Mets Interview - Andre Zapico

Question 1:

```r
require(rstan);
require(cmdstanr);
require(DescTools);

set.seed(1);
```

Question and Tools

Here, I am requested to develop a model that forecasts whether something was a strike or whether the batter swings or not. Although I haven't been using Stan latelty, my go-to modeling language is Stan. Although using one-linear modeling functions in base-R, for example, lm or the package "glmer," may be faster, modeling in Stan is great because when forecasting, if your forecasting accuracy isn't great, you can add additional information that can make the model more accurate and interpretable, which you cannot do with classical R statistical packages or neural networks. There are optimization features available, that estimate uncertainty using analytic computations as opposed to MCMC to estimate parameters, which gives drastic speed ups.

Classical analysis is easy for interpretation, for example, whether a coefficient is statistically significant, and associated with an outcome of interest. Bayesian analysis can be interpreted similarly, at least in the linear regression case. However, using a modeling language such as Stan, allows for additional flexibility. If we need to implement a customized model, for example, something like an HMM with a generalized Poisson likelihood function, we are not dependent on existing software developed as a result of a research paper, we can simply add to the model script, as opposed to having to find additional software. Using Stan, we can swap the likelihood, add a non-linear (GP) prior, add additional hierarchies, etc.

Here's an example. During my masters, consulting I was working with a grad student modeling marketing data, attempting to "gamify" an education phone application. He was using a hurdle-HMM, once probability was sufficient, he was attempting to discover states (using an HMM) at which the user would use the app longer or shorter. Using all of the data at once, the model fit diagnostics weren't working. After some discussion and thinking about the generative process (i.e. what was motivating the user to hold onto the phone longer), and we added different states which removed divergences, a Hamiltonion Monte Carlo (HMC) model fit diagnostic, that indicates your modeling assumptions are not accurate, and you should consider modifying your model.

Note: I forgot to set some priors and some bad priors. The first 2025 prediction has a really strong initial covariance in between players and it's causing shrinkage, the next model should take care of that, and it is additive

The Basics

Admittedly, I should do a lot more exploratory analysis, but here I kind of got straight to the modeling. And, I could do more posterior predictive checks and validation and add parameters, but I hope you get the point.

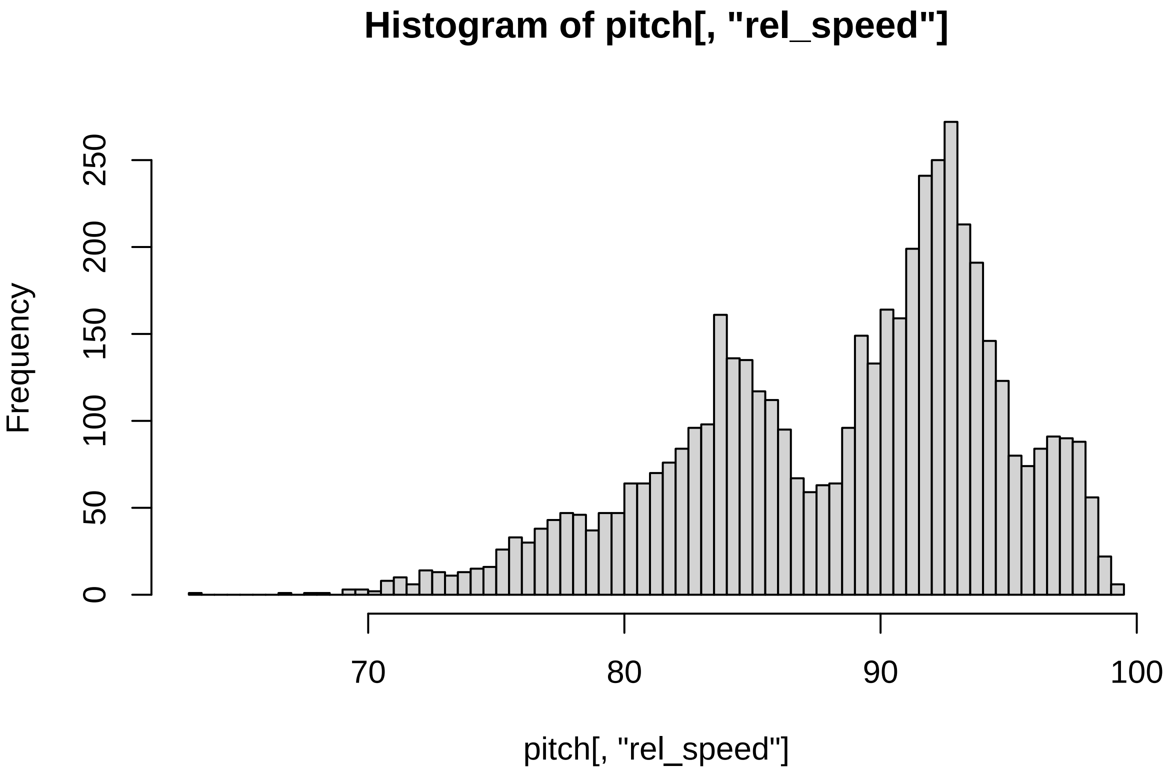
We're asked to model a binary outcome. The go-to is a logistic regression model.

$$p(y_i) \sim \frac{1}{1+e^{-\pi_x}}$$

where π could be an affine function, that is conditional on parameters π and data x, for example: $\pi_x := \beta_0 + \mathbf{B}'X$, where X is design matrix and $\mathbf{B}$ is a vector of parameters. This is the first model we try. We start simple, and build up. As an initial test, I just fit strike as the outcome, and then swing, but was getting model fit, issues, so I added speed, with some conditionals. I also downsampled the data, only so that models run faster. This shows I know how to make conditional predictions. The model is trained on speed in the interval 88 and 92.

Retrospectively I should have used all of speed, and then only made forecasts with just the conditional. But anyway, data preparation and Stan code are below.

```
setwd("C:/Users/andre/likely/mets/tech_interview/");


## run model command:
require(rstan);
require(cmdstanr);
require(DescTools);

set.seed(1);
standardize = function(x) {
  x_mu = mean(x);
  x_sd = sd(x);
  x_std = (x - x_mu) / x_sd;
  return(list(x_std, x_mu, x_sd));
}

pitch_raw = read.csv("pitch_data.csv");


## TODO: Downsample
nrow_praw = nrow(pitch_raw);
ds = sample(nrow_praw, size = 5000);
pitch = pitch_raw[1:5000,];
n_pitch = nrow(pitch);

## some exploratory analysis
hist(pitch[, "rel_speed"], breaks = 100); ## sweet something to model (GP prior)
```



```r
## standardize after downsampling
rel_speed_std = standardize(pitch[,"rel_speed"]);
pitch = cbind(pitch, rel_speed_std[[1]]);


## split train/test 75%/25%
## later, I'd do cross validation, but
## this is just because this is for speed,
## for a demo

ind = 1:n_pitch;
train_ind = sample(ind, size = floor(.75 * n_pitch));
test_ind = ind[-train_ind];

# separate data into random test/train sets, .75/.25
train = pitch[train_ind,];
test = pitch[test_ind,];

## dummy model

rel_speed_std = standardize(train[, "rel_speed"]);
## make design matrix
X = cbind(train[,c("is_swing", "rel_speed_std[[1]]")]);

## format data
y = train[, "is_strike"];
```

```
N = nrow(train);
M = ncol(X);
X_pred = test[, c("is_swing", "rel_speed_std[[1]]")];
N_pred = nrow(test);

#dt = list(y = y, X = X,  N = N, M = M, X_pred = X_pred, N_pred = N_pred);
#write_stan_json(dt, "//wsl.localhost/Ubuntu/home/andre/likely/mets/tech_interview/regr1_1_5000_no_avg.
```

Input Data, Question One

In the above block of code, I'm doing the following: reading in the dataset, I have a standardize function that reduces range of the input data of continuous covariates so that it's more compatable with MCMC algorithms, and we can also recover the orginal scale later for that it's interpretable. I'm also downsampling the data, so there's less FLOPS (floating point operations) in any internal algorithm, and it runs faster. I combine this into a design matrix, as well as number of observations and prediction output and the code for generating and testing out of sample predictions. Below is the Stan code, which is pretty straight forward.

And the Stan code for a logistic regression, using speed:

```
data {
  int N;
  int M;
  int N_pred;
  matrix[N, M] X;
  array[N] int y;
  matrix[N_pred, M] X_pred;
}
parameters {
  real beta0;
  vector[M] beta;
}
model {
  y ~ bernoulli_logit(beta0 + X * beta);
}
generated quantities {
  vector[N] p_pred_in;             // probability p from model in sample
  vector[N] p_pred_out;             // probability p from model out of sample
  array[N] int y_pred_in;       // in sample predictions
  array[N_pred] int y_pred_out; // out of sample predictions

  for (i in 1:N) {
    p_pred_in[i] = inv_logit(beta0 + X[i, ] * beta);
    y_pred_in[i] = bernoulli_logit_rng(beta0 + X[i, ] * beta);
  }

  for (i in 1:N_pred) {
   p_pred_out[i] = inv_logit(beta0 + X_pred[i, ] * beta);
   y_pred_out[i] = bernoulli_logit_rng(beta0 + X_pred[i, ] * beta);
  }
}
```

Results

This question answers question one, although accuracy could be improved with further modeling. I am starting simple in order to put out a model as quickly as possible. With model expansion, we can improve the accuracy and interpretability of the model. I can proceed with model developments in the future, and I'll explain future modeling expansion capabilities, and I can develop these in the future. My time is limited so I'm giving simple examples.

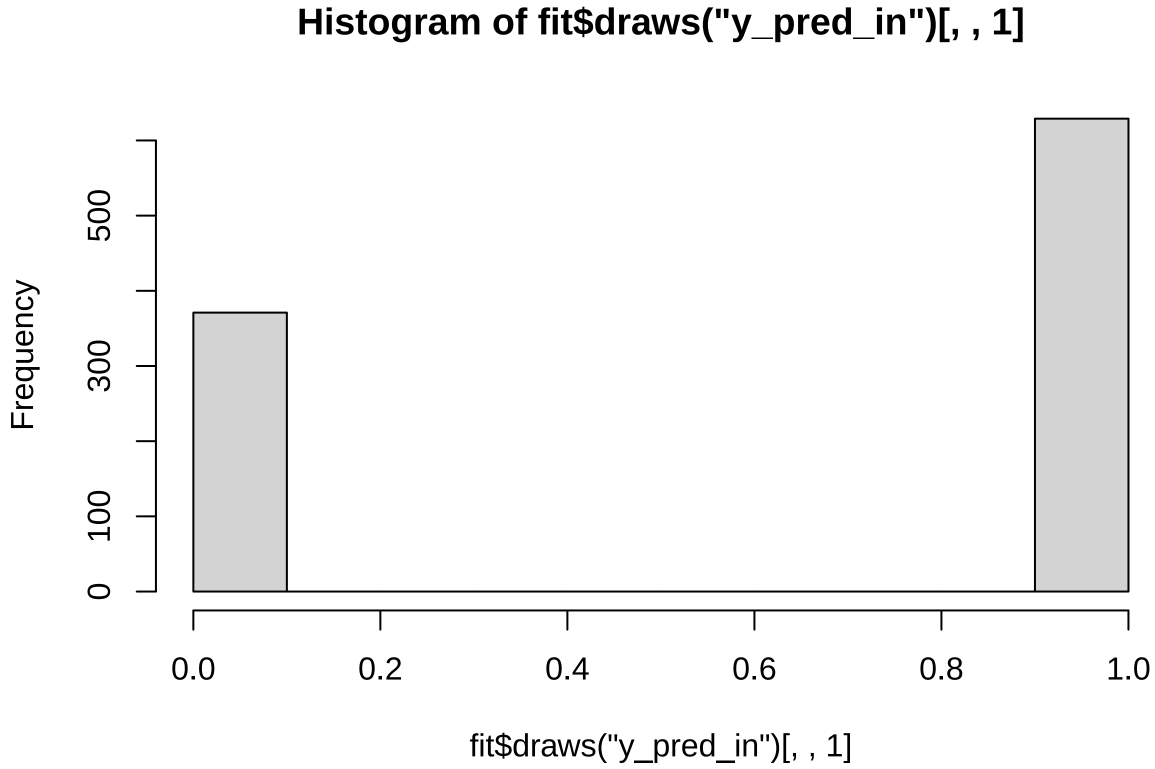
For training, I downsampled. For predictions, I made conditional predictions, with speed $(88, 92)$ miles per hour. To make predictions you can loop through the data.

Here's the R code and some plots.

