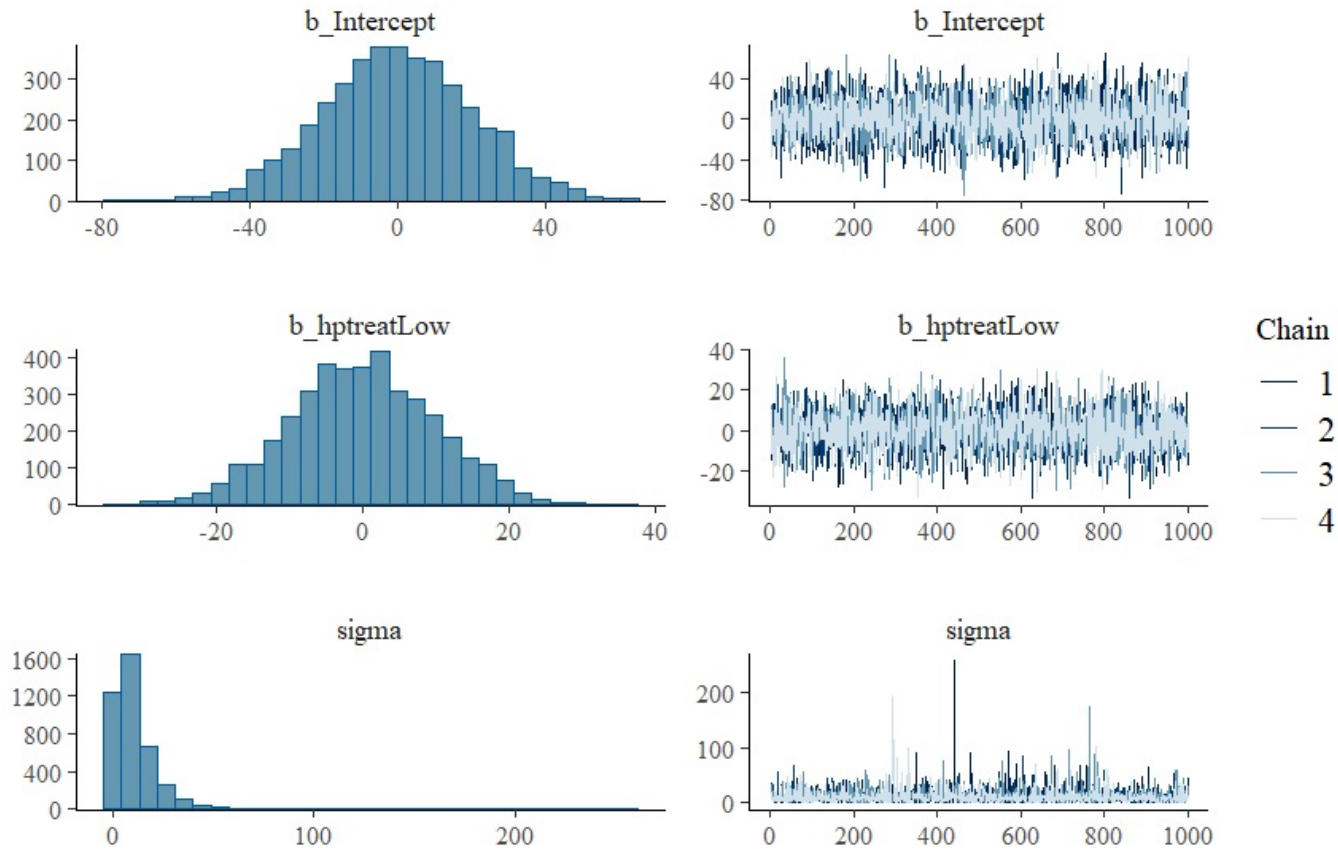
* Assignment-4 *

Part 1: A simple linear regression: Power posing and testosterone

1
```
library(brms)
library(dplyr)

getwd()
df_powerpose <- read.table("C:/Users/my pc/Downloads/df_powerpose.csv", header = TRUE, sep
= ",")
df_powerpose <- mutate(df_powerpose, change = testm2 - testm1)

# Specify weakly informative priors
priors <- c(
  set_prior("normal(0, 10)", class = "b"),          # Prior for regression coefficients
  set_prior("normal(0, 20)", class = "Intercept"), # Prior for intercept
  set_prior("student_t(3, 0, 10)", class = "sigma") # Prior for residual standard
deviation
)

# Perform a prior predictive check
prior_predict <- brm(change ~ hptreat, data = df_powerpose, prior = priors, sample_prior =
"only")
plot(prior_predict)

# Specify and fit the Bayesian linear regression model with priors
fit_powerpose <- brm(change ~ hptreat, data = df_powerpose, prior = priors)
print(fit_powerpose)

get_prior(change ~ hptreat, df_powerpose)
print(get_prior)
```



```
SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
Chain 1: 
Chain 1: Gradient evaluation took 5e-05 seconds
Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.5 seconds.
Chain 1: Adjust your expectations accordingly!
Chain 1: 
Chain 1: 
Chain 1: Iteration:    1 / 2000 [  0%]  (Warmup)
Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
Chain 1: Iteration:  600 / 2000 [ 30%]  (Warmup)
Chain 1: Iteration:  800 / 2000 [ 40%]  (Warmup)
Chain 1: Iteration: 1000 / 2000 [ 50%]  (Warmup)
Chain 1: Iteration: 1001 / 2000 [ 50%]  (Sampling)
Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
Chain 1: 
Chain 1:  Elapsed Time: 0.022 seconds (Warm-up)
Chain 1:                0.012 seconds (Sampling)
Chain 1:                0.034 seconds (Total)
Chain 1: 

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 2).
Chain 2: 
Chain 2: Gradient evaluation took 2e-06 seconds
Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.02 seconds.
Chain 2: Adjust your expectations accordingly!
Chain 2: 
Chain 2: 
Chain 2: Iteration:    1 / 2000 [  0%]  (Warmup)
Chain 2: Iteration:  200 / 2000 [ 10%]  (Warmup)
Chain 2: Iteration:  400 / 2000 [ 20%]  (Warmup)
Chain 2: Iteration:  600 / 2000 [ 30%]  (Warmup)
Chain 2: Iteration:  800 / 2000 [ 40%]  (Warmup)
Chain 2: Iteration: 1000 / 2000 [ 50%]  (Warmup)
Chain 2: Iteration: 1001 / 2000 [ 50%]  (Sampling)
