Code ▾

HW3

This is an R Markdown (http://rmarkdown.rstudio.com) Notebook. When you execute code within the notebook, the results appear beneath the code.

Try executing this chunk by clicking the *Run* button within the chunk or by placing your cursor inside it and pressing *Cmd+Shift+Enter*.

1. How do valence, instrumentalness, and dance affect popularity?

Hide

```
library("ggplot2")
library("DescTools")
library(rstan)
rstan_options(auto_write = TRUE)
```

2. I selected rock music and artists with at least 8 songs to ensure at least some data for each artist.

Hide

```
### I like rock music
rock <- spotify_data[which(spotify_data$genre == "rock"),]
counts_artists <- as.data.frame(table(rock$track_artist))
r <- counts_artists[which(counts_artists$Freq>=8),"Var1"]

r = droplevels(r)

rock <- rock[which(rock$track_artist %in% r),]
```

3. Hierarchical model is below, and it’s description

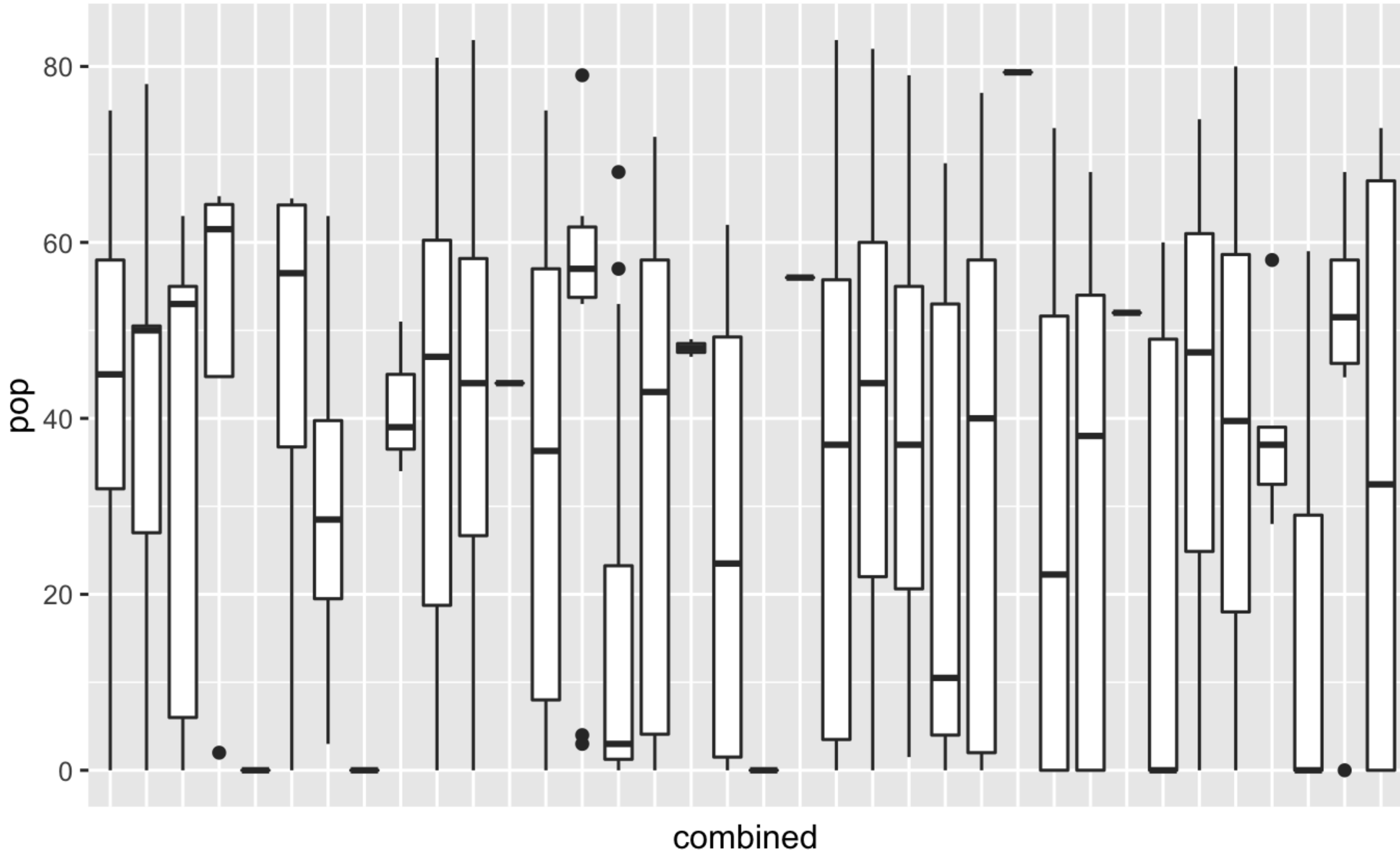
alpha ~ N(40, 10) beta ~ N(0.5, 0.1) gamma ~ N(0.5, 0.1) delta ~ N(0.5, 0.1) y ~ N(40, 20)

Hide

```
valence = as.factor(RoundTo(rock$valence,multiple=0.33))
levels(valence) = c("a","b","c","d")
instrumentalness = as.factor(RoundTo(rock$instrumentalness,multiple=0.33))
levels(instrumentalness) = c("a","b","c","d")
dance = as.factor(RoundTo(rock$danceability,multiple=0.33))
levels(dance) = c("a","b","c","d")
pop = rock$popularity
df_vars = data.frame(pop,valence, instrumentalness,dance)
df_vars$combined = paste(df_vars[,"valence"],df_vars[,"instrumentalness"],df_vars[,"dance"],sep="")
```

Hide

```
par(mar = c(3,3,2,1), mgp = c(1.8, 0.5, 0))
ggplot(df_vars, aes(x = combined, y = pop)) +
  theme(axis.text.x = element_blank(),
        axis.ticks.x = element_blank())+
        #axis.text.y = element_blank())+
        #axis.ticks.y = element_blank())+
  geom_boxplot()
```



4.

Hide

```
thing = sort(unique(df_vars$combined))
for (i in 1:length(thing)){
  thing[i] = paste(thing[i],as.character(i),sep = " : ")
}
vals <- data.frame(matrix(thing,    # Create empty data frame
                                  nrow = 6,
                                  ncol = 6,
                                  byrow = TRUE))
vals
```

X1 <chr>	X2 <chr>	X3 <chr>	X4 <chr>	X5 <chr>	X6 <chr>
aab : 1	aac : 2	abb : 3	abc : 4	acb : 5	acc : 6
adb : 7	adc : 8	baa : 9	bab : 10	bac : 11	bad : 12
bbb : 13	bbc : 14	bcb : 15	bcc : 16	bda : 17	bdb : 18
bdc : 19	caa : 20	cab : 21	cac : 22	cad : 23	cbb : 24
cbc : 25	cbd : 26	ccb : 27	ccc : 28	cdb : 29	cdc : 30
dab : 31	dac : 32	dad : 33	dbc : 34	dcc : 35	ddc : 36

6 rows

Hide

```
### aab is the boxplot on the far left, aac is next,..., ddc is on the far right
```

Hide

```
a <- rock$valence
b <- rock$instrumentalness
c <- rock$danceability
y <- rock$popularity

a_mean <- mean(a)
a_sd <- sd(a)
b_mean <- mean(b)
b_sd <- sd(b)
c_mean <- mean(c)
c_sd <- sd(c)

y_mean <- mean(y)
y_sd <- sd(y)

mu_std_alpha <- 40
sigma_std_alpha <- 10

mu_std_beta <- 0.5 * a_sd/y_sd
sigma_std_beta <- 0.05 * a_sd/y_sd

mu_std_gamma <- 0.5 * b_sd/y_sd
sigma_std_gamma <- 0.05 * b_sd/y_sd

mu_std_delta <- 0.5 * c_sd/y_sd
sigma_std_delta <- 0.05 * c_sd/y_sd

a_grid <- seq(0, 1, by = 0.05)
m_grid <- length(a_grid)
b_grid <- seq(0, 1, by = 0.05)
o_grid <- length(b_grid)
c_grid <- seq(0, 1, by = 0.05)
q_grid <- length(c_grid)
```

Hide

```
prior_pred_model <- stan_model(file = "linear_regression_prior_line.stan")
```

```
'config' variable 'CPP' is deprecated
```

```
clang -mmacosx-version-min=10.13 -E
```

```
Warning in system(paste(CPP, ARGS), ignore.stdout = TRUE, ignore.stderr = TRUE) :
  error in running command
```

Hide

```
rock_data <- list(y_mean = y_mean, y_sd = y_sd,
                  a_mean = a_mean, a_sd = a_sd,
                  m_grid = m_grid, a_grid = a_grid,
                  b_mean = b_mean, b_sd = b_sd,
                  o_grid = o_grid, b_grid = b_grid,
                  c_mean = c_mean, c_sd = c_sd,
                  q_grid = q_grid, c_grid = c_grid,
                  mu_std_alpha = mu_std_alpha, sigma_std_alpha = sigma_std_alpha,
                  mu_std_beta = mu_std_beta, sigma_std_beta = sigma_std_beta,
                  mu_std_gamma = mu_std_gamma, sigma_std_gamma = sigma_std_gamma,
                  mu_std_delta = mu_std_delta, sigma_std_delta = sigma_std_delta)
```

