expectation of this over FUTURE data

$$\text{ELPD} = \mathbb{E}_Y \ln f\left(y_{N+1}, y_{N+2}, \dots, y_{2N} \mid y_1, y_2, \dots, y_N\right) =$$

$$\ln \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f\left(y_{N+1}, y_{N+2}, \dots, y_{2N} \mid \mathbf{y}\right) dy_{N+1} dy_{N+2} \dots dy_{2N} \approx$$

$$\sum_{n=1}^{N} \ln f\left(y_n \mid \mathbf{y}_{-n}\right) = \sum_{n=1}^{N} \ln \int_{\Theta} f\left(y_n \mid \boldsymbol{\theta}\right) f\left(\boldsymbol{\theta} \mid \mathbf{y}_{-n}\right) d\theta_1 d\theta_2 \dots d\theta_K$$

- $f\left(y_n \mid \boldsymbol{\theta}\right)$ is just the $n$-th likelihood contribution, but can we somehow obtain $f\left(\boldsymbol{\theta} \mid \mathbf{y}_{-n}\right)$ from $f\left(\boldsymbol{\theta} \mid \mathbf{y}\right)$?
- Yes, assuming $y_n$ does not have an outsized influence on the posterior

# Pareto Smoothed Importance Sampling

- Let $r_n^s = \frac{1}{f\left(y_n | \widehat{\boldsymbol{\theta}^s}\right)} \propto \frac{f\left(\widehat{\boldsymbol{\theta}^s} | \mathbf{y}_{-n}\right)}{f\left(\widehat{\boldsymbol{\theta}^s} | \mathbf{y}\right)}$ be the $s$-th importance ratio for $y_n$
- Fit a generalized Pareto model to these importance ratios, which have PDF

$$f\left(r_n \mid \mu, \sigma, k\right) = \frac{1}{\sigma}\left(1 + \frac{k\left(\mu - r_n\right)}{\sigma}\right)^{-1-\frac{1}{k}}$$

- In the 20% right tail, use an interpolated $\hat{r}_n^s$ rather than $r_n^s$
- Doing so stabilizes the variances as long as the estimated shape parameters of the generalized Pareto distribution are not too large
  - $\hat{k}_n < 0.5$ is good and $\hat{k}_n \in [0.5, 0.7]$ is okay
  - $\hat{k}_n > 0.7$ is bad and $\hat{k}_n > 1.0$ is very bad

# PSISLOOCV with the Kid IQ Example

```
loo(post)
```

```
##
## Computed from 4000 by 434 log-likelihood matrix
##
##          Estimate   SE
## elpd_loo  -1876.2 14.2
## p_loo         4.0  0.4
## looic      3752.3 28.5
## ------
## Monte Carlo SE of elpd_loo is 0.0.
##
## All Pareto k estimates are good (k < 0.5).
## See help('pareto-k-diagnostic') for details.
```

# Model with Interaction Term

```
interaction <- update(post, formula. = . ~ . + mom_hs:mom_iq)
```

```
compare_models(flat = loo(post), interaction = loo(interaction))
```

```
##
## Model comparison:
## (negative 'elpd_diff' favors 1st model, positive favors 2nd)
##
## elpd_diff        se
##       3.6       3.0
```

# Data on Diamonds

```
data("diamonds", package = "ggplot2")
diamonds <- diamonds[diamonds$z > 0, ] # probably mistakes in the data
str(diamonds)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    53920 obs. of  10 variables:
##  $ carat  : num  0.23 0.21 0.23 0.29 0.31 0.24 0.24 0.26 0.22 0.23 ...
##  $ cut    : Ord.factor w/ 5 levels "Fair"<"Good"<..: 5 4 2 4 2 3 3 3 1 3 ...
##  $ color  : Ord.factor w/ 7 levels "D"<"E"<"F"<"G"<..: 2 2 2 6 7 7 6 5 2 5 ...
##  $ clarity: Ord.factor w/ 8 levels "I1"<"SI2"<"SI1"<..: 2 3 5 4 2 6 7 3 4 5 ...
##  $ depth  : num  61.5 59.8 56.9 62.4 63.3 62.8 62.3 61.9 65.1 59.4 ...
##  $ table  : num  55 61 65 58 58 57 57 55 61 61 ...
##  $ price  : int  326 326 327 334 335 336 336 337 337 338 ...
##  $ x      : num  3.95 3.89 4.05 4.2 4.34 3.94 3.95 4.07 3.87 4 ...
##  $ y      : num  3.98 3.84 4.07 4.23 4.35 3.96 3.98 4.11 3.78 4.05 ...
##  $ z      : num  2.43 2.31 2.31 2.63 2.75 2.48 2.47 2.53 2.49 2.39 ...
```

