Regression Demo

Data Description

The data was obtained from https://www.kaggle.com/fivethirtyeight/the-ultimate-halloween-candy-power-ranking.
The original article and analysis is available at https://fivethirtyeight.com/features/the-ultimate-halloween-candy-power-ranking

```r
candy <- read_csv('http://math.montana.edu/ahoegh/teaching/stat446/candy-data.csv')
```

```
## Parsed with column specification:
## cols(
##   competitorname = col_character(),
##   chocolate = col_double(),
##   fruity = col_double(),
##   caramel = col_double(),
##   peanutyalmondy = col_double(),
##   nougat = col_double(),
##   crispedricewafer = col_double(),
##   hard = col_double(),
##   bar = col_double(),
##   pluribus = col_double(),
##   sugarpercent = col_double(),
##   pricepercent = col_double(),
##   winpercent = col_double()
## )
```

```r
candy <- candy %>% mutate(chocolate_factor = as.factor(chocolate),
                          nut_factor = as.factor(peanutyalmondy))
```

Context

What's the best (or at least the most popular) Halloween candy? That was the question this dataset was collected to answer. Data was collected by creating a website where participants were shown presenting two fun-sized candies and asked to click on the one they would prefer to receive. In total, more than 269 thousand votes were collected from 8,371 different IP addresses.

Content

candy-data.csv includes attributes for each candy along with its ranking. For binary variables, 1 means yes, 0 means no. The data contains the following fields:

- chocolate: Does it contain chocolate?
- fruity: Is it fruit flavored?
- caramel: Is there caramel in the candy?
- peanutalmondy: Does it contain peanuts, peanut butter or almonds?
- nougat: Does it contain nougat?
- crispedricewafer: Does it contain crisped rice, wafers, or a cookie component?
- hard: Is it a hard candy?
- bar: Is it a candy bar?
- pluribus: Is it one of many candies in a bag or box?
- sugarpercent: The percentile of sugar it falls under within the data set.
- pricepercent: The unit price percentile compared to the rest of the set.
- winpercent: The overall win percentage according to 269,000 matchups.

Acknowledgements:

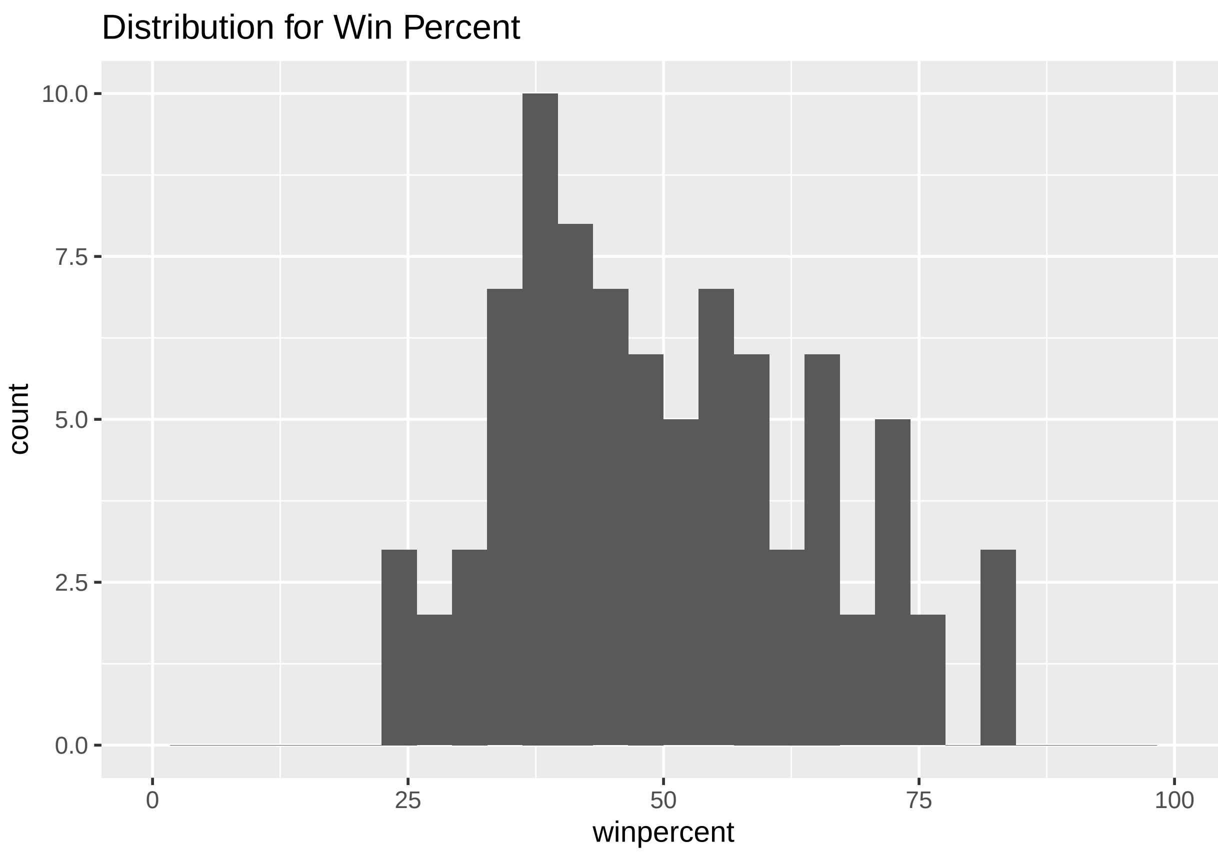
This dataset is Copyright (c) 2014 ESPN Internet Ventures and distributed under an MIT license. Check out the analysis and write-up here: The Ultimate Halloween Candy Power Ranking. Thanks to Walt Hickey for making the data available.

