STAT 656: HW2

Rohan Dekate

```
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)
```

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

library('bayesplot')
```

```
## This is bayesplot version 1.14.0
```

```
## - 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
```

```
library('ggplot2')
library('GGally')
library('patchwork')
bayesplot_theme_set(theme_default(base_size = 24, base_family = "sans"))
```

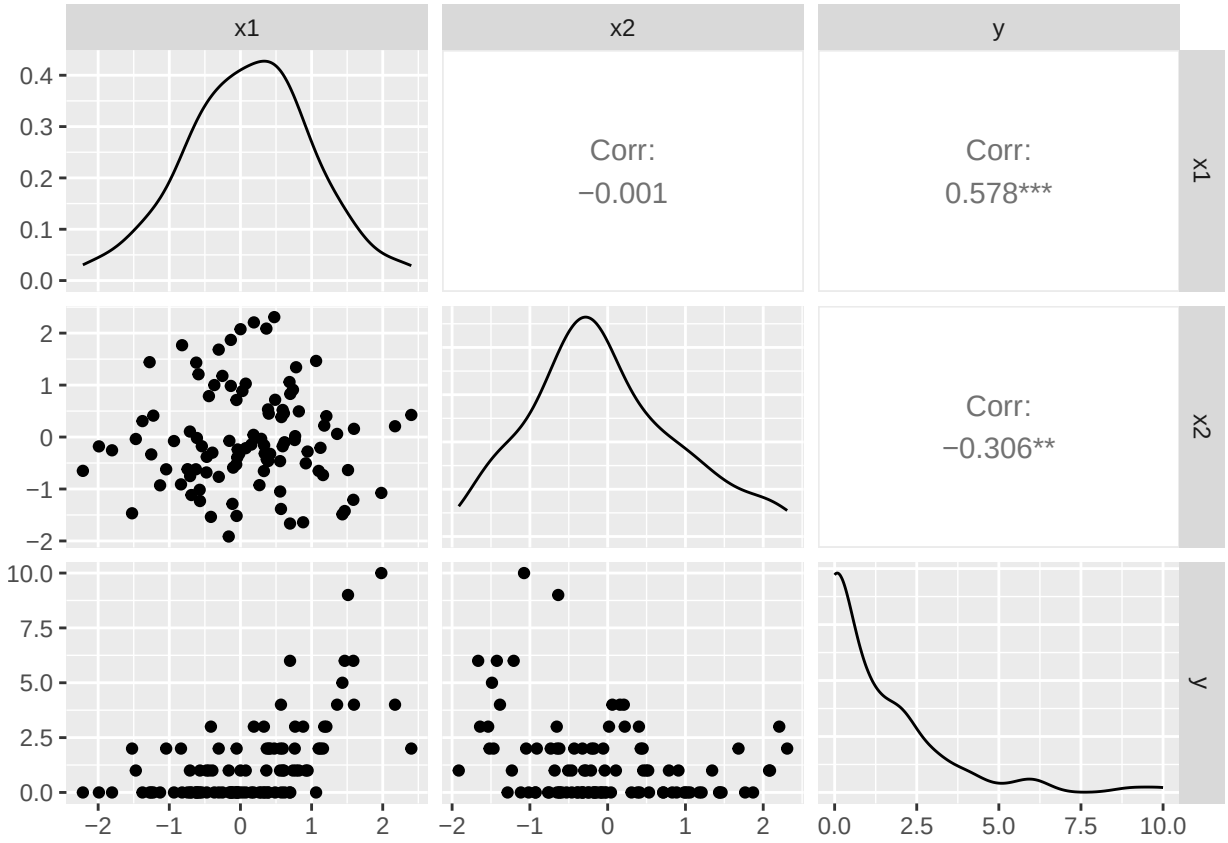
1.) With the provided data, perform a Bayesian analysis on the parameters of the model above to decide which terms in the expression for $f(x)$ you think are important. State clearly what your prior over β is, and how you arrived at your conclusion, including any useful figures (especially of the posterior distribution). You can use Stan.

```
data_p1 = read.csv(file="hw2_synthetic.csv", header=TRUE)
head(data_p1)
```

```
##           x1          x2 y
## 1 -0.6264538 -0.62036668 0
## 2  0.1836433  0.04211587 0
```

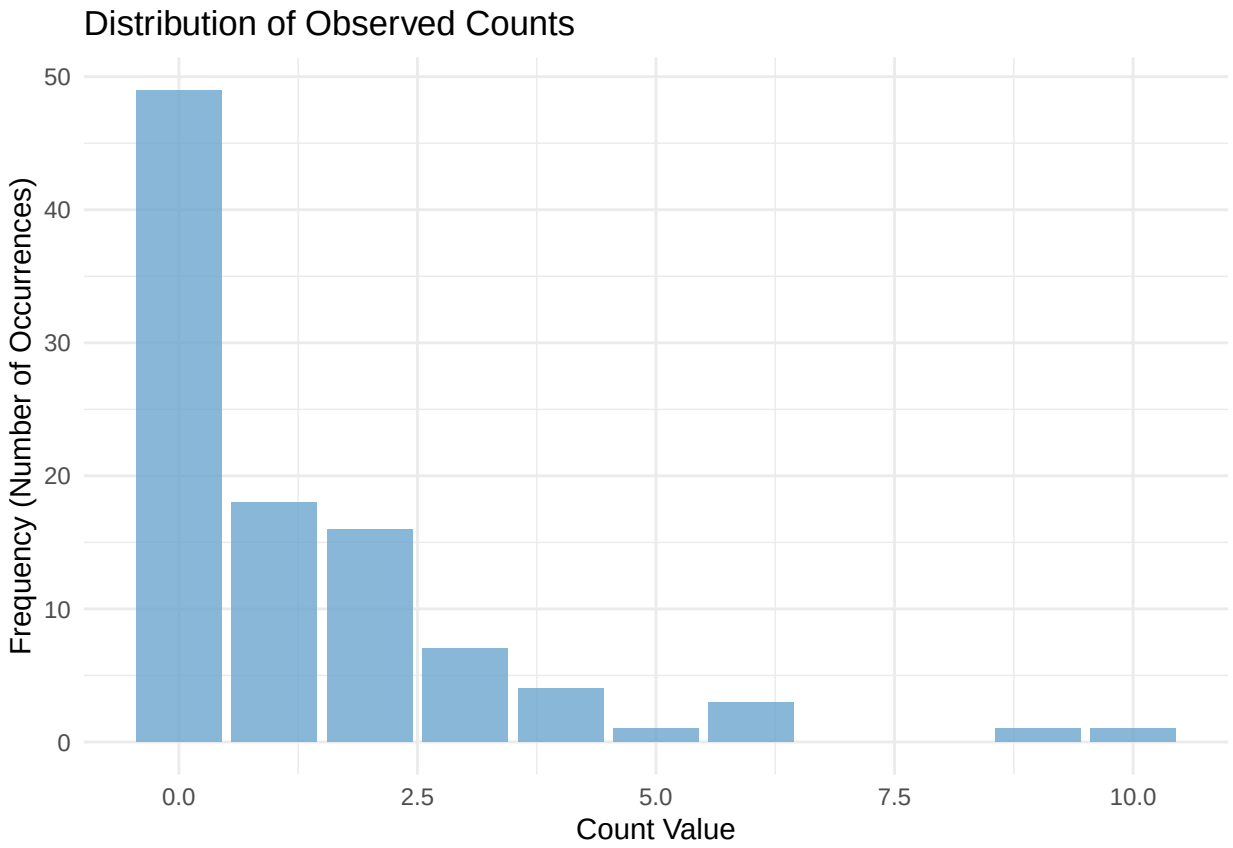
```
## 3 -0.8356286 -0.91092165 2
## 4  1.5952808  0.15802877 4
## 5  0.3295078 -0.65458464 3
## 6 -0.8204684  1.76728727 0
```

```
ggpairs(data_p1)
```



The pair-plot above does not indicate any correlations. Let's make a bar-plot for the variable we want to predict.

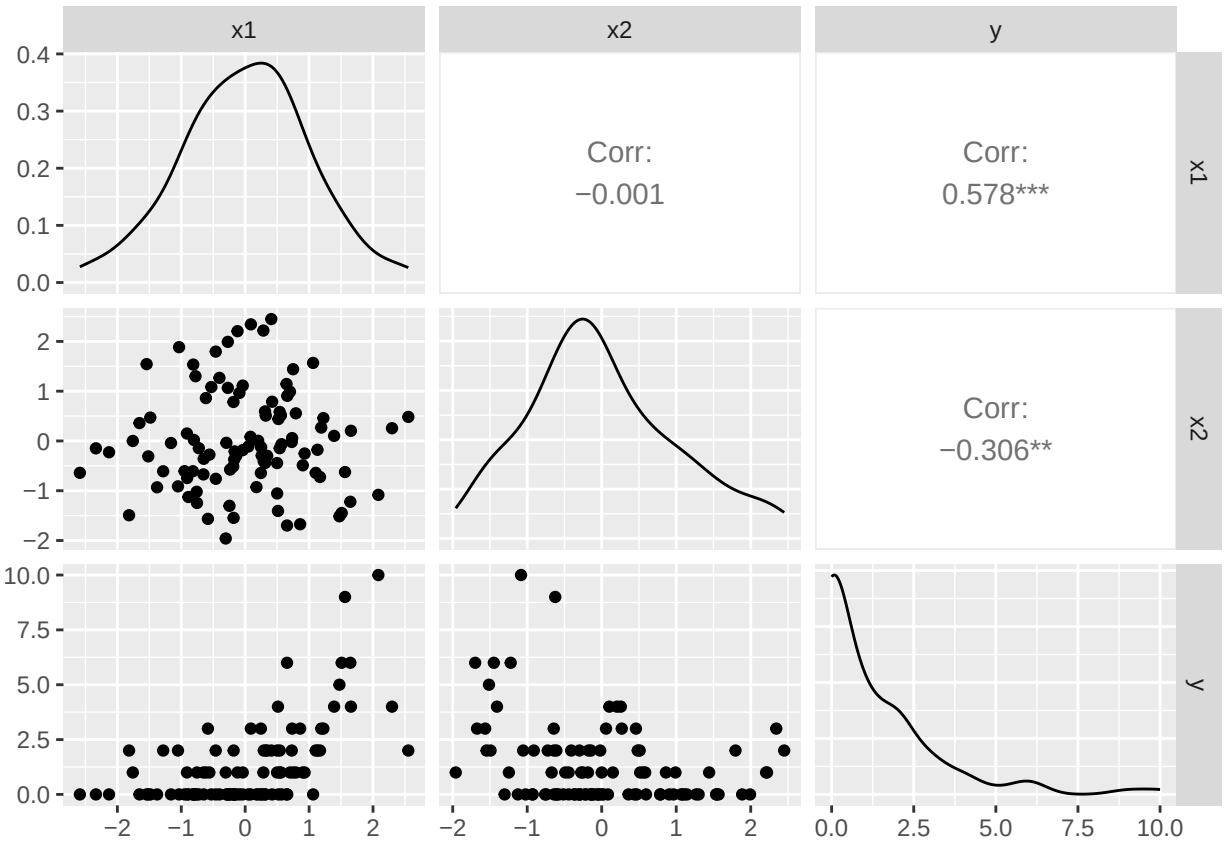
```
ggplot(data_p1, aes(x = y)) +
  geom_bar(fill = "skyblue3", alpha = 0.8) +
  labs(
    title = "Distribution of Observed Counts",
    x = "Count Value",
    y = "Frequency (Number of Occurrences)"
  ) +
  theme_minimal()
```



We should standardize the input/predictor variables.

```
synthetic_data <- data_p1
synthetic_data$x1 = (data_p1$x1 - mean(data_p1$x1))/sd(data_p1$x1)
synthetic_data$x2 = (data_p1$x2 - mean(data_p1$x2))/sd(data_p1$x2)
```

```
ggpairs(synthetic_data);
```



It doesn't look like standardization scaled the data. But we will keep it as its standard practise.

We will not log transform the output as we are told that the output variable y_i is count-valued or integer value.

We want to model y_i as follows:

$$y_i|\beta \sim \text{Poisson}(e^{f(x_i,\beta)})$$

Here the function $f(x_i, \beta)$ is the linear regression function specified in the problem as:

$$f(x_i, \beta) = \beta_0 + \beta_1 x_{i,1} + \beta_2 x_{i,2} + \beta_3 x_{i,1}^2 + \beta_4 x_{i,2}^2 + \beta_5 x_{i,1} x_{i,2}$$

We need to create a design matrix accordingly.

```
x1 <- synthetic_data$x1
x2 <- synthetic_data$x2
# Build design matrix
X <- cbind(
  intercept = 1,           # For (beta_0)
  x1        = x1,          # For (beta_1)
  x2        = x2,          # For (beta_2)
  x1_sq     = x1^2,        # For (beta_3)
  x2_sq     = x2^2,        # For (beta_4)
  x1_x2     = x1 * x2      # For (beta_5)
)
y = data_p1$y
head(X)
```

```
##      intercept          x1          x2       x1_sq       x2_sq        x1_x2
## [1,]         1 -0.81868370 -0.60817552 0.670243005 0.369877462  0.497903386
## [2,]         1  0.08322869  0.08343846 0.006927015 0.006961976  0.006944473
## [3,]         1 -1.05156608 -0.91150710 1.105791220 0.830845192  0.958509947
## [4,]         1  1.65485916  0.20444841 2.738558830 0.041799151  0.338333317
## [5,]         1  0.24562521 -0.64389816 0.060331743 0.414604836 -0.158157619
## [6,]         1 -1.03468761  1.88447102 1.070578456 3.551231020 -1.949838819
```

Now we write a Stan model where the likelihood is sampled from a Poisson distribution.

The β_i coefficients for a linear model can be any real number. Thus we need to choose a prior that sample real values. This narrows down our options to choosing a Normal prior.

The prior is chosen as $\beta \sim \text{Normal}(0, 10^2)$. This is a weakly informative prior which is quite flexible to account for the variation in the observations.

```
linreg_poisson_code = "
// Data are things you observe/condition on

data {
int<lower=0> N; // number of data items
int<lower=0> K; // number of predictors
real<lower=0> pr_sd; // std dev of the prior
matrix[N, K] x; // predictor matrix
int y[N]; // output vector is integer/count-valued
}

// parameters are latent quantities whose conditional/posterior distributions you are interested in
parameters {
vector[K] beta; // coefficients for predictors
}


// The actual Bayesian model goes here
model {
beta ~ normal(0,pr_sd); // Note: beta is k-dim
y ~ poisson_log(x * beta); // likelihood
}

// useful for posterior predictive checks
generated quantities {
int y_rep[N];
y_rep = poisson_log_rng(x * beta);
}"
#linreg_poisson = stan_model(model_code=linreg_poisson_code)
```

```
# Compiling a model can take time. If you plan to use it across R sessions,
# you can save it as below
#saveRDS(linreg_poisson, "compiled_linreg_poisson.rds")
# Later, you can load it as:
linreg_poisson = readRDS("compiled_linreg_poisson.rds")
#print(linreg_poisson)
```

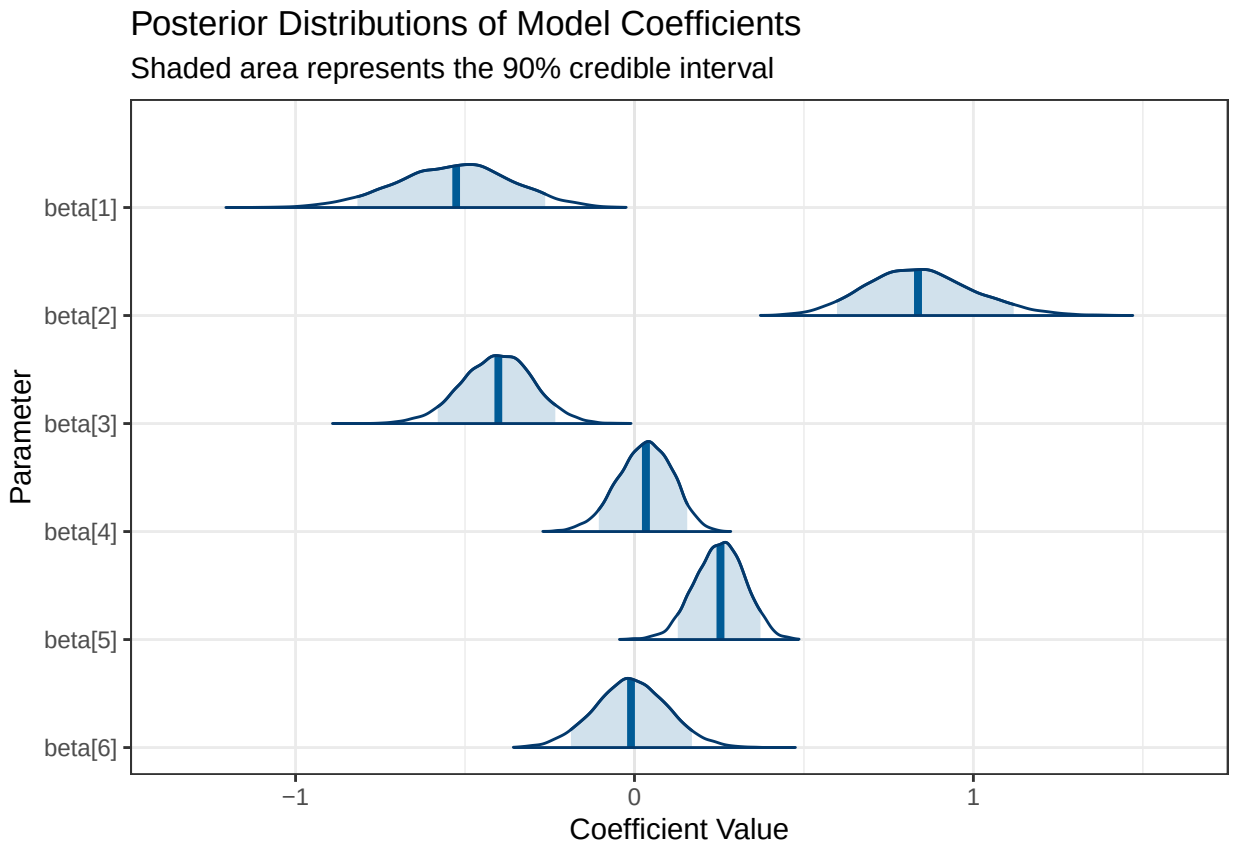
```
reg1_data = list(N = nrow(X), K = ncol(X), pr_sd = 10, x=X, y=y)
nfit_full = sampling(linreg_poisson, data=reg1_data, iter=10000, warmup=2000, chains=1)
```

