An Analysis of the Baseball Pitch

Introduction:

Baseball. The sport of America. It is one of the most complex sports in the world, requiring batters to observe and determine a pitch type, speed, and location in only 100 milliseconds. There is almost no room for error and absolutely no time for hesitation. So how do they do it? Well, for starters they all have insanely fast reflexes. But beyond that, they use the game script, and other various factors to attempt to predict what type of pitch is going to come next. From there, they study the opposing pitchers in preparation for their matchup and keep in mind all sorts of data for each pitch the pitcher can throw such as how fast they can throw it, the location of where the ball typically travels, and much more.

As a long time baseball fan, I decided that I would investigate this prediction further, and determine if there could be a mathematical science to predicting a pitchers pitch. I will attempt to analyze various models for predicting whether the next pitch will be a fastball or a curveball using historical MLB pitch data as the basis of these models. I will conduct this analysis through the use of a Generalized Linear Model, a Bayesian Generalized Linear Model, and Lasso regression model for feature selection. Beyond that, I will then utilize Monte-Carlo simulation to estimate the bias and variance of my models and use mean squared error as a measure of "goodness" of my models. A further description of my methods and processes can be found in the Methods component of this paper.

As one might expect, predicting something as random as what kind of pitch a professional baseball pitcher will throw is extremely difficult. I found that while there are certain indicators dependent on the exact game script that one can use to attempt to predict the next pitch, there is no certainty behind it and it remains as random as flipping a coin. Furthermore, it is a pitchers' job to do exactly the opposite; that is, to make it as difficult as possible for the batter or an organization to be able to predict what pitch will come next. As we have seen with the Houston Astro's pitch-sign stealing scandal, knowing the next pitch can have a huge impact on a game. The Astro's even went on to win the World Series, the MLB Championship. Therefore, pitchers do everything in their power to prevent this from happening.

Data:

I pulled data from 2 sources: kaggle.com and baseballsavant.com. From kaggle, I used 3 datasets. One of which, gave me information on batters, and included vital information such as the inning and the outs. The second included information on the pitches thrown, giving me things such as location, speed, type of pitch, etc. for each individual pitch thrown from 2015 to 2018. The third dataset gave me player names associated with player ID's in the first two tables. Using all three datasets, I joined the data together using common ID's and consolidated into one table. I then found additional pitching information by pitcher from baseball savant, such as the percentages of the type of pitch each pitcher threw. Once I had my table for that data, I then combined with my all-encompassing dataset by player name. Due to large amounts of missing or inconsistent data, I trimmed the dataset to include on the pitching data in years 2017 and 2018.

The dataset, after removing all missing data and excluding years 2015 and 2016, contained 340,950 rows where each row is one pitch, and 23 features. These 23 features are Year, at-bat ID, pitcher name, outs, inning, previous pitch speed, previous pitch location horizontally, previous pitch location vertically, pitch type, ball count, strike count, pitch number (of current batter), number of runners on base, and then the pitcher's 4-seam fastball%, slider %, change-up%, curve-ball %, sinker %, cutter %, splitter %, fastball%, breaking %, and offspeed %. My binary response variable is the pitch type, as that is what I am trying to predict. Due to the many different types of pitches available and shown in the dataset, and given that certain pitchers don't throw certain variations of pitches, I decided to group by category. Each pitch type was grouped into one of four classes for prediction: Fastballs, Modified Fastballs, Curveballs, and Offspeed. Fastballs include the pitches 4-seam fastball, 2-seam fastball, the two most common fastball pitches. Modified Fastballs included the cutter, and sinker. These are variations of a fastball. Curveballs include the slider, curveball, knuckle curveball, and knuckleball pitches. They are all variations of a curveball. Lastly, Offspeed includes the change-up and splitter pitches, as they are both considered "offspeed".

Once I finalized this dataset, I found a concerning issue. As you can see from Figure 1, the distribution of the types of pitches thrown is skewed severely. This is a result of the fact that almost every single pitcher can throw some form of the Fastball well, while the Offspeed and Curveball pitches tend to be a specialty of sorts for most other pitchers. This causes us to have a significant bias in our dataset where any prediction model we use will put to great an emphasis on the Fastball pitch.

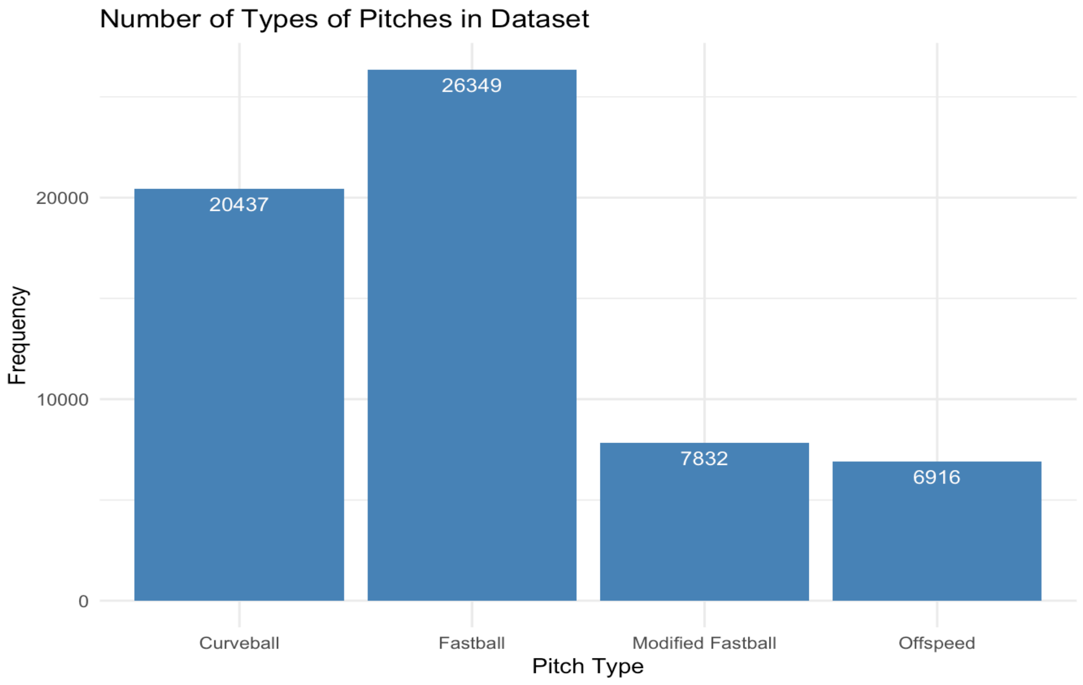
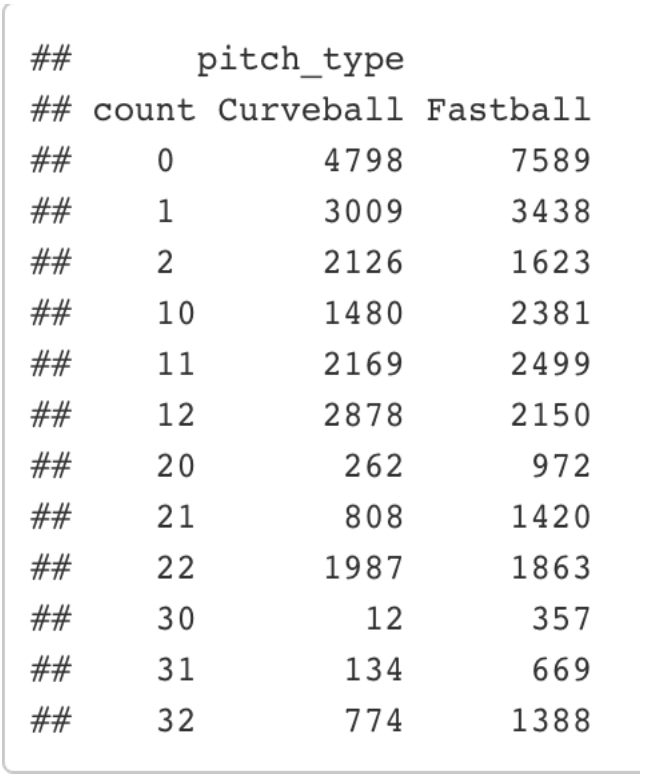


Figure 1: Bar plot of each pitch type in the dataset, plotted against the frequency of occurrence in the data set.