Data Exploration

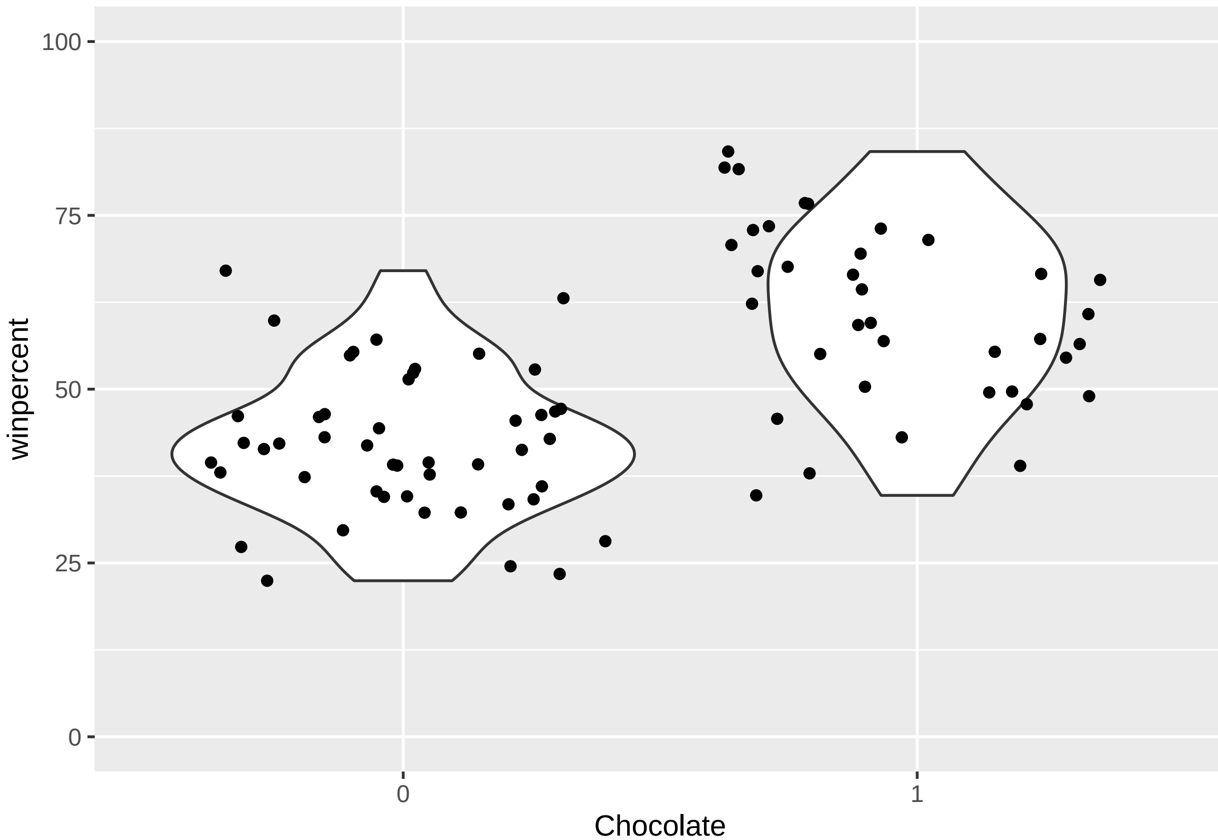
```
candy %>% ggplot(aes(x = winpercent)) + geom_histogram() + ggtitle('Distribution for Win Percent') +
  xlim(0,100)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

