HW 10 Key

This homework will reuse the candy dataset, not with the goal of predicting whether a candy contains chocolate.

Recall: 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

```
candy <- read_csv('http://math.montana.edu/ahoegh/teaching/stat446/candy-data.csv')
candy <- candy %>% mutate(chocolate_factor = as.factor(chocolate))
```

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.

1 (3 points)

Use the `glm()` function to fit a regression model to understand if other factors can be used to model whether a candy item contains chocolate. Make sure to include `winpercent` but you are free to consider other variables too. Interpret the output.

```
glm.mod <- glm(chocolate ~ winpercent, data = candy, family = binomial)
summary(glm.mod)
```

```
##
## Call:
```

```
## glm(formula = chocolate ~ winpercent, family = binomial, data = candy)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.0276  -0.6559  -0.2933   0.5687   2.2495
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -7.13787    1.48568  -4.804 1.55e-06 ***
## winpercent   0.13510    0.02865   4.716 2.41e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 116.407  on 84  degrees of freedom
## Residual deviance:  74.795  on 83  degrees of freedom
## AIC: 78.795
##
## Number of Fisher Scoring iterations: 5
```

2. (8 points)

Using Stan implement Bayesian logistic regression.

a. (2 points)

Write out the model, including priors. Note you did this in the last quiz.

$$y_i|\tilde{x}_i, \tilde{\beta} \sim Bernoulli(\theta_i)$$
$$\tilde{\beta}^T \tilde{x}_i = log\left(\frac{\theta_i}{1-\theta_i}\right)$$
$$\tilde{\beta} = (\alpha, \beta)$$

where the prior distributions for α and β are: $p(\alpha) \sim N(0, 1000)$ and $p(\beta) \sim N(0, 1000)$, y_i represents whether the i^{th} candy contains chocolate, and $\tilde{x}$ is a matrix that contains the intercept and the `winpercent` values for each candy.

b. (3 points)

Implement Bayesian logistic regression using Stan. This Stan link will be helpful.

```
data {
  int<lower=0> N;
  vector[N] winpercent;
  int<lower=0,upper=1> y[N];
}
parameters {
  real alpha;
  real beta;
}
model {
```

```
  alpha ~ normal(0, 1000);
  beta ~ normal(0, 1000);
  y ~ bernoulli_logit(alpha + beta * winpercent);
}
```

```
library(rstan)
post <- rstan::sampling(stan.model,
             data = list(y = candy$chocolate, N = nrow(candy), iter = 4000,
                         winpercent = candy$winpercent))
```

```
##
## SAMPLING FOR MODEL '55332fc85f8420af5ec01932db1f41e0' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 4.5e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.45 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: 0.237636 seconds (Warm-up)
## Chain 1:                0.139503 seconds (Sampling)
## Chain 1:                0.377139 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL '55332fc85f8420af5ec01932db1f41e0' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 1e-05 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.1 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: 0.267642 seconds (Warm-up)
## Chain 2:                0.115728 seconds (Sampling)
## Chain 2:                0.38337 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL '55332fc85f8420af5ec01932db1f41e0' 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: 0.211019 seconds (Warm-up)
## Chain 3:                0.118117 seconds (Sampling)
## Chain 3:                0.329136 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL '55332fc85f8420af5ec01932db1f41e0' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 1.3e-05 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.13 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: 0.225896 seconds (Warm-up)
## Chain 4:                0.107946 seconds (Sampling)
## Chain 4:                0.333842 seconds (Total)
```

```
## Chain 4:
```

```
post
```

```
## Inference for Stan model: 55332fc85f8420af5ec01932db1f41e0.
## 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 Rhat
## alpha  -7.48    0.06 1.48 -10.38  -8.52  -7.42  -6.44  -4.77   604    1
## beta    0.14    0.00 0.03   0.09   0.12   0.14   0.16   0.20   610    1
## lp__  -38.39    0.03 0.94 -40.88 -38.82 -38.11 -37.70 -37.43  1125    1
##
## Samples were drawn using NUTS(diag_e) at Wed Nov 27 15:08:15 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).
```

c. (3 points)

Interpret your output and explain the results without statistical jargon.

