# Generalized Linear Models with the rstanarm R Package

Ben Goodrich

April 08, 2019

# *Ex Ante* Probability Density / Mass Function

A likelihood function is the same expression as a P{D,M}F with 3 distinctions:

1. For the PDF or PMF, $f\left(x|\,\boldsymbol{\theta}\right)$, we think of $X$ as a random variable and $\boldsymbol{\theta}$ as given, whereas we conceive of the likelihood function, $\mathcal{L}\left(\boldsymbol{\theta};x\right)$, to be a function of $\boldsymbol{\theta}$ evaluted at the OBSERVED data, $x$
   - As a consequence, $\int\limits_{-\infty}^{\infty} f\left(x|\,\boldsymbol{\theta}\right)dx = 1$ or $\sum\limits_{i:x_i\in\Omega} f\left(x_i|\,\boldsymbol{\theta}\right) = 1$ while $\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\cdots\int_{-\infty}^{\infty}\mathcal{L}\left(\boldsymbol{\theta};x\right)d\theta_1 d\theta_2\dots d\theta_K$ is positive but not 1
2. We often think of "the likelihood function" for $N$ conditionally independent observations, so $\mathcal{L}\left(\boldsymbol{\theta};\mathbf{x}\right) = \prod_{n=1}^{N}\mathcal{L}\left(\boldsymbol{\theta};x_n\right)$
3. By "the likelihood function", we often really mean the natural logrithm thereof: $\ell\left(\boldsymbol{\theta};\mathbf{x}\right) = \ln\mathcal{L}\left(\boldsymbol{\theta},\mathbf{x}\right) = \sum_{n=1}^{N}\ln\mathcal{L}\left(\boldsymbol{\theta};x_n\right)$

- Fisher was concerned with the distribution of the mode across datasets

# Hamiltonian Monte Carlo

- Instead of simply drawing from the posterior distribution whose PDF is $f\left(\boldsymbol{\theta}|\,\mathbf{y}\right) \propto f\left(\boldsymbol{\theta}\right) L\left(\boldsymbol{\theta};\mathbf{y}\right)$ Stan augments the "position" variables $\boldsymbol{\theta}$ with an equivalent number of "momentum" variables $\boldsymbol{\phi}$ and draws from

$$f\left(\boldsymbol{\theta}|\,\mathbf{y}\right) \propto \int_{-\infty}^{\infty} \dots \int_{-\infty}^{\infty} \left[\prod_{k=1}^{K} \frac{1}{\sigma_k\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{\phi_k}{\sigma_k}\right)^2}\right] f\left(\boldsymbol{\theta}\right) L\left(\boldsymbol{\theta};\mathbf{y}\right) d\phi_1 \dots d\phi_K$$

- Since the likelihood is NOT a function of $\phi_k$, the posterior distribution of $\phi_k$ is the same as its prior, which is normal with a "tuned" standard deviation. So, at the $s$-th MCMC iteration, we just draw each $\widetilde{\phi}_k$ from its normal distribution.
- Using physics, the realizations of each $\widetilde{\phi}_k$ at iteration $s$ "push" $\boldsymbol{\theta}$ from iteration $s-1$ through the parameter space whose topology is defined by the negated log-kernel of the posterior distribution: $-\ln f\left(\boldsymbol{\theta}\right) - \ln L\left(\boldsymbol{\theta};\mathbf{y}\right)$

# No U-Turn Sampling (NUTS)

- The location of $\boldsymbol{\theta}$ moving according to Hamiltonian physics at any instant would be a valid draw from the posterior distribution
- But (in the absence of friction) $\boldsymbol{\theta}$ moves indefinitely so when do you stop?
- Can simulate Hamiltonian dynamics “forward” and “backward” in “time”
- Hoffman and Gelman (2014) proposed stopping the forward dynamics and the backward dynamics start to get closer together. Hence, the name No U-Turn Sampling.
- After the U-Turn, one footprint is selected with probability proportional to the posterior kernel to be the realization of $\boldsymbol{\theta}$ on iteration $s$ and the process repeates itself
- NUTS discretizes a continuous-time Hamiltonian process in order to solve a system of Ordinary Differential Equations (ODEs), which requires a stepsize that is also tuned during the warmup phase

# What is Stan?

- Includes a high-level probabilistic programming language
- Includes a translator of high-level Stan syntax to somewhat low-level C++
- Includes new (and old) gradient-based algorithms for statistical inference, such as NUTS
- Includes a matrix and scalar math library that supports autodifferentiation
- Includes interfaces from R and other high-level software
- Includes R packages with pre-written Stan programs
- Includes (not Stan specific) post-estimation R functions
- Includes a large community of users and many developers

# Using Stan via R

1. Write the program in a (text) .stan file w/ R-like syntax that ultimately defines a posterior log-kernel. We will not do this until May. Stan's parser, `rstan::stanc`, does two things
   - checks that program is syntactically valid and tells you if not
   - writes a conceptually equivalent C++ source file to disk
2. C++ compiler creates a binary file from the C++ source
3. Execute the binary from R (can be concurrent with 2)
4. Analyze the resulting samples from the posterior
   - Posterior predictive checks
   - Model comparison
   - Decision

- For the first several weeks, we are just going to focus on writing small functions in the Stan language

# 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_));
    for (s in 1:S) {
      vector[N] y; vector[N] e;
      for (n in 1:N)
        y[n] = mu[n] + normal_rng(0, sigma);
      out[s, 1] = mean(y);
      out[s, 2] = sum(y .* x_) / SSX;
      e = y - (out[s, 1] + out[s, 2] * x_);
      out[s, 3] = sum(square(e)) / (N - 2);
    }
    return out;
  }
}
```

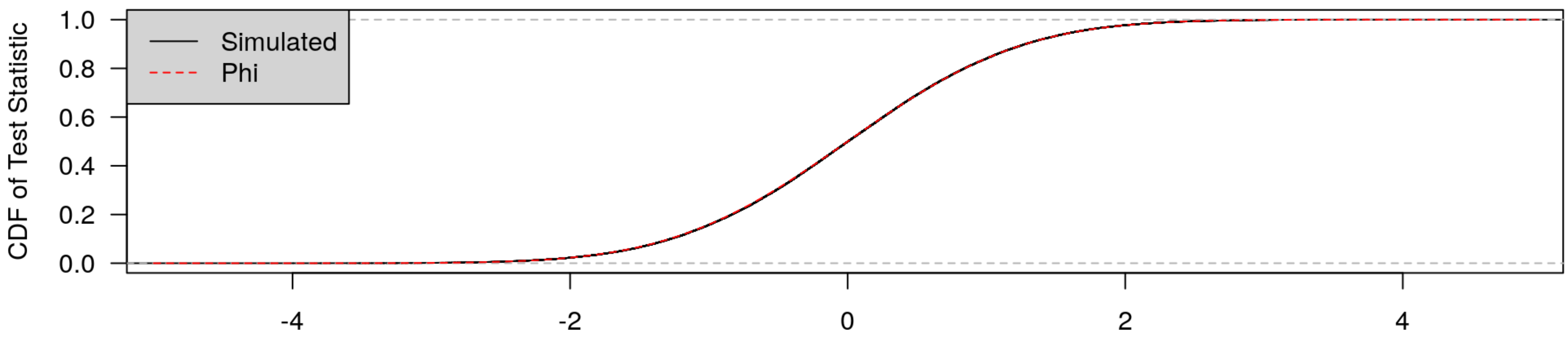
```
functions { /* saved as lm_kernel.stan*/
  real lm_kernel(real alpha, real beta, real tau,
                 vector y, vector x) {
    int N = rows(x);
    vector[N] x_ = x - mean(x);
    vector[N] mu = alpha + beta * x_;
    real sigma = inv_sqrt(tau);
    //        ^^^ inv_sqrt(tau) = 1 / sqrt(tau)
    // alpha and beta have improper priors ...
    // ... so they add nothing to the log-kernel
    return -log(tau) // Jeffreys prior on tau
           + normal_lpdf(y | mu, sigma);
           // ^^^ log-likelihood of parameters
  }
}
```

# Normal Distribution of the True Test Statistic