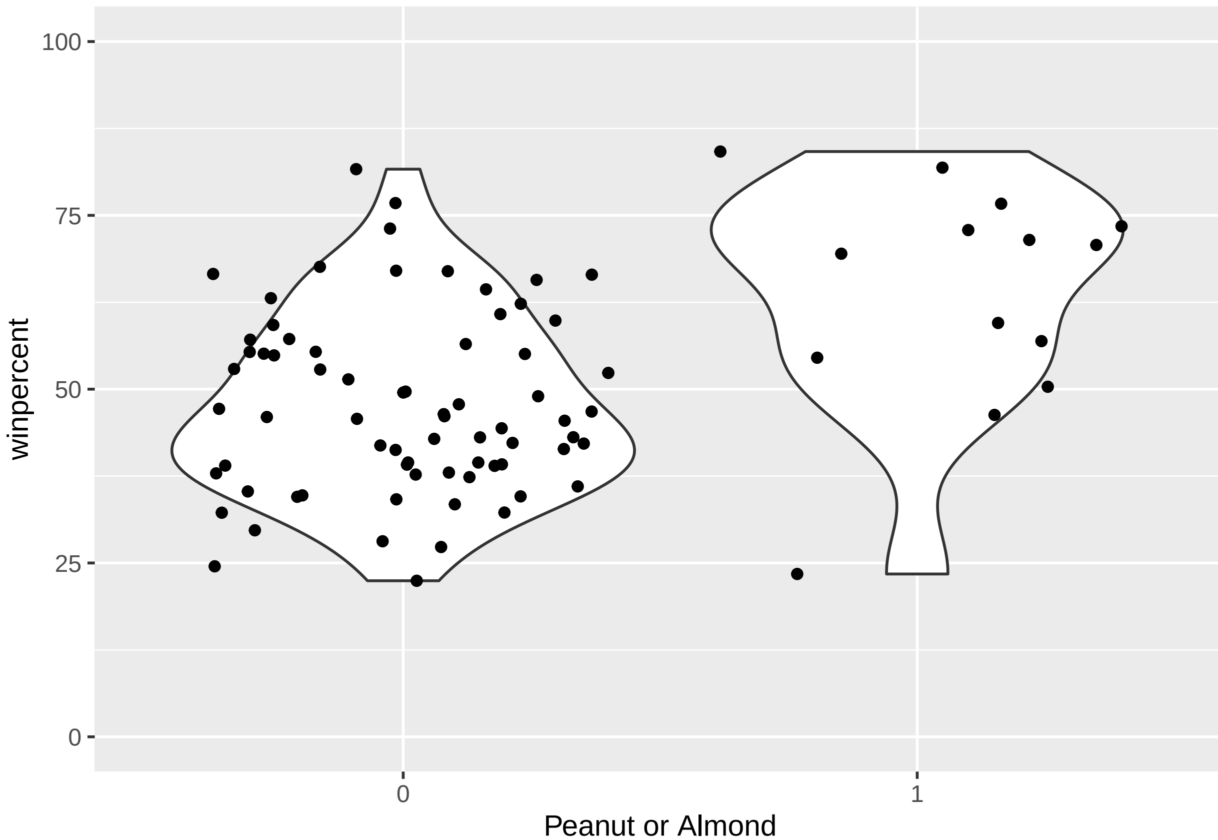
```
## Warning: Removed 2 rows containing missing values (geom_bar).
```




```
candy %>% ggplot(aes(y = winpercent, x = chocolate_factor)) + geom_violin() + geom_jitter() +
  xlab('Chocolate') + ylim(0,100)
```




