```r
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.2.0     v readr     2.1.6
## v forcats   1.0.1     v stringr   1.6.0
## v ggplot2   4.0.2     v tibble    3.3.1
## v lubridate 1.9.5     v tidyr     1.3.2
## v purrr     1.2.1
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become error
```

```r
library(brms)
```

```
## Loading required package: Rcpp
## Loading 'brms' package (version 2.23.0). Useful instructions
## can be found by typing help('brms'). A more detailed introduction
## to the package is available through vignette('brms_overview').
##
## Attaching package: 'brms'
##
## The following object is masked from 'package:stats':
##
##     ar
```

```r
library(loo)
```

```
## This is loo version 2.9.0
## - Online documentation and vignettes at mc-stan.org/loo
## - As of v2.0.0 loo defaults to 1 core but we recommend using as many as possible. Use the 'cores' ar
```

```r
library(bayesplot)
```

```
## This is bayesplot version 1.15.0
## - Online documentation and vignettes at mc-stan.org/bayesplot
## - bayesplot theme set to bayesplot::theme_default()
##    * Does _not_ affect other ggplot2 plots
##    * See ?bayesplot_theme_set for details on theme setting
##
## Attaching package: 'bayesplot'
##
## The following object is masked from 'package:brms':
##
##     rhat
```

```r
library(rstan)
```

```
## Loading required package: StanHeaders
##
```

```
## rstan version 2.32.7 (Stan version 2.32.2)
##
## For execution on a local, multicore CPU with excess RAM we recommend calling
## options(mc.cores = parallel::detectCores()).
## To avoid recompilation of unchanged Stan programs, we recommend calling
## rstan_options(auto_write = TRUE)
## For within-chain threading using `reduce_sum()` or `map_rect()` Stan functions,
## change `threads_per_chain` option:
## rstan_options(threads_per_chain = 1)
##
##
## Attaching package: 'rstan'
##
## The following object is masked from 'package:tidyr':
##
##     extract
```

```
rstan_options(auto_write = TRUE)
options(mc.cores = parallel::detectCores())
```

```
df_p1 <- read.csv("/Users/debarpita/Desktop/arjun/trial_wise_dataset_post.csv")
df_p2 <- read.csv("/Users/debarpita/Desktop/arjun/trial_wise_dataset_pre.csv")
library(dplyr)
df <- bind_rows(df_p1, df_p2)
```

```
cat(sprintf("Total observations: %d\n", nrow(df)))
```

```
## Total observations: 6016
```

```
cat(sprintf("Pre-stim: %d\n", sum(df$stim_cat == "pre")))
```

```
## Pre-stim: 2882
```

```
cat(sprintf("Post-stim: %d\n", sum(df$stim_cat == "post")))
```

```
## Post-stim: 3134
```

```
# IQR-based outlier removal on HandlingTime (global)
Q1  <- quantile(df$HandlingTime, 0.25, na.rm = TRUE)
Q3  <- quantile(df$HandlingTime, 0.75, na.rm = TRUE)
IQR <- Q3 - Q1
df  <- df %>%
  filter(HandlingTime >= Q1 - 1.5 * IQR,
         HandlingTime <= Q3 + 1.5 * IQR)

cat(sprintf("After outlier removal: %d\n", nrow(df)))
```

```
## After outlier removal: 5079
```

```r
# Safety offset on HandlingTime to prevent division-by-zero
df <- df %>%
  mutate(HandlingTime_safe = HandlingTime + 1e-6)

# Trial index within each patch
df <- df %>%
  group_by(Participant_ID, stim_cat, env, patch_id) %>%
  mutate(trial_in_patch = row_number()) %>%
  ungroup()

# Lagged average reward rate (from previous trial within participant x stim x env)
df <- df %>%
  group_by(Participant_ID, stim_cat, env) %>%
  mutate(AvgRewardRate_before = lag(AvgRewardRate)) %>%
  ungroup() %>%
  filter(!is.na(AvgRewardRate_before))
```

```r
df <- df %>%
  mutate(
    tt_base_env1 = ifelse(stim_cat == "pre",  3,  5),
    tt_base_env2 = ifelse(stim_cat == "pre", 10, 12),
    rew_base    = ifelse(stim_cat == "pre",  91, 181),
    rew_hi      = ifelse(stim_cat == "pre",   9,  19),
    rew_dec     = ifelse(stim_cat == "pre",  10,  20)
  )
```

```r
# ----------------------------------------------------------------------
# THEORETICAL BOUNDS - computed per trial using task parameters
# R_max           : highest possible reward (start of a fresh patch)
# R_min_theo      : expected floor at this trial depth (can reach 0)
# Both are condition-specific (pre vs post) via the parameters above
# ----------------------------------------------------------------------
df <- df %>%
  mutate(
    R_max_theo      = rew_base + rew_hi,
    R_min_theo      = pmax(rew_base - rew_dec * trial_in_patch, 0)
  )
```

```r
# ----------------------------------------------------------------------
# EMPIRICAL BOUNDS - computed within Participant x stim_cat x env
# This respects both condition AND environment simultaneously
# ----------------------------------------------------------------------
df <- df %>%
  group_by(Participant_ID, stim_cat, env) %>%
  mutate(
    R_min_empirical = min(Reward, na.rm = TRUE),
    R_max_empirical = max(Reward, na.rm = TRUE)