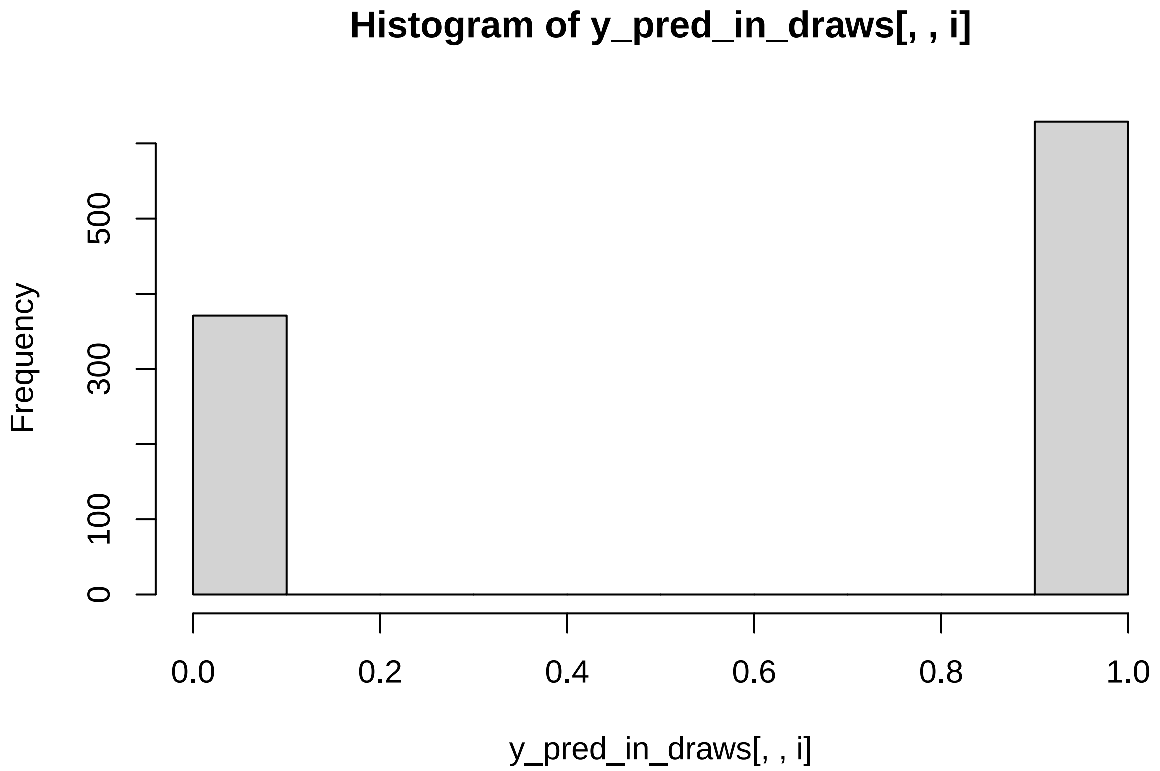
```
fit = cmdstanr::as_cmdstan_fit("output_preds_5000.csv");

dim(fit$draws("y_pred_in"));
```

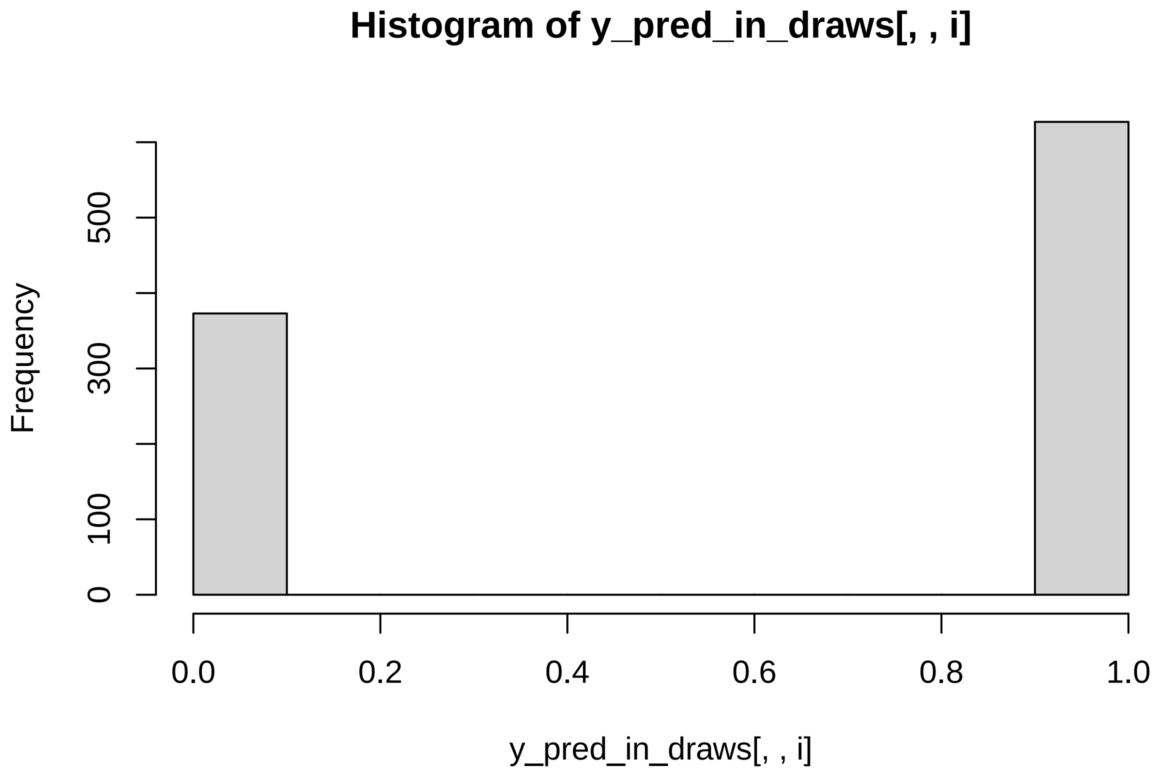
```
## [1] 1000    1 3750
```

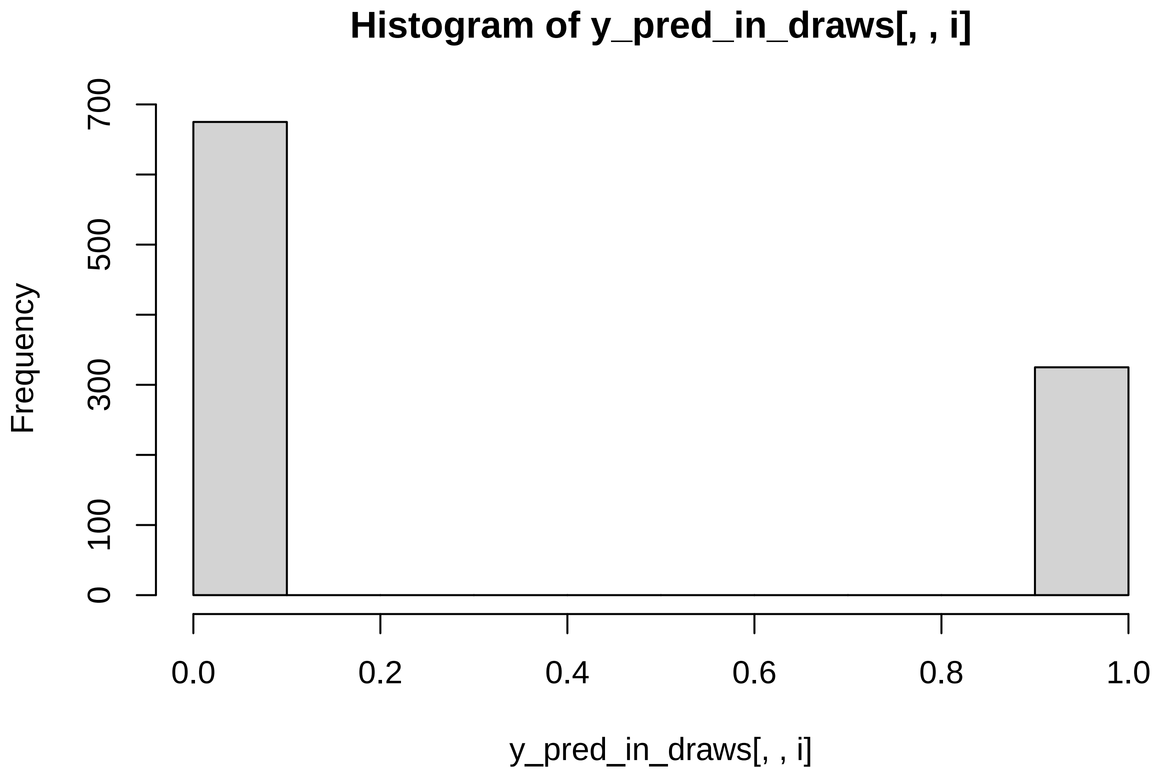
```
y_pred_in_draws = fit$draws("y_pred_in");

hist(fit$draws("y_pred_in")[,,1]); ## predictions for the first observation (and we can get p, not p)
```



```
## first 3 prediction from a RNG (you can change the predictions)
for (i in 1:3) {
  hist(y_pred_in_draws[,,i], hist = 100);
}
```


Histogram of y_pred_in_draws[, , i]
Frequency
0
100
300
500
0.0
0.2
0.4
0.6
0.8
1.0
y_pred_in_draws[, , i]

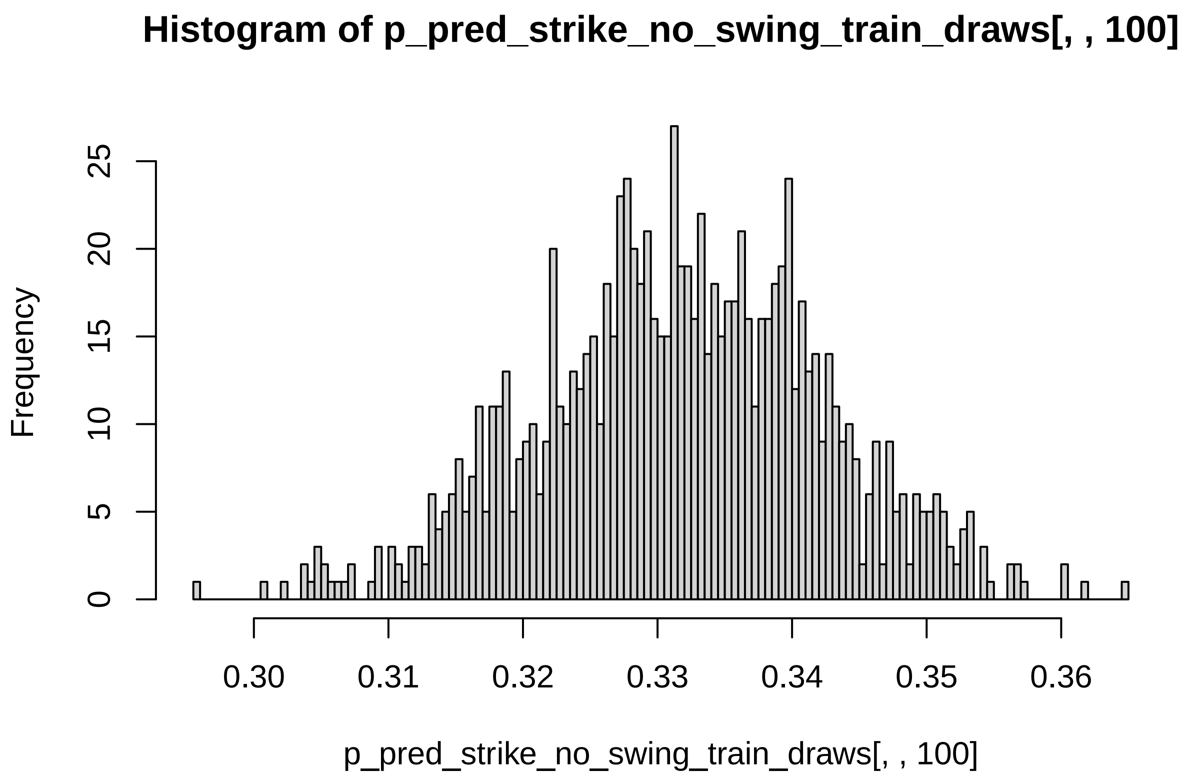

Histogram of y_pred_in_draws[, , i]
Frequency
0
100
300
500
0.0
0.2
0.4
0.6
0.8
1.0
y_pred_in_draws[, , i]



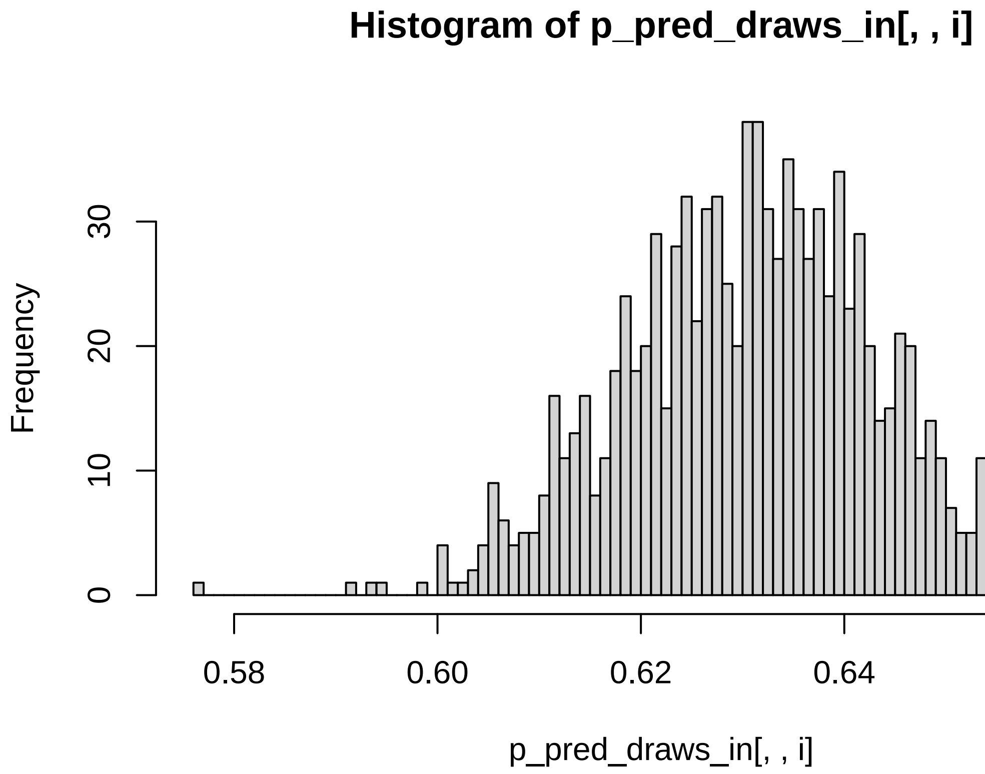
```r
## train/test which swing or not
swing_in = which(train$is_swing == 1);
no_swing_in = which(train$is_swing == 0);

swing_out = which(test$is_swing == 1);
no_swing_out = which(test$is_swing == 0);

## probability strike | no swing,
speed_min = 88; speed_max = 92;
speed_min_std = (speed_min - rel_speed_std[[2]]) / rel_speed_std[[3]];
speed_max_std = (speed_min - rel_speed_std[[2]]) / rel_speed_std[[3]];

## in sample p
p_strike_no_swing_train = train[which(train$is_swing == 0 & train$rel_speed < speed_max & train$rel_spe
w_p_strike_no_swing_train = which(train$is_swing == 0 & train$rel_speed < speed_max & train$rel_speed >

## out sample p
p_strike_no_swing_test = test[which(test$is_swing == 0 & test$rel_speed < speed_max & train$rel_speed >
w_p_strike_no_swing_test = which(test$is_swing == 0 & test$rel_speed < speed_max & test$rel_speed > spe

## extract in sample p for this subset, conditional data:
p_pred_draws_in = fit$draws("p_pred_in");

p_pred_strike_no_swing_train_draws = p_pred_draws_in[,,w_p_strike_no_swing_train];
dim(p_pred_strike_no_swing_train_draws);
```