To counteract this issue and attempt to balance my data, I tightened my inclusion criteria. I selected 10 of the top pitchers in years 2017 and 2018 from the data set. This trimmed the data down to 46,786 pitches. Furthermore, I removed the pitches of Offspeed and Modified Fastball from my dataset, as there was an insignificant quantity of them present in the dataset. Further data exploration that I conducted was to see the types of pitches thrown (fastballs and curveballs) for every single batter count. Figure 2 shows the table output, and the frequency of each type of pitch thrown for every possible batter count.



```
##      pitch_type
## count Curveball Fastball
##     0      4798     7589
##     1      3009     3438
##     2      2126     1623
##    10      1480     2381
##    11      2169     2499
##    12      2878     2150
##    20       262      972
##    21       808     1420
##    22      1987     1863
##    30        12      357
##    31       134      669
##    32       774     1388
```

Figure 2: Table of the frequencies of each type of pitch thrown for every batter count. Count 0 indicates 0-0, 0 balls and 0 strikes count, Count 12 indicates 1-2 count (1 ball, 2 strikes,) and so on.

As you can see from the table, the favored pitch is usually the fastball. It is a popular pitch that nearly every pitcher can throw so it is intuitive that it boasts high frequencies for every count. However, it is apparent that when the count is 0-1(0 balls and 1 strikes = “1”), 0-2, and 1-2, which in baseball terms is when the pitcher is ahead in the count, the number of curveballs is significantly greater. Specifically, at 1-2, the number of curveballs thrown exceeds the number of fastballs thrown which is an interesting exception. However, when the pitcher is behind like with a count of 3-0 (3 balls, 0 strikes = “30”), curveballs are thrown only 3% of the time, which is an extraordinarily low percentage.

Methods:

After building a finalized dataset, I was then able to proceed with my methodology for analysis. The first step was to conduct dimension reduction and determine which predictors were most important in predicting whether the next pitch was going to be a Fastball or a Curveball. The shrinkage method that I chose to use was the Lasso regression of a Generalized Linear Model. The reason I chose to do Lasso regression over Ridge regression (another popular shrinkage method) was due to the interesting feature that Lasso shrinks coefficients to zero, effectively performing variable selection during the regression process. Figure 2 shows the coefficients of the Lasso regression for each predictor in my dataset, and it is clear which variables have shrunk to 0 (denoted by the "."). Furthermore, I was able to generate predictions of my Lasso regression model and I compared this to the actual data to get a prediction error for my model.

Once I had shrunk my model parameters down to the important predictors, I was then able to build my GLM model of determining pitch type. From here, I split my data into a training set and a test set. The split I used was by year, where 2017 data represented my training data and 2018 was the test data. I used cross-validation to determine the performance of this model fitting procedure by classification error. That is, I used my training data to fit a model, and then used that model to predict pitches and compared this to my test data. The results were underwhelming, as there was a relatively large classification error.

After completing my model fit, I then used Monte-Carlo simulation to estimate the operating characteristics of estimators: bias, variance, and Mean

Squared Error (MSE). MSE builds off of finding the bias and variance of my model using the formula:

$$MSE = Var(\theta) + \ bias(\theta)^2$$

Similar to rejecting a true null hypothesis, we can under/over estimate the truth on average:

$$\left|E(\hat{\theta}) - \ \theta\right| > 0$$

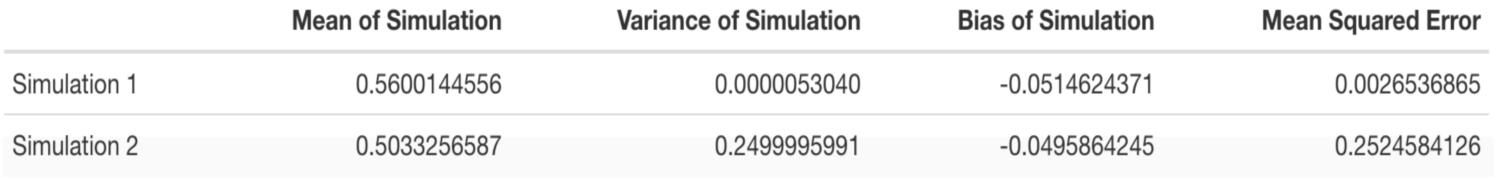
We call $\left|E(\hat{\theta}) - \ \theta\right|$ the bias of the estimator, and the estimator is unbiased if $E(\hat{\theta}) = 0$. When we are determining how often we could reject a false null hypothesis, we are finding the power of the test. The equivalent for estimation is the variance of the sampling distribution. Monte-Carlo simulation gives me a way to quantifiably test the validity of my model. Reporting a low MSE, through minimal variance and ideally no bias, is a sign of a strong model fit for the data. However, due to the complexity of the data, it should be noted not to expect such clean results. As we have already seen, the classification error for my model is significantly large, and the following results and analyses are contingent upon this fairly inaccurate model.

I utilized 2 Monte-Carlo simulations for my data. The first simulation is a fairly basic simulation where I took the percentage of Fastballs in my dataset (26,349/46786 = 0.56) and the percentage of curveballs in my dataset (20,437/46786 = 0.44) and set these as the probability weights of a binomial sampling procedure. Each sample was of size 46,786 and I repeated this 23,454 times (the size of my test data). The second simulation was a little more complex. I made assumptions on the distributions of my predictor variables. I gave all binary/factor predictors an equal probability of selection at each level and using a binomial distribution, I generated data. I normally distributed the previous pitch speed and previous pitch location using the means of these values from my dataset as the means of this distribution. From there, I then used my GLM model formula given the coefficients of each predictor to build an equation that will predict a pitch. I set a cutoff value for the output of this equation, where if it was <=0.5 then it was classified as a curveball and if it was >0.5 then it was classified as a fastball. I finally then compared the MSE, means and variances of the two simulations and drew conclusions about my model overall.

<u>Simulations:</u>

As noted above in the Methodology section, I ran 2 simulations for data generation. The first simulation generated data on a basic binary sampling method with weighted probabilities. In my dataset, I found that there were 26, 349 fastballs and 20,437 curveballs thrown. I used this to assign a weighted probability distribution with p(fastball) = 26,349/46,786 = 0.56 and p(curveball) = 20,437/46786 = 0.44. I generated 46,786 samples drawing from a sample space of {0, 1}, where 0 represents a curveball and 1 represents a fastball, along with their respective probabilities. My Monte Carlo simulation then replicated the generation of these samples 10,000 times. I reported a mean of 0.560017 and a variance of 5.309e-06. The mean value indicates that in every sample, there was on average 56% of the pitches generated were fastballs. This is intuitive given that I had preset the distribution of the fastballs to curveballs to this value. My bias of this simulation was -0.051462, which is fairly low. This was found by subtracting the expected value of pitch type in my test data from the expected value of pitch type in my simulation data. The Mean Squared error was then found to be 0.00265, which is an extremely low MSE. Again, this is an intuitive result as the simulation data was modeled after the pitch type distribution in my dataset.

In the second Monte-Carlo simulation, I applied my GLM model to the predictor values to determine a pitch type. I started by generated data for the count, the outs, the inning, the previous pitch speed, and the previous pitch location. The count, outs, and inning were all generated using a sampling method where the possible outcomes were the levels of each factor, and each level had equal opportunity of being selected. I then used the assumption that previous pitch speed and previous pitch location were normally distributed and drew values from normal distributions where the mean and standard deviation were the respective means and standard deviations of these predictors in the dataset. From there, I then aggregated the coefficients of each level of the factor-type predictors in my GLM model to get a coefficient for that factor. I determined the pitch class using the linear model of the GLM, referring to the coefficients reported in the model summary and aggregated coefficients that I found. I then set a cutoff and classified the pitch as a curveball if the value was less than the cutoff, and fastball otherwise. I found that the mean of this simulation was about 0.50332, which means that about 50% of the pitches were Fastballs. This is a little off from what our expectation was, however it is still pretty close. The variance of the simulation was 0.2499, which is relative high. The bias was -0.0495, which is lower than what we found in the first simulation and the overall MSE was 0.2524. Figure 3 depicts the results of the operating characteristics of the simulations.