  ) %>%
  ungroup()

# PATCH-LEVEL empirical max - for patch-wise pct-of-max normalization
df <- df %>%
  group_by(Participant_ID, stim_cat, env, patch_id) %>%
  mutate(
    R_max_empirical_patch = max(Reward, na.rm = TRUE)
  ) %>%
  ungroup()
```

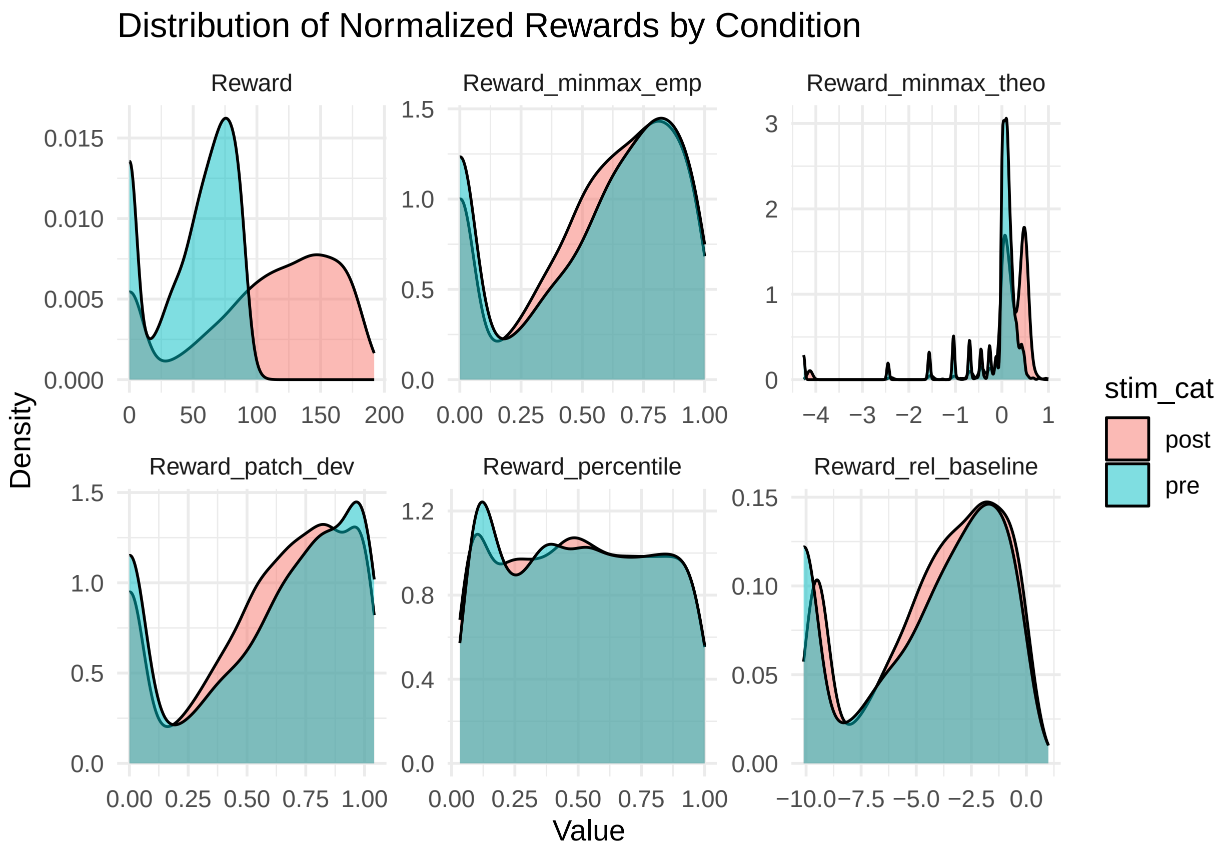
```r
# ---------------------------------------------------------------------------
# NORMALIZATION METHODS
# All within-group operations grouped by Participant_ID x stim_cat x env
# ---------------------------------------------------------------------------
df <- df %>%
  group_by(Participant_ID, stim_cat, env) %>%
  mutate(

    # -------------------------------------------------------------------
    # METHOD 1a: Min-Max (Theoretical bounds, per-trial)
    # Uses condition-specific R_max_theo and trial-depth-specific R_min_theo
    # Both pre and post get their own theoretical ceiling/floor
    # -------------------------------------------------------------------
    Reward_minmax_theo = (Reward - R_min_theo) /
                         (R_max_theo - R_min_theo + 1e-6),

    # -------------------------------------------------------------------
    # METHOD 1b: Min-Max (Empirical bounds, within Participant x stim x env)
    # Stretches observed range to [0,1] separately for env1 and env2
    # and separately for pre and post stimulation
    # -------------------------------------------------------------------
    Reward_minmax_emp = (Reward - R_min_empirical) /
                        (R_max_empirical - R_min_empirical + 1e-6),

    # -------------------------------------------------------------------
    # METHOD 2a: Percent of theoretical maximum
    # Reward as a fraction of the theoretical ceiling for that condition
    # -------------------------------------------------------------------
    Reward_pct_max_theo = Reward / R_max_theo,

    # -------------------------------------------------------------------
    # METHOD 2b: Percent of empirical maximum
    # Reward as a fraction of the highest reward observed for this
    # participant in this condition x environment
    # -------------------------------------------------------------------
    Reward_pct_max_emp = Reward / (R_max_empirical + 1e-6),

    # -------------------------------------------------------------------
    # METHOD 2c: Percent of empirical patch-level maximum
    # Reward as a fraction of the best reward in the SAME patch
    # Captures depletion relative to that specific patch's peak
    # -------------------------------------------------------------------
    Reward_pct_max_patch = Reward / (R_max_empirical_patch + 1e-6),

    # -------------------------------------------------------------------
    # METHOD 3: Within-participant x stim x env Z-score
    # Centers and scales relative to each person's own distribution
    # in each condition and environment separately
    # ------------------------------------------------------------------
    Reward_z_within = scale(Reward)[, 1],

    # ------------------------------------------------------------------
    # METHOD 4: Relative to baseline (task-anchored)
    # (Reward - baseline) / reward_increment
    # 0 = exactly at baseline; 1 = one full increment above baseline
    # ------------------------------------------------------------------
    Reward_rel_baseline = (Reward - rew_base) / (rew_hi + 1e-6),

    # ------------------------------------------------------------------
    # METHOD 5: Percentile rank within Participant x stim x env
    # r = rank(Reward) / N
    # Converts reward to its percentile position in the local distribution
    # Robust to outliers and skew; does not assume any distribution shape
    # ------------------------------------------------------------------
    Reward_percentile = rank(Reward, ties.method = "average") / n()

  ) %>%
  ungroup()
```

```r
# ----------------------------------------------------------------------
# METHOD 6: Softmax Normalization - within each PATCH
# r_i = exp(beta * Reward_i) / sum(exp(beta * Reward_j))
# beta = 0.1 (controls sensitivity; higher = more peaked toward top reward)
# Captures relative value of each trial within the local patch context
# ----------------------------------------------------------------------
beta <- 0.1

df <- df %>%
  group_by(Participant_ID, stim_cat, env, patch_id) %>%
  mutate(
    exp_reward       = exp(beta * Reward),
    Reward_softmax   = exp_reward / sum(exp_reward, na.rm = TRUE)
  ) %>%
  ungroup() %>%
  select(-exp_reward)
```

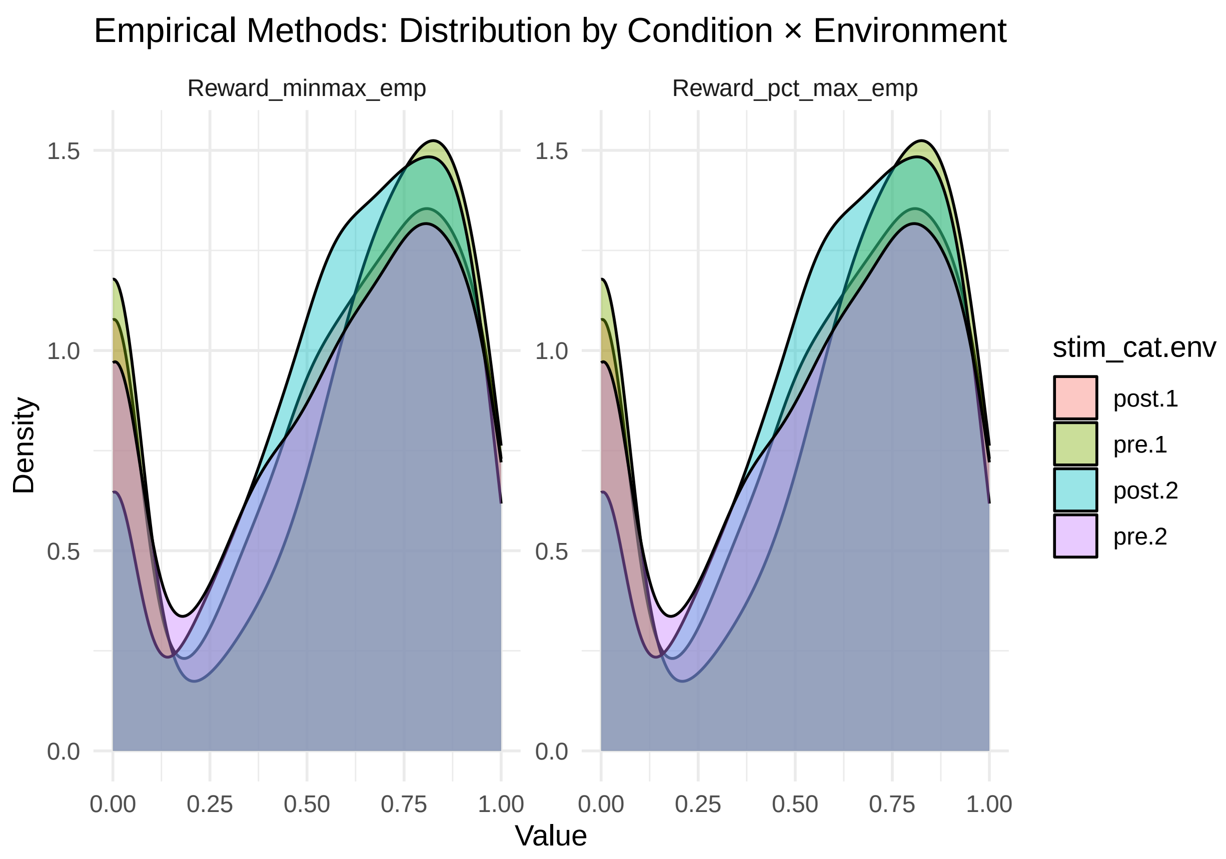
```r