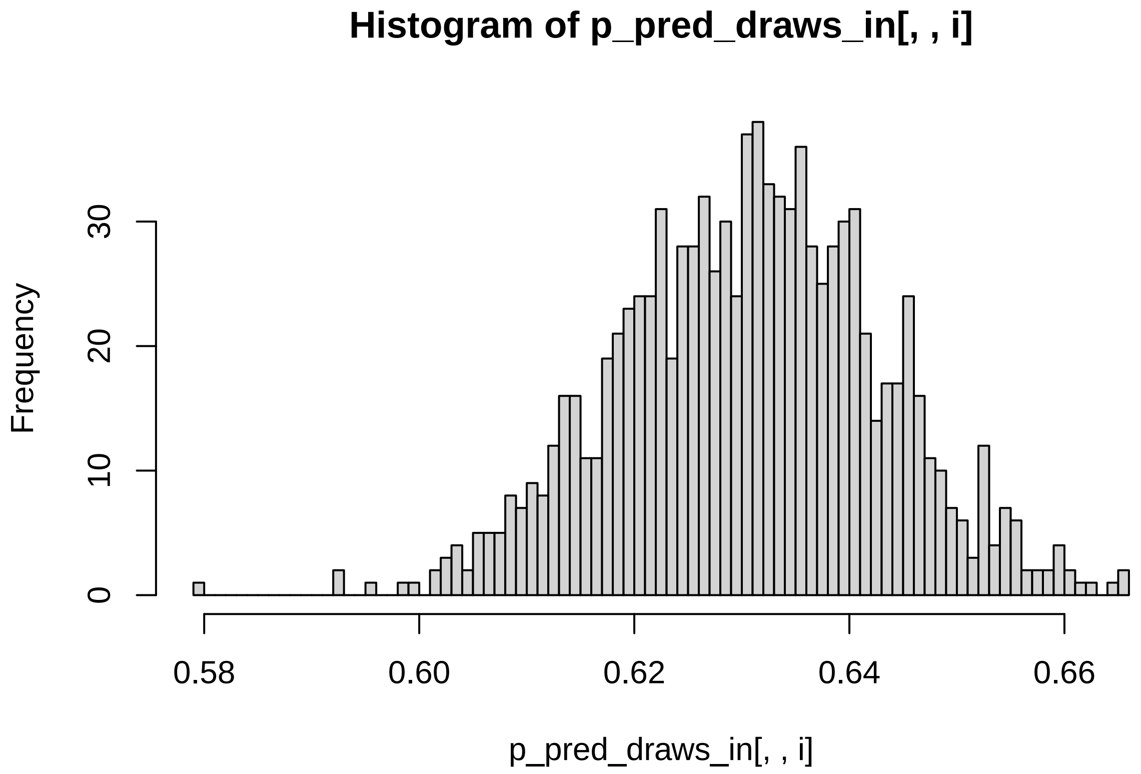
```
## [1] 1000    1  515
```

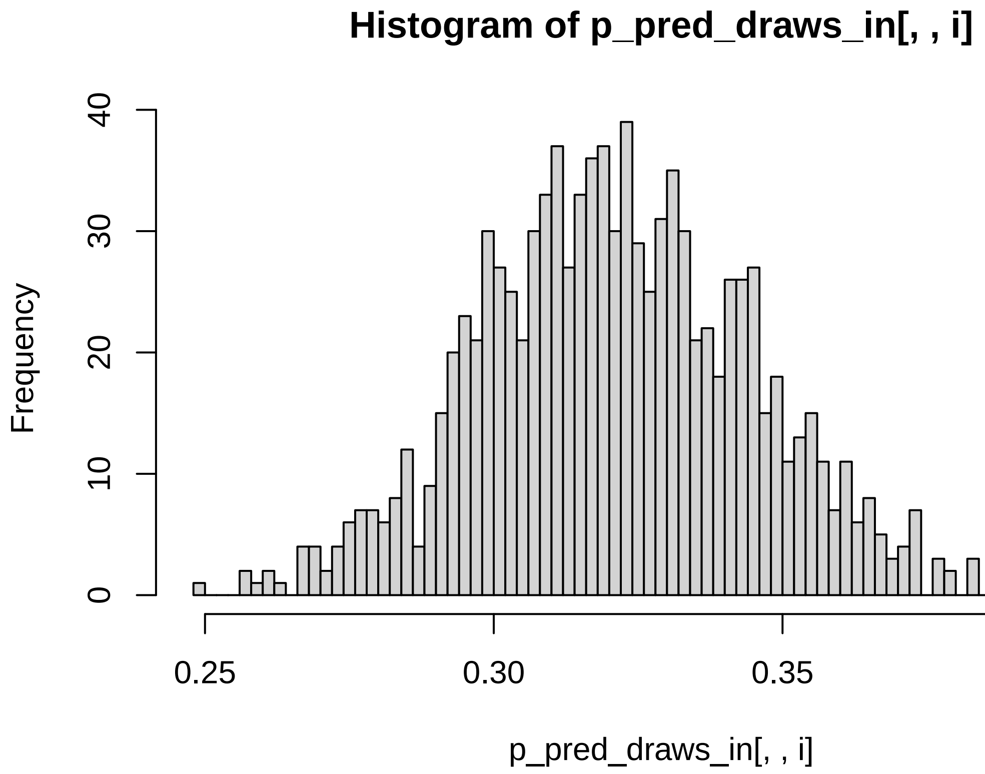
```
hist(p_pred_strike_no_swing_train_draws[,,100], breaks = 100); ## p for the ith observation
```



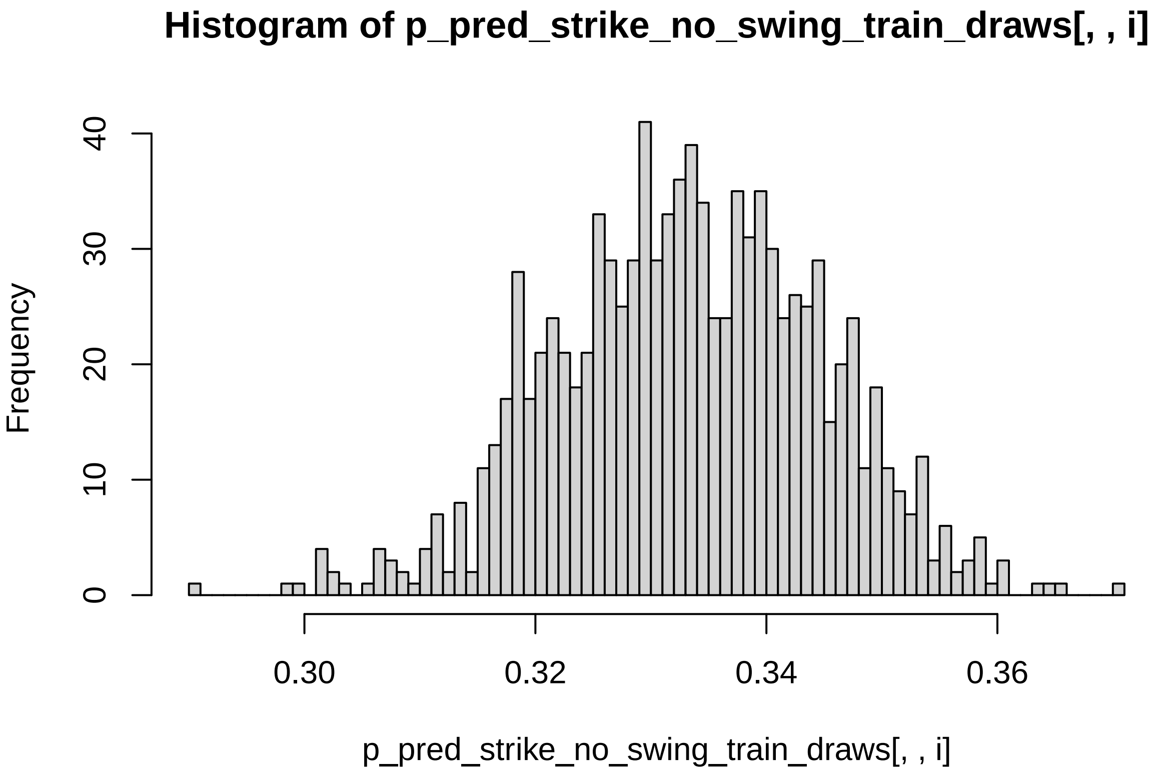
```
#################
## p_pred draws hist based on no conditions
for (i in 1:3) { ## it switches around
  hist(p_pred_draws_in[,,i], breaks = 100);
}
```

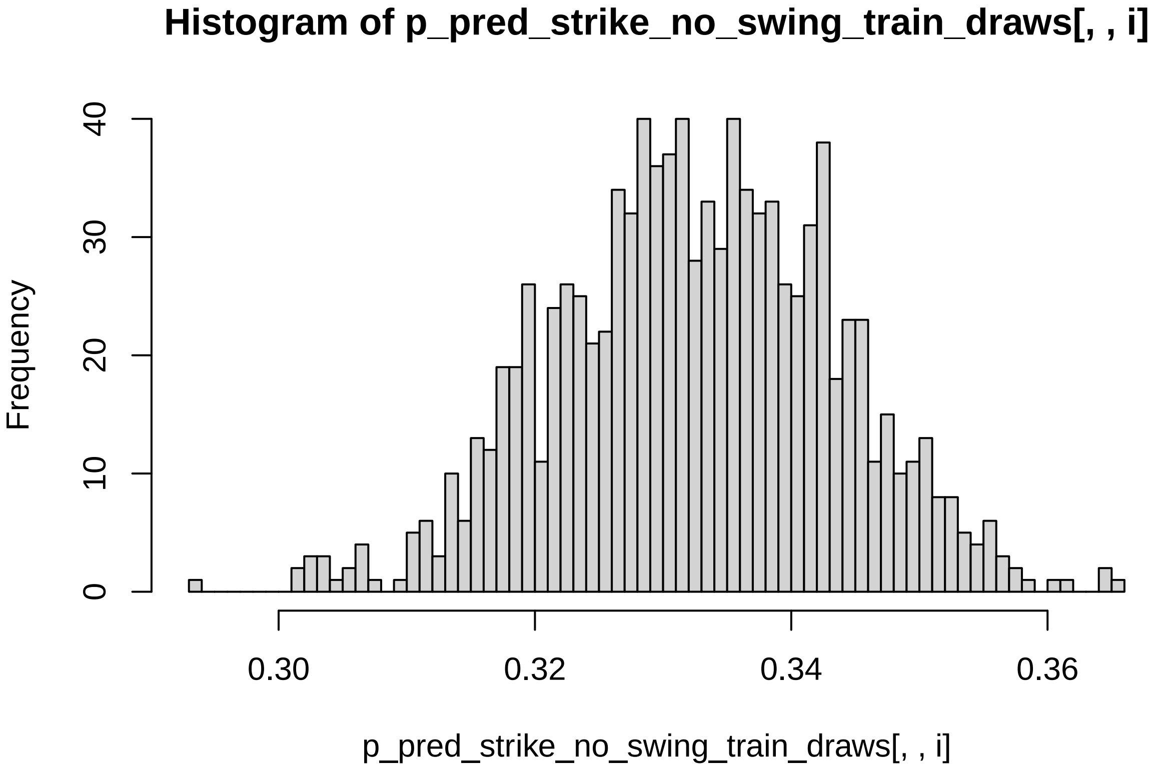

Histogram of p_pred_draws_in[, , i]
Frequency
0
10
20
30
0.58
0.60
0.62
0.64
0.66
p_pred_draws_in[, , i]

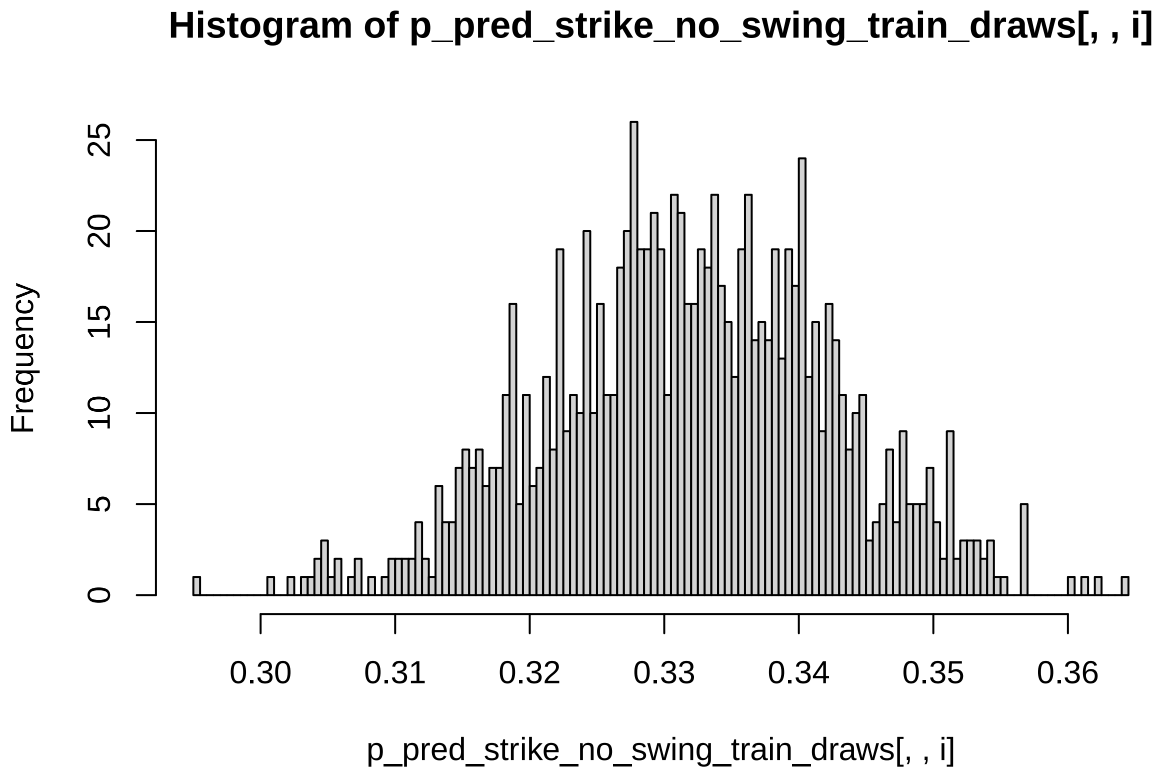

Histogram of p_pred_draws_in[, , i]
Frequency
0
10
20
30
0.58
0.60
0.62
0.64
0.66
p_pred_draws_in[, , i]