- What do you think is the prior $R^2$ for a model of `log(price)`?

# Do This Once on Each Computer You Use

- R comes with a terrible default coding for ordered factors in regressions known as “Helmert” contrasts
- Execute this once to change them to “treatment” contrasts, which is the conventional coding in the social sciences with dummy variables relative to a baseline category

```
cat('options(contrasts = c(unordered = "contr.treatment", ordered = "contr.treatment"))',
    file = "~/.Rprofile", sep = "\n", append = TRUE)
```

- Without this, you will get a weird rotation of the coefficients on the `cut` and `clarity` dummy variables
- `"contr.sum"` is another reasonable (but rare) choice

# The `stan_lm` Function

```
post <- stan_lm(log(price) ~ carat * (log(x) + log(y) + log(z)) + cut + color + clarity,
                data = diamonds, prior = R2(location = 0.8, what = "mode"), adapt_delta = 0.9)
```

```
str(as.data.frame(post), vec.len = 3, digits.d = 2)
```

```
## 'data.frame':    4000 obs. of  28 variables:
##  $ (Intercept)  : num  0.71 0.71 0.71 0.71 ...
##  $ carat        : num  7.3 7.5 7.3 7.4 ...
##  $ log(x)       : num  4.5 4.5 4.6 4.5 ...
##  $ log(y)       : num  -2.5 -2.4 -2.5 -2.4 ...
##  $ log(z)       : num  0.97 0.86 0.98 0.93 ...
##  $ cutGood      : num  0.083 0.086 0.09 0.079 ...
##  $ cutVery Good : num  0.12 0.13 0.13 0.12 ...
##  $ cutPremium   : num  0.13 0.14 0.14 0.13 ...
##  $ cutIdeal     : num  0.17 0.17 0.17 0.16 ...
##  $ colorE       : num  -0.054 -0.056 -0.052 -0.06 ...
##  $ colorF       : num  -0.093 -0.098 -0.092 -0.103 ...
##  $ colorG       : num  -0.16 -0.17 -0.16 -0.17 ...
##  $ colorH       : num  -0.26 -0.26 -0.25 -0.26 ...
##  $ colorI       : num  -0.37 -0.38 -0.37 -0.38 ...
##  $ colorJ       : num  -0.51 -0.51 -0.51 -0.52 ...
##  $ claritySI2   : num  0.42 0.42 0.41 0.42 ...
##  $ claritySI1   : num  0.58 0.59 0.58 0.59 ...
##  $ clarityVS2   : num  0.73 0.73 0.73 0.73 ...
##  $ clarityVS1   : num  0.8 0.8 0.8 0.81 ...
##  $ clarityVVS2  : num  0.93 0.94 0.93 0.93 ...
##  $ clarityVVS1  : num  1 1 1 1 ...
##  $ clarityIF    : num  1.1 1.1 1.1 1.1 ...
##  $ carat:log(x) : num  -3.9 -4 -3.9 -3.9 ...
##  $ carat:log(y) : num  1.9 1.9 1.9 1.8 ...
##  $ carat:log(z) : num  -1.1 -1 -1.1 -1 ...
##  $ sigma        : num  0.13 0.13 0.13 0.13 ...
##  $ log-fit_ratio: num  -0.00102 0.00107 -0.00123 -0.00015
##  $ R2           : num  0.98 0.98 0.98 0.98 ...
```