```
candy %>% ggplot(aes(y = winpercent, x = nut_factor)) + geom_violin() + geom_jitter() +
  xlab('Peanut or Almond') + ylim(0,100)
```



Linear Models Demo

The goal of this analysis will be to model the `winpercent` variables in the dataset. I'll use the `chocolate` and `peanutyalmondy` variables, but you are welcome to try your own covariates.

Linear Model Framework

Sampling Model

$$\tilde{Y} \sim N(X\tilde{\beta}, \sigma^2 I)$$

where $\tilde{Y}$ is a vector of length n = 85 where the i^{th} response is the winpercent of candy i,

$$X = \begin{bmatrix} 1 & chocolate_1 & peanutyalmondy_1 \\ 1 & chocolate_2 & peanutyalmondy_2 \\ \vdots & \ddots & \vdots \\ 1 & chocolate_n & peanutyalmondy_n \end{bmatrix}$$

Priors

$$\tilde{\beta} \sim N(\tilde{\beta_0}, \Sigma_0)$$

$$\sigma^2 \sim IG\left(\frac{\nu_0}{2}, \frac{\nu_0 \sigma_0^2}{2}\right)$$

0. `lm()`

```r
lm_candy <- lm(winpercent ~ chocolate_factor + nut_factor, data = candy)
summary(lm_candy)
```

```
##
## Call:
## lm(formula = winpercent ~ chocolate_factor + nut_factor, data = candy)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -26.0296  -7.6657   0.0797   7.3629  25.2130
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)         41.825      1.619  25.828  < 2e-16 ***
## chocolate_factor1   16.625      2.640   6.297 1.42e-08 ***
## nut_factor1          7.623      3.529   2.160   0.0337 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 11.17 on 82 degrees of freedom
## Multiple R-squared:  0.4372, Adjusted R-squared:  0.4235
## F-statistic: 31.85 on 2 and 82 DF,  p-value: 5.822e-11
```

```r
predict(lm_candy, data.frame(chocolate_factor = as.factor(c(1,1,0,0)), nut_factor = as.factor(c(1,0,1,0
```

```
##        1        2        3        4
## 66.07208 58.44926 49.44746 41.82464
```

- **Interpret the coefficients in this model**

1. Gibbs Sampler

```r
set.seed(10262018)
y <- candy$winpercent
X <- model.matrix(winpercent ~ chocolate_factor + nut_factor, data = candy)
p <- ncol(X)
n <- nrow(X)

# Initialization and Prior
num_mcmc <- 5000
beta_0 <- rep(0,p)
Sigma_0 <- diag(p) * 1000
Sigma_0_inv <- solve(Sigma_0)
nu_0 <- .02
sigmasq_0 <- 1
beta_samples <- matrix(0, nrow = num_mcmc, ncol = p)
sigmasq_samples <- rep(1, num_mcmc)

for (iter in 2:num_mcmc){
  # sample beta
  cov_beta <- solve(Sigma_0_inv + t(X) %*% X / sigmasq_samples[iter - 1])
  exp_beta <- cov_beta %*% (Sigma_0_inv %*% beta_0 + t(X) %*% y / sigmasq_samples[iter-1])
  beta_samples[iter,] <- rmnorm(1, exp_beta, cov_beta)
```

```r
  # sample sigmasq
  sigmasq_samples[iter] <- rigamma(1, .5 * (nu_0 + n) ,
          .5 * (nu_0 * sigmasq_0 + t(y - X %*% beta_samples[iter,]) %*%
                  (y - X %*% beta_samples[iter,])) )
}

burn_in <- 100
beta_samples[(burn_in+1):num_mcmc,] %>% as.mcmc() %>% summary()
```

```
##
## Iterations = 1:4900
## Thinning interval = 1
## Number of chains = 1
## Sample size per chain = 4900
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##        Mean    SD Naive SE Time-series SE
## [1,] 41.742 1.670  0.02385        0.02385
## [2,] 16.660 2.706  0.03866        0.03866
## [3,]  7.571 3.569  0.05098        0.05268
##
## 2. Quantiles for each variable:
##
##       2.5%    25%    50%    75% 97.5%
## var1 38.49 40.608 41.743 42.866 44.98
## var2 11.45 14.853 16.630 18.410 22.16
## var3  0.42  5.191  7.536  9.936 14.49
```