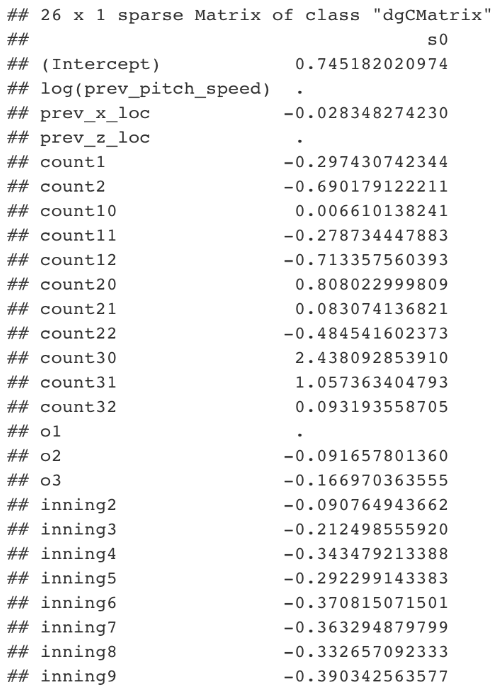
	Mean of Simulation	Variance of Simulation	Bias of Simulation	Mean Squared Error
Simulation 1	0.5600144556	0.0000053040	-0.0514624371	0.0026536865
Simulation 2	0.5033256587	0.2499995991	-0.0495864245	0.2524584126

Figure 3: Table of the results of the operating characteristics of the simulations. Simulation 1 represents the simple data generation. Simulation 2 represents the data generation under the GLM model.

Interestingly enough, with the application of the GLM model in the generation of classifying pitches we saw a lower bias (although not by much) than in the simple data classification method. We can then conclude that Simulation 2 has a smaller type 1 error, or a smaller probability of rejecting a true null hypothesis, than Simulation 1. However, Simulation 2 has a substantially greater power than simulation 1, indicating that it would more often fail to reject false null hypotheses.

Analysis:

I will first analyze the Lasso regression performed to reduce the dimensions of my dataset/model. This shrinkage method was beneficial for this because it shrinks non-significant predictors to 0 directly, and therefore performs dimension reduction simultaneously. Figure 4 reports the output of the Lasso regression dimension reduction.



```
## 26 x 1 sparse Matrix of class "dgCMatrix"
##                                      s0
## (Intercept)            0.745182020974
## log(prev_pitch_speed)  .
## prev_x_loc            -0.028348274230
## prev_z_loc             .
## count1                -0.297430742344
## count2                -0.690179122211
## count10                0.006610138241
## count11               -0.278734447883
## count12               -0.713357560393
## count20                0.808022999809
## count21                0.083074136821
## count22               -0.484541602373
## count30                2.438092853910
## count31                1.057363404793
## count32                0.093193558705
## o1                     .
## o2                    -0.091657801360
## o3                    -0.166970363555
## inning2               -0.090764943662
## inning3               -0.212498555920
## inning4               -0.343479213388
## inning5               -0.292299143383
## inning6               -0.370815071501
## inning7               -0.363294879799
## inning8               -0.332657092333
## inning9               -0.390342563577
```

Figure 4: Output matrix of Lasso regression coefficients. Non-significant coefficients that are shrunk to 0 are represented by a "." and important predictor coefficients are reported.

We can see that there were 3 variables that were shrunk to 0: previous pitch speed, previous z location (height of ball when it crosses home plate), and when there is 1 out. In order to compute an optimal Lasso regression, I utilized cross-validation to select the minimum lambda and then generated the results reported in Figure 3. An additional feature of Lasso regression is that I was able to generate prediction based off of my GLM lasso regression model. I compared these predicted values with the actual values (values being pitch type) and found that the prediction model had a misclassification error of 0.40781. This is notably high for a prediction error and indicates that the model used is not an optimal model for predicting a type of pitch, however, I do find particular interest in the fact that the model predicts with greater than 50% accuracy. Since there are only two outcomes, a fastball or a curveball, my expectation was for the error to be close to 50%. Given that it is about 60%, I conclude that the model I have constructed, while still missing key information and predictors about the game script and other factors in play, seems to be on the right track.

Once I was able to remove unnecessary predictors from my model, I then fit the training data to a Generalized Linear Model (GLM) and used it as a basis for prediction on the test data. The prediction of the GLM model turned out to be different from the prediction of the Lasso regression. I found that the expected value of the prediction of pitch type was 0.7138, indicating that the prediction model had predicted 71% of pitches to be fastballs. The actual expected value of the test data pitch type was 0.5529 indicating 55% were fastballs. We can see that our prediction model was overestimating the truth on average and reported a bias of 0.1609. This bias is smaller in magnitude than the bias we found using Monte-Carlo simulation. Figure 5 reports the results of this prediction method.

	Prediction	Actual
Mean	0.7138143283	0.552912083226742
Bias	0.1609022451	NA

Figure 5: Table of the results of the GLM model prediction. Reports the expected value of classification and the bias of our predicted model.

Discussion:

It is clear that the model I have generated to attempt to predict pitches is neither accurate nor reliable. My misclassification error in using our model to predict what the next pitch will be is very large. We are over predicting the probability of a fastball pitch and the reliance of such a model of prediction is not recommended. We have a relatively large bias, and a large MSE found when running a Monte-Carlo simulation on the model and our Lasso regression model prediction reported an even larger error of prediction. Overall, the results of this analysis indicate a much deeper complexity of a model is required in order to predict a pitch with statistical accuracy.

The main focus of this research was to determine if, to some degree of accuracy, I could predict whether the next pitch was going to be a fastball or a curveball. I developed a model using in-game factors such as the number of outs, the inning, the previous pitch's speed and location, and the batter's count. While our results were not completely out of the realm of trustworthy prediction, it was clear that our model required deeper complexity and potentially more predictors. One shortcoming of this research was the lack of clear and consistent data that easily allows for analysis.

Despite the flaws with this research, I was able to produce a model that was able to predict with a decently small amount of bias. My Monte-Carlo simulation showed that using this GLM model to predict a pitch type reports a lower bias than a model that predicts given the dataset probabilities of a pitch type, which I found quite impressive.

Overall, while I cannot conclude that this research confirms that the prediction of a fastball or curveball is evident, I cannot conclude it is not evident either. My results reveal that I am on the right track and given further analysis, a more complex all encompassing model, and even additional predictors that might hold an impact, the prediction of a pitch is all but in the realm of possibility. Perhaps, future analysis of predicting a pitch given a specific pitcher may yield more accurate prediction results as every pitcher has their own quirks and tells of what pitch they are going to throw. Aggregating this data across 10 different pitchers can dilute these effects and in turn have a negative effect on the prediction performance rather than improve it.

It is no surprise that it was difficult to predict something that is not meant to be predicted. However, I do believe this research can serve as a foundation of further analysis into the topic and building block to the formulation of a model that can accurately predict a pitchers' pitch.

Bibliography

Data:

Schale, Paul. "MLB Pitch Data 2015-2018." *Kaggle*, 16 Apr. 2019, www.kaggle.com/pschale/mlb-pitch-data-20152018#pitches.csv.

Motivation:

Bell, Alexander. "Predicting the Next Baseball Pitch: Utilizing Machine Learning to Gain an Advantage." *Medium*, Medium, 1 Nov. 2018, medium.com/@alexwbell/predicting-the-next-baseball-pitch-utilizing-machine-learning-to-gain-an-advantage-cef35bbd6e26.

More Information:

Agresti, Alan. *Foundations of Linear and Generalized Linear Models*. Wiley-Blackwell, 2015.

Robert, Christian P., and George Casella. *Introducing Monte Carlo Methods with R*. Springer, 2010.