```
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)
```

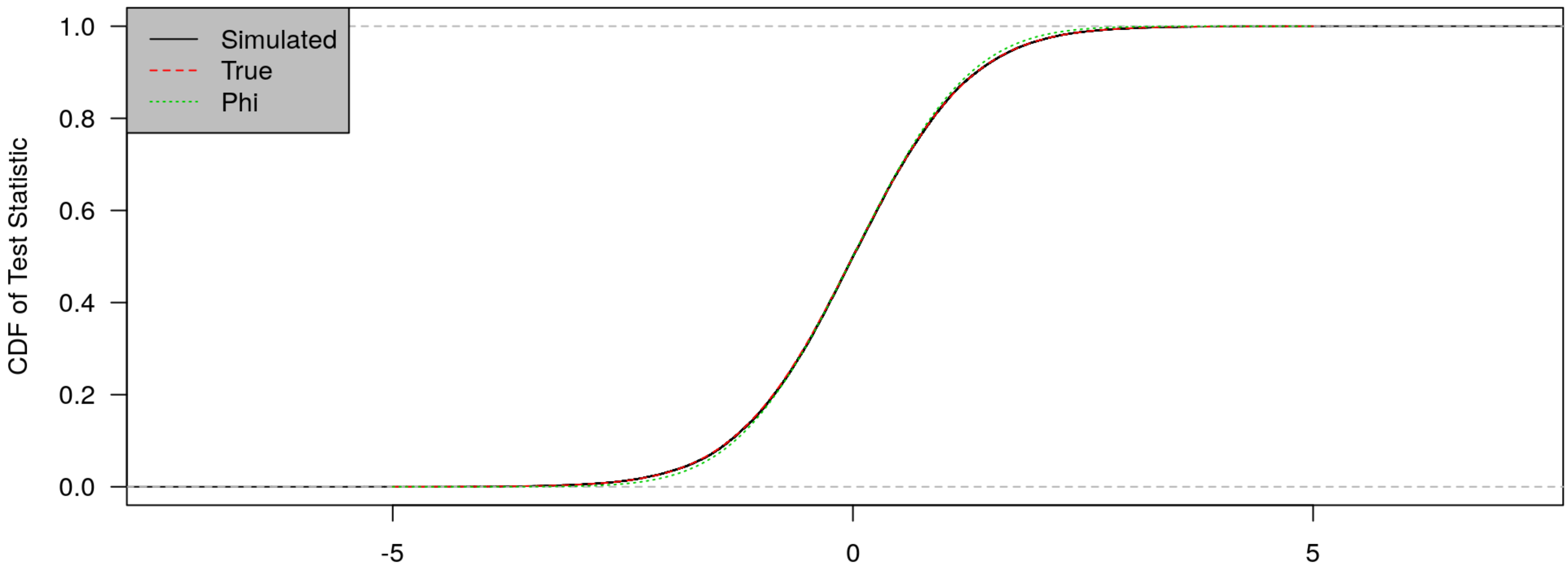
```
## [1]  -0.002718179   0.500792448 100.012222281
```

```
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

# Data on Diamonds

```
data("diamonds", package = "ggplot2")
str(diamonds)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    53940 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 ...
```

# Generative Models

- In order to draw from the prior predictive distribution, you have to have a model that you can simulate from
- Without a generative model, you cannot update your beliefs with Bayes Rule

| Concept | Known | Unknowable |
|---|---|---|
| **Exogenous** | sizes, predictors, prior modes / medians / etc. | parameters |
| **Endogenous** | outcomes | intermediates, predictions, utility |

- **Endogenous Known**: Log of diamond prices
- **Exogenous Knowns**: $N$, physical characteristics of diamonds
- **Exogenous Unknowables** : $\alpha$, $\boldsymbol{\beta}$, $R^2$
- **Endogenous Unknowable** : (counterfactual?) predicted prices and functions thereof

# 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

# The `stan_lm` Function

```
library(rstanarm); options(mc.cores = parallel::detectCores())
post <- stan_lm(log(price) ~ carat * (log(x) + log(y) + log(z)) + cut + color + clarity, data = diamonds,
                prior = R2(location = 0.8, what = "mode"), subset = z > 0, adapt_delta = 0.95)
```

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

```
## 'data.frame':    4000 obs. of  28 variables:
##  $ (Intercept)  : num  0.73 0.73 0.72 0.72 ...
##  $ carat        : num  7.4 7.5 7.3 7.3 ...
##  $ log(x)       : num  4.7 4.7 4.5 4.5 ...
##  $ log(y)       : num  -2.6 -2.6 -2.5 -2.4 ...
##  $ log(z)       : num  0.92 0.9 1.01 0.93 ...
##  $ cutGood      : num  0.083 0.087 0.088 0.079 ...
##  $ cutVery Good : num  0.12 0.12 0.13 0.12 ...
##  $ cutPremium   : num  0.13 0.14 0.14 0.13 ...
##  $ cutIdeal     : num  0.16 0.17 0.17 0.16 ...
##  $ colorE       : num  -0.057 -0.052 -0.053 -0.056 ...
##  $ colorF       : num  -0.096 -0.096 -0.095 -0.096 ...
##  $ colorG       : num  -0.16 -0.16 -0.16 -0.16 ...
##  $ colorH       : num  -0.26 -0.26 -0.26 -0.26 ...
##  $ colorI       : num  -0.37 -0.38 -0.37 -0.37 ...
##  $ colorJ       : num  -0.51 -0.51 -0.51 -0.51 ...
##  $ claritySI2   : num  0.41 0.41 0.42 0.42 ...
##  $ claritySI1   : num  0.58 0.58 0.59 0.58 ...
##  $ clarityVS2   : num  0.73 0.73 0.73 0.73 ...
##  $ clarityVS1   : num  0.8 0.8 0.8 0.8 ...
##  $ clarityVVS2  : num  0.93 0.93 0.94 0.93 ...
##  $ clarityVVS1  : num  1 1 1 1 ...
##  $ clarityIF    : num  1.1 1.1 1.1 1.1 ...
##  $ carat:log(x) : num  -4.1 -4.1 -3.9 -3.9 ...
##  $ carat:log(y) : num  2 1.9 1.9 1.9 ...
##  $ carat:log(z) : num  -1.1 -1 -1.1 -1.1 ...
##  $ sigma        : num  0.13 0.13 0.13 0.13 ...
##  $ log-fit_ratio: num  -6.5e-04 -6.6e-05 2.7e-04 2.0e-04
##  $ R2           : num  0.98 0.98 0.98 0.98 ...
```

