Stats 551 Final Project

Alyssa Yang

```
library(ggplot2)
library(dplyr)
```

```
Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union
```

```
library(tidyr)
library(rstan)
```

```
Loading required package: StanHeaders


rstan version 2.32.6 (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
```

```
library(bayesplot)
```

```
This is bayesplot version 1.11.1

- 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
```

```
library(gridExtra)
```

```
Attaching package: 'gridExtra'

The following object is masked from 'package:dplyr':

    combine
```

```
library(rstanarm)
```

```
Loading required package: Rcpp

This is rstanarm version 2.32.1

- See https://mc-stan.org/rstanarm/articles/priors for changes to default priors!

- Default priors may change, so it's safest to specify priors, even if equivalent to the defa
```

```
- For execution on a local, multicore CPU with excess RAM we recommend calling

  options(mc.cores = parallel::detectCores())


Attaching package: 'rstanarm'

The following object is masked from 'package:rstan':

    loo
```

EDA

```
# Read in plant data
plant <- read.csv("plant_moniter_health_data.csv")

# Rename columns
colnames(plant) <- c("plant_id", "temp", "humidity", "soil_moisture", "soil_ph",
                     "nutrient_level", "light_intensity", "health_score", "health_status")
```

```
# Find types of variables
sapply(plant, typeof)
```

```
       plant_id            temp        humidity   soil_moisture         soil_ph
    "character"        "double"        "double"        "double"        "double"
 nutrient_level light_intensity    health_score   health_status
       "double"        "double"        "double"       "integer"
```

```
# Change health_status to categorical variable
plant$health_status <- as.character(plant$health_status)
```

```
# Find if there are any missing values
sum(is.na(plant))
```

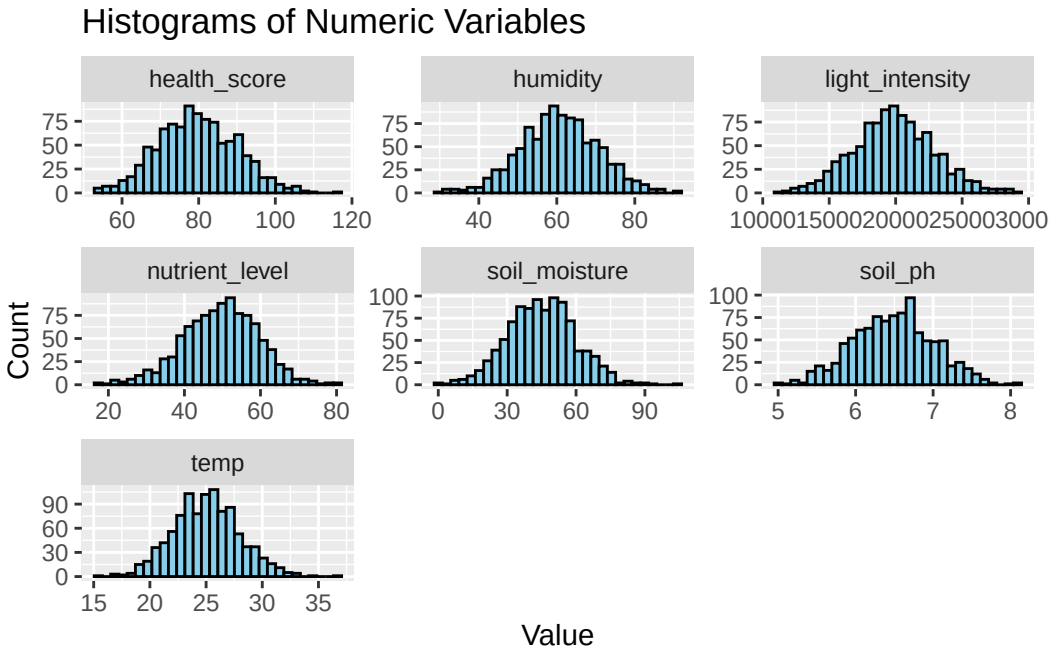
```
[1] 0
```

Univariate summaries

```r
# Find summaries of numeric variables
summary(plant[sapply(plant, is.numeric)])
```

```
      temp          humidity     soil_moisture         soil_ph
 Min.   :15.28   Min.   :30.60   Min.   : -0.2927   Min.   :5.035
 1st Qu.:23.06   1st Qu.:53.94   1st Qu.: 35.2800   1st Qu.:6.131
 Median :25.08   Median :60.63   Median : 44.9962   Median :6.500
 Mean   :25.06   Mean   :60.71   Mean   : 45.0875   Mean   :6.491
 3rd Qu.:26.94   3rd Qu.:67.29   3rd Qu.: 54.9137   3rd Qu.:6.833
 Max.   :36.56   Max.   :91.93   Max.   :103.8936   Max.   :8.122
 nutrient_level  light_intensity  health_score
 Min.   :18.23   Min.   :11301   Min.   : 52.87
 1st Qu.:43.17   1st Qu.:17919   1st Qu.: 72.45
 Median :49.82   Median :19872   Median : 79.45
 Mean   :49.51   Mean   :19860   Mean   : 79.72
 3rd Qu.:56.39   3rd Qu.:21837   3rd Qu.: 87.00
 Max.   :81.13   Max.   :29295   Max.   :115.29
```