Chain 2: Iteration: 1200 / 2000 [ 60%]  (Sampling)
Chain 2: Iteration: 1400 / 2000 [ 70%]  (Sampling)
Chain 2: Iteration: 1600 / 2000 [ 80%]  (Sampling)
Chain 2: Iteration: 1800 / 2000 [ 90%]  (Sampling)
Chain 2: Iteration: 2000 / 2000 [100%]  (Sampling)
Chain 2: 
Chain 2:  Elapsed Time: 0.019 seconds (Warm-up)
Chain 2:                0.013 seconds (Sampling)
Chain 2:                0.032 seconds (Total)
Chain 2: 

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 3).
Chain 3: 
Chain 3: Gradient evaluation took 3e-06 seconds
Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.03 seconds.
Chain 3: Adjust your expectations accordingly!
Chain 3: 
Chain 3: 
Chain 3: Iteration:    1 / 2000 [  0%]  (Warmup)
Chain 3: Iteration:  200 / 2000 [ 10%]  (Warmup)
Chain 3: Iteration:  400 / 2000 [ 20%]  (Warmup)
Chain 3: Iteration:  600 / 2000 [ 30%]  (Warmup)
Chain 3: Iteration:  800 / 2000 [ 40%]  (Warmup)
Chain 3: Iteration: 1000 / 2000 [ 50%]  (Warmup)
Chain 3: Iteration: 1001 / 2000 [ 50%]  (Sampling)
Chain 3: Iteration: 1200 / 2000 [ 60%]  (Sampling)
Chain 3: Iteration: 1400 / 2000 [ 70%]  (Sampling)
Chain 3: Iteration: 1600 / 2000 [ 80%]  (Sampling)
Chain 3: Iteration: 1800 / 2000 [ 90%]  (Sampling)
Chain 3: Iteration: 2000 / 2000 [100%]  (Sampling)
Chain 3: 
Chain 3:  Elapsed Time: 0.018 seconds (Warm-up)
Chain 3:                0.016 seconds (Sampling)
Chain 3:                0.034 seconds (Total)
Chain 3: 

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 4).
Chain 4: 
Chain 4: Gradient evaluation took 2e-06 seconds
Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.02 seconds.
Chain 4: Adjust your expectations accordingly!
Chain 4: 
Chain 4: 
Chain 4: Iteration:    1 / 2000 [  0%]  (Warmup)
Chain 4: Iteration:  200 / 2000 [ 10%]  (Warmup)
Chain 4: Iteration:  400 / 2000 [ 20%]  (Warmup)
Chain 4: Iteration:  600 / 2000 [ 30%]  (Warmup)
Chain 4: Iteration:  800 / 2000 [ 40%]  (Warmup)
Chain 4: Iteration: 1000 / 2000 [ 50%]  (Warmup)
Chain 4: Iteration: 1001 / 2000 [ 50%]  (Sampling)
Chain 4: Iteration: 1200 / 2000 [ 60%]  (Sampling)
Chain 4: Iteration: 1400 / 2000 [ 70%]  (Sampling)
Chain 4: Iteration: 1600 / 2000 [ 80%]  (Sampling)
Chain 4: Iteration: 1800 / 2000 [ 90%]  (Sampling)
Chain 4: Iteration: 2000 / 2000 [100%]  (Sampling)
Chain 4: 
Chain 4:  Elapsed Time: 0.018 seconds (Warm-up)
Chain 4:                0.011 seconds (Sampling)
Chain 4:                0.029 seconds (Total)