```
##
```

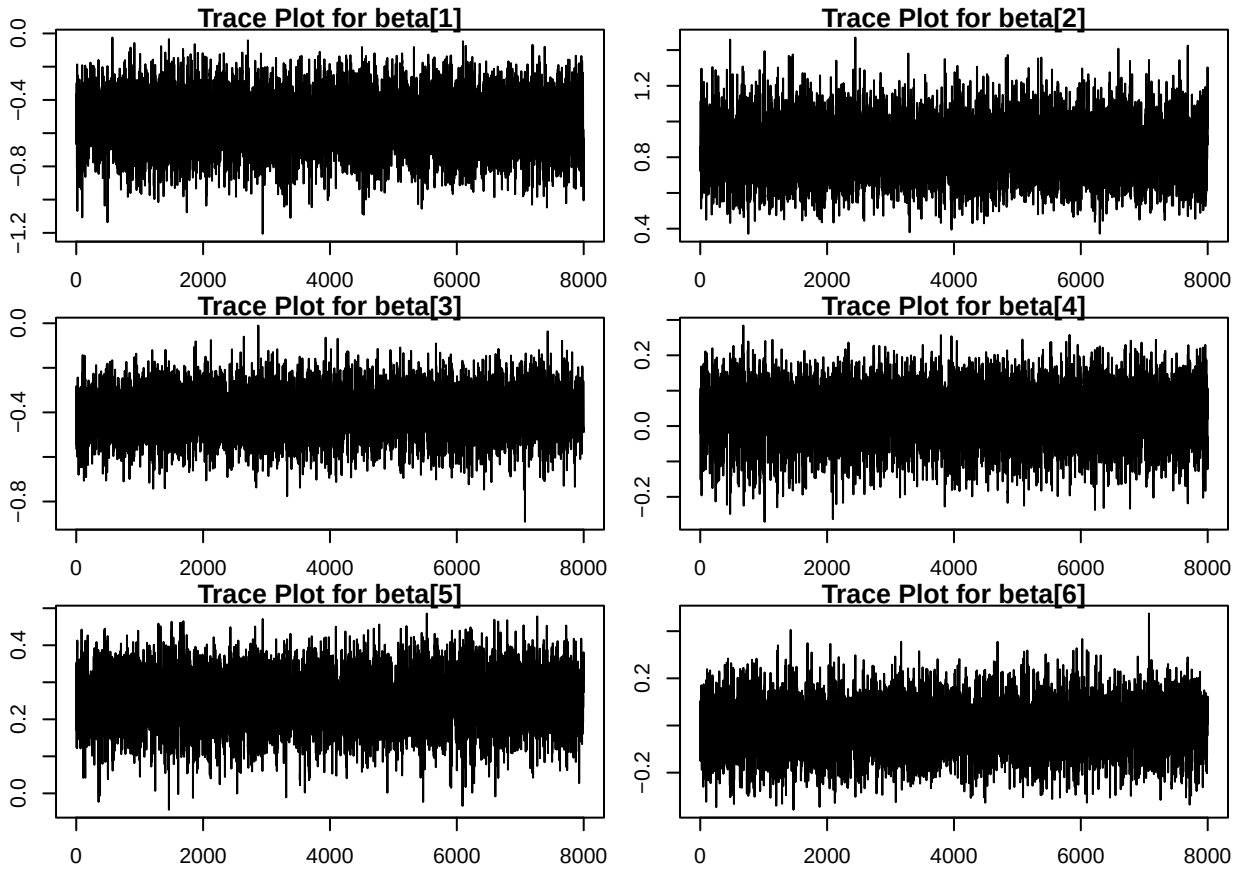
```
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 3.6e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.36 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 10000 [  0%]  (Warmup)
## Chain 1: Iteration: 1000 / 10000 [ 10%]  (Warmup)
## Chain 1: Iteration: 2000 / 10000 [ 20%]  (Warmup)
## Chain 1: Iteration: 2001 / 10000 [ 20%]  (Sampling)
## Chain 1: Iteration: 3000 / 10000 [ 30%]  (Sampling)
## Chain 1: Iteration: 4000 / 10000 [ 40%]  (Sampling)
## Chain 1: Iteration: 5000 / 10000 [ 50%]  (Sampling)
## Chain 1: Iteration: 6000 / 10000 [ 60%]  (Sampling)
## Chain 1: Iteration: 7000 / 10000 [ 70%]  (Sampling)
## Chain 1: Iteration: 8000 / 10000 [ 80%]  (Sampling)
## Chain 1: Iteration: 9000 / 10000 [ 90%]  (Sampling)
## Chain 1: Iteration: 10000 / 10000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.098 seconds (Warm-up)
## Chain 1:                0.423 seconds (Sampling)
## Chain 1:                0.521 seconds (Total)
## Chain 1:
```

```r
post_smp_full <- as.data.frame(nfit_full)
#head(post_smp_full)
```

```r
p <- mcmc_areas(post_smp_full[,1:6], prob = 0.9)
p +
  labs(
    title = "Posterior Distributions of Model Coefficients",
    subtitle = "Shaded area represents the 90% credible interval",
    x = "Coefficient Value",
    y = "Parameter"
  ) +
  theme_bw()
```




```r
beta_smp <- post_smp_full[, 1:6]

par(mfrow = c(3, 2), mar = c(2, 2, 1, 1)) # mar = (bottom, left, top, right)

for (i in 1:ncol(beta_smp)) {
  plot(
    beta_smp[[i]],
    type = 'l',
    main = paste("Trace Plot for", colnames(beta_smp)[i]),
    xlab = "MCMC Iteration",
    ylab = "Value"
  )
}
```




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

I want to check the effect of increasing the standard deviation of my prior distribution.

```
reg2_data = list(N = nrow(X), K = ncol(X), pr_sd = 100, x=X, y=y)
nfit_full_2 = sampling(linreg_poisson, data=reg2_data, iter=10000, warmup=2000, chains=1)
```

```
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 7e-06 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.07 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 10000 [  0%]  (Warmup)
## Chain 1: Iteration: 1000 / 10000 [ 10%]  (Warmup)
## Chain 1: Iteration: 2000 / 10000 [ 20%]  (Warmup)
## Chain 1: Iteration: 2001 / 10000 [ 20%]  (Sampling)
## Chain 1: Iteration: 3000 / 10000 [ 30%]  (Sampling)
## Chain 1: Iteration: 4000 / 10000 [ 40%]  (Sampling)
## Chain 1: Iteration: 5000 / 10000 [ 50%]  (Sampling)
## Chain 1: Iteration: 6000 / 10000 [ 60%]  (Sampling)
## Chain 1: Iteration: 7000 / 10000 [ 70%]  (Sampling)
## Chain 1: Iteration: 8000 / 10000 [ 80%]  (Sampling)
## Chain 1: Iteration: 9000 / 10000 [ 90%]  (Sampling)
## Chain 1: Iteration: 10000 / 10000 [100%]  (Sampling)
```

```
## Chain 1:
## Chain 1:  Elapsed Time: 0.102 seconds (Warm-up)
## Chain 1:                0.489 seconds (Sampling)
## Chain 1:                0.591 seconds (Total)
## Chain 1:
```

```
post_smp_full_2 <- as.data.frame(nfit_full_2)
#head(post_smp_full_2)
```

```
p <- mcmc_areas(post_smp_full_2[,1:6], prob = 0.9)
p +
  labs(
    title = "Posterior Distributions of Model Coefficients",
    subtitle = "Shaded area represents the 90% credible interval",
    x = "Coefficient Value",
    y = "Parameter"
  ) +
  theme_bw()
```

We see from the two plots above that the mean of β_3 (beta[4]) and β_5 (beta[6]) parameters are 0 almost 0. I suspect that they do not contribute much to the model $f(x_i, \beta)$. Thus they are not important and my next model and design matrix will ignore these parameters.

Changing the prior standard deviation did not impact the posterior distributions much.

2.) Having decided which terms in f are important, keep only those and discard the rest, resulting in a possibly simpler model. Now perform a Bayesian analysis over the parameters of this model. Note that you are using the data twice, once to select the model and next to fit the it, but we will not worry about that. Compare the posteriors for both models.

```
# Build new design matrix
x1 <- synthetic_data$x1
x2 <- synthetic_data$x2
X_2 <- cbind(
  intercept = 1,
  x1        = x1,
  x2        = x2,
  x2_sq     = x2^2
)

head(X_2)
```

```
##      intercept          x1          x2       x2_sq
## [1,]         1 -0.81868370 -0.60817552 0.369877462
## [2,]         1  0.08322869  0.08343846 0.006961976
## [3,]         1 -1.05156608 -0.91150710 0.830845192
## [4,]         1  1.65485916  0.20444841 0.041799151
## [5,]         1  0.24562521 -0.64389816 0.414604836
## [6,]         1 -1.03468761  1.88447102 3.551231020
```

Create a new regression data for the updated design matrix. We can reuse the same stan model and sample it with the same prior distribution over β. We have used the larger standard deviation for this model.

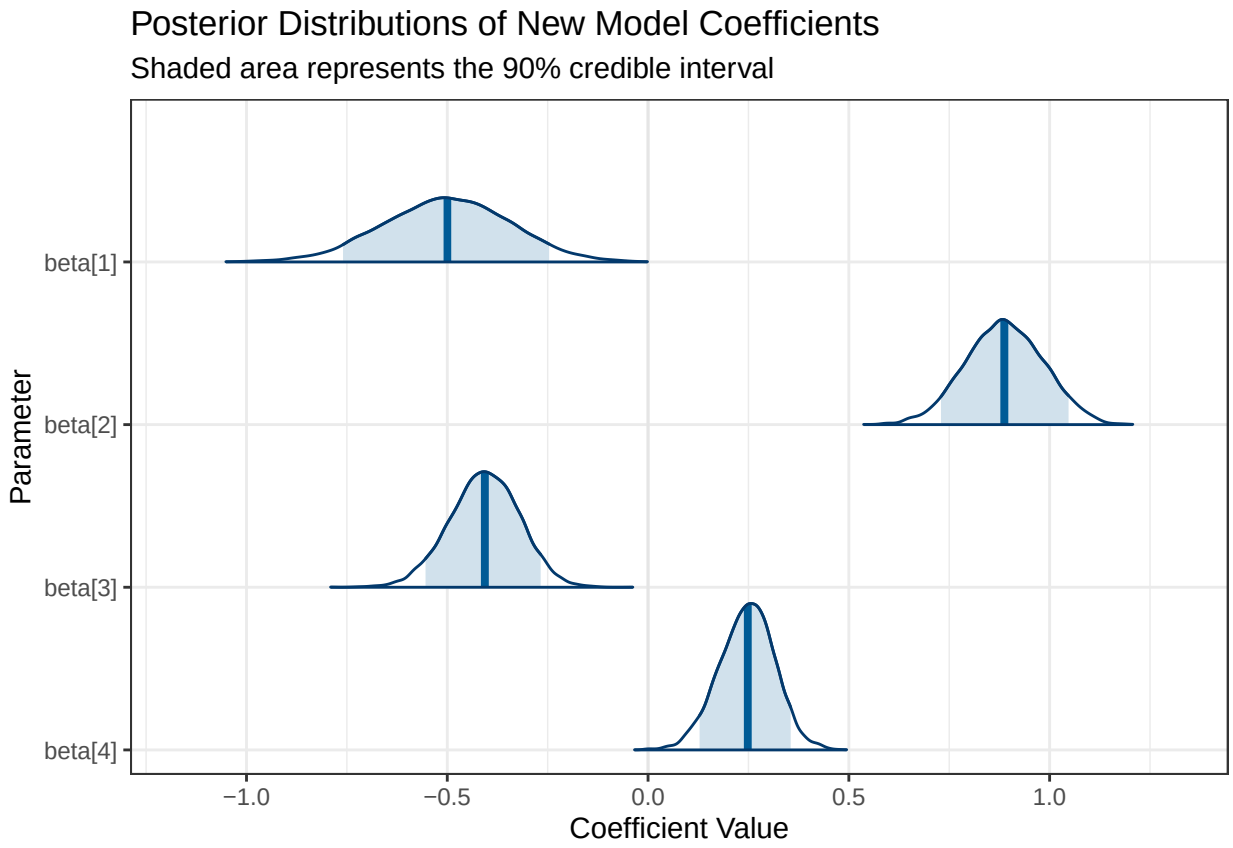
```
reg2_data = list(N = nrow(X_2), K = ncol(X_2), pr_sd = 100, x=X_2, y=y)
nfit_reduced = sampling(linreg_poisson, data=reg2_data, iter=10000, warmup=2000, chains=1)
```

```
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 7e-06 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.07 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 10000 [  0%]  (Warmup)
## Chain 1: Iteration: 1000 / 10000 [ 10%]  (Warmup)
## Chain 1: Iteration: 2000 / 10000 [ 20%]  (Warmup)
## Chain 1: Iteration: 2001 / 10000 [ 20%]  (Sampling)
## Chain 1: Iteration: 3000 / 10000 [ 30%]  (Sampling)
## Chain 1: Iteration: 4000 / 10000 [ 40%]  (Sampling)
## Chain 1: Iteration: 5000 / 10000 [ 50%]  (Sampling)
## Chain 1: Iteration: 6000 / 10000 [ 60%]  (Sampling)
## Chain 1: Iteration: 7000 / 10000 [ 70%]  (Sampling)
## Chain 1: Iteration: 8000 / 10000 [ 80%]  (Sampling)
## Chain 1: Iteration: 9000 / 10000 [ 90%]  (Sampling)
## Chain 1: Iteration: 10000 / 10000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.068 seconds (Warm-up)
## Chain 1:                0.297 seconds (Sampling)
```

```
## Chain 1:                0.365 seconds (Total)
## Chain 1:
```

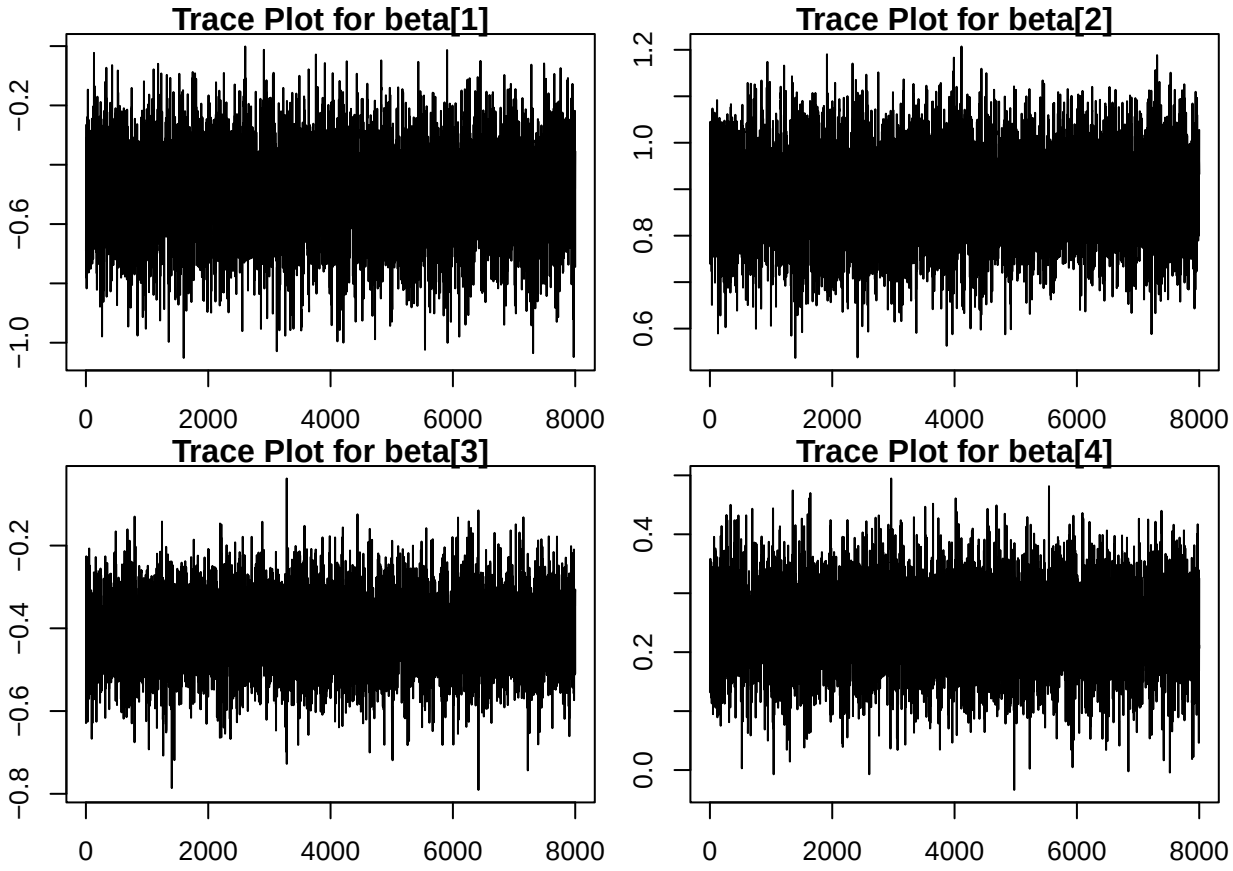
```
post_smp_reduced <- as.data.frame(nfit_reduced)
#head(post_smp_reduced)
```

```
p <- mcmc_areas(post_smp_reduced[,1:4], prob = 0.9)
p +
  labs(
    title = "Posterior Distributions of New Model Coefficients",
    subtitle = "Shaded area represents the 90% credible interval",
    x = "Coefficient Value",
    y = "Parameter"
  ) +
  theme_bw()
```




```
beta_smp <- post_smp_reduced[, 1:4]

par(mfrow = c(2, 2), mar = c(2, 2, 1, 1)) # mar = (bottom, left, top, right)

for (i in 1:ncol(beta_smp)) {
  plot(
    beta_smp[[i]],
    type = 'l',
    main = paste("Trace Plot for", colnames(beta_smp)[i]),
    xlab = "MCMC Iteration version 2",
    ylab = "Value"
  )
```

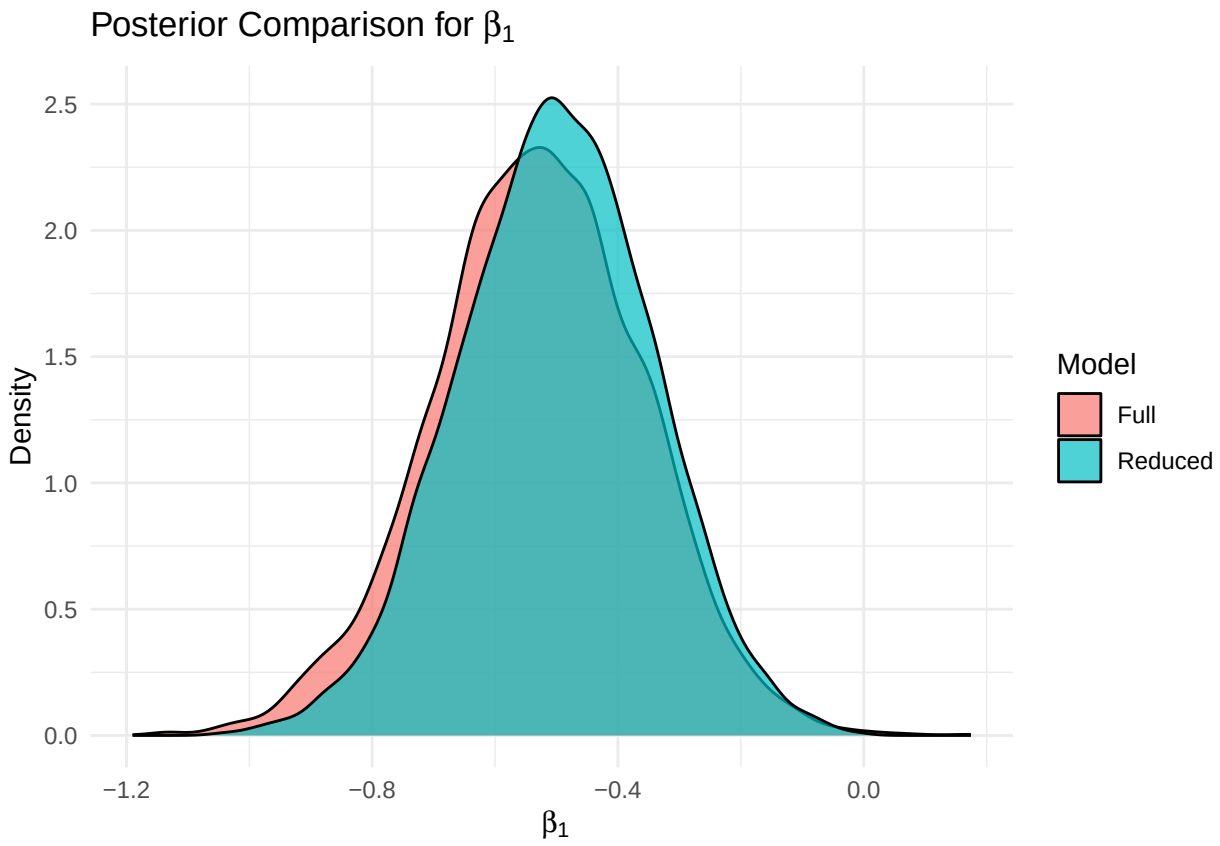
```
}
```



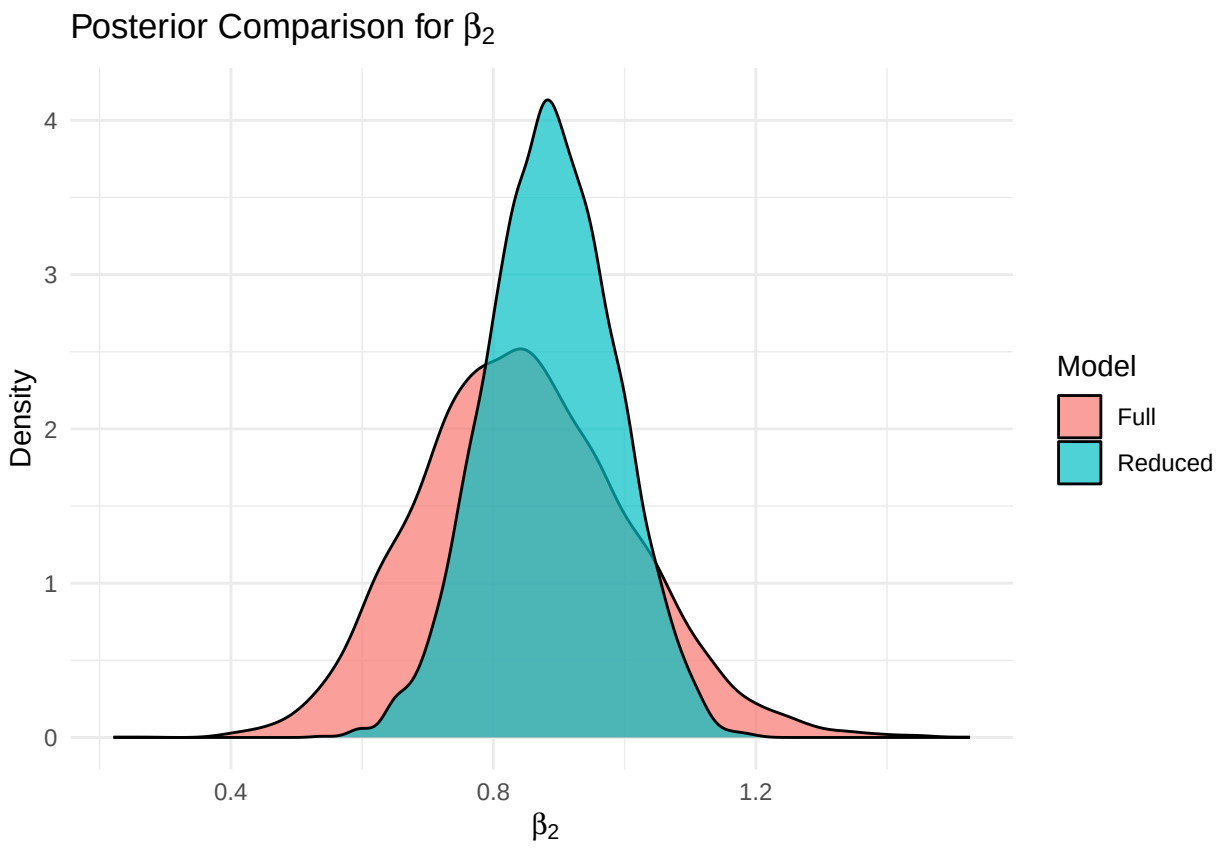

```
par(mfrow = c(1, 1))
```

We would need to compare the posteriors of the two models now.