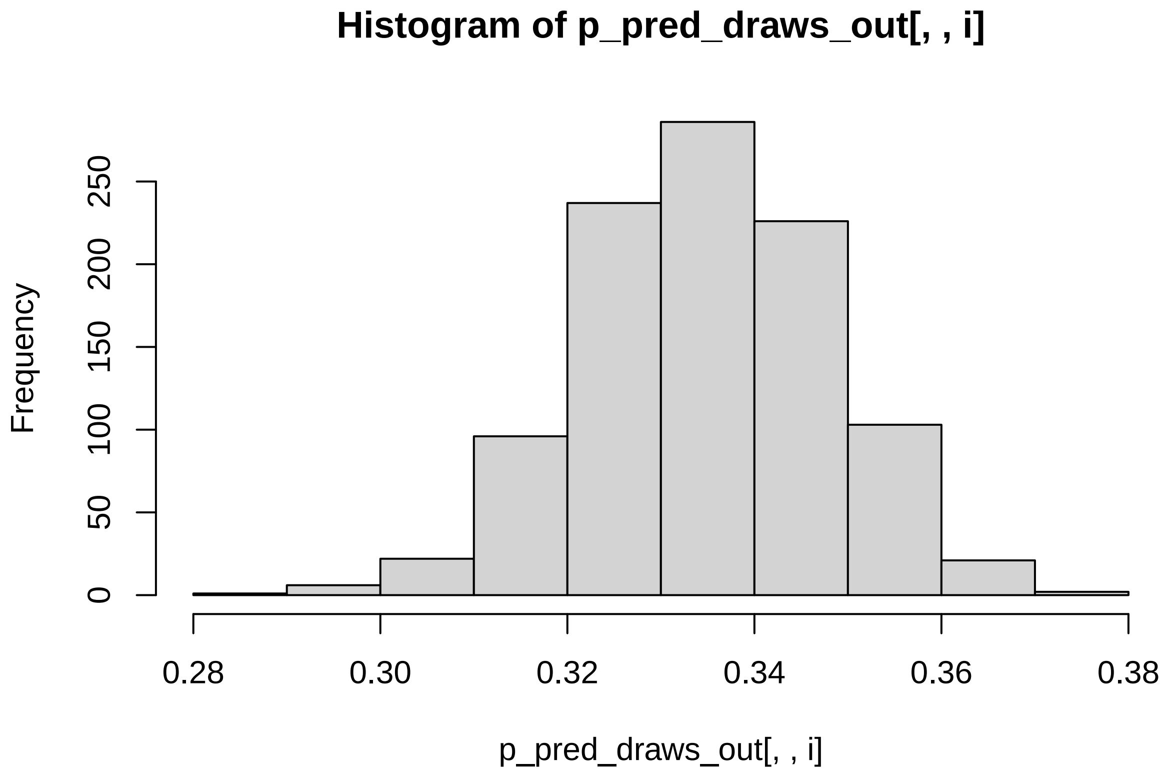
```
### p based on above conditions
## p for speed conditions and no swing, in sample
for (i in 1:3) { ## first few observations, p should change
  hist(p_pred_strike_no_swing_train_draws[,,i], breaks = 100); ## it's usually around 30%
}
```

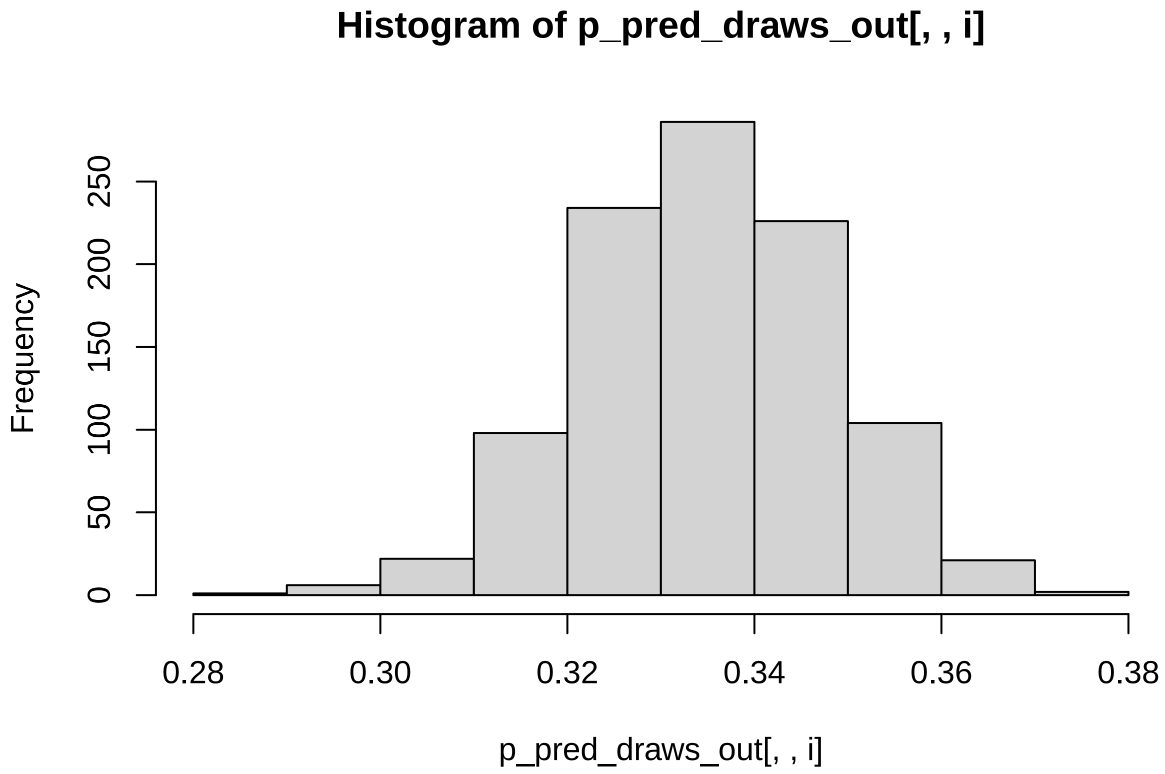

Histogram of p_pred_strike_no_swing_train_draws[, , i]
Frequency
0
10
20
30
40
0.30
0.32
0.34
0.36
p_pred_strike_no_swing_train_draws[, , i]

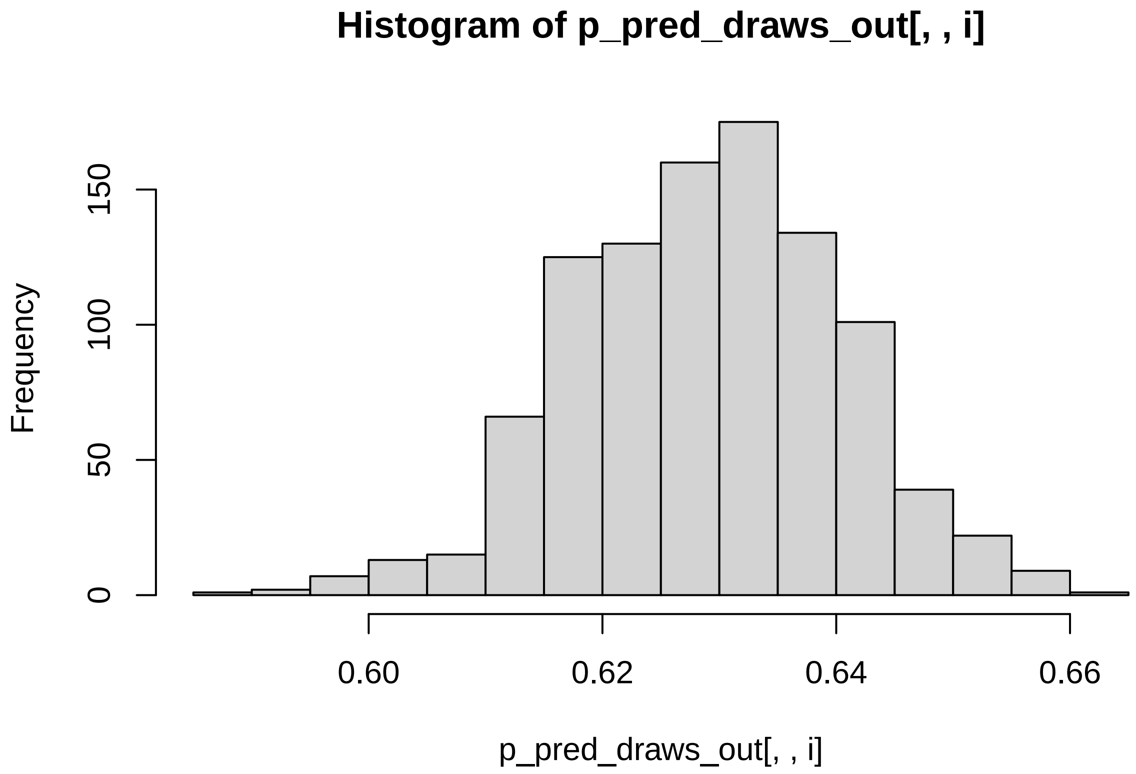

Histogram of p_pred_strike_no_swing_train_draws[, , i]
Frequency
0
10
20
30
40
0.30
0.32
0.34
0.36
p_pred_strike_no_swing_train_draws[, , i]



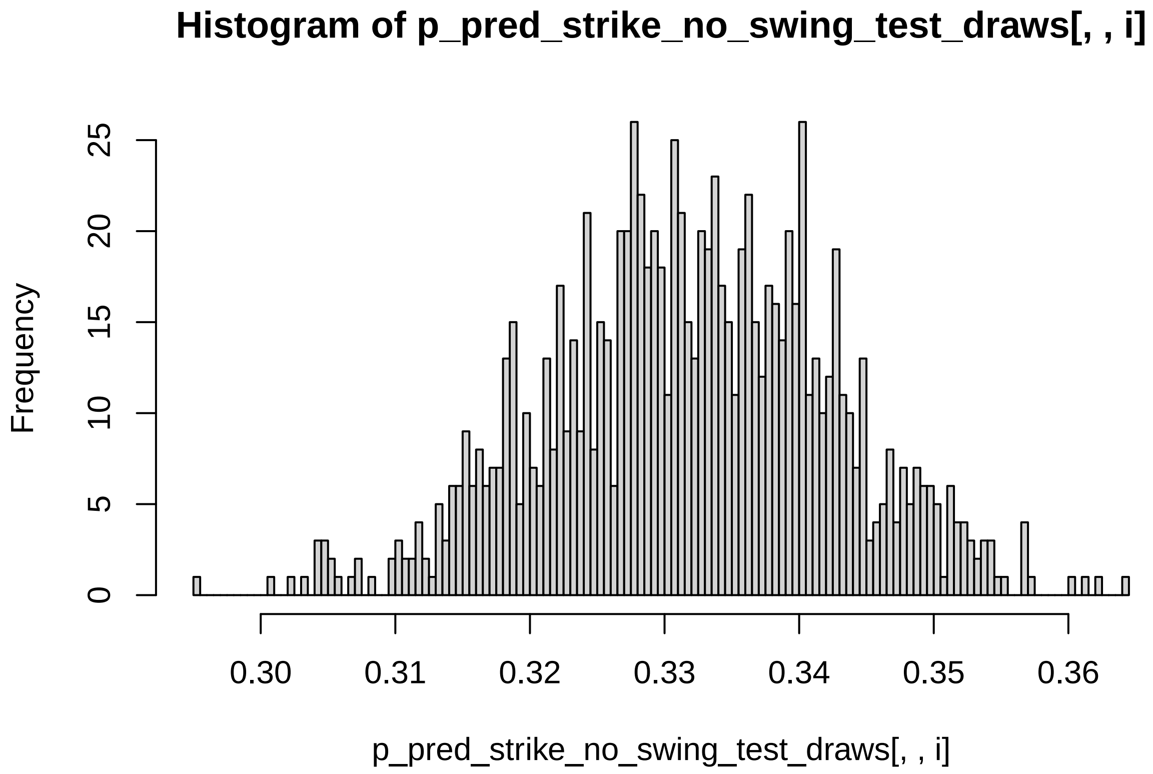
```
######

p_pred_draws_out = fit$draws("p_pred_out");
## we can see p switches between the two
for (i in 1:3) {
  hist(p_pred_draws_out[,,i]);
}
```

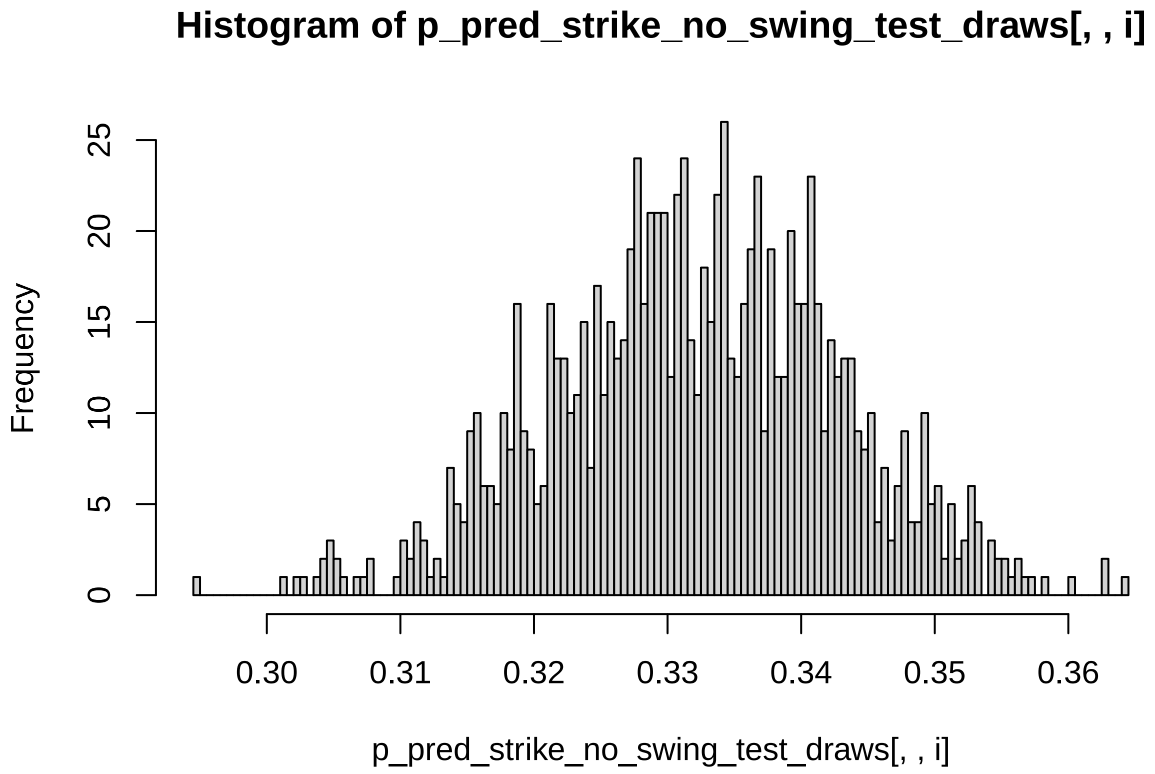

Histogram of p_pred_draws_out[, , i]
Frequency
0
50
100
150
200
250
0.28
0.30
0.32
0.34
0.36
0.38
p_pred_draws_out[, , i]