# ----------------------------------------------------------------------
# METHOD 7: Deviation from Patch Start
# r = Reward / Reward_first_trial
# Measures how much reward has depleted since entering the patch
# Values < 1 indicate depletion; > 1 would indicate an unexpected increase
# ----------------------------------------------------------------------
df <- df %>%
  group_by(Participant_ID, stim_cat, env, patch_id) %>%
  mutate(
    Reward_first_trial   = first(Reward),
    Reward_patch_dev     = Reward / (Reward_first_trial + 1e-6)
  ) %>%
  ungroup()
```

```r
# ------------------------------------------------------------------------
# GLOBAL Z-SCORES for all normalized reward variables
# Puts all predictors on a common scale for model comparison
# and improves MCMC sampler efficiency
# ------------------------------------------------------------------------
df <- df %>%
  mutate(
    # Min-max
    Reward_minmax_theo_z        = scale(Reward_minmax_theo)[, 1],
    Reward_minmax_emp_z         = scale(Reward_minmax_emp)[, 1],

    # Pct of max (three variants)
    Reward_pct_max_theo_z       = scale(Reward_pct_max_theo)[, 1],
    Reward_pct_max_emp_z        = scale(Reward_pct_max_emp)[, 1],
    Reward_pct_max_patch_z      = scale(Reward_pct_max_patch)[, 1],

    # Within z-score (already centered/scaled within group, but global z for consistency)
    Reward_z_within_z           = scale(Reward_z_within)[, 1],

    # Relative to baseline
    Reward_rel_baseline_z       = scale(Reward_rel_baseline)[, 1],

    # Percentile rank
    Reward_percentile_z         = scale(Reward_percentile)[, 1],

    # Softmax
    Reward_softmax_z            = scale(Reward_softmax)[, 1],

    # Patch deviation
    Reward_patch_dev_z          = scale(Reward_patch_dev)[, 1],

    # Covariates
    trial_in_patch_z            = scale(trial_in_patch)[, 1],
    patch_id_z                  = scale(patch_id)[, 1],
    AvgRewardRate_before_z      = scale(AvgRewardRate_before)[, 1]
  )
```

```r
# ------------------------------------------------------------------------
# DISTRIBUTION CHECK: visualize all normalized reward methods
# ------------------------------------------------------------------------
p1 <- df %>%
  select(stim_cat, env,
         Reward,
         Reward_minmax_theo, Reward_minmax_emp,
         Reward_rel_baseline,
         Reward_percentile,
         Reward_patch_dev) %>%
  pivot_longer(cols = -c(stim_cat, env),
               names_to = "method", values_to = "value") %>%
  ggplot(aes(x = value, fill = stim_cat)) +
  geom_density(alpha = 0.5) +
  facet_wrap(~method, scales = "free") +
  theme_minimal() +
  labs(title = "Distribution of Normalized Rewards by Condition",
       x = "Value", y = "Density")
print(p1)
```



```
# Also facet by environment to verify env-wise separation
p1b <- df %>%
  select(stim_cat, env,
              Reward_minmax_emp, Reward_pct_max_emp) %>%
  pivot_longer(cols = -c(stim_cat, env),
                  names_to = "method", values_to = "value") %>%
  ggplot(aes(x = value, fill = interaction(stim_cat, env))) +
  geom_density(alpha = 0.4) +
  facet_wrap(~method, scales = "free") +
  theme_minimal() +
  labs(title = "Empirical Methods: Distribution by Condition × Environment",
       x = "Value", y = "Density",
       fill = "stim_cat.env")
print(p1b)
```



```r
# -----------------------------------------------------------------------
# MODEL LIST: all normalized reward predictors
# -----------------------------------------------------------------------
reward_vars <- c(
  "Reward_minmax_theo_z",
  "Reward_minmax_emp_z",
  "Reward_pct_max_theo_z",
  "Reward_pct_max_emp_z",
  "Reward_pct_max_patch_z",
  "Reward_z_within_z",
  "Reward_rel_baseline_z",
  "Reward_percentile_z",
  "Reward_softmax_z",
  "Reward_patch_dev_z"
)
```

```r
# -----------------------------------------------------------------------
# FIT BAYESIAN MIXED-EFFECTS MODELS (Student-t for heavy tails)
# Formula: HandlingTime ~ stim_cat * reward_var + covariates + (1 + stim_cat | ID)
# -----------------------------------------------------------------------
models <- list()

for (reward_var in reward_vars) {

  cat(sprintf("\nFitting model: %s\n", reward_var))
  cat("----------------------------------------\n")
  formula_str <- sprintf(
    "HandlingTime_safe ~ stim_cat * %s +
     trial_in_patch_z + patch_id_z + AvgRewardRate_before_z +
     (1 | Participant_ID)",
    reward_var
  )

  models[[reward_var]] <- brm(
    as.formula(formula_str),
    data      = df,
    family    = student(),
    chains    = 4,
    iter      = 3000,
    warmup    = 1000,
    cores     = 4,
    seed      = 123,
    save_pars = save_pars(all = TRUE),
    file      = sprintf("model_%s", reward_var),
    file_refit = "on_change"
  )
}
```

```
##
## Fitting model: Reward_minmax_theo_z
## ----------------------------------------

## Compiling Stan program...

## Trying to compile a simple C file