# Typical Output

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

```
...
##                  Median  MAD_SD
## (Intercept)       0.7664  0.0355
## carat             7.4186  0.0724
## log(x)            4.5230  0.0804
## log(y)           -2.5166  0.0721
## log(z)            0.9599  0.0424
## cutGood           0.0852  0.0038
## cutVery Good      0.1223  0.0037
## cutPremium        0.1353  0.0036
## cutIdeal          0.1665  0.0036
## colorE           -0.0551  0.0020
## colorF           -0.0961  0.0020
## colorG           -0.1628  0.0019
## colorH           -0.2572  0.0021
## colorI           -0.3750  0.0024
## colorJ           -0.5116  0.0029
## claritySI2        0.4166  0.0050
## claritySI1        0.5821  0.0049
## clarityVS2        0.7290  0.0050
## clarityVS1        0.8001  0.0050
## clarityVVS2       0.9309  0.0051
## clarityVVS1       1.0022  0.0053
## clarityIF         1.0974  0.0059
## carat:log(x)     -3.9631  0.0661
## carat:log(y)      1.9113  0.0568
## carat:log(z)     -1.1213  0.0443
## sigma             0.1257  0.0004
## log-fit_ratio     0.0000  0.0005
## R2                0.9846  0.0001
##
## Sample avg. posterior predictive distribution
##          Median MAD_SD
## mean_PPD 7.7864 0.0007
##
## ------
## For info on the priors used see help('prior_su
...
```

# More Detailed Output

```r
print(summary(post), digits = 3) # shows estimated effective sample sizes at the bottom
```

```
##
## Model Info:
##
##  function:     stan_lm
##  family:       gaussian [identity]
##  formula:      log(price) ~ carat * (log(x) + log(y) + log(z)) + cut + color +
##     clarity
##  algorithm:    sampling
##  priors:       see help('prior_summary')
##  sample:       4000 (posterior sample size)
##  observations: 53920
##  predictors:   25
##
## Estimates:
##                  mean     sd       2.5%     25%      50%      75%      97.5%
## (Intercept)        0.767    0.035    0.699    0.743    0.766    0.791    0.836
## carat              7.419    0.074    7.276    7.370    7.419    7.468    7.562
## log(x)             4.523    0.080    4.368    4.469    4.523    4.578    4.682
## log(y)            -2.517    0.075   -2.660   -2.565   -2.517   -2.468   -2.371
## log(z)             0.960    0.042    0.878    0.931    0.960    0.989    1.043
## cutGood            0.085    0.004    0.078    0.083    0.085    0.088    0.093
## cutVery Good       0.122    0.004    0.115    0.120    0.122    0.125    0.130
## cutPremium         0.135    0.004    0.128    0.133    0.135    0.138    0.142
## cutIdeal           0.167    0.004    0.160    0.164    0.167    0.169    0.173
## colorE            -0.055    0.002   -0.059   -0.056   -0.055   -0.054   -0.051
## colorF            -0.096    0.002   -0.100   -0.098   -0.096   -0.095   -0.092
## colorG            -0.163    0.002   -0.167   -0.164   -0.163   -0.162   -0.159
## colorH            -0.257    0.002   -0.261   -0.259   -0.257   -0.256   -0.253
## colorI            -0.375    0.002   -0.379   -0.377   -0.375   -0.373   -0.370
## colorJ            -0.512    0.003   -0.517   -0.514   -0.512   -0.510   -0.506
## claritySI2         0.417    0.005    0.407    0.413    0.417    0.420    0.426
```