Final Project

Nakoul Makim (namakim)

3/22/2020

R Program’s external requirements

```
library(tidyverse)
```

```
## ── Attaching packages ─────────────────────────────────────── tidyverse 1.3.0 ──
```

```
## ✓ ggplot2 3.3.0     ✓ purrr   0.3.4
## ✓ tibble  3.0.0     ✓ dplyr   0.8.5
## ✓ tidyr   1.0.2     ✓ stringr 1.4.0
## ✓ readr   1.3.1     ✓ forcats 0.5.0
```

```
## ── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(caret)
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

```
library(dplyr)
library(ggplot2)
library(standardize)
library(nnet)
library(glmnet)
```

```
## Loading required package: Matrix
```

```
##
## Attaching package: 'Matrix'
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
```

```
## Loaded glmnet 3.0-2
```

```
library(arm)
```

```
## Loading required package: MASS
```

```
##
## Attaching package: 'MASS'
```

```
## The following object is masked from 'package:dplyr':
##
##     select
```

```
## Loading required package: lme4
```

```
##
## arm (Version 1.10-1, built: 2018-4-12)
```

```
## Working directory is /Users/nakoulmakim/Documents/Stats 406/Final Project
```

```
##
## Attaching package: 'arm'
```

```
## The following object is masked from 'package:standardize':
##
##     standardize
```

```
library(multcomp)
```

```
## Loading required package: mvtnorm
```

```
## Loading required package: survival
```

```
##
## Attaching package: 'survival'
## The following object is masked from 'package:caret':
##
##     cluster
```

```
## Loading required package: TH.data
```

```
##
## Attaching package: 'TH.data'
```

```
## The following object is masked from 'package:MASS':
##
##     geyser
```

Data Cleaning

```
#at-bat data
at_bats <- read_csv("atbats.csv")
```

```
## Parsed with column specification:
## cols(
##   ab_id = col_double(),
##   batter_id = col_double(),
##   event = col_character(),
##   g_id = col_double(),
##   inning = col_double(),
##   o = col_double(),
##   p_score = col_double(),
##   p_throws = col_character(),
##   pitcher_id = col_double(),
##   stand = col_character(),
##   top = col_logical()
## )
```

```
#at_bats

#pitch data
pitches <- read_csv("pitches.csv")
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   code = col_character(),
##   type = col_character(),
##   pitch_type = col_character()
## )
## See spec(...) for full column specifications.
```

```
#pitches

#player name data
player_names <- read_csv("player_names.csv")
```

```
## Parsed with column specification:
## cols(
##   id = col_double(),
##   first_name = col_character(),
##   last_name = col_character()
## )
```

```
#combine first and last name of players
player_names$player_name <- paste(player_names$first_name, player_names$last_name)
players <- subset(player_names, select = c("id","player_name"))
#players
```

```
#trimmed datasets
at_bats_trimmed <- subset(at_bats, select = c("ab_id", "batter_id", "pitcher_id", "innin
g", "o", "p_score", "p_throws", "stand"))
#at_bats_trimmed

pitches_trimmed <- subset(pitches, select = c("ab_id","end_speed", "px", "pz", "pitch_ty
pe","b_count","s_count","outs","pitch_num","on_1b","on_2b","on_3b"))
#pitches_trimmed
```

```
# match the column names of the 2 tables in order to join
names(at_bats_trimmed) [names(at_bats_trimmed) == "batter_id"] <- "id"
names(at_bats_trimmed) [names(at_bats_trimmed) == "pitcher_id"] <- "id_p"

# joining batter and player name data
combined <- full_join(at_bats_trimmed, players, "id")

# joining pitcher and player name data
names(players)[names(players) == "id"] <- "id_p"
combined_player_data <- full_join(combined, players, "id_p")
#combined_player_data
```

```
# Trimming the data more, removed batter name, player ID's, score data
clean_data <- subset(combined_player_data, select = c("ab_id","player_name.y","inning",
"o","p_throws","stand"))

# combined pitch data with batter data
combined_bat_pitch <- full_join(clean_data, pitches_trimmed, "ab_id")
#combined_bat_pitch
```

```
# added year column
combined_bat_pitch$year <- strtrim(combined_bat_pitch$ab_id, 4)

# removed data from 2015 and 2016 years due to large amounts of missing data in those ye
ars
final_data <- subset(combined_bat_pitch, year %in% c("2017","2018"))
names(final_data)[names(final_data) == "player_name.y"] = "pitcher_name"
#final_data
```

```
# read in additional pitching stats for each pitcher from 2017-2019
pitcher_stats <- read_csv("stats.csv")
```

```
## Warning: Missing column names filled in: 'X20' [20]
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   last_name = col_character(),
##   first_name = col_character(),
##   X20 = col_logical()
## )
```

```
## See spec(...) for full column specifications.
```

```
# combine first and last name of pitchers
pitcher_stats$pitcher_name <- paste(pitcher_stats$first_name, pitcher_stats$last_name)

# make pitch-type column names more readable and easier to understand
names(pitcher_stats) [names(pitcher_stats) == "n_ff_formatted"] <- "4-seam %"
names(pitcher_stats) [names(pitcher_stats) == "n_sl_formatted"] <- "slider %"
names(pitcher_stats) [names(pitcher_stats) == "n_ch_formatted"] <- "change-up %"
names(pitcher_stats) [names(pitcher_stats) == "n_cukc_formatted"] <- "curve ball %"
names(pitcher_stats) [names(pitcher_stats) == "n_sift_formatted"] <- "sinker %"
names(pitcher_stats) [names(pitcher_stats) == "n_fc_formatted"] <- "cutter %"
names(pitcher_stats) [names(pitcher_stats) == "n_fs_formatted"] <- "splitter %"
names(pitcher_stats) [names(pitcher_stats) == "n_fastball_formatted"] <- "fastball %"
names(pitcher_stats) [names(pitcher_stats) == "n_breaking_formatted"] <- "breaking %"
names(pitcher_stats) [names(pitcher_stats) == "n_offspeed_formatted"] <- "offspeed %"

# when a pitcher cannot throw certain type of pitch, replace NA with 0
pitcher_stats[is.na(pitcher_stats)] = 0

# Trim the additional pitch stats
pitcher_stats_trimmed <- subset(pitcher_stats, select = c("pitcher_name","year","p_ball"
,"p_called_strike","exit_velocity_avg","4-seam %","slider %","change-up %","curve ball
 %","sinker %","cutter %","splitter %","fastball %","breaking %","offspeed %","pitch_cou
nt"))
#pitcher_stats_trimmed
```

```r
# seperate dataset by year: 2017, 2018, 2019
pitcher_2017 <- subset(pitcher_stats_trimmed, year %in% "2017")
pitcher_2018 <- subset(pitcher_stats_trimmed, year %in% "2018")
pitcher_2019 <- subset(pitcher_stats_trimmed, year %in% "2019")

pitch_2017 <- subset(final_data, year %in% "2017")
pitch_2018 <- subset(final_data, year %in% "2018")
pitch_2019 <- subset(final_data, year %in% "2019")


combined_2017 <- full_join(pitch_2017, pitcher_2017, "pitcher_name")
combined_2018 <- full_join(pitch_2018, pitcher_2018, "pitcher_name")
combined_2017$ab_id <- substr(combined_2017$ab_id, 5, 10)
combined_2018$ab_id <- substr(combined_2018$ab_id, 5, 10)
#combined_2017
#combined_2018
```