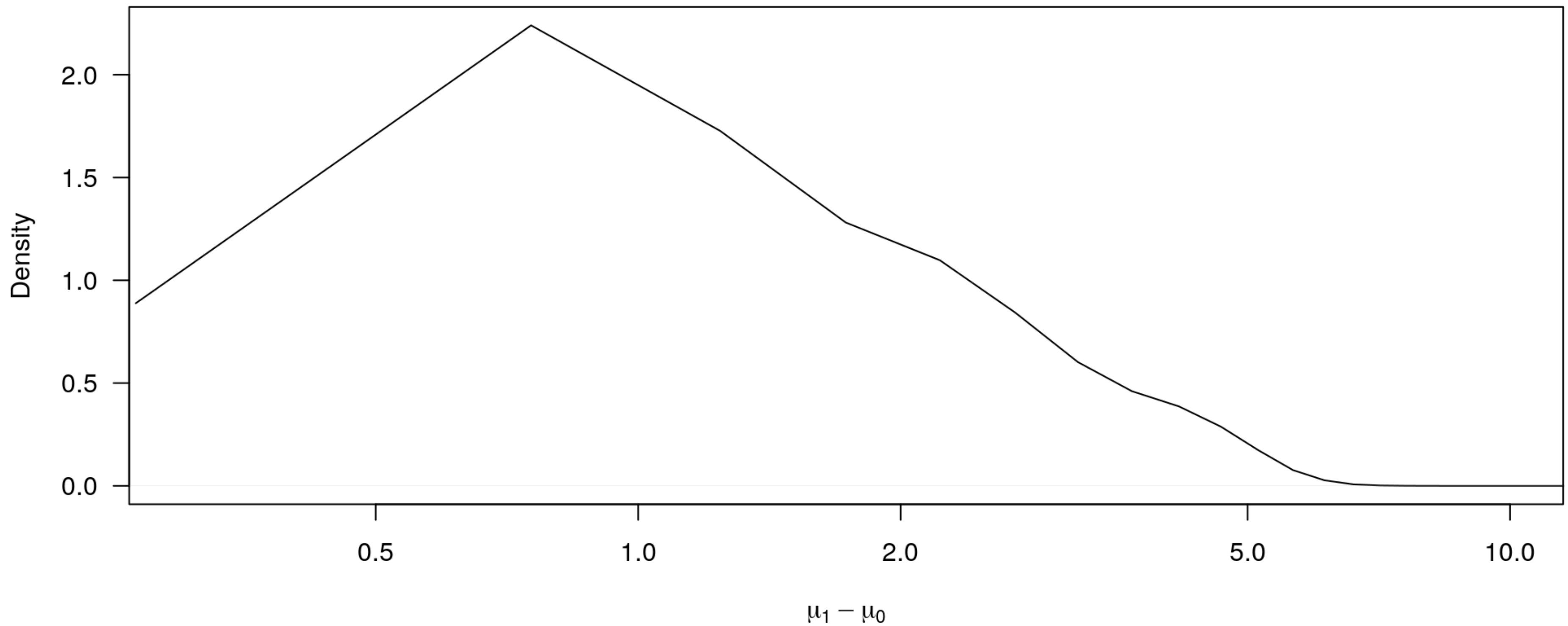
# Typical Output

```r
print(post, digits = 4)
```

```
...
##                 Median  MAD_SD
## (Intercept)      0.7655  0.0351
## carat            7.4180  0.0714
## log(x)           4.5248  0.0803
## log(y)          -2.5175  0.0728
## log(z)           0.9587  0.0416
## cutGood          0.0851  0.0037
## cutVery Good     0.1222  0.0035
## cutPremium       0.1353  0.0034
## cutIdeal         0.1665  0.0034
## colorE          -0.0552  0.0020
## colorF          -0.0962  0.0020
## colorG          -0.1629  0.0020
## colorH          -0.2573  0.0021
## colorI          -0.3750  0.0023
## colorJ          -0.5116  0.0030
## claritySI2       0.4165  0.0049
## claritySI1       0.5820  0.0049
## clarityVS2       0.7287  0.0049
## clarityVS1       0.8000  0.0050
## clarityVVS2      0.9307  0.0050
## clarityVVS1      1.0021  0.0053
## clarityIF        1.0971  0.0056
## carat:log(x)    -3.9644  0.0655
## carat:log(y)     1.9116  0.0541
## carat:log(z)    -1.1190  0.0429
##
## Auxiliary parameter(s):
##                 Median MAD_SD
## R2              0.9846 0.0001
## log-fit_ratio   0.0000 0.0005
## sigma           0.1257 0.0004
##
## Sample avg. posterior predictive distribution
##          Median MAD_SD
## mean_PPD 7.7864 0.0008
##
## ------
## * For help interpreting the printed output see
## * For info on the priors used see ?prior_summa
...
```