Chain 4: 
> plot(prior_predict)
> 
> # Specify and fit the Bayesian linear regression model with priors
> fit_powerpose <- brm(change ~ hptreat, data = df_powerpose, prior = priors)
Compiling Stan program...
Start sampling

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
Chain 1: 
Chain 1: Gradient evaluation took 7.6e-05 seconds
Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.76 seconds.
Chain 1: Adjust your expectations accordingly!
Chain 1: 
Chain 1: 
Chain 1: Iteration:    1 / 2000 [  0%]  (Warmup)
Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
Chain 1: Iteration:  600 / 2000 [ 30%]  (Warmup)
Chain 1: Iteration:  800 / 2000 [ 40%]  (Warmup)
Chain 1: Iteration: 1000 / 2000 [ 50%]  (Warmup)
Chain 1: Iteration: 1001 / 2000 [ 50%]  (Sampling)
Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
Chain 1: 
Chain 1:  Elapsed Time: 0.055 seconds (Warm-up)
Chain 1:                0.018 seconds (Sampling)
Chain 1:                0.073 seconds (Total)
Chain 1: 

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 2).
Chain 2: 
Chain 2: Gradient evaluation took 8e-06 seconds
Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.08 seconds.
Chain 2: Adjust your expectations accordingly!
Chain 2: 
Chain 2: 
Chain 2: Iteration:    1 / 2000 [  0%]  (Warmup)
Chain 2: Iteration:  200 / 2000 [ 10%]  (Warmup)
Chain 2: Iteration:  400 / 2000 [ 20%]  (Warmup)
Chain 2: Iteration:  600 / 2000 [ 30%]  (Warmup)
Chain 2: Iteration:  800 / 2000 [ 40%]  (Warmup)
Chain 2: Iteration: 1000 / 2000 [ 50%]  (Warmup)
Chain 2: Iteration: 1001 / 2000 [ 50%]  (Sampling)
Chain 2: Iteration: 1200 / 2000 [ 60%]  (Sampling)
Chain 2: Iteration: 1400 / 2000 [ 70%]  (Sampling)
Chain 2: Iteration: 1600 / 2000 [ 80%]  (Sampling)
Chain 2: Iteration: 1800 / 2000 [ 90%]  (Sampling)
Chain 2: Iteration: 2000 / 2000 [100%]  (Sampling)
Chain 2: 
Chain 2:  Elapsed Time: 0.037 seconds (Warm-up)
Chain 2:                0.02 seconds (Sampling)
Chain 2:                0.057 seconds (Total)
Chain 2: 
```

```
SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 3).
Chain 3:
Chain 3: Gradient evaluation took 1.2e-05 seconds
Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.12 seconds.
Chain 3: Adjust your expectations accordingly!
Chain 3:
Chain 3:
Chain 3: Iteration:    1 / 2000 [  0%]  (Warmup)
Chain 3: Iteration:  200 / 2000 [ 10%]  (Warmup)
Chain 3: Iteration:  400 / 2000 [ 20%]  (Warmup)