## Running /Library/Frameworks/R.framework/Resources/bin/R CMD SHLIB foo.c
## using C compiler: ‘Apple clang version 17.0.0 (clang-1700.6.3.2)’
## using SDK: ‘MacOSX26.2.sdk’
## clang -arch arm64 -std=gnu2x -I"/Library/Frameworks/R.framework/Resources/include" -DNDEBUG   -I"/Li
## In file included from <built-in>:1:
## In file included from /Library/Frameworks/R.framework/Versions/4.5-arm64/Resources/library/StanHeade
## In file included from /Library/Frameworks/R.framework/Versions/4.5-arm64/Resources/library/RcppEigen
## In file included from /Library/Frameworks/R.framework/Versions/4.5-arm64/Resources/library/RcppEigen
## /Library/Frameworks/R.framework/Versions/4.5-arm64/Resources/library/RcppEigen/include/Eigen/src/Cor
##   679 | #include <cmath>
##       |          ^~~~~~~
## 1 error generated.
## make: *** [foo.o] Error 1

## Start sampling

##
## Fitting model: Reward_minmax_emp_z
## ----------------------------------------
## 00 [  0%]  (Warmup)
## Chain 4:
```

```
## Chain 4: Gradient evaluation took 0.000664 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 6.64 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:    1 / 3000 [  0%]  (Warmup)
## Chain 4: Iteration:  300 / 3000 [ 10%]  (Warmup)
## Chain 3: Iteration:  300 / 3000 [ 10%]  (Warmup)
## Chain 2: Iteration:  300 / 3000 [ 10%]  (Warmup)
## Chain 1: Iteration:  300 / 3000 [ 10%]  (Warmup)
## Chain 4: Iteration:  600 / 3000 [ 20%]  (Warmup)
## Chain 3: Iteration:  600 / 3000 [ 20%]  (Warmup)
## Chain 2: Iteration:  600 / 3000 [ 20%]  (Warmup)
## Chain 1: Iteration:  600 / 3000 [ 20%]  (Warmup)
## Chain 4: Iteration:  900 / 3000 [ 30%]  (Warmup)
## Chain 3: Iteration:  900 / 3000 [ 30%]  (Warmup)
## Chain 1: Iteration:  900 / 3000 [ 30%]  (Warmup)
## Chain 2: Iteration:  900 / 3000 [ 30%]  (Warmup)
## Chain 4: Iteration: 1001 / 3000 [ 33%]  (Sampling)
## Chain 1: Iteration: 1001 / 3000 [ 33%]  (Sampling)
## Chain 3: Iteration: 1001 / 3000 [ 33%]  (Sampling)
## Chain 2: Iteration: 1001 / 3000 [ 33%]  (Sampling)
## Chain 1: Iteration: 1300 / 3000 [ 43%]  (Sampling)
## Chain 4: Iteration: 1300 / 3000 [ 43%]  (Sampling)
## Chain 2: Iteration: 1300 / 3000 [ 43%]  (Sampling)
## Chain 3: Iteration: 1300 / 3000 [ 43%]  (Sampling)
## Chain 1: Iteration: 1600 / 3000 [ 53%]  (Sampling)
## Chain 2: Iteration: 1600 / 3000 [ 53%]  (Sampling)
## Chain 4: Iteration: 1600 / 3000 [ 53%]  (Sampling)
## Chain 1: Iteration: 1900 / 3000 [ 63%]  (Sampling)
## Chain 3: Iteration: 1600 / 3000 [ 53%]  (Sampling)
## Chain 2: Iteration: 1900 / 3000 [ 63%]  (Sampling)
## Chain 1: Iteration: 2200 / 3000 [ 73%]  (Sampling)
## Chain 4: Iteration: 1900 / 3000 [ 63%]  (Sampling)
## Chain 3: Iteration: 1900 / 3000 [ 63%]  (Sampling)
## Chain 1: Iteration: 2500 / 3000 [ 83%]  (Sampling)
## Chain 2: Iteration: 2200 / 3000 [ 73%]  (Sampling)
## Chain 4: Iteration: 2200 / 3000 [ 73%]  (Sampling)
## Chain 3: Iteration: 2200 / 3000 [ 73%]  (Sampling)
## Chain 1: Iteration: 2800 / 3000 [ 93%]  (Sampling)
## Chain 2: Iteration: 2500 / 3000 [ 83%]  (Sampling)
## Chain 1: Iteration: 3000 / 3000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 25.926 seconds (Warm-up)
## Chain 1:                41.917 seconds (Sampling)
## Chain 1:                67.843 seconds (Total)
## Chain 1:
## Chain 4: Iteration: 2500 / 3000 [ 83%]  (Sampling)
## Chain 3: Iteration: 2500 / 3000 [ 83%]  (Sampling)
## Chain 2: Iteration: 2800 / 3000 [ 93%]  (Sampling)
## Chain 4: Iteration: 2800 / 3000 [ 93%]  (Sampling)
## Chain 2: Iteration: 3000 / 3000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 27.412 seconds (Warm-up)
## Chain 2:                51.482 seconds (Sampling)
## Chain 2:                78.894 seconds (Total)
## Chain 2:
## Chain 3: Iteration: 2800 / 3000 [ 93%]  (Sampling)
## Chain 4: Iteration: 3000 / 3000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 25.38 seconds (Warm-up)
## Chain 4:                58.501 seconds (Sampling)
## Chain 4:                83.881 seconds (Total)
## Chain 4:
## Chain 3: Iteration: 3000 / 3000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 26.87 seconds (Warm-up)
## Chain 3:                58.4 seconds (Sampling)
## Chain 3:                85.27 seconds (Total)
## Chain 3:
```

```
## Compiling Stan program...
```

```
## recompiling to avoid crashing R session
```

```
## Trying to compile a simple C file
```

```
## Start sampling
```

```
## :  900 / 3000 [ 30%]  (Warmup)
## Chain 2: Iteration: 1001 / 3000 [ 33%]  (Sampling)
## Chain 4: Iteration: 1001 / 3000 [ 33%]  (Sampling)
## Chain 3: Iteration: 1300 / 3000 [ 43%]  (Sampling)
## Chain 1: Iteration: 1300 / 3000 [ 43%]  (Sampling)
## Chain 2: Iteration: 1300 / 3000 [ 43%]  (Sampling)
## Chain 3: Iteration: 1600 / 3000 [ 53%]  (Sampling)
## Chain 4: Iteration: 1300 / 3000 [ 43%]  (Sampling)
## Chain 1: Iteration: 1600 / 3000 [ 53%]  (Sampling)
## Chain 2: Iteration: 1600 / 3000 [ 53%]  (Sampling)
## Chain 3: Iteration: 1900 / 3000 [ 63%]  (Sampling)
## Chain 1: Iteration: 1900 / 3000 [ 63%]  (Sampling)
## Chain 4: Iteration: 1600 / 3000 [ 53%]  (Sampling)
## Chain 2: Iteration: 1900 / 3000 [ 63%]  (Sampling)
## Chain 3: Iteration: 2200 / 3000 [ 73%]  (Sampling)
## Chain 1: Iteration: 2200 / 3000 [ 73%]  (Sampling)
## Chain 4: Iteration: 1900 / 3000 [ 63%]  (Sampling)
## Chain 1: Iteration: 2500 / 3000 [ 83%]  (Sampling)
## Chain 2: Iteration: 2200 / 3000 [ 73%]  (Sampling)
## Chain 3: Iteration: 2500 / 3000 [ 83%]  (Sampling)
## Chain 4: Iteration: 2200 / 3000 [ 73%]  (Sampling)
## Chain 1: Iteration: 2800 / 3000 [ 93%]  (Sampling)
## Chain 2: Iteration: 2500 / 3000 [ 83%]  (Sampling)
## Chain 3: Iteration: 2800 / 3000 [ 93%]  (Sampling)
## Chain 1: Iteration: 3000 / 3000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 23.262 seconds (Warm-up)
## Chain 1:                51.856 seconds (Sampling)
## Chain 1:                75.118 seconds (Total)
## Chain 1:
## Chain 3: Iteration: 3000 / 3000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 21.399 seconds (Warm-up)
## Chain 3:                55.956 seconds (Sampling)
## Chain 3:                77.355 seconds (Total)
## Chain 3:
## Chain 4: Iteration: 2500 / 3000 [ 83%]  (Sampling)
## Chain 2: Iteration: 2800 / 3000 [ 93%]  (Sampling)
## Chain 2: Iteration: 3000 / 3000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 26.567 seconds (Warm-up)
## Chain 2:                59.208 seconds (Sampling)
## Chain 2:                85.775 seconds (Total)
## Chain 2:
## Chain 4: Iteration: 2800 / 3000 [ 93%]  (Sampling)
## Chain 4: Iteration: 3000 / 3000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 28.606 seconds (Warm-up)
## Chain 4:                68.314 seconds (Sampling)
## Chain 4:                96.92 seconds (Total)
## Chain 4:

## Compiling Stan program...

## recompiling to avoid crashing R session

## Trying to compile a simple C file

## Start sampling

## ctations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:    1 / 3000 [  0%]  (Warmup)
## Chain 3: Iteration:  300 / 3000 [ 10%]  (Warmup)
## Chain 1: Iteration:  300 / 3000 [ 10%]  (Warmup)
## Chain 4: Iteration:  300 / 3000 [ 10%]  (Warmup)
## Chain 2: Iteration:  300 / 3000 [ 10%]  (Warmup)
## Chain 3: Iteration:  600 / 3000 [ 20%]  (Warmup)
## Chain 2: Iteration:  600 / 3000 [ 20%]  (Warmup)
## Chain 4: Iteration:  600 / 3000 [ 20%]  (Warmup)
## Chain 1: Iteration:  600 / 3000 [ 20%]  (Warmup)
## Chain 2: Iteration:  900 / 3000 [ 30%]  (Warmup)
## Chain 3: Iteration:  900 / 3000 [ 30%]  (Warmup)
## Chain 1: Iteration:  900 / 3000 [ 30%]  (Warmup)
## Chain 4: Iteration:  900 / 3000 [ 30%]  (Warmup)
## Chain 2: Iteration: 1001 / 3000 [ 33%]  (Sampling)
## Chain 3: Iteration: 1001 / 3000 [ 33%]  (Sampling)
## Chain 4: Iteration: 1001 / 3000 [ 33%]  (Sampling)
## Chain 1: Iteration: 1001 / 3000 [ 33%]  (Sampling)
## Chain 2: Iteration: 1300 / 3000 [ 43%]  (Sampling)
## Chain 4: Iteration: 1300 / 3000 [ 43%]  (Sampling)
## Chain 3: Iteration: 1300 / 3000 [ 43%]  (Sampling)
## Chain 1: Iteration: 1300 / 3000 [ 43%]  (Sampling)
## Chain 2: Iteration: 1600 / 3000 [ 53%]  (Sampling)
## Chain 4: Iteration: 1600 / 3000 [ 53%]  (Sampling)
## Chain 3: Iteration: 1600 / 3000 [ 53%]  (Sampling)
## Chain 2: Iteration: 1900 / 3000 [ 63%]  (Sampling)
## Chain 4: Iteration: 1900 / 3000 [ 63%]  (Sampling)
## Chain 1: Iteration: 1600 / 3000 [ 53%]  (Sampling)
## Chain 3: Iteration: 1900 / 3000 [ 63%]  (Sampling)
## Chain 4: Iteration: 2200 / 3000 [ 73%]  (Sampling)
## Chain 2: Iteration: 2200 / 3000 [ 73%]  (Sampling)
## Chain 1: Iteration: 1900 / 3000 [ 63%]  (Sampling)
## Chain 3: Iteration: 2200 / 3000 [ 73%]  (Sampling)
## Chain 4: Iteration: 2500 / 3000 [ 83%]  (Sampling)
## Chain 2: Iteration: 2500 / 3000 [ 83%]  (Sampling)
## Chain 3: Iteration: 2500 / 3000 [ 83%]  (Sampling)
## Chain 4: Iteration: 2800 / 3000 [ 93%]  (Sampling)
## Chain 1: Iteration: 2200 / 3000 [ 73%]  (Sampling)
## Chain 2: Iteration: 2800 / 3000 [ 93%]  (Sampling)
## Chain 4: Iteration: 3000 / 3000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 24.628 seconds (Warm-up)
## Chain 4:                47.514 seconds (Sampling)
## Chain 4:                72.142 seconds (Total)
## Chain 4:
## Chain 2: Iteration: 3000 / 3000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 22.567 seconds (Warm-up)
## Chain 2:                51.739 seconds (Sampling)
## Chain 2:                74.306 seconds (Total)
## Chain 2:
## Chain 3: Iteration: 2800 / 3000 [ 93%]  (Sampling)
## Chain 1: Iteration: 2500 / 3000 [ 83%]  (Sampling)
## Chain 3: Iteration: 3000 / 3000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 22.91 seconds (Warm-up)
## Chain 3:                57.788 seconds (Sampling)
## Chain 3:                80.698 seconds (Total)
## Chain 3:
## Chain 1: Iteration: 2800 / 3000 [ 93%]  (Sampling)
## Chain 1: Iteration: 3000 / 3000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 24.673 seconds (Warm-up)
## Chain 1:                73.298 seconds (Sampling)
## Chain 1:                97.971 seconds (Total)
## Chain 1:
```

```
## Compiling Stan program...
```

```
## recompiling to avoid crashing R session
```

```
## Trying to compile a simple C file
## Start sampling

## teration: 2200 / 3000 [ 73%]  (Sampling)
## Chain 4: Iteration: 1900 / 3000 [ 63%]  (Sampling)
## Chain 2: Iteration: 2200 / 3000 [ 73%]  (Sampling)
## Chain 1: Iteration: 2500 / 3000 [ 83%]  (Sampling)
## Chain 3: Iteration: 2500 / 3000 [ 83%]  (Sampling)
## Chain 4: Iteration: 2200 / 3000 [ 73%]  (Sampling)
## Chain 1: Iteration: 2800 / 3000 [ 93%]  (Sampling)
## Chain 2: Iteration: 2500 / 3000 [ 83%]  (Sampling)
## Chain 3: Iteration: 2800 / 3000 [ 93%]  (Sampling)
## Chain 1: Iteration: 3000 / 3000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 23.503 seconds (Warm-up)
## Chain 1:                52.275 seconds (Sampling)
## Chain 1:                75.778 seconds (Total)
## Chain 1:
## Chain 3: Iteration: 3000 / 3000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 21.656 seconds (Warm-up)
## Chain 3:                56.472 seconds (Sampling)
## Chain 3:                78.128 seconds (Total)
## Chain 3:
## Chain 2: Iteration: 2800 / 3000 [ 93%]  (Sampling)
## Chain 4: Iteration: 2500 / 3000 [ 83%]  (Sampling)
## Chain 2: Iteration: 3000 / 3000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 26.28 seconds (Warm-up)
## Chain 2:                59.365 seconds (Sampling)
## Chain 2:                85.645 seconds (Total)
## Chain 2:
## Chain 4: Iteration: 2800 / 3000 [ 93%]  (Sampling)
## Chain 4: Iteration: 3000 / 3000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 28.936 seconds (Warm-up)
## Chain 4:                68.589 seconds (Sampling)
## Chain 4:                97.525 seconds (Total)
## Chain 4:

## Compiling Stan program...

## recompiling to avoid crashing R session

## Trying to compile a simple C file

## Start sampling

##  1001 / 3000 [ 33%]  (Sampling)
## Chain 1: Iteration:  900 / 3000 [ 30%]  (Warmup)
## Chain 4: Iteration: 1300 / 3000 [ 43%]  (Sampling)
## Chain 1: Iteration: 1001 / 3000 [ 33%]  (Sampling)
## Chain 2: Iteration: 1300 / 3000 [ 43%]  (Sampling)
## Chain 4: Iteration: 1600 / 3000 [ 53%]  (Sampling)
## Chain 3: Iteration: 1300 / 3000 [ 43%]  (Sampling)
## Chain 1: Iteration: 1300 / 3000 [ 43%]  (Sampling)
## Chain 4: Iteration: 1900 / 3000 [ 63%]  (Sampling)
## Chain 2: Iteration: 1600 / 3000 [ 53%]  (Sampling)
## Chain 3: Iteration: 1600 / 3000 [ 53%]  (Sampling)
## Chain 1: Iteration: 1600 / 3000 [ 53%]  (Sampling)
## Chain 4: Iteration: 2200 / 3000 [ 73%]  (Sampling)
## Chain 2: Iteration: 1900 / 3000 [ 63%]  (Sampling)
## Chain 3: Iteration: 1900 / 3000 [ 63%]  (Sampling)
## Chain 1: Iteration: 1900 / 3000 [ 63%]  (Sampling)
## Chain 4: Iteration: 2500 / 3000 [ 83%]  (Sampling)
## Chain 2: Iteration: 2200 / 3000 [ 73%]  (Sampling)
## Chain 4: Iteration: 2800 / 3000 [ 93%]  (Sampling)
## Chain 1: Iteration: 2200 / 3000 [ 73%]  (Sampling)
## Chain 3: Iteration: 2200 / 3000 [ 73%]  (Sampling)
## Chain 2: Iteration: 2500 / 3000 [ 83%]  (Sampling)
## Chain 4: Iteration: 3000 / 3000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 20.051 seconds (Warm-up)
## Chain 4:                49.154 seconds (Sampling)
## Chain 4:                69.205 seconds (Total)
## Chain 4:
## Chain 2: Iteration: 2800 / 3000 [ 93%]  (Sampling)
## Chain 1: Iteration: 2500 / 3000 [ 83%]  (Sampling)
## Chain 3: Iteration: 2500 / 3000 [ 83%]  (Sampling)
## Chain 2: Iteration: 3000 / 3000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 26.88 seconds (Warm-up)
## Chain 2:                51.859 seconds (Sampling)
## Chain 2:                78.739 seconds (Total)
## Chain 2:
## Chain 1: Iteration: 2800 / 3000 [ 93%]  (Sampling)
## Chain 3: Iteration: 2800 / 3000 [ 93%]  (Sampling)
## Chain 1: Iteration: 3000 / 3000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 29.705 seconds (Warm-up)
## Chain 1:                58.516 seconds (Sampling)
## Chain 1:                88.221 seconds (Total)
## Chain 1:
## Chain 3: Iteration: 3000 / 3000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 25.701 seconds (Warm-up)
## Chain 3:                66.374 seconds (Sampling)
## Chain 3:                92.075 seconds (Total)
## Chain 3:

## Warning: Rows containing NAs were excluded from the model.