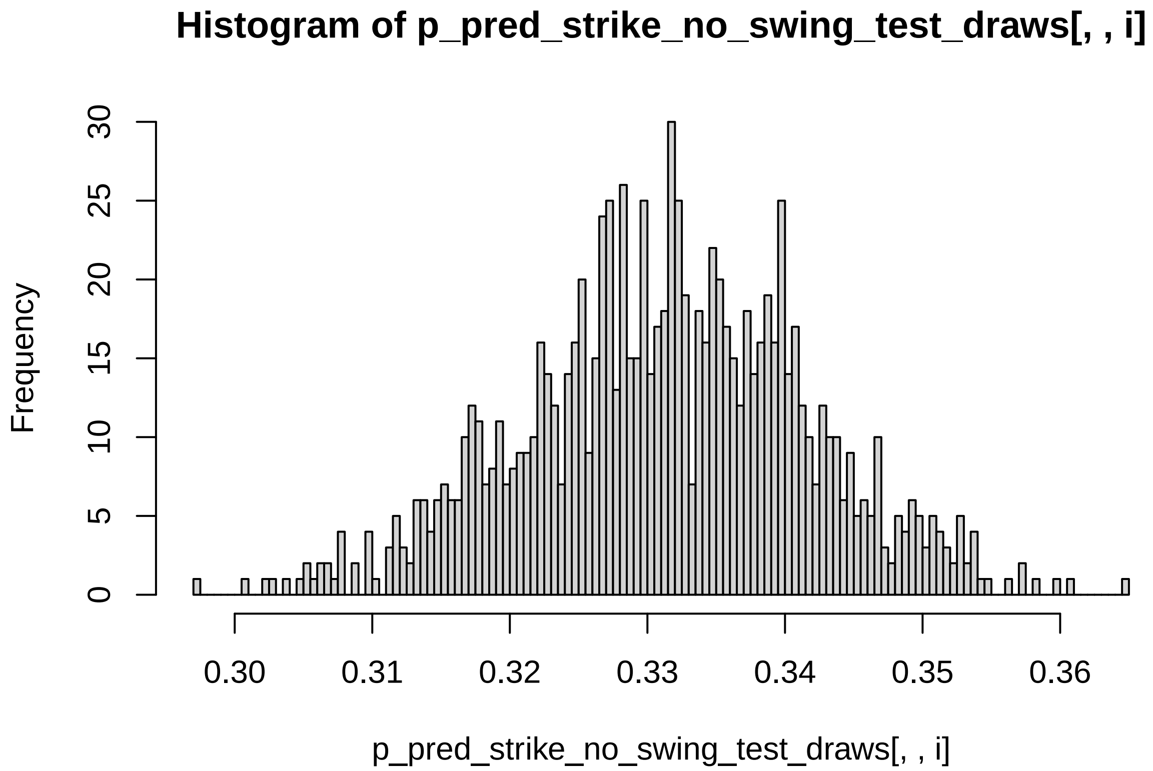

Histogram of p_pred_draws_out[, , i]
Frequency
0
50
100
150
200
250
0.28
0.30
0.32
0.34
0.36
0.38
p_pred_draws_out[, , i]



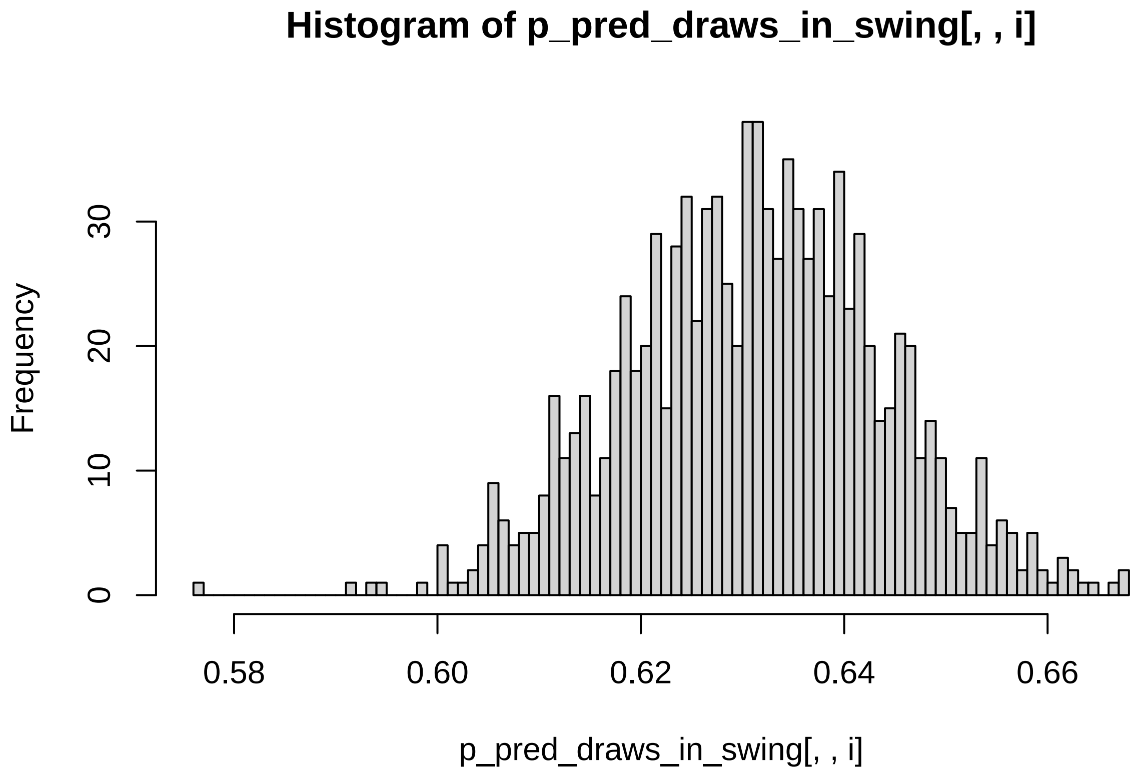
```
p_pred_strike_no_swing_test_draws = p_pred_draws_out[,,w_p_strike_no_swing_test];
for (i in 1:3) {
  hist(p_pred_strike_no_swing_test_draws[,,i], breaks = 100);
}
```

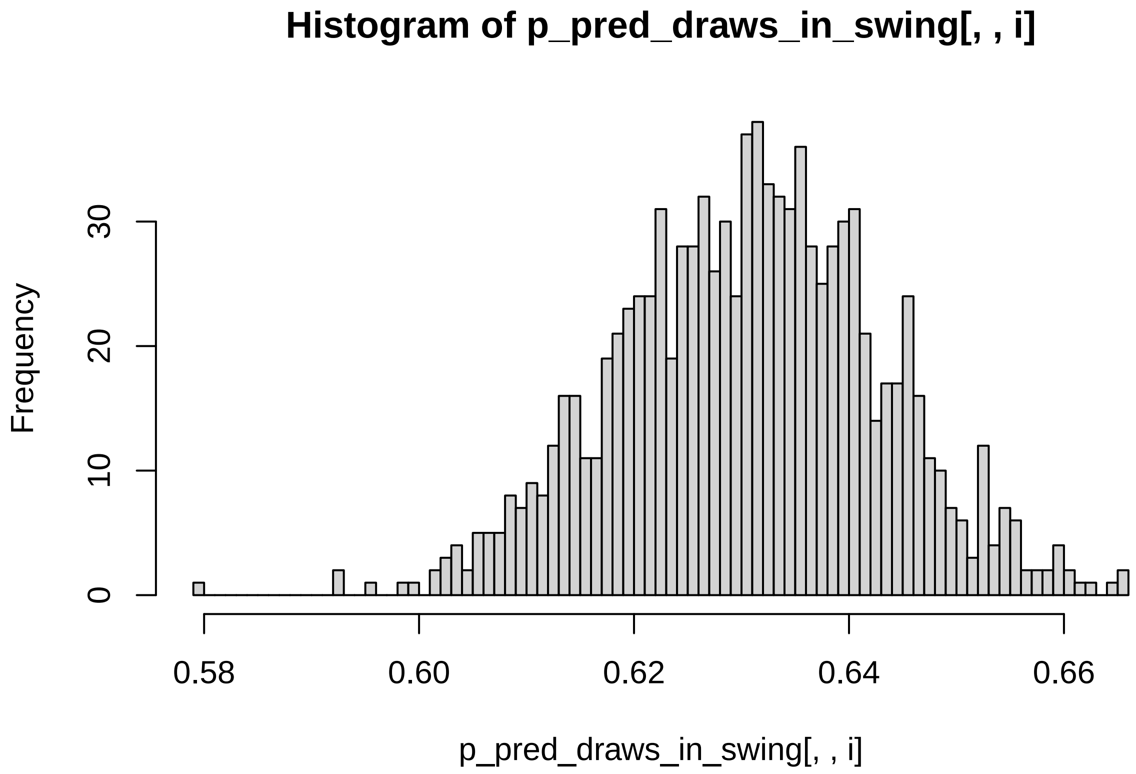

Histogram of p_pred_strike_no_swing_test_draws[, , i]
Frequency
0
5
10
15
20
25
0.30
0.31
0.32
0.33
0.34
0.35
0.36
p_pred_strike_no_swing_test_draws[, , i]

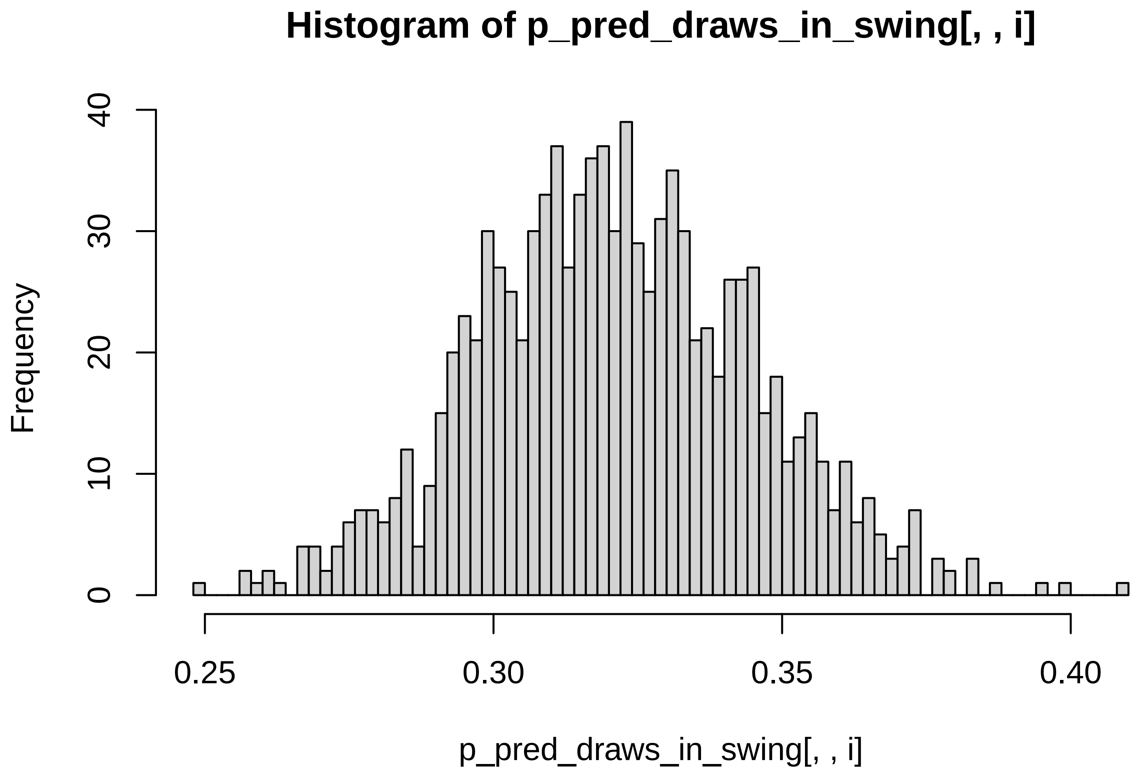

Histogram of p_pred_strike_no_swing_test_draws[, , i]
Frequency
0
5
10
15
20
25
0.30
0.31
0.32
0.33
0.34
0.35
0.36
p_pred_strike_no_swing_test_draws[, , i]



```
## p(strike | swing, speed conditionals)
p_pred_draws_in_swing = p_pred_draws_in[,,-w_p_strike_no_swing_train];
p_pred_draws_out_swing = p_pred_draws_out[,,-w_p_strike_no_swing_test];


for (i in 1:3) {
  hist(p_pred_draws_in_swing[,,i], breaks = 100)
}
```


Histogram of p_pred_draws_in_swing[, , i]
Frequency
0
10
20
30
0.58
0.60
0.62
0.64
0.66
p_pred_draws_in_swing[, , i]

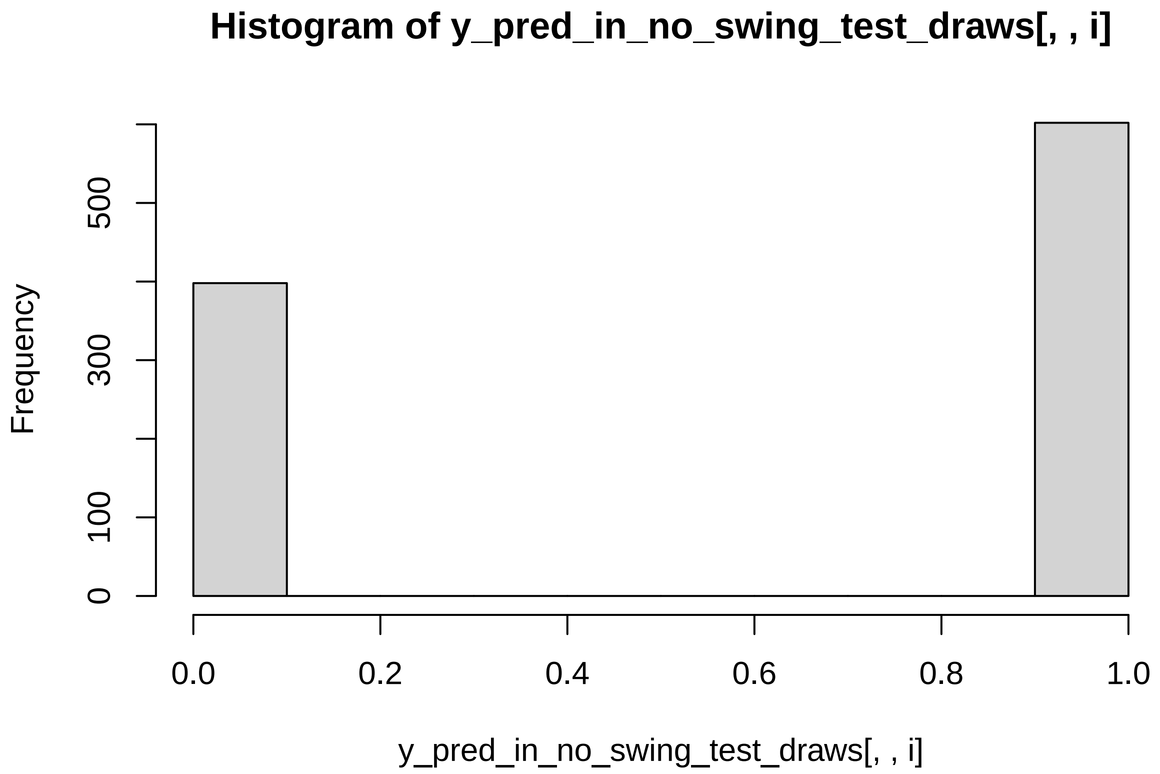

Histogram of p_pred_draws_in_swing[, , i]
Frequency
0
10
20
30
0.58
0.60
0.62
0.64
0.66
p_pred_draws_in_swing[, , i]