```r
beta_samples[(burn_in+1):num_mcmc,] %>% as.mcmc() %>% HPDinterval()
```

```
##           lower    upper
## var1 38.4224542 44.85806
## var2 11.2968299 21.89317
## var3  0.5692462 14.59080
## attr(,"Probability")
## [1] 0.95
```

```r
beta_samples[(burn_in+1):num_mcmc,] %>% as.mcmc() %>% effectiveSize()
```

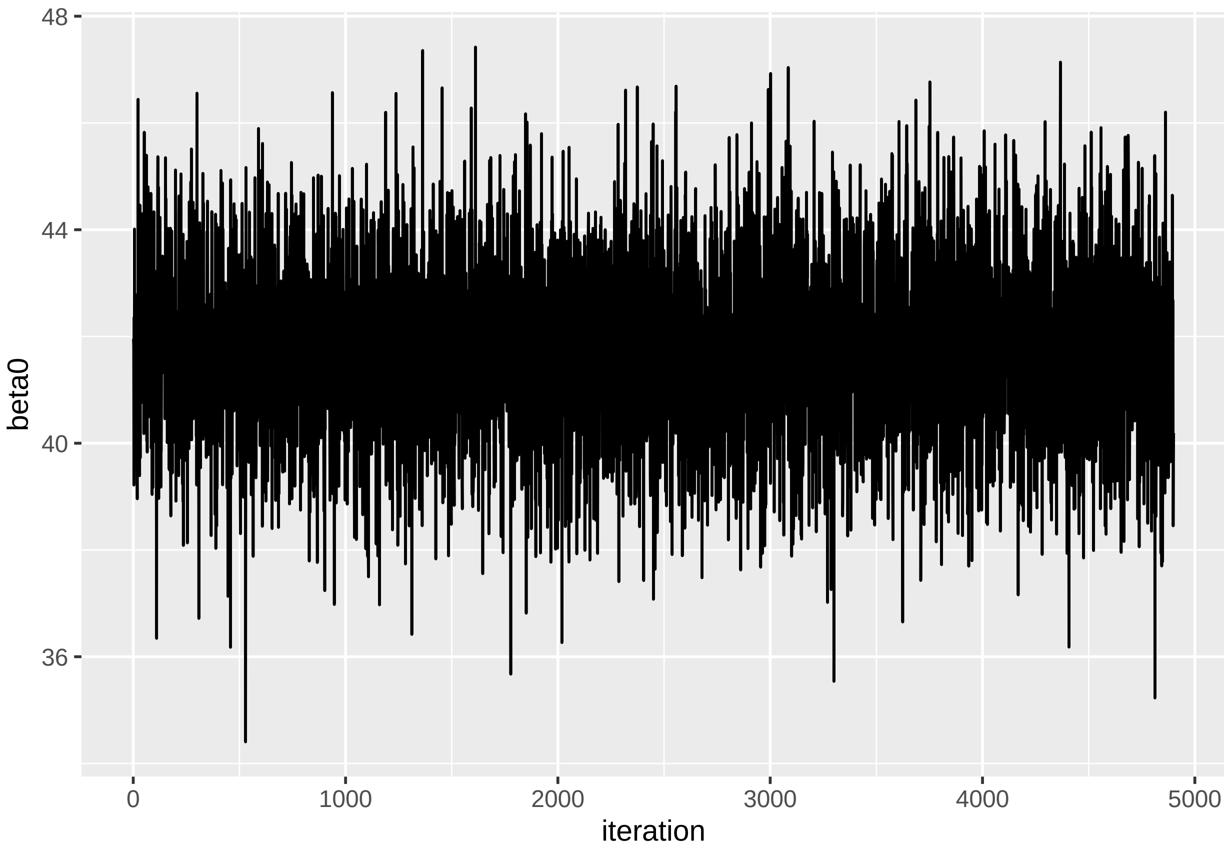
```
##     var1     var2     var3
## 4900.000 4900.000 4587.946
```

```r
sqrt(sigmasq_samples[(burn_in+1):num_mcmc]) %>% as.mcmc() %>% summary()
```

```
##
## Iterations = 1:4900
## Thinning interval = 1
## Number of chains = 1
## Sample size per chain = 4900
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##           Mean             SD       Naive SE Time-series SE
```

```
##        11.26077        0.88413        0.01263        0.01308
##
## 2. Quantiles for each variable:
##
##   2.5%    25%    50%    75%  97.5%
##  9.728 10.638 11.189 11.831 13.168
```