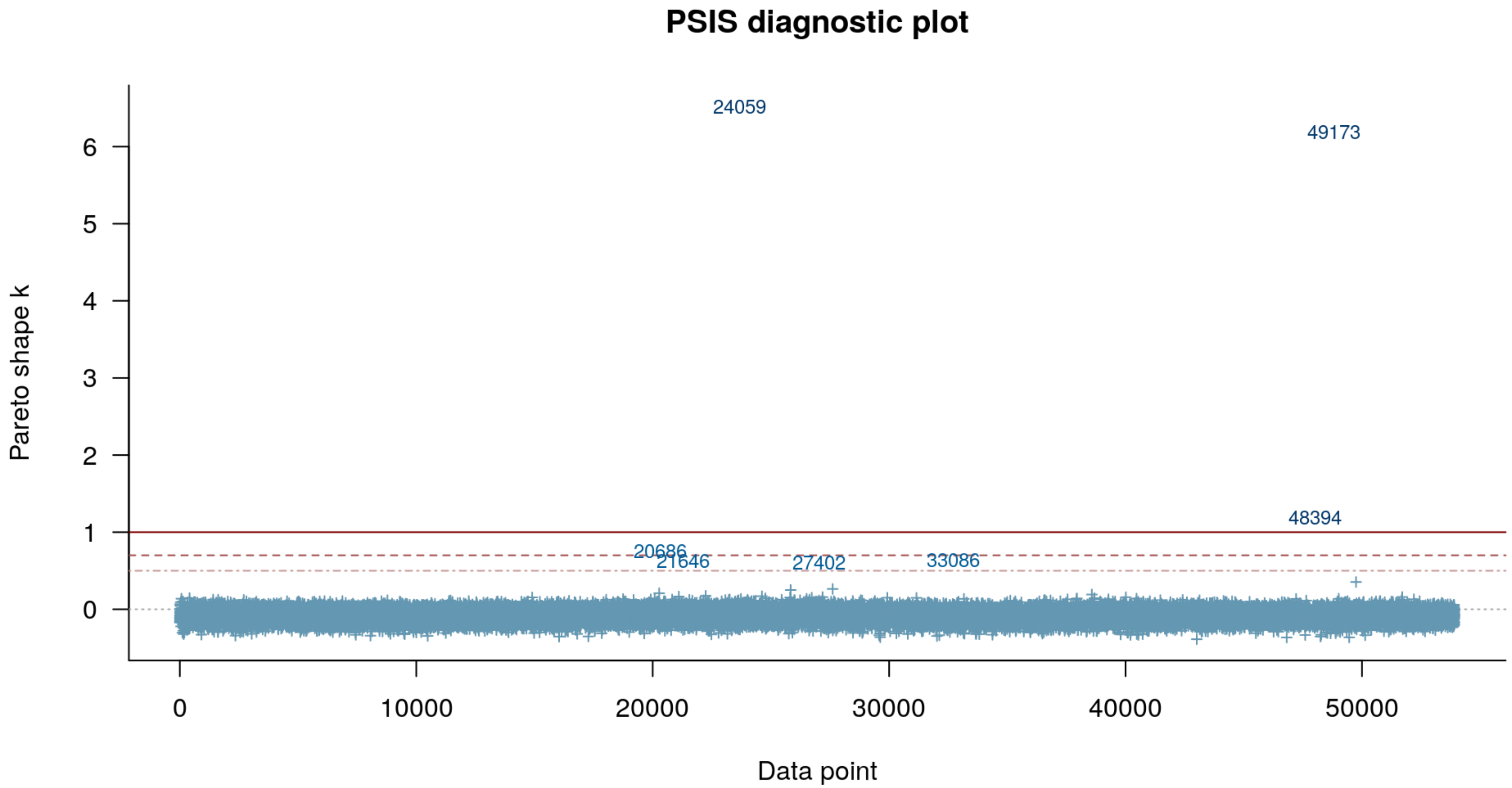
# What Is the Effect of an Increase in Carat?

```
mu_0 <- exp(posterior_linpred(post, draws = 500)) / 1000
df <- diamonds[diamonds$z > 0, ]; df$carat <- df$carat + 0.2
mu_1 <- exp(posterior_linpred(post, newdata = df, draws = 500)) / 1000
plot(density(mu_1 - mu_0), xlab = expression(mu[1] - mu[0]), xlim = c(.3, 10), log = "x", main
```