```
comparison_df <- rbind(
  data.frame(value = post_smp_full_2$`beta[1]`, model = "Full"),
  data.frame(value = post_smp_reduced$`beta[1]`, model = "Reduced")
)

ggplot(comparison_df, aes(x = value, fill = model)) +
  geom_density(alpha = 0.7) +
  labs(
    title = bquote("Posterior Comparison for" ~ beta[1]),
    x = bquote(~ beta[1]),
    y = "Density",
    fill = "Model"
  ) +
  theme_minimal()
```

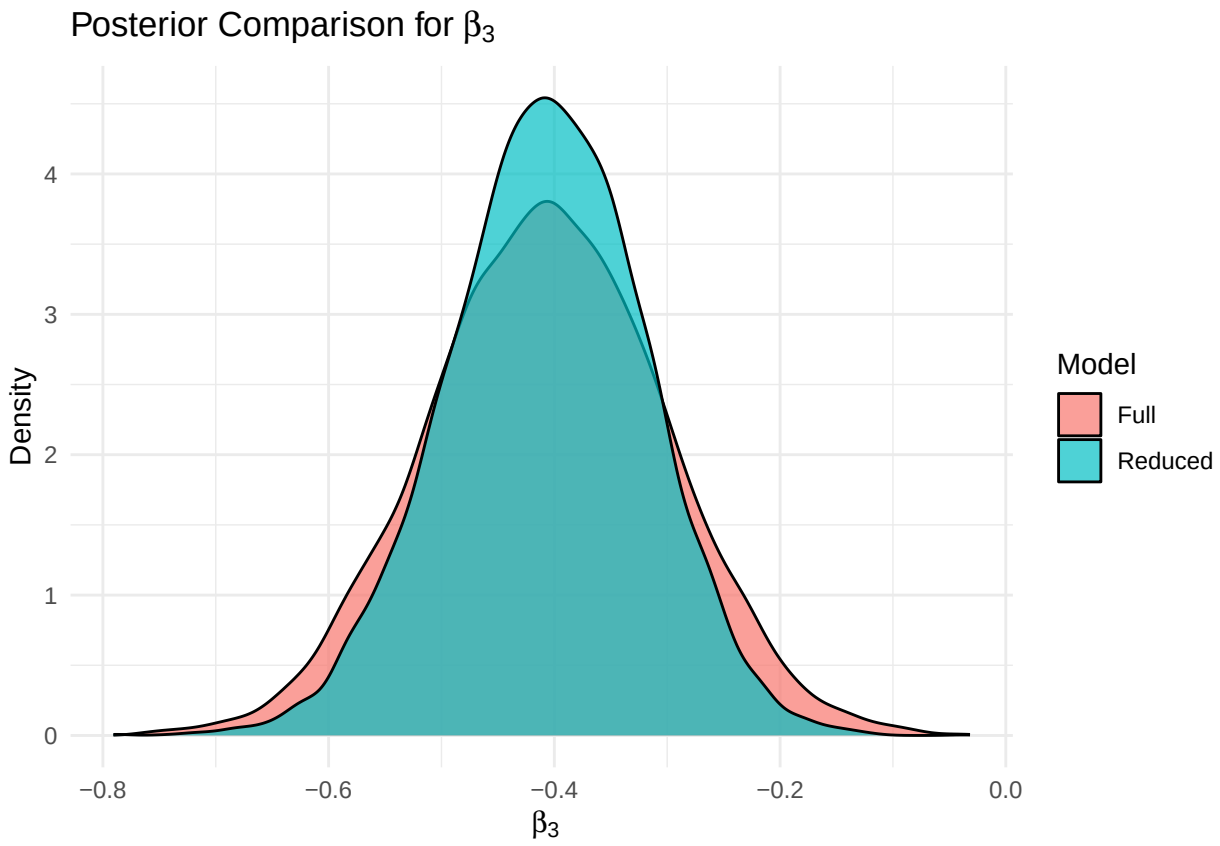



```
comparison_df <- rbind(
  data.frame(value = post_smp_full_2$`beta[2]`, model = "Full"),
  data.frame(value = post_smp_reduced$`beta[2]`, model = "Reduced")
)

ggplot(comparison_df, aes(x = value, fill = model)) +
  geom_density(alpha = 0.7) +
  labs(
    title = bquote("Posterior Comparison for" ~ beta[2]),
    x = bquote(~ beta[2]),
    y = "Density",
    fill = "Model"
  ) +
  theme_minimal()
```

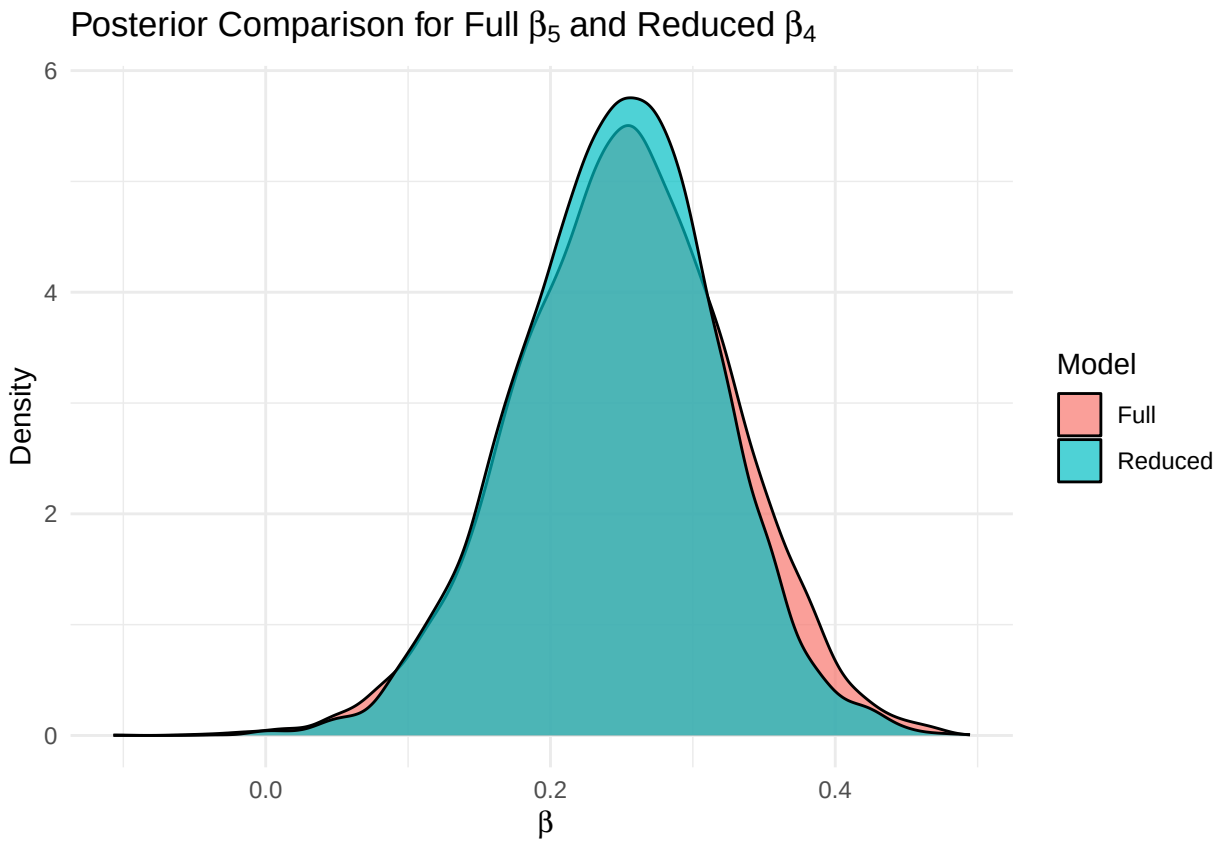


We see that for β_2 the new model has lower variance than the first model.

```
comparison_df <- rbind(
  data.frame(value = post_smp_full_2$`beta[3]`, model = "Full"),
  data.frame(value = post_smp_reduced$`beta[3]`, model = "Reduced")
)

ggplot(comparison_df, aes(x = value, fill = model)) +
  geom_density(alpha = 0.7) +
  labs(
    title = bquote("Posterior Comparison for" ~ beta[3]),
    x = bquote(~ beta[3]),
    y = "Density",
    fill = "Model"
  ) +
  theme_minimal()
```




```
comparison_df <- rbind(
  data.frame(value = post_smp_full_2$`beta[5]`, model = "Full"),
  data.frame(value = post_smp_reduced$`beta[4]`, model = "Reduced")
)

ggplot(comparison_df, aes(x = value, fill = model)) +
  geom_density(alpha = 0.7) +
  labs(
    title = bquote("Posterior Comparison for Full" ~ beta[5] ~ "and Reduced" ~ beta[4]),
    x = bquote(~ beta),
    y = "Density",
    fill = "Model"
  ) +
  theme_minimal()
```



3.) Perform posterior predictive checks for both models, being sure to explain what you are doing. Which model do you think fits the data better?

```
y_rep_full = extract(nfit_full_2)$y_rep
y_rep_reduced = extract(nfit_reduced)$y_rep
```

```
# Print the root mean-squared error (RMS) and sum of absolute errors
print("Full Model Errors")
```

```
## [1] "Full Model Errors"
```

```
sqrt(sum((colMeans(y_rep_full)-y)^2))
```

```
## [1] 11.41828
```

```
sum(abs(colMeans(y_rep_full)-y))
```

```
## [1] 82.94488
```

```
print("Reduced Model Errors")
```

```
## [1] "Reduced Model Errors"
```

```
sqrt(sum((colMeans(y_rep_reduced)-y)^2))
```

```
## [1] 11.29479
```

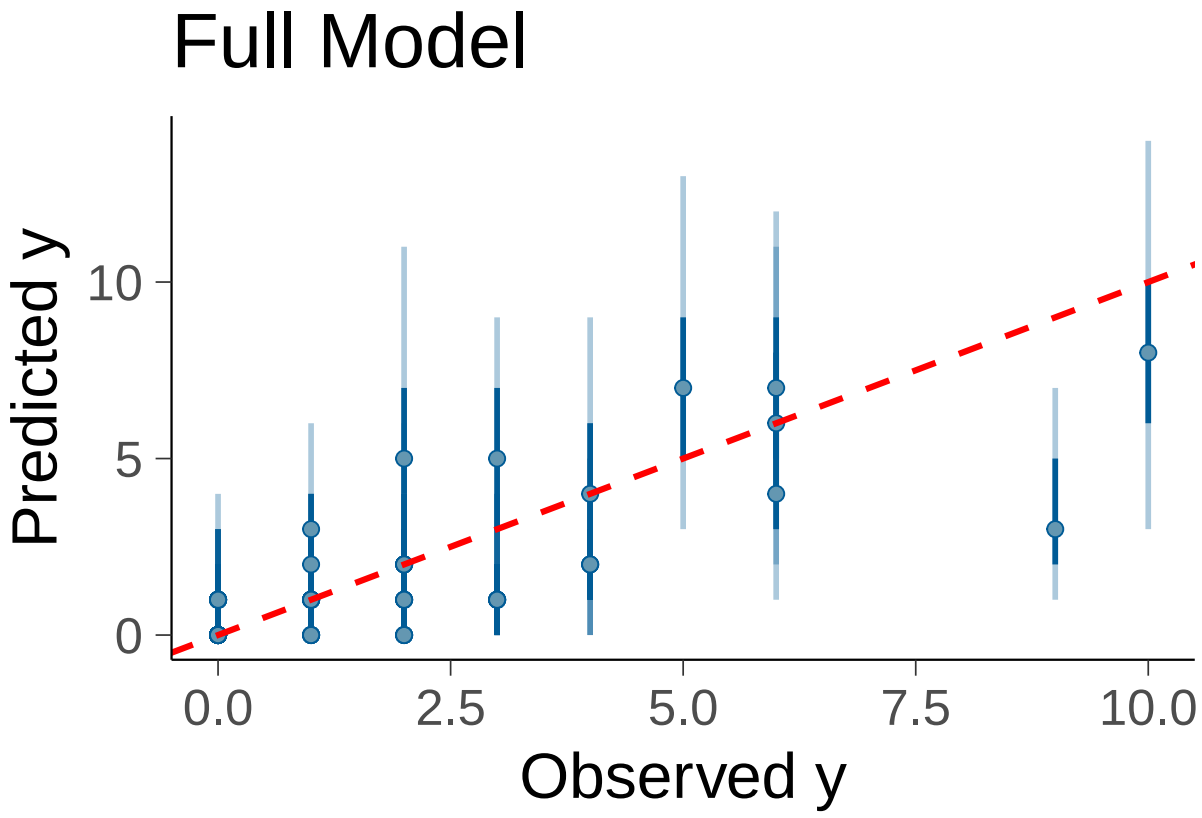
```
sum(abs(colMeans(y_rep_reduced)-y))
```

```
## [1] 82.35025
```

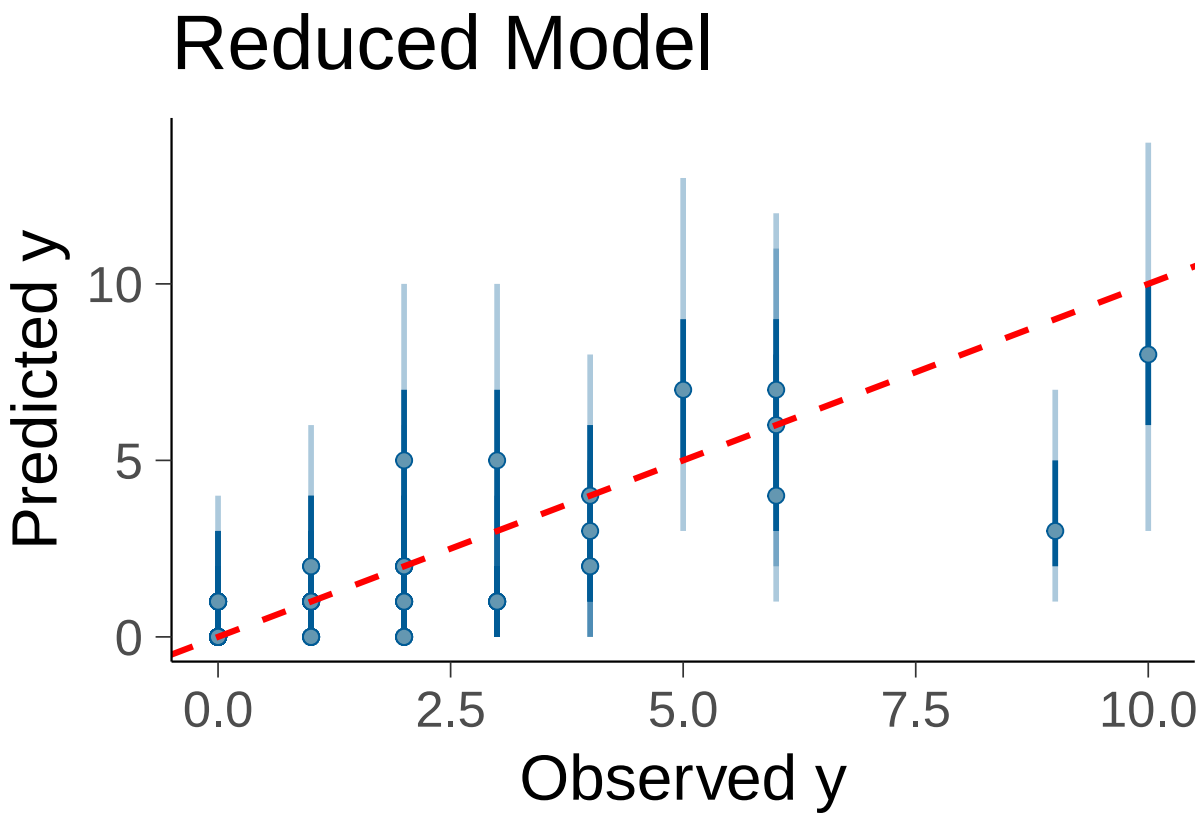
Very small improvement in the errors with the reduced model.