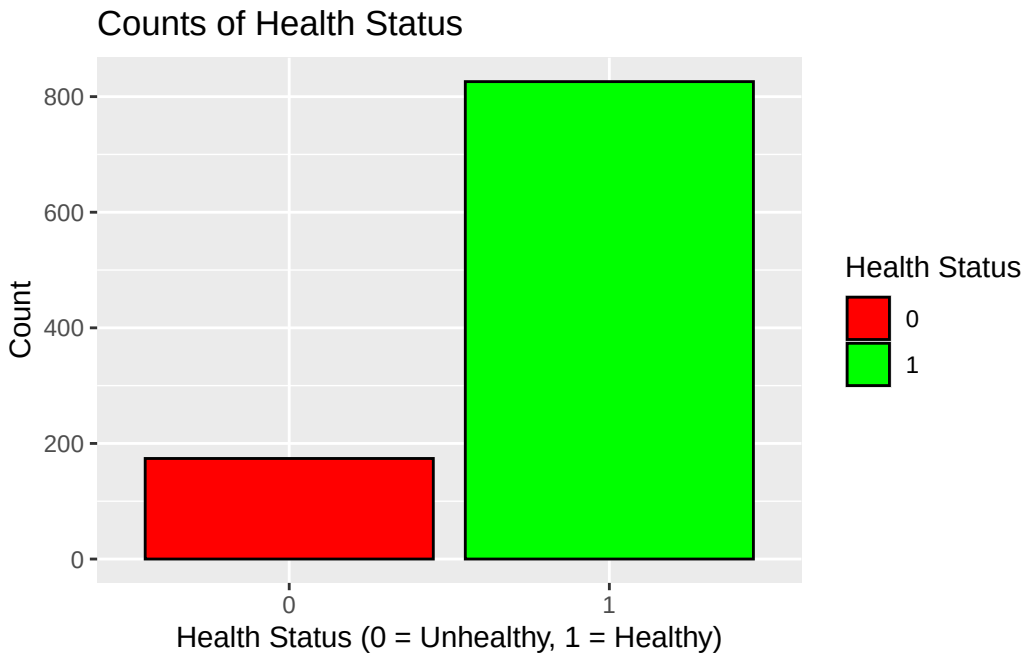
```r
# Select only numeric variables
numeric_vars <- plant %>%
  select(-plant_id, -health_status) %>%
  pivot_longer(cols = everything(), names_to = "variable", values_to = "value")

# Create histograms of numeric variables
numeric_plots <- ggplot(numeric_vars, aes(x = value)) +
  geom_histogram(bins = 30, color = "black", fill = "skyblue") +
  facet_wrap(~ variable, scales = "free", ncol = 3) +
  labs(title = "Histograms of Numeric Variables",
       x = "Value",
       y = "Count")

ggsave("numeric_variables.png", plot = numeric_plots, width = 8, height = 6, dpi = 300)
numeric_plots
```




```
# Find counts for health status
table(plant$health_status)
```

```
  0   1
174 826
```

```
# Create bar chart for health status
health_status <- ggplot(plant, aes(x = factor(health_status), fill = factor(health_status))) +
  geom_bar(color = "black") +
  scale_fill_manual(values = c("red", "green")) +
  labs(title = "Counts of Health Status",
       x = "Health Status (0 = Unhealthy, 1 = Healthy)",
       y = "Count",
       fill = "Health Status")

ggsave("health_status.png", plot = health_status, width = 5, height = 3, dpi = 300)
health_status
```