```
tibble(beta0 = beta_samples[(burn_in+1):num_mcmc,1], iteration = 1:(num_mcmc - burn_in)) %>%
  ggplot(aes(y = beta0,iteration)) + geom_line()
```



2. JAGS

Model Specification

```
model_string <- "model{
  # Likelihood
  for(i in 1:n){
    y[i]   ~ dnorm(mu[i],inv.var)
    mu[i] <- beta[1] + beta[2]*chocolate[i] + beta[3]*peanut[i]
  }


  # Note priors are hard-coded, but could be variables

  # Prior for beta
  for(j in 1:3){
    beta[j] ~ dnorm(0,0.001)
  }
```

```
  # Prior for the inverse variance
  inv.var   ~ dgamma(0.01, 0.01)
  sigma     <- 1/sqrt(inv.var)

}"
```

Compile in JAGS

```r
model <- jags.model(textConnection(model_string),
                    data = list(y = candy$winpercent,n = nrow(candy),
                                chocolate = candy$chocolate,
                                peanut=candy$peanutyalmondy))
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 85
##    Unobserved stochastic nodes: 4
##    Total graph size: 274
##
## Initializing model
```

Draw Samples

```r
# Burnin for 1000 samples
update(model, 1000, progress.bar="none")

samp <- coda.samples(model,
        variable.names=c("beta","sigma"),
        n.iter=5000, progress.bar="none")

summary(samp)
```

```
##
## Iterations = 1001:6000
## Thinning interval = 1
## Number of chains = 1
## Sample size per chain = 5000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##           Mean    SD Naive SE Time-series SE
## beta[1] 41.778 1.632  0.02308        0.03693
## beta[2] 16.633 2.621  0.03707        0.06064
## beta[3]  7.573 3.523  0.04983        0.06572
## sigma   11.260 0.894  0.01264        0.01330
##
## 2. Quantiles for each variable:
##
##            2.5%    25%    50%    75% 97.5%
## beta[1] 38.5361 40.697 41.774 42.889 45.00
## beta[2] 11.4550 14.893 16.674 18.386 21.81
## beta[3]  0.7091  5.263  7.518  9.928 14.56
```

```
## sigma    9.6811 10.633 11.199 11.842 13.16
```

```r
#plot(samp)
```

3. Stan

Model Statement

```
data {
  int<lower=0> N;
  vector[N] chocolate;
  vector[N] peanut;
  vector[N] y;
}
parameters {
  real beta0;
  real beta1;
  real beta2;
  real<lower=0> sigma;
}
model {
  y ~ normal(beta0 + beta1 * chocolate + beta2 * peanut, sigma);
}
```

```r
library(rstan)
```

```
## Loading required package: StanHeaders

## rstan (Version 2.18.2, GitRev: 2e1f913d3ca3)

## 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)

##
## Attaching package: 'rstan'

## The following object is masked from 'package:coda':
##
##     traceplot
```

```r
post <- rstan::sampling(lm_stan,
             data = list(y = candy$winpercent, N = nrow(candy), peanut = candy$peanutyalmondy, chocolat
```

```
##
## SAMPLING FOR MODEL '741bdbb7939f755ffcc8d9dfc2e48905' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 2.3e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.23 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: 0.093347 seconds (Warm-up)
## Chain 1:                0.073123 seconds (Sampling)
## Chain 1:                0.16647 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL '741bdbb7939f755ffcc8d9dfc2e48905' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 9e-06 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.09 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: 0.098196 seconds (Warm-up)
## Chain 2:                0.052056 seconds (Sampling)
## Chain 2:                0.150252 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL '741bdbb7939f755ffcc8d9dfc2e48905' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 9e-06 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.09 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: 0.105443 seconds (Warm-up)
## Chain 3:                0.058578 seconds (Sampling)
## Chain 3:                0.164021 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL '741bdbb7939f755ffcc8d9dfc2e48905' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 8e-06 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.08 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: 0.098213 seconds (Warm-up)
## Chain 4:                0.054931 seconds (Sampling)
## Chain 4:                0.153144 seconds (Total)
## Chain 4:
```