```
ppd_intervals(y = y_rep_full, x = y) +
  geom_abline(intercept = 0, slope = 1, linetype = "dashed", color = "red") +
  labs(title = "Full Model", y = "Predicted y", x = "Observed y")
```

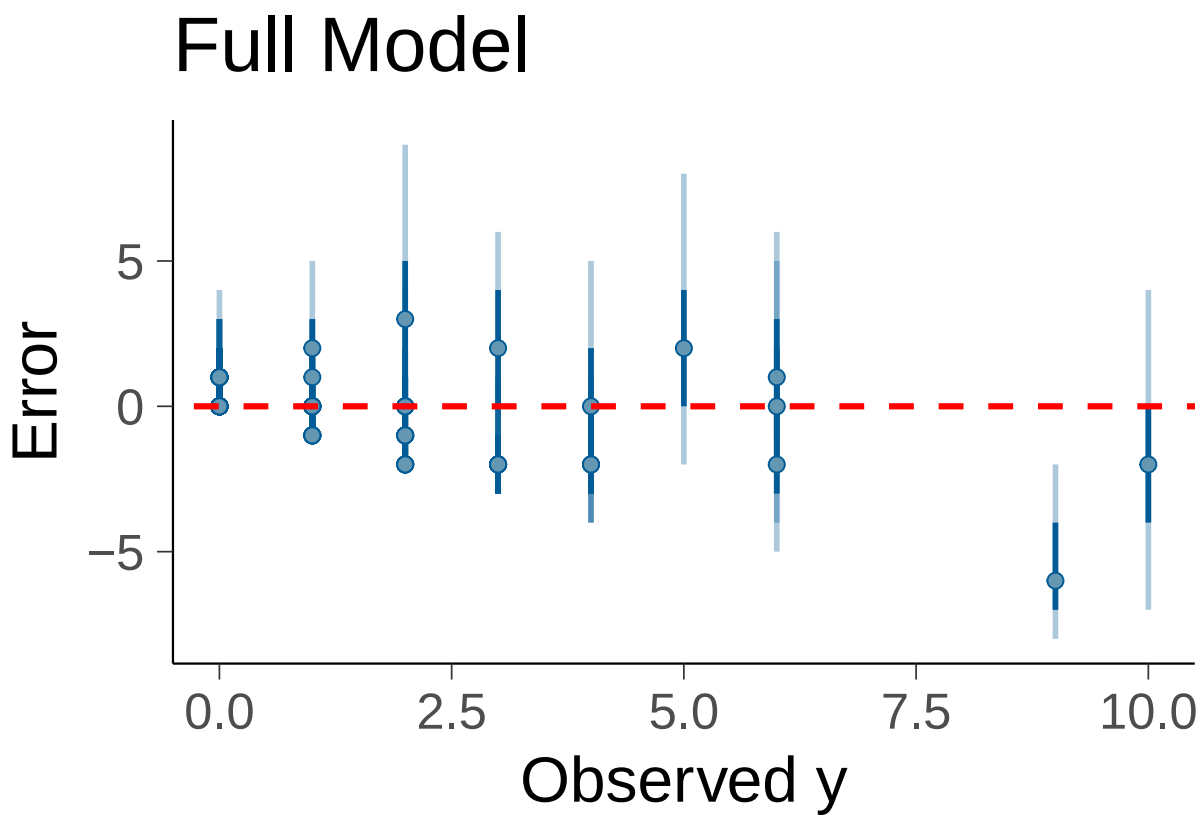
```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use `linewidth` instead.
## i The deprecated feature was likely used in the bayesplot package.
##   Please report the issue at <https://github.com/stan-dev/bayesplot/issues/>.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



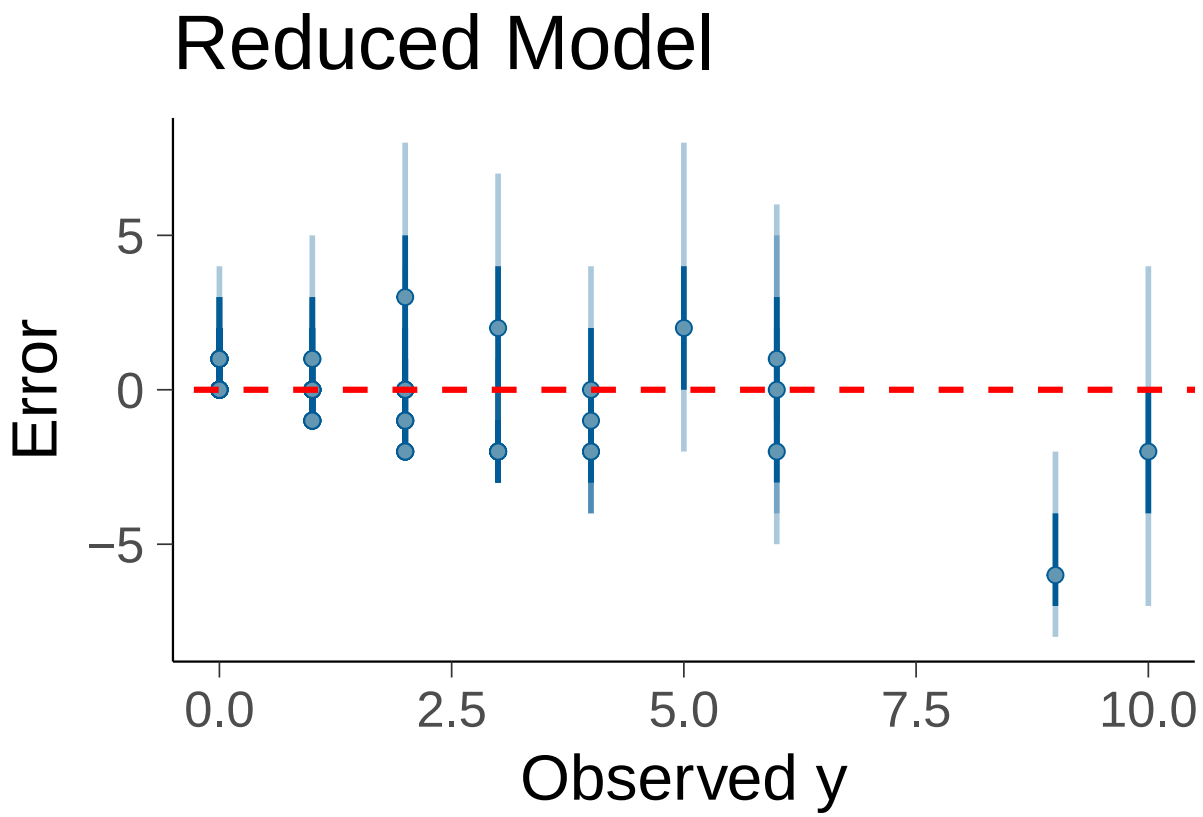

```
ppd_intervals(y = y_rep_reduced, x = y) +
  geom_abline(intercept = 0, slope = 1, linetype = "dashed", color = "red") +
  labs(title = "Reduced Model", y = "Predicted y", x = "Observed y")
```



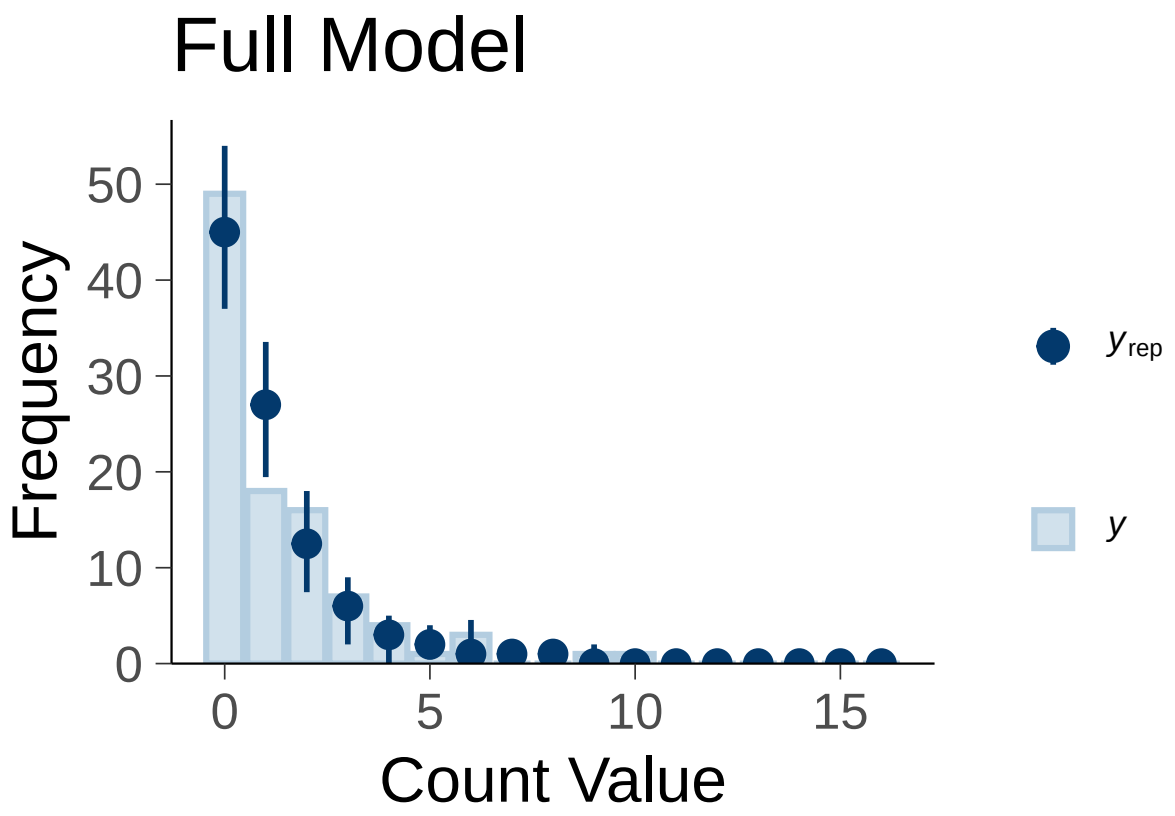

```
ppd_intervals(t(t(y_rep_full) - y), x = y) +
  geom_abline(intercept = 0, slope = 0, linetype = "dashed", color = "red") +
  labs(title = "Full Model",
       y = "Error",
       x = "Observed y")
```



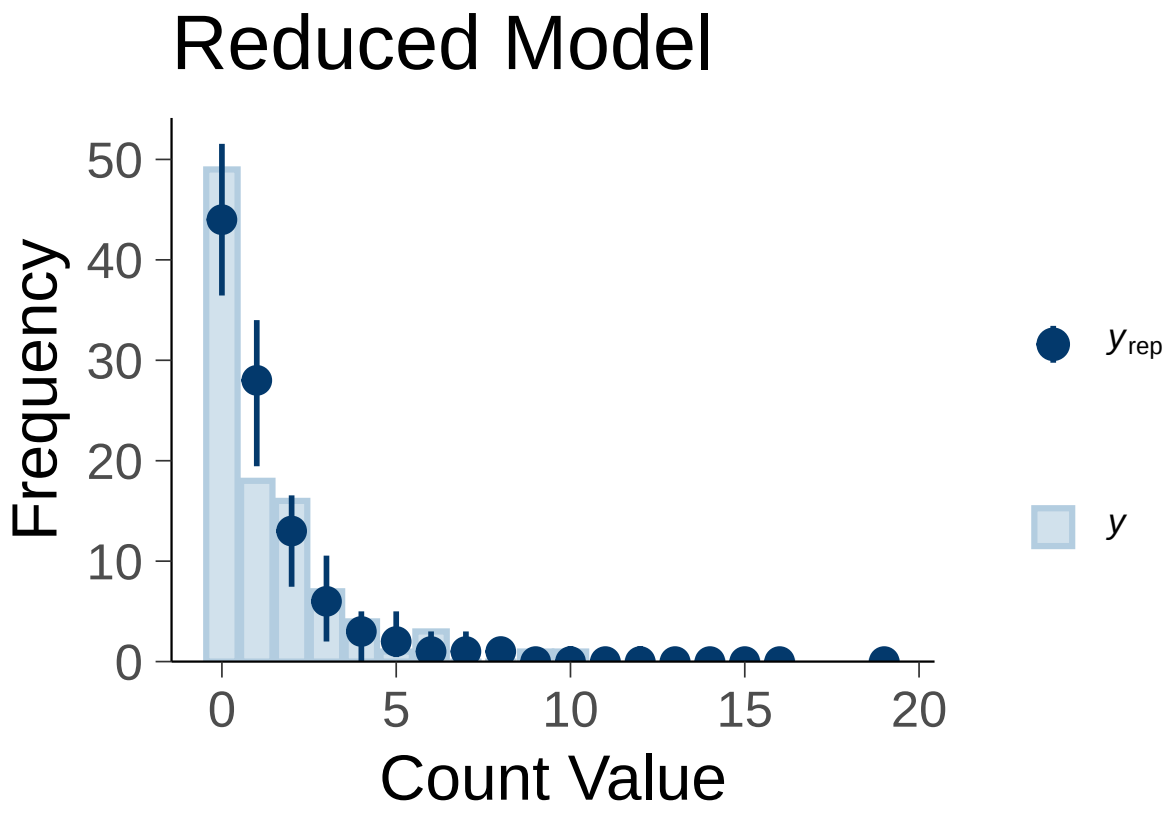

```
ppd_intervals(t(t(y_rep_reduced) - y), x = y) +
  geom_abline(intercept = 0, slope = 0, linetype = "dashed", color = "red") +
  labs(title = "Reduced Model",
       y = "Error",
       x = "Observed y")
```



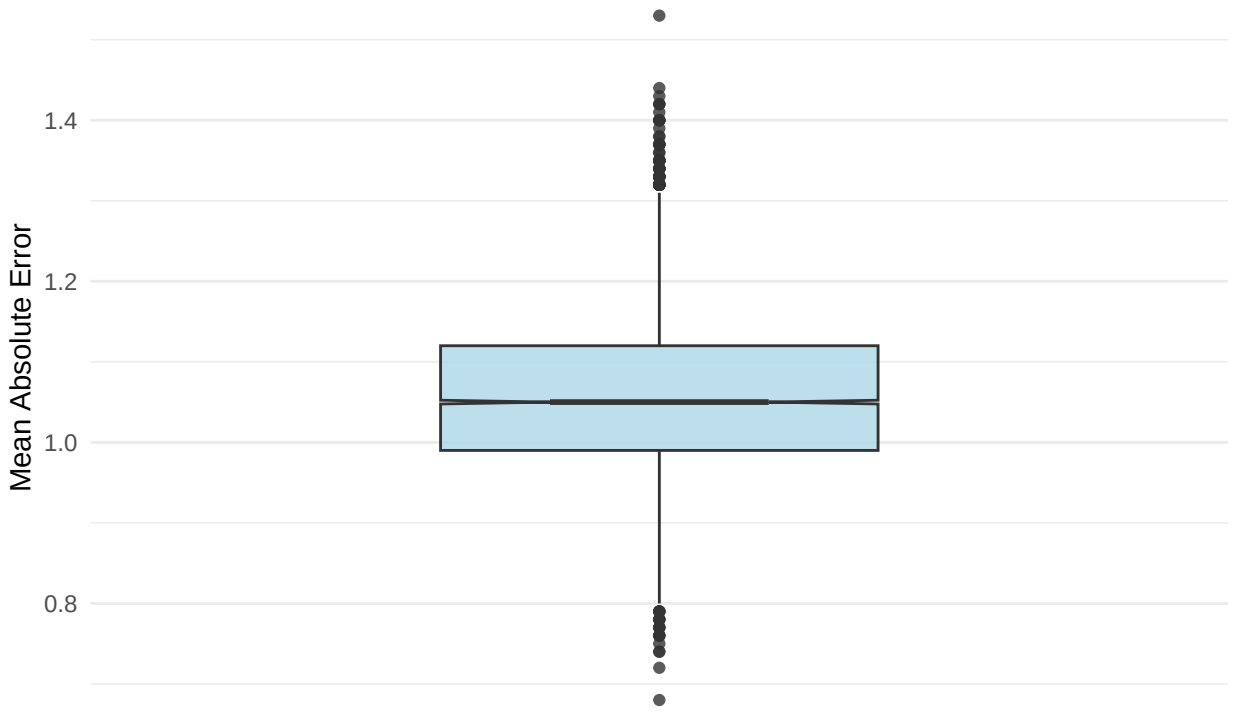

```
ppc_bars(y, y_rep_full[1:50, ]) +
  labs(title = "Full Model",
       x = "Count Value",
       y = "Frequency")
```



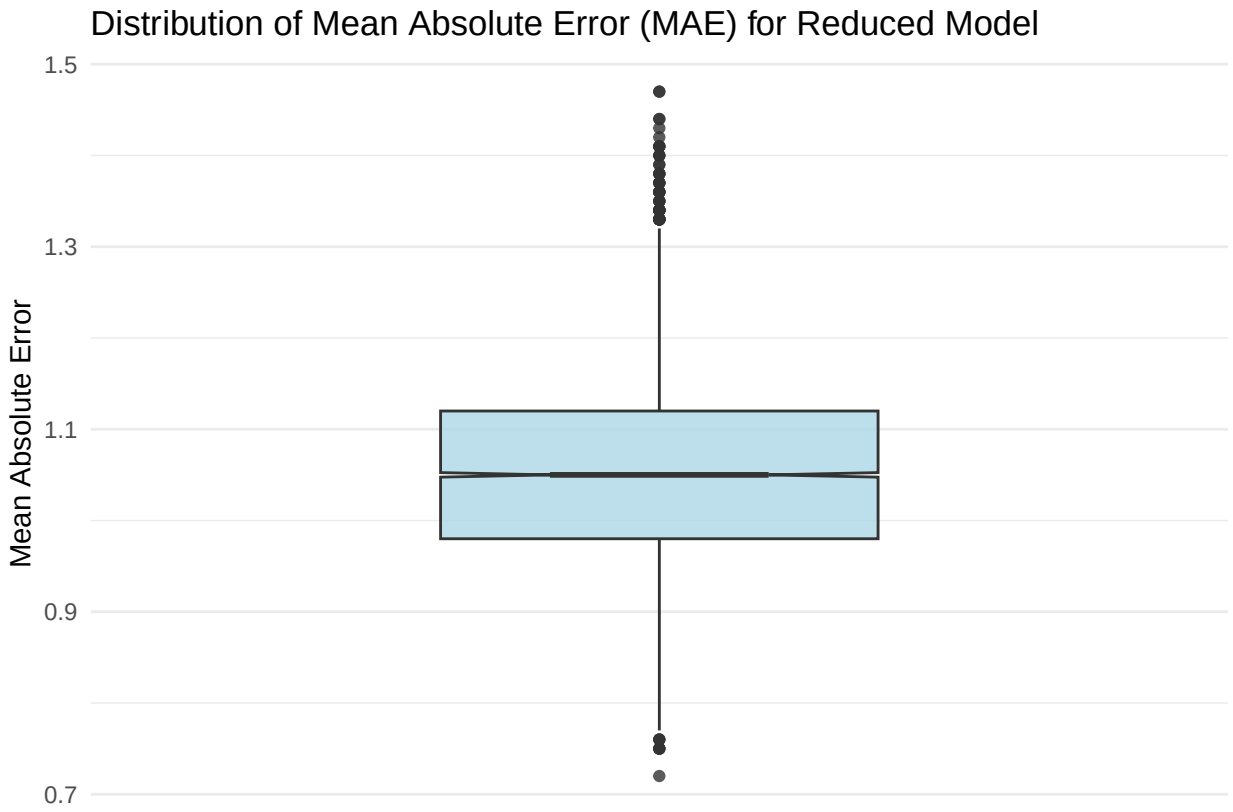

```
ppc_bars(y, y_rep_reduced[1:50, ]) +
  labs(title = "Reduced Model",
       x = "Count Value",
       y = "Frequency")
```




```r
error_matrix_full <- t(t(y_rep_full) - y)
mae_distribution_full <- apply(abs(error_matrix_full), 1, mean)
mae_df_full <- data.frame(mae = mae_distribution_full)

error_matrix_reduced <- t(t(y_rep_reduced) - y)
mae_distribution_reduced <- apply(abs(error_matrix_reduced), 1, mean)
mae_df_reduced <- data.frame(mae = mae_distribution_reduced)

ggplot(mae_df_full, aes(y = mae)) +
  geom_boxplot(
    fill = "lightblue",
    alpha = 0.8,
    notch = TRUE
  ) +
  scale_x_discrete(name = "") +
  labs(
    title = "Distribution of Mean Absolute Error (MAE) for Full Model",
    y = "Mean Absolute Error"
  ) +
  theme_minimal()
```



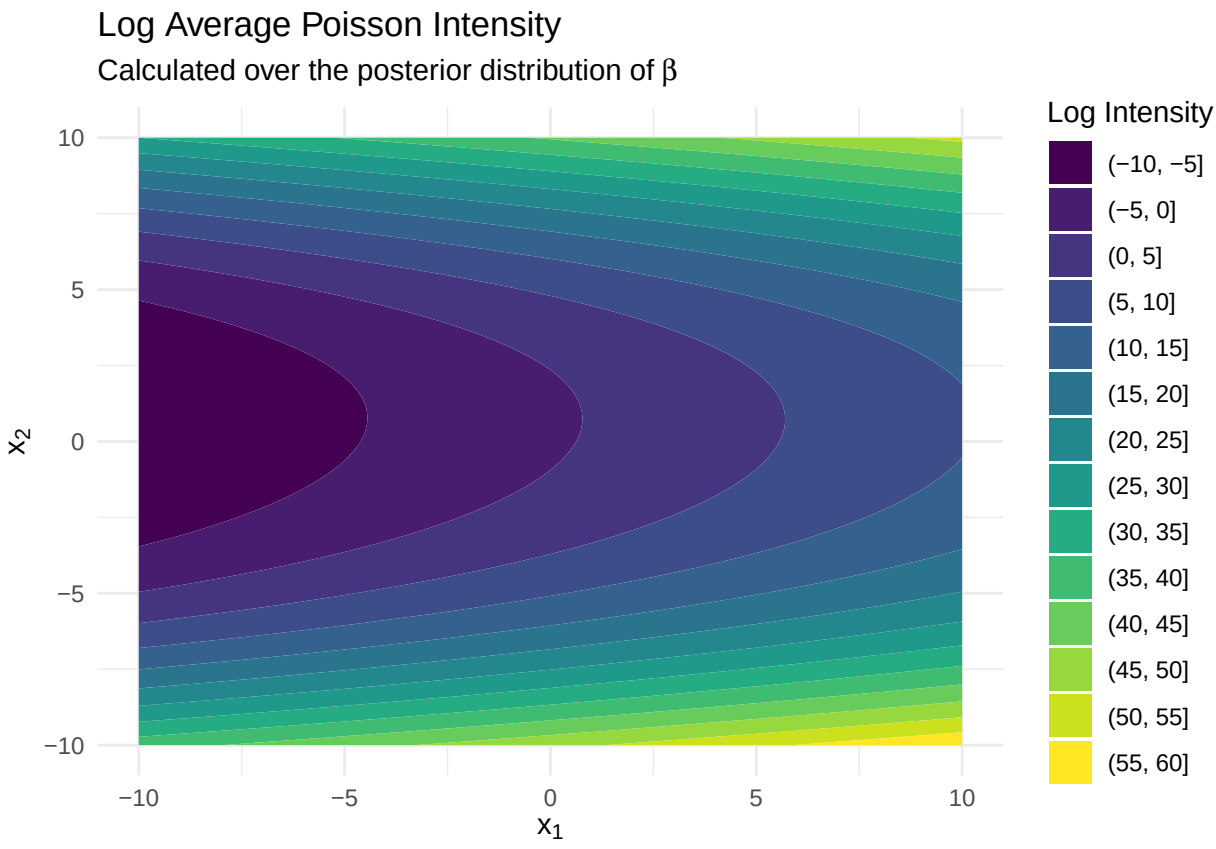

```
ggplot(mae_df_reduced, aes(y = mae)) +
  geom_boxplot(
    fill = "lightblue",
    alpha = 0.8,
    notch = TRUE
  ) +
  scale_x_discrete(name = "") +
  labs(
    title = "Distribution of Mean Absolute Error (MAE) for Reduced Model",
    y = "Mean Absolute Error"
  ) +
  theme_minimal()
```