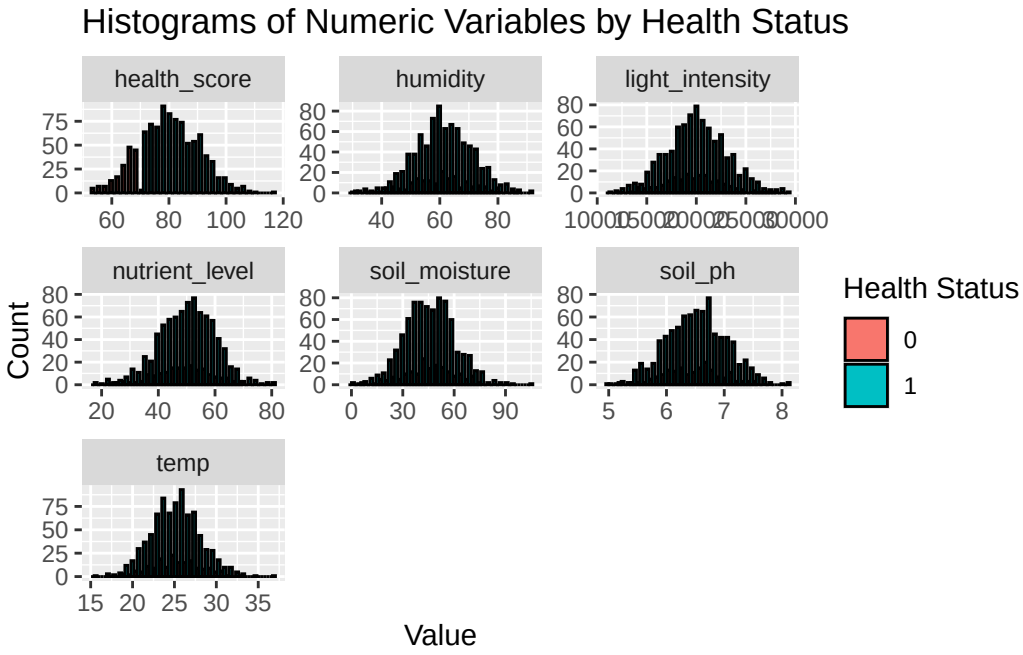
Summaries across health status

```
# Split the dataset by health_status
plant_numeric <- plant[sapply(plant, is.numeric)]
split_data <- split(plant_numeric, plant$health_status)  # Split by health_status

# Apply summary to each group
lapply(split_data, summary)
```

```
$`0`
      temp          humidity     soil_moisture       soil_ph
 Min.   :18.80   Min.   :30.79   Min.   : 5.937   Min.   :5.035
 1st Qu.:23.33   1st Qu.:53.46   1st Qu.:37.318   1st Qu.:6.123
 Median :25.29   Median :60.85   Median :47.214   Median :6.528
 Mean   :25.43   Mean   :60.25   Mean   :46.301   Mean   :6.480
 3rd Qu.:27.30   3rd Qu.:66.35   3rd Qu.:56.048   3rd Qu.:6.801
 Max.   :32.90   Max.   :85.80   Max.   :75.184   Max.   :7.592
 nutrient_level  light_intensity  health_score
 Min.   :23.75   Min.   :12694   Min.   :52.87
 1st Qu.:43.52   1st Qu.:17863   1st Qu.:62.72
 Median :49.97   Median :19852   Median :65.95
```

```
 Mean   :49.33   Mean   :19792   Mean   :64.90
 3rd Qu.:55.75   3rd Qu.:21504   3rd Qu.:68.01
 Max.   :69.45   Max.   :27618   Max.   :69.99

$`1`
      temp          humidity     soil_moisture         soil_ph
 Min.   :15.28   Min.   :30.60   Min.   : -0.2927   Min.   :5.047
 1st Qu.:22.97   1st Qu.:54.00   1st Qu.: 35.0050   1st Qu.:6.133
 Median :24.96   Median :60.48   Median : 44.6637   Median :6.498
 Mean   :24.98   Mean   :60.80   Mean   : 44.8319   Mean   :6.493
 3rd Qu.:26.90   3rd Qu.:67.36   3rd Qu.: 54.3840   3rd Qu.:6.839
 Max.   :36.56   Max.   :91.93   Max.   :103.8936   Max.   :8.122
 nutrient_level  light_intensity  health_score
 Min.   :18.23   Min.   :11301   Min.   : 70.02
 1st Qu.:43.15   1st Qu.:17943   1st Qu.: 76.42
 Median :49.77   Median :19874   Median : 81.43
 Mean   :49.55   Mean   :19874   Mean   : 82.84
 3rd Qu.:56.55   3rd Qu.:21905   3rd Qu.: 88.72
 Max.   :81.13   Max.   :29295   Max.   :115.29
```

```r
# Select only numeric variables and include health_status for grouping
numeric_vars <- plant %>%
  select(-plant_id) %>%
  pivot_longer(cols = -health_status, names_to = "variable", values_to = "value")

# Create histograms of numeric variables grouped by health_status
grouped_variables <- ggplot(numeric_vars, aes(x = value, fill = factor(health_status))) +
  geom_histogram(position = "dodge", bins = 30, color = "black") +
  facet_wrap(~ variable, scales = "free", ncol = 3) +
  labs(title = "Histograms of Numeric Variables by Health Status",
       x = "Value",
       y = "Count",
       fill = "Health Status")

ggsave("grouped_variables.png", plot = grouped_variables, width = 8, height = 6, dpi = 300)
grouped_variables
```



One-parameter model (Beta-Binomial conjugate model)

MCMC

```
# Convert health status back to integer
plant$health_status <- as.integer(plant$health_status)

# Find number of healthy plants and the total nummber of plants
y <- sum(plant$health_status)
n <- nrow(plant)

# Data list for Stan
data <- list(
  y = y,
  n = n,
  alpha = 1,  # Prior shape parameter alpha
  beta = 1     # Prior shape parameter beta
)
```

```
# Stan model code
stan_code <- "
data {
  int y;  // Number of healthy plants
  int n;  // Total number of plants
  real alpha;  // Prior parameter for Beta distribution
  real beta;   // Prior parameter for Beta distribution
}

parameters {
  real<lower=0, upper=1> p;  // Probability of being healthy
}

model {
  p ~ beta(alpha, beta);  // Prior: Beta distribution for p
  y ~ binomial(n, p);  // Likelihood: Binomial distribution
}

generated quantities {
  int simulated_y = binomial_rng(n, p);  // Simulated number of healthy plants
}
"

# Compile the Stan model
model_mcmc <- stan_model(model_code = stan_code)
```

```
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 14.0.0 (clang-1400.0.29.202)’
using SDK: ‘MacOSX13.1.sdk’
clang -arch x86_64 -I"/Library/Frameworks/R.framework/Resources/include" -DNDEBUG   -I"/Libra
In file included from <built-in>:1:
In file included from /Library/Frameworks/R.framework/Versions/4.4-x86_64/Resources/library/S
In file included from /Library/Frameworks/R.framework/Versions/4.4-x86_64/Resources/library/R
In file included from /Library/Frameworks/R.framework/Versions/4.4-x86_64/Resources/library/R
/Library/Frameworks/R.framework/Versions/4.4-x86_64/Resources/library/RcppEigen/include/Eigen
#include <cmath>
         ^~~~~~~
1 error generated.
make: *** [foo.o] Error 1
```

```r
# Fit the model
fit_mcmc <- sampling(model_mcmc, data = data, iter = 2000, chains = 4)
```

```
SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
Chain 1:
Chain 1: Gradient evaluation took 1.7e-05 seconds
Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.17 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: 0.011 seconds (Warm-up)
Chain 1:                0.01 seconds (Sampling)
Chain 1:                0.021 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: 0.011 seconds (Warm-up)
Chain 2:                0.011 seconds (Sampling)
Chain 2:                0.022 seconds (Total)
Chain 2:

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 3).
Chain 3:
Chain 3: Gradient evaluation took 2e-06 seconds
Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.02 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: 0.011 seconds (Warm-up)
Chain 3:                0.01 seconds (Sampling)
Chain 3:                0.021 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:
```