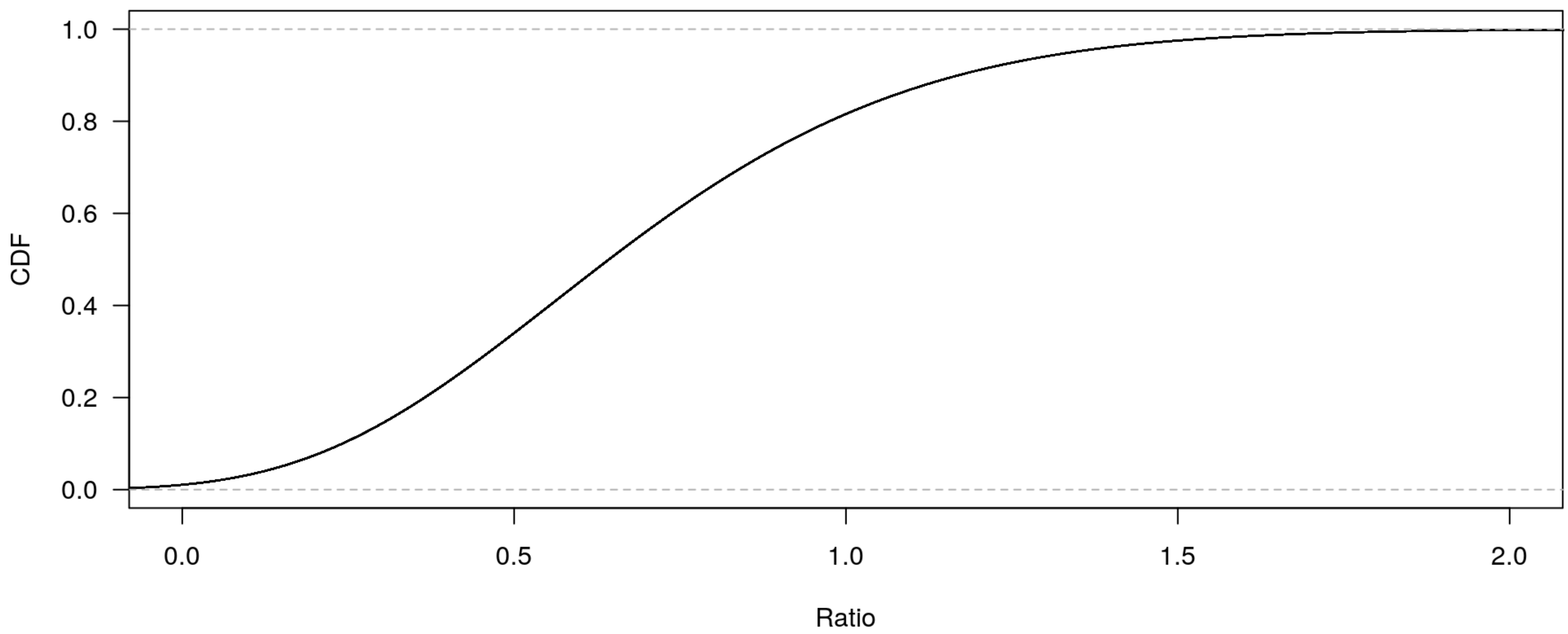
# Credible Intervals

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

```
##                   10%   90%
## (Intercept)      0.72  0.81
## carat            7.32  7.51
## log(x)           4.42  4.62
## log(y)          -2.61 -2.42
## log(z)           0.90  1.01
## cutGood          0.08  0.09
## cutVery Good     0.12  0.13
## cutPremium       0.13  0.14
## cutIdeal         0.16  0.17
## colorE          -0.06 -0.05
## colorF          -0.10 -0.09
## colorG          -0.17 -0.16
## colorH          -0.26 -0.25
## colorI          -0.38 -0.37
## colorJ          -0.52 -0.51
## claritySI2       0.41  0.42
## claritySI1       0.58  0.59
## clarityVS2       0.72  0.74
## clarityVS1       0.79  0.81
## clarityVVS2      0.92  0.94
## clarityVVS1      1.00  1.01
## clarityIF        1.09  1.10
## carat:log(x)    -4.05 -3.88
## carat:log(y)     1.84  1.98
## carat:log(z)    -1.18 -1.06
## sigma            0.13  0.13
## log-fit_ratio    0.00  0.00
## R2               0.98  0.98
```

# What Is the Effect of an Increase in Carat?

```
PPD_0 <- exp(posterior_predict(post, draws = 500))
df <- diamonds[diamonds$z > 0, ]; df$carat <- df$carat + 0.2
PPD_1 <- exp(posterior_predict(post, newdata = df, draws = 500))
plot(ecdf((PPD_1 - PPD_0) / PPD_0), main = "", xlim = c(0, 2), xlab = "Ratio", ylab = "CDF")
```



# 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
##          0.44          0.78          0.91          1.03          0.83          1.08
##  cutVery Good    cutPremium      cutIdeal        colorE        colorF        colorG
##          1.11          1.14          1.11          1.08          1.02          1.07
##        colorH        colorI        colorJ    claritySI2    claritySI1    clarityVS2
##          0.69          1.11          0.89          1.05          1.04          1.03
##    clarityVS1   clarityVVS2   clarityVVS1     clarityIF  carat:log(x)  carat:log(y)
##          1.01          0.96          1.03          1.01          1.04          1.02
##  carat:log(z)         sigma log-fit_ratio            R2
##          0.57          0.43          1.08          0.42
```

# 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

# The `stan_glm` Function in rstanarm

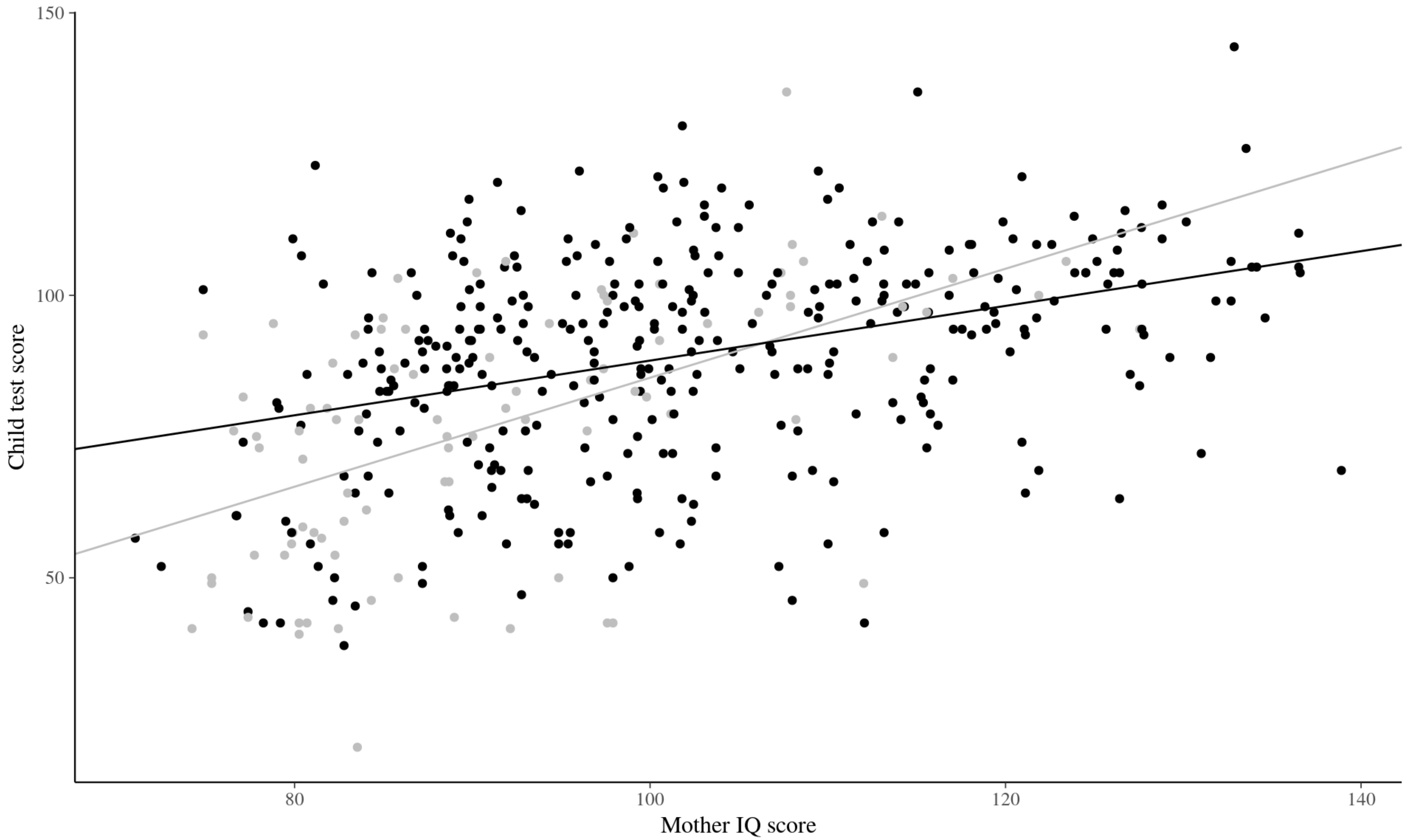
- All examples from the reading (plus more) are available at https://github.com/avehtari/RAOS-Examples

```
library(rstanarm); options(mc.cores = parallel::detectCores());
data(kidiq, package = "rstanarm")
fit_4 <- stan_glm(kid_score ~ mom_hs * mom_iq, data = kidiq, family = gaussian(),
                  prior_intercept = normal(location = 100, scale = 15, autoscale = FALSE),
                  prior = normal(autoscale = FALSE), QR = TRUE)
```