```r
baseball_temp <- rbind(combined_2017, combined_2018)
baseball <- subset(baseball_temp, select =-c(year.y))
baseball$event <- NULL
baseball$pitch_count <- NULL
baseball$year.y <- NULL
baseball$p_ball <- NULL
baseball$p_called_strike <- NULL
baseball$on_base <- baseball$on_1b + baseball$on_2b + baseball$on_3b
baseball$on_1b <- NULL
baseball$on_2b <- NULL
baseball$on_3b <- NULL
baseball$p_throws <- NULL
baseball$stand <- NULL
baseball$spin_rate <- NULL
baseball$spin_dir <- NULL
baseball$type_confidence <- NULL
baseball$nasty <- NULL
baseball$code <- NULL
baseball$type <- NULL
baseball$exit_velocity_avg <- NULL
options(digits = 10)
names(baseball)[names(baseball) == "year.x"] = "Year"
baseball$pitch_type <- factor(baseball$pitch_type)
#names(baseball)[names(baseball) == "end_speed"] = "prev_pitch_speed"
#names(baseball)[names(baseball) == "px"] = "prev_x_loc"
#names(baseball)[names(baseball) == "pz"] = "prev_z_loc"
baseball$px <- round(baseball$px, 3)
baseball$pz <- round(baseball$pz, 3)
#baseball
baseball_2018 <- subset(baseball, baseball$Year == "2018")
#baseball_2018[complete.cases(baseball_2018),]
```

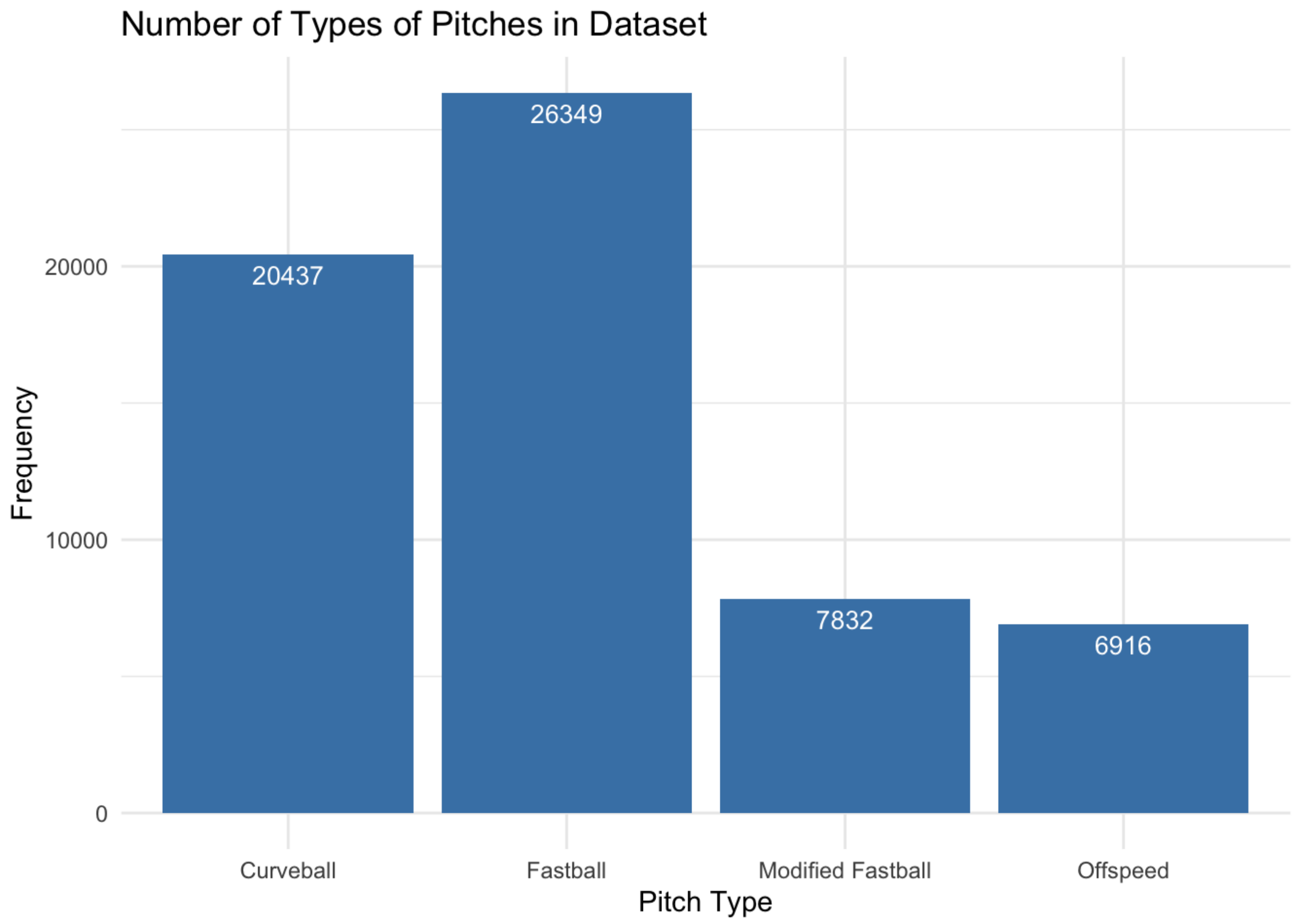
```
baseball <- mutate(baseball, prev_pitch_speed = lag(end_speed))
baseball <- mutate(baseball, prev_x_loc = lag(px))
baseball <- mutate(baseball, prev_z_loc = lag(pz))
baseball_no_na <- na.omit(baseball)
#baseball
baseball_no_na <- baseball_no_na[!(baseball_no_na$pitch_type %in% c("EP","FO","IN","PO",
"SC","UN","AB")),]
baseball_no_na %>% mutate_if(is.factor, as.character) -> baseball_no_na
baseball_no_na$pitch_type[baseball_no_na$pitch_type %in% c("FF","FT")] <- "Fastball"
baseball_no_na$pitch_type[baseball_no_na$pitch_type %in% c("FC","SI")] <- "Modified Fast
ball"
baseball_no_na$pitch_type[baseball_no_na$pitch_type %in% c("SL","CU","KC","KN")] <- "Cur
veball"
baseball_no_na$pitch_type[baseball_no_na$pitch_type %in% c("CH","FS")] <- "Offspeed"
baseball_no_na$pitch_type <- factor(baseball_no_na$pitch_type)
#baseball_no_na
```

```
mlb_temp <- subset(baseball_no_na, baseball_no_na$inning %in% c(1, 2, 3, 4, 5, 6, 7, 8,
9))
mlb_temp$count <- paste0(mlb_temp$b_count, mlb_temp$s_count)
mlb_temp$count <- as.numeric(mlb_temp$count)
mlb_temp$count <- as.factor(mlb_temp$count)
mlb_temp$inning <- as.factor(mlb_temp$inning)
mlb_temp$o <- as.factor(mlb_temp$o)
#mlb_temp
```