```
post
```

```
## Inference for Stan model: 741bdbb7939f755ffcc8d9dfc2e48905.
## 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
## beta0   41.89    0.03 1.67   38.67   40.73   41.87   43.02   45.22  2946
## beta1   16.62    0.05 2.73   11.36   14.76   16.57   18.47   21.99  2807
## beta2    7.50    0.06 3.52    0.66    5.22    7.52    9.84   14.24  3349
## sigma   11.35    0.02 0.89    9.72   10.74   11.32   11.91   13.23  3355
## lp__  -245.77    0.03 1.45 -249.43 -246.45 -245.45 -244.73 -243.95  1899
##       Rhat
## beta0    1
## beta1    1
## beta2    1
## sigma    1
## lp__     1
##
## Samples were drawn using NUTS(diag_e) at Tue Nov  5 13:51:04 2019.
```

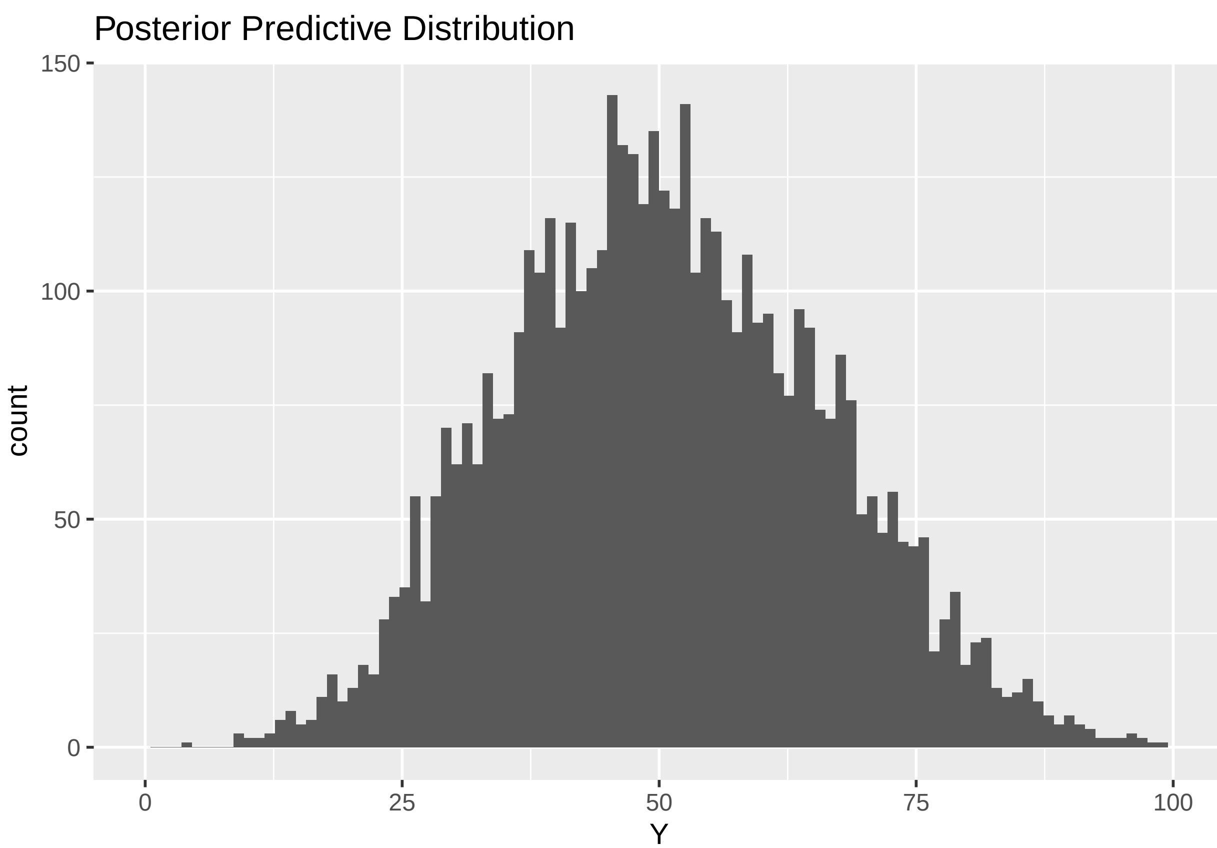
```
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

Posterior Predictive Distribution

A quote from the Bayesian Data Analysis textbook highlights the goal of model checking. >> We do not like to ask: 'Is our model true or false?', since probability models in most data analyses will not be perfectly true... The more relevant question is 'Do the model's deficiencies have a noticeable effect on the substantive inference?'