```
fit_4
```

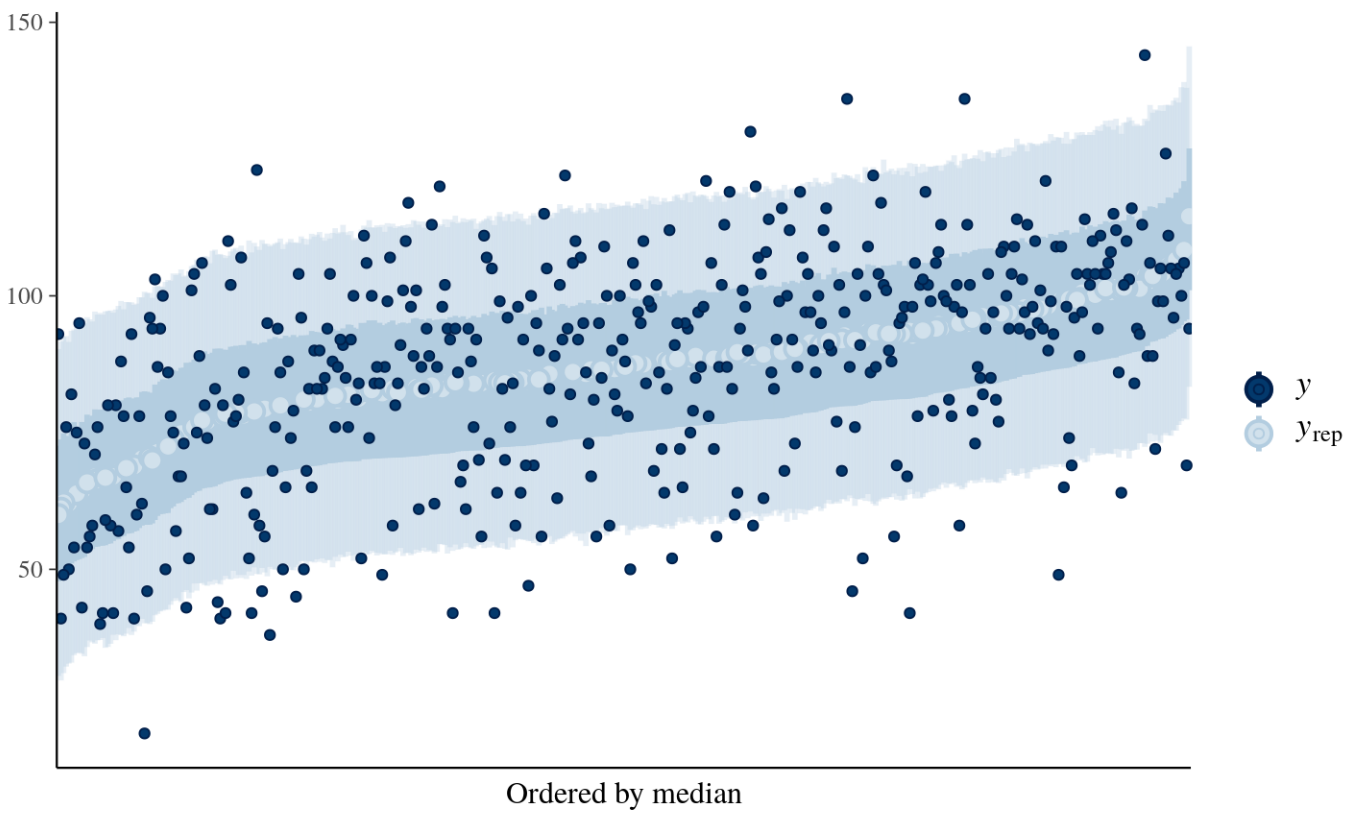
```
...
##               Median MAD_SD
## (Intercept)   -11.1   14.0
## mom_hs         51.1   15.5
## mom_iq          1.0    0.1
## mom_hs:mom_iq  -0.5    0.2
## sigma          18.0    0.6
##
## Sample avg. posterior predictive distribution of y:
##          Median MAD_SD
## mean_PPD 86.8    1.2
##
```

# Plot at the Posterior Median Estimates



# Correct Plot

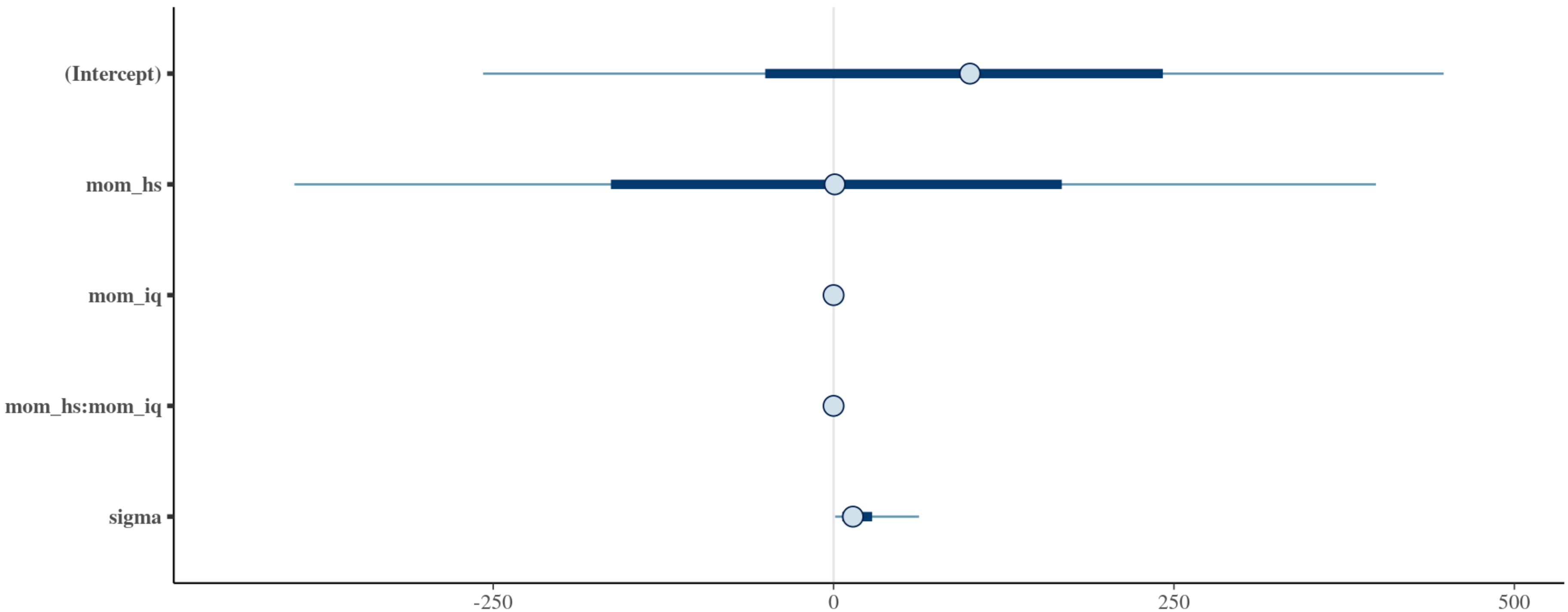
```
pp_check(fit_4, plotfun = "loo_intervals", order = "median")
```



# What Did the Priors Imply?

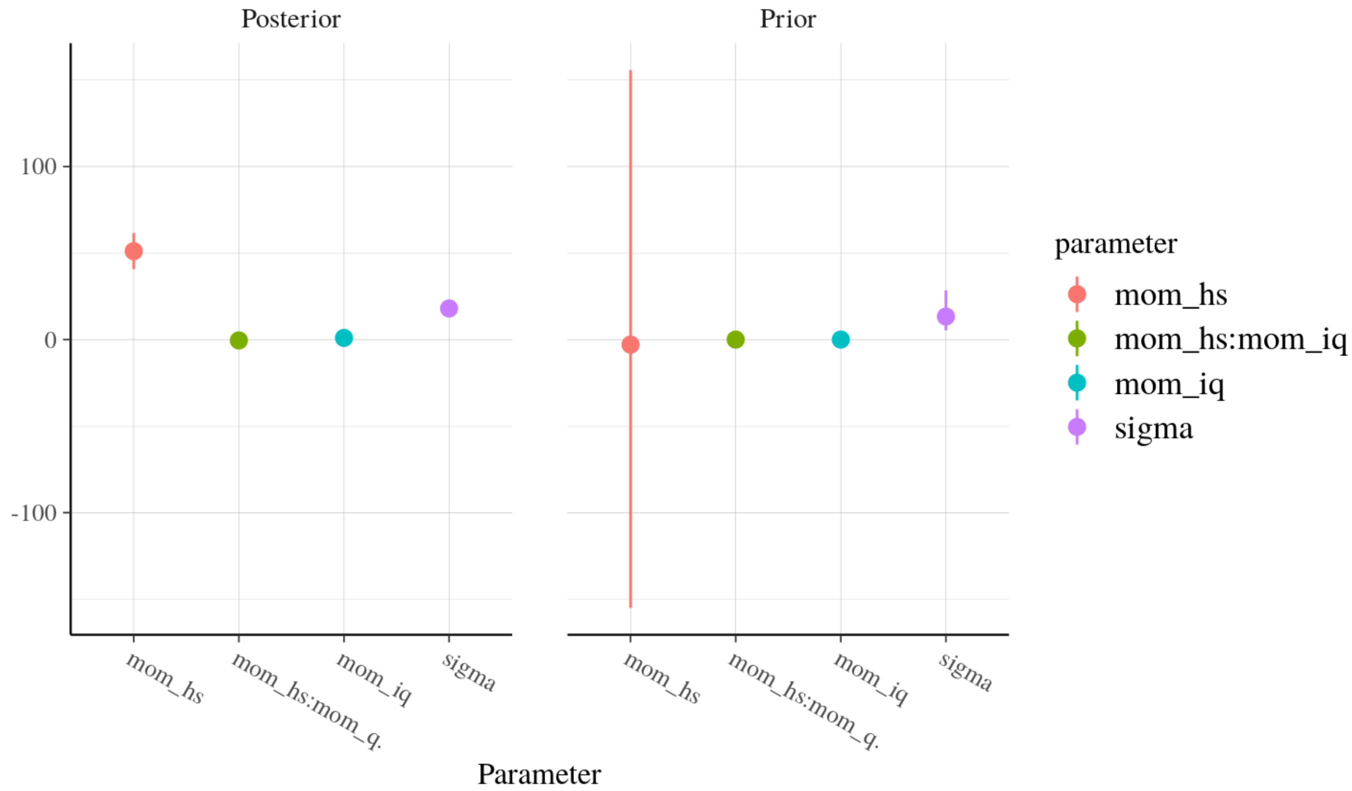
```r
prior_4 <- update(fit_4, prior_PD = TRUE) # calls stan_glm() again with new arguments
```