Chain 3: Iteration:  600 / 2000 [ 30%]  (Warmup)
Chain 3: Iteration:  800 / 2000 [ 40%]  (Warmup)
Chain 3: Iteration: 1000 / 2000 [ 50%]  (Warmup)
Chain 3: Iteration: 1001 / 2000 [ 50%]  (Sampling)
Chain 3: Iteration: 1200 / 2000 [ 60%]  (Sampling)
Chain 3: Iteration: 1400 / 2000 [ 70%]  (Sampling)
Chain 3: Iteration: 1600 / 2000 [ 80%]  (Sampling)
Chain 3: Iteration: 1800 / 2000 [ 90%]  (Sampling)
Chain 3: Iteration: 2000 / 2000 [100%]  (Sampling)
Chain 3:
Chain 3:  Elapsed Time: 0.049 seconds (Warm-up)
Chain 3:                0.023 seconds (Sampling)
Chain 3:                0.072 seconds (Total)
Chain 3:

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 4).
Chain 4:
Chain 4: Gradient evaluation took 1.4e-05 seconds
Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.14 seconds.
Chain 4: Adjust your expectations accordingly!
Chain 4:
Chain 4:
Chain 4: Iteration:    1 / 2000 [  0%]  (Warmup)
Chain 4: Iteration:  200 / 2000 [ 10%]  (Warmup)
Chain 4: Iteration:  400 / 2000 [ 20%]  (Warmup)
Chain 4: Iteration:  600 / 2000 [ 30%]  (Warmup)
Chain 4: Iteration:  800 / 2000 [ 40%]  (Warmup)
Chain 4: Iteration: 1000 / 2000 [ 50%]  (Warmup)
Chain 4: Iteration: 1001 / 2000 [ 50%]  (Sampling)
Chain 4: Iteration: 1200 / 2000 [ 60%]  (Sampling)
Chain 4: Iteration: 1400 / 2000 [ 70%]  (Sampling)
Chain 4: Iteration: 1600 / 2000 [ 80%]  (Sampling)
Chain 4: Iteration: 1800 / 2000 [ 90%]  (Sampling)
Chain 4: Iteration: 2000 / 2000 [100%]  (Sampling)
Chain 4:
Chain 4:  Elapsed Time: 0.04 seconds (Warm-up)
Chain 4:                0.019 seconds (Sampling)
Chain 4:                0.059 seconds (Total)
Chain 4:
>
> # Print the model summary
> print(fit_powerpose)
```

```
 Family: gaussian
  Links: mu = identity; sigma = identity
Formula: change ~ hptreat
   Data: df_powerpose (Number of observations: 39)
  Draws: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
         total post-warmup draws = 4000

Regression Coefficients:
           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
Intercept      3.19      4.13    -5.07    11.32 1.00     3648     3031
hptreatLow    -6.32      5.43   -17.10     4.24 1.00     3296     2735

Further Distributional Parameters:
      Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
sigma    20.19      2.31    16.25    25.17 1.00     3249     3088

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

Answers