The model suggests that the odds that a piece of candy contains chocolate increases as win percentage increases, more specifically, for a 1 percent increase in win percentage, there is a 95% chance that the odds of a piece of candy being chocolate is between 1.094 and 1.234 times that of piece of candy with a 1 percent lower win percentage. This suggests that win percentage is a somewhat useful variable in determining whether a candy contains chocolate.

3. (9 points)

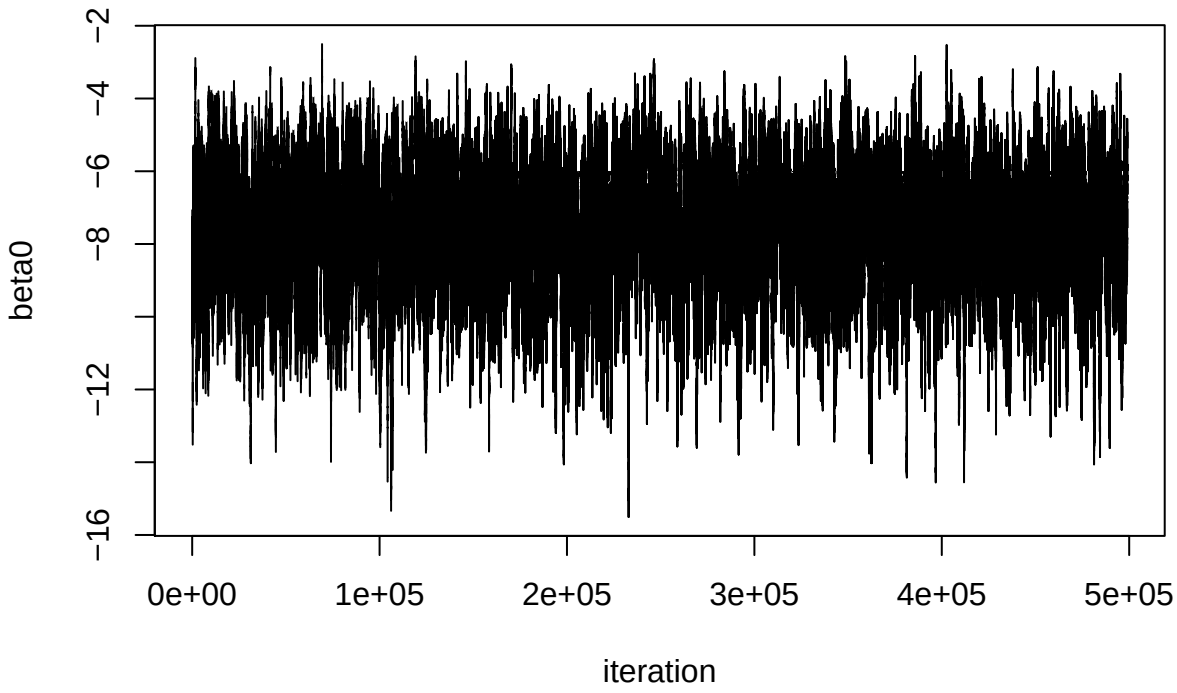
Building on Quiz 9. Write your own Metropolis-Hastings sampler to estimate the parameters in the model specified in Q2a. Print out trace plots for a few of your variables. Note your answers should be very similar to those found with Stan.

```
library(mnormt) # rmnorm
set.seed(11272019)

# Data from sample
X <- model.matrix(chocolate_factor ~ winpercent, data = candy)
y <- candy$chocolate
n <- length(y)
p <- dim(X)[2]

# Run Metropolis Algorithm
num_mcmc <- 500000
step_size <- c(1,0.02)
accept_ratio <- matrix(0,nrow = num_mcmc, ncol = p)
beta_mcmc <- matrix(0,num_mcmc,p)
#beta_mcmc[1,] <- coef(glm.mod)
beta_prior_var <- diag(p)*1000
beta_current <- beta_mcmc[1,]

for (i in 2:num_mcmc){
  for (j in 1:p){
    beta_star <- beta_current
```