```r
plot(prior_4)
```



# Posterior vs. Prior

```r
posterior_vs_prior(fit_4, prob = 0.5, regex_pars = "^[^(]") # excludes (Intercept)
```



# Logit Models with Normal Priors

```
functions { /* saved as logit_PPD_rng.stan */
  matrix
    logit_PPD_rng(int S, vector x, real a_loc,
                  real a_scale, real b_loc,
                  real b_scale) {
    int N = rows(x); matrix[S, N] y_tilde;
    vector[N] x_ = x - mean(x);
    for (s in 1:S) {
      real alpha = normal_rng(a_loc, a_scale);
      real beta = normal_rng(b_loc, b_scale);
      vector[N] eta = alpha + beta * x_;
      for (n in 1:N) {
        real utils = eta[n] + logistic_rng(0, 1);
        y_tilde[s, n] = utils > 0;
      }
    }
    return y_tilde;
  }
}
```

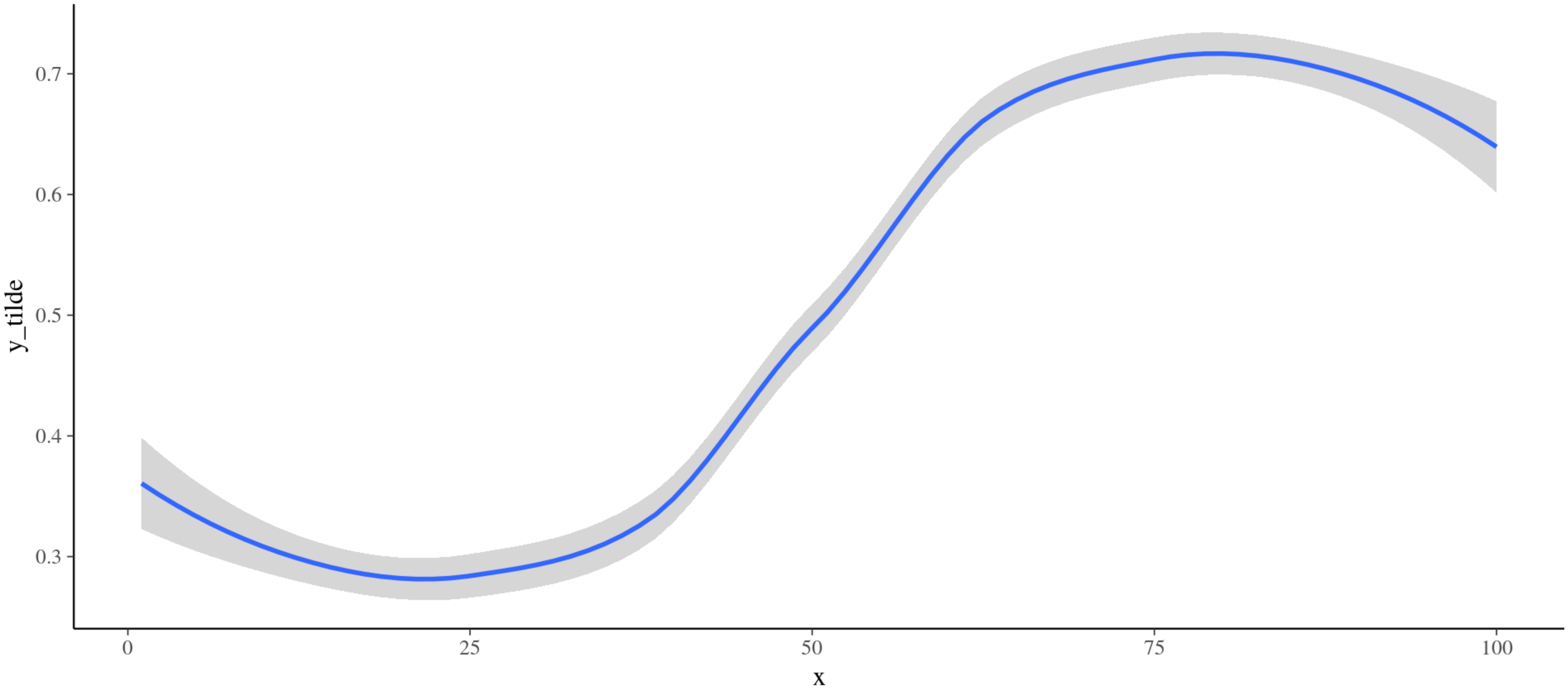
```
functions { /* saved as logit_kernel.stan */
  real
    logit_kernel(real alpha, real beta,
                 real a_loc, real a_scale,
                 real b_loc, real b_scale,
                 int[] y, vector x) {
    int N = rows(x); vector[N] x_ = x - mean(x);