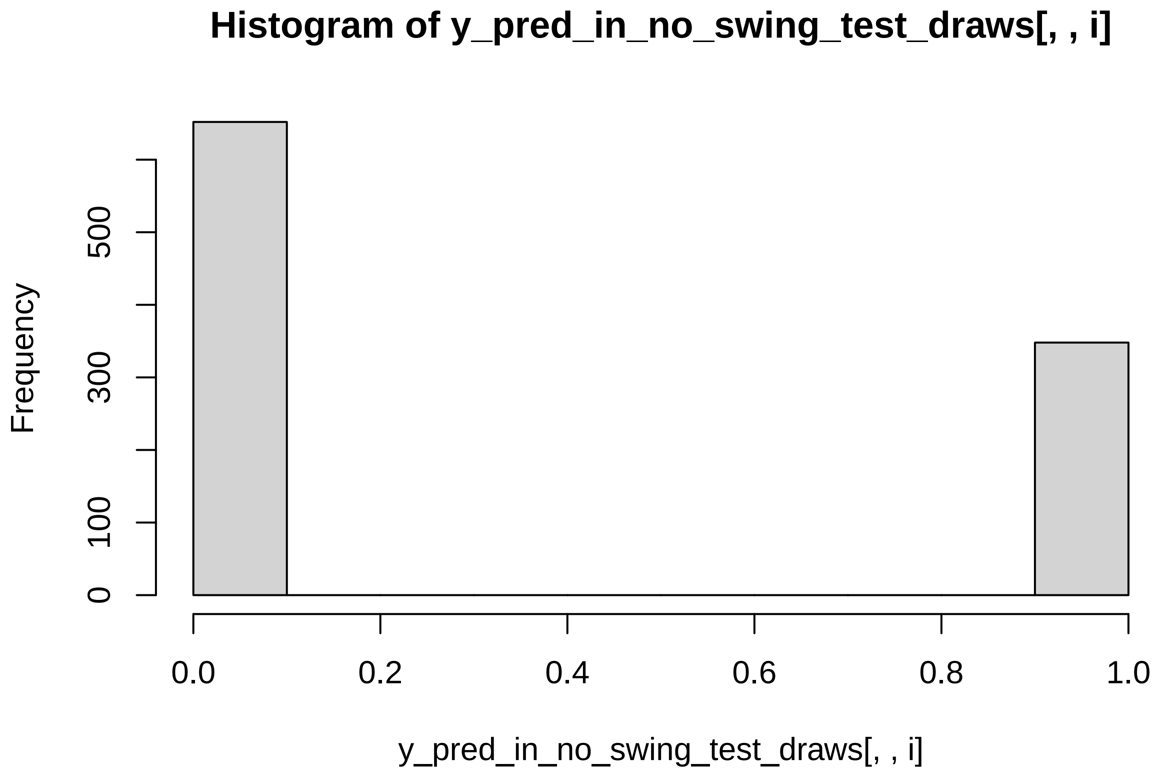


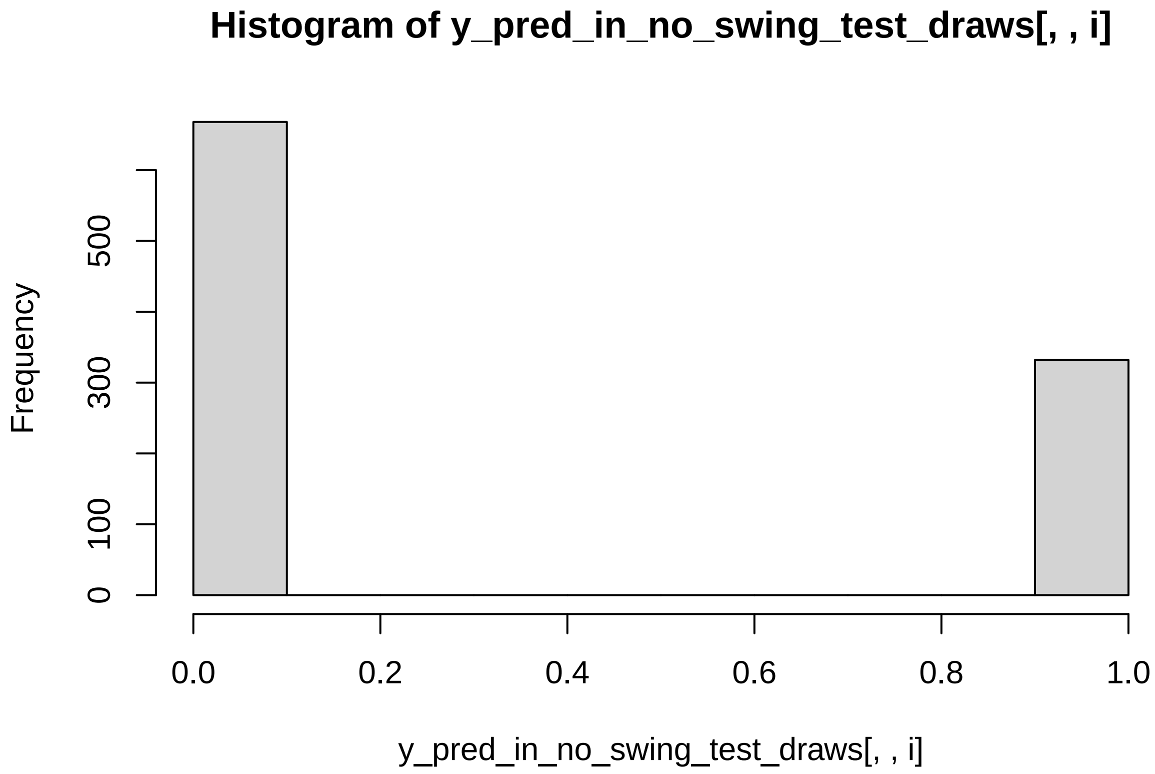
```
## p_pred for swinging on testt
train_swing_p_pred = apply(p_pred_draws_in_swing, MARGIN = 3, FUN = mean);
test_swing_p_pred = apply(p_pred_draws_out_swing, MARGIN = 3, FUN = mean);

###################################
## TODO predictive accuracy

y_pred_in_draws = fit$draws("y_pred_in");
y_pred_out_draws = fit$draws("y_pred_out");


## no swing
y_pred_in_no_swing_train_draws = y_pred_in_draws[,,w_p_strike_no_swing_train];
y_pred_in_no_swing_test_draws = y_pred_in_draws[,,w_p_strike_no_swing_test];

## swing


for(i in 1:3) {
  #hist(y_pred_in_no_swing_train_draws[,,i]);
  print(mean(y_pred_in_no_swing_train_draws[,,i]));
}
```

```
## [1] 0.348
## [1] 0.331
## [1] 0.329
```

```
for(i in 1:3) {
  hist(y_pred_in_no_swing_test_draws[,,i]);
}
```




Histogram of y_pred_in_no_swing_test_draws[, , i]
Frequency
0
100
300
500
0.0
0.2
0.4
0.6
0.8
1.0
y_pred_in_no_swing_test_draws[, , i]



```
n_obs_y_pred_in_train = dim(y_pred_in_no_swing_train_draws)[3]
y_preds_in_train = c();
for (i in 1:n_obs_y_pred_in_train) {
  y_preds_in_train = c(y_preds_in_train, Mode(y_pred_in_no_swing_train_draws[,,i])[1]);
}

n_obs_y_pred_in = dim(y_pred_in_no_swing_test_draws)[3]
y_preds_in = c();
for (i in 1:n_obs_y_pred_in) {
  y_preds_in_test = c(y_preds_in, Mode(y_pred_in_no_swing_test_draws[,,i])[1]);
}
```

We have randomly generated predictions, but we can compute a distribution of predictions, for example, if the result was not binary, by simulating $\sim y_{pred}$ in or out for N simulations. In this case, we have the predictive probability, and it will be similar. We can compute confidence intervals for a proportion using a logit transform, or for a continuous probability distribution, we can just compute the 95% confidence intervals of the distribution directly, using the .05 and .95 quantiles of the samples dt[,floor(.95 * nsamples)]. For example, if we generate from the poster predictive, order the samples and compute the lower 5% and upper 95% and this will yeild confidence intervals.

Adding hierarchies/multilevel and pooling

The model above is very basic, but we can begin to add hierarchies and pool within similar groups. Using multilevel models (varying slopes and intercepts). I haven't done this for the entire dataset, but a good place to add hierarchies might be pitch type, player, within league, etc. This is a multilevel regression model. We can further add different levels and slopes. Partial pooling adds shrinkage to similar groups. Here, I did pitch type. Below is the model and the Stan code.

$$y_i \sim inv\_logit(\beta_0 + \beta^{pt}_{j[i]} x_{pt} + B'X)$$

```
data {
  int N;       // N observations
  int M;       // M covariates
  int Pt;       // num levels (here, we can pool on pitch type for simplicity
  int N_pred;  // num out of sample predictions

  array[N] int<lower=1, upper=Pt> pt;  // pt: pitch type here this is pitch type but I left it arbitrary
  matrix[N, M] X;
  array[N] int y;
  matrix[N_pred, M] X_pred;
}
parameters {
  real beta0;
  vector[M] beta;
  real mu_beta_pt;
  real<lower=0> sigma_beta_pt;
  vector<offset=mu_beta_pt, multiplier=sigma_beta_pt>[Pt] beta_pt;  // non-centered parameterization
}
model {
  beta0 ~ normal(0, 1);
  beta_pt ~ normal(mu_beta_pt, sigma_beta_pt);
  beta ~ normal(0, 1);
  y ~ bernoulli_logit(beta0 + beta_pt[pt] + X * beta);
}
generated quantities {
  vector[N] p_pred_in;             // probability p from model in sample
  vector[N] p_pred_out;             // probability p from model out of sample
  array[N] int y_pred_in;       // in sample predictions
  array[N_pred] int y_pred_out; // out of sample predictions

  for (i in 1:N) {
    p_pred_in[i] = inv_logit(beta0 + beta_pt[pt[i]] + X[i, ] * beta);
    y_pred_in[i] = bernoulli_logit_rng(beta0 + beta_pt[pt[i]] + X[i, ] * beta);
  }

  for (i in 1:N_pred) {
    p_pred_out[i] = inv_logit(beta0 + beta_pt[pt[i]] + X_pred[i, ] * beta);
    y_pred_out[i] = bernoulli_logit_rng(beta0 + beta_pt[pt[i]] + X_pred[i, ] * beta);
  }
}
```

We can determine the level effect of strike out percentage given pitch type by plotting $\beta_{pt}|pt = p$, where p is the pitch time. The rest of the code is the same.

Additions

Suppose we want to do inference on, meaning, draw practically useful interpretation on a certain parameter, that has a non-linear effect, we could use a Gaussian process (GP) prior, on a certain parameter to model non-linear effects. As a test, for an accelerated time to failure model (survival model), I was accurately able to model the non-linear effect of heart rate (or blood pressure) to stroke. I honestly do not remember, I'd have to ask Dr. Murthy but this was years ago. The paramter of interest is speed, the model is below. There is code for this on my website, and given more time I can fit this for you. In a day, and this depends on data size and run time.

$$priors \sim \cdot \beta_{speed} = f(x_i) f(x_i) \sim GP(m(x_i), K(x, x')) y_i = \beta_0 + f(x_i) + B'X + \epsilon_i$$

And I can implement this given more time.

Phyisical System:

It would good to consider physical system. Uniform prior of the rest of the box, distance to the plate, size of the hitters body, in order to estiamte the strike-out zone, and then, perhaps depending on there the pitcher pitches the ball in the box, some model of uncertainty that incorporates prior information accounting for prior information that determines how likely is to pitch in that region. For example, probability is models, "a square," and margin of error of a sinker landing below the strike zone.

Question 2:

Question 2 a

This is a forecasting model that's an adaptation of one of Andrew Gelman, PhD's election forecasting models, which has been applied succesfully on sports applications before.

Forecasting one ahead, A Basic Random-Walk Model

As a primer, we're going to forecast one step ahead so we can get an idea of the prediction accuracy of our model. The model and Stan code are below. This model can, and should be expanded, but due to time constraints, I've only included pitcher level conditions. But this can easily be expanded and we can include additional covariates and hierarchies.

$$\mu_{t|t-1} \sim MVN(\mu_{t-1}, \Sigma), \epsilon \sim \eta_{t[i]}$$

This is the basic model, I've made some tweaks and adjusted the within pitcher vs without pitcher covariance matrix. Givenmore time, I'd add more adjustments.