# But Wait

```r
plot(loo(post), label_points = TRUE)
```



# Why NUTS Is Better than Other MCMC Samplers

- With Stan, it is almost always the case that things either go well or you get warning messages saying things did not go well
- Because Stan uses gradients, it scales well as models get more complex
- The first-order autocorrelation tends to be negative so you can get greater effective sample sizes (for mean estimates) than nominal sample size

```r
round(bayesplot::neff_ratio(post), digits = 2)
```

```
##   (Intercept)         carat        log(x)        log(y)        log(z)       cutGood
##          0.32          0.74          1.02          1.18          0.75          1.21
##  cutVery Good    cutPremium      cutIdeal        colorE        colorF        colorG
##          1.14          1.13          1.21          1.09          0.93          0.94
##        colorH        colorI        colorJ    claritySI2    claritySI1    clarityVS2
##          0.77          1.08          1.11          0.93          0.95          0.91
##    clarityVS1   clarityVVS2   clarityVVS1     clarityIF  carat:log(x)  carat:log(y)
##          0.97          0.92          0.98          0.94          1.19          1.16
##  carat:log(z)         sigma log-fit_ratio            R2
##          0.53          0.50          1.05          0.48
```

# Divergent Transitions

- NUTS only uses first derivatives
- First order approximations to Hamiltonian physiscs are fine for if either the second derivatives are constant or the discrete step size is sufficiently small
- When the second derviatives are very not constant across $\Theta$, Stan can (easily) mis-tune to a step size that is not sufficiently small and $\theta_k$ gets pushed to $\pm\infty$
- When this happens there will be a warning message, suggesting to increase `adapt_delta`
- When `adapt_delta` is closer to 1, Stan will tend to take smaller steps
- Unfortunately, even as `adapt_delta` $\lim 1$, there may be no sufficiently small step size and you need to try to reparameterize your model

# Exceeding Maximum Treedepth

- When the step size is small, NUTS needs many (small) steps to cross the "typical" subset of $\Theta$ and hit the U-turn point
- Sometimes, NUTS has not U-turned when it reaches its limit of 10 steps (by default)
- When this happens there will be a warning message, suggesting to increase `max_treedepth`
- There is always a sufficiently high value of `max_treedepth` to allow NUTS to reach the U-turn point, but increasing `max_treedepth` by 1 approximately doubles the wall time to obtain $S$ draws

# Low Bayesian Fraction of Missing Information

- When the tails of the posterior PDF are very light, NUTS can have difficulty moving through $\Theta$ efficiently
- This will manifest itself in a low (and possibly unreliable) estimate of $n_{eff}$
- When this happens there will be a warning message, saying that the Bayesian Fraction of Missing Information (BFMI) is low
- In this situation, there is not much you can do except increase $S$ or preferably reparameterize your model to make it easier for NUTS

# Runtime Exceptions

- Sometimes you will get a “informational” (not error, not warning) message saying that some parameter that should be positive is zero or some parameter that should be finite is infinite
- This means that a 64bit computer could not represent the number accurately
- If it only happens a few times and only during the warmup phase, do not worry
- Otherwise, you might try to use functions that are more numerically stable, which is discussed throughout the Stan User Manual