```
>
> get_prior(change ~ hptreat, df_powerpose)
                prior     class       coef group resp dpar nlpar lb ub       source
               (flat)         b                                             default
               (flat)         b hptreatLow                             (vectorized)
 student_t(3, -1.9, 18.7) Intercept                                         default
    student_t(3, 0, 18.7)     sigma                                 0       default
> print(get_prior)
function (formula, ...)
{
    default_prior(formula, ...)
}
<bytecode: 0x00000227daea9fe0>
<environment: namespace:brms>
```

Part 2: Poisson regression models and hypothesis testing

2.1

```
#this function will return the required number of crossings
expected_crossings <- function(Li, alpha, beta) {
  lambda_i <- exp(alpha + beta * Li)
  Ni <- rpois(1, lambda = lambda_i)
  return(Ni)
}
expected_crossings(Li, alpha, beta)
```

2.2

```
# Set seed for reproducibility
set.seed(123)

# Number of prior samples to generate
num_samples <- 1000

# Prior distributions for alpha and beta
alpha_prior_mean <- 0.15
alpha_prior_sd <- 0.1
beta_prior_mean <- 0.25
beta_prior_sd <- 0.05

# Generate prior samples for alpha and beta
alpha_samples <- rnorm(num_samples, mean = alpha_prior_mean, sd = alpha_prior_sd)
beta_samples <- rnorm(num_samples, mean = beta_prior_mean, sd = beta_prior_sd)

# Function to simulate number of crossings
simulate_crossings <- function(L, alpha_samples, beta_samples) {
  # Calculate lambda_i for each sample
  lambda_i <- exp(alpha_samples + beta_samples * L)

  # Simulate number of crossings Ni from Poisson distribution for each lambda_i
  Ni_samples <- rpois(num_samples, lambda = lambda_i)

  return(Ni_samples)
}

#          Generate prior predictions for sentences of length 4
L <- 4
prior_predictions <- simulate_crossings(L, alpha_samples, beta_samples)

# Summary of prior predictions
print((prior_predictions))
print(summary(prior_predictions))
```

```
> print(summary(prior_predictions))
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  0.000   2.000   3.000   3.184   4.000  11.000 
```