## Compiling Stan program...

## recompiling to avoid crashing R session

## Trying to compile a simple C file
## Start sampling

##  3000 [ 33%]  (Sampling)
## Chain 4: Iteration: 1001 / 3000 [ 33%]  (Sampling)
## Chain 1: Iteration: 1001 / 3000 [ 33%]  (Sampling)
## Chain 4: Iteration: 1300 / 3000 [ 43%]  (Sampling)
## Chain 3: Iteration: 1300 / 3000 [ 43%]  (Sampling)
## Chain 1: Iteration: 1300 / 3000 [ 43%]  (Sampling)
## Chain 2: Iteration: 1300 / 3000 [ 43%]  (Sampling)
## Chain 4: Iteration: 1600 / 3000 [ 53%]  (Sampling)
## Chain 3: Iteration: 1600 / 3000 [ 53%]  (Sampling)
## Chain 1: Iteration: 1600 / 3000 [ 53%]  (Sampling)
## Chain 2: Iteration: 1600 / 3000 [ 53%]  (Sampling)
## Chain 4: Iteration: 1900 / 3000 [ 63%]  (Sampling)
## Chain 1: Iteration: 1900 / 3000 [ 63%]  (Sampling)
## Chain 3: Iteration: 1900 / 3000 [ 63%]  (Sampling)
## Chain 4: Iteration: 2200 / 3000 [ 73%]  (Sampling)
## Chain 2: Iteration: 1900 / 3000 [ 63%]  (Sampling)
## Chain 1: Iteration: 2200 / 3000 [ 73%]  (Sampling)
## Chain 3: Iteration: 2200 / 3000 [ 73%]  (Sampling)
## Chain 4: Iteration: 2500 / 3000 [ 83%]  (Sampling)
## Chain 2: Iteration: 2200 / 3000 [ 73%]  (Sampling)
## Chain 1: Iteration: 2500 / 3000 [ 83%]  (Sampling)
## Chain 4: Iteration: 2800 / 3000 [ 93%]  (Sampling)
## Chain 3: Iteration: 2500 / 3000 [ 83%]  (Sampling)
## Chain 4: Iteration: 3000 / 3000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 27.562 seconds (Warm-up)
## Chain 4:                48.708 seconds (Sampling)
## Chain 4:                76.27 seconds (Total)
## Chain 4:
## Chain 1: Iteration: 2800 / 3000 [ 93%]  (Sampling)
## Chain 2: Iteration: 2500 / 3000 [ 83%]  (Sampling)
## Chain 3: Iteration: 2800 / 3000 [ 93%]  (Sampling)
## Chain 1: Iteration: 3000 / 3000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 28.124 seconds (Warm-up)
## Chain 1:                55 seconds (Sampling)
## Chain 1:                83.124 seconds (Total)
## Chain 1:
## Chain 3: Iteration: 3000 / 3000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 26.349 seconds (Warm-up)
## Chain 3:                59.418 seconds (Sampling)
## Chain 3:                85.767 seconds (Total)
## Chain 3:
## Chain 2: Iteration: 2800 / 3000 [ 93%]  (Sampling)
## Chain 2: Iteration: 3000 / 3000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 26.214 seconds (Warm-up)
## Chain 2:                71.047 seconds (Sampling)
## Chain 2:                97.261 seconds (Total)
## Chain 2:
```

```
## Compiling Stan program...

## recompiling to avoid crashing R session

## Trying to compile a simple C file

## Start sampling

## on:  900 / 3000 [ 30%]  (Warmup)
## Chain 2: Iteration: 1001 / 3000 [ 33%]  (Sampling)
## Chain 3: Iteration: 1001 / 3000 [ 33%]  (Sampling)
## Chain 2: Iteration: 1300 / 3000 [ 43%]  (Sampling)
## Chain 4: Iteration: 1300 / 3000 [ 43%]  (Sampling)
## Chain 1: Iteration: 1300 / 3000 [ 43%]  (Sampling)
## Chain 3: Iteration: 1300 / 3000 [ 43%]  (Sampling)
## Chain 2: Iteration: 1600 / 3000 [ 53%]  (Sampling)
## Chain 4: Iteration: 1600 / 3000 [ 53%]  (Sampling)
## Chain 1: Iteration: 1600 / 3000 [ 53%]  (Sampling)
## Chain 3: Iteration: 1600 / 3000 [ 53%]  (Sampling)
## Chain 2: Iteration: 1900 / 3000 [ 63%]  (Sampling)
## Chain 4: Iteration: 1900 / 3000 [ 63%]  (Sampling)
## Chain 3: Iteration: 1900 / 3000 [ 63%]  (Sampling)
## Chain 1: Iteration: 1900 / 3000 [ 63%]  (Sampling)
## Chain 2: Iteration: 2200 / 3000 [ 73%]  (Sampling)
## Chain 4: Iteration: 2200 / 3000 [ 73%]  (Sampling)
## Chain 3: Iteration: 2200 / 3000 [ 73%]  (Sampling)
## Chain 2: Iteration: 2500 / 3000 [ 83%]  (Sampling)
## Chain 1: Iteration: 2200 / 3000 [ 73%]  (Sampling)
## Chain 4: Iteration: 2500 / 3000 [ 83%]  (Sampling)
## Chain 2: Iteration: 2800 / 3000 [ 93%]  (Sampling)
## Chain 3: Iteration: 2500 / 3000 [ 83%]  (Sampling)
## Chain 2: Iteration: 3000 / 3000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 25.566 seconds (Warm-up)
## Chain 2:                50.924 seconds (Sampling)
## Chain 2:                76.49 seconds (Total)
## Chain 2:
## Chain 1: Iteration: 2500 / 3000 [ 83%]  (Sampling)
## Chain 4: Iteration: 2800 / 3000 [ 93%]  (Sampling)
## Chain 3: Iteration: 2800 / 3000 [ 93%]  (Sampling)
## Chain 4: Iteration: 3000 / 3000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 25.247 seconds (Warm-up)
## Chain 4:                58.818 seconds (Sampling)
## Chain 4:                84.065 seconds (Total)
## Chain 4:
## Chain 3: Iteration: 3000 / 3000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 28.462 seconds (Warm-up)
## Chain 3:                58.358 seconds (Sampling)
## Chain 3:                86.82 seconds (Total)
## Chain 3:
## Chain 1: Iteration: 2800 / 3000 [ 93%]  (Sampling)
## Chain 1: Iteration: 3000 / 3000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 23.43 seconds (Warm-up)
## Chain 1:                70.143 seconds (Sampling)
## Chain 1:                93.573 seconds (Total)
## Chain 1:
```

```
## Compiling Stan program...
```

```
## recompiling to avoid crashing R session
```

```
## Trying to compile a simple C file
```

```
## Start sampling
```

```
## 1001 / 3000 [ 33%]  (Sampling)
## Chain 1: Iteration:  900 / 3000 [ 30%]  (Warmup)
## Chain 1: Iteration: 1001 / 3000 [ 33%]  (Sampling)
## Chain 2: Iteration: 1300 / 3000 [ 43%]  (Sampling)
## Chain 4: Iteration: 1300 / 3000 [ 43%]  (Sampling)
## Chain 3: Iteration: 1300 / 3000 [ 43%]  (Sampling)
## Chain 1: Iteration: 1300 / 3000 [ 43%]  (Sampling)
## Chain 3: Iteration: 1600 / 3000 [ 53%]  (Sampling)
## Chain 4: Iteration: 1600 / 3000 [ 53%]  (Sampling)
## Chain 2: Iteration: 1600 / 3000 [ 53%]  (Sampling)
## Chain 1: Iteration: 1600 / 3000 [ 53%]  (Sampling)
## Chain 3: Iteration: 1900 / 3000 [ 63%]  (Sampling)
## Chain 4: Iteration: 1900 / 3000 [ 63%]  (Sampling)
## Chain 2: Iteration: 1900 / 3000 [ 63%]  (Sampling)
## Chain 3: Iteration: 2200 / 3000 [ 73%]  (Sampling)
## Chain 4: Iteration: 2200 / 3000 [ 73%]  (Sampling)
## Chain 1: Iteration: 1900 / 3000 [ 63%]  (Sampling)
## Chain 2: Iteration: 2200 / 3000 [ 73%]  (Sampling)
## Chain 3: Iteration: 2500 / 3000 [ 83%]  (Sampling)
## Chain 4: Iteration: 2500 / 3000 [ 83%]  (Sampling)
## Chain 1: Iteration: 2200 / 3000 [ 73%]  (Sampling)
## Chain 3: Iteration: 2800 / 3000 [ 93%]  (Sampling)
## Chain 2: Iteration: 2500 / 3000 [ 83%]  (Sampling)
## Chain 4: Iteration: 2800 / 3000 [ 93%]  (Sampling)
## Chain 1: Iteration: 2500 / 3000 [ 83%]  (Sampling)
## Chain 3: Iteration: 3000 / 3000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 27.036 seconds (Warm-up)
## Chain 3:                45.948 seconds (Sampling)
## Chain 3:                72.984 seconds (Total)
## Chain 3:
## Chain 4: Iteration: 3000 / 3000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 26.997 seconds (Warm-up)
## Chain 4:                47.722 seconds (Sampling)
## Chain 4:                74.719 seconds (Total)
## Chain 4:
## Chain 2: Iteration: 2800 / 3000 [ 93%]  (Sampling)
## Chain 1: Iteration: 2800 / 3000 [ 93%]  (Sampling)
## Chain 2: Iteration: 3000 / 3000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 22.316 seconds (Warm-up)
## Chain 2:                62.09 seconds (Sampling)
## Chain 2:                84.406 seconds (Total)
## Chain 2:
## Chain 1: Iteration: 3000 / 3000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 29.748 seconds (Warm-up)
## Chain 1:                57.293 seconds (Sampling)
## Chain 1:                87.041 seconds (Total)
## Chain 1:
```