4.) Use the results from the second model to create a contour plot showing the log average Poisson intensity as a function of x. In other words, plot $\log \mathbb{E}_{p(\beta|x,y)}[\exp(f(x,\beta))]$ as a function of the two components of x (you can restrict the component ranges from -10 to $+10$).

```
x1_grid <- seq(-10, 10, length.out = 500)
x2_grid <- seq(-10, 10, length.out = 500)
grid_to_predict <- expand.grid(x1 = x1_grid, x2 = x2_grid)

beta_samples <- as.matrix(nfit_reduced, pars = c("beta[1]", "beta[2]", "beta[3]", "beta[4]"))
log_intensity_values <- numeric(nrow(grid_to_predict))

for (i in 1:nrow(grid_to_predict)) {
  x1 <- (grid_to_predict$x1[i] - mean(data_p1$x1))/sd(data_p1$x1)
  x2 <- (grid_to_predict$x2[i] - mean(data_p1$x2))/sd(data_p1$x2)

  X_i <- c(1, x1, x2, x2^2)
  f_xb_samples <- beta_samples %*% X_i
  avg_intensity <- mean(exp(f_xb_samples))
  log_intensity_values[i] <- log(avg_intensity)
}

grid_to_predict$log_intensity <- log_intensity_values

ggplot(grid_to_predict, aes(x = x1, y = x2, z = log_intensity)) +
  geom_contour_filled(binwidth = 5) +
  labs(
```

```
    title = "Log Average Poisson Intensity",
    subtitle = expression("Calculated over the posterior distribution of" ~ beta),
    x = expression(x[1]),
    y = expression(x[2]),
    fill = "Log Intensity"
  ) +
  theme_minimal()
```



Applied Problem

Phase 1

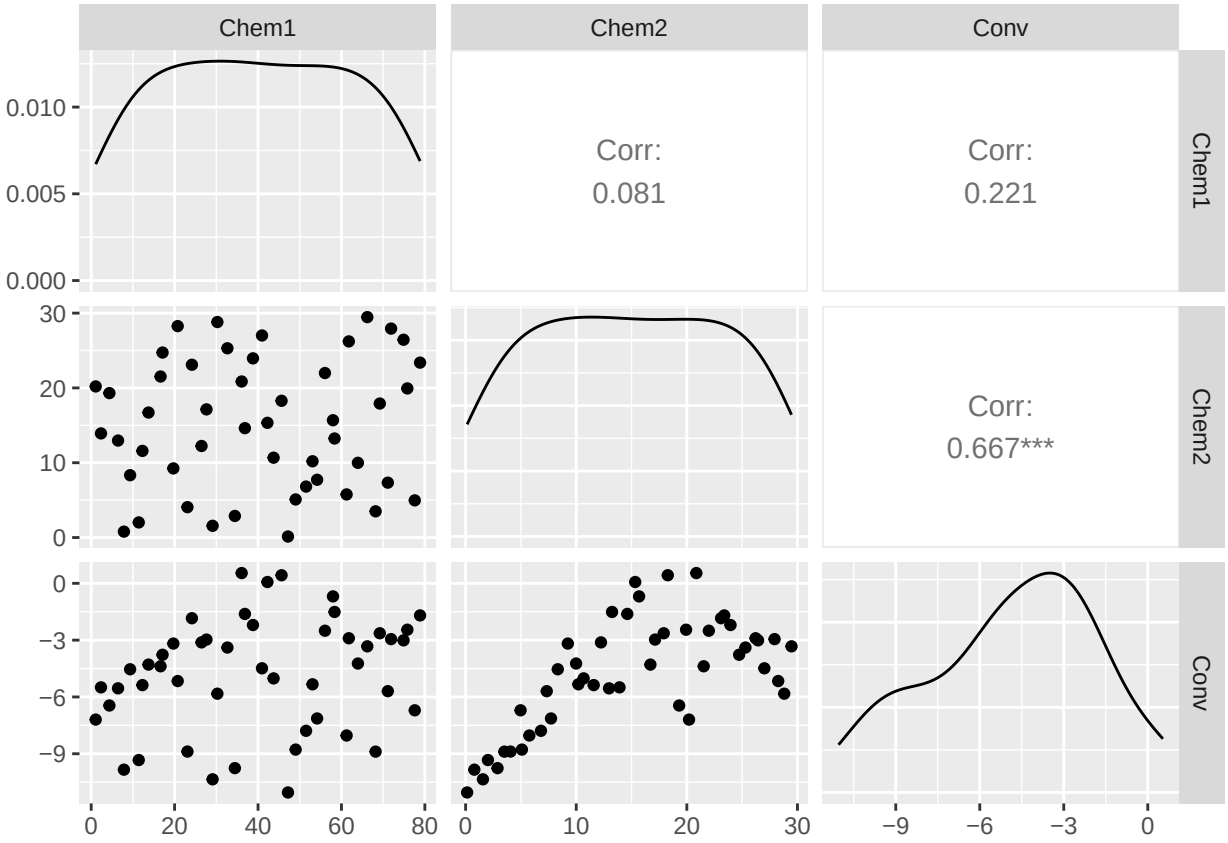
I submitted 48 pairs of concentration for this phase. I chose 48 instead of 24 because I wanted to explore the entire design space with my first model. I chose Latin Hypercube Sampling (LHS) so that the concentration pairs are spread well across the ranges provided for each chemical. The resulting data I received is explored here.

```
design1_data <- read.table(
  "rslt.csv",
  header = FALSE,
  col.names = c("Chem1", "Chem2", "Conv")
)
head(design1_data)
```

```
##       Chem1     Chem2        Conv
## 1 57.970491 15.683497  -0.6911902
## 2 29.111298  1.569691 -10.3494742
```

```
## 3  9.333315  8.333015  -4.5371871
## 4 24.147688 23.096546  -1.8430720
## 5 75.804837 19.931469  -2.4488661
## 6 68.190730  3.505459  -8.8877009
```

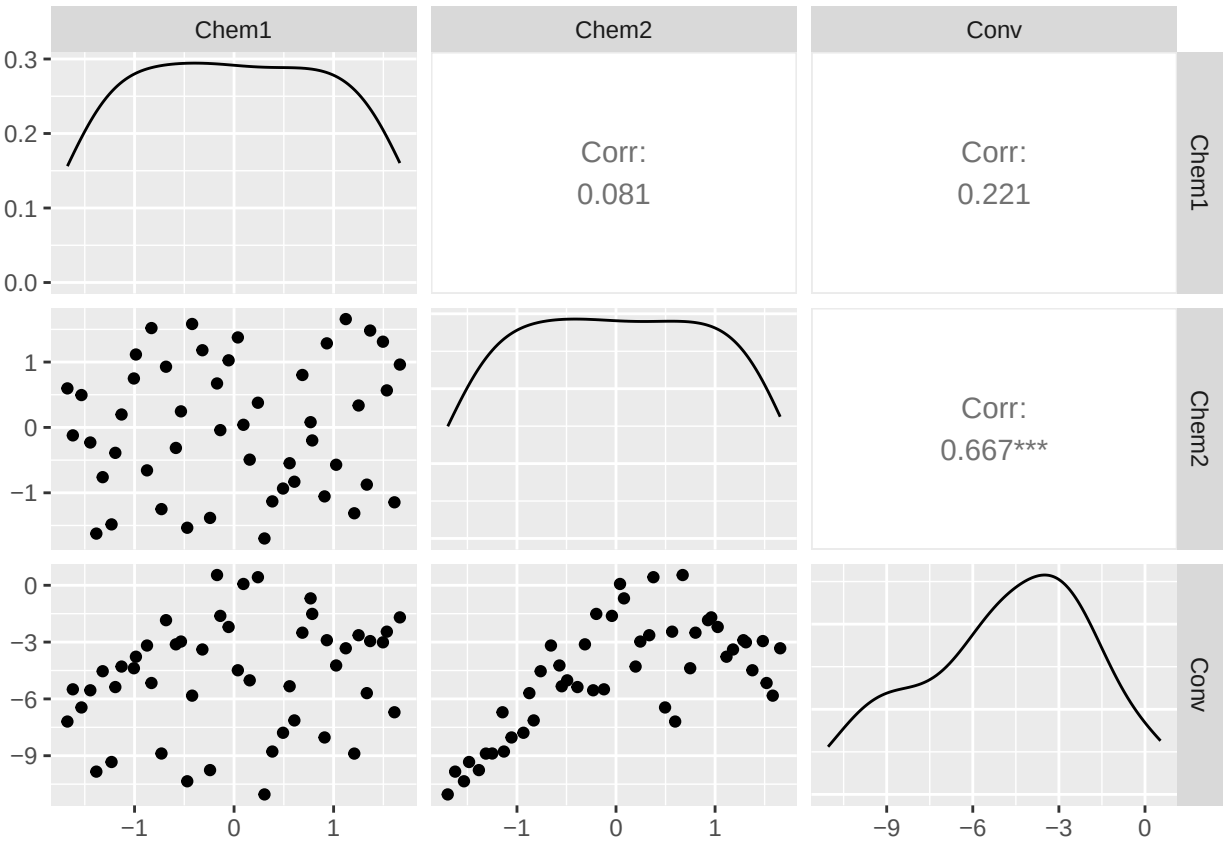
```
ggpairs(design1_data)
```



In this data the first two columns are the feature variables that are independent and we want to predict the third column. Let's standardize the data first.

```
exp1_data <- design1_data
exp1_data$Chem1 = (design1_data$Chem1 - mean(design1_data$Chem1))/sd(design1_data$Chem1)

exp1_data$Chem2 = (design1_data$Chem2 - mean(design1_data$Chem2))/sd(design1_data$Chem2)
```

```
ggpairs(exp1_data);
```



We will make the design matrix similar to the previous problem with synthetic data as a baseline.

```
x1 <- exp1_data$Chem1
x2 <- exp1_data$Chem2
# Build design matrix
X <- cbind(
  intercept = 1,
  x1        = x1,
  x2        = x2,
  x1_sq     = x1^2,
  x2_sq     = x2^2,
  x1_x2     = x1 * x2
)
y = design1_data$Conv
head(X)
```

```
##      intercept         x1          x2     x1_sq       x2_sq       x1_x2
## [1,]         1  0.7690319  0.08035687 0.5914100 0.006457226  0.06179699
## [2,]         1 -0.4705489 -1.53481905 0.2214162 2.355669528  0.72220736
## [3,]         1 -1.3200670 -0.76082813 1.7425769 0.578859439  1.00434411
## [4,]         1 -0.6837494  0.92870201 0.4675132 0.862487420 -0.63499944
## [5,]         1  1.5350655  0.56649237 2.3564262 0.320913603  0.86960291
## [6,]         1  1.2080190 -1.31329081 1.4593098 1.724732762 -1.58648019
```

Assuming Gaussian zero mean Gaussian noise in the model, we will create a Stan model as before but replace Poisson distribution with Gaussian distribution.

```
linreg_normal_code = "
// Data are things you observe/condition on
```

```
data {
int<lower=0> N; // number of data items
int<lower=0> K; // number of predictors
real<lower=0> pr_sd; // std dev of the prior
matrix[N, K] x; // predictor matrix
real y[N]; // output vector is integer/count-valued
}

// parameters are latent quantities whose conditional/posterior distributions you are interested in
parameters {
vector[K] beta; // coefficients for predictors
real<lower=0> sigma; // error scale
}

// useful to avoid repeating calculations.Stan will return values of these variables for each MCMC samp
transformed parameters {
vector[N] mu = x * beta;
}

// The actual Bayesian model goes here
model {
beta ~ normal(0,pr_sd); // Note: beta is k-dim
sigma ~ inv_gamma(0.001,0.001); // Close to a flat prior
y ~ normal(mu,sigma); // likelihood
}

// useful for posterior predictive checks
generated quantities {
  real y_rep[N];
  y_rep = normal_rng(mu, sigma);
}"
#linreg_normal = stan_model(model_code=linreg_normal_code)
```

```
# Compiling a model can take time. If you plan to use it across R sessions,
# you can save it as below
#saveRDS(linreg_normal, "compiled_linreg_normal.rds")
# Later, you can load it as:
linreg_normal = readRDS("compiled_linreg_normal.rds")
```

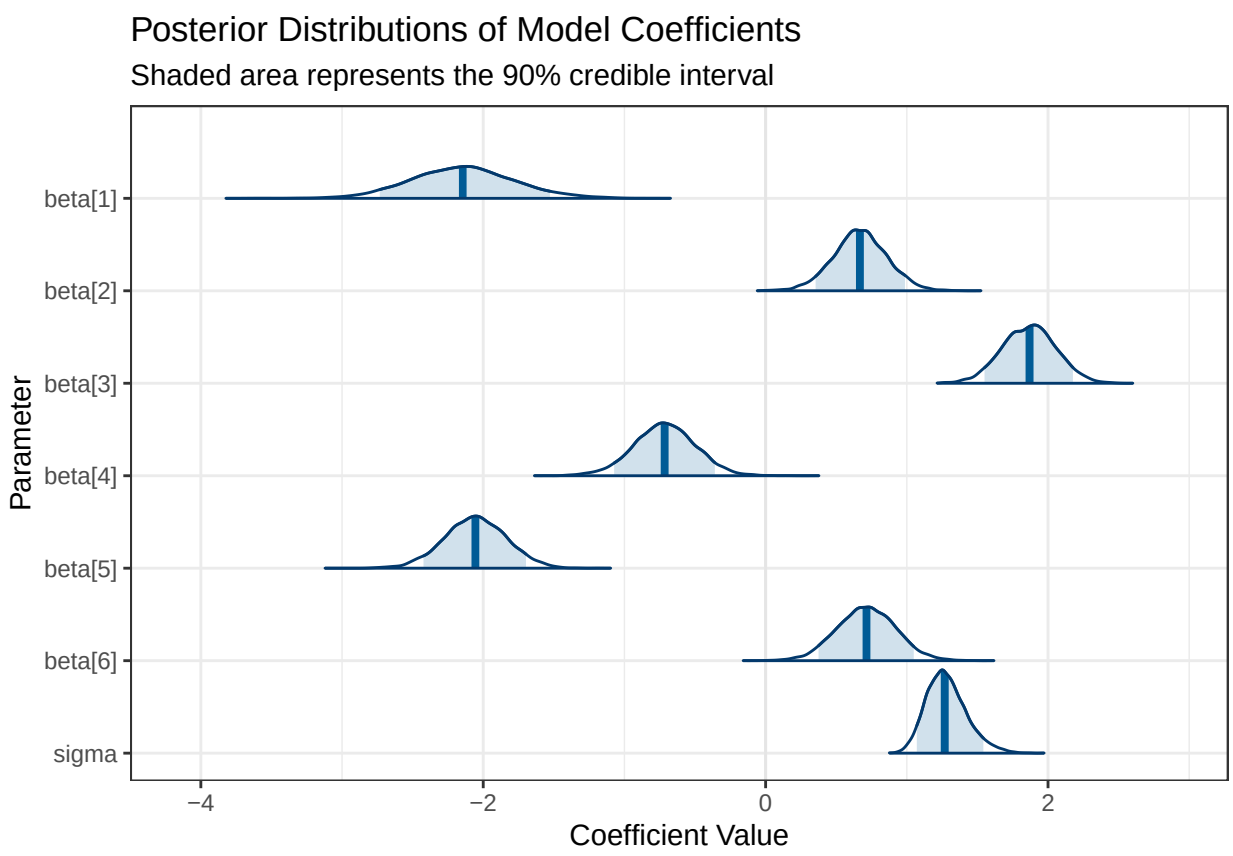
```
reg1_data = list(N = nrow(X), K = ncol(X), pr_sd = 100, x=X, y=y)
nfit = sampling(linreg_normal, data=reg1_data, iter=10000, warmup=2000, chains=1)
```

```
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 3.5e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.35 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 10000 [  0%]  (Warmup)
## Chain 1: Iteration: 1000 / 10000 [ 10%]  (Warmup)
## Chain 1: Iteration: 2000 / 10000 [ 20%]  (Warmup)
## Chain 1: Iteration: 2001 / 10000 [ 20%]  (Sampling)
```

```
## Chain 1: Iteration: 3000 / 10000 [ 30%]  (Sampling)
## Chain 1: Iteration: 4000 / 10000 [ 40%]  (Sampling)
## Chain 1: Iteration: 5000 / 10000 [ 50%]  (Sampling)
## Chain 1: Iteration: 6000 / 10000 [ 60%]  (Sampling)
## Chain 1: Iteration: 7000 / 10000 [ 70%]  (Sampling)
## Chain 1: Iteration: 8000 / 10000 [ 80%]  (Sampling)
## Chain 1: Iteration: 9000 / 10000 [ 90%]  (Sampling)
## Chain 1: Iteration: 10000 / 10000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.064 seconds (Warm-up)
## Chain 1:                0.254 seconds (Sampling)
## Chain 1:                0.318 seconds (Total)
## Chain 1:
```

```r
post_smp <- as.data.frame(nfit)
#colnames(post_smp)[1:6] <- colnames(X)
#head(post_smp)
```

```r
p <- mcmc_areas(post_smp[,1:7], prob = 0.9)
p +
  labs(
    title = "Posterior Distributions of Model Coefficients",
    subtitle = "Shaded area represents the 90% credible interval",
    x = "Coefficient Value",
    y = "Parameter"
  ) +
  theme_bw()
```



```
y_rep = extract(nfit)$y_rep
sg = post_smp$sigma

# Print the root mean-squared error (RMS) and sum of absolute errors
sqrt(sum((colMeans(y_rep)-y)^2))
```

```
## [1] 8.176827
```

```
sum(abs(colMeans(y_rep)-y))
```

```
## [1] 48.07117
```

We need to reduce the above error in our model by introducing higher order polynomial terms to the design matrix. Let's introduce higher order terms in the design matrix and re-run the above code.