```
> print((prior_predictions))
   [1]  2  4  4  5  1  2  6  0  4  6  5 11  0  2  2  7  8  3  2  4  1  4  5  3  3  4  6  5  2
  [30]  5  4  3  1  4  4  5  2  5  4  5  2  4  5  4  8  0  2  1  4  5  2  3  3  2  4  5  6  2
  [59]  2  4  2  1  3  2  4  3  4  2  2  5  3  3  3  1  5  8  5  4  2  3  5  1  4  4  1  5  2
  [88]  5  4  3  4  1  3  1  3  3  3  9  2  3  3  2  2  8  4  2  5  4  2  5  3  3  4  3  5  3
 [117]  3  3  4  4  5  4  2  2  6  9  4  4  3  0  3  3  3  5  2  0  0  3  4  4  4  4  2  1  1
 [146]  3  1  3  4  1  1  0  5  4  0  3  2  4  1  5  2  1  1  3  3  1  3  3  2 10  0  6  1  2
 [175]  2  4  1  1  1  1  5  4  3  4  2  3  2  0  4  3  2  5  3  4  2  1  4  6  3  3  4  1  4
 [204]  3  5  4  2  6  4  1  1  5  3  2  2  5  2  1  3  1  3  2  1  2  1  3  3  3  6  2  2  5
 [233]  2  4  3  1  2  0  4  3  2  5  3  0  4  2  5  0  6  5  6  3  2  6  4  5  3  4  4  2  3
 [262]  3  2  4  3  7  6  3  0  7  4  6  4  2  1  1  4  3  0  1  4  4  1  3  0  3  2  3  2  3
 [291]  3  4  2  2  4  0  2  2  3  4  3 11  5  6  4  5  5  2  4  1  5  4  0  1  1  5  2  2  6
 [320]  2  2  2  5  7  1  3  3  3  4  1  2  6  3  4  2  1  4  2  3  1  3  8  0  2  1  2  0  4
 [349]  0  2  2  2  2  9  4  4  5  2  3  7  7  1  3  4  2  5  7  4  4  5  1  1  2  4  4  5  1
 [378]  0  3  4  3  3  2  2  3  2  4  2  4  3  5  5  4  4  2  1  3  6  4  3  7  7  2  1  3  3
 [407]  5  5  4  4  2  3  4  2  2  5  2  1  5  3  4  5  4  5  2  9  8  1  5  2  3  0  5  1  0
 [436]  4  2  1  5  4  2  2  4  4  3  1  0  2  1  3  6  3  7  5  2  3  5  3  2  0  1  4  2  1
 [465]  6  6  5  5  4  4  6  3  2  3  3  2  5  2  1  2  2  2  4  7  3  3  2  3  3  4  0  0  4
 [494]  3  2  1  5  4  1  3  1  2  3  4  1  4  0  1  5  5  4  0  2  5  2  2  1  2  3  3  3  5
 [523]  1  4  2  2  3  4  4  3  6  4  3  6  2  4  1  2  4  5  3  2  3  0  5  5  1  4  5  2  3
 [552]  4  2  1  4  1  4  3  2  1  1  4  2  1  0  4  3  7  4  5  4  5  3  2  3  7  5  4  4  1
 [581]  0  3  2  7  5  4  2  3  6  5  2  1  4  4  3  1  2  1  2  5  4  6  2  1  2  1  6  4  3
 [610]  2  6  2  9  4  6  3  2  3  1  6  2  5  6  6  4  3  1  6  1  7  2  2  3  3  4  4  6  4
 [639]  1  4  0  1  1  5  5  1  3  4  5  7  0  1  8  5  2  2  6  1  5  2  7  5  5  1  2  2  2
 [668]  6  2  5  2  4  4  1  4  2  3  3  3  3  2  0  2  5  0  7  4  4  2  2  3  1  1  3  2  5
 [697]  1  3  1  3  0  4  3  3  4  3  1  5  6  8  4  0  4  3  3  0  2  2  4  5  4  2  4  2  6
 [726]  3  2  1  1  6  5  5  9  4  2  4  6  7  2  5  2  3  2  6  3  3  6  0  6  5  4  6  5  5
 [755]  8  4  5  3  0  3  1  2  1  4  0  1  6  8  3  4  3  3  8  4  1  2  3  3  4  4  4  3  2
 [784]  2  4  2  5  4  2  2  1  2  6  3  6  0  6  3  5  2  2  2  2  2  3  2  2  0  4  2  3  1
 [813]  0  1  0  6  3  6  3  1  0  4  2  4  3  4  5  1  0  4  1  2  1  1  3  1  2  5  3  3  1
 [842]  4  2  2  3  3  2  3  7  2  1  5  6  1  3  1  1  4  4  2  2  5  7  0  3  7  2  2  4  4
 [871]  7  5  4  0  3  4  3  1  2  3  2  2  6  3  3  3  2  0  5  6  4  3  2  3  3  3  4  1  3
 [900]  1  5  6  7  8  3  1  3  5  3  4  6  3  5  2  3  4  5  3  1  4  2  2  1  2  1  7  2  1
 [929]  5  4  3  3  5  4  5  2  4  2  1  2  2  0  3  1  5  4  2  4  0  6  4  1  3  4  3  3  3
 [958]  2  5  2  5  8  2  4  2  4  1  1  1  1  8  1  5  6  2  5  2  4  1  2  9  1  3  3  3  2
 [987]  5  5  1  4  4  1  6  3  2  1  7  2  4  0
```

2.3