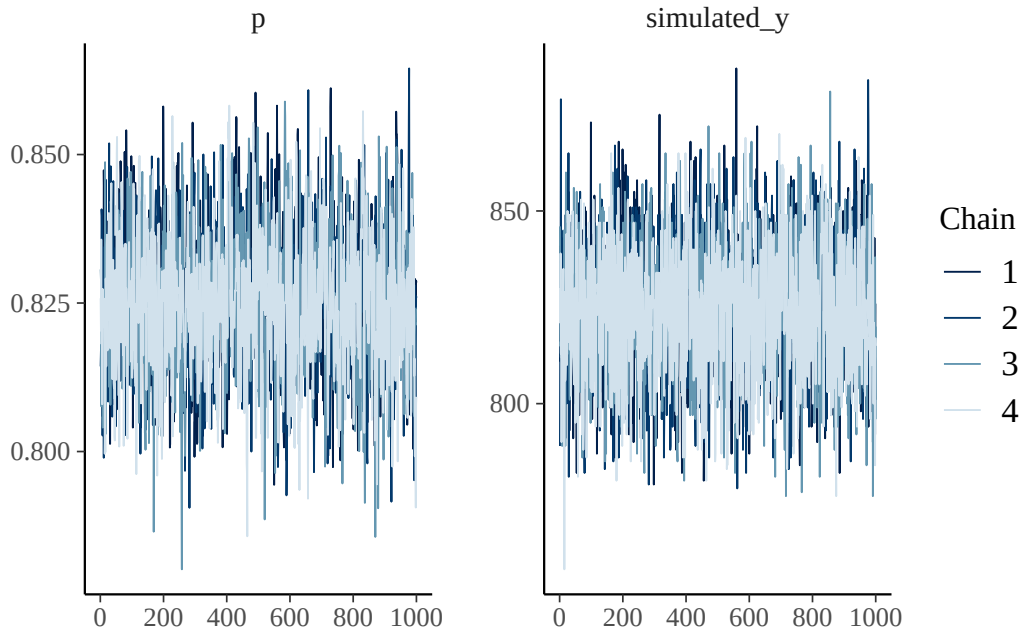
```
Chain 4:
Chain 4: Iteration:    1 / 2000 [  0%]  (Warmup)
Chain 4: Iteration:  200 / 2000 [ 10%]  (Warmup)
Chain 4: Iteration:  400 / 2000 [ 20%]  (Warmup)
Chain 4: Iteration:  600 / 2000 [ 30%]  (Warmup)
Chain 4: Iteration:  800 / 2000 [ 40%]  (Warmup)
Chain 4: Iteration: 1000 / 2000 [ 50%]  (Warmup)
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: 0.011 seconds (Warm-up)
Chain 4:                0.011 seconds (Sampling)
Chain 4:                0.022 seconds (Total)
Chain 4:
```

```r
# Print the summary of the model
print(fit_mcmc)
```

```
Inference for Stan model: anon_model.
4 chains, each with iter=2000; warmup=1000; thin=1;
post-warmup draws per chain=1000, total post-warmup draws=4000.

              mean se_mean    sd   2.5%     25%     50%     75%   97.5% n_eff
p             0.83    0.00  0.01    0.8    0.82    0.83    0.83    0.85  1321
simulated_y 825.49    0.37 16.55  791.0  814.00  826.00  837.00  856.00  1986
lp__        -464.57    0.01  0.66 -466.4 -464.71 -464.32 -464.16 -464.11  1982
            Rhat
p              1
simulated_y    1
lp__           1

Samples were drawn using NUTS(diag_e) at Wed Dec 18 17:53:05 2024.
For each parameter, n_eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor on split chains (at
convergence, Rhat=1).
```

Checks

```
# Traceplots
mcmc_trace(fit_mcmc, pars = c("p", "simulated_y"))
```



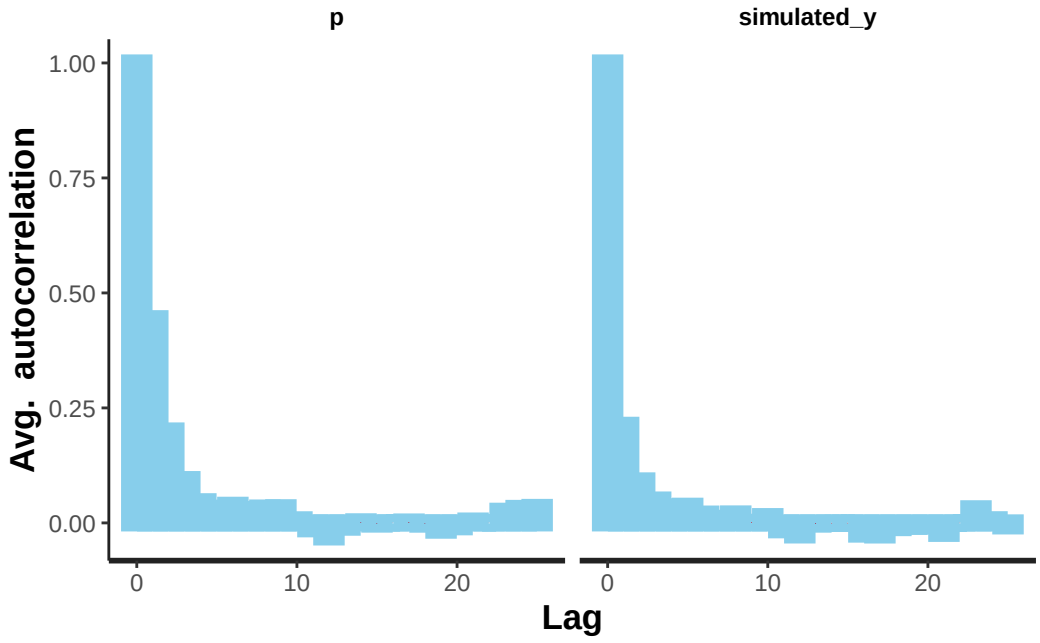

```
# Save to png
png("mcmc_traceplots.png", width = 2000, height = 1000, res = 300)

# Traceplots
mcmc_trace(fit_mcmc, pars = c("p", "simulated_y"))

dev.off()
```

```
pdf
  2
```

```
# Effective sample size
summary(fit_mcmc)$summary[,"n_eff"]
```

```
          p simulated_y         lp__
   1321.424    1986.435    1982.467
```

```r
# Autocorrelation plot
stan_ac(fit_mcmc, pars = c("p", "simulated_y"), color = "skyblue")
```