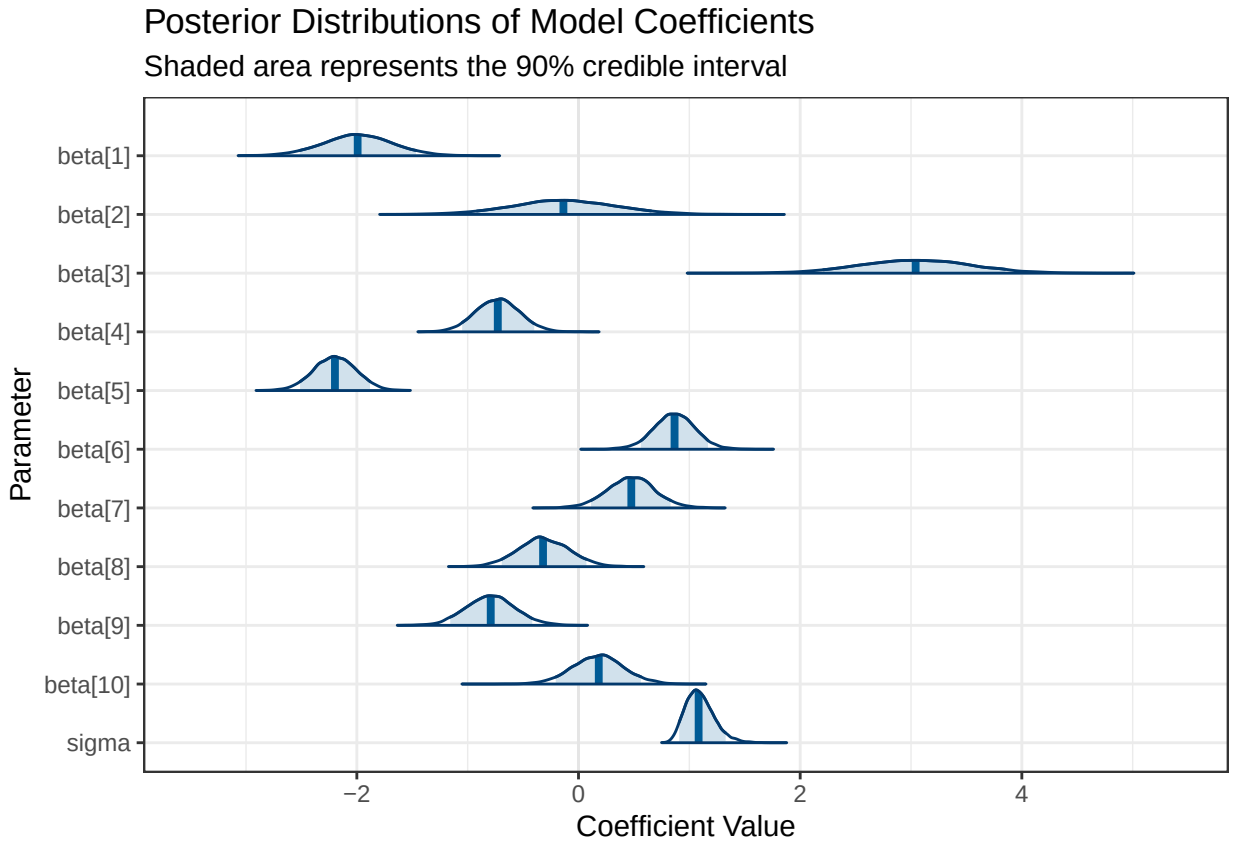
```
# Build design matrix
X <- cbind(
  intercept = 1,
  x1        = x1,
  x2        = x2,
  x1_sq     = x1^2,
  x2_sq     = x2^2,
  x1_x2     = x1 * x2,
  x1_cu     = x1^3,
  x2_cu     = x2^3,
  x1_sq_x2   = (x1^2) * x2,
  x1_x2_sq   = x1 * (x2^2)
)
#head(X)
```

```
reg1_data = list(N = nrow(X), K = ncol(X), pr_sd = 100, x=X, y=y)
nfit = sampling(linreg_normal, data=reg1_data, iter=10000, warmup=2000, chains=1)
```

```
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 7e-06 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.07 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 10000 [  0%]  (Warmup)
## Chain 1: Iteration: 1000 / 10000 [ 10%]  (Warmup)
## Chain 1: Iteration: 2000 / 10000 [ 20%]  (Warmup)
## Chain 1: Iteration: 2001 / 10000 [ 20%]  (Sampling)
## Chain 1: Iteration: 3000 / 10000 [ 30%]  (Sampling)
## Chain 1: Iteration: 4000 / 10000 [ 40%]  (Sampling)
## Chain 1: Iteration: 5000 / 10000 [ 50%]  (Sampling)
## Chain 1: Iteration: 6000 / 10000 [ 60%]  (Sampling)
## Chain 1: Iteration: 7000 / 10000 [ 70%]  (Sampling)
## Chain 1: Iteration: 8000 / 10000 [ 80%]  (Sampling)
## Chain 1: Iteration: 9000 / 10000 [ 90%]  (Sampling)
## Chain 1: Iteration: 10000 / 10000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.12 seconds (Warm-up)
## Chain 1:                0.488 seconds (Sampling)
```

```
## Chain 1:                0.608 seconds (Total)
## Chain 1:
```

```
post_smp <- as.data.frame(nfit)
p <- mcmc_areas(post_smp[,1:11], prob = 0.9)
p +
  labs(
    title = "Posterior Distributions of Model Coefficients",
    subtitle = "Shaded area represents the 90% credible interval",
    x = "Coefficient Value",
    y = "Parameter"
  ) +
  theme_bw()
```




```
y_rep = extract(nfit)$y_rep
sg = post_smp$sigma

# Print the root mean-squared error (RMS) and sum of absolute errors
sqrt(sum((colMeans(y_rep)-y)^2))
```

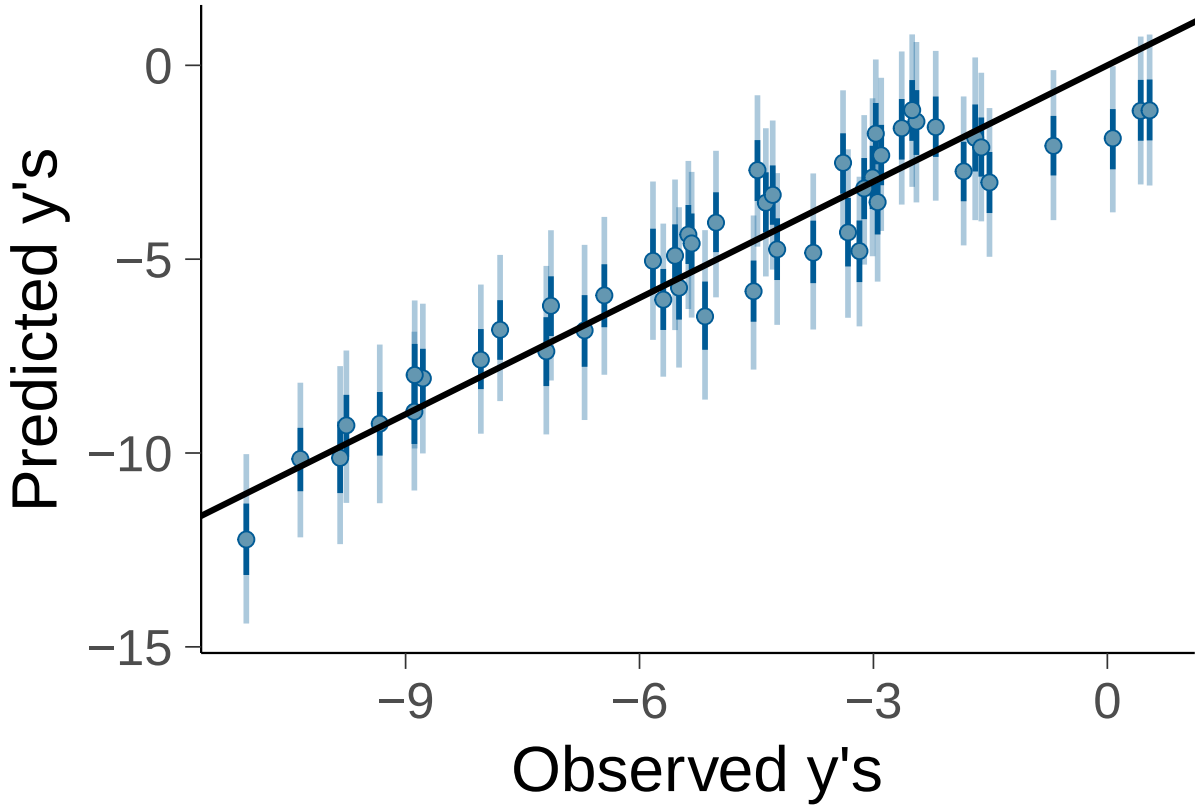
```
## [1] 6.615538
```

```
sum(abs(colMeans(y_rep)-y))
```

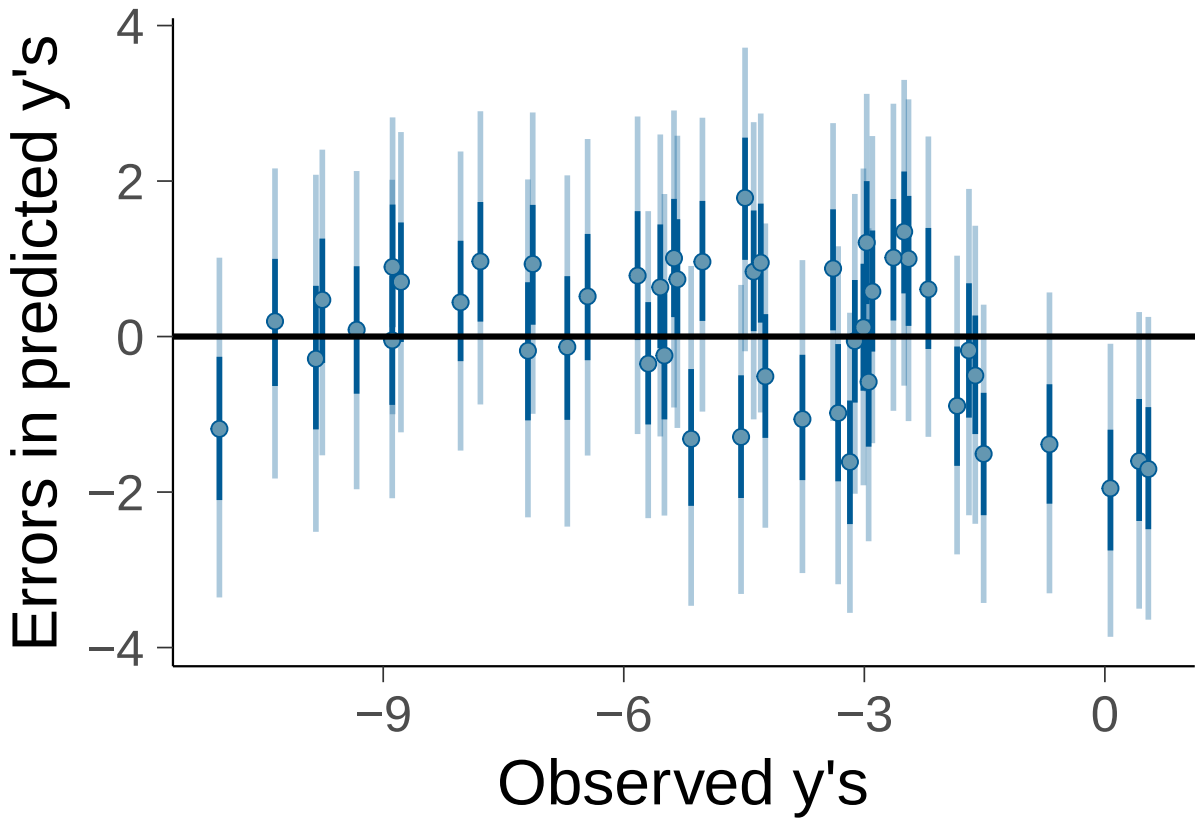
```
## [1] 39.17081
```

We got a reduction in error by introducing higher order terms and their coefficients are not zero.

```
ppd_intervals(y=y_rep,x=y) + geom_abline(intercept = 0, slope = 1) +
ggplot2::labs(y = "Predicted y's", x = "Observed y's");
```



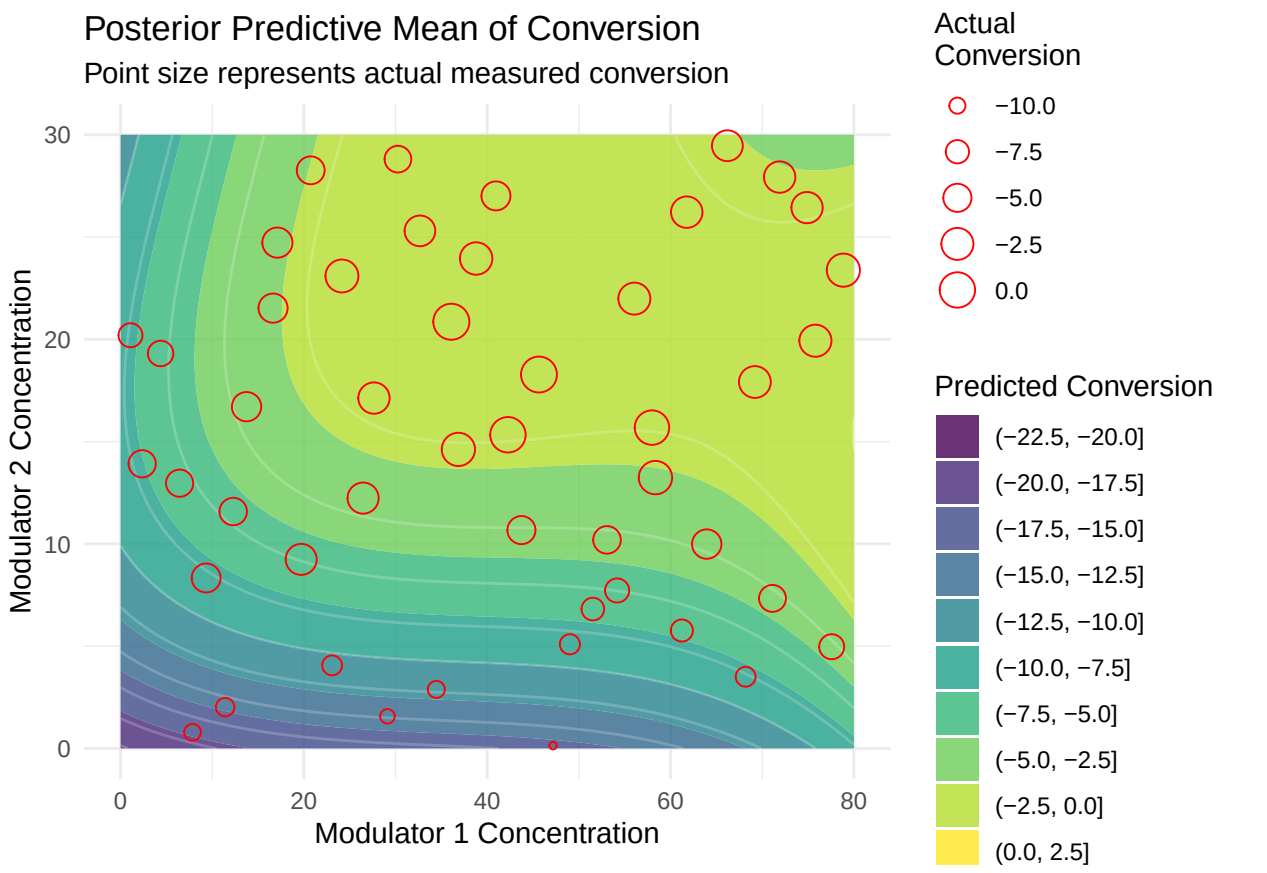

```
ppd_intervals(t(t(y_rep)-y),x=y) + geom_abline(intercept = 0, slope = 0) +
ggplot2::labs(y = "Errors in predicted y's", x = "Observed y's")
```




```
mean_betas <- colMeans(post_smp[, grep("beta", colnames(post_smp))])

grid_orig <- expand.grid(
  x1 = seq(0, 80, length.out = 500),
  x2 = seq(0, 30, length.out = 500)
)

grid_scaled <- grid_orig
grid_scaled$C1 <- (grid_orig$x1 - mean(design1_data$Chem1)) / sd(design1_data$Chem1)
grid_scaled$C2 <- (grid_orig$x2 - mean(design1_data$Chem2)) / sd(design1_data$Chem2)

grid_matrix <- model.matrix(
  ~ C1 + C2 + I(C1^2) + I(C2^2) + C1:C2 + I(C1^3) + I(C2^3) + I(C1^2):C2 + C1:I(C2^2),
  data = grid_scaled
)

predicted_mean <- grid_matrix %*% mean_betas
grid_orig$predicted_mean <- predicted_mean
```

```
ggplot(grid_orig, aes(x = x1, y = x2, z = predicted_mean)) +
  geom_contour_filled(alpha = 0.8, binwidth = 2.5) +
  geom_contour(color = "white", alpha = 0.2) +

  geom_point(data = design1_data,
             aes(x = Chem1, y = Chem2, size = Conv),
             inherit.aes = FALSE,
             shape = 1, color = "red") +
```

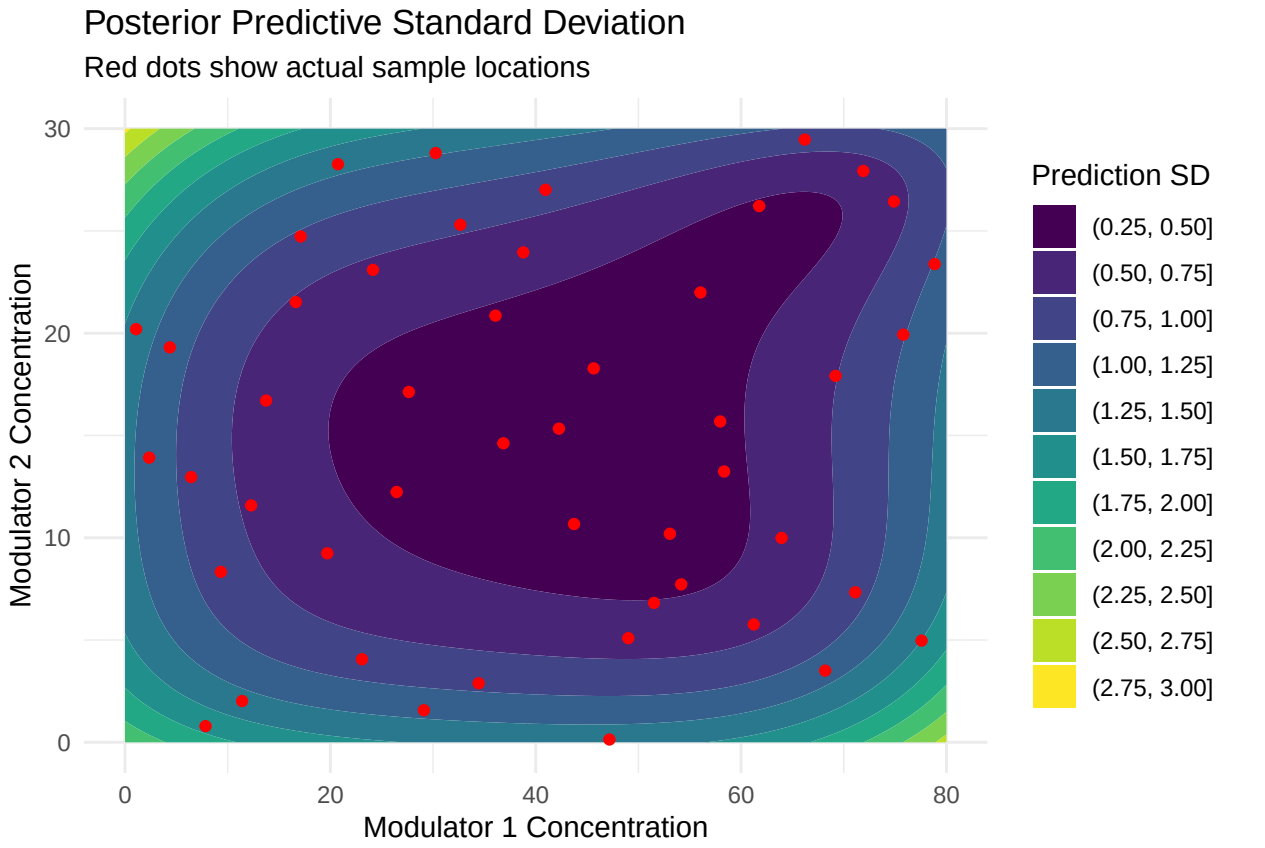
```
  labs(
    title = "Posterior Predictive Mean of Conversion",
    subtitle = "Point size represents actual measured conversion",
    x = "Modulator 1 Concentration",
    y = "Modulator 2 Concentration",
    fill = "Predicted Conversion",
    size = "Actual\nConversion"
  ) +
  theme_minimal()
```



From the posterior mean plot, we see that the highest mean conversion happens for Chemical 1 Concentration in [20, 80] and Chemical 2 Concentration in [15,30].

```
all_betas <- post_smp[, grep("beta", colnames(post_smp))]
all_predictions <- grid_matrix %*% t(all_betas)

# calculate the standard deviation for each row
grid_orig$prediction_sd <- apply(all_predictions, 1, sd)

# Create a contour plot of the uncertainty
ggplot(grid_orig, aes(x = x1, y = x2, z = prediction_sd)) +
  geom_contour_filled(binwidth = 0.25) +

  geom_point(data = design1_data,
             aes(x = Chem1, y = Chem2),
             inherit.aes = FALSE,
             color = "red") +
```

```
  labs(
    title = "Posterior Predictive Standard Deviation",
    subtitle = "Red dots show actual sample locations",
    x = "Modulator 1 Concentration",
    y = "Modulator 2 Concentration",
    fill = "Prediction SD"
  ) +
  theme_minimal()
```