Hide

```
prior_sim <- sampling(object = prior_pred_model, algorithm = "Fixed_param",
                      data = rock_data)
```

```
SAMPLING FOR MODEL 'linear_regression_prior_line' NOW (CHAIN 1).
Chain 1: Iteration:    1 / 2000 [  0%]  (Sampling)
Chain 1: Iteration:  200 / 2000 [ 10%]  (Sampling)
Chain 1: Iteration:  400 / 2000 [ 20%]  (Sampling)
Chain 1: Iteration:  600 / 2000 [ 30%]  (Sampling)
Chain 1: Iteration:  800 / 2000 [ 40%]  (Sampling)
Chain 1: Iteration: 1000 / 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 seconds (Warm-up)
Chain 1:                0.389674 seconds (Sampling)
Chain 1:                0.389674 seconds (Total)
Chain 1: 

SAMPLING FOR MODEL 'linear_regression_prior_line' NOW (CHAIN 2).
Chain 2: Iteration:    1 / 2000 [  0%]  (Sampling)
Chain 2: Iteration:  200 / 2000 [ 10%]  (Sampling)
Chain 2: Iteration:  400 / 2000 [ 20%]  (Sampling)
Chain 2: Iteration:  600 / 2000 [ 30%]  (Sampling)
Chain 2: Iteration:  800 / 2000 [ 40%]  (Sampling)
Chain 2: Iteration: 1000 / 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 seconds (Warm-up)
Chain 2:                0.338878 seconds (Sampling)
Chain 2:                0.338878 seconds (Total)
Chain 2: 

SAMPLING FOR MODEL 'linear_regression_prior_line' NOW (CHAIN 3).
Chain 3: Iteration:    1 / 2000 [  0%]  (Sampling)
```

```
Chain 3: Iteration:  200 / 2000 [ 10%]  (Sampling)
Chain 3: Iteration:  400 / 2000 [ 20%]  (Sampling)
Chain 3: Iteration:  600 / 2000 [ 30%]  (Sampling)
Chain 3: Iteration:  800 / 2000 [ 40%]  (Sampling)
Chain 3: Iteration: 1000 / 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 seconds (Warm-up)
Chain 3:                0.089447 seconds (Sampling)
Chain 3:                0.089447 seconds (Total)
Chain 3: 

SAMPLING FOR MODEL 'linear_regression_prior_line' NOW (CHAIN 4).
Chain 4: Iteration:    1 / 2000 [  0%]  (Sampling)
Chain 4: Iteration:  200 / 2000 [ 10%]  (Sampling)
Chain 4: Iteration:  400 / 2000 [ 20%]  (Sampling)
Chain 4: Iteration:  600 / 2000 [ 30%]  (Sampling)
Chain 4: Iteration:  800 / 2000 [ 40%]  (Sampling)
Chain 4: Iteration: 1000 / 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 seconds (Warm-up)
Chain 4:                0.089508 seconds (Sampling)
Chain 4:                0.089508 seconds (Total)
Chain 4: 
```

Let’s extract the prior draws of the regression line

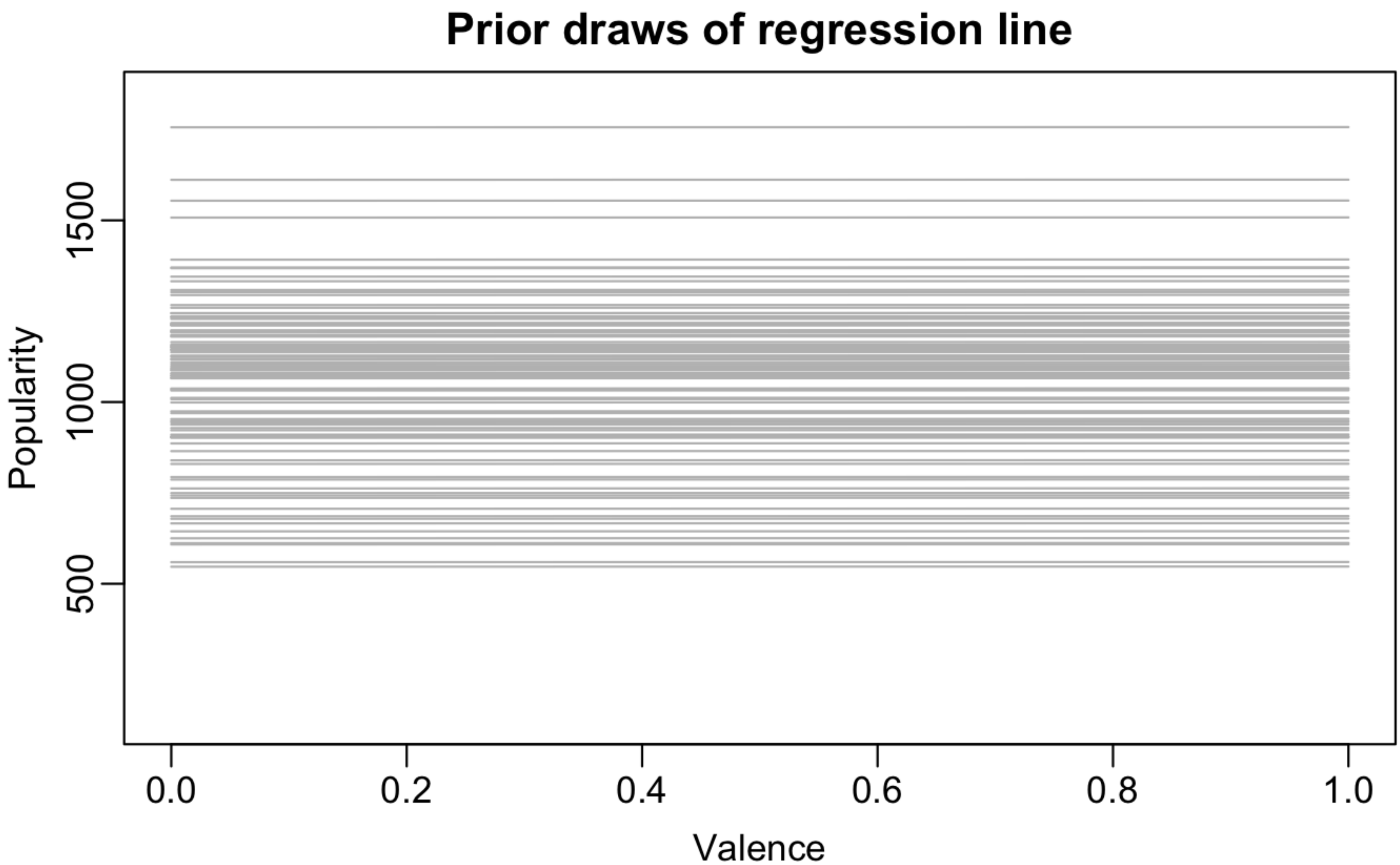
Hide

```
line_samples <- extract(prior_sim, pars = "prior_line")[["prior_line"]]
line_samples2 <- extract(prior_sim, pars = "prior2_line")[["prior2_line"]]
line_samples3 <- extract(prior_sim, pars = "prior3_line")[["prior3_line"]]
```

line_samples is a matrix where the rows index individual prior draws and columns index each point in the grid of x values

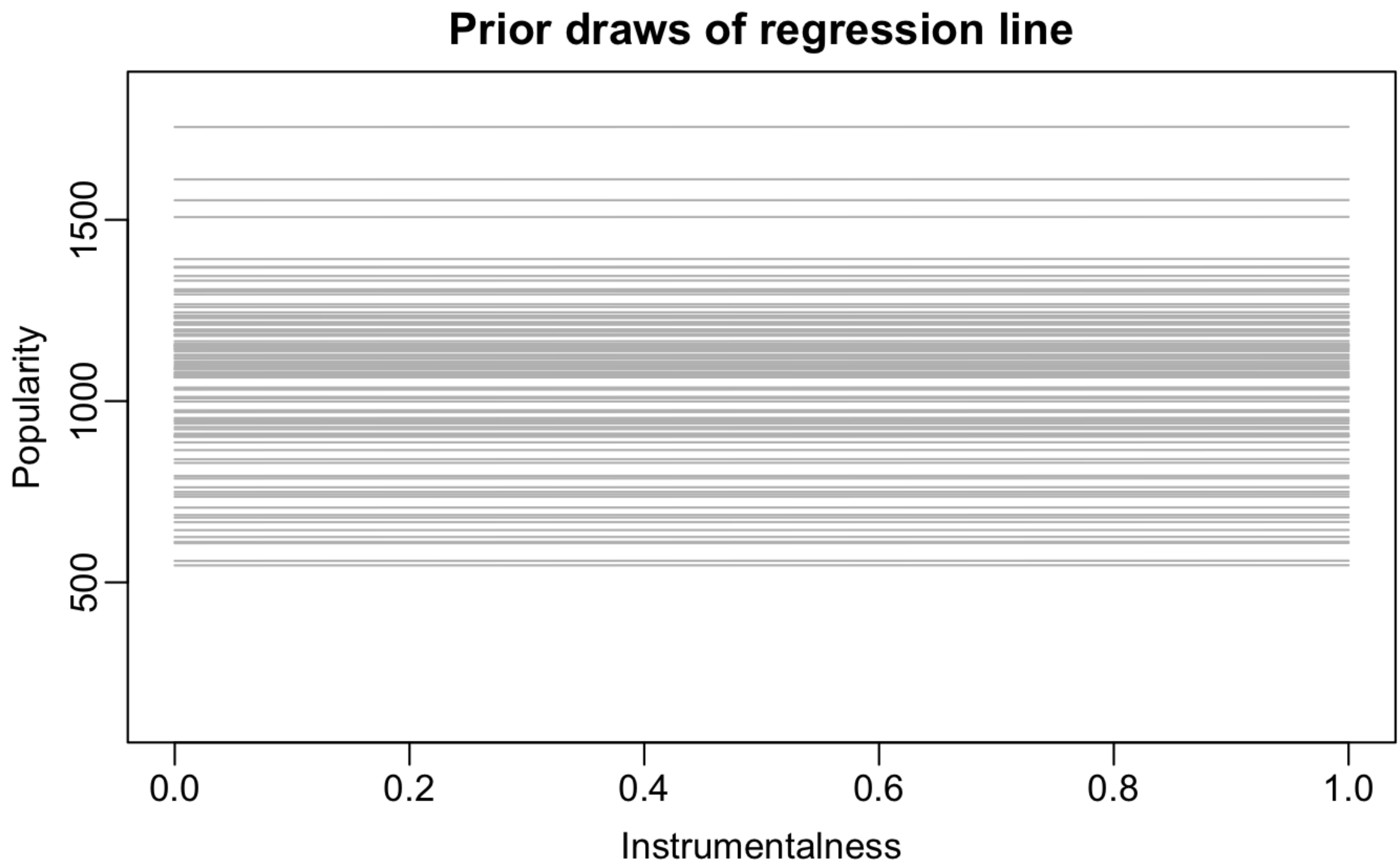
Hide

```
par(mar = c(3,3,2,1), mgp = c(1.8, 0.5, 0))
plot(1, type = "n", xlim = c(0, 1), ylim = c(range(line_samples)),
     xlab = "Valence", ylab = "Popularity", main ="Prior draws of regression line")
for(i in 1:100){
  lines(a_grid, line_samples[i,], col = 'gray', lwd = 1)
}
```