    real p = normal_lpdf(alpha | a_loc, a_scale);
    real q = normal_lpdf(beta | b_loc, b_scale);
    vector[N] eta = alpha + beta * x_;

    return p + q + // priors & log-likelihood
           bernoulli_logit_lpmf(y | eta);
  }
}
```

# What Does `bernoulli_logit_lpmf` Do?

- What is the logarithm of the Bernoulli PMF?
- The Bernoulli PMF is $\Pr\left(y \mid \mu\right) = \mu^y \left(1 - \mu\right)^{1-y}$, so its logarithm is $\ell\left(\mu; y\right) = y \ln \mu + \left(1 - y\right) \ln(1 - \mu)$
- What is the conditional distribution of utility in a logit model?
- What is an expression for probability that $Y = 1$ if utility is logistic with expectation $\eta$ and scale 1?
- $\Pr\left(Y = 1\right) = \Pr\left(\eta + \epsilon > 0\right) = 1 - F\left(0 \mid \eta, 1\right) = F\left(\eta \mid 0, 1\right) = \frac{1}{1+e^{-\eta}}$
- Combining these, we get $\ell\left(\mu; y\right) = -y \ln(1 + e^{-\eta}) + \left(1 - y\right)\left(-\eta - \ln(1 + e^{-\eta})\right)$
- Can save time by only calculating $\ln(1 + e^{-\eta})$ once and can preserve numerical precision by using `log1p_exp(-eta)`

# Checking the Prior Predictive Distribution

```
rstan::expose_stan_functions("logit_PPD_rng.stan"); N <- 100; x <- 1:N
y_tilde <- logit_PPD_rng(S = 10000, x, a_loc = 0, a_scale = 4, b_loc = 2, b_scale = 4)
ggplot(data.frame(x, y_tilde = colMeans(y_tilde))) + geom_smooth(aes(x = x, y = y_tilde))
```



# A Logit Model for Romney vs Obama in 2012

```
poll <- readRDS("GooglePoll.rds") # WantToWin is coded as 1 for Romney and 0 for Obama
library(dplyr)
collapsed <- filter(poll, !is.na(WantToWin)) %>%
             group_by(Region, Gender, Urban_Density, Age, Income) %>%
             summarize(Romney = sum(grepl("Romney", WantToWin)), Obama = n() - Romney) %>%
             na.omit
```

```
post <- stan_glm(cbind(Romney, Obama) ~ ., data = collapsed, family = binomial(link = "logit"), QR = TRUE)
```

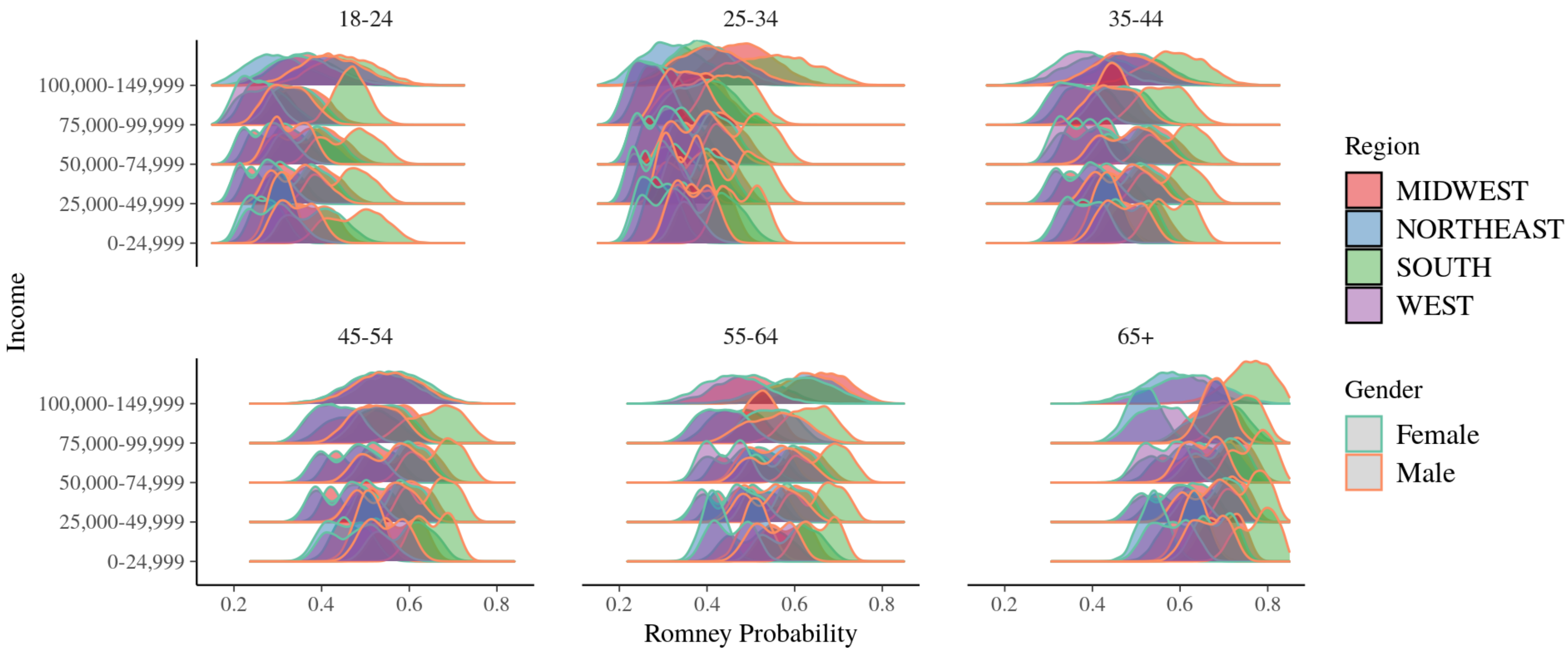
```
print(post, digits = 2)
```