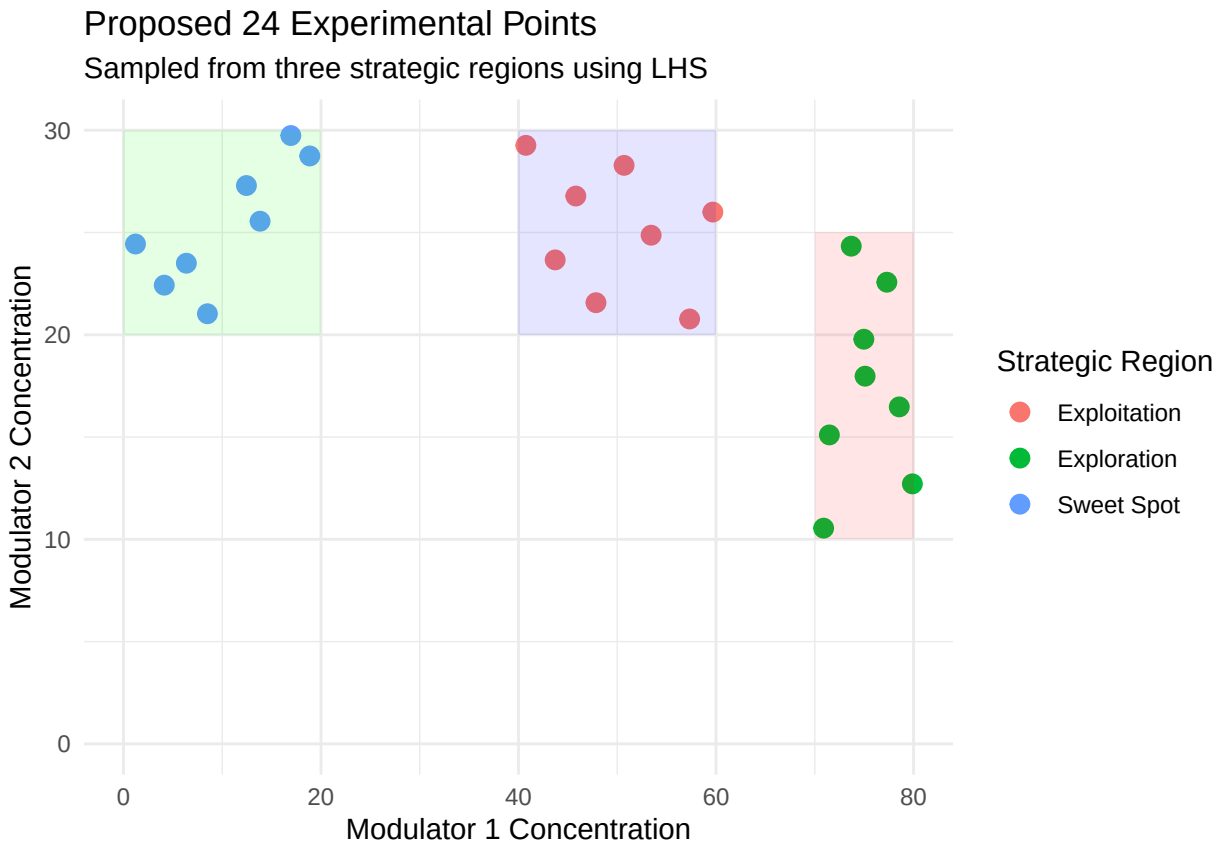


From the standard deviation plot we see we have high variance in the region [70,80] for Chemical 1 and [10,20] for Chemical 2. The second design should contain points from these regions. Here's how I created the 2nd design space.

```
library(lhs)
library(ggplot2)

# Set a seed for reproducibility
set.seed(456)

# --- Region 1: Pure Exploration (High Uncertainty) ---
n1 <- 8
region1_lhs <- randomLHS(n = n1, k = 2)
region1_points <- data.frame(
  mod1 = qunif(region1_lhs[,1], min = 70, max = 80),
  mod2 = qunif(region1_lhs[,2], min = 10, max = 25),
  region = "Exploration"
)
```

```r
# --- Region 2: Pure Exploitation (High Mean) ---
n2 <- 8
region2_lhs <- randomLHS(n = n2, k = 2)
region2_points <- data.frame(
  mod1 = qunif(region2_lhs[,1], min = 40, max = 60),
  mod2 = qunif(region2_lhs[,2], min = 20, max = 30),
  region = "Exploitation"
)

# --- Region 3: The "Sweet Spot" (High Mean, High Uncertainty) ---
n3 <- 8
region3_lhs <- randomLHS(n = n3, k = 2)
region3_points <- data.frame(
  mod1 = qunif(region3_lhs[,1], min = 0, max = 20),
  mod2 = qunif(region3_lhs[,2], min = 20, max = 30),
  region = "Sweet Spot"
)

# --- Combine into a single data frame ---
next_experiments <- rbind(region1_points, region2_points, region3_points)
```

```r
ggplot(next_experiments, aes(x = mod1, y = mod2, color = region)) +
  geom_point(size = 3) +
  # Add boxes to show the sampling regions
  annotate("rect", xmin=70, xmax=80, ymin=10, ymax=25, alpha=0.1, fill="red") +
  annotate("rect", xmin=40, xmax=60, ymin=20, ymax=30, alpha=0.1, fill="blue") +
  annotate("rect", xmin=0, xmax=20, ymin=20, ymax=30, alpha=0.1, fill="green") +
  # Set limits to the full experimental space
  xlim(0, 80) +
  ylim(0, 30) +
  labs(
    title = "Proposed 24 Experimental Points",
    subtitle = "Sampled from three strategic regions using LHS",
    x = "Modulator 1 Concentration",
    y = "Modulator 2 Concentration",
    color = "Strategic Region"
  ) +
  theme_minimal()
```

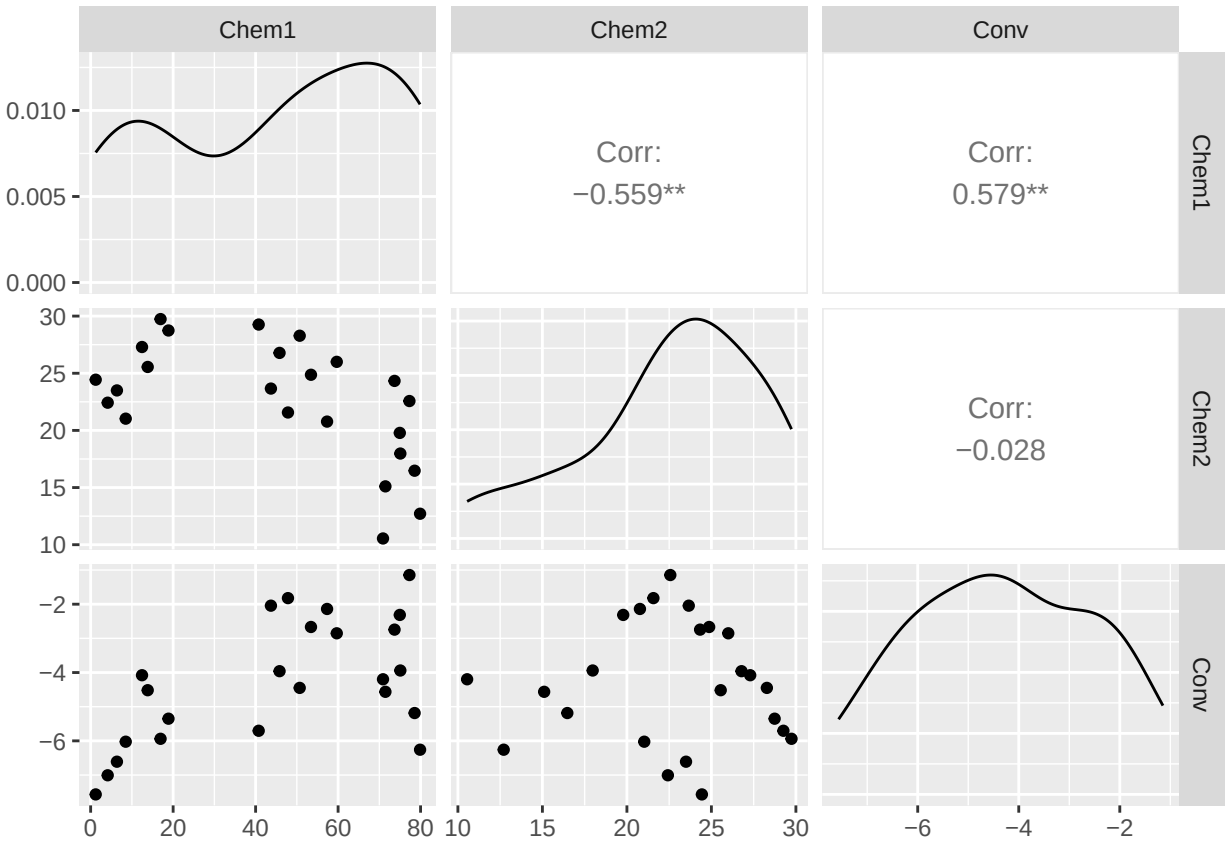


Phase 2

```
design2_data <- read.table(
  "rslt2.csv",
  header = FALSE,
  col.names = c("Chem1", "Chem2", "Conv")
)
head(design2_data)
```

```
##      Chem1    Chem2      Conv
## 1 78.55390 16.47482 -5.185237
## 2 70.89595 10.54604 -4.197242
## 3 71.47485 15.10327 -4.563940
## 4 79.88136 12.71081 -6.258676
## 5 77.29825 22.56905 -1.146171
## 6 73.68866 24.33091 -2.743942
```

```
ggpairs(design2_data)
```

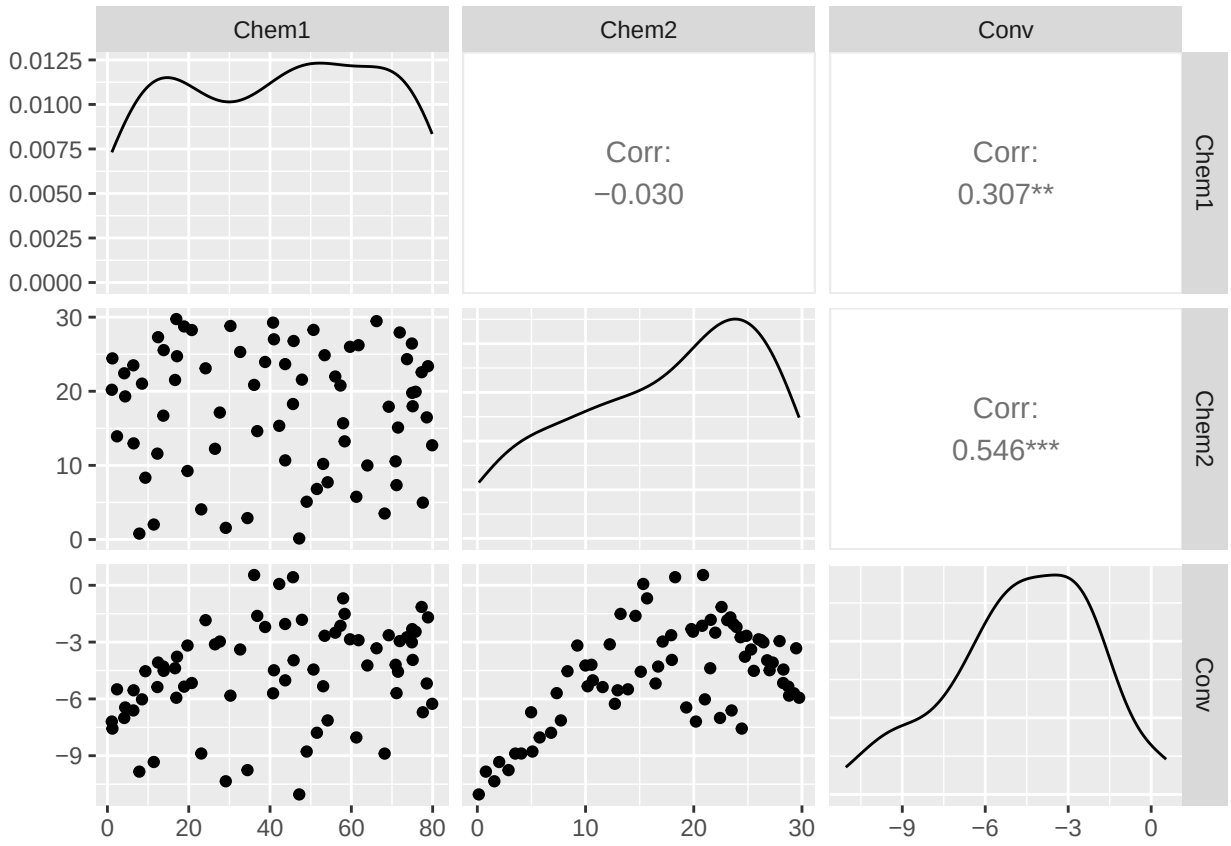


We need to pool the two datasets together and run the analysis again.

```
design_data_72 <- rbind(design1_data, design2_data)
print(dim(design_data_72))
```

```
## [1] 72  3
```

```
ggpairs(design_data_72)
```




```
exp_data <- design_data_72
exp_data$Chem1 = (design_data_72$Chem1 - mean(design_data_72$Chem1))/sd(design_data_72$Chem1)
exp_data$Chem2 = (design_data_72$Chem2 - mean(design_data_72$Chem2))/sd(design_data_72$Chem2)
```

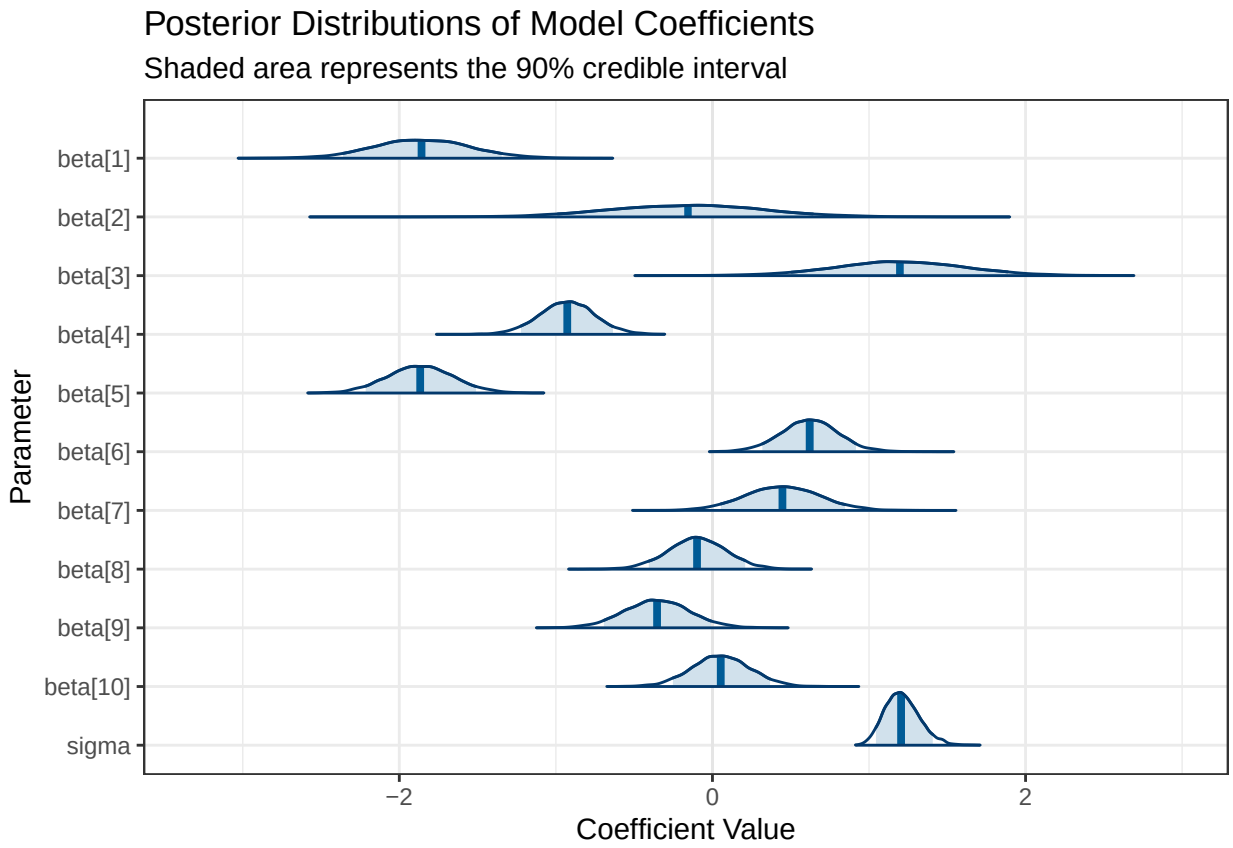
```
x1 <- exp_data$Chem1
x2 <- exp_data$Chem2

# Build design matrix
X <- cbind(
  intercept = 1,
  x1        = x1,
  x2        = x2,
  x1_sq     = x1^2,
  x2_sq     = x2^2,
  x1_x2     = x1 * x2,
  x1_cu     = x1^3,
  x2_cu     = x2^3,
  x1_sq_x2   = (x1^2) * x2,
  x1_x2_sq   = x1 * (x2^2)
)

y = design_data_72$Conv

print(dim(X))
```

```
## [1] 72 10
```

```r
reg1_data = list(N = nrow(X), K = ncol(X), pr_sd = 100, x=X, y=y)
nfit = sampling(linreg_normal, data=reg1_data, iter=10000, warmup=2000, chains=1)
```

```
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0.000403 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 4.03 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 10000 [  0%]  (Warmup)
## Chain 1: Iteration: 1000 / 10000 [ 10%]  (Warmup)
## Chain 1: Iteration: 2000 / 10000 [ 20%]  (Warmup)
## Chain 1: Iteration: 2001 / 10000 [ 20%]  (Sampling)
## Chain 1: Iteration: 3000 / 10000 [ 30%]  (Sampling)
## Chain 1: Iteration: 4000 / 10000 [ 40%]  (Sampling)
## Chain 1: Iteration: 5000 / 10000 [ 50%]  (Sampling)
## Chain 1: Iteration: 6000 / 10000 [ 60%]  (Sampling)
## Chain 1: Iteration: 7000 / 10000 [ 70%]  (Sampling)
## Chain 1: Iteration: 8000 / 10000 [ 80%]  (Sampling)
## Chain 1: Iteration: 9000 / 10000 [ 90%]  (Sampling)
## Chain 1: Iteration: 10000 / 10000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.143 seconds (Warm-up)
## Chain 1:                0.653 seconds (Sampling)
## Chain 1:                0.796 seconds (Total)
## Chain 1:
```

```r
post_smp <- as.data.frame(nfit)
p <- mcmc_areas(post_smp[,1:11], prob = 0.9)
p +
  labs(
    title = "Posterior Distributions of Model Coefficients",
    subtitle = "Shaded area represents the 90% credible interval",
    x = "Coefficient Value",
    y = "Parameter"
  ) +
  theme_bw()
```




```
y_rep = extract(nfit)$y_rep
sg = post_smp$sigma

# Print the root mean-squared error (RMS) and sum of absolute errors
sqrt(sum((colMeans(y_rep)-y)^2))
```

```
## [1] 9.459893
```

```
sum(abs(colMeans(y_rep)-y))
```

```
## [1] 65.97961
```

We will drop model coefficients close to 0.