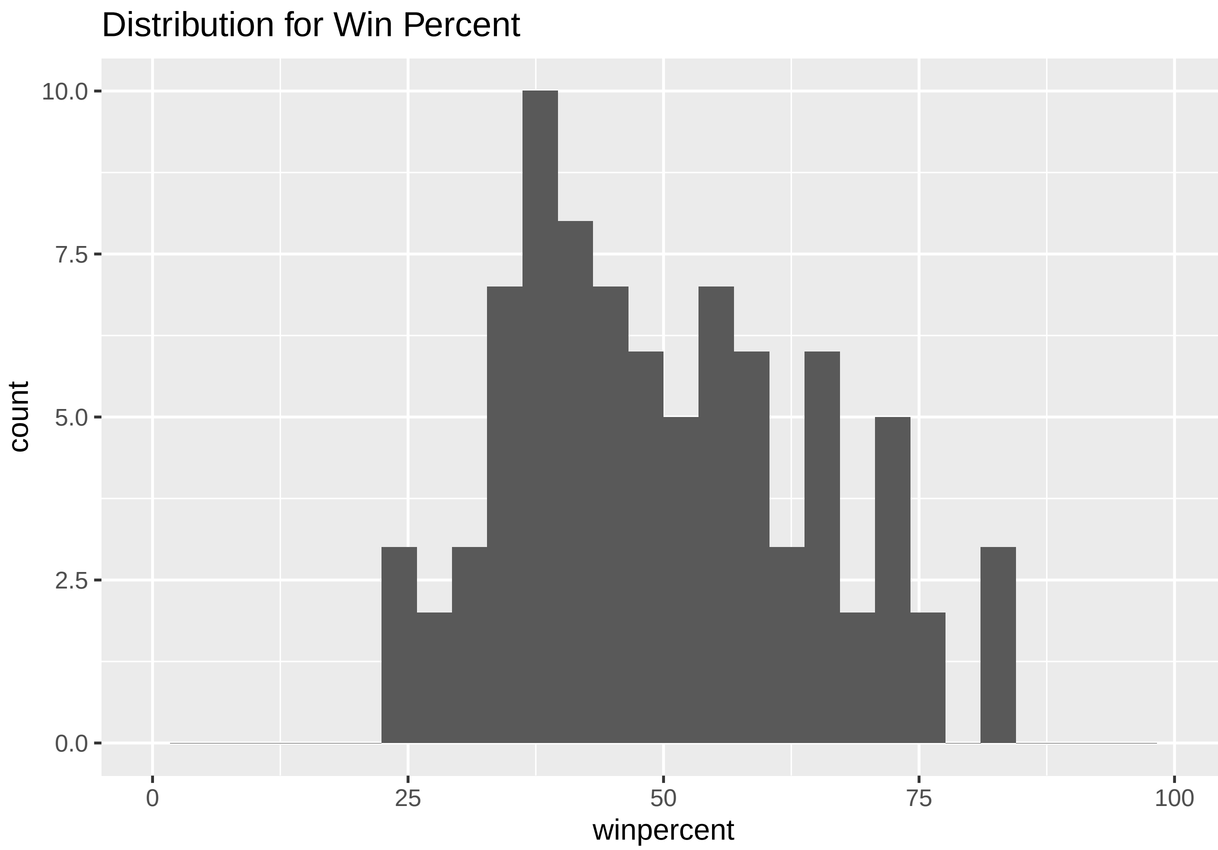
```
Y <- rep(0, (num_mcmc - burn_in))
for (i in 1:(num_mcmc - burn_in)){
  Y[i] <- rnorm(1,  beta_samples[i+burn_in,] %*% X[sample(n,1),], sqrt(sigmasq_samples[i+burn_in]))
}

tibble(Y = Y) %>% ggplot(aes(x = Y)) + geom_histogram(bins = 100) + ggtitle('Posterior Predictive Distr
  xlim(0,100)
```




```
candy %>% ggplot(aes(x = winpercent)) + geom_histogram() + ggtitle('Distribution for Win Percent') +
  xlim(0,100)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```


Distribution for Win Percent
10.0
7.5
5.0
2.5
0.0
count
0
25
50
75
100
winpercent