```
mlb <- subset(mlb_temp, pitcher_name %in% c("Max Scherzer","Aaron Nola","Jacob deGrom",
"Corey Kluber","Patrick Corbin","Jake Arrieta","Justin Verlander","Trevor Bauer","Zack G
reinke","Carlos Carrasco"))
#mlb
mlb_2017 <- subset(mlb, Year == "2017")
mlb_2017$pitcher_name <- as.factor(mlb_2017$pitcher_name)
#mlb_2017
mlb_2018 <- subset(mlb, Year == "2018")
mlb_2018$pitcher_name <- as.factor(mlb_2018$pitcher_name)
#mlb_2018
```

Data Exploration

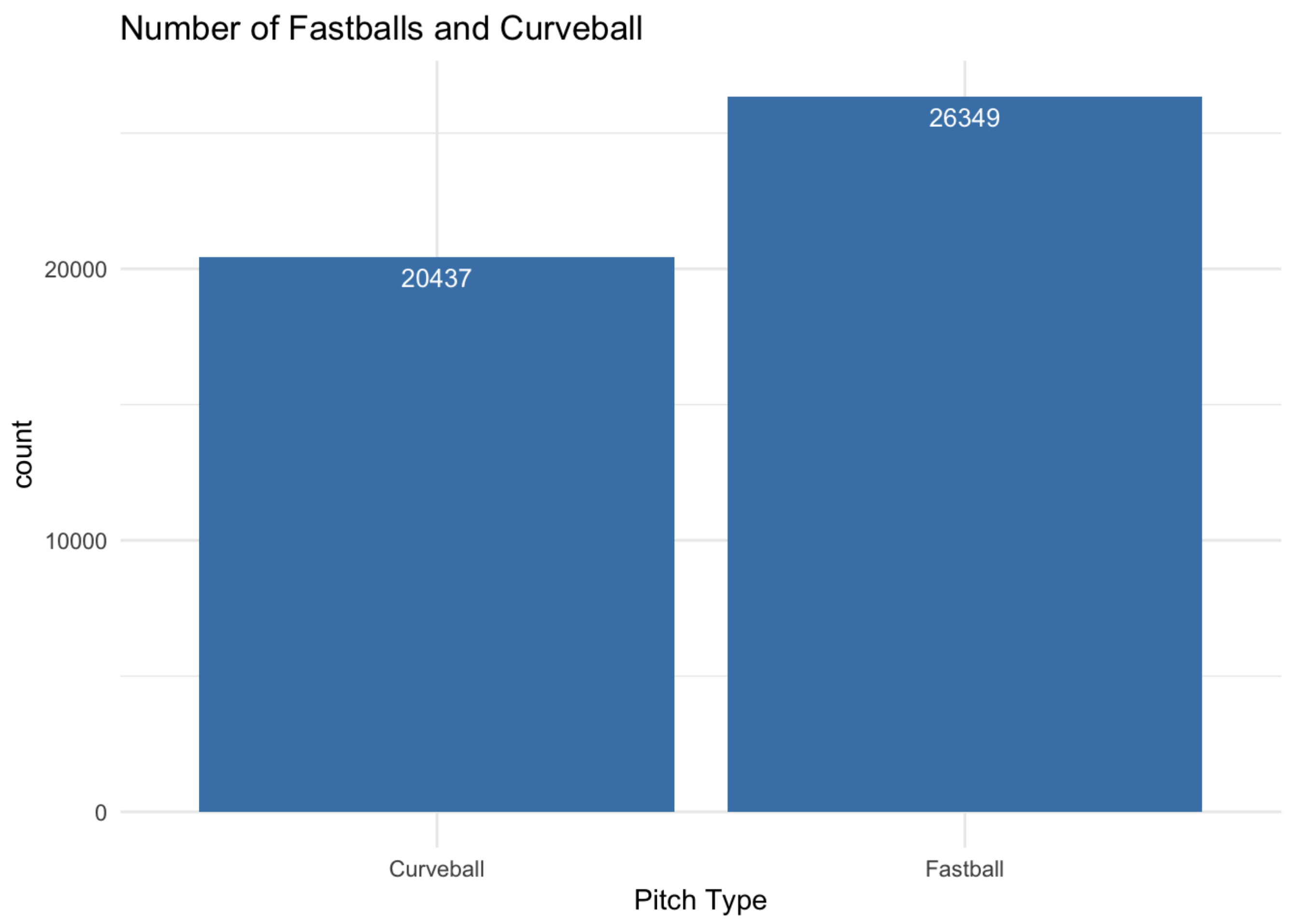
```
ggplot(mlb, aes(x = pitch_type)) + geom_bar(fill = "steelblue") + geom_text(aes(label =
 stat(count)), stat = 'count', vjust = 1.6, color = "white", size = 3.5) + theme_minimal
() + labs(y = "Frequency", x = "Pitch Type", title = "Number of Types of Pitches in Data
set")
```



```
fastball <- subset(mlb, mlb$pitch_type == "Fastball")
curveball <- subset(mlb, mlb$pitch_type == "Curveball")
modified_fast <- subset(mlb, mlb$pitch_type == "Modified Fastball")
offspeed <- subset(mlb, mlb$pitch_type == "Offspeed")
```

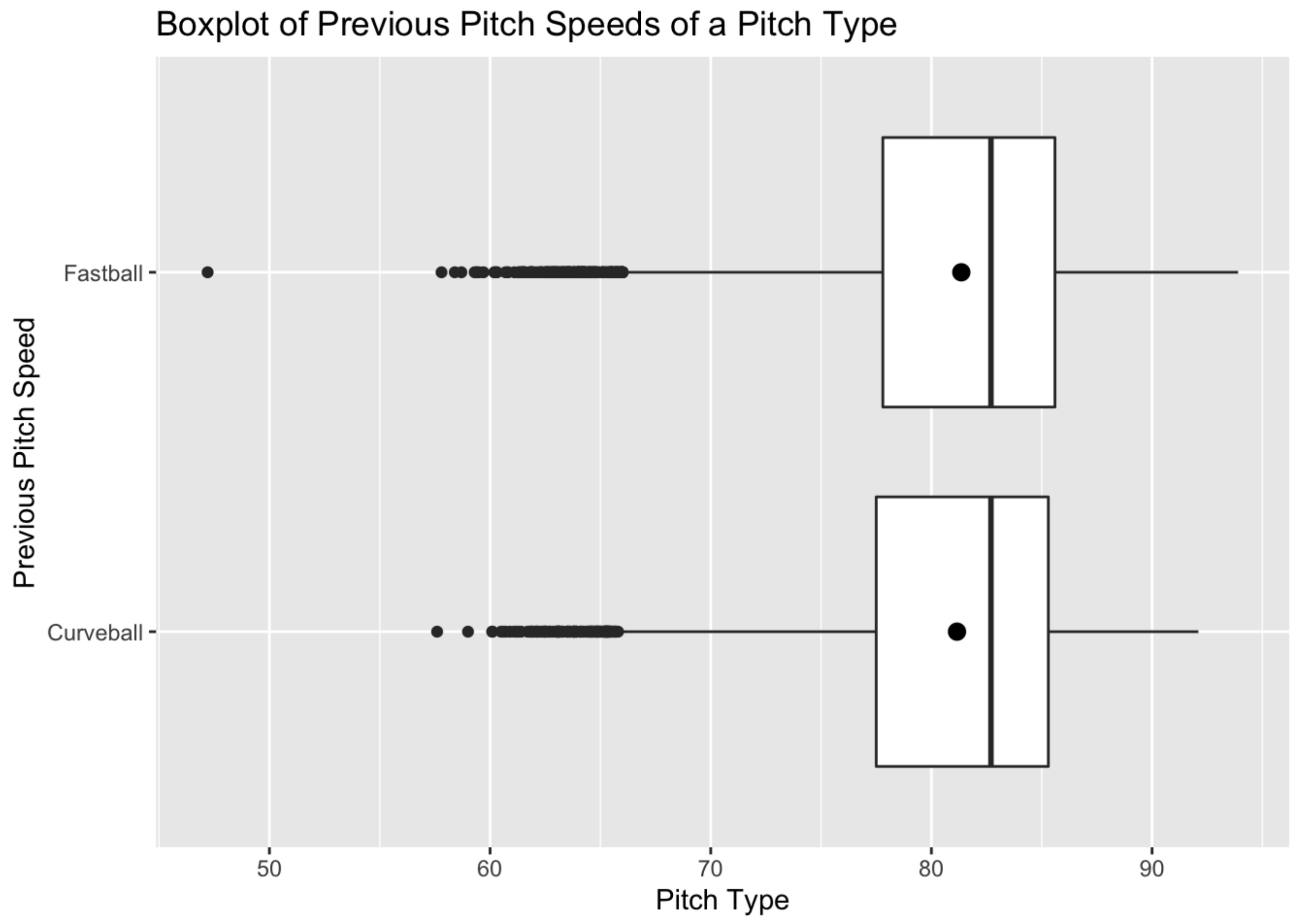
Methodology & Models:

```
mlb_glm <- rbind(fastball, curveball)
mlb_glm$pitch_type <- droplevels(mlb_glm$pitch_type)
train_temp <- subset(mlb_glm, Year == "2017")
test_temp <- subset(mlb_glm, Year == "2018")
x <- which(colnames(train_temp) %in% c("4-seam %", "slider %", "change-up %", "curve bal
l %", "sinker %", "cutter %", "splitter %", "fastball %", "breaking %", "offspeed %"))
train <- train_temp[,-x]
y <- which(colnames(test_temp) %in% c("4-seam %", "slider %", "change-up %", "curve ball
%", "sinker %", "cutter %", "splitter %", "fastball %", "breaking %", "offspeed %"))
test <- test_temp[,-y]
ggplot(mlb_glm, aes(x = pitch_type)) + geom_bar(fill = "steelblue") + geom_text(aes(labe
l = stat(count)), stat = 'count', vjust = 1.6, color = "white", size = 3.5) + theme_mini
mal() + labs(x = "Pitch Type", y = "count", title = "Number of Fastballs and Curveball")
```

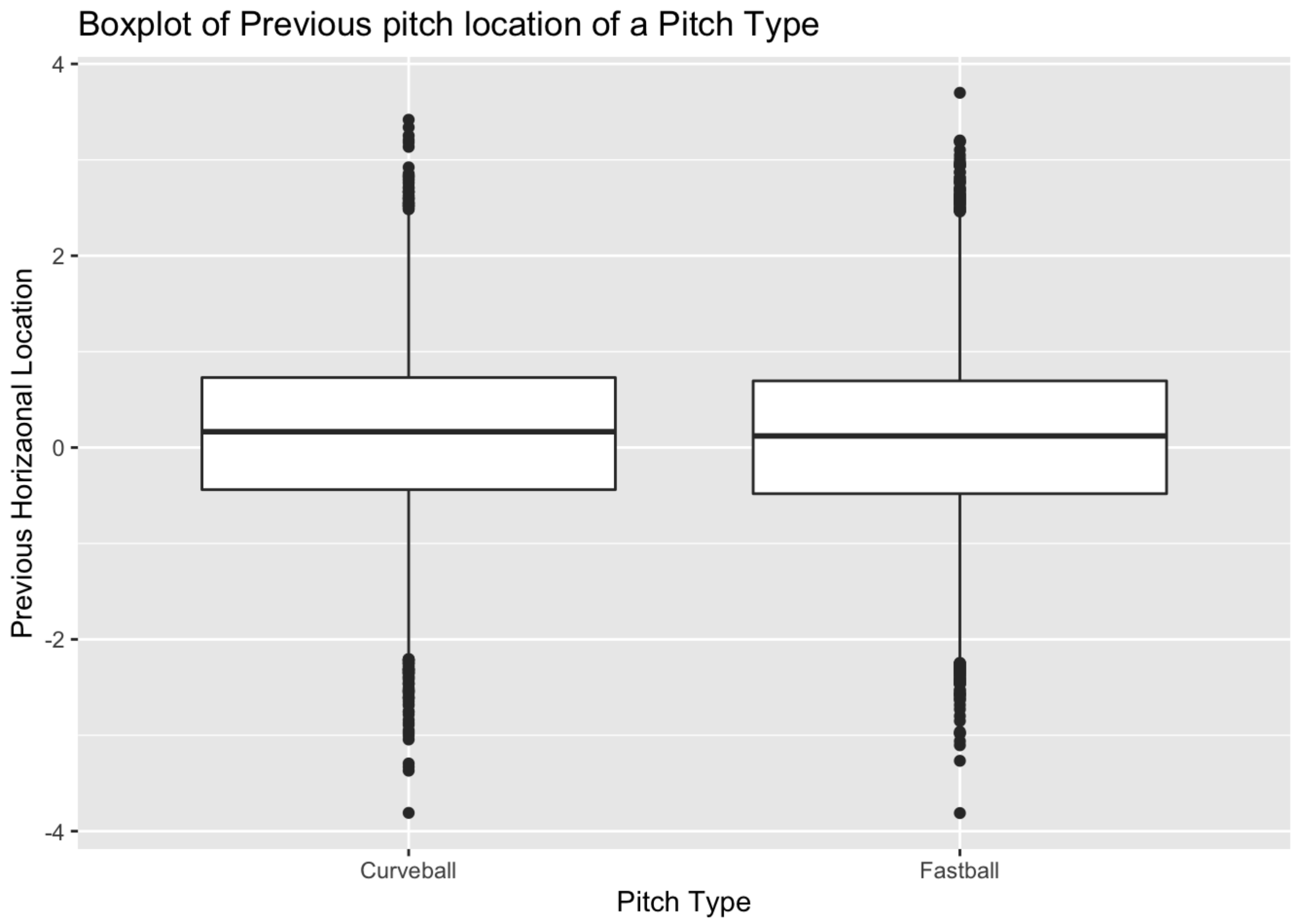


```
ggplot(mlb_glm, aes(x = pitch_type, y = prev_pitch_speed)) + geom_boxplot() + coord_flip
() + stat_summary() + labs(x = "Previous Pitch Speed", y = "Pitch Type", title = "Boxplo
t of Previous Pitch Speeds of a Pitch Type")
```

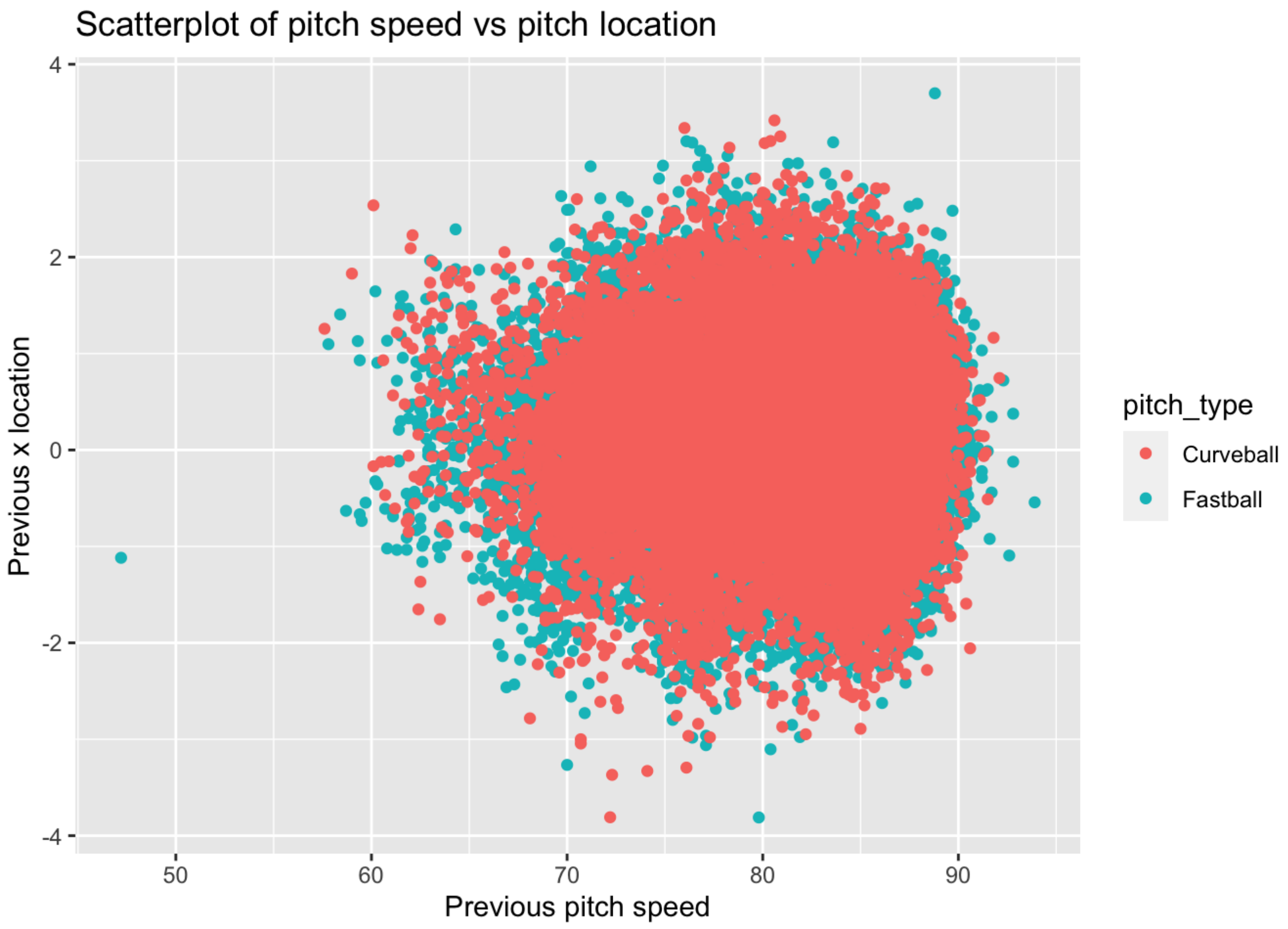
```
## No summary function supplied, defaulting to `mean_se()`
```



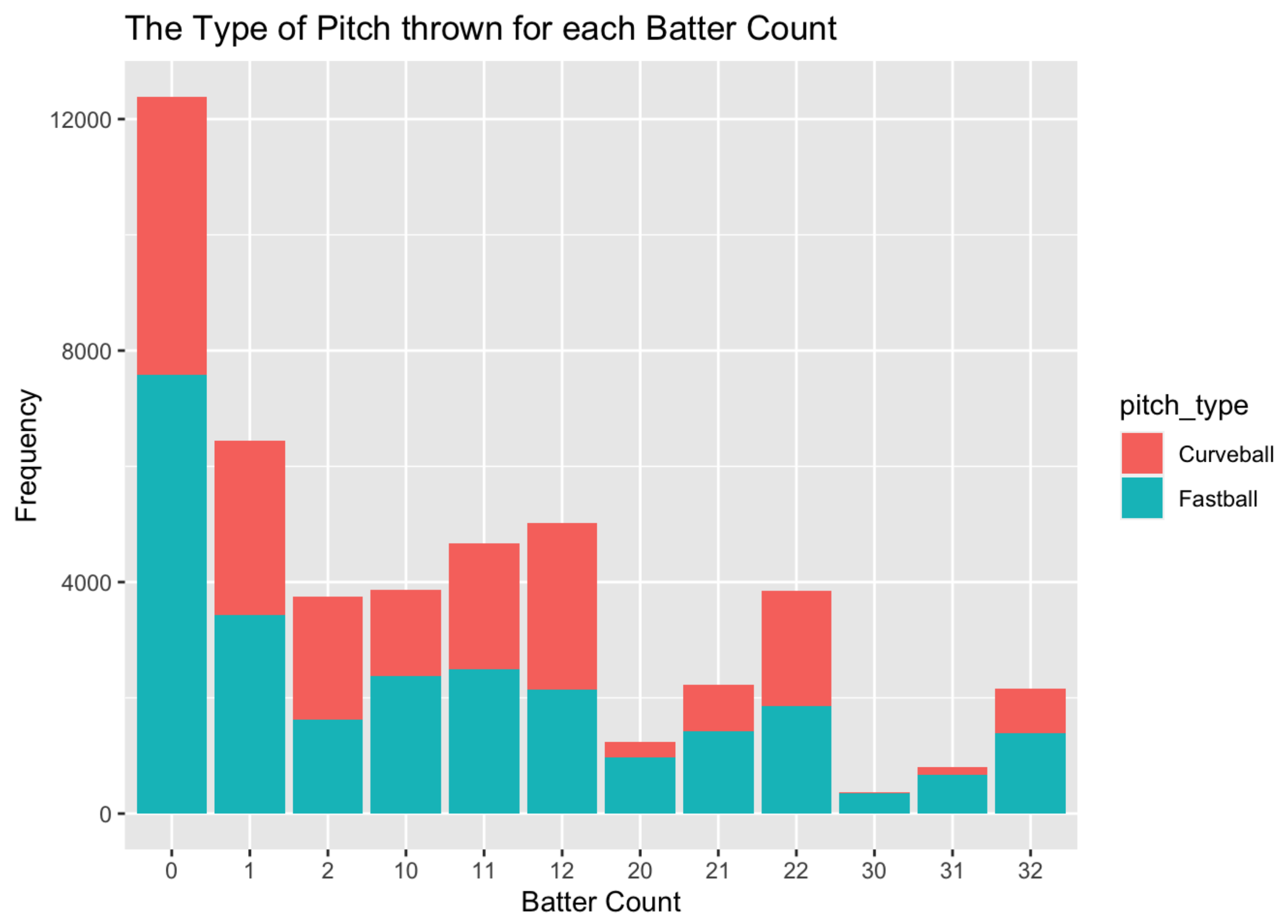
```
ggplot(mlb_glm, aes(x = pitch_type, y = prev_x_loc)) + geom_boxplot() + labs(x = "Pitch
 Type", y = "Previous Horizaonal Location", title = "Boxplot of Previous pitch location
 of a Pitch Type")
```