```
# Load necessary libraries
library(brms)
crossings_data <- read.csv("C:/Users/my pc/Downloads/crossings.csv", header = TRUE, sep =
",")
head(crossings_data)
summary(crossings_data)

# Model M1 formula adjusted
formula_M1 <- bf(nCross ~ s.length + (1 | Language), family = poisson)

# Prior specifications
prior_M1 <- c(
  prior(normal(0.15, 0.1), class = Intercept),
  prior(normal(0, 0.15), class = b)
)

# Fit Model M1
fit_M1 <- brm(formula_M1, data = crossings_data, family = poisson, prior = prior_M1)

# Summary of Model M1
summary(fit_M1)
# Create indicator variable Rj (0 for English, 1 for German)
crossings_data$Rj <- as.integer(crossings_data$Language == "German")

# Model M2 formula directly using s.length * Rj
formula_M2 <- bf(nCross ~ s.length * Rj + (1 | Language), family = poisson)

# Prior specifications (assuming previously defined priors)
prior_M2 <- c(
  prior(normal(0.15, 0.1), class = Intercept),
  prior(normal(0, 0.15), class = b),
  prior(normal(0, 0.15), class = b, coef = "Rj"),
  prior(normal(0, 0.15), class = b, coef = "s.length:Rj")
)

# Fit Model M2
fit_M2 <- brm(formula_M2, data = crossings_data, family = poisson, prior = prior_M2)

# Summary of Model M2
summary(fit_M2)
```

```
> # Summary of Model M1
> summary(fit_M1)
 Family: poisson
  Links: mu = log
Formula: nCross ~ s.length + (1 | Language)
   Data: crossings_data (Number of observations: 1900)
  Draws: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
         total post-warmup draws = 4000

Multilevel Hyperparameters:
~Language (Number of levels: 2)
              Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
sd(Intercept)     0.46      0.32     0.13     1.29 1.01      569      385

Regression Coefficients:
          Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
Intercept    -1.49      0.10    -1.68    -1.29 1.01     1015     1229
s.length      0.15      0.00     0.14     0.16 1.00     1510     1316

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

```
> # Summary of Model M2
> summary(fit_M2)
 Family: poisson
  Links: mu = log
Formula: nCross ~ s.length * Rj + (1 | Language)
   Data: crossings_data (Number of observations: 1900)
  Draws: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
         total post-warmup draws = 4000

Multilevel Hyperparameters:
~Language (Number of levels: 2)
              Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
sd(Intercept)     1.13      0.69     0.27     2.51 1.09       36      905

Regression Coefficients:
            Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
Intercept      -1.43      0.12    -1.67    -1.17 1.04      738     1361
s.length        0.10      0.01     0.09     0.11 1.07     1746     1887
Rj              0.01      0.17    -0.31     0.26 1.11       23      358
s.length:Rj     0.09      0.01     0.08     0.11 1.07     1778     2011

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

2.4

```
# Load required libraries
library(brms)
library(tidyverse)  # For data manipulation and visualization

observed <- read.csv("C:/Users/my pc/Downloads/crossings.csv", header = TRUE, sep = ",")
# Read the dataset

# Visualize average rate of crossings
observed %>%
  group_by(Language, s.length) %>%
  summarise(mean.crossings = mean(nCross)) %>%
  ggplot(aes(x = s.length, y = mean.crossings, group = Language, color = Language)) +
  geom_point() + geom_line() +
  labs(x = "Sentence Length", y = "Mean Crossings", title = "Average Rate of Crossings by
Sentence Length")