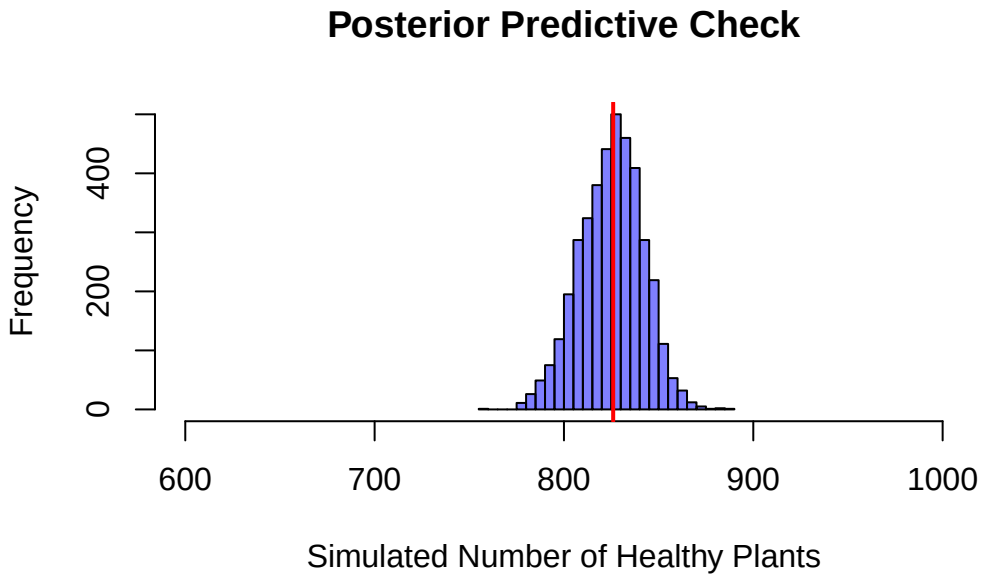



```r
# Save to png
png("mcmc_acplots.png", width = 2000, height = 1000, res = 300)

# Autocorrelation plot
stan_ac(fit_mcmc, pars = c("p", "simulated_y"), color = "skyblue")

dev.off()
```

```
pdf
  2
```

```r
# Extract the posterior samples
posterior_samples <- extract(fit_mcmc)

# Posterior predictive samples: flatten the simulated_y into a numeric vector
simulated_y <- c(posterior_samples$simulated_y)

# Plot histogram of posterior predictive samples
```

```r
hist(simulated_y, breaks = 30, col = rgb(0, 0, 1, 0.5),
     xlim = c(600, n), main = "Posterior Predictive Check",
     xlab = "Simulated Number of Healthy Plants", ylab = "Frequency")
abline(v = y, col = "red", lwd = 2)  # Add vertical line for observed y
```




```r
# Interpretation
cat("Observed number of healthy plants:", y, "\n")
```

```
Observed number of healthy plants: 826
```

```r
cat("Posterior predictive mean number of healthy plants:", mean(simulated_y), "\n")
```

```
Posterior predictive mean number of healthy plants: 825.4855
```

```r
cat("Posterior predictive credible interval for number of healthy plants:",
    quantile(simulated_y, probs = c(0.025, 0.975)), "\n")
```

```
Posterior predictive credible interval for number of healthy plants: 791 856
```