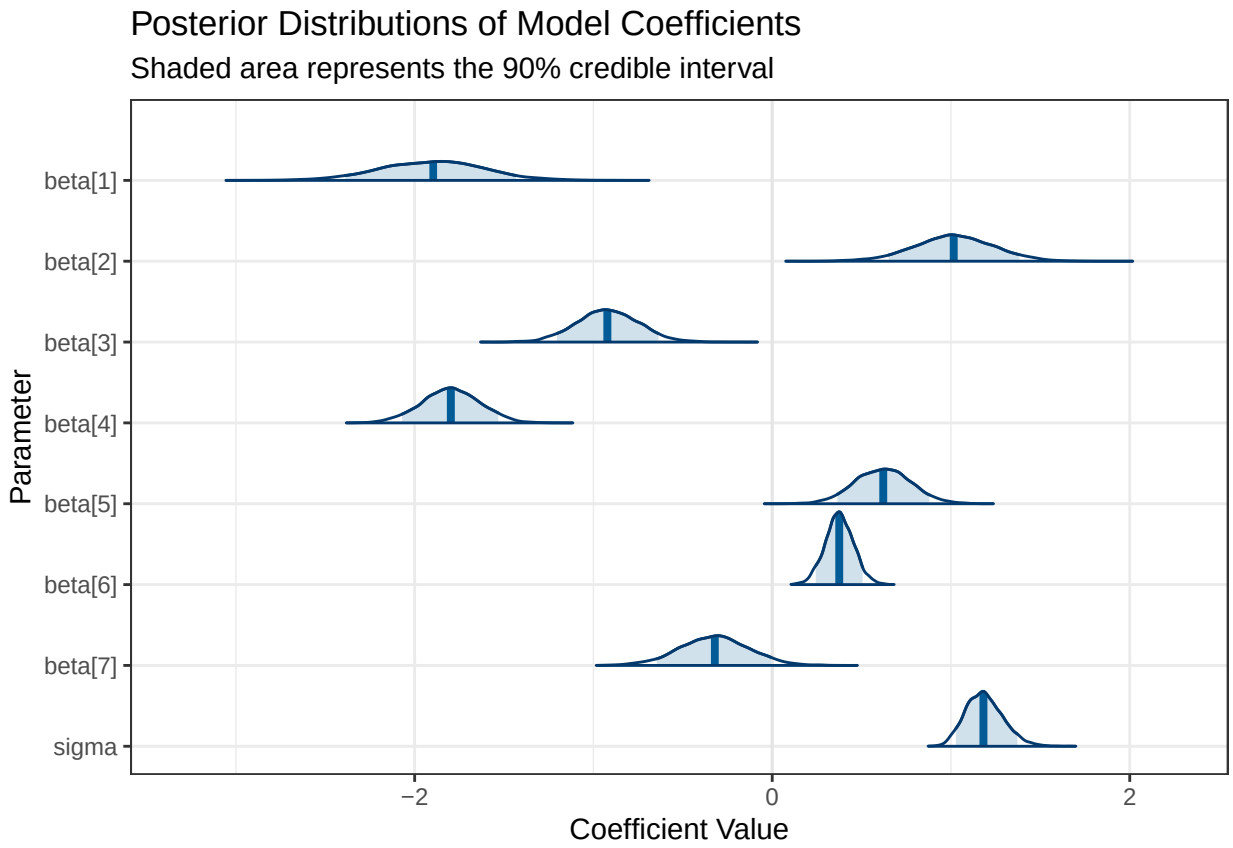
```
# Build design matrix
X <- cbind(
  intercept = 1,
  #x1         = x1,
  x2         = x2,
  x1_sq      = x1^2,
  x2_sq      = x2^2,
  x1_x2      = x1 * x2,
  x1_cu      = x1^3,
  #x2_cu      = x2^3,
  x1_sq_x2   = (x1^2) * x2
  #x1_x2_sq   = x1 * (x2^2)
)
print(dim(X))
```

```
## [1] 72  7
```

```
reg1_data = list(N = nrow(X), K = ncol(X), pr_sd = 100, x=X, y=y)
nfit = sampling(linreg_normal, data=reg1_data, iter=10000, warmup=2000, chains=1)
```

```
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 8e-06 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.08 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 10000 [  0%]  (Warmup)
## Chain 1: Iteration: 1000 / 10000 [ 10%]  (Warmup)
## Chain 1: Iteration: 2000 / 10000 [ 20%]  (Warmup)
## Chain 1: Iteration: 2001 / 10000 [ 20%]  (Sampling)
## Chain 1: Iteration: 3000 / 10000 [ 30%]  (Sampling)
## Chain 1: Iteration: 4000 / 10000 [ 40%]  (Sampling)
## Chain 1: Iteration: 5000 / 10000 [ 50%]  (Sampling)
## Chain 1: Iteration: 6000 / 10000 [ 60%]  (Sampling)
## Chain 1: Iteration: 7000 / 10000 [ 70%]  (Sampling)
## Chain 1: Iteration: 8000 / 10000 [ 80%]  (Sampling)
## Chain 1: Iteration: 9000 / 10000 [ 90%]  (Sampling)
## Chain 1: Iteration: 10000 / 10000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.084 seconds (Warm-up)
## Chain 1:                0.38 seconds (Sampling)
## Chain 1:                0.464 seconds (Total)
## Chain 1:
```

```
post_smp <- as.data.frame(nfit)
p <- mcmc_areas(post_smp[,1:8], prob = 0.9)
p +
  labs(
    title = "Posterior Distributions of Model Coefficients",
    subtitle = "Shaded area represents the 90% credible interval",
    x = "Coefficient Value",
    y = "Parameter"
  ) +
  theme_bw()
```




```
y_rep = extract(nfit)$y_rep
sg = post_smp$sigma

# Print the root mean-squared error (RMS) and sum of absolute errors
sqrt(sum((colMeans(y_rep)-y)^2))
```

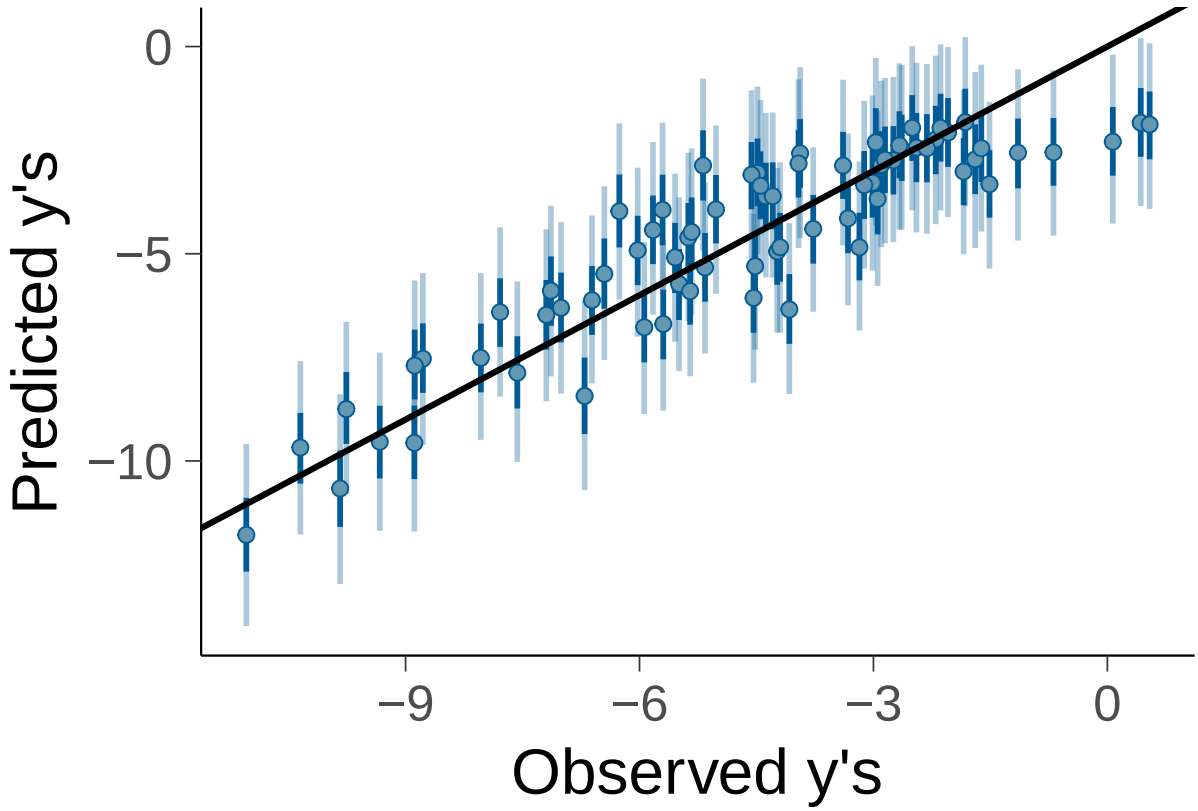
```
## [1] 9.475118
```

```
sum(abs(colMeans(y_rep)-y))
```

```
## [1] 65.71888
```

We do not see much improvement in loss.

```
ppd_intervals(y=y_rep,x=y) + geom_abline(intercept = 0, slope = 1) +
ggplot2::labs(y = "Predicted y's", x = "Observed y's");
```

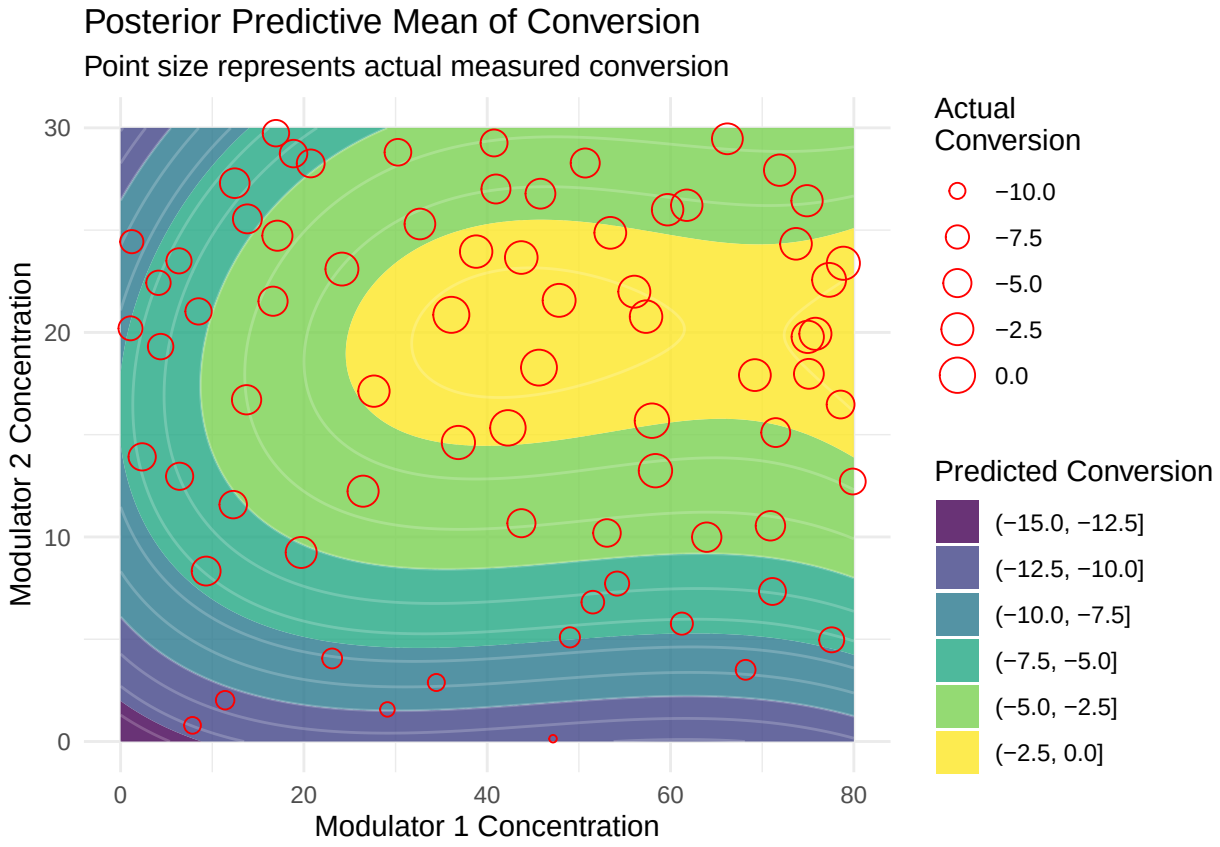



```
mean_betas <- colMeans(post_smp[, grep("beta", colnames(post_smp))])

grid_orig <- expand.grid(
  x1 = seq(0, 80, length.out = 500),
  x2 = seq(0, 30, length.out = 500)
)

grid_scaled <- grid_orig
grid_scaled$C1 <- (grid_orig$x1 - mean(design_data_72$Chem1)) / sd(design_data_72$Chem1)
grid_scaled$C2 <- (grid_orig$x2 - mean(design_data_72$Chem2)) / sd(design_data_72$Chem2)

grid_matrix <- model.matrix(
  ~ C2 + I(C1^2) + I(C2^2) + C1:C2 + I(C1^3) + I(C1^2):C2,
  data = grid_scaled
)

predicted_mean <- grid_matrix %*% mean_betas
grid_orig$predicted_mean <- predicted_mean
```

Let's make a final contour plot of the posterior predictive mean of conversion as a function of concentration settings.

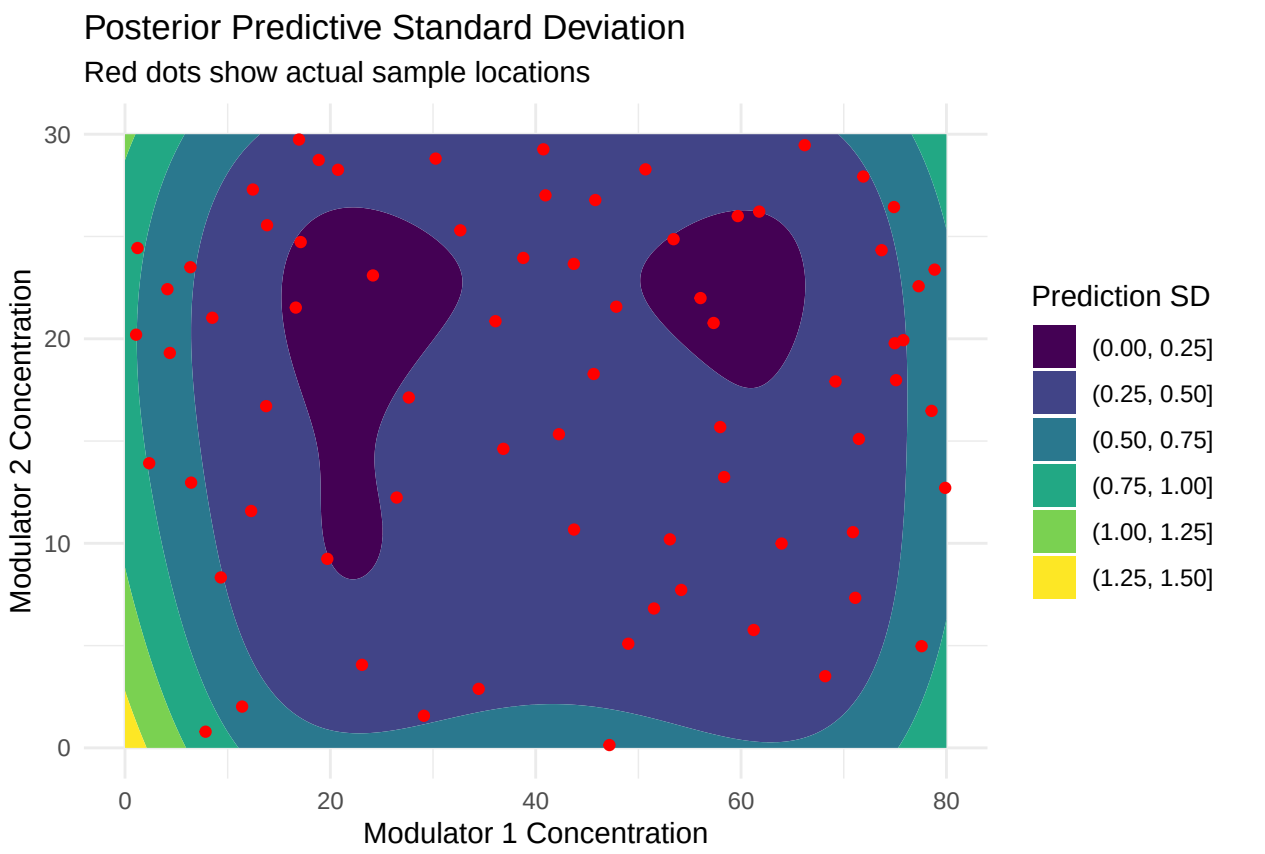
```
ggplot(grid_orig, aes(x = x1, y = x2, z = predicted_mean)) +
  geom_contour_filled(alpha = 0.8, binwidth = 2.5) +
  geom_contour(color = "white", alpha = 0.2) +

  geom_point(data = design_data_72,
             aes(x = Chem1, y = Chem2, size = Conv),
             inherit.aes = FALSE,
```

```
            shape = 1, color = "red") +

  labs(
    title = "Posterior Predictive Mean of Conversion",
    subtitle = "Point size represents actual measured conversion",
    x = "Modulator 1 Concentration",
    y = "Modulator 2 Concentration",
    fill = "Predicted Conversion",
    size = "Actual\nConversion"
  ) +
  theme_minimal()
```




```
all_betas <- post_smp[, grep("beta", colnames(post_smp))]
all_predictions <- grid_matrix %*% t(all_betas)

# calculate the standard deviation for each row
grid_orig$prediction_sd <- apply(all_predictions, 1, sd)

# Create a contour plot of the uncertainty
ggplot(grid_orig, aes(x = x1, y = x2, z = prediction_sd)) +
  geom_contour_filled(binwidth = 0.25) +

  geom_point(data = design_data_72,
             aes(x = Chem1, y = Chem2),
             inherit.aes = FALSE,
             color = "red") +
```

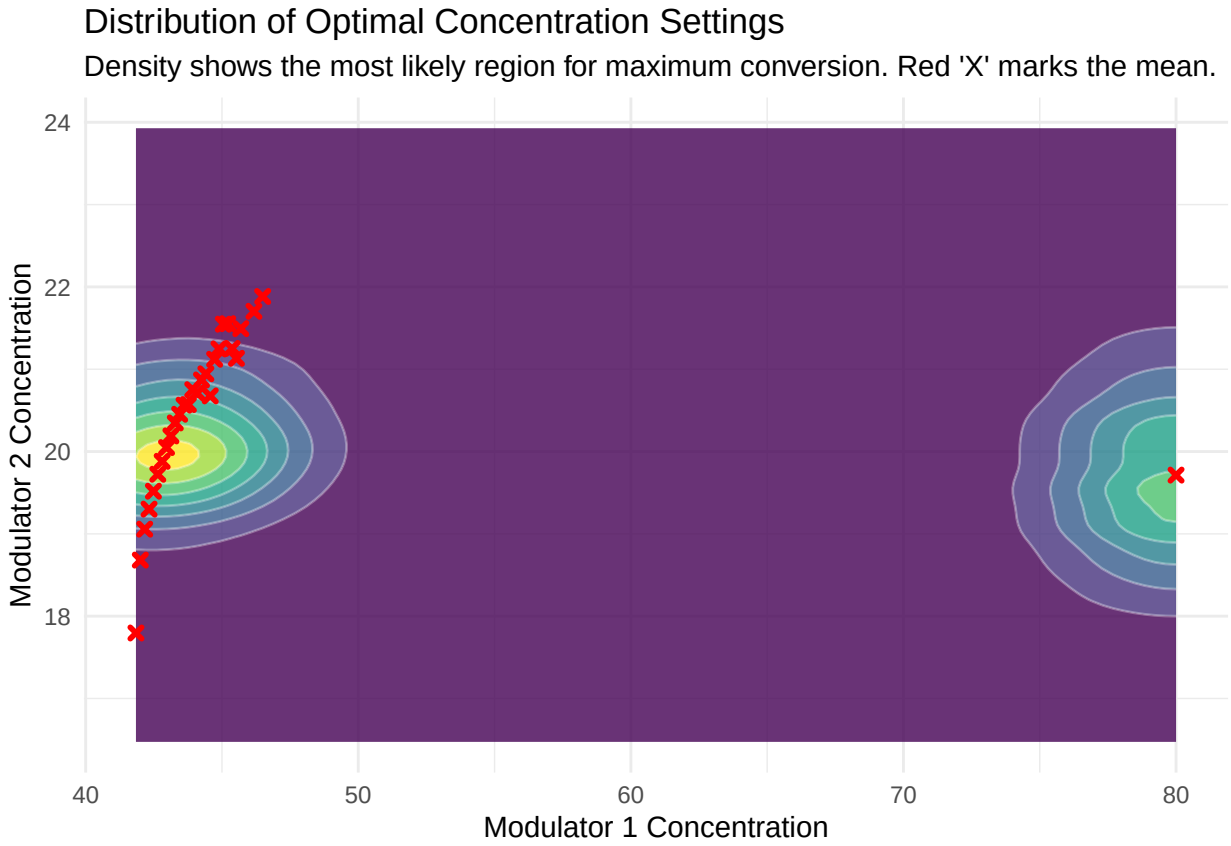
```
  labs(
    title = "Posterior Predictive Standard Deviation",
    subtitle = "Red dots show actual sample locations",
    x = "Modulator 1 Concentration",
    y = "Modulator 2 Concentration",
    fill = "Prediction SD"
  ) +
  theme_minimal()
```



Now, we'll make a contour plot of the posterior predictive distribution of the concentration settings that yield maximum conversion.

```
max_indices <- apply(all_predictions, 2, which.max)
optimal_points <- grid_orig[max_indices, c("x1", "x2")]
```

```
ggplot(optimal_points, aes(x = x1, y = x2)) +

  geom_density_2d_filled(alpha = 0.8, show.legend = FALSE) +
  geom_density_2d(color = "white", alpha = 0.3) +
  stat_summary(fun = "mean", geom = "point", color = "red", size = 1, shape = 4, stroke = 1.5) +

  labs(
    title = "Distribution of Optimal Concentration Settings",
    subtitle = "Density shows the most likely region for maximum conversion. Red 'X' marks the mean.",
    x = "Modulator 1 Concentration",
    y = "Modulator 2 Concentration"
  ) +
  theme_minimal()
```


Distribution of Optimal Concentration Settings
Density shows the most likely region for maximum conversion. Red 'X' marks the mean.
Modulator 2 Concentration
24
22
20
18
40
50
60
70
80
Modulator 1 Concentration