The model below does predict out of sample, but I only trained it up to 2023 and predicted for 2024. Then, I messed up indexing, due to some lost work, or getting lost in somefiles and the rw1_2025.stan, and then when I used it, I was getting global mean for 2025, although in this model, I did predict using data I didn't train data for, so I'm working this out.

```
## rw1.stan
data {
  // rw1 stan
  int N;    // N observations
  int TIME;    // time points
  int P;    // num pitchers

  array[N] real y;   // outcome
  vector[N] stuff;
  array[N] int<lower=1, upper=TIME+1> time; // were looking at 2024, prediction for 2025 and then anothe
  array[N] int<lower=1, upper=P> pitcher;

  matrix[P, P] ss_cov_mu_b_T_P;
  matrix[P, P] ss_cov_mu_b_walk_P;
}
transformed data {
  cholesky_factor_cov[P] cholesky_ss_cov_mu_b_T_P;
  cholesky_factor_cov[P] cholesky_ss_cov_mu_b_walk_P;

  cholesky_ss_cov_mu_b_T_P = cholesky_decompose(ss_cov_mu_b_T_P);
  cholesky_ss_cov_mu_b_walk_P = cholesky_decompose(ss_cov_mu_b_walk_P);
}
parameters {
  vector[P] mu_b_prior_P;

  vector[P] raw_mu_b_T_P;
  matrix[P, TIME] raw_mu_b_P;

  real<lower=0> sigma;
}
transformed parameters {
  vector[N] pi;

  matrix[P, TIME] mu_b_P;

  mu_b_P[:,TIME] = cholesky_ss_cov_mu_b_T_P * raw_mu_b_T_P + mu_b_prior_P;
  for (i in 1:(TIME-1)) {
    mu_b_P[:, TIME - i] = cholesky_ss_cov_mu_b_walk_P * raw_mu_b_P[:, TIME - i] + mu_b_P[:, TIME + 1 -
  }

  for (i in 1:N) {
    pi[i] = mu_b_P[pitcher[i], time[i]];
  }
}
model {
  raw_mu_b_T_P ~ normal(.5, .1);
  to_vector(raw_mu_b_P) ~ normal(0, 1);
  sigma ~ normal(0, 1);

  mu_b_prior_P ~ normal(.5, 1);
  for (i in 1:N) {
    y ~ normal(pi[i], sigma);
  }
}
generated quantities {
  matrix[P, TIME] pitcher_preds;
  vector[TIME] pitcher1_pred;

  for ( p in 1:P) {
    for (t in 1:TIME)   {
      pitcher_preds[p, t] = mu_b_P[p, t];
    }
  }
}
```

And then we prepare the data below.

```
setwd("C:/Users/andre/likely/mets/tech_interview/");
pitch = read.csv("pitcher_strikeout_data.csv");

standardize = function(x) {
  x_mu = mean(x);
  x_sd = sd(x);
  x_std = (x - x_mu) / x_sd;
  return(list(x_std, x_mu, x_sd));
}

cov_matrix <- function(n, sigma2, rho){
  m <- matrix(nrow = n, ncol = n)
  m[upper.tri(m)] <- rho
  m[lower.tri(m)] <- rho
  diag(m) <- 1
  (sigma2^.5 * diag(n))  %*% m %*% (sigma2^.5 * diag(n))
}

## aggregate over strikeout, so we have a percentagae, and stuff (may be I wouldn't do this next time)
tmp = aggregate(list(pitch$is_strikeout, pitch$stuff), list(pitch$year, pitch$pitcher_id), FUN = mean)
names(tmp) = c("year", "pitcher_id", "strikeout_perc", "stuff");

## downsample by pitchers
N_pitchers = 50;
pitchers = unique(tmp$pitcher_id); ## there is 200 pitchers, let's take 50
pitchers_downsample = sample(pitchers, N_pitchers);
pitch_ds = tmp[which(tmp$pitcher_id %in% pitchers_downsample), ];

pitch_ds_strikeout_prc_std = standardize(pitch_ds$strikeout_perc);
pitch_ds_stuff_std = standardize(pitch_ds$stuff);

pitch_ds = cbind(pitch_ds, pitch_ds_strikeout_prc_std[[1]], pitch_ds_stuff_std[[1]]);
###################################
# separate data into test/train sets,
#

train = pitch_ds[which(pitch_ds$year < 2024),];
test = pitch_ds[which(pitch_ds$year == 2024),];

## take a look at some things, histogram and a time series
## the estimates, because I'm getting future estimates close to
## these checks I should do anyway, but I'm rushing.
#hist(pitch_ds$strikeout_perc, breaks = 100);

## you can cycle through all the pitchers to see that you have unique
## I should have done this initially
#plot(pitch_ds$strikeout_perc[which(pitch_ds$pitcher_id == 19)], type = "l");
```

```
N = nrow(pitch_ds);
TIME = length(unique(pitch_ds$year)); ## 2020... 2024, predicting for 2025
y = as.numeric(pitch_ds$`pitch_ds_strikeout_prc_std[[1]]`);
P = length(unique(pitch_ds$pitcher_id));
time = as.numeric(pitch_ds$year - 2019);
stuff = pitch_ds$`pitch_ds_stuff_std[[1]]`;
#pitcher = as.numeric(pitch_ds$pitcher_id + 1);
pitcher_map = cbind(1:length(unique(pitch_ds$pitcher_id)), unique(pitch_ds$pitcher_id));
pitcher = c();
for (i in 1:N) {
  pitcher = c(pitcher, pitcher_map[which(pitch_ds$pitcher_id[i] == pitcher_map[,2]), 1]);
}
## BAD PRIOR
ss_cov_mu_b_T_P = cov_matrix(P, 0.1, 0.9);
ss_cov_mu_b_walk_P = cov_matrix(P, 0.1, 0.9);

#dt = list(y = y, N = N, P = P, TIME = TIME, time = time, pitcher = pitcher, stuff = stuff, ss_cov_mu_b
#          ss_cov_mu_b_walk_P = ss_cov_mu_b_walk_P);
#write_stan_json(dt, "//wsl.localhost/Ubuntu/home/andre/likely/mets/tech_interview/rw1_p50_ds.data.json
```

Looking at the individual plots, I think we should add another update.

We can look at posterior predictive projections for 2025, to see how accurate the model is, comparing one year ahead projections from 2020 to 2024. Then, we can provide forecasts for the next year, where the outcome is unkown. We can then update the model as we get new data, I believe, in this version of Stan. As a start, I'm just going to plot year ahead forecasts and compare them with the actual estimates. We can plot several year ahead forecasts, as well.

Reading in the Data, Summarizing Results

So I ran this model, and it's important to do diagnostics (like checking) model fit (R-squared), or something in the classical case, although there are known warnings with that diagnostic. One of the MCMC diagnostics, we look at is $\hat{R}$. We had no divergences. Posterior estimates you should not use at all. It means your model is totally off, or you have some multimodality you're not accounting for. You're trying to sample some posterior mass that doesn't exist conditional on your data.

We run multiple chains, and order to assess convergence in order to determine whether our estimates are consistent accross changes, we want the variance between chains to be low. It's like a "hypothesis test" to determine whether two distributions are the same. Sometimes we can remedy this by running longer chains, but better to re-parameterize it.

Some of the $\hat{R}$ values are high, but this model tool a while to run, so I'm going to interpret the results I have. They were all on the covariance matrix, and these were fixed initial conditions, so I might, set within pitcher correlation higher and out of pitcher correlation lower.

I changed the covariance matrix initialization, and I got rid of most of convergence issues. I added more within pitcher correlation, and then left off the off diagonals really low. First, diagonal was .04,I changed it to .1. Off-diagonal is .036. Could be more correlated.

See supplement section A.

Reading in and Generating Forecasts

```r
setwd("C:/Users/andre/likely/mets/tech_interview/");

fit = cmdstanr::as_cmdstan_fit("rw1_out_p50.csv");

dim(fit$draws("pitcher_preds"))
```

```
## [1] 1000    1  250
```

```r
samples = fit$draws("pitcher_preds");

## get predictions for pitcher1
pitcher1_ind = which(pitcher == 1);

head(pitcher_map);
```

```
##      [,1] [,2]
## [1,]    1    7
## [2,]    2   13
## [3,]    3   14
## [4,]    4   16
## [5,]    5   20
## [6,]    6   21
```

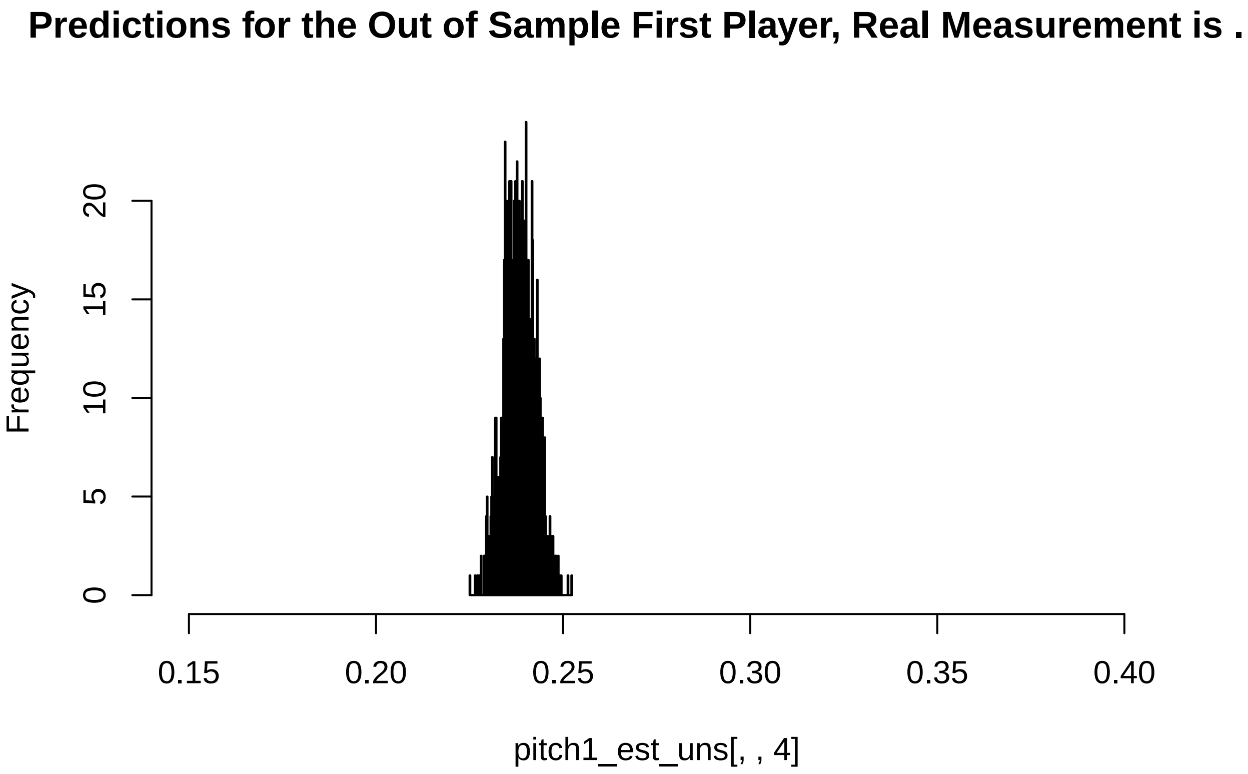
```r
head(pitcher);
```

```
## [1] 1 1 1 1 1 2
```

```r
head(time);
```

```
## [1] 1 2 3 4 5 1
```

```r
## so 1,2,3,4,5 are 2020, 2021, 2022, 2023, 2024, 2025 predictions for pitcher1
pitcher1_est = samples[,,pitcher1_ind];

## untransform y
pitch1_est_uns = (pitcher1_est * pitch_ds_strikeout_prc_std[[3]] + pitch_ds_strikeout_prc_std[[2]]);

pitch1_act = pitch_ds[which(pitch_ds$pitcher_id == 1),];

hist(pitch1_est_uns[,,4], breaks = 100, main = "Predictions for the Out of Sample First Player, Real Mea
#abline(v = mean(pitch1_est_uns[,,4]), col = "black");
par(new = FALSE);
abline(v = pitch1_act$strikeout_perc[5], col = "red"); ## it was .22
```



#### Making Predictions

The actual estimate is .22, I can't get the plots to work

note, I was supposed to predict for 2025, but I removed 2024 and predicted for 2024 using the other data, but the code is essentially the same

```
## just do year 4 predictions, I haven't check out to make sure the ID's match up exactly
## I think they should, but you get the point.
actual_2024 = pitch_ds[which(pitch_ds$year == 2024), 3];
means_est_samps = samples[,,seq(5, 250, by = 5)];
means_est = apply(means_est_samps, 3, FUN=mean)

means_est = (means_est) * pitch_ds_strikeout_prc_std[[3]] + pitch_ds_strikeout_prc_std[[2]];

#sum_sq_errors = sum((means_est) **2) / N_pitchers;
```

We computer a MSE as a metric, or whatever other distributional approximation of that you need, i.e. are the forecasts within a 95% Confidence interval. We can loop through the results in order to more thoroughly investigate forecasts. We're off by about 5% per pitcher when forecasting for the 2024 year. To get 2025, we can just add +2 to TIME, like so, and the same logic applies.

```
sum_sq_errors = sum((means_est) **2) / N_pitchers;
print(sum_sq_errors) ## we're off by about 5% by pitcher for the forecasting  for the 2024 year.
```

```
## [1] 0.0567554
```

$$\Sigma_i (y - \hat{y})^2 / N \approx 0.0537$$

2025 predictions and a bit of model expansion

See rw1_2025.stan, it's essentially the same as above, I just removed 2024, and then used it as a 1 time step ahead test set.

```r
setwd("C:/Users/andre/likely/mets/tech_interview/");
pitch = read.csv("pitcher_strikeout_data.csv");

standardize = function(x) {
  x_mu = mean(x);
  x_sd = sd(x);
  x_std = (x - x_mu) / x_sd;
  return(list(x_std, x_mu, x_sd));
}

cov_matrix <- function(n, sigma2, rho){
  m <- matrix(nrow = n, ncol = n)
  m[upper.tri(m)] <- rho
  m[lower.tri(m)] <- rho
  diag(m) <- 1
  (sigma2^.5 * diag(n))  %*% m %*% (sigma2^.5 * diag(n))
}

## aggregate over strikeout, so we have a percentagae, and stuff (may be I wouldn't do this next time)
tmp = aggregate(list(pitch$is_strikeout, pitch$stuff), list(pitch$year, pitch$pitcher_id), FUN = mean)
names(tmp) = c("year", "pitcher_id", "strikeout_perc", "stuff");

## downsample by pitchers
N_pitchers = 50;
pitchers = unique(tmp$pitcher_id); ## there is 200 pitchers, let's take 50
pitchers_downsample = sample(pitchers, N_pitchers);
pitch_ds = tmp[which(tmp$pitcher_id %in% pitchers_downsample), ];

pitch_ds_strikeout_prc_std = standardize(pitch_ds$strikeout_perc);
pitch_ds_stuff_std = standardize(pitch_ds$stuff);

pitch_ds = cbind(pitch_ds, pitch_ds_strikeout_prc_std[[1]], pitch_ds_stuff_std[[1]]);



N = nrow(pitch_ds);
TIME = length(unique(pitch_ds$year)); ## 2020... 2024, predicting for 2025
y = as.numeric(pitch_ds$`pitch_ds_strikeout_prc_std[[1]]`);
P = length(unique(pitch_ds$pitcher_id));
time = as.numeric(pitch_ds$year - 2019);
stuff = pitch_ds$`pitch_ds_stuff_std[[1]]`;
#pitcher = as.numeric(pitch_ds$pitcher_id + 1);
pitcher_map = cbind(1:length(unique(pitch_ds$pitcher_id)), unique(pitch_ds$pitcher_id));
pitcher = c();
for (i in 1:N) {
  pitcher = c(pitcher, pitcher_map[which(pitch_ds$pitcher_id[i] == pitcher_map[,2]), 1]);
}

### BAD PRIORS
ss_cov_mu_b_T_P = cov_matrix(P, 0.1, 0.9);
ss_cov_mu_b_walk_P = cov_matrix(P, 0.1, 0.9);

dt = list(y = y, N = N, P = P, TIME = TIME, time = time, pitcher = pitcher, stuff = stuff, ss_cov_mu_b_T
         ss_cov_mu_b_walk_P = ss_cov_mu_b_walk_P);
write_stan_json(dt, "//wsl.localhost/Ubuntu/home/andre/likely/mets/tech_interview/rw1_p50_ds_2025.data.j
```

```
setwd("C:/Users/andre/likely/mets/tech_interview/");

fit = cmdstanr::as_cmdstan_fit("rw1_out_p50_2025.csv");

dim(fit$draws("pitcher_preds"))
```

```
## [1] 1000    1  300
```

```
preds = fit$draws("pitcher_preds");

means_est_samps = preds[,,seq(2, 300, by = 6)];
means_est = apply(means_est_samps, 3, FUN=mean)

means_est = (means_est) * pitch_ds_strikeout_prc_std[[3]] + pitch_ds_strikeout_prc_std[[2]];
head(means_est);
```

```
##  pitcher_preds[2,1]  pitcher_preds[8,1] pitcher_preds[14,1] pitcher_preds[20,1]
##           0.2367483           0.2368010           0.2370479           0.2369304
## pitcher_preds[26,1] pitcher_preds[32,1]
##           0.2369946           0.2370315
```

Some Model Expansion, Adding Stuff

I made some 2025 forecasts last might, and I got results, and they were all the same for the 2025 year, so I think adding a varying intercept, for player, so the forecasts are more realistic. Now that we have a working model, we can add some additions.