```
ggplot(mlb_glm, aes(prev_pitch_speed, y = prev_x_loc, color = pitch_type)) + geom_point
() + labs(x = "Previous pitch speed", y = "Previous x location", title = "Scatterplot of
pitch speed vs pitch location")
```



```
ggplot(mlb_glm, aes(count, fill = pitch_type)) + geom_bar() + labs(x = "Batter Count", y
= "Frequency", title = "The Type of Pitch thrown for each Batter Count")
```



More Data Exploration of trimmed dataset

```
with(mlb_glm, table(count, pitch_type))
```

```
##      pitch_type
## count Curveball Fastball
##    0       4798     7589
##    1       3009     3438
##    2       2126     1623
##    10      1480     2381
##    11      2169     2499
##    12      2878     2150
##    20       262      972
##    21       808     1420
##    22      1987     1863
##    30        12      357
##    31       134      669
##    32       774     1388
```

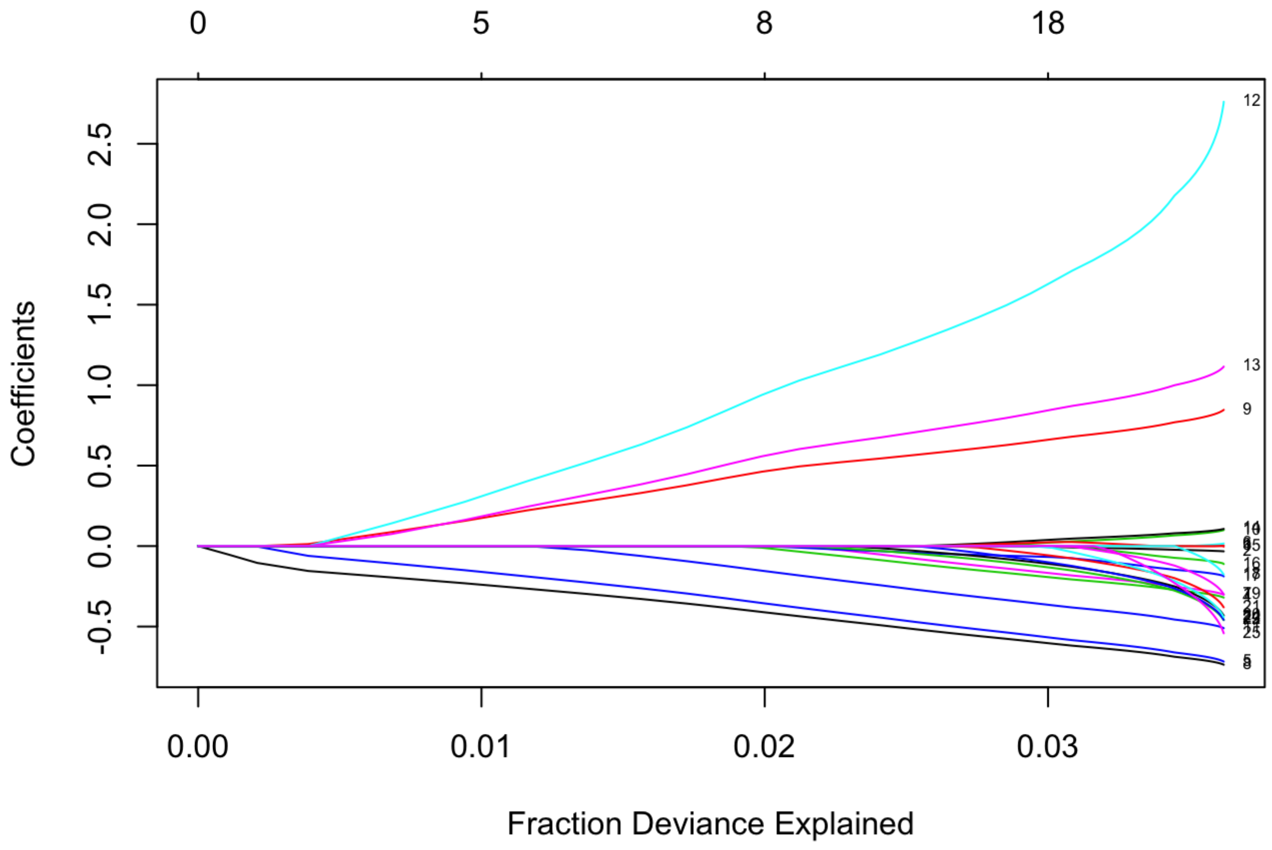
Binary Logistic Regression Model

```
#Binary Logistic Regression
glm_test2 <- glm(pitch_type ~ log(prev_pitch_speed) + prev_x_loc + prev_z_loc + count +
 o + inning, family = "binomial", data = mlb_glm)
summary(glm_test2)
```