```
## Compiling Stan program...
```

```
## recompiling to avoid crashing R session
```

```
## Trying to compile a simple C file
```

```
## Start sampling
```

```
## 3000 [ 33%]  (Sampling)
## Chain 3: Iteration: 1001 / 3000 [ 33%]  (Sampling)
## Chain 1: Iteration: 1001 / 3000 [ 33%]  (Sampling)
## Chain 2: Iteration: 1300 / 3000 [ 43%]  (Sampling)
## Chain 4: Iteration: 1300 / 3000 [ 43%]  (Sampling)
## Chain 1: Iteration: 1300 / 3000 [ 43%]  (Sampling)
## Chain 3: Iteration: 1300 / 3000 [ 43%]  (Sampling)
## Chain 4: Iteration: 1600 / 3000 [ 53%]  (Sampling)
## Chain 2: Iteration: 1600 / 3000 [ 53%]  (Sampling)
## Chain 1: Iteration: 1600 / 3000 [ 53%]  (Sampling)
## Chain 3: Iteration: 1600 / 3000 [ 53%]  (Sampling)
## Chain 1: Iteration: 1900 / 3000 [ 63%]  (Sampling)
## Chain 4: Iteration: 1900 / 3000 [ 63%]  (Sampling)
## Chain 2: Iteration: 1900 / 3000 [ 63%]  (Sampling)
## Chain 3: Iteration: 1900 / 3000 [ 63%]  (Sampling)
## Chain 1: Iteration: 2200 / 3000 [ 73%]  (Sampling)
## Chain 4: Iteration: 2200 / 3000 [ 73%]  (Sampling)
## Chain 2: Iteration: 2200 / 3000 [ 73%]  (Sampling)
## Chain 3: Iteration: 2200 / 3000 [ 73%]  (Sampling)
## Chain 1: Iteration: 2500 / 3000 [ 83%]  (Sampling)
## Chain 4: Iteration: 2500 / 3000 [ 83%]  (Sampling)
## Chain 2: Iteration: 2500 / 3000 [ 83%]  (Sampling)
## Chain 3: Iteration: 2500 / 3000 [ 83%]  (Sampling)
## Chain 1: Iteration: 2800 / 3000 [ 93%]  (Sampling)
## Chain 4: Iteration: 2800 / 3000 [ 93%]  (Sampling)
## Chain 1: Iteration: 3000 / 3000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 27.311 seconds (Warm-up)
## Chain 1:                50.411 seconds (Sampling)
## Chain 1:                77.722 seconds (Total)
## Chain 1:
```

```
## Chain 2: Iteration: 2800 / 3000 [ 93%]  (Sampling)
## Chain 3: Iteration: 2800 / 3000 [ 93%]  (Sampling)
## Chain 4: Iteration: 3000 / 3000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 24.983 seconds (Warm-up)
## Chain 4:                57.2 seconds (Sampling)
## Chain 4:                82.183 seconds (Total)
## Chain 4:
## Chain 2: Iteration: 3000 / 3000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 23.898 seconds (Warm-up)
## Chain 2:                61.184 seconds (Sampling)
## Chain 2:                85.082 seconds (Total)
## Chain 2:
## Chain 3: Iteration: 3000 / 3000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 26.144 seconds (Warm-up)
## Chain 3:                61.639 seconds (Sampling)
## Chain 3:                87.783 seconds (Total)
## Chain 3:
```

```
## Compiling Stan program...
```

```
## recompiling to avoid crashing R session
```

```
## Trying to compile a simple C file
```

```
## Start sampling
```

```r
models$Reward_z_within_z <- NULL
loo_list <- lapply(models, loo)
loo_results <- loo_compare(loo_list)
print(loo_results)
```

```
##                         elpd_diff se_diff
## Reward_pct_max_theo_z    0.0       0.0
## Reward_rel_baseline_z   -0.1       0.1
## Reward_minmax_emp_z     -1.5       0.7
## Reward_pct_max_emp_z    -1.5       0.7
## Reward_pct_max_patch_z  -2.4       1.0
## Reward_patch_dev_z      -2.7       1.0
## Reward_minmax_theo_z    -3.1       4.6
## Reward_percentile_z     -8.1       3.3
## Reward_softmax_z       -12.9       7.2
```

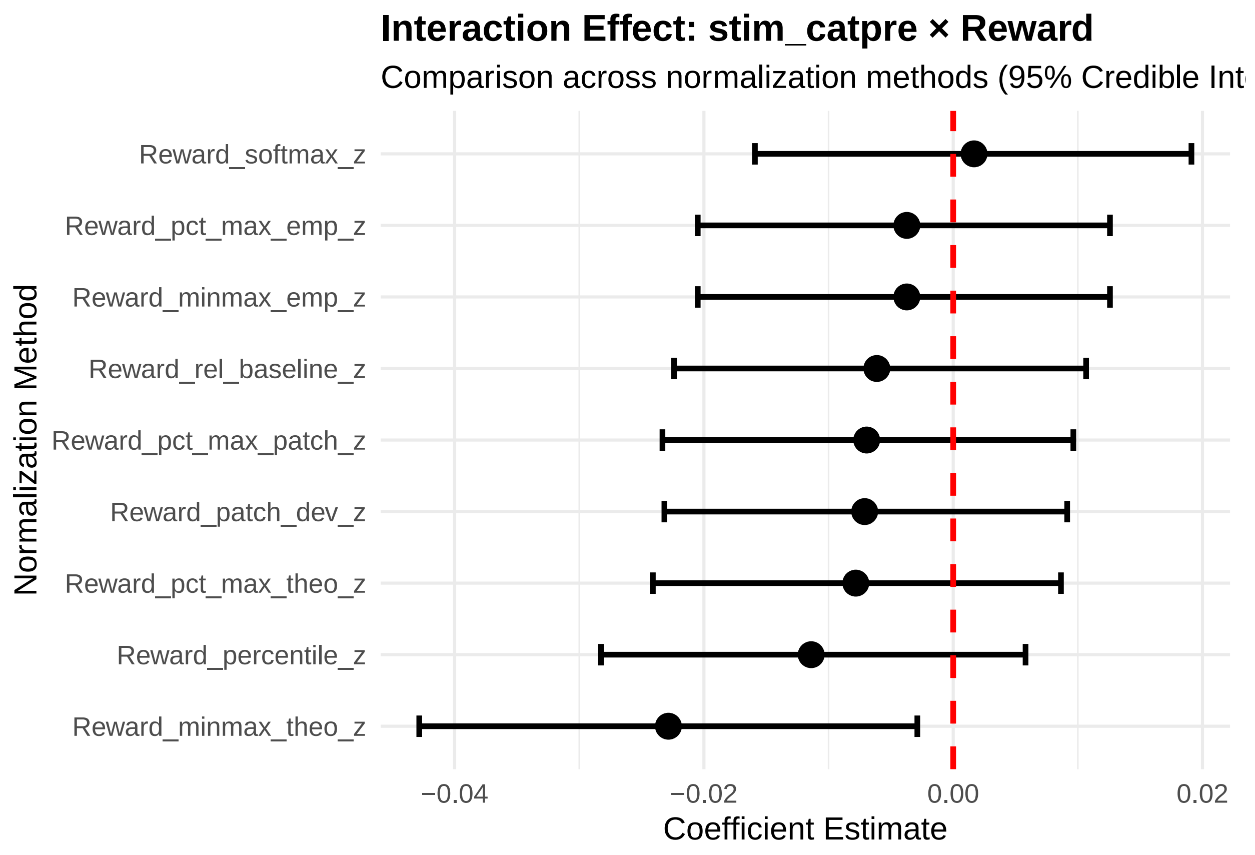
```r
r2_results <- map_df(names(models), function(nm) {
  r2 <- bayes_R2(models[[nm]])
  data.frame(
    Normalization = nm,
    R2_mean  = r2[1, "Estimate"],
    R2_lower = r2[1, "Q2.5"],
    R2_upper = r2[1, "Q97.5"]
  )
}) %>% arrange(desc(R2_mean))

print(r2_results)
```

```
##            Normalization   R2_mean  R2_lower  R2_upper
## 1   Reward_minmax_theo_z 0.3739037 0.3575443 0.3899699
## 2       Reward_softmax_z 0.3728149 0.3566664 0.3890301
## 3    Reward_percentile_z 0.3715942 0.3555220 0.3873790
## 4  Reward_rel_baseline_z 0.3713855 0.3553822 0.3875249
## 5  Reward_pct_max_theo_z 0.3713325 0.3549229 0.3870100
## 6    Reward_minmax_emp_z 0.3711413 0.3553039 0.3868881
## 7   Reward_pct_max_emp_z 0.3711413 0.3553039 0.3868881
## 8 Reward_pct_max_patch_z 0.3710543 0.3549332 0.3869816
## 9     Reward_patch_dev_z 0.3709998 0.3547009 0.3868618
```

```r
# -----------------------------------------------------------------------
# EXTRACT INTERACTION TERM: stim_catpre × reward_var
# -----------------------------------------------------------------------
extract_interaction <- function(model, name) {
  sum_df <- as.data.frame(summary(model)$fixed)