# Center the predictors
observed$s.length <- observed$s.length - mean(observed$s.length)
observed$lang <- ifelse(observed$Language == "German", 1, 0)

# Initialize vectors to store log predictive densities in each fold
lpds.m1 <- c()
lpds.m2 <- c()

# Define the number of folds for cross-validation
k <- 5

# Perform k-fold cross-validation
for (fold in 1:k) {
  # Prepare test data and training data for this fold
  set.seed(123 + fold)  # Set seed for reproducibility
  fold_size <- nrow(observed) %/% k
  fold_indices <- ((fold - 1) * fold_size + 1):(fold * fold_size)

  ytest <- observed[fold_indices, ]
  ytrain <- observed[-fold_indices, ]

  # Fit Model M1 on training data
  fit.m1 <- brm(
    formula = nCross ~ 1 + s.length,
    data = ytrain,
    family = poisson(link = "log"),
    prior = c(prior(normal(0.15, 0.1), class = Intercept),
              prior(normal(0, 0.15), class = b)),
    chains = 4, cores = 4
  )

  # Fit Model M2 on training data
  fit.m2 <- brm(
    formula = nCross ~ 1 + s.length + lang + s.length * lang,
    data = ytrain,
    family = poisson(link = "log"),
    prior = c(prior(normal(0.15, 0.1), class = Intercept),
              prior(normal(0, 0.15), class = b)),
    chains = 4, cores = 4
  )

  # Retrieve posterior samples
  post.m1 <- posterior_samples(fit.m1)
  post.m2 <- posterior_samples(fit.m2)

  # Calculate log pointwise predictive density (lppd) using test data
  lppd.m1 <- 0
  lppd.m2 <- 0
```

```
  for (i in 1:nrow(ytest)) {
    lpd_im1 <- log(mean(dpois(ytest[i, ]$nCross,
                             lambda = exp(post.m1[, 1] + post.m1[, 2] * ytest[i,
]$s.length))))
    lppd.m1 <- lppd.m1 + lpd_im1

    lpd_im2 <- log(mean(dpois(ytest[i, ]$nCross,
                             lambda = exp(post.m2[, 1] +
                                          post.m2[, 2] * ytest[i, ]$s.length +
                                          post.m2[, 3] * ytest[i, ]$lang +
                                          post.m2[, 4] * ytest[i, ]$s.length * ytest[i,
]$lang))))
    lppd.m2 <- lppd.m2 + lpd_im2
  }

  # Store lppd values for this fold
  lpds.m1 <- c(lpds.m1, lppd.m1)
  lpds.m2 <- c(lpds.m2, lppd.m2)
}

# Calculate expected log predictive density (elpd) for each model
elpd.m1 <- sum(lpds.m1)
elpd.m2 <- sum(lpds.m2)

# Calculate evidence in favor of M2 over M1
difference_elpd <- elpd.m2 - elpd.m1

# Print results
cat("Expected Log Predictive Density (elpd) for Model M1:", elpd.m1, "\n")
cat("Expected Log Predictive Density (elpd) for Model M2:", elpd.m2, "\n")
cat("Difference in elpd (M2 - M1):", difference_elpd, "\n")
```

```
> # Print results
> cat("Expected Log Predictive Density (elpd) for Model M1:", elpd.m1, "\n")
Expected Log Predictive Density (elpd) for Model M1: -3042.343
> cat("Expected Log Predictive Density (elpd) for Model M2:", elpd.m2, "\n")
Expected Log Predictive Density (elpd) for Model M2: -2684.654
> cat("Difference in elpd (M2 - M1):", difference_elpd, "\n")
Difference in elpd (M2 - M1): 357.689
```

This clearly indicates M2 is better in accuracy over M1