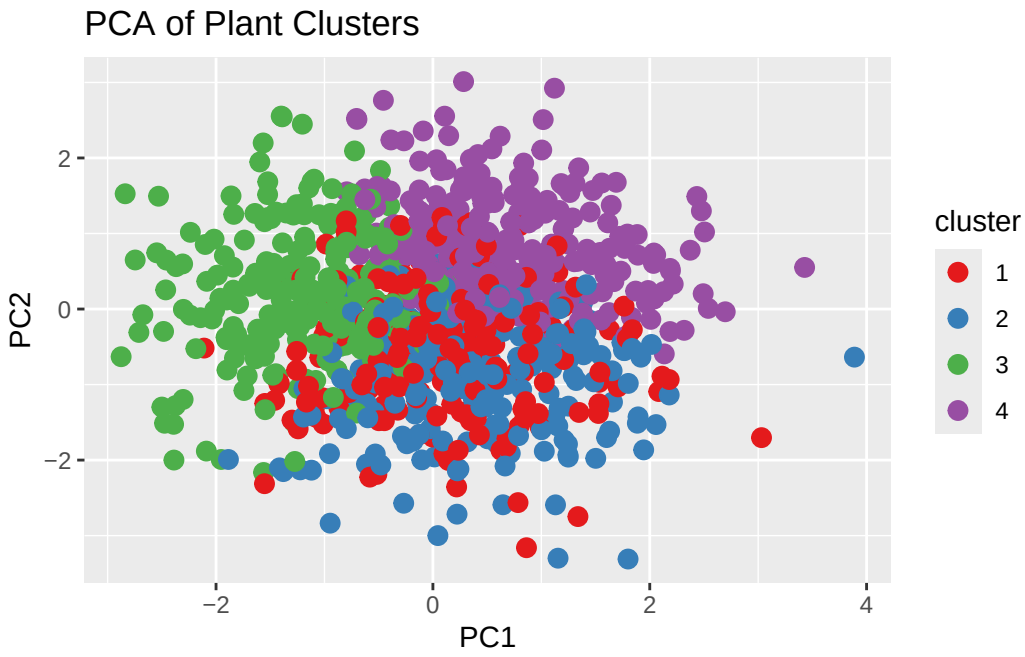
```r
# Save to png
png("mcmc_post_samples.png", width = 2000, height = 1700, res = 300)

# Plot histogram of posterior predictive samples
hist(simulated_y, breaks = 30, col = rgb(0, 0, 1, 0.5),
     xlim = c(600, n), main = "Posterior Predictive Check",
     xlab = "Simulated Number of Healthy Plants", ylab = "Frequency")
abline(v = y, col = "red", lwd = 2)  # Add vertical line for observed y

dev.off()
```

```
pdf
  2
```

Hierarchical model

```r
# Cluster plants based on predictors
set.seed(123)
kmeans_res <- kmeans(scale(plant[, c("temp", "humidity", "soil_moisture", "soil_ph",
                                     "nutrient_level", "light_intensity")]), centers = 4)
plant$cluster <- factor(kmeans_res$cluster)
plant$cluster <- as.factor(plant$cluster)
```

```r
# Visualize the clusters with PCA
pca_res <- prcomp(plant[, c("temp", "humidity", "soil_moisture", "soil_ph", "nutrient_level"

# Add PCA scores to the data
plant$pca1 <- pca_res$x[, 1]
plant$pca2 <- pca_res$x[, 2]

# Plot the clusters on the first two principal components
clusters <- ggplot(plant, aes(x = pca1, y = pca2, color = cluster)) +
  geom_point(size = 3) +
  scale_color_brewer(palette = "Set1") +
  labs(title = "PCA of Plant Clusters", x = "PC1", y = "PC2")

ggsave("clusters.png", plot = clusters, width = 6, height = 4, dpi = 300)
clusters
```




```
hierarchical_code <- "
data {
  int<lower=0> n;                    // Number of plants
  int<lower=0, upper=1> y[n];        // Health status (binary)
  int<lower=1> k;                    // Number of predictors
  matrix[n, k] X;                    // Predictor matrix (k predictors: Temp, Humidity, etc.)
  int<lower=1> J;                    // Number of clusters (latent groups)
  int<lower=1, upper=J> cluster[n]; // Cluster assignment for each plant
}

parameters {
  real alpha[J];                    // Random intercepts for each cluster
  vector[k] beta;                   // Fixed effects for predictors
  real<lower=0> sigma_alpha;        // Standard deviation for random intercepts
}

model {
  // Priors
  beta ~ normal(0, 5);              // Weakly informative prior for predictors
  alpha ~ normal(0, sigma_alpha); // Random intercepts for clusters
  sigma_alpha ~ cauchy(0, 2);       // Prior for standard deviation

  // Likelihood
```

```
  for (i in 1:n) {
    y[i] ~ bernoulli_logit(alpha[cluster[i]] + X[i] * beta);
  }
}

generated quantities {
  int simulated_y;  // Total number of healthy plants (simulated)

  // Simulate the total number of healthy plants using Bernoulli trials directly
  simulated_y = 0;
  for (i in 1:n) {
    // Calculate the probability for plant i (log-odds transformed to probability)
    real p = inv_logit(alpha[cluster[i]] + dot_product(X[i], beta));  // Use inv_logit for pr

    // Simulate a binary outcome for plant i
    simulated_y += bernoulli_rng(p);  // bernoulli_rng gives 0 or 1 based on p
  }
}
"

# Compile the model
hierarchical_model <- stan_model(model_code = hierarchical_code)
```

```
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 14.0.0 (clang-1400.0.29.202)'
using SDK: 'MacOSX13.1.sdk'
clang -arch x86_64 -I"/Library/Frameworks/R.framework/Resources/include" -DNDEBUG   -I"/Libra
In file included from <built-in>:1:
In file included from /Library/Frameworks/R.framework/Versions/4.4-x86_64/Resources/library/S
In file included from /Library/Frameworks/R.framework/Versions/4.4-x86_64/Resources/library/R
In file included from /Library/Frameworks/R.framework/Versions/4.4-x86_64/Resources/library/R
/Library/Frameworks/R.framework/Versions/4.4-x86_64/Resources/library/RcppEigen/include/Eigen
#include <cmath>
         ^~~~~~
1 error generated.
make: *** [foo.o] Error 1
```

```
# Define the data
data_hierarchical <- list(
```

```
  n = nrow(plant),  # Number of plants
  y = plant$health_status,  # Binary health status
  k = 6,  # Number of predictors
  X = scale(plant[, c("temp", "humidity", "soil_moisture", "soil_ph", "nutrient_level", "ligh
  J = length(unique(plant$cluster)),  # Number of clusters
  cluster = as.integer(plant$cluster)  # Cluster assignments as integers
)
```

```
# Fit the model
fit_hierarchical <- sampling(
  hierarchical_model,
  data = data_hierarchical,
  iter = 2000,           # Number of iterations per chain
  chains = 4,            # Number of chains
  control = list(
    adapt_delta = 0.99,   # Higher adapt_delta for stability
    max_treedepth = 15    # Increase max tree depth to avoid divergent transitions
  ),
  cores = 4,
  seed = 123             # For reproducibility
)
```