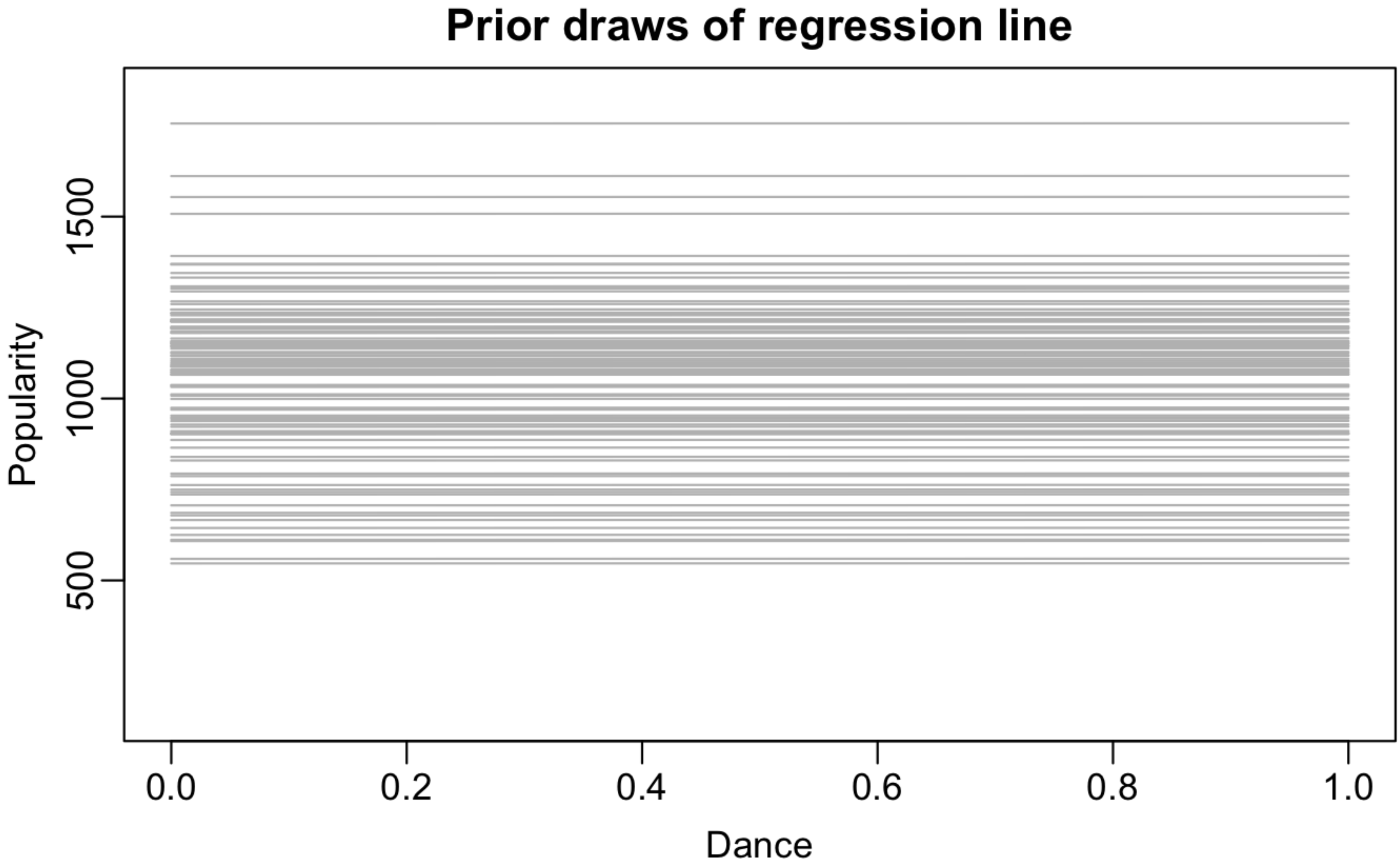
Hide

```
par(mar = c(3,3,2,1), mgp = c(1.8, 0.5, 0))
plot(1, type = "n", xlim = c(0, 1), ylim = c(range(line_samples2)),
     xlab = "Instrumentalness", ylab = "Popularity", main ="Prior draws of regression line")
for(i in 1:100){
  lines(b_grid, line_samples2[i,], col = 'gray', lwd = 1)
}
```



Hide

```
par(mar = c(3,3,2,1), mgp = c(1.8, 0.5, 0))
plot(1, type = "n", xlim = c(0, 1), ylim = c(range(line_samples3)),
     xlab = "Dance", ylab = "Popularity", main ="Prior draws of regression line")
for(i in 1:100){
  lines(c_grid, line_samples3[i,], col = 'gray', lwd = 1)
}
```


Prior draws of regression line
Popularity
500
1000
1500
0.0
0.2
0.4
0.6
0.8
1.0
Dance

Hide

```
# To get a slightly nicer picture, we can look at the posterior mean
# and the 95% prior credible interval for each alpha + beta*x value

line_prior_mean <- apply(line_samples, MARGIN = 2, FUN = mean)
line_prior_l95 <- apply(line_samples, MARGIN = 2, FUN = quantile, probs = 0.025)
line_prior_u95 <- apply(line_samples, MARGIN = 2, FUN = quantile, probs = 0.975)
line2_prior_mean <- apply(line_samples2, MARGIN = 2, FUN = mean)
line2_prior_l95 <- apply(line_samples2, MARGIN = 2, FUN = quantile, probs = 0.025)
line2_prior_u95 <- apply(line_samples2, MARGIN = 2, FUN = quantile, probs = 0.975)
line3_prior_mean <- apply(line_samples3, MARGIN = 2, FUN = mean)
line3_prior_l95 <- apply(line_samples3, MARGIN = 2, FUN = quantile, probs = 0.025)
line3_prior_u95 <- apply(line_samples3, MARGIN = 2, FUN = quantile, probs = 0.975)
```

Hide

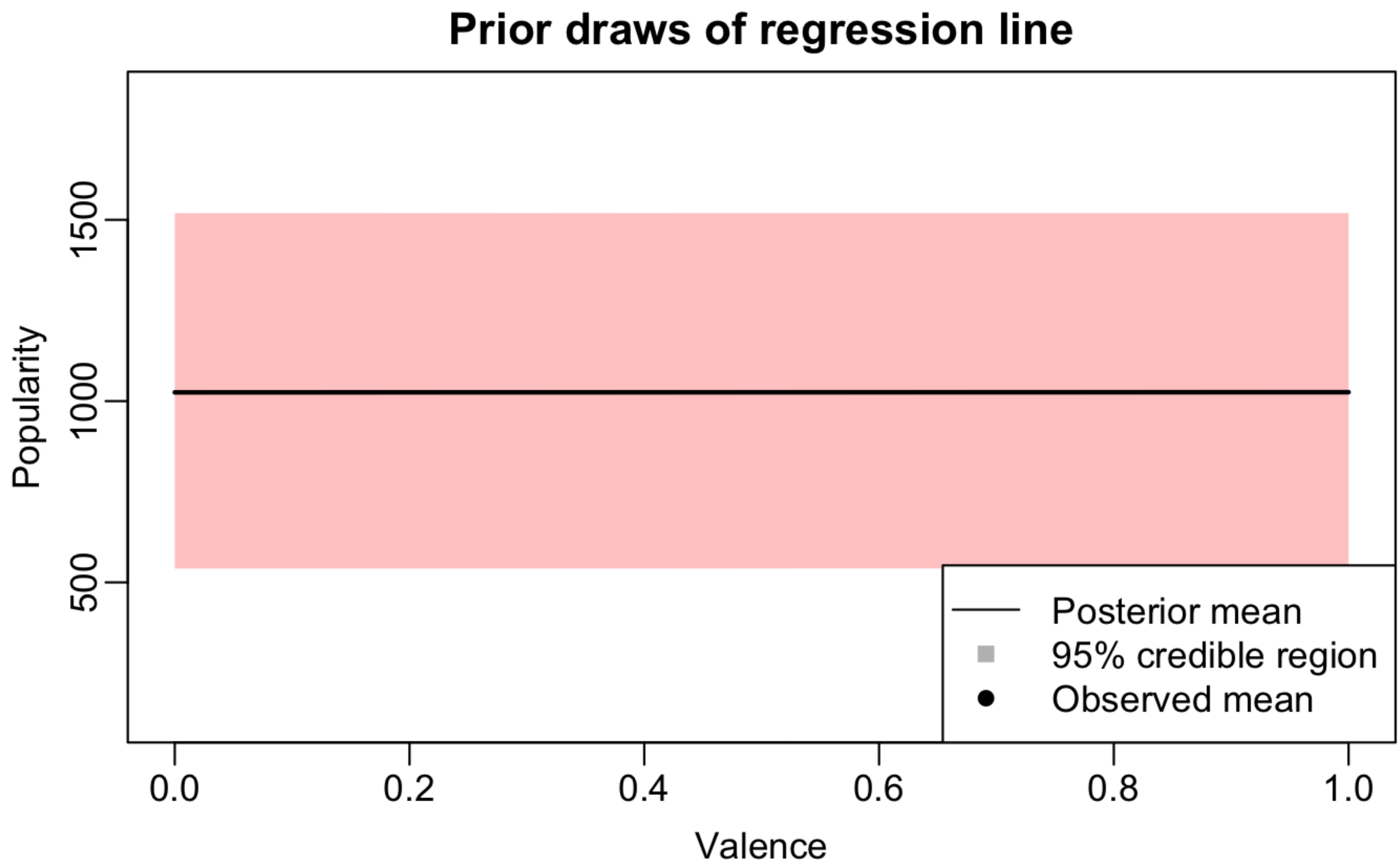
```
par(mar = c(3,3,2,1), mgp = c(1.8, 0.5, 0))
plot(1, type = "n", xlim = c(0, 1), ylim = range(line_samples),
     xlab = "Valence", ylab = "Popularity", main ="Prior draws of regression line")
polygon(x = c(a_grid, rev(a_grid)),
        y = c(line_prior_l95, rev(line_prior_u95)),
        col = rgb(1, 0, 0, 0.25), border = NA)
```