```
...
##                         Median MAD_SD
## (Intercept)             -0.53   0.14
## RegionNORTHEAST         -0.14   0.09
## RegionSOUTH              0.31   0.07
## RegionWEST              -0.14   0.07
## GenderMale               0.38   0.06
## Urban_DensitySuburban   -0.21   0.09
## Urban_DensityUrban      -0.52   0.09
```

```
## Age25-34                 0.11   0.10
## Age35-44                 0.54   0.10
## Age45-54                 0.84   0.09
## Age55-64                 0.85   0.09
## Age65+                   1.35   0.11
## Income25,000-49,999     -0.12   0.08
## Income50,000-74,999     -0.07   0.09
## Income75,000-99,999     -0.09   0.14
## Income100,000-149,999    0.17   0.29
## Income150,000+           0.84   1.01
...
```

# Posterior Distribution of Conditional Expectations

```r
library(tidybayes); filter(collapsed, Income != "150,000+") %>% add_fitted_draws(post) %>%
  ggplot(aes(x = .value, y = Income, fill = Region, color = Gender)) + xlim(0.15, 0.85) +
  scale_fill_brewer(palette = "Set1") + scale_color_brewer(palette = "Set2") +
  ggridges::geom_density_ridges(alpha = .5) + facet_wrap(~ Age) + xlab("Romney Probability")
```



# Generalized Linear Models, in General

- BEFORE you see the outcome data $Y$ has some probability distribution, which for Frequentists must be in the exponential family but that includes a lot of familiar probability distributions
- That probability distribution has a location parameter, which for Frequentists must be the conditional expectation given the predictors
- There is a one-to-one “link” function mapping from this conditional expectation, $\mu(x)$ to the “linear predictor”, $\eta(x)$, although it is often more natural to think about the “inverse link” function that maps from the “linear predictor”, $\eta(x)$ to the conditional expectation, $\mu(x)$
- The “linear predictor”, $\eta(x)$, must be a linear function of the PARAMETERS for Frequentists, although this is not necessary for Bayesians

# Generalized Linear Model Examples This Week

| Model | *Ex ante* Outcome Distribution | Inverse Link |
|---|---|---|
| Linear | `normal_lpdf(y | mu, sigma)` | Identity: $\mu = \eta$ |
| Logit | `bernoulli_lpmf(y | mu)` | Inverse logit: $\mu = \frac{1}{1+e^{-\eta}}$ |
| Probit | `bernoulli_lpmf(y | mu)` | Std. Normal: $\mu = \Phi\left(\eta\right)$ |
| Poisson | `poisson_lpmf(y | mu)` | Antilog: $\mu = e^{\eta}$ |
| Neg Binomial* | `neg_binomial_2_lpmf(y | mu, phi)` | Antilog: $\mu = e^{\eta}$ |

- Not actually a Generalized Linear Model in the Frequentist sense
- In each case, $\eta_i = \alpha + \sum_{k=1}^{K} x_i \beta_k$
- The normal priors on $\alpha$ and each $\beta_k$ could be changed to (an)other family / families

# Poisson Models with Normal Priors

```
functions { /* named poisson_PPD_rng.stan */
  matrix poisson_PPD_rng(int S, vector x,
                         real a_loc,
                         real a_scale,
                         real b_loc,
                         real b_scale) {
    int N = rows(x); matrix[S, N] y_tilde;
    vector[N] x_ = x - mean(x);
    for (s in 1:S) {
      real alpha = normal_rng(a_loc, a_scale);
      real beta = normal_rng(b_loc, b_scale);
      vector[N] eta = alpha + beta * x_;
      vector[N] mu = exp(eta);
      for (n in 1:N)
        y_tilde[s, n] = poisson_rng(mu[n]);
    }
    return y_tilde;
  }
}
```

```
functions { /* named poisson_kernel.stan */
  real poisson_kernel(real alpha, real beta,
                      real a_loc,
                      real a_scale,
                      real b_loc,
                      real b_scale,
                      int[] y, vector x) {
    int N = rows(x); vector[N] x_ = x - mean(x);

    real p = normal_lpdf(alpha | a_loc, a_scale);
    real q = normal_lpdf(beta | b_loc, b_scale);
    vector[N] eta = alpha + beta * x_;


    return p + q + // priors & log-likelihood
           poisson_log_lpmf(y | eta);
  }
}
```

# Negative Binomial Models with Normal Priors

```
functions { /* saved as nb_PPD_rng.stan */
  matrix
    nb_PPD_rng(int S, vector x, real a_loc,
               real a_scale, real b_loc,
               real b_scale, real rate) {
    int N = rows(x); matrix[S, N] y_tilde;
    vector[N] x_ = x - mean(x); for (s in 1:S) {
      real alpha = normal_rng(a_loc, a_scale);
      real beta = normal_rng(b_loc, b_scale);
      real phi = 1 / exponential_rng(rate);
      for (n in 1:N) {
        real z = exp(alpha + beta * x_[n]) / phi;
        real lambda = gamma_rng(z, z);
        y_tilde[s, n] = poisson_rng(lambda);
      }
    }
    return y_tilde;
  }
}
```

```
functions { /* saved as nb_kernel.stan */
  real nb_PPD_rng(real alpha, real beta,
                  real inv_phi, real a_loc,
                  real a_scale, real b_loc,
                  real b_scale, real rate,
                  vector x, int[] y) {
    int N = rows(x); vector[N] x_ = x - mean(x);
    real p = normal_lpdf(alpha | a_loc, a_scale);
    real q = normal_lpdf(beta | b_loc, b_scale);
    real r = exponential_lpdf(inv_phi | rate);

    vector[N] eta = alpha + beta * x_;

    return p + q + r + // priors & loglikelihood
      neg_binomial_2_log_lpmf(y | eta,
                              inv(inv_phi));
  }
}
```

# Count Models with stan_glm{.nb}

```r
post <- stan_glm.nb(y ~ I(roach1 / 100) + treatment + senior, data = roaches,
                    offset = log(exposure2), QR = TRUE)
```

```r
post
```

```
...
##                        Median MAD_SD
## (Intercept)             2.8    0.2  
## I(roach1/100)           1.3    0.2  
## treatment              -0.8    0.3  
## senior                 -0.3    0.3  
## reciprocal_dispersion  0.3    0.0  
##
## Sample avg. posterior predictive distribution of y:
##          Median MAD_SD
## mean_PPD 49.1   28.7  
##
## ------
## For info on the priors used see help('prior_summary.stanreg').
...
```

# Estimating Treatment Effects

```
df <- roaches; df$treatment <- 0
Y_0 <- posterior_linpred(post, newdata = df, offset = log(df$exposure2), transform = TRUE)
df$treatment <- 1
Y_1 <- posterior_linpred(post, newdata = df, offset = log(df$exposure2), transform = TRUE)
plot(ecdf((Y_1 - Y_0) / Y_0), pch = ".", xlab = "Relative treatment effect", main = "")
```