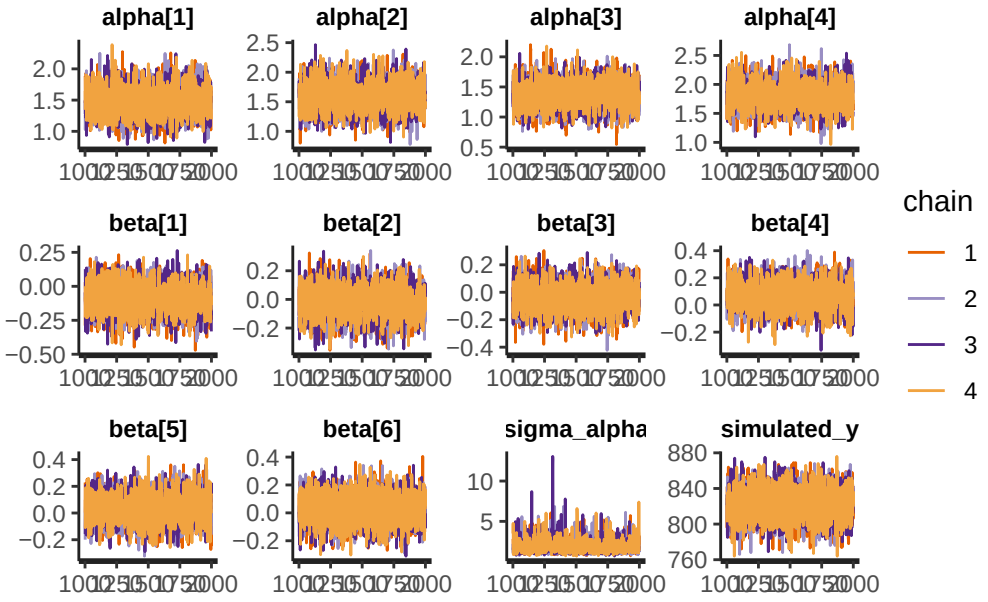
```
print(fit_hierarchical, probs = c(0.025, 0.5, 0.975))  # Posterior mean and credible interval
```

```
Inference for Stan model: anon_model.
4 chains, each with iter=2000; warmup=1000; thin=1;
post-warmup draws per chain=1000, total post-warmup draws=4000.

               mean se_mean    sd    2.5%     50%   97.5% n_eff Rhat
alpha[1]       1.49    0.00  0.23    1.07    1.49    1.95  2300    1
alpha[2]       1.59    0.00  0.23    1.15    1.59    2.06  2458    1
alpha[3]       1.37    0.00  0.21    0.95    1.36    1.79  2285    1
alpha[4]       1.82    0.00  0.22    1.40    1.82    2.25  2521    1
beta[1]       -0.09    0.00  0.10   -0.29   -0.09    0.12  2421    1
beta[2]       -0.02    0.00  0.10   -0.23   -0.02    0.19  2675    1
beta[3]       -0.04    0.00  0.10   -0.23   -0.04    0.16  2960    1
beta[4]        0.07    0.00  0.10   -0.12    0.07    0.26  2904    1
beta[5]        0.03    0.00  0.10   -0.17    0.03    0.23  2619    1
beta[6]        0.03    0.00  0.10   -0.18    0.03    0.23  2662    1
sigma_alpha    1.94    0.02  0.86    0.95    1.74    4.05  2009    1
simulated_y  823.21    0.24 16.81  789.00  823.00  855.00  5063    1
```

```
lp__         -467.94    0.06   2.44 -473.64 -467.55 -464.25  1629    1

Samples were drawn using NUTS(diag_e) at Wed Dec 18 17:54:46 2024.
For each parameter, n_eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor on split chains (at
convergence, Rhat=1).
```

Checks

```
traceplot(fit_hierarchical, pars = c("alpha", "beta", "sigma_alpha", "simulated_y"))
```