Hide

```
lines(a_grid, line_prior_mean, lwd = 2)
points(x = a_mean, y = y_mean, pch = 16)
```

Hide

```
abline(h = 0, lty = 2, col = 'gray')
legend("bottomright",
       legend = c("Posterior mean", "95% credible region", "Observed mean"),
       col = c("black", "gray", "black"),
       pch = c(NA,15,16), lty = c(1, NA, 0))
```



Hide

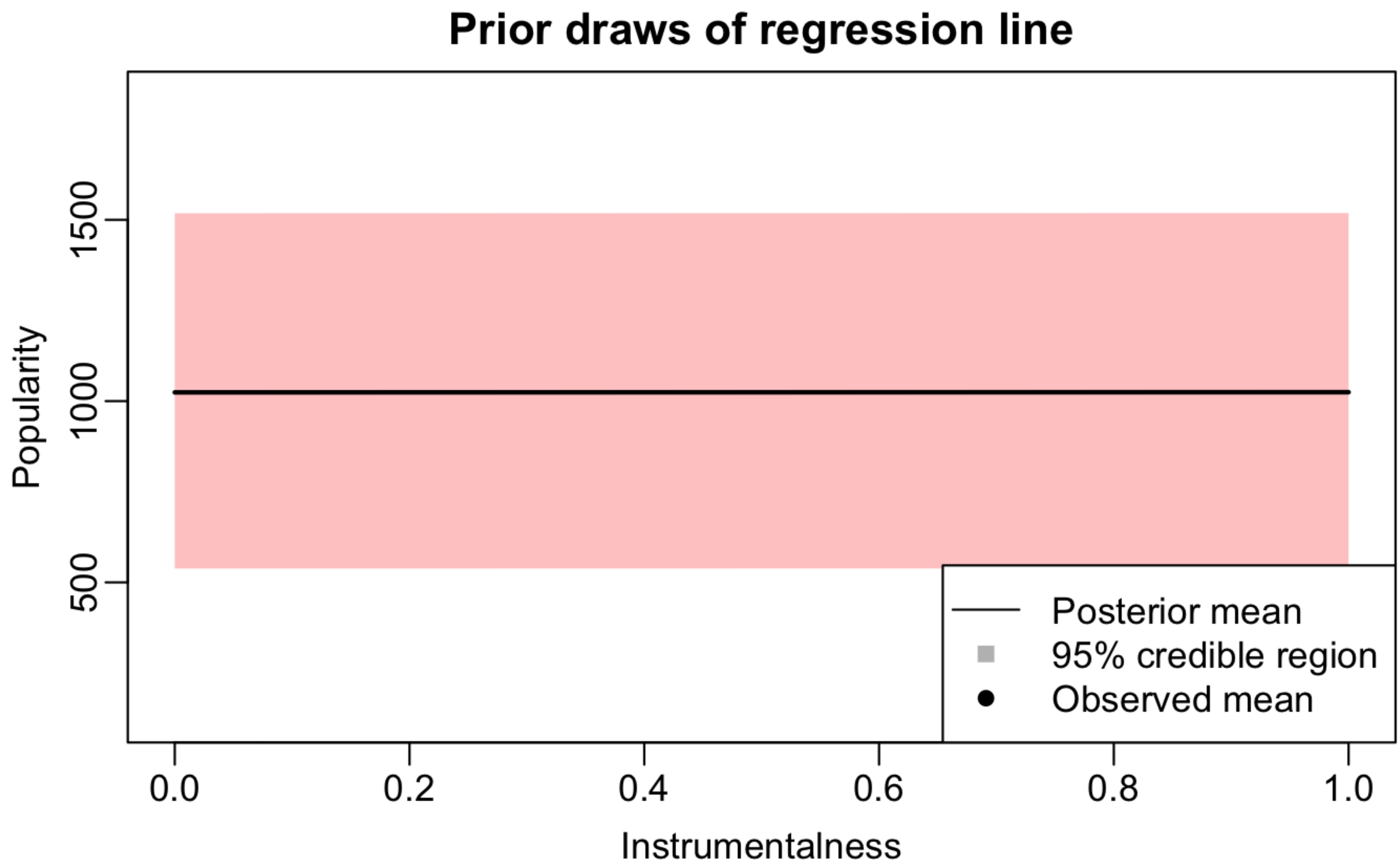
```
par(mar = c(3,3,2,1), mgp = c(1.8, 0.5, 0))
plot(1, type = "n", xlim = c(0, 1), ylim = range(line_samples2),
     xlab = "Instrumentalness", ylab = "Popularity", main ="Prior draws of regression line")
polygon(x = c(b_grid, rev(b_grid)),
        y = c(line2_prior_l95, rev(line2_prior_u95)),
        col = rgb(1, 0, 0, 0.25), border = NA)
```

Hide

```
lines(b_grid, line2_prior_mean, lwd = 2)
points(x = b_mean, y = y_mean, pch = 16)
```

Hide

```
abline(h = 0, lty = 2, col = 'gray')
legend("bottomright",
       legend = c("Posterior mean", "95% credible region", "Observed mean"),
       col = c("black", "gray", "black"),
       pch = c(NA,15,16), lty = c(1, NA, 0))
```



Hide

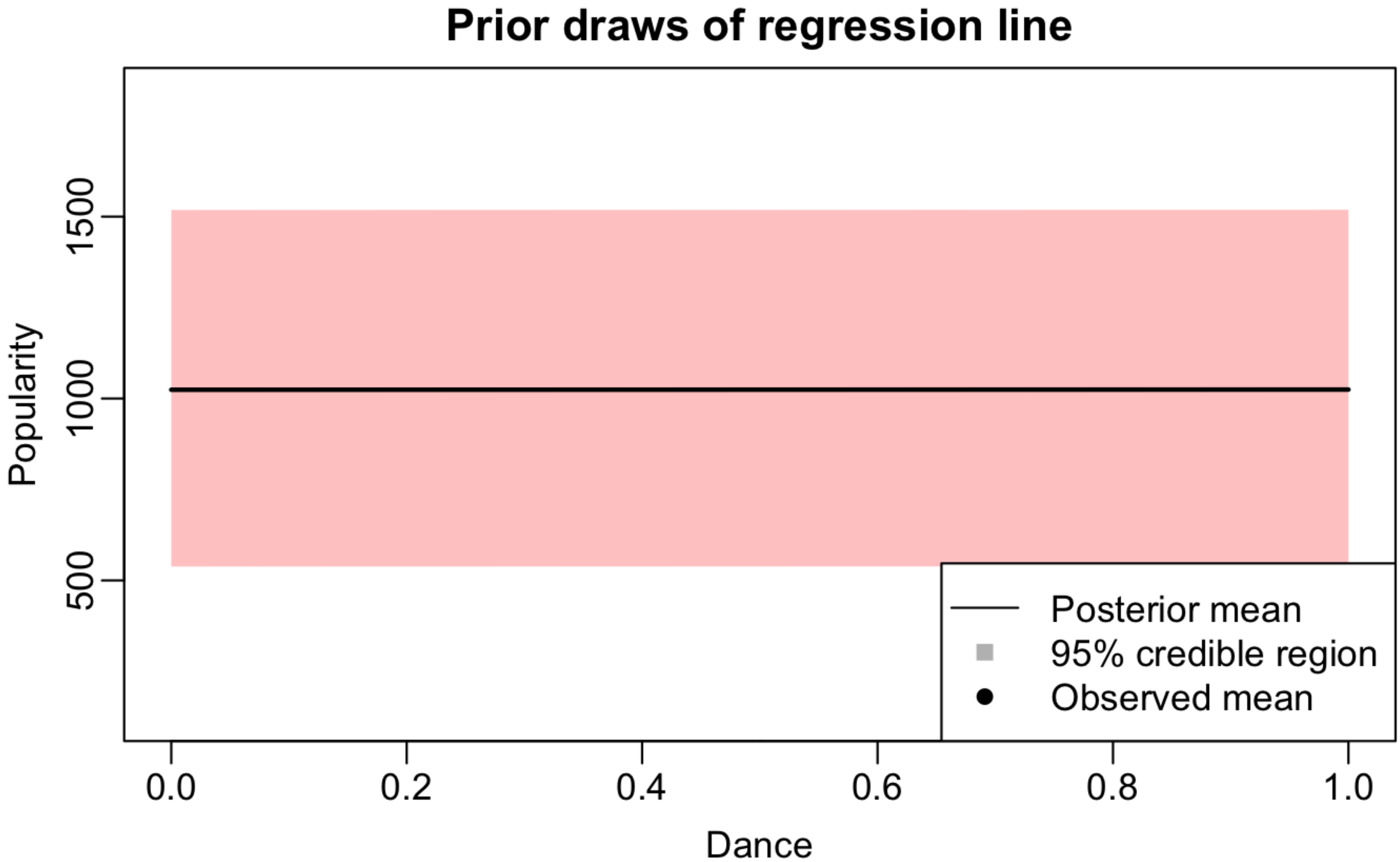
```
par(mar = c(3,3,2,1), mgp = c(1.8, 0.5, 0))
plot(1, type = "n", xlim = c(0, 1), ylim = range(line_samples3),
     xlab = "Dance", ylab = "Popularity", main ="Prior draws of regression line")
polygon(x = c(c_grid, rev(c_grid)),
        y = c(line3_prior_l95, rev(line3_prior_u95)),
        col = rgb(1, 0, 0, 0.25), border = NA)
```