  interaction_row <- grep("stim_catpre:", rownames(sum_df), value = TRUE)[1]

  if (length(interaction_row) > 0 && !is.na(interaction_row)) {
    data.frame(
      Normalization = name,
      Estimate      = sum_df[interaction_row, "Estimate"],
      SE            = sum_df[interaction_row, "Est.Error"],
      CI_lower      = sum_df[interaction_row, "l-95% CI"],
      CI_upper      = sum_df[interaction_row, "u-95% CI"],
      Rhat          = sum_df[interaction_row, "Rhat"],
      Bulk_ESS      = sum_df[interaction_row, "Bulk_ESS"],
      row.names = NULL
    )
  }
}

interaction_df <- map2_df(models, names(models), extract_interaction)
interaction_df <- interaction_df %>% arrange(Estimate)
print(interaction_df)
```

```
##            Normalization     Estimate          SE    CI_lower     CI_upper
## 1   Reward_minmax_theo_z -0.022846908 0.010234814 -0.04283733 -0.002878037
## 2    Reward_percentile_z -0.011393260 0.008641431 -0.02826189  0.005805620
## 3  Reward_pct_max_theo_z -0.007815711 0.008345735 -0.02409744  0.008645744
## 4     Reward_patch_dev_z -0.007103914 0.008369624 -0.02317386  0.009144844
## 5 Reward_pct_max_patch_z -0.006942772 0.008433055 -0.02332682  0.009637701
## 6  Reward_rel_baseline_z -0.006127318 0.008595539 -0.02239367  0.010663998
## 7    Reward_minmax_emp_z -0.003721222 0.008497421 -0.02049451  0.012576890
## 8   Reward_pct_max_emp_z -0.003721222 0.008497421 -0.02049451  0.012576890
## 9       Reward_softmax_z  0.001671771 0.008977244 -0.01591141  0.019113575
##       Rhat Bulk_ESS
## 1 1.000699 5793.101
## 2 1.000149 5162.363
## 3 1.000166 6308.871
## 4 1.000480 5966.548
## 5 1.002124 5488.977
## 6 1.001345 5449.155
## 7 1.001384 5500.618
## 8 1.001384 5500.618
## 9 1.000347 6122.905
```

```
# ----------------------------------------------------------------------
# PLOT: Interaction effects across normalization methods
# ----------------------------------------------------------------------
p2 <- ggplot(interaction_df,
             aes(x = reorder(Normalization, Estimate), y = Estimate)) +
  geom_point(size = 4) +
  geom_errorbar(aes(ymin = CI_lower, ymax = CI_upper),
                width = 0.3, linewidth = 1) +
  geom_hline(yintercept = 0, linetype = "dashed",
             color = "red", linewidth = 1) +
  coord_flip() +
  theme_minimal(base_size = 12) +
  labs(
    title    = "Interaction Effect: stim_catpre × Reward",
    subtitle = "Comparison across normalization methods (95% Credible Intervals)",
    x        = "Normalization Method",
    y        = "Coefficient Estimate"
  ) +
  theme(plot.title = element_text(face = "bold", size = 14))
print(p2)
```



```r
# ----------------------------------------------------------------------
# EXTRACT MAIN EFFECT OF stim_catpre across all models
# ----------------------------------------------------------------------
extract_stim_main <- function(model, name) {
  sum_df   <- as.data.frame(summary(model)$fixed)
  stim_row <- grep("^stim_catpre$", rownames(sum_df), value = TRUE)

  if (length(stim_row) > 0) {
    data.frame(
      Normalization = name,
      Estimate      = sum_df[stim_row, "Estimate"],
      SE            = sum_df[stim_row, "Est.Error"],
      CI_lower      = sum_df[stim_row, "l-95% CI"],
      CI_upper      = sum_df[stim_row, "u-95% CI"],
      Rhat          = sum_df[stim_row, "Rhat"],
      Bulk_ESS      = sum_df[stim_row, "Bulk_ESS"],
      row.names = NULL
    )
  }
}

stim_df <- purrr::map2_df(models, names(models), extract_stim_main)
stim_df <- stim_df %>% arrange(Estimate)
print(stim_df)
```

```
##              Normalization  Estimate          SE    CI_lower  CI_upper      Rhat
## 1   Reward_minmax_theo_z 0.1117688 0.01883633 0.07466459 0.1489680 1.000366
## 2  Reward_rel_baseline_z 0.1130692 0.01808360 0.07795368 0.1485880 1.000483
## 3    Reward_percentile_z 0.1144288 0.01876256 0.07781968 0.1507257 1.000665
## 4    Reward_minmax_emp_z 0.1153919 0.01816868 0.08024778 0.1506422 1.000858
## 5   Reward_pct_max_emp_z 0.1153919 0.01816868 0.08024778 0.1506422 1.000858
## 6  Reward_pct_max_theo_z 0.1159788 0.01822981 0.08023913 0.1513698 1.000319
## 7     Reward_patch_dev_z 0.1181448 0.01845374 0.08101429 0.1534005 1.001298
## 8 Reward_pct_max_patch_z 0.1183982 0.01845643 0.08136555 0.1546772 1.001806
## 9       Reward_softmax_z 0.1192356 0.01892901 0.08244810 0.1565743 1.001407
##   Bulk_ESS
## 1 4677.597
## 2 5120.391
## 3 4824.920
## 4 5603.266
## 5 5603.266
## 6 5464.692
## 7 5169.954
## 8 5107.994
## 9 5372.917
```