I'm running out of time, but I could have used a simple AR model. The linear component, I'll add some varying intercepts. We'll just write the mean process, and keep it simple. Will a vague prior.

Actually all we have to add is stuff.

$$\mu_t|\mu_{t-1} \sim \mu_{t,p} + \beta_s \mu_{[t-1],s,p}\mu_{p,t} \sim N(.5,.8)$$

This is what I was working on. I was able to get 1 ahead predictions, the first random walk model was fine, the second one I messed up indices. Sometimes things get lost because there's so many degrees of freedom.

```
## rw1_2025_mup.stan
data {
  // rw1 stan
  int N;    // N observations
  int TIME;    // time points
  int P;    // num pitchers

  array[N] real y;   // outcome
  array[N] int<lower=1, upper=TIME + 1> time; // we're looking at 2024, prediction for 2025 and then an
  array[N] int<lower=1, upper=P> pitcher;

  array[N]  real stuff;

  matrix[P, P] ss_cov_mu_b_T_P;
  matrix[P, P] ss_cov_mu_b_walk_P;

  // matrix[P, P] ss_cov_mu_b_T_S;
  // matrix[P, P] ss_cov_mu_b_walk_S;
}
transformed data {
  cholesky_factor_cov[P] cholesky_ss_cov_mu_b_T_P;
  cholesky_factor_cov[P] cholesky_ss_cov_mu_b_walk_P;

  // cholesky_factor_cov[P] cholesky_ss_cov_mu_b_T_S;
  // cholesky_factor_cov[P] cholesky_ss_cov_mu_b_walk_S;

  cholesky_ss_cov_mu_b_T_P = cholesky_decompose(ss_cov_mu_b_T_P);
  cholesky_ss_cov_mu_b_walk_P = cholesky_decompose(ss_cov_mu_b_walk_P);

  // cholesky_ss_cov_mu_b_T_S = cholesky_decompose(ss_cov_mu_b_T_S);
  // cholesky_ss_cov_mu_b_walk_S = cholesky_decompose(ss_cov_mu_b_walk_S);
}
parameters {
  vector[P] mu_b_prior_P;
//  vector[P] mu_b_prior_S;

  vector[P] raw_mu_b_T_P;
  matrix[P, TIME] raw_mu_b_P;

//  vector[P] raw_mu_b_T_S;
//  matrix[P, TIME] raw_mu_b_S;

  real<lower=0> sigma;
}
transformed parameters {
  vector[N] pi;

  matrix[P, TIME] mu_b_P;
  matrix[P, TIME] mu_b_S;
//  matrix[P, TIME] stuff_beta;
  array[P, TIME] real<lower=-1, upper=1> stuff_beta;
  for (i in 1:N) {
    stuff_beta[pitcher[i], time[i]] = stuff[i];
  }

  mu_b_P[:,TIME] = cholesky_ss_cov_mu_b_T_P * raw_mu_b_T_P + mu_b_prior_P;
//  mu_b_S[:,TIME] = cholesky_ss_cov_mu_b_T_S * raw_mu_b_T_S + mu_b_prior_S;
```

```
  for (i in 1:(TIME-1)) {
    mu_b_P[:, TIME - i] = cholesky_ss_cov_mu_b_walk_P * raw_mu_b_P[:, TIME - i] + mu_b_P[:, TIME + 1 -
//    mu_b_S[:, TIME - i] = cholesky_ss_cov_mu_b_walk_S * raw_mu_b_S[:, TIME - i] + mu_b_S[:, TIME + 1
    mu_b_S[:, TIME - i] = mu_b_S[:, TIME + 1 - i]; // * stuff_beta[:, TIME + 1 - i];
}

  for (i in 1:N) {
    pi[i] = mu_b_P[pitcher[i], time[i]] + mu_b_S[pitcher[i], time[i]] * stuff_beta[pitcher[i], time[i]]
  }
}
model {


raw_mu_b_T_P ~ normal(.5, .1);
//  raw_mu_b_T_S ~ normal(.5, .1);
  to_vector(raw_mu_b_P) ~ normal(0, 1);
//  to_vector(raw_mu_b_S) ~ normal(0, 1);
  sigma ~ normal(0, 1);

  mu_b_prior_P ~ normal(.5, .4);
//  mu_b_prior_S ~ normal(.5, .4);
  for (i in 1:N) {
    y ~ normal(pi[i], sigma);
  }
}
generated quantities {
  matrix[P, TIME] pitcher_preds;
  // matrix[P, TIME] stuff_beta;
// for (i in 1:N) {
 //    stuff_beta[pitcher[i], time[i]] = stuff[i];
 // }

  for ( p in 1:P) {
    for (t in 1:TIME) {
      pitcher_preds[p, t] = mu_b_P[p, t] + stuff_beta[p, t] * mu_b_S[p, t];

    }
  }
}
```

```
setwd("C:/Users/andre/likely/mets/tech_interview/");
pitch = read.csv("pitcher_strikeout_data.csv");

standardize = function(x) {
  x_mu = mean(x);
  x_sd = sd(x);
  x_std = (x - x_mu) / x_sd;
  return(list(x_std, x_mu, x_sd));
}

cov_matrix <- function(n, sigma2, rho){
  m <- matrix(nrow = n, ncol = n)
  m[upper.tri(m)] <- rho
```

```r
  m[lower.tri(m)] <- rho
  diag(m) <- 1
  (sigma2^.5 * diag(n))  %*% m %*% (sigma2^.5 * diag(n))
}

## aggregate over strikeout, so we have a percentagae, and stuff (may be I wouldn't do this next time)
tmp = aggregate(list(pitch$is_strikeout, pitch$stuff), list(pitch$year, pitch$pitcher_id), FUN = mean)
names(tmp) = c("year", "pitcher_id", "strikeout_perc", "stuff");

## downsample by pitchers
N_pitchers = 50;
pitchers = unique(tmp$pitcher_id); ## there is 200 pitchers, let's take 50
pitchers_downsample = sample(pitchers, N_pitchers);
pitch_ds = tmp[which(tmp$pitcher_id %in% pitchers_downsample), ];

pitch_ds_strikeout_prc_std = standardize(pitch_ds$strikeout_perc);
pitch_ds_stuff_std = standardize(pitch_ds$stuff);

pitch_ds = cbind(pitch_ds, pitch_ds_strikeout_prc_std[[1]], pitch_ds_stuff_std[[1]]);



N = nrow(pitch_ds);
TIME = length(unique(pitch_ds$year)) + 1; ## 2020... 2024, predicting for 2025
y = as.numeric(pitch_ds$`pitch_ds_strikeout_prc_std[[1]]`);
P = length(unique(pitch_ds$pitcher_id));
time = as.numeric(pitch_ds$year - 2019);
stuff = pitch_ds$`pitch_ds_stuff_std[[1]]`;
#pitcher = as.numeric(pitch_ds$pitcher_id + 1);
pitcher_map = cbind(1:length(unique(pitch_ds$pitcher_id)), unique(pitch_ds$pitcher_id));
pitcher = c();
for (i in 1:N) {
  pitcher = c(pitcher, pitcher_map[which(pitch_ds$pitcher_id[i] == pitcher_map[,2]), 1]);
}

ss_cov_mu_b_T_P = cov_matrix(P, 0.1, 0.3);
ss_cov_mu_b_walk_P = cov_matrix(P, 0.1, 0.3);

ss_cov_mu_b_T_S = cov_matrix(P, 0.1, 0.3);
ss_cov_mu_b_walk_S = cov_matrix(P, 0.1, 0.3);


dt = list(y = y, N = N, P = P, TIME = TIME, time = time, pitcher = pitcher, stuff = stuff, ss_cov_mu_b_T
          ss_cov_mu_b_walk_S = ss_cov_mu_b_walk_S);
#write_stan_json(dt, "//wsl.localhost/Ubuntu/home/andre/likely/mets/tech_interview/rw1_p50_ds_2025_stuf
```

```r
setwd("C:/Users/andre/likely/mets/tech_interview/");

fit = cmdstanr::as_cmdstan_fit("rw1_out_p50_2025_stuff.csv");

dim(fit$draws("pitcher_preds"))
```

```
## [1] 1000    1  300
```

```
preds = fit$draws("pitcher_preds");

means_est_samps = preds[,,seq(2, 300, by = 6)];
means_est = apply(means_est_samps, 3, FUN=mean)

means_est = (means_est) * pitch_ds_strikeout_prc_std[[3]] + pitch_ds_strikeout_prc_std[[2]];
head(means_est);
```

```
##  pitcher_preds[2,1]  pitcher_preds[8,1] pitcher_preds[14,1] pitcher_preds[20,1]
##           0.2379088           0.2381231           0.2379268           0.2379854
## pitcher_preds[26,1] pitcher_preds[32,1]
##           0.2378970           0.2378896
```

Old Estimates, trying to figure out why I'm getting a global mean.

pitcher_preds[2,1] pitcher_preds[8,1] pitcher_preds[14,1] pitcher_preds[20,1] pitcher_preds[26,1] 0.2310806 0.2312163 0.2314587 0.2310048 0.2310272 pitcher_preds[32,1] 0.2311393

```
setwd("C:/Users/andre/likely/mets/tech_interview/");

fit2 = cmdstanr::as_cmdstan_fit("rw1_out_p50_2025_stuff.csv");

dim(fit2$draws("pitcher_preds"))
```

```
## [1] 1000    1  300
```

```
preds = fit2$draws("pitcher_preds");

means_est_samps = preds[,,seq(2, 300, by = 6)];
means_est = apply(means_est_samps, 3, FUN=mean)

means_est = (means_est) * pitch_ds_strikeout_prc_std[[3]] + pitch_ds_strikeout_prc_std[[2]];
head(means_est);
```

```
##  pitcher_preds[2,1]  pitcher_preds[8,1] pitcher_preds[14,1] pitcher_preds[20,1]
##           0.2379088           0.2381231           0.2379268           0.2379854
## pitcher_preds[26,1] pitcher_preds[32,1]
##           0.2378970           0.2378896
```

Moving forward Not the best model, but make some tweaks, like add additional terms in the sum and levels, and we'll be good.

We can add more covariates, use the entire dataset, and add additional model parameters in order to see how we can improve predictions and interpretability. I'm running out of time.

With more, time, I'd make sure the forecasts are accurate, because I have to, but this is a demo.

NOTE: rw1.stan

To add: Age, team, league, experience, pitcher/batter combos, league, inning, score, number of outs, top/bottom, number on deck, etc. We can pooling and informative priors, including non-linear components in an additive model.

Question 2 b

We could add physical attributes, external factors, such as home or away (hours of sleep can affect performance). Physical fitness, is this person's blood pressure good, are they in shape, what is the condition of their limbs, etc. We can add background, baseball players from different countries, may have a different hitting style that's more effective if they have a different style, lefty, etc, kind of ball, etc.

I might take more qualitative aspects of baseball, inuries, or optimize player value, and see if it's possible to capitalize that, prior to adding them into a model.

For example, sometimes it's useful to use statistics like strike out percentage, obviously, as opposed to hit percentage, because we know the players will get on base.

Some Convex Optimization.

I've used a little bit of convex optimization when studying radar. I'm wondering, if you have a huge dataset of possible draft picks, we could, within budget constraints, do a comibinatorial search of players with desirabale combinations. Or something like, maximize over win probability given certain attributed of players, so rather than trying to build a team based on individual attributes, we can build on combinations of players. Similary to a Markowitz portfolio optimization problem but adapted to baseball with extra baseball additions. Based on player attributes, do a Monte Carlo simulation of win probabilities of different teams. Here I was using statistical models, but you can simulate disease evolution with random number generator without using Stan or statistical models, although I haven't done this in practice in a while.

Supplements:

A: Showing reduction in R-hat by a re-parameterization. This is my terminal. Using a little more domain knowledge we can improve this.

```
Rank-normalized split effective sample size satisfactory for all parameters.

The following parameters had rank-normalized split R-hat greater than 1.01:
  mu_b_prior_P[3], mu_b_prior_P[4], mu_b_prior_P[7], mu_b_prior_P[9], mu_b_prior_P[10], mu_b_prior_P[12
Such high values indicate incomplete mixing and biased estimation.
You should consider regularizing your model with additional prior information or a more effective parame

Processing complete.
andre@compy:~/cmdstan/bin$
andre@compy:~/cmdstan/bin$
andre@compy:~/cmdstan/bin$
andre@compy:~/cmdstan/bin$

andre@compy:~/cmdstan/bin$
andre@compy:~/cmdstan/bin$
andre@compy:~/cmdstan/bin$ ./diagnose /home/andre/likely/mets/tech_interview/rw1_out_p50.csv
Checking sampler transitions treedepth.
Treedepth satisfactory for all transitions.
```

```
Checking sampler transitions for divergences.
No divergent transitions found.

Checking E-BFMI - sampler transitions HMC potential energy.
E-BFMI satisfactory.

Rank-normalized split effective sample size satisfactory for all parameters.

The following parameters had rank-normalized split R-hat greater than 1.01:
  raw_mu_b_P[12,2], raw_mu_b_P[45,2]
Such high values indicate incomplete mixing and biased estimation.
You should consider regularizing your model with additional prior information or a more effective parame

Processing complete.
andre@compy:~/cmdstan/bin$
```