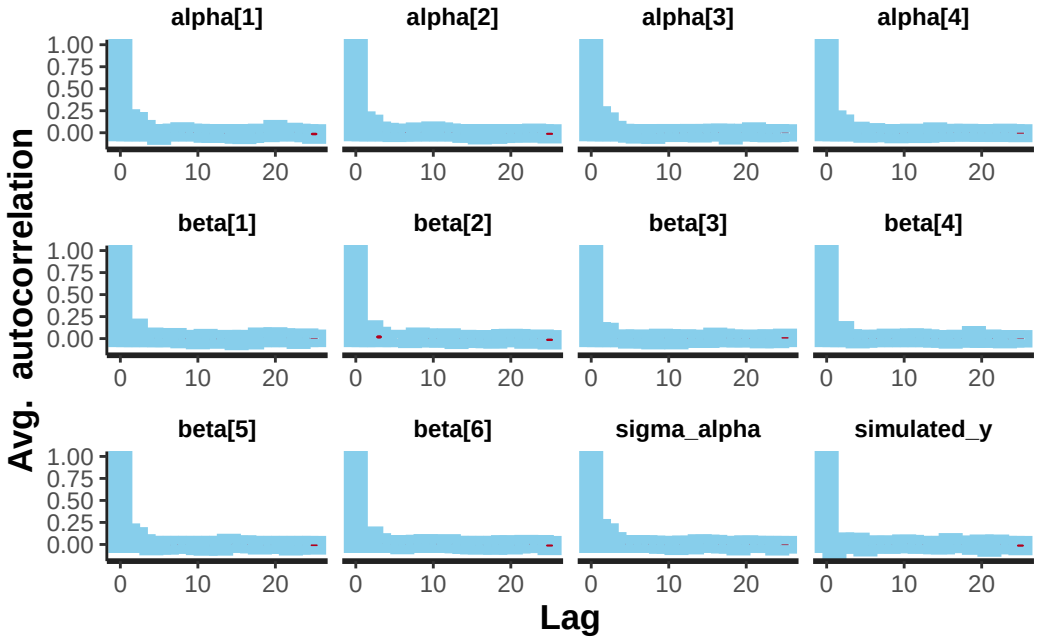

```
# Save to png
png("hierarchical_traceplots.png", width = 2000, height = 1700, res = 300)

traceplot(fit_hierarchical, pars = c("alpha", "beta", "sigma_alpha", "simulated_y"))

dev.off()
```

```
pdf
  2
```

```r
# Effective sample size
summary(fit_hierarchical)$summary[,"n_eff"]
```

```
   alpha[1]    alpha[2]    alpha[3]    alpha[4]     beta[1]     beta[2]
   2300.339    2458.338    2285.003    2521.494    2421.235    2675.320
    beta[3]     beta[4]     beta[5]     beta[6] sigma_alpha simulated_y
   2960.297    2903.990    2618.904    2662.159    2009.038    5062.672
       lp__
   1629.139
```

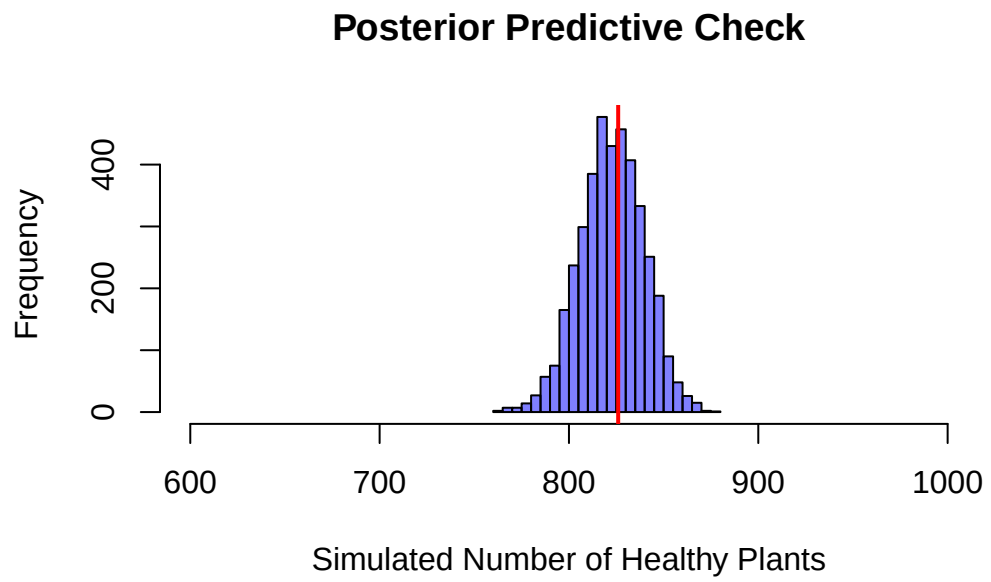
```r
# Autocorrelation plot
stan_ac(fit_hierarchical, pars = c("alpha", "beta", "sigma_alpha", "simulated_y"), color = "s
```




```r
# Save to png
png("hierarchical_acplots.png", width = 2000, height = 1700, res = 300)

# Autocorrelation plot
stan_ac(fit_hierarchical, pars = c("alpha", "beta", "sigma_alpha", "simulated_y"), color = "s

dev.off()
```

```
pdf
  2
```

```r
# Extract the posterior samples
posterior_samples <- extract(fit_hierarchical)

# Posterior predictive samples: flatten the simulated_y into a numeric vector
simulated_y <- c(posterior_samples$simulated_y)

# Plot histogram of posterior predictive samples
hist(simulated_y, breaks = 30, col = rgb(0, 0, 1, 0.5),
     xlim = c(600, n), main = "Posterior Predictive Check",
     xlab = "Simulated Number of Healthy Plants", ylab = "Frequency")
abline(v = y, col = "red", lwd = 2)  # Add vertical line for observed y
```




```r
# Interpretation
cat("Observed number of healthy plants:", y, "\n")
```

```
Observed number of healthy plants: 826
```

```r
cat("Posterior predictive mean number of healthy plants:", mean(simulated_y), "\n")
```

```
Posterior predictive mean number of healthy plants: 823.2065
```

```r
cat("Posterior predictive credible interval for number of healthy plants:",
    quantile(simulated_y, probs = c(0.025, 0.975)), "\n")
```

```
Posterior predictive credible interval for number of healthy plants: 789 855
```

```r
# Save to png
png("hierarchical_post_samples.png", width = 2000, height = 1700, res = 300)

# Plot histogram of posterior predictive samples
hist(simulated_y, breaks = 30, col = rgb(0, 0, 1, 0.5),
     xlim = c(600, n), main = "Posterior Predictive Check",
     xlab = "Simulated Number of Healthy Plants", ylab = "Frequency")
abline(v = y, col = "red", lwd = 2)  # Add vertical line for observed y

dev.off()
```

```
pdf
  2
```