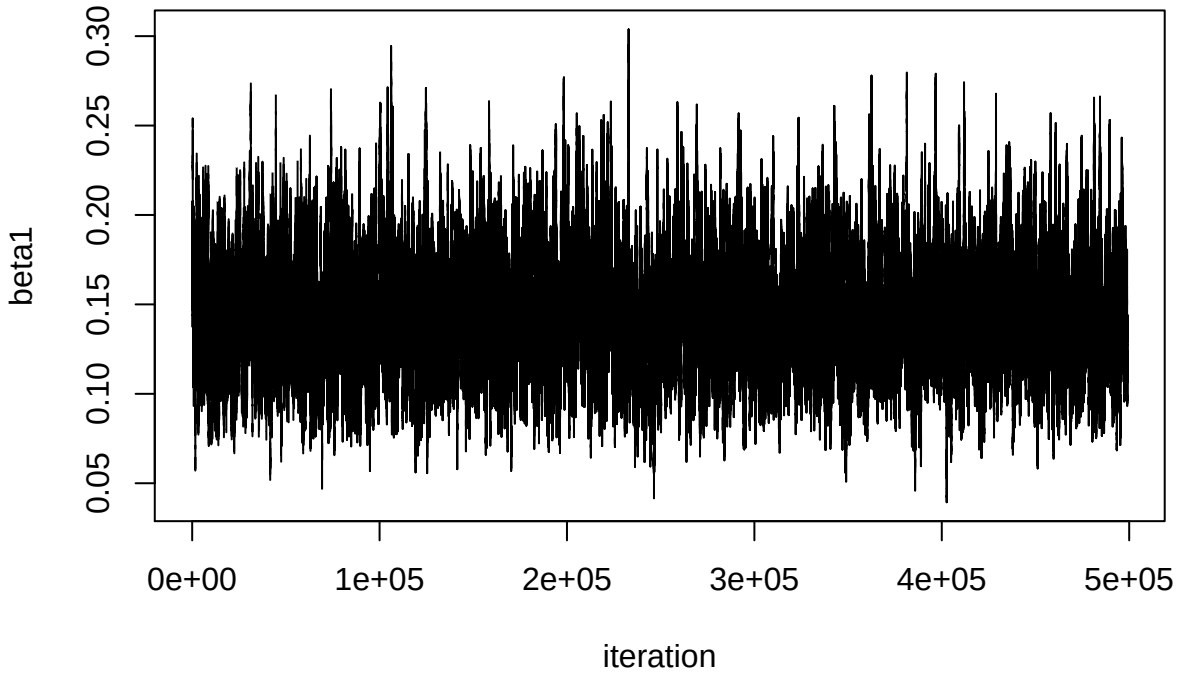
```
    beta_star[j] <- beta_star[j] + rnorm(1, 0, step_size[j])

    #compute r
    theta_current <- exp(X %*% beta_current)/(1+exp(X %*% beta_current))
    theta_star <- exp(X %*% beta_star)/(1+exp(X %*% beta_star))

    log_pi_current <- sum(dbinom(y,1,theta_current,log=T))
    log_pi_star <- sum(dbinom(y,1,theta_star,log=T))

    log_r <- log_pi_star - log_pi_current

    if (log(runif(1)) < log_r){
      beta_mcmc[i,] <- beta_star
      accept_ratio[i,j] <- 1
      beta_current <- beta_star
    } else{
      beta_mcmc[i,] <- beta_current
    }
  }
}

colMeans(accept_ratio)
```

```
## [1] 0.338878 0.328248
```

```
burnin <- 1000
plot(beta_mcmc[(burnin+1):num_mcmc,1],type='l',xlab="iteration",ylab="beta0")
```




```
plot(beta_mcmc[(burnin+1):num_mcmc,2],type='l',xlab="iteration",ylab="beta1")
```




```
summary(as.mcmc(beta_mcmc[(burnin+1):num_mcmc,]))
```

```
##
## Iterations = 1:499000
## Thinning interval = 1
## Number of chains = 1
## Sample size per chain = 499000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##          Mean      SD  Naive SE Time-series SE
## [1,] -7.5579 1.55875 0.0022066      0.0356354
## [2,]  0.1433 0.03009 0.0000426      0.0006887
##
## 2. Quantiles for each variable:
##
##           2.5%     25%     50%     75%   97.5%
## var1 -10.85650 -8.5452 -7.4701 -6.4731 -4.7652
## var2   0.08962  0.1223  0.1415  0.1622  0.2072
```

```
effectiveSize(as.mcmc(beta_mcmc[(burnin+1):num_mcmc,]))
```

```
##     var1     var2
## 1913.333 1909.015
```