Hide

```
lines(c_grid, line3_prior_mean, lwd = 2)
points(x = c_mean, y = y_mean, pch = 16)
```

Hide

```
abline(h = 0, lty = 2, col = 'gray')
legend("bottomright",
       legend = c("Posterior mean", "95% credible region", "Observed mean"),
       col = c("black", "gray", "black"),
       pch = c(NA,15,16), lty = c(1, NA, 0))
```


Prior draws of regression line
Popularity
500
1000
1500
0.0
0.2
0.4
0.6
0.8
1.0
Dance
Posterior mean
95% credible region
Observed mean

5.

Hide

```
set.seed(0)
idx = as.character(sample(1:75,50,replace = FALSE))
rock$track_artist <- as.factor(rock$track_artist)
rock$track_artist = as.numeric(rock$track_artist)
rock$track_artist = as.character(rock$track_artist)
rocktrain = rock[which(rock$track_artist %in% idx),]
rocktest = rock[which(!(rock$track_artist %in% idx)),]

a <- rocktrain$valence
b <- rocktrain$instrumentalness
c <- rocktrain$danceability
y <- rocktrain$popularity
```

Hide

```
a_mean <- mean(a)
a_sd <- sd(a)
b_mean <- mean(b)
b_sd <- sd(b)
c_mean <- mean(c)
c_sd <- sd(c)

y_mean <- mean(y)
y_sd <- sd(y)

n <- length(y)
std_a <- (a - a_mean)/a_sd
std_b <- (b - b_mean)/b_sd
std_c <- (c - c_mean)/c_sd
std_y <- (y - y_mean)/y_sd

#########
# Load in our test dataset
a_test <- rocktest$valence
b_test <- rocktest$instrumentalness
c_test <- rocktest$danceability
y_test <- rocktest$popularity
n_test <- length(y_test)
```

Hide

```
# Define a grid of x-values at which we want to evaluate the regression line
# (i.e. so that we can make a spaghetti plot of the posteiror draws of the lines)
a_grid <- seq(0, 1, by = 0.05)
m_grid <- length(a_grid)
b_grid <- seq(0, 1, by = 0.05)
o_grid <- length(b_grid)
c_grid <- seq(0, 1, by = 0.05)
q_grid <- length(c_grid)
```

Set all of our prior hyperparameters

Hide

```
mu_std_alpha <- 40
sigma_std_alpha <- 10

mu_std_beta <- 0.5 * a_sd/y_sd
sigma_std_beta <- 0.05 * a_sd/y_sd

mu_std_gamma <- 0.5 * b_sd/y_sd
sigma_std_gamma <- 0.05 * b_sd/y_sd

mu_std_delta <- 0.5 * c_sd/y_sd
sigma_std_delta <- 0.05 * c_sd/y_sd

nu <- 7
A <- 1
```

Hide

```
rock_data <- list(n = n, std_y = std_y, std_a = std_a, std_b = std_b, std_c= std_c,
                   y_mean = y_mean, y_sd = y_sd, a_mean = a_mean, a_sd = a_sd, b_mean = b_mean, b_sd = b_sd, c_me
an = c_mean, c_sd = c_sd,
                   m_grid = m_grid, a_grid = a_grid, o_grid = o_grid, b_grid = b_grid, q_grid = q_grid, c_grid =
c_grid,
                   n_test = n_test, a_test = a_test,  b_test = b_test,  c_test = c_test,
                   mu_std_alpha = mu_std_alpha, sigma_std_alpha = sigma_std_alpha,
                   mu_std_beta = mu_std_beta, sigma_std_beta = sigma_std_beta, mu_std_gamma = mu_std_gamma, sigma
_std_gamma = sigma_std_gamma, mu_std_delta = mu_std_delta, sigma_std_delta = sigma_std_delta,
                   nu = nu, A = A)
```

Hide

```
linear_model <- stan_model(file = "linear_regression_single_predictor.stan")
```

```
'config' variable 'CPP' is deprecated
```

```
clang -mmacosx-version-min=10.13 -E
```

```
Warning in system(paste(CPP, ARGS), ignore.stdout = TRUE, ignore.stderr = TRUE) :
  error in running command
```

Hide

```
fit <- sampling(object = linear_model,
                data = rock_data,
                pars = c("alpha", "beta", "gamma","delta","sigma", "post_linem","post_lineo", "post_lineq","ysta
r"))
```

```
SAMPLING FOR MODEL 'linear_regression_single_predictor' NOW (CHAIN 1).
Chain 1: 
Chain 1: Gradient evaluation took 0.000181 seconds
Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 1.81 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: 2.3415 seconds (Warm-up)
Chain 1:                1.6586 seconds (Sampling)
Chain 1:                4.0001 seconds (Total)
Chain 1: 

SAMPLING FOR MODEL 'linear_regression_single_predictor' NOW (CHAIN 2).
Chain 2: 
Chain 2: Gradient evaluation took 7.9e-05 seconds
Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.79 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: 2.32786 seconds (Warm-up)
Chain 2:                1.80324 seconds (Sampling)
Chain 2:                4.1311 seconds (Total)
Chain 2: 

SAMPLING FOR MODEL 'linear_regression_single_predictor' NOW (CHAIN 3).
Chain 3: 
Chain 3: Gradient evaluation took 0.000113 seconds
Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 1.13 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: 2.9564 seconds (Warm-up)
Chain 3:                2.00235 seconds (Sampling)
Chain 3:                4.95875 seconds (Total)
Chain 3: 

SAMPLING FOR MODEL 'linear_regression_single_predictor' NOW (CHAIN 4).
Chain 4: 
```

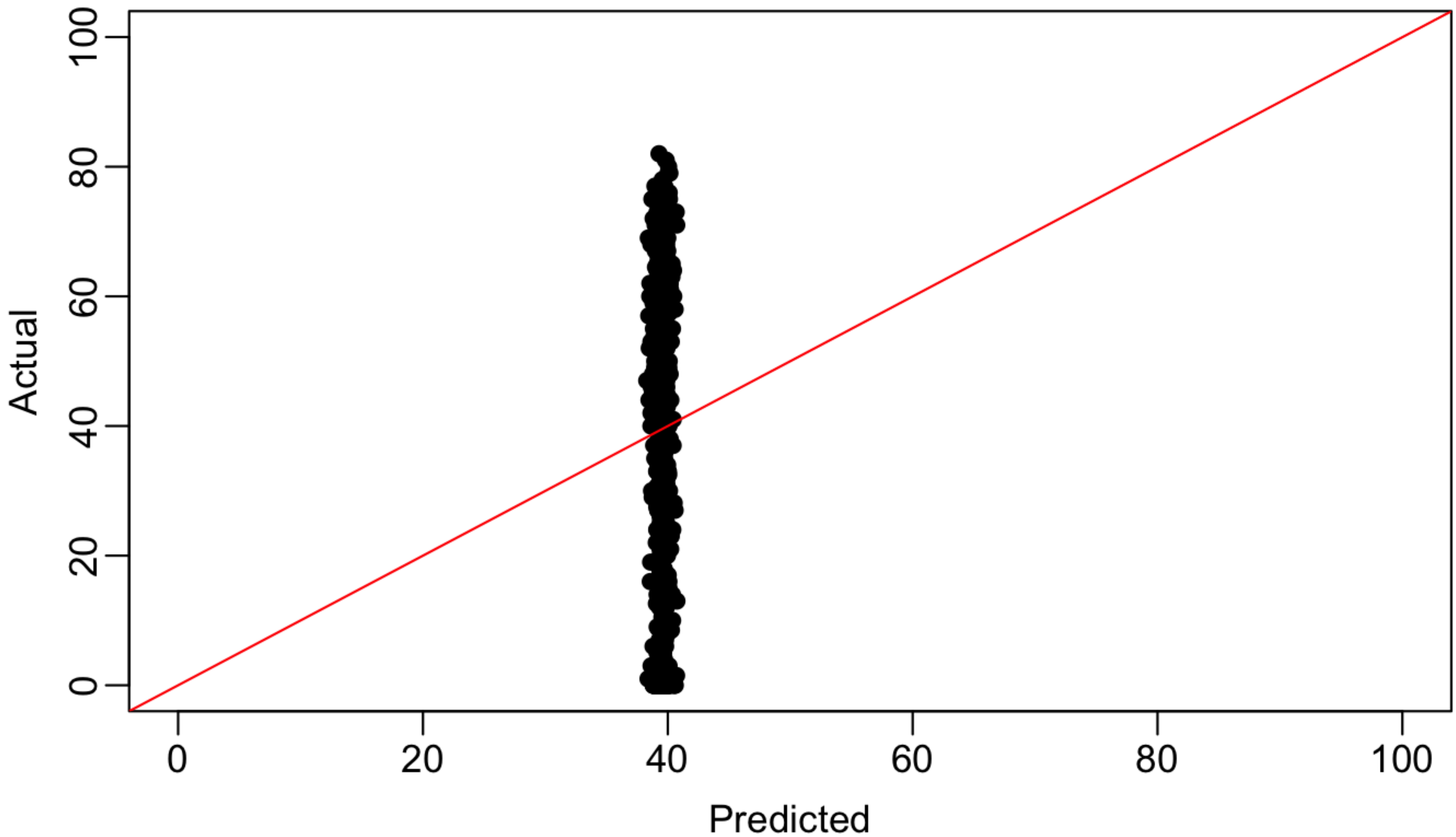
```
Chain 4: Gradient evaluation took 8.6e-05 seconds
Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.86 seconds.
Chain 4: Adjust your expectations accordingly!
Chain 4:
Chain 4:
Chain 4: Iteration:    1 / 2000 [  0%]  (Warmup)
Chain 4: Iteration:  200 / 2000 [ 10%]  (Warmup)
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: 3.48012 seconds (Warm-up)
Chain 4:                2.00095 seconds (Sampling)
Chain 4:                5.48106 seconds (Total)
Chain 4:
```

