

## Slide 12 - R Output

Elvan Ceyhan

11/14/2024

```
#Set Working Directory to Source File Location  
library("rstudioapi") # Load rstudioapi package  
#setwd(dirname(getActiveDocumentContext()$path)) # Set working directory to source file location  
#getwd()
```

Some required packages:

```
# Load packages  
library(rstan)  
library(mvtnorm) # For multivariate normal sampling  
library(coda) # For MCMC diagnostics  
library(TeachingDemos) # For HPD intervals  
library(ggplot2)
```

## Poisson Regression with Metropolis-Hastings on Simulated High School Awards Data

```
# Load necessary libraries  
library(mvtnorm) # For multivariate normal sampling  
library(coda) # For MCMC diagnostics  
library(TeachingDemos) # For HPD intervals  
  
# Load the dataset  
award_data <- read.csv("https://stats.idre.ucla.edu/stat/data/poisson_sim.csv")  
head(award_data)
```

```
##      id num_awards prog math  
## 1   45          0    3   41  
## 2  108          0    1   41  
## 3   15          0    3   44  
## 4   67          0    3   42  
## 5  153          0    3   40  
## 6   51          0    1   42
```

```
# Define the response variable and predictors  
y <- award_data$num_awards  
x1 <- award_data$math
```

```

x2 <- ifelse(award_data$prog == 2, 1, 0) # Academic track
x3 <- ifelse(award_data$prog == 3, 1, 0) # Vocational track

# Construct design matrix for regression
X <- cbind(1, x1, x2, x3)
p <- ncol(X) # Number of parameters in the model

# Define prior parameters
beta_pri_mean <- c(0, 3, 2, -1) # Prior means for each beta
beta_pri_sd <- rep(10, p) # Standard deviation for prior

# Set up Metropolis-Hastings algorithm parameters
k<-1 # Can adjust this k up or down if the acceptance rate is too high or low...
pro_cov_mat <- k*var(log(y+1/2))*solve(t(X)%*%X) # Proposal covariance matrix
S <- 100000 # Number of MCMC iterations
beta_cur <- beta_pri_mean # Initial values for beta
acs <- 0 # Track acceptance count

# Storage for sampled beta values
beta_vals <- matrix(0, nrow = S, ncol = p)

# Metropolis-Hastings Sampling
for (s in 1:S) {
  # Propose new beta values from multivariate normal distribution
  beta_proposed <- t(rmvnorm(1, beta_cur, pro_cov_mat))

  # Calculate the log-acceptance ratio
  log_acc_rat <- sum(dpois(y, exp(X %*% beta_proposed), log = TRUE)) -
    sum(dpois(y, exp(X %*% beta_cur), log = TRUE)) +
    sum(dnorm(beta_proposed, beta_pri_mean, beta_pri_sd, log = TRUE)) -
    sum(dnorm(beta_cur, beta_pri_mean, beta_pri_sd, log = TRUE))

  # Accept or reject the proposed beta values
  if (log_acc_rat > log(runif(1))) {
    beta_cur <- beta_proposed
    acs <- acs + 1 # Increase acceptance count
  }

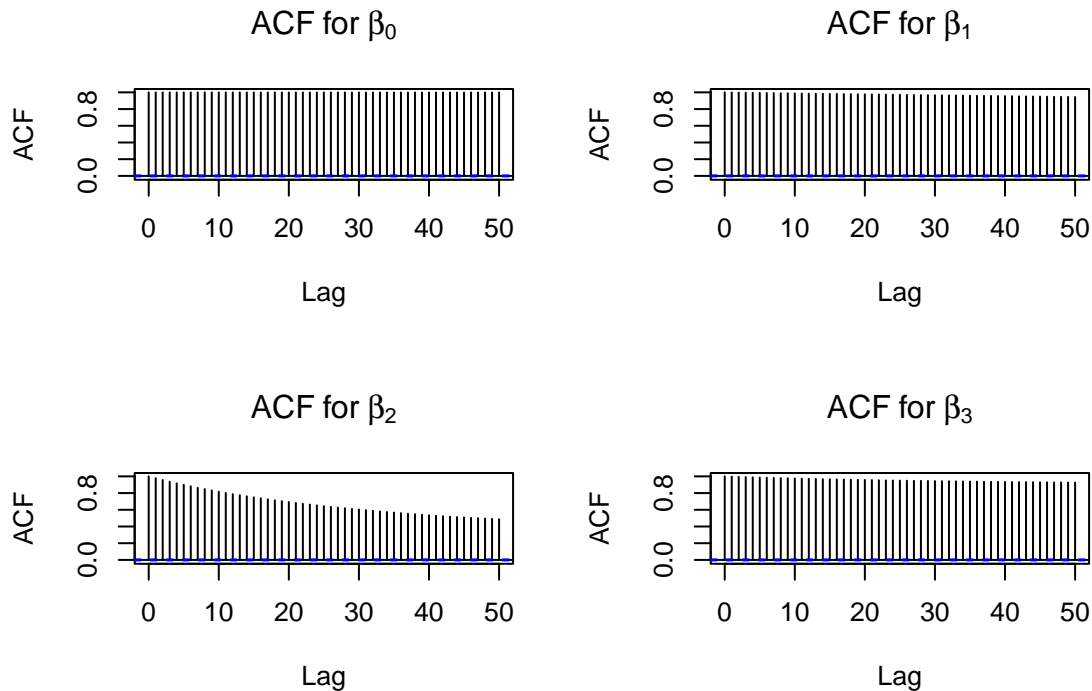
  # Store current beta values
  beta_vals[s, ] <- beta_cur
}

# Calculate acceptance rate
acceptance_rate <- acs / S
print(paste("Acceptance Rate:", round(acceptance_rate, 4)))

## [1] "Acceptance Rate: 0.6131"

# Visualize ACF and trace plots for beta parameters
par(mfrow = c(2, 2))
for (i in 1:p) {
  acf(beta_vals[, i], main = bquote(ACF ~ "for" ~ beta[.(i - 1)]))
}

```



```
par(mfrow = c(1, 1))

#to compute effective sample size (ratios)
# Convert beta_vals to an mcmc object
beta_mcmc <- as.mcmc(beta_vals)
# Calculate the effective sample size for each parameter
ess_values <- effectiveSize(beta_mcmc)
# Calculate the effective sample size ratio for each parameter
ess_ratios <- round(ess_values / nrow(beta_vals), 6)
# Display the results
ess_results <- data.frame(parameter = paste0("beta_", 0:(ncol(beta_vals) - 1)),
                          eff_SS_ratio = ess_ratios)

print("Effective Sample Size Ratios for Parameters:")
```

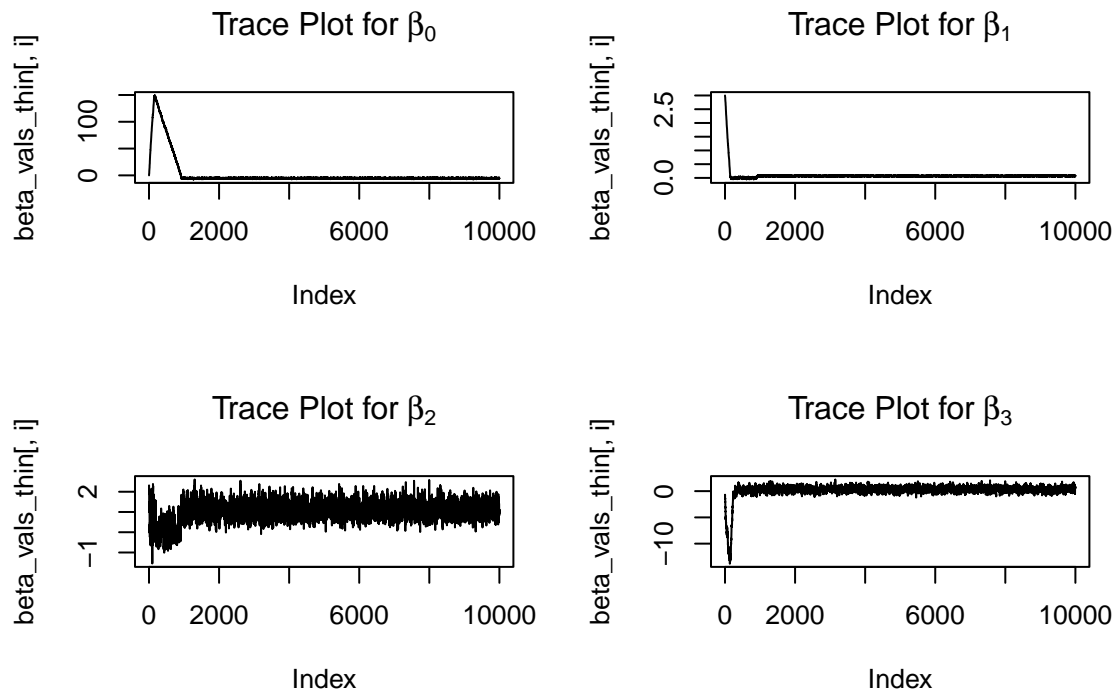
```
## [1] "Effective Sample Size Ratios for Parameters:"
```

```
row.names(ess_results) = c("intercept", "math_score", "academic", "vocational")
print(ess_results)
```

```
##           parameter eff_SS_ratio
## intercept    beta_0      0.000008
## math_score    beta_1      0.000592
## academic      beta_2      0.006157
## vocational    beta_3      0.000530
```

```
# Thinning the chain to reduce autocorrelation
thin <- 10
beta_vals_thin <- beta_vals[seq(1, S, by = thin), ]
```

```
# Trace plots after thinning
par(mfrow = c(2, 2))
for (i in 1:p) {
  plot(beta_vals_thin[, i], type = 'l', main = bquote("Trace Plot for" ~ beta[.(i - 1)]))
}
```



```
par(mfrow = c(1, 1))

# Perform Geweke diagnostic for convergence on the thinned chain
# Geweke diagnostic for checking convergence of an MCMC chain:
# This gives a z-statistic that compares the mean values of
# (by default) the first 10% of the chain vs. the last 50% of the chain:
geweke_diag <- sapply(1:p, function(i) {
  z <- geweke.diag(beta_vals_thin[, i])
  list(statistic = z[[1]], p_value = 2 * pnorm(abs(z[[1]]), lower.tail = FALSE))
})
print("Geweke Diagnostic Results:")
```

```
## [1] "Geweke Diagnostic Results:"
```

```
print(geweke_diag)
```

```
##           [,1]      [,2]      [,3]      [,4]
## statistic 1.728257  0.5683026 -8.903178  -0.6062523
## p_value   0.08394217 0.5698295 5.427004e-19 0.5443473
```

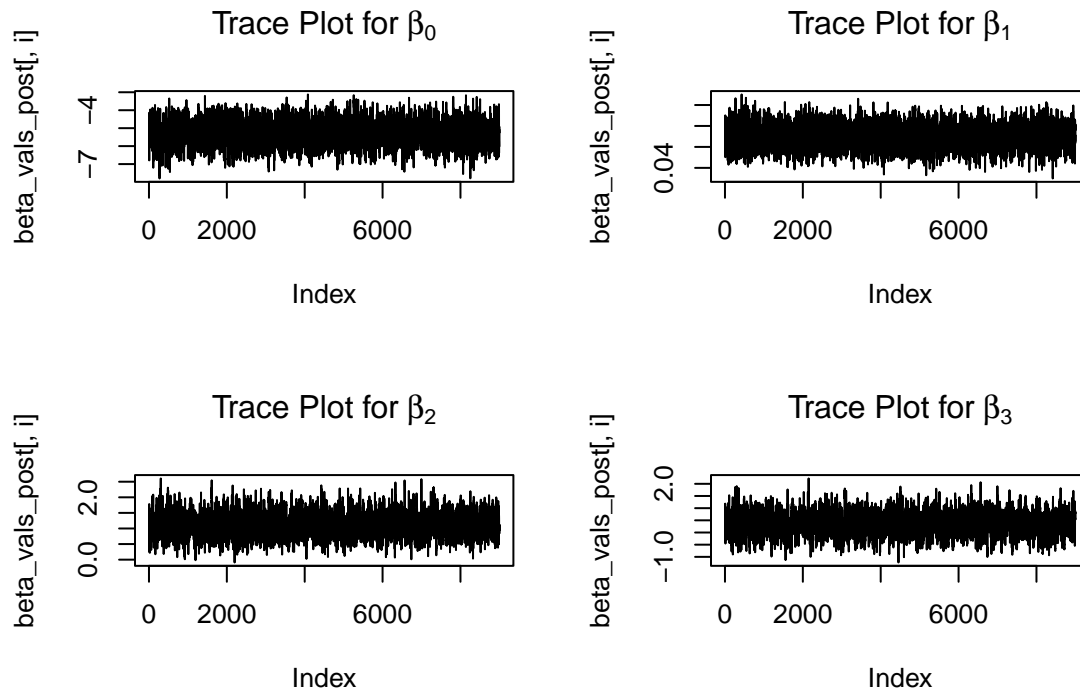
```
# P-value associated with the Geweke diagnostic:
# (a small p-value indicates a LACK of convergence):
```

```

# Discard the first 10% as burn-in for posterior analysis
burn_in <- 1000
beta_vals_post <- beta_vals_thin[-(1:burn_in), ]

# Trace plots after thinning and burn-in
par(mfrow = c(2, 2))
for (i in 1:p) {
  plot(beta_vals_post[, i], type = 'l', main = bquote("Trace Plot for" ~ beta[.(i - 1)]))
}