Hide

```
# Let's get the posterior predictive draws of ystar
ystar_samples <- extract(object = fit, pars = "ystar")[["ystar"]]

ystar_mean <- apply(ystar_samples, MARGIN = 2, FUN = mean)
ystar_l95 <- apply(ystar_samples, MARGIN = 2, FUN = quantile, probs = 0.025)
ystar_u95 <- apply(ystar_samples, MARGIN = 2, FUN = quantile, probs = 0.975)

par(mar = c(3,3,2,1), mgp = c(1.8, 0.5, 0))
plot(ystar_mean, y_test, xlab = "Predicted", ylab = "Actual", pch = 16,
     xlim = c(0, 100), ylim = c(0, 100), )
abline(a = 0, b = 1, col = 'red')
```



Hide

```
# Compute the mean square error
mean( (y_test - ystar_mean)^2 )
```

```
[1] 624.2348
```

Hide

```
# To get a sense of how large that is, compare it to a constant prediction
# in which we predict everything in the test dataset to the mean of the training dataset
mean( (y_test - ystar_mean)^2 )/mean( (y_test - y_mean)^2 )
```

```
[1] 1.001246
```

Hide

```
# Standardized mean square error of 0.66 means we've explained about 34% of variation

# How often did observed value lie in our uncertainty interval
test_in_interval <- ((y_test >= ystar_l95) & (y_test <= ystar_u95)) # vector of true/false values
mean(test_in_interval ) # 96% of the time!
```

```
[1] 1
```

We can visualize this coverage easily To help visualize, it helps to sort the testing data in order of increasing y

Hide

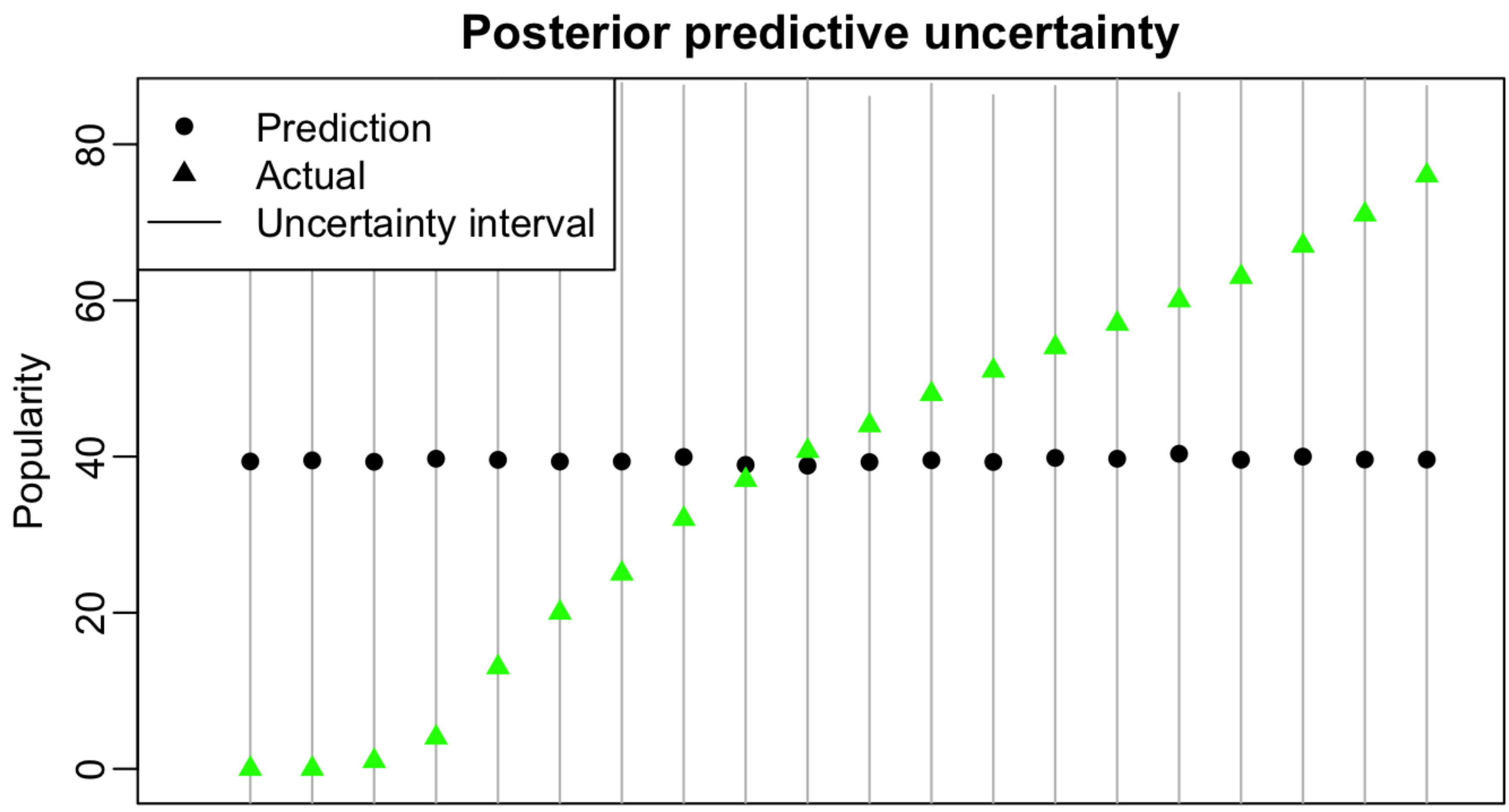
```
sorted_test <- sort(y_test, decreasing = FALSE, index.return = TRUE)
sorted_index <- sorted_test$ix # tells us the order of test set indices corresponding to increasing y's

idx_graph = which((1:1308)%%30 == 0)

par(mar = c(3,3,2,1), mgp = c(1.8, 0.5, 0))
plot(1, type = "n", xlim = c(0, 613), ylim = range(c(-1,85)),
     main = "Posterior predictive uncertainty", ylab = "Popularity", xlab = "", xaxt = "n")
for(i in idx_graph){
  lines(x = c(i,i), y = c(ystar_l95[sorted_index[i]], ystar_u95[sorted_index[i]]), col = 'gray')
  points(x = i, y = ystar_mean[sorted_index[i]], col = 'black', pch = 16)
  # We will add the actual observed values to the plot
  # If the observed value lies in the interval
  points(x = i, y = y_test[sorted_index[i]], pch = 17,
         col = ifelse(test_in_interval[sorted_index[i]],'green', 'red'))
}
```

Hide

```
legend("topleft", legend = c("Prediction", "Actual", "Uncertainty interval"),
       pch = c(16, 17, NA), lty = c(NA, NA, 1))
```



Hide

```
linem_samples <- extract(object = fit, pars = "post_linem")[["post_linem"]]
lineo_samples <- extract(object = fit, pars = "post_lineo")[["post_lineo"]]
lineq_samples <- extract(object = fit, pars = "post_lineq")[["post_lineq"]]
linem_post_mean <- apply(linem_samples, MARGIN = 2, FUN = mean)
linem_post_l95 <- apply(linem_samples, MARGIN = 2, FUN = quantile, probs = 0.025)
linem_post_u95 <- apply(linem_samples, MARGIN = 2, FUN = quantile, probs = 0.975)
lineo_post_mean <- apply(lineo_samples, MARGIN = 2, FUN = mean)
lineo_post_l95 <- apply(lineo_samples, MARGIN = 2, FUN = quantile, probs = 0.025)
lineo_post_u95 <- apply(lineo_samples, MARGIN = 2, FUN = quantile, probs = 0.975)
lineq_post_mean <- apply(lineq_samples, MARGIN = 2, FUN = mean)
lineq_post_l95 <- apply(lineq_samples, MARGIN = 2, FUN = quantile, probs = 0.025)
lineq_post_u95 <- apply(lineq_samples, MARGIN = 2, FUN = quantile, probs = 0.975)

par(mar = c(3,3,2,1), mgp = c(1.8, 0.5, 0))
plot(1, type = "n", xlim = c(0,1), ylim = range(c(y, linem_samples)),
     xlab = "Valence", ylab = "Popularity", main ="Prior draws of regression line")
polygon(x = c(a_grid, rev(a_grid)),
        y = c(linem_post_l95, rev(linem_post_u95)),
        col = rgb(0.9, 0.9, 0.9), border = NA)
```

Hide

```
lines(a_grid, linem_post_mean, lwd = 2)
points(x = a_mean, y = y_mean, pch = 16)
```

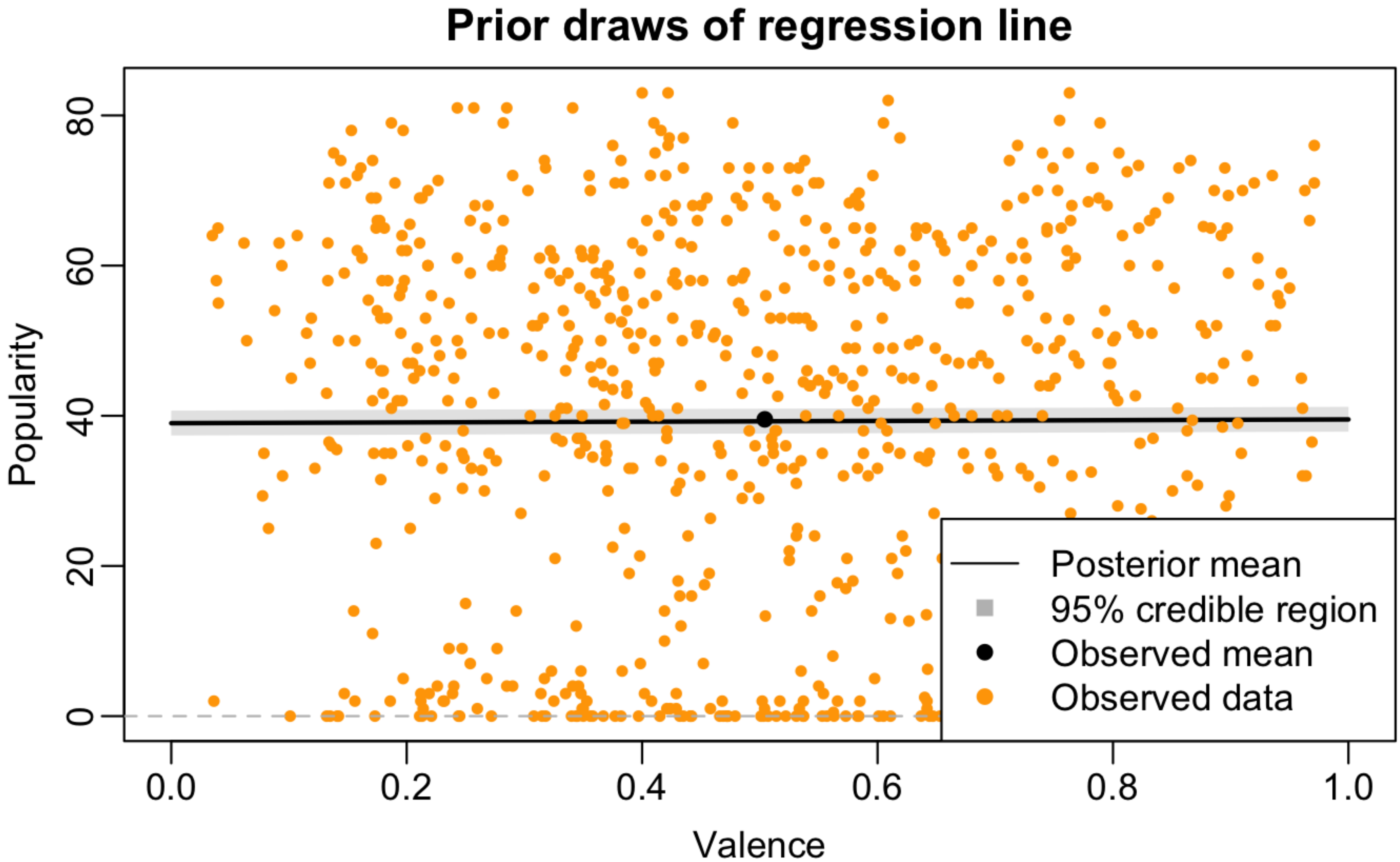
Hide

```
points(a,y, pch = 16, col = 'orange', cex = 0.7)

abline(h = 0, lty = 2, col = 'gray')
```

Hide

```
legend("bottomright",
       legend = c("Posterior mean", "95% credible region", "Observed mean", "Observed data"),
       col = c("black", "gray", "black", "orange"),
       pch = c(NA,15,16, 16), lty = c(1, NA, 0, NA))
```



Hide

```
par(mar = c(3,3,2,1), mgp = c(1.8, 0.5, 0))
plot(1, type = "n", xlim = c(0, 1), ylim = range(c(y, lineo_samples)),
     xlab = "Instrumentalness", ylab = "Popularity", main ="Prior draws of regression line")
polygon(x = c(b_grid, rev(b_grid)),
        y = c(lineo_post_l95, rev(lineo_post_u95)),
        col = rgb(0.9, 0.9, 0.9), border = NA)
```

Hide

```
lines(b_grid, lineo_post_mean, lwd = 2)
points(x = b_mean, y = y_mean, pch = 16)
```

Hide

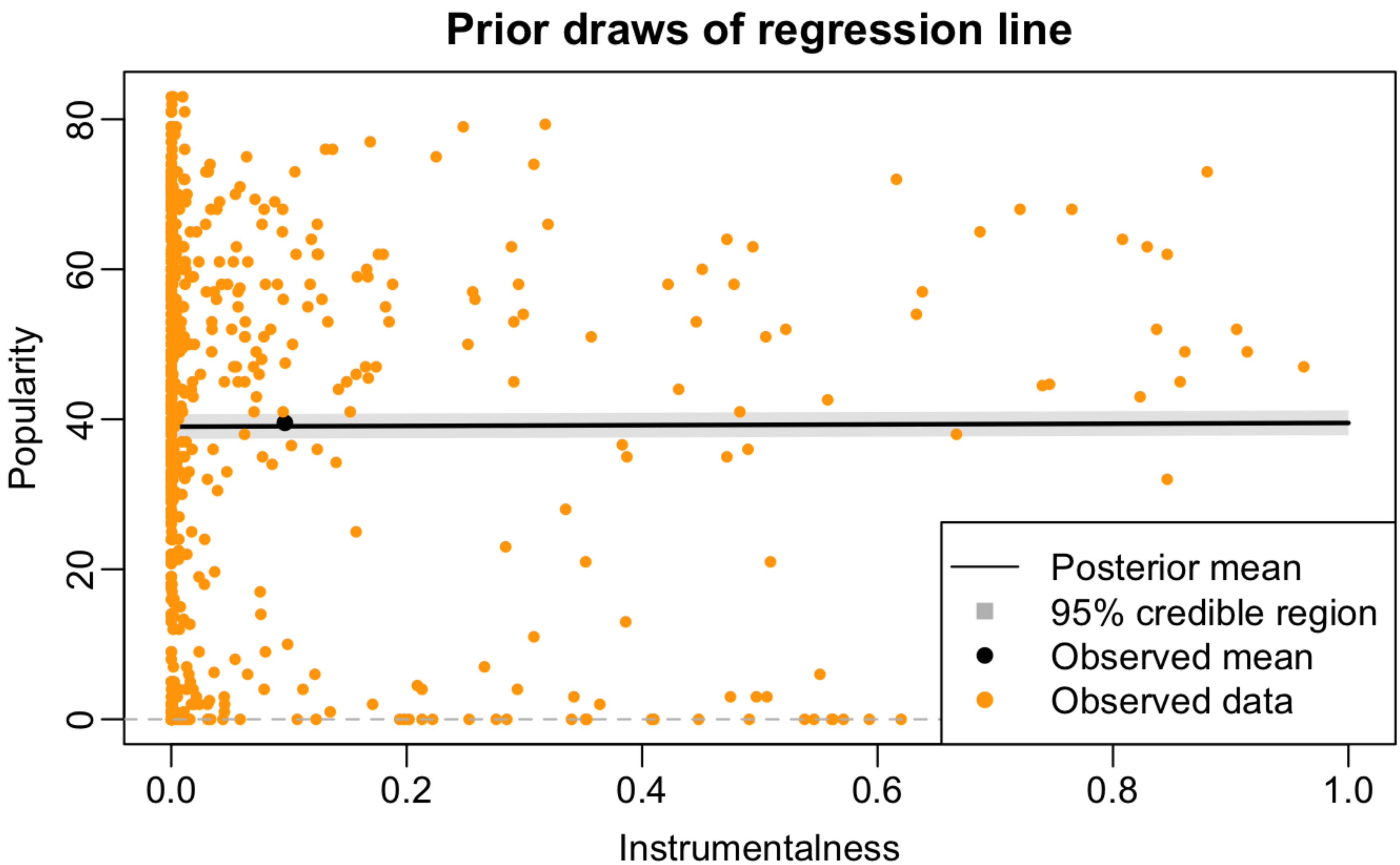
```


points(b,y, pch = 16, col = 'orange', cex = 0.7)

abline(h = 0, lty = 2, col = 'gray')
```

Hide

```
legend("bottomright",
       legend = c("Posterior mean", "95% credible region", "Observed mean", "Observed data"),
       col = c("black", "gray", "black", "orange"),
       pch = c(NA,15,16, 16), lty = c(1, NA, 0, NA))
```



Hide

```
par(mar = c(3,3,2,1), mgp = c(1.8, 0.5, 0))
plot(1, type = "n", xlim = c(0, 1), ylim = range(c(y, lineq_samples)),
     xlab = "Danceability", ylab = "Popularity", main ="Prior draws of regression line")
polygon(x = c(c_grid, rev(c_grid)),
        y = c(lineq_post_l95, rev(lineq_post_u95)),
        col = rgb(0.9, 0.9, 0.9), border = NA)
```