```



```

par(mfrow = c(1, 1))

# Nice hairy caterpillars now ...

# to compute effective sample size (ratios)
# Convert beta_vals to an mcmc object
beta_mcmc <- as.mcmc(beta_vals_post)
# Calculate the effective sample size for each parameter
ess_values <- effectiveSize(beta_mcmc)
# Calculate the effective sample size ratio for each parameter
ess_ratios <- round(ess_values / nrow(beta_vals), 6)
# Display the results
ess_results <- data.frame(parameter = paste0("beta_", 0:(ncol(beta_vals) - 1)),
                          eff_SS_ratio = ess_ratios)
print("Effective Sample Size Ratios for Parameters:")

```

```
## [1] "Effective Sample Size Ratios for Parameters:"
```

```

row.names(ess_results) = c("intercept", "math_score", "academic", "vocational")
print(ess_results)

```

```
##           parameter eff_SS_ratio
## intercept      beta_0      0.020451
## math_score      beta_1      0.031754
## academic        beta_2      0.014825
## vocational      beta_3      0.014106
```

```
### Posterior summary:
```

```
# Posterior summary: medians and 95% credible intervals
```

```
post_meds <- apply(beta_vals_post, 2, median)
cred_ints <- t(apply(beta_vals_post, 2, quantile, probs = c(0.025, 0.975)))
hpd_ints <- t(apply(beta_vals_post, 2, emp.hpd, conf = 0.95))

print("Posterior Medians:")
```

```
## [1] "Posterior Medians:"
```

```
names(post_meds) = c("intercept", "math_score", "academic", "vocational")
print(post_meds)
```

```
## intercept math_score academic vocational
## -5.27552930 0.06979104 1.10451339 0.35372723
```

```
print("95% Credible Intervals:")
```

```
## [1] "95% Credible Intervals:"
```

```
row.names(cred_ints) = c("intercept", "math_score", "academic", "vocational")
print(cred_ints)
```

```
##           2.5%      97.5%
## intercept -6.5733174 -4.00929243
## math_score 0.0493878 0.09100337
## academic  0.4527466 1.89200544
## vocational -0.5040326 1.26071136
```

```
print("95% HPD Intervals:")
```

```
## [1] "95% HPD Intervals:"
```

```
row.names(hpd_ints) = c("intercept", "math_score", "academic", "vocational")
print(hpd_ints)
```

```
##           [,1]      [,2]
## intercept -7.80265836 -4.21917386
## math_score 0.02985088 0.08750584
## academic  -0.09117116 1.76002060
## vocational -1.23175588 1.12596074
```

```

### Posterior Prediction for Model Fit
Ndraws <- nrow(beta_vals_post)
y_hats <- exp(X %*% t(beta_vals_post)) # Posterior predictions for each sample

# Calculate in-sample Mean Absolute Error (MAE)
mae <- mean(abs(y - rowMeans(y_hats)))
print(paste("In-sample MAE:", round(mae, 4)))

```

```
## [1] "In-sample MAE: 0.5867"
```

```

# Consider fitting different models (e.g., without `x1` or adding interaction terms)
# and comparing MAE across models for a more robust analysis.

```

## Bayesian Poisson Regression using ‘rstanarm’ Package

```

# Load necessary packages
library(bayesrules)
library(rstanarm)
library(bayesplot)
library(tidyverse)
library(tidybayes)
library(broom.mixed)

# Prepare data
prog <- as.factor(award_data$prog)
awards_data <- data.frame(num_awards = y, math_score = x1, academic= x2, vocational = x3, program = prog)
# 'rstanarm' requires data in a data frame format

# Fit Bayesian Poisson regression model with 4 MCMC chains
awards_mod <- stan_glm(
  num_awards ~ math_score + academic + vocational,
  data = awards_data,
  family = poisson,
  prior_intercept = normal(0, 0.5),
  prior = normal(c(3, 2, -1), 2.5, autoscale = TRUE),
  chains = 4, iter = 10000
)

```

```

##
## SAMPLING FOR MODEL 'count' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 6.9e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.69 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 10000 [  0%] (Warmup)
## Chain 1: Iteration: 1000 / 10000 [ 10%] (Warmup)
## Chain 1: Iteration: 2000 / 10000 [ 20%] (Warmup)
## Chain 1: Iteration: 3000 / 10000 [ 30%] (Warmup)

```

```

## Chain 1: Iteration: 4000 / 10000 [ 40%] (Warmup)
## Chain 1: Iteration: 5000 / 10000 [ 50%] (Warmup)
## Chain 1: Iteration: 5001 / 10000 [ 50%] (Sampling)
## Chain 1: Iteration: 6000 / 10000 [ 60%] (Sampling)
## Chain 1: Iteration: 7000 / 10000 [ 70%] (Sampling)
## Chain 1: Iteration: 8000 / 10000 [ 80%] (Sampling)
## Chain 1: Iteration: 9000 / 10000 [ 90%] (Sampling)
## Chain 1: Iteration: 10000 / 10000 [100%] (Sampling)
## Chain 1:
## Chain 1: Elapsed Time: 1.869 seconds (Warm-up)
## Chain 1: 1.697 seconds (Sampling)
## Chain 1: 3.566 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'count' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 2.6e-05 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.26 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration: 1 / 10000 [ 0%] (Warmup)
## Chain 2: Iteration: 1000 / 10000 [ 10%] (Warmup)
## Chain 2: Iteration: 2000 / 10000 [ 20%] (Warmup)
## Chain 2: Iteration: 3000 / 10000 [ 30%] (Warmup)
## Chain 2: Iteration: 4000 / 10000 [ 40%] (Warmup)
## Chain 2: Iteration: 5000 / 10000 [ 50%] (Warmup)
## Chain 2: Iteration: 5001 / 10000 [ 50%] (Sampling)
## Chain 2: Iteration: 6000 / 10000 [ 60%] (Sampling)
## Chain 2: Iteration: 7000 / 10000 [ 70%] (Sampling)
## Chain 2: Iteration: 8000 / 10000 [ 80%] (Sampling)
## Chain 2: Iteration: 9000 / 10000 [ 90%] (Sampling)
## Chain 2: Iteration: 10000 / 10000 [100%] (Sampling)
## Chain 2:
## Chain 2: Elapsed Time: 2.886 seconds (Warm-up)
## Chain 2: 1.39 seconds (Sampling)
## Chain 2: 4.276 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'count' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 2.5e-05 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.25 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration: 1 / 10000 [ 0%] (Warmup)
## Chain 3: Iteration: 1000 / 10000 [ 10%] (Warmup)
## Chain 3: Iteration: 2000 / 10000 [ 20%] (Warmup)
## Chain 3: Iteration: 3000 / 10000 [ 30%] (Warmup)
## Chain 3: Iteration: 4000 / 10000 [ 40%] (Warmup)
## Chain 3: Iteration: 5000 / 10000 [ 50%] (Warmup)
## Chain 3: Iteration: 5001 / 10000 [ 50%] (Sampling)
## Chain 3: Iteration: 6000 / 10000 [ 60%] (Sampling)

```



```

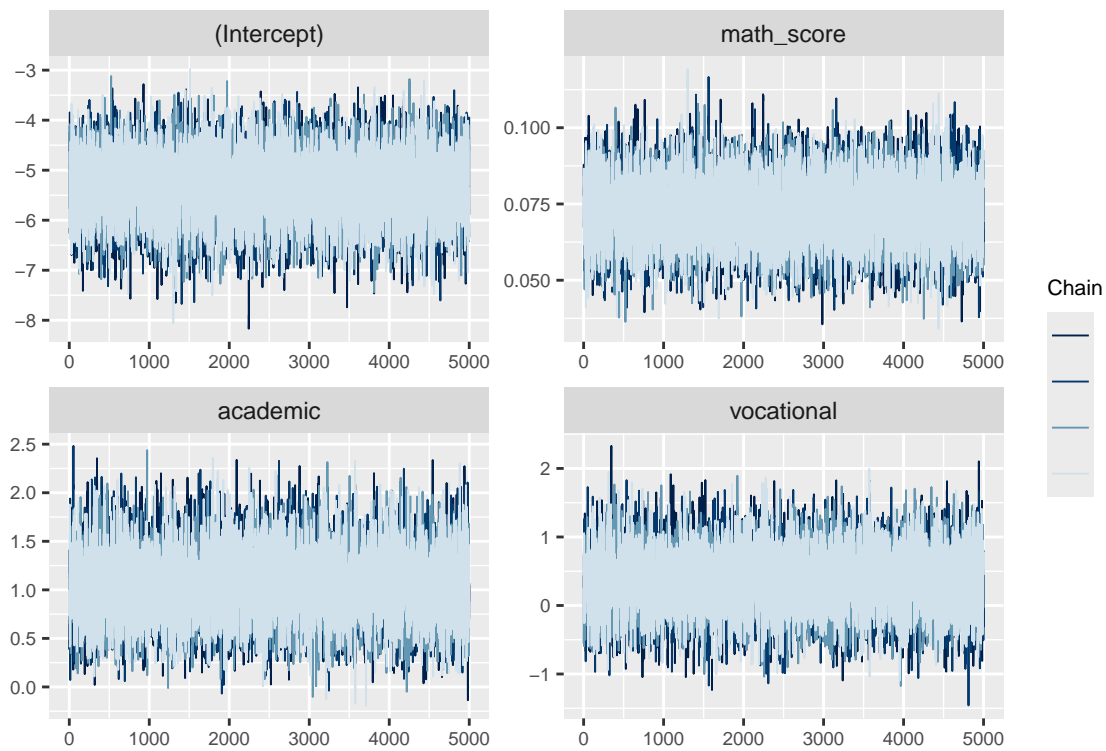
## Chain 3: Iteration: 7000 / 10000 [ 70%] (Sampling)
## Chain 3: Iteration: 8000 / 10000 [ 80%] (Sampling)
## Chain 3: Iteration: 9000 / 10000 [ 90%] (Sampling)
## Chain 3: Iteration: 10000 / 10000 [100%] (Sampling)
## Chain 3:
## Chain 3: Elapsed Time: 2.605 seconds (Warm-up)
## Chain 3: 1.632 seconds (Sampling)
## Chain 3: 4.237 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'count' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 2.6e-05 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.26 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration: 1 / 10000 [ 0%] (Warmup)
## Chain 4: Iteration: 1000 / 10000 [ 10%] (Warmup)
## Chain 4: Iteration: 2000 / 10000 [ 20%] (Warmup)
## Chain 4: Iteration: 3000 / 10000 [ 30%] (Warmup)
## Chain 4: Iteration: 4000 / 10000 [ 40%] (Warmup)
## Chain 4: Iteration: 5000 / 10000 [ 50%] (Warmup)
## Chain 4: Iteration: 5001 / 10000 [ 50%] (Sampling)
## Chain 4: Iteration: 6000 / 10000 [ 60%] (Sampling)
## Chain 4: Iteration: 7000 / 10000 [ 70%] (Sampling)
## Chain 4: Iteration: 8000 / 10000 [ 80%] (Sampling)
## Chain 4: Iteration: 9000 / 10000 [ 90%] (Sampling)
## Chain 4: Iteration: 10000 / 10000 [100%] (Sampling)
## Chain 4:
## Chain 4: Elapsed Time: 3.054 seconds (Warm-up)
## Chain 4: 1.67 seconds (Sampling)
## Chain 4: 4.724 seconds (Total)
## Chain 4:

```

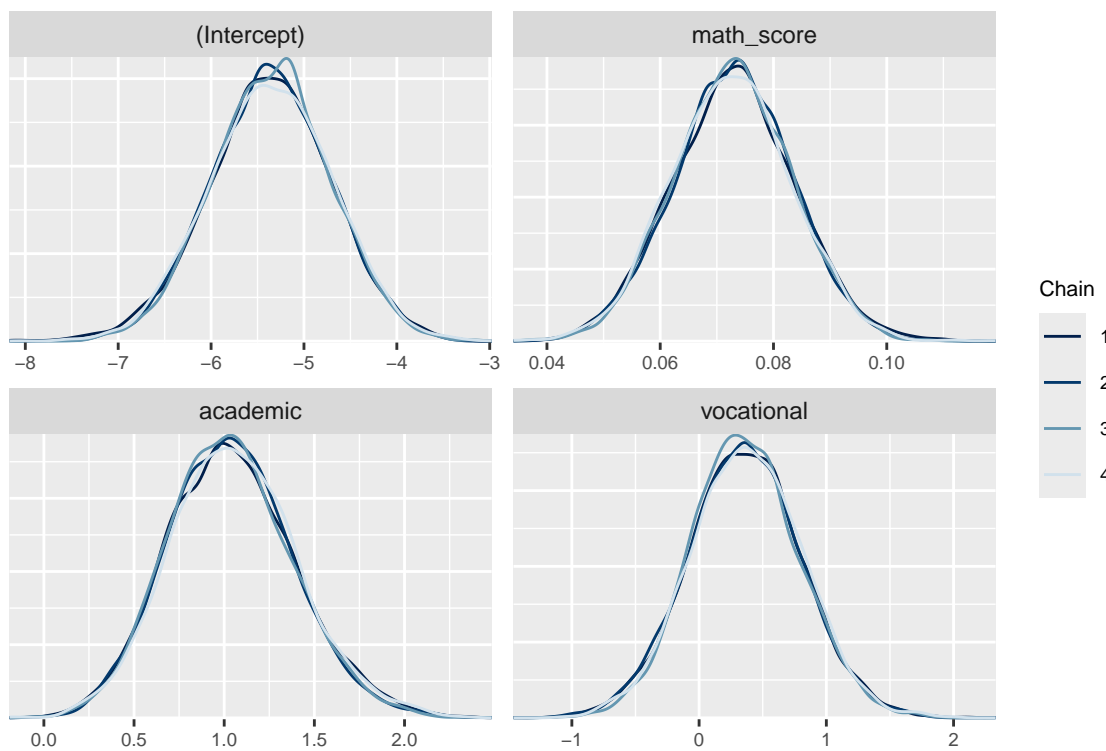
```

# MCMC Diagnostics - trace plots, density overlays, and autocorrelation checks
mcmc_trace(awards_mod)

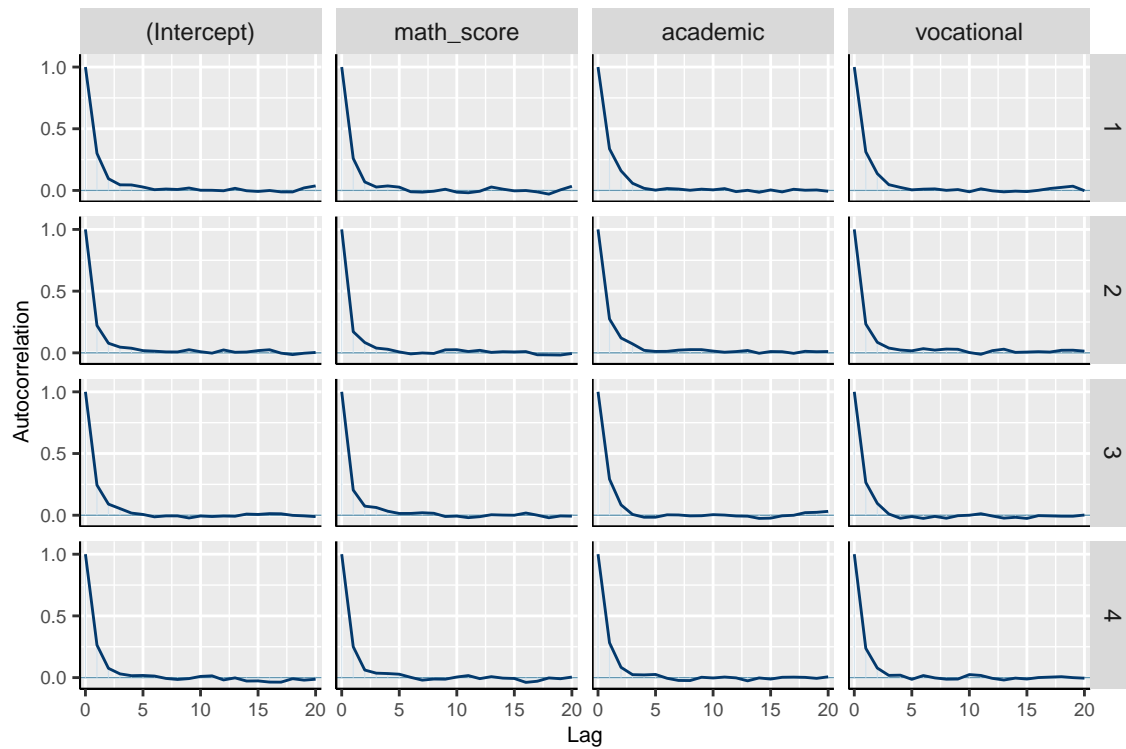
```



```
mcmc_dens_overlay(awards_mod)
```



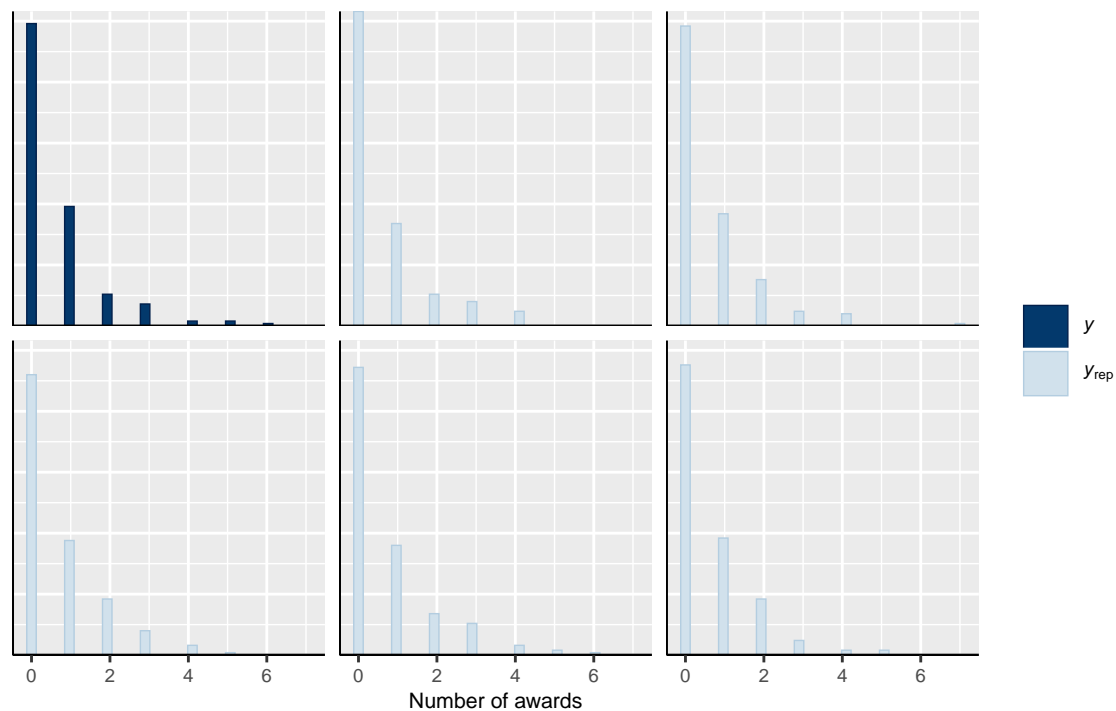
```
mcmc_acf(awards_mod)
```



```
# Posterior Predictive Check: Visualize 5 predictive histograms
pp_check(awards_mod, plotfun = "hist", nreps = 5) +
  xlab("Number of awards") +
  ggtitle("Posterior Predictive Check: Awards Count")
```

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

## Posterior Predictive Check: Awards Count



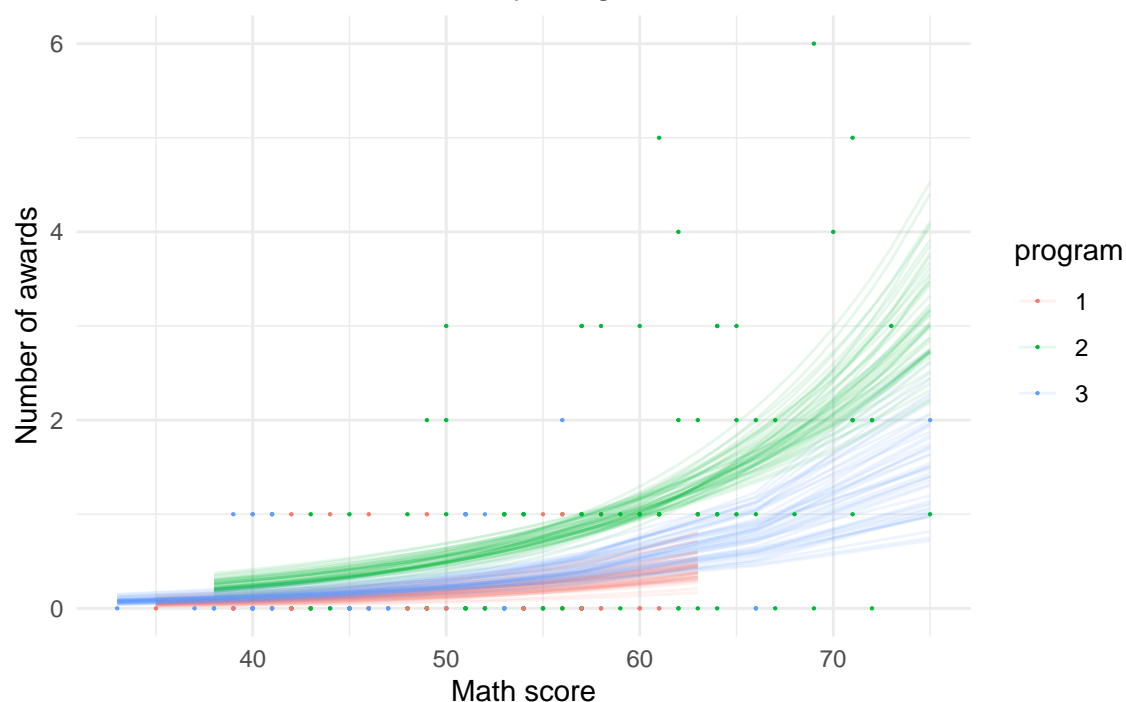
*# The model seems to match the observed awards distribution well.*

*# Visualize 50 posterior plausible models by plotting fitted draws*

```
awards_data %>%
  add_fitted_draws(awards_mod, n = 50) %>%
  ggplot(aes(x = math_score, y = num_awards, color = program)) +
  geom_line(aes(y = .value, group = paste(program, .draw)), alpha = 0.1) +
  geom_point(data = awards_data, size = 0.1) +
  labs(
    x = "Math score",
    y = "Number of awards",
    title = "Posterior Plausible Models by Program Track"
  ) +
  theme_minimal()
```

```
## Warning in fitted_draws.default(model = model, newdata = newdata, ..., n = n): 'fitted_draws' and 'a
## - Use [add_]epred_draws() to get the expectation of the posterior predictive.
## - Use [add_]linpred_draws() to get the distribution of the linear predictor.
## - For example, you used [add_]fitted_draws(..., scale = "response"), which
##   means you most likely want [add_]epred_draws(...).
## NOTE: When updating to the new functions, note that the 'model' parameter is now
##   named 'object' and the 'n' parameter is now named 'ndraws'.
```

## Posterior Plausible Models by Program Track



```
# Point estimates and interval estimates for regression coefficients (betas)
tidy(awards_mod, conf.int = TRUE, conf.level = 0.95)
```

```
## # A tibble: 4 x 5
##   term      estimate std.error conf.low conf.high
##   <chr>      <dbl>    <dbl>   <dbl>   <dbl>
## 1 (Intercept) -5.35      0.649   -6.65   -4.10
## 2 math_score   0.0728    0.0105   0.0520   0.0932
## 3 academic     1.03     0.351    0.388    1.78
## 4 vocational   0.360     0.431   -0.501    1.24
```

```
# Alternative summary of posterior estimates
summary(awards_mod, digits = 4)
```

```
##
## Model Info:
## function:    stan_glm
## family:      poisson [log]
## formula:     num_awards ~ math_score + academic + vocational
## algorithm:    sampling
## sample:      20000 (posterior sample size)
## priors:      see help('prior_summary')
## observations: 200
## predictors:  4
##
## Estimates:
##           mean      sd      10%      50%      90%
## (Intercept) -5.3570  0.6493 -6.1937 -5.3550 -4.5220
```

```
## math_score    0.0728  0.0106  0.0591  0.0728  0.0864
## academic      1.0430  0.3516  0.6070  1.0302  1.4975
## vocational    0.3639  0.4387 -0.1900  0.3599  0.9273
##
## Fit Diagnostics:
##           mean    sd      10%    50%    90%
## mean_PPD 0.6482 0.0807 0.5450 0.6450 0.7550
##
## The mean_ppd is the sample average posterior predictive distribution of the outcome variable (for de
##
## MCMC diagnostics
##           mcse    Rhat    n_eff
## (Intercept) 0.0063 1.0001 10557
## math_score   0.0001 1.0000 11263
## academic     0.0035 1.0007  9984
## vocational   0.0042 1.0004 10816
## mean_PPD     0.0006 0.9998 18749
## log-posterior 0.0168 1.0010  7648
##
## For each parameter, mcse is Monte Carlo standard error, n_eff is a crude measure of effective sample
```

```
# Posterior Prediction for a New Observation:
# Predict awards for a "general" track student with a math score of 40
new_data <- data.frame(math_score = 40, academic = 0, vocational = 0)
ex_pred <- posterior_predict(awards_mod, newdata = new_data)

# Tabulate the posterior predictive distribution
pred_table <- table(ex_pred)
pred_dist <- pred_table / length(ex_pred) # Proportion for each count
print(pred_dist)
```

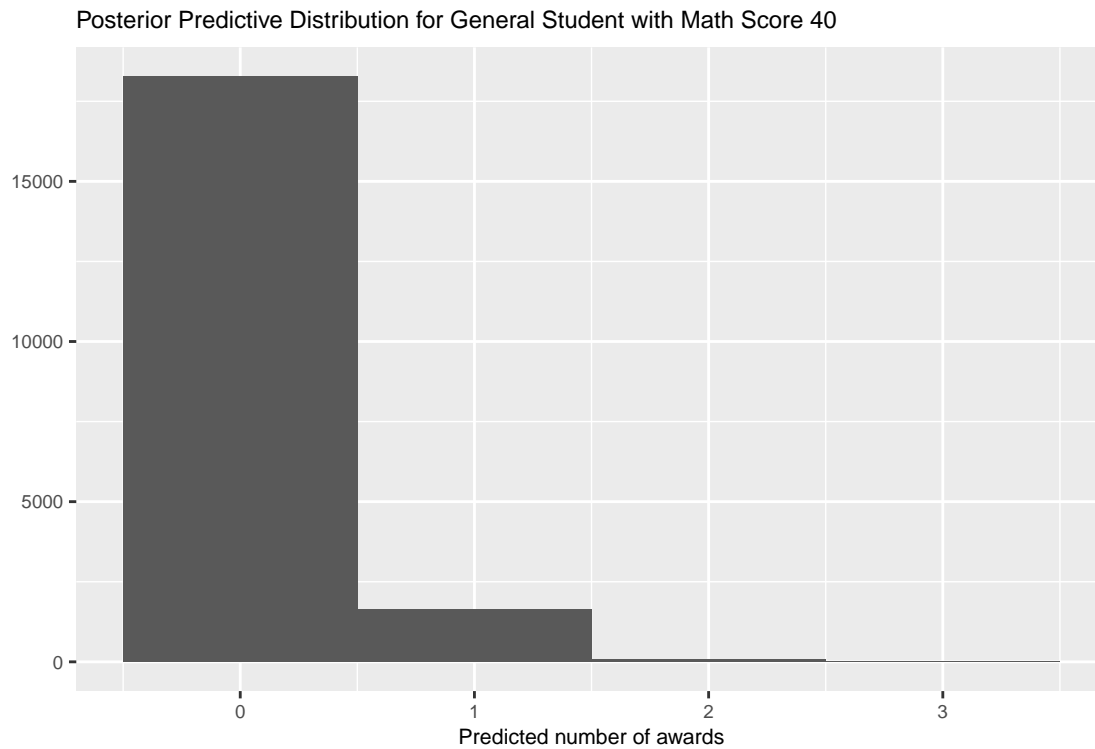
```
## ex_pred
##      0      1      2      3
## 0.91345 0.08240 0.00400 0.00015
```

```
# Calculate and print the posterior predictive mean
post_pred_mean <- sum(as.numeric(names(pred_table)) * pred_dist)
cat("Posterior Predictive Mean:", post_pred_mean, "\n")
```

```
## Posterior Predictive Mean: 0.09085
```

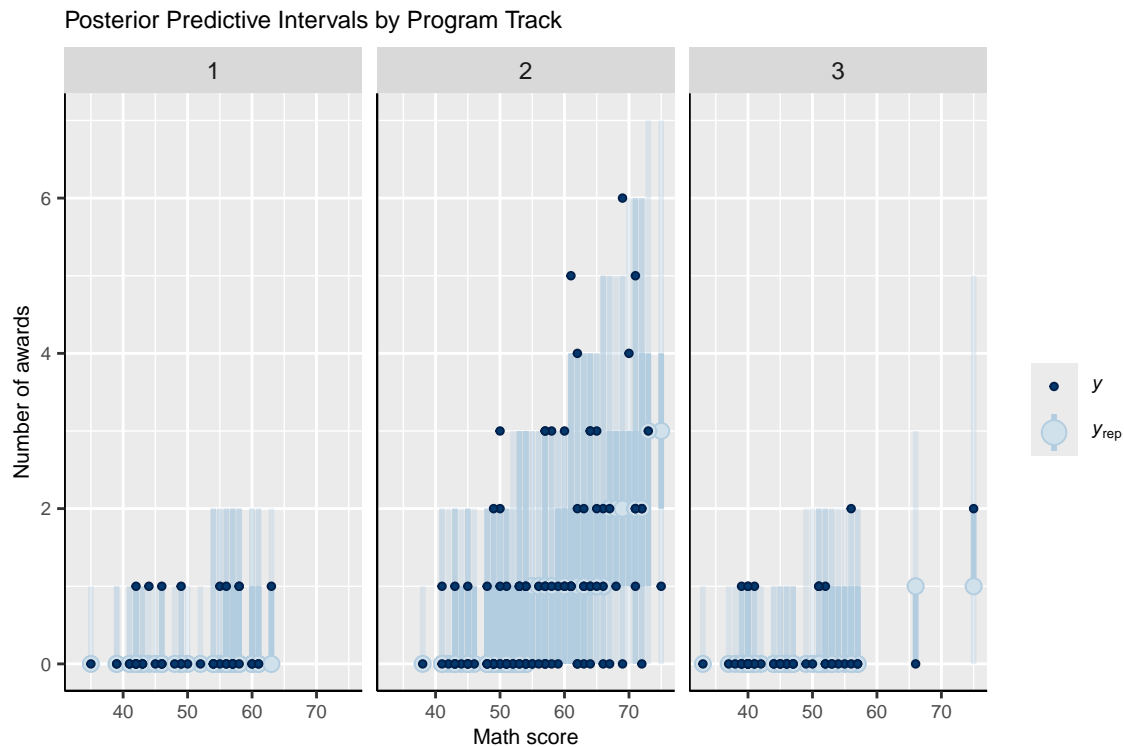
```
# Plot the posterior predictive distribution for this individual
qplot(ex_pred, binwidth = 1, geom = "histogram") +
  xlab("Predicted number of awards") +
  ggtitle("Posterior Predictive Distribution for General Student with Math Score 40")
```

```
## Warning: 'qplot()' was deprecated in ggplot2 3.4.0.
## This warning is displayed once every 8 hours.
## Call 'lifecycle::last_lifecycle_warnings()' to see where this warning was
## generated.
```



```
# Model Fit Evaluation
# Generate posterior predictive values for the observed data
poi_preds <- posterior_predict(awards_mod, newdata = awards_data)

# Plot posterior predictive intervals for each student by program track
ppc_intervals_grouped(
  awards_data$num_awards, yrep = poi_preds,
  x = awards_data$math_score, group = awards_data$prog,
  prob = 0.5, prob_outer = 0.95,
  facet_args = list(scales = "fixed")
) +
  labs(
    x = "Math score",
    y = "Number of awards",
    title = "Posterior Predictive Intervals by Program Track"
  )
```



```
# Numerical Summary of In-Sample Fit
in_sample_fit <- prediction_summary(model = awards_mod, data = awards_data)
print(in_sample_fit)
```

```
##           mae mae_scaled within_50 within_95
## 1 0.367525  0.5594232      0.83      0.985
```

```
# Out-of-Sample Predictive Accuracy using 10-fold Cross-Validation
set.seed(84735) # For reproducibility
poi_cv <- prediction_summary_cv(model = awards_mod, data = awards_data, k = 10)
print(poi_cv$cv) # Cross-validated metrics
```

```
##           mae mae_scaled within_50 within_95
## 1 0.3978025  0.5831736      0.825      0.985
```

```
# Could fit a different model (like without x1? or with an interaction term?)
# and compare MAE's across models ... smaller would be better.
```

## Negative Binomial Regression with Bayesian Model Checking

```
# Load necessary libraries
library(bayesrules)
library(rstanarm)
library(bayesplot)
library(tidyverse)
```

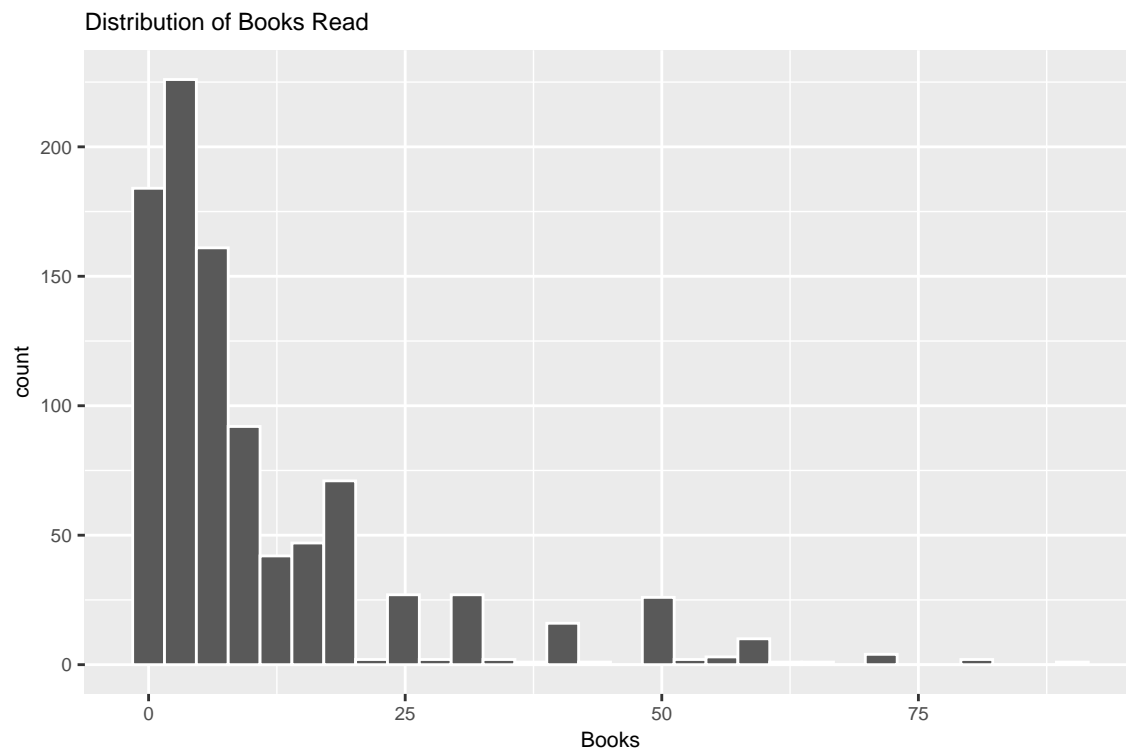


```
library(tidybayes)
library(broom.mixed)

# Load and clean data
data(pulse_of_the_nation)
pulse <- pulse_of_the_nation %>%
  filter(books < 100) # Remove outliers with 100+ books

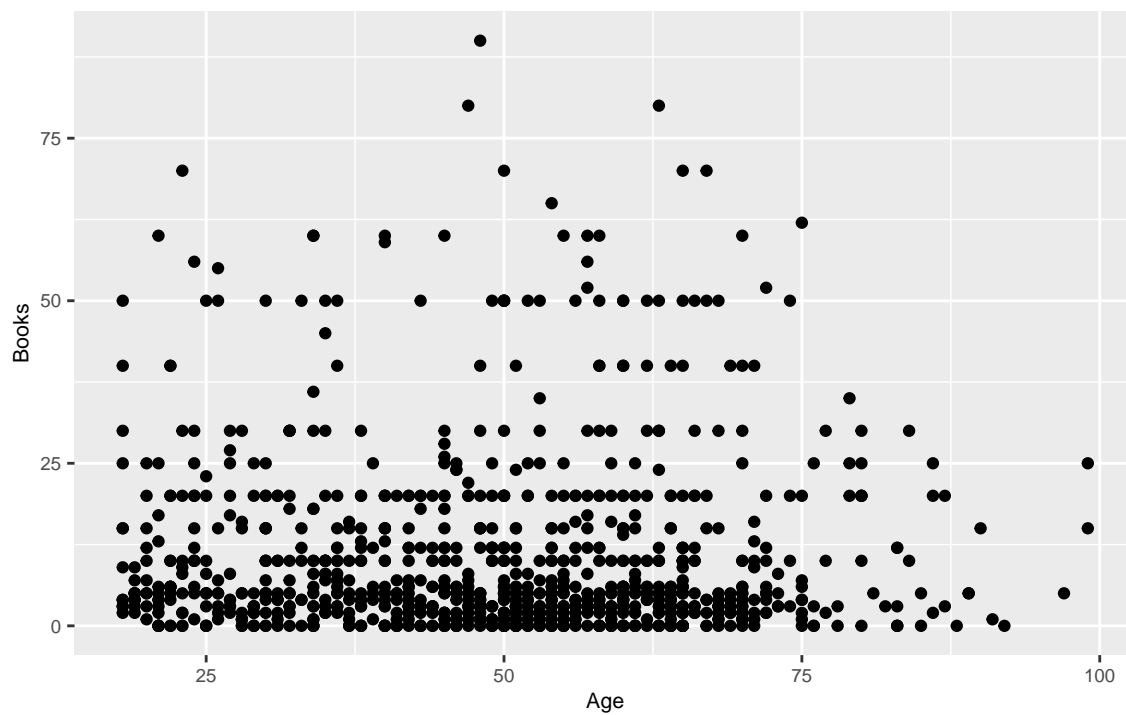
# Summary plots to examine data distribution and relationships
ggplot(pulse, aes(x = books)) +
  geom_histogram(color = "white") +
  labs(x = "Books", title = "Distribution of Books Read")
```

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



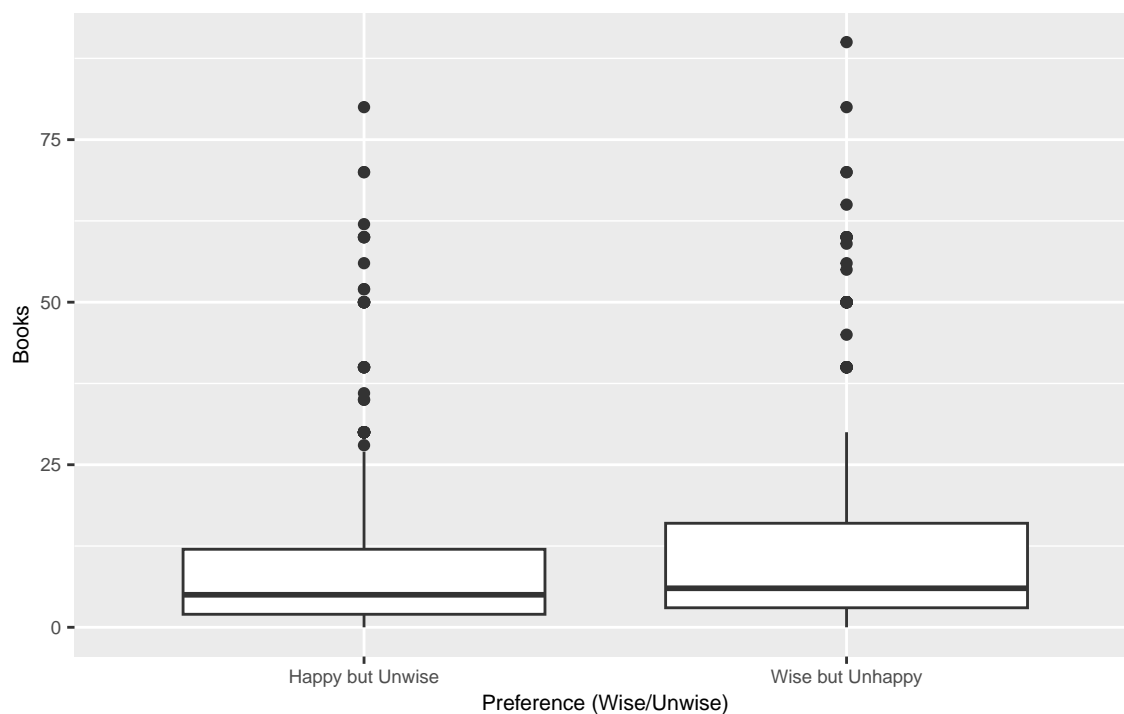
```
ggplot(pulse, aes(y = books, x = age)) +
  geom_point() +
  labs(x = "Age", y = "Books", title = "Books Read vs. Age")
```

Books Read vs. Age



```
ggplot(pulse, aes(y = books, x = wise_unwise)) +  
  geom_boxplot() +  
  labs(x = "Preference (Wise/Unwise)", y = "Books", title = "Books Read by Preference")
```

Books Read by Preference



```
# Attempt Poisson model first to assess fit
```

```
books_poi_sim <- stan_glm(  
  books ~ age + wise_unwise,  
  data = pulse, family = poisson,  
  prior_intercept = normal(0, 2.5, autoscale = TRUE),  
  prior = normal(0, 2.5, autoscale = TRUE),  
  prior_aux = exponential(1, autoscale = TRUE),  
  chains = 4, iter = 10000, seed = 84735  
)
```

```
##
```

```
## SAMPLING FOR MODEL 'count' NOW (CHAIN 1).
```

```
## Chain 1:
```

```
## Chain 1: Gradient evaluation took 0.000225 seconds
```

```
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 2.25 seconds.
```

```
## Chain 1: Adjust your expectations accordingly!
```

```
## Chain 1:
```

```
## Chain 1:
```

```
## Chain 1: Iteration:    1 / 10000 [ 0%] (Warmup)
```

```
## Chain 1: Iteration: 1000 / 10000 [ 10%] (Warmup)
```

```
## Chain 1: Iteration: 2000 / 10000 [ 20%] (Warmup)
```

```
## Chain 1: Iteration: 3000 / 10000 [ 30%] (Warmup)
```

```
## Chain 1: Iteration: 4000 / 10000 [ 40%] (Warmup)
```

```
## Chain 1: Iteration: 5000 / 10000 [ 50%] (Warmup)
```

```
## Chain 1: Iteration: 5001 / 10000 [ 50%] (Sampling)
```

```
## Chain 1: Iteration: 6000 / 10000 [ 60%] (Sampling)
```

```
## Chain 1: Iteration: 7000 / 10000 [ 70%] (Sampling)
```

```
## Chain 1: Iteration: 8000 / 10000 [ 80%] (Sampling)
```

```
## Chain 1: Iteration: 9000 / 10000 [ 90%] (Sampling)
```

```
## Chain 1: Iteration: 10000 / 10000 [100%] (Sampling)
```

```
## Chain 1:
```

```
## Chain 1: Elapsed Time: 4.961 seconds (Warm-up)
```

```
## Chain 1:           5.54 seconds (Sampling)
```

```
## Chain 1:           10.501 seconds (Total)
```

```
## Chain 1:
```

```
##
```

```
## SAMPLING FOR MODEL 'count' NOW (CHAIN 2).
```

```
## Chain 2:
```

```
## Chain 2: Gradient evaluation took 0.000144 seconds
```

```
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 1.44 seconds.
```

```
## Chain 2: Adjust your expectations accordingly!
```

```
## Chain 2:
```

```
## Chain 2:
```

```
## Chain 2: Iteration:    1 / 10000 [ 0%] (Warmup)
```

```
## Chain 2: Iteration: 1000 / 10000 [ 10%] (Warmup)
```

```
## Chain 2: Iteration: 2000 / 10000 [ 20%] (Warmup)
```

```
## Chain 2: Iteration: 3000 / 10000 [ 30%] (Warmup)
```

```
## Chain 2: Iteration: 4000 / 10000 [ 40%] (Warmup)
```

```
## Chain 2: Iteration: 5000 / 10000 [ 50%] (Warmup)
```

```
## Chain 2: Iteration: 5001 / 10000 [ 50%] (Sampling)
```

```
## Chain 2: Iteration: 6000 / 10000 [ 60%] (Sampling)
```

```
## Chain 2: Iteration: 7000 / 10000 [ 70%] (Sampling)
```

```
## Chain 2: Iteration: 8000 / 10000 [ 80%] (Sampling)
```

```

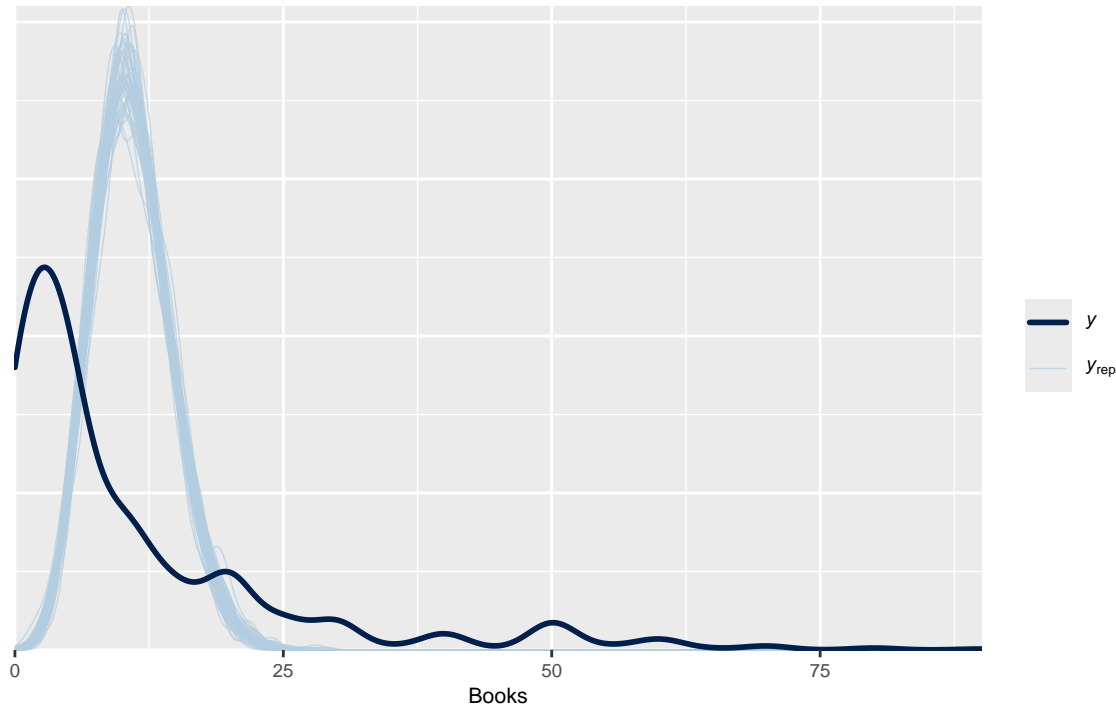
## Chain 2: Iteration: 9000 / 10000 [ 90%] (Sampling)
## Chain 2: Iteration: 10000 / 10000 [100%] (Sampling)
## Chain 2:
## Chain 2: Elapsed Time: 4.823 seconds (Warm-up)
## Chain 2: 5.135 seconds (Sampling)
## Chain 2: 9.958 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'count' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 0.000147 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 1.47 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration: 1 / 10000 [ 0%] (Warmup)
## Chain 3: Iteration: 1000 / 10000 [ 10%] (Warmup)
## Chain 3: Iteration: 2000 / 10000 [ 20%] (Warmup)
## Chain 3: Iteration: 3000 / 10000 [ 30%] (Warmup)
## Chain 3: Iteration: 4000 / 10000 [ 40%] (Warmup)
## Chain 3: Iteration: 5000 / 10000 [ 50%] (Warmup)
## Chain 3: Iteration: 5001 / 10000 [ 50%] (Sampling)
## Chain 3: Iteration: 6000 / 10000 [ 60%] (Sampling)
## Chain 3: Iteration: 7000 / 10000 [ 70%] (Sampling)
## Chain 3: Iteration: 8000 / 10000 [ 80%] (Sampling)
## Chain 3: Iteration: 9000 / 10000 [ 90%] (Sampling)
## Chain 3: Iteration: 10000 / 10000 [100%] (Sampling)
## Chain 3:
## Chain 3: Elapsed Time: 4.656 seconds (Warm-up)
## Chain 3: 5.029 seconds (Sampling)
## Chain 3: 9.685 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'count' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 0.000139 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 1.39 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration: 1 / 10000 [ 0%] (Warmup)
## Chain 4: Iteration: 1000 / 10000 [ 10%] (Warmup)
## Chain 4: Iteration: 2000 / 10000 [ 20%] (Warmup)
## Chain 4: Iteration: 3000 / 10000 [ 30%] (Warmup)
## Chain 4: Iteration: 4000 / 10000 [ 40%] (Warmup)
## Chain 4: Iteration: 5000 / 10000 [ 50%] (Warmup)
## Chain 4: Iteration: 5001 / 10000 [ 50%] (Sampling)
## Chain 4: Iteration: 6000 / 10000 [ 60%] (Sampling)
## Chain 4: Iteration: 7000 / 10000 [ 70%] (Sampling)
## Chain 4: Iteration: 8000 / 10000 [ 80%] (Sampling)
## Chain 4: Iteration: 9000 / 10000 [ 90%] (Sampling)
## Chain 4: Iteration: 10000 / 10000 [100%] (Sampling)
## Chain 4:
## Chain 4: Elapsed Time: 4.771 seconds (Warm-up)

```

```
## Chain 4:          5.78 seconds (Sampling)
## Chain 4:          10.551 seconds (Total)
## Chain 4:
```

```
# Posterior predictive check to evaluate fit
pp_check(books_poi_sim) +
  xlab("Books") +
  ggtitle("Posterior Predictive Check (Poisson Model)")
```

Posterior Predictive Check (Poisson Model)



```
# Check distribution of data's mean and variance
pulse %>%
  summarize(mean = mean(books), var = var(books))
```

```
## # A tibble: 1 x 2
##   mean  var
##   <dbl> <dbl>
## 1  10.9  198.
```

*# Variance is much larger than the mean, indicating overdispersion; Negative Binomial may be better suited*

## Negative Binomial regression to handle overdispersion

```
books_negbin_sim <- stan_glm(
  books ~ age + wise_unwise,
  data = pulse, family = neg_binomial_2,
  prior_intercept = normal(0, 2.5, autoscale = TRUE),
```

```
prior = normal(0, 2.5, autoscale = TRUE),
prior_aux = exponential(1, autoscale = TRUE),
chains = 4, iter = 10000
)
```

```
##
## SAMPLING FOR MODEL 'count' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0.000324 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 3.24 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 10000 [ 0%] (Warmup)
## Chain 1: Iteration: 1000 / 10000 [ 10%] (Warmup)
## Chain 1: Iteration: 2000 / 10000 [ 20%] (Warmup)
## Chain 1: Iteration: 3000 / 10000 [ 30%] (Warmup)
## Chain 1: Iteration: 4000 / 10000 [ 40%] (Warmup)
## Chain 1: Iteration: 5000 / 10000 [ 50%] (Warmup)
## Chain 1: Iteration: 5001 / 10000 [ 50%] (Sampling)
## Chain 1: Iteration: 6000 / 10000 [ 60%] (Sampling)
## Chain 1: Iteration: 7000 / 10000 [ 70%] (Sampling)
## Chain 1: Iteration: 8000 / 10000 [ 80%] (Sampling)
## Chain 1: Iteration: 9000 / 10000 [ 90%] (Sampling)
## Chain 1: Iteration: 10000 / 10000 [100%] (Sampling)
## Chain 1:
## Chain 1: Elapsed Time: 9.403 seconds (Warm-up)
## Chain 1:           10.773 seconds (Sampling)
## Chain 1:           20.176 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'count' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 0.000276 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 2.76 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:    1 / 10000 [ 0%] (Warmup)
## Chain 2: Iteration: 1000 / 10000 [ 10%] (Warmup)
## Chain 2: Iteration: 2000 / 10000 [ 20%] (Warmup)
## Chain 2: Iteration: 3000 / 10000 [ 30%] (Warmup)
## Chain 2: Iteration: 4000 / 10000 [ 40%] (Warmup)
## Chain 2: Iteration: 5000 / 10000 [ 50%] (Warmup)
## Chain 2: Iteration: 5001 / 10000 [ 50%] (Sampling)
## Chain 2: Iteration: 6000 / 10000 [ 60%] (Sampling)
## Chain 2: Iteration: 7000 / 10000 [ 70%] (Sampling)
## Chain 2: Iteration: 8000 / 10000 [ 80%] (Sampling)
## Chain 2: Iteration: 9000 / 10000 [ 90%] (Sampling)
## Chain 2: Iteration: 10000 / 10000 [100%] (Sampling)
## Chain 2:
## Chain 2: Elapsed Time: 9.456 seconds (Warm-up)
## Chain 2:           10.768 seconds (Sampling)
```

```

## Chain 2:                20.224 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'count' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 0.000282 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 2.82 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:      1 / 10000 [ 0%] (Warmup)
## Chain 3: Iteration: 1000 / 10000 [ 10%] (Warmup)
## Chain 3: Iteration: 2000 / 10000 [ 20%] (Warmup)
## Chain 3: Iteration: 3000 / 10000 [ 30%] (Warmup)
## Chain 3: Iteration: 4000 / 10000 [ 40%] (Warmup)
## Chain 3: Iteration: 5000 / 10000 [ 50%] (Warmup)
## Chain 3: Iteration: 5001 / 10000 [ 50%] (Sampling)
## Chain 3: Iteration: 6000 / 10000 [ 60%] (Sampling)
## Chain 3: Iteration: 7000 / 10000 [ 70%] (Sampling)
## Chain 3: Iteration: 8000 / 10000 [ 80%] (Sampling)
## Chain 3: Iteration: 9000 / 10000 [ 90%] (Sampling)
## Chain 3: Iteration: 10000 / 10000 [100%] (Sampling)
## Chain 3:
## Chain 3: Elapsed Time: 9.577 seconds (Warm-up)
## Chain 3:                11.623 seconds (Sampling)
## Chain 3:                21.2 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'count' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 0.000302 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 3.02 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:      1 / 10000 [ 0%] (Warmup)
## Chain 4: Iteration: 1000 / 10000 [ 10%] (Warmup)
## Chain 4: Iteration: 2000 / 10000 [ 20%] (Warmup)
## Chain 4: Iteration: 3000 / 10000 [ 30%] (Warmup)
## Chain 4: Iteration: 4000 / 10000 [ 40%] (Warmup)
## Chain 4: Iteration: 5000 / 10000 [ 50%] (Warmup)
## Chain 4: Iteration: 5001 / 10000 [ 50%] (Sampling)
## Chain 4: Iteration: 6000 / 10000 [ 60%] (Sampling)
## Chain 4: Iteration: 7000 / 10000 [ 70%] (Sampling)
## Chain 4: Iteration: 8000 / 10000 [ 80%] (Sampling)
## Chain 4: Iteration: 9000 / 10000 [ 90%] (Sampling)
## Chain 4: Iteration: 10000 / 10000 [100%] (Sampling)
## Chain 4:
## Chain 4: Elapsed Time: 10.119 seconds (Warm-up)
## Chain 4:                11.654 seconds (Sampling)
## Chain 4:                21.773 seconds (Total)
## Chain 4:

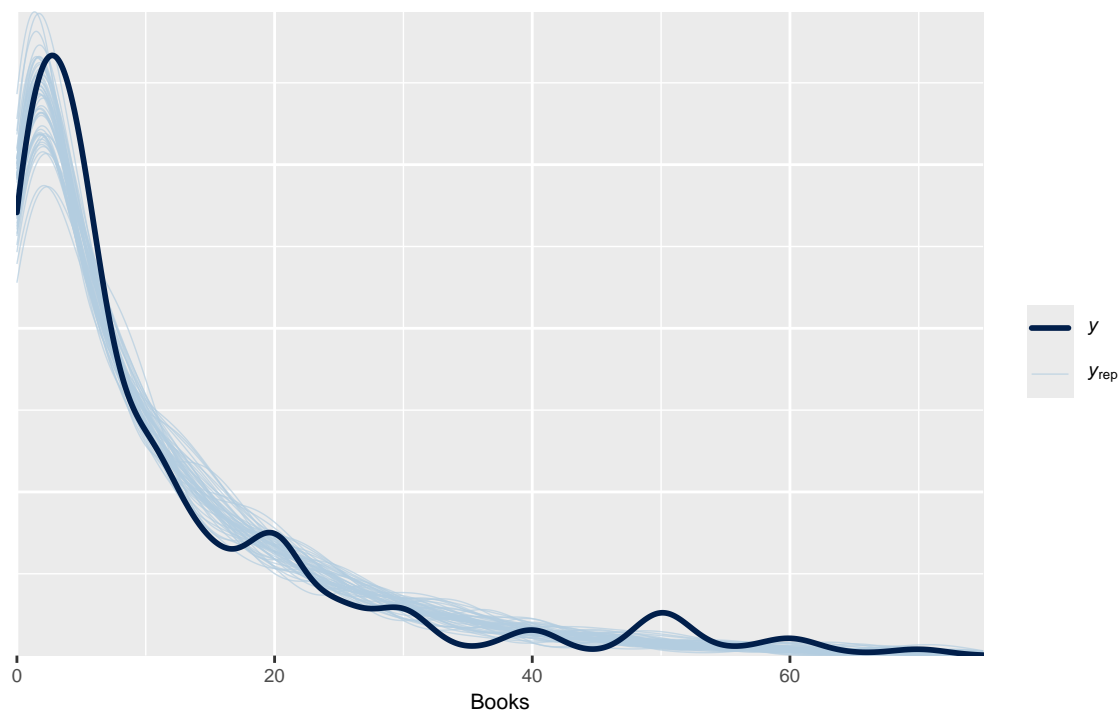
```

```
# Posterior predictive check for the Negative Binomial model
pp_check(books_negbin_sim) +
  xlim(0, 75) +
  xlab("Books") +
  ggtitle("Posterior Predictive Check (Negative Binomial Model)")
```

```
## Warning: Removed 231 rows containing non-finite outside the scale range
## ('stat_density()').
```

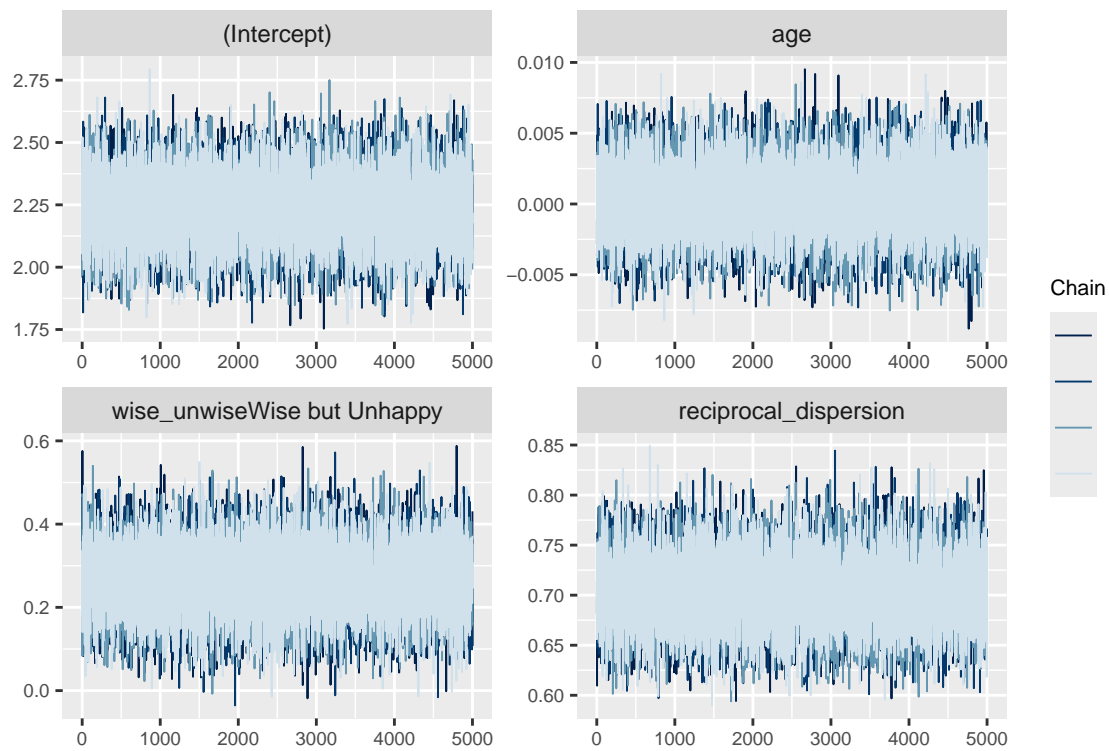
```
## Warning: Removed 3 rows containing non-finite outside the scale range
## ('stat_density()').
```

Posterior Predictive Check (Negative Binomial Model)

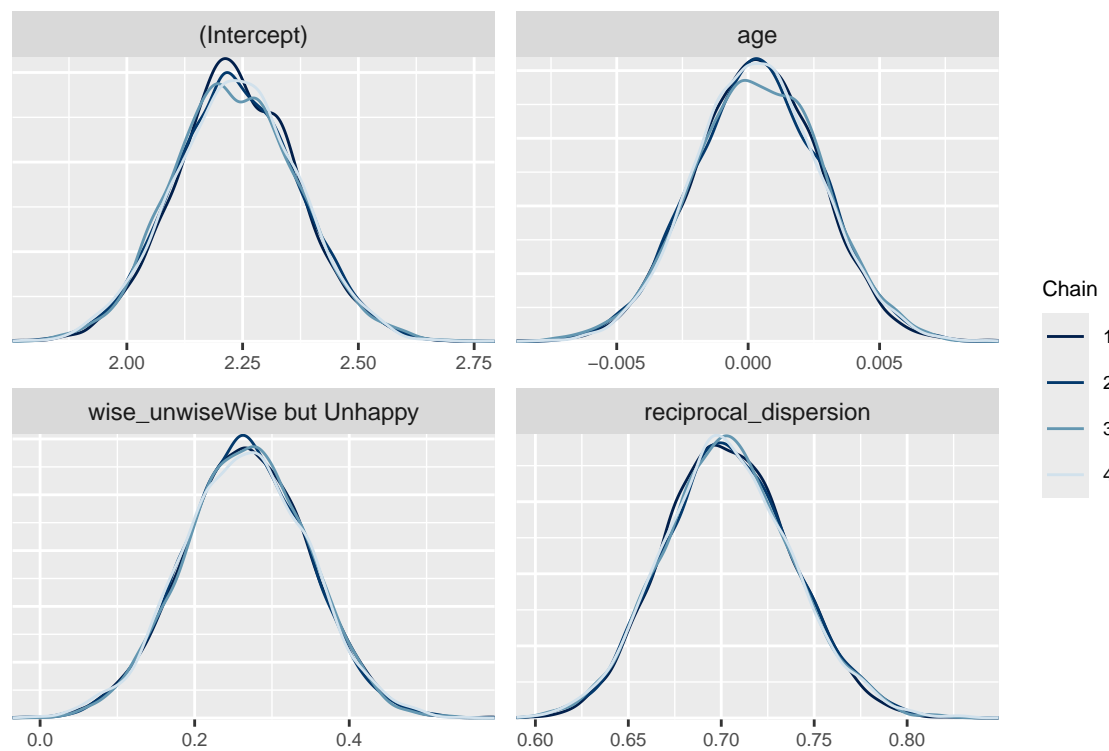


```
# MCMC diagnostics: trace plots, density overlays, and autocorrelation
mcmc_trace(books_negbin_sim)
```

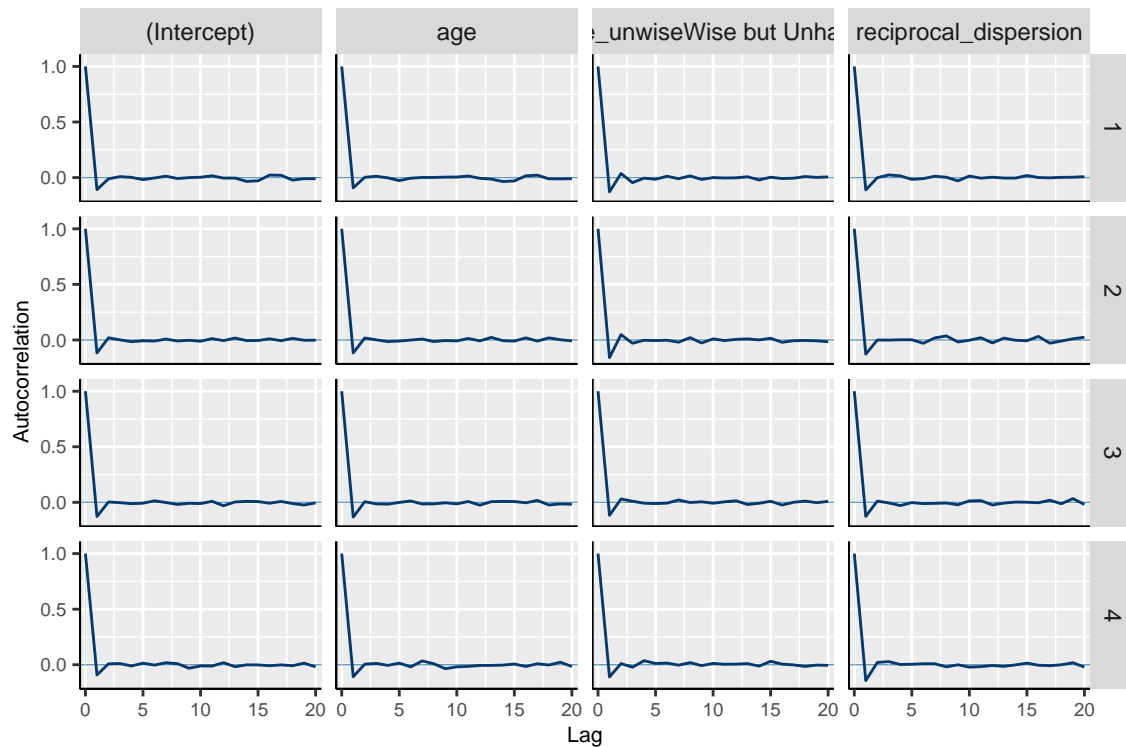




```
mcmc_dens_overlay(books_negbin_sim)
```



```
mcmc_acf(books_negbin_sim)
```



```
# Summarize posterior distributions of parameters
tidy(books_negbin_sim, conf.int = TRUE, conf.level = 0.95)
```

```
## # A tibble: 3 x 5
##   term                estimate std.error conf.low conf.high
##   <chr>              <dbl>      <dbl>   <dbl>   <dbl>
## 1 (Intercept)        2.23        0.134    1.98    2.50
## 2 age                0.000340    0.00242 -0.00435 0.00505
## 3 wise_unwiseWise but Unhappy 0.266      0.0814   0.108   0.423
```

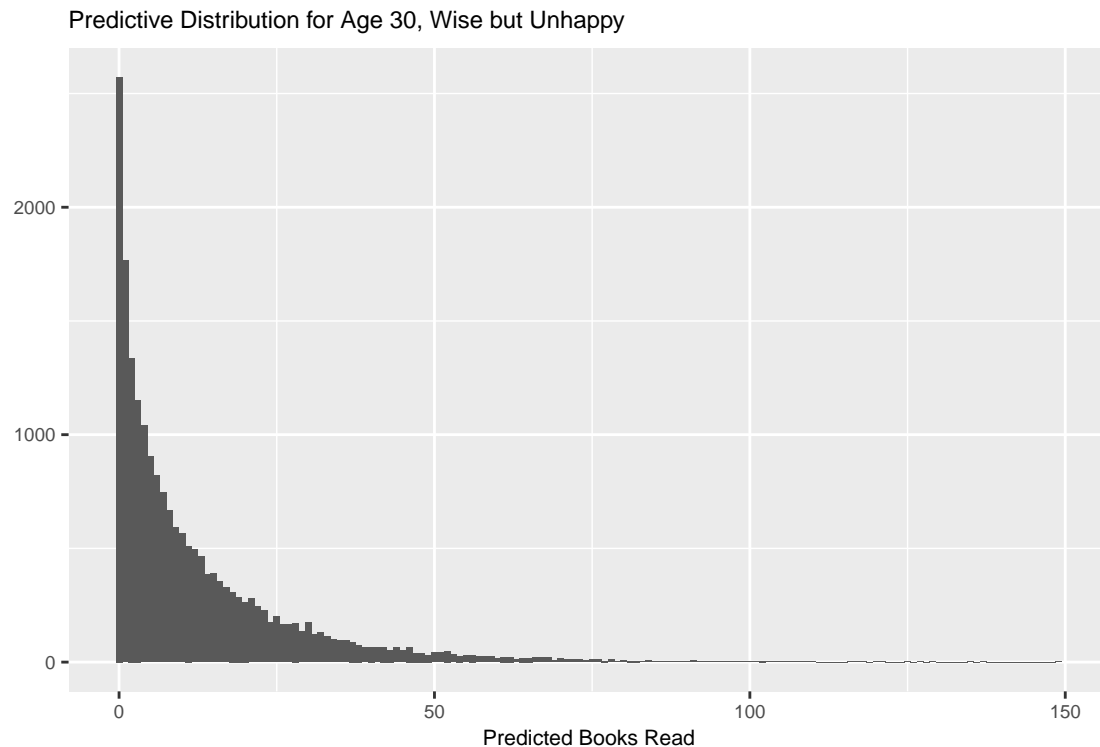
```
# Posterior prediction for a specific case: age 30, "Wise but Unhappy"
ex_pred_nb <- posterior_predict(
  books_negbin_sim, newdata = data.frame(age = 30, wise_unwise = 'Wise but Unhappy')
)
```

```
# Summary and visualization of the predictive distribution for this individual
pred_table <- table(ex_pred_nb) / length(ex_pred_nb)
post_pred_mean <- sum(as.numeric(names(pred_table)) * pred_table)
cat("Posterior Predictive Mean:", post_pred_mean, "\n")
```

```
## Posterior Predictive Mean: 12.36975
```

```
### Easiest to see with a plot here:
# Histogram of posterior predictive distribution for this individual:
qplot(ex_pred_nb, binwidth = 1, geom = "histogram") +
  xlab("Predicted Books Read") +
  ggtitle("Predictive Distribution for Age 30, Wise but Unhappy")
```

```
## Warning: 'qplot()' was deprecated in ggplot2 3.4.0.
## This warning is displayed once every 8 hours.
## Call 'lifecycle::last_lifecycle_warnings()' to see where this warning was
## generated.
```

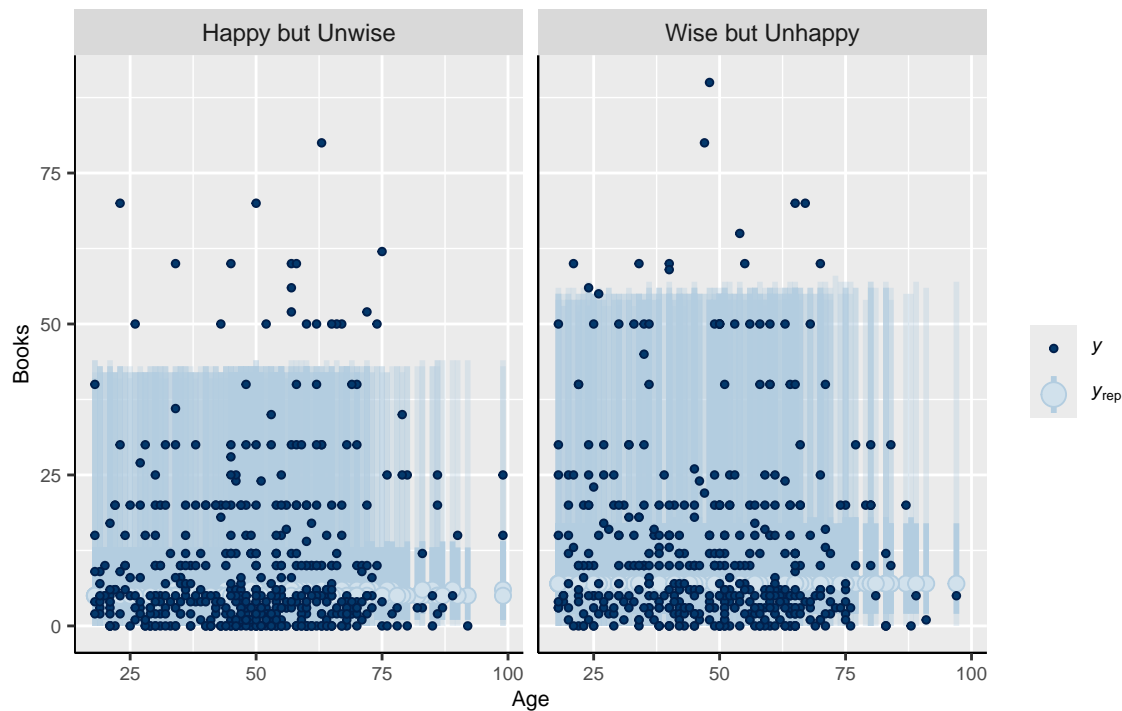


```
# Posterior predictive intervals for all observations
nb_preds <- posterior_predict(books_negbin_sim, newdata = pulse)

# Plot the posterior predictive models for each observation:
# Big data set, so this take a while...

ppc_intervals_grouped(
  pulse$books, yrep = nb_preds,
  x = pulse$age,
  group = pulse$wise_unwise,
  prob = 0.5, prob_outer = 0.95,
  facet_args = list(scales = "fixed")
) + labs(
  x = "Age",
  y = "Books",
  title = "Posterior Predictive Intervals by Age and Preference"
)
```

Posterior Predictive Intervals by Age and Preference



*# Do the dark blue circles (observed counts) fall near the light blue circles (predicted counts)?*  
*# Do most of the dark blue circles (observed counts) fall within the 95% prediction intervals?*

*# A numerical summary of in-sample model fit:*

```
in_sample_fit <- prediction_summary(model = books_negbin_sim, data = pulse)
print(in_sample_fit)
```

```
##           mae mae_scaled within_50 within_95
## 1 7.58945  0.5599491 0.5962145 0.9652997
```

*# Out-of-sample predictive accuracy with 10-fold cross-validation*

```
set.seed(84735)
nb_cv <- prediction_summary_cv(model = books_negbin_sim, data = pulse, k = 10)
print(nb_cv$cv)
```

```
##           mae mae_scaled within_50 within_95
## 1 7.694463  0.5947143 0.5941118 0.9664145
```

*# Model comparison using Leave-One-Out (LOO) cross-validation*

*# Comparing a couple of other models:*

*# Setting chains=1 for speed reasons...*

```
books_negbin <- stan_glm(
  books ~ wise_unwise,
  data = pulse, family = neg_binomial_2,
  prior_intercept = normal(0, 2.5, autoscale = TRUE),
  prior = normal(0, 2.5, autoscale = TRUE),
  prior_aux = exponential(1, autoscale = TRUE),
  chains = 1, iter = 10000
)
```

```
##
## SAMPLING FOR MODEL 'count' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0.000277 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 2.77 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 10000 [ 0%] (Warmup)
## Chain 1: Iteration: 1000 / 10000 [ 10%] (Warmup)
## Chain 1: Iteration: 2000 / 10000 [ 20%] (Warmup)
## Chain 1: Iteration: 3000 / 10000 [ 30%] (Warmup)
## Chain 1: Iteration: 4000 / 10000 [ 40%] (Warmup)
## Chain 1: Iteration: 5000 / 10000 [ 50%] (Warmup)
## Chain 1: Iteration: 5001 / 10000 [ 50%] (Sampling)
## Chain 1: Iteration: 6000 / 10000 [ 60%] (Sampling)
## Chain 1: Iteration: 7000 / 10000 [ 70%] (Sampling)
## Chain 1: Iteration: 8000 / 10000 [ 80%] (Sampling)
## Chain 1: Iteration: 9000 / 10000 [ 90%] (Sampling)
## Chain 1: Iteration: 10000 / 10000 [100%] (Sampling)
## Chain 1:
## Chain 1: Elapsed Time: 8.848 seconds (Warm-up)
## Chain 1:                10.57 seconds (Sampling)
## Chain 1:                19.418 seconds (Total)
## Chain 1:
```

```
books_negbin_interact <- stan_glm(
  books ~ age + wise_unwise + age:wise_unwise,
  data = pulse, family = neg_binomial_2,
  prior_intercept = normal(0, 2.5, autoscale = TRUE),
  prior = normal(0, 2.5, autoscale = TRUE),
  prior_aux = exponential(1, autoscale = TRUE),
  chains = 1, iter = 10000
)
```

```
##
## SAMPLING FOR MODEL 'count' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0.000274 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 2.74 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 10000 [ 0%] (Warmup)
## Chain 1: Iteration: 1000 / 10000 [ 10%] (Warmup)
## Chain 1: Iteration: 2000 / 10000 [ 20%] (Warmup)
## Chain 1: Iteration: 3000 / 10000 [ 30%] (Warmup)
## Chain 1: Iteration: 4000 / 10000 [ 40%] (Warmup)
## Chain 1: Iteration: 5000 / 10000 [ 50%] (Warmup)
## Chain 1: Iteration: 5001 / 10000 [ 50%] (Sampling)
## Chain 1: Iteration: 6000 / 10000 [ 60%] (Sampling)
## Chain 1: Iteration: 7000 / 10000 [ 70%] (Sampling)
## Chain 1: Iteration: 8000 / 10000 [ 80%] (Sampling)
## Chain 1: Iteration: 9000 / 10000 [ 90%] (Sampling)
```

```
## Chain 1: Iteration: 10000 / 10000 [100%] (Sampling)
## Chain 1:
## Chain 1: Elapsed Time: 23.95 seconds (Warm-up)
## Chain 1: 29.305 seconds (Sampling)
## Chain 1: 53.255 seconds (Total)
## Chain 1:
```

```
# This takes a while ....
```

```
# Compare models with LOO
```

```
loo(books_negbin_sim)
```

```
##
## Computed from 20000 by 951 log-likelihood matrix.
##
##      Estimate   SE
## elpd_loo -3231.4 38.2
## p_loo      4.2  0.3
## looic      6462.8 76.5
## -----
## MCSE of elpd_loo is 0.0.
## MCSE and ESS estimates assume MCMC draws (r_eff in [1.2, 1.4]).
##
## All Pareto k estimates are good (k < 0.7).
## See help('pareto-k-diagnostic') for details.
```

```
loo(books_negbin)
```

```
##
## Computed from 5000 by 951 log-likelihood matrix.
##
##      Estimate   SE
## elpd_loo -3230.4 38.2
## p_loo      3.1  0.2
## looic      6460.8 76.4
## -----
## MCSE of elpd_loo is 0.0.
## MCSE and ESS estimates assume MCMC draws (r_eff in [0.9, 1.1]).
##
## All Pareto k estimates are good (k < 0.7).
## See help('pareto-k-diagnostic') for details.
```

```
loo(books_negbin_interact)
```

```
##
## Computed from 5000 by 951 log-likelihood matrix.
##
##      Estimate   SE
## elpd_loo -3229.9 38.2
## p_loo      5.2  0.4
## looic      6459.8 76.4
## -----
```

```
## MCSE of elpd_loo is 0.0.
## MCSE and ESS estimates assume MCMC draws (r_eff in [0.5, 0.8]).
##
## All Pareto k estimates are good (k < 0.7).
## See help('pareto-k-diagnostic') for details.
```

*#This output provides diagnostics from the LOO-CV analysis of the model.*

*# elpd\_loo (Expected Log Predictive Density for LOO-CV) is a measure of the model's predictive accuracy*

*# Lower negative values indicate better predictive performance.*

*# p\_loo (Effective Number of Parameters) accounts for model complexity and flexibility.*

*# looic (Leave-One-Out Information Criterion) calculated as  $\text{looic} = -2 \times \text{elpd\_loo}$*

*# Similar to AIC or DIC, lower values suggest better predictive performance.*

*# Monte Carlo Standard Error (MCSE) of elpd\_loo, lower values indicate high precision in the LOO-CV estimates*

*# Effective Sample Size (ESS) and r\_eff indicates that the effective sample size is calculated under the assumption*

*# of "MCMC draws (r\_eff in [0.5, 0.8])" reflecting a good level of autocorrelation control in the MCMC chain*

*#Pareto k Estimates, low values indicate that the LOO-CV estimates are reliable.*

*# If k values were higher, it would signal potential issues with the stability of the LOO estimates.*

**## Fitting the model using regular R coding can be done with the negative binomial model,**

**## but the code is not as easy since there is an extra parameter (mean plus dispersion)**

**## rather than the one-parameter Poisson.**

**## I won't provide the base R code for it here...**