Hide

```
lines(c_grid, lineq_post_mean, lwd = 2)
points(x = c_mean, y = y_mean, pch = 16)
```

Hide

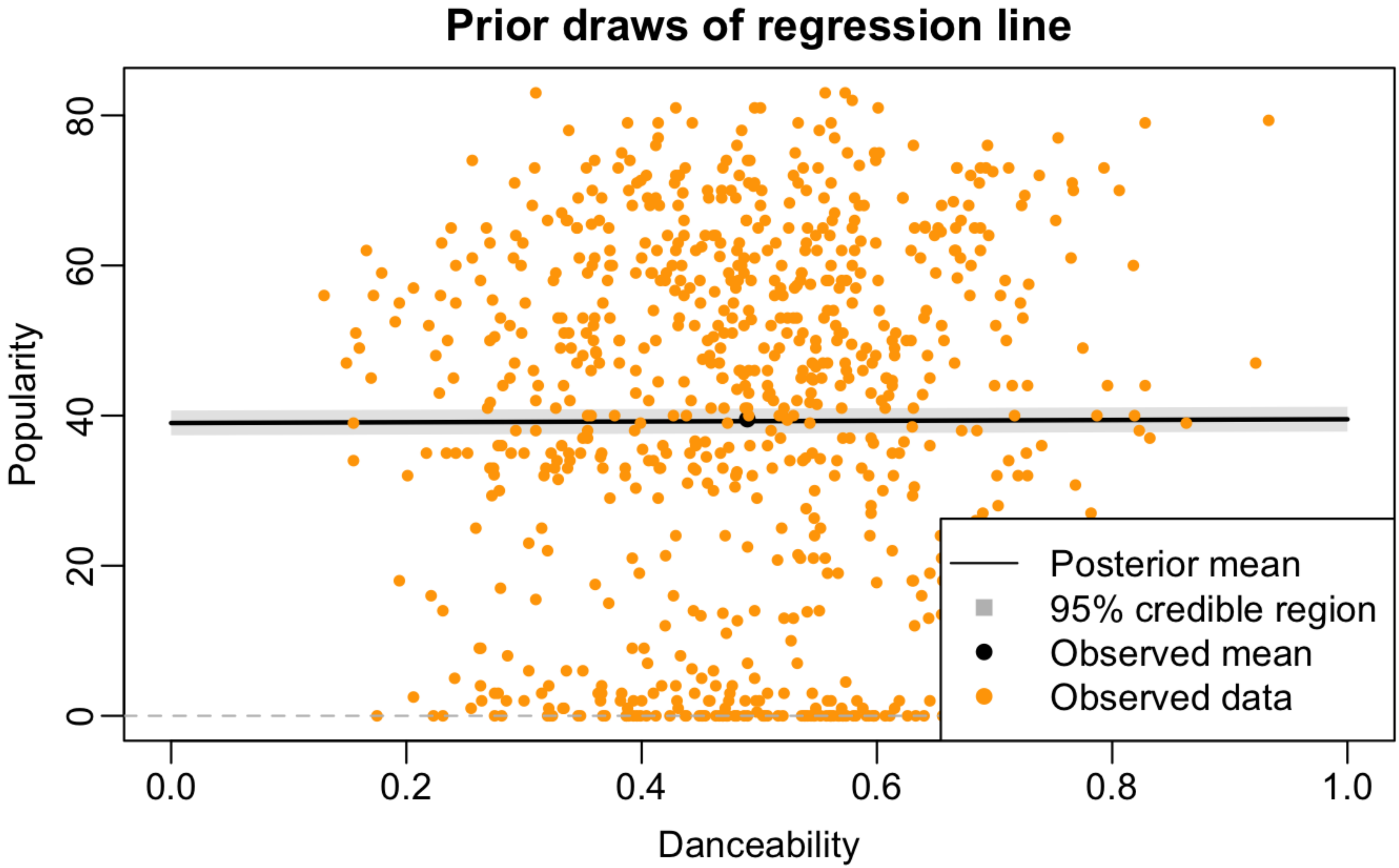
```


points(c,y, pch = 16, col = 'orange', cex = 0.7)

abline(h = 0, lty = 2, col = 'gray')
```

Hide

```
legend("bottomright",
       legend = c("Posterior mean", "95% credible region", "Observed mean", "Observed data"),
       col = c("black", "gray", "black", "orange"),
       pch = c(NA,15,16, 16), lty = c(1, NA, 0, NA))
```


Prior draws of regression line
Popularity
0
20
40
60
80
0.0
0.2
0.4
0.6
0.8
1.0
Danceability
Posterior mean
95% credible region
Observed mean
Observed data

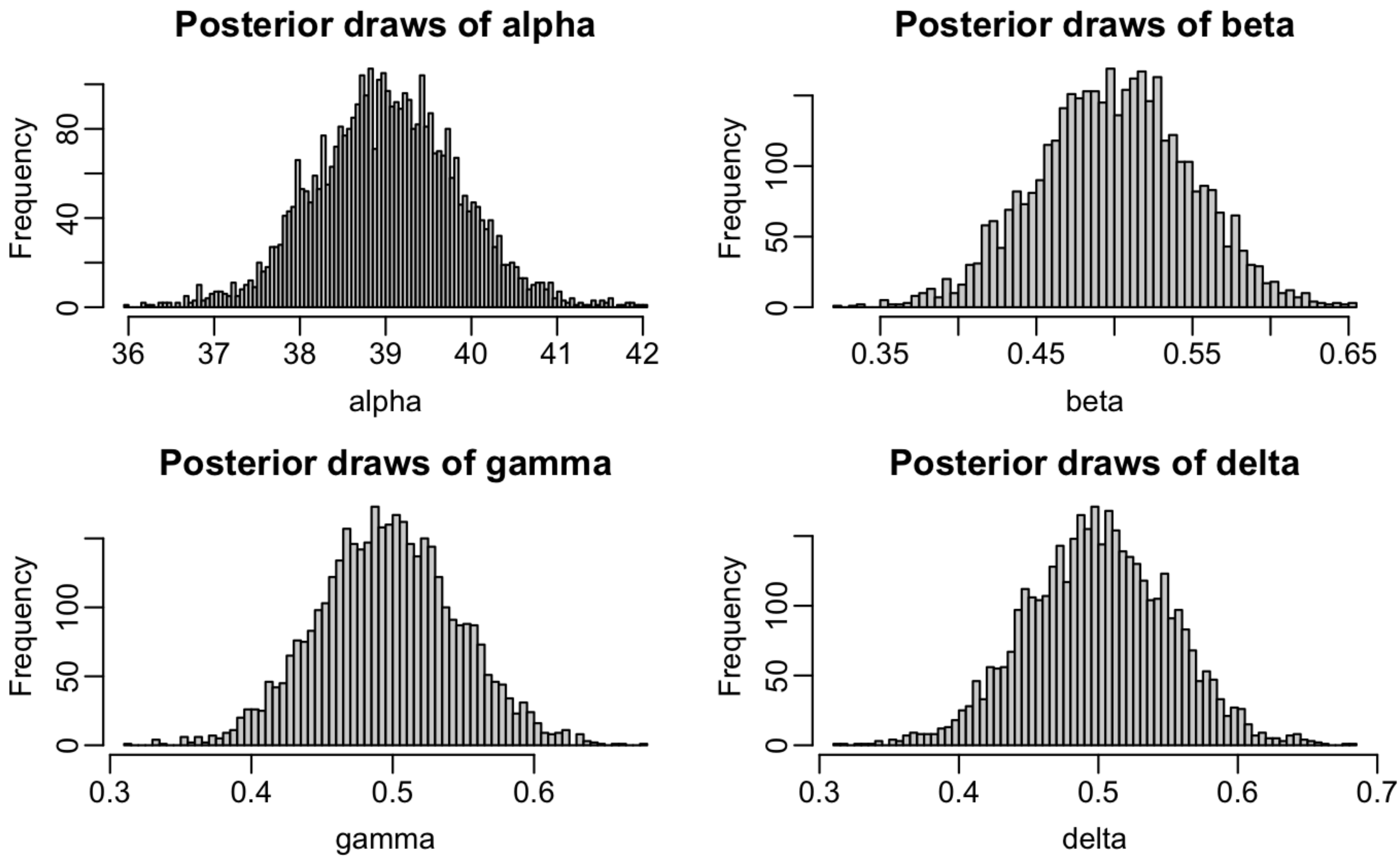
Hide

```
alpha_samples <- extract(object = fit, pars = "alpha")[["alpha"]]
beta_samples <- extract(object = fit, pars = "beta")[["beta"]]
gamma_samples <- extract(object = fit, pars = "gamma")[["gamma"]]
delta_samples <- extract(object = fit, pars = "delta")[["delta"]]

par(mar = c(3,3,2,1), mgp = c(1.8, 0.5, 0), mfrow = c(2,2))
hist(alpha_samples, breaks = 100, main = "Posterior draws of alpha", xlab = "alpha")
hist(beta_samples, breaks = 100, main = "Posterior draws of beta", xlab = "beta")
```

Hide

```
hist(gamma_samples, breaks = 100, main = "Posterior draws of gamma", xlab = "gamma")
hist(delta_samples, breaks = 100, main = "Posterior draws of delta", xlab = "delta")
```


Posterior draws of alpha
Frequency
0
40
80
36 37 38 39 40 41 42
alpha
Posterior draws of beta
Frequency
0
50
100
0.35 0.45 0.55 0.65
beta
Posterior draws of gamma
Frequency
0
50
100
0.3 0.4 0.5 0.6
gamma
Posterior draws of delta
Frequency
0
50
100
0.3 0.4 0.5 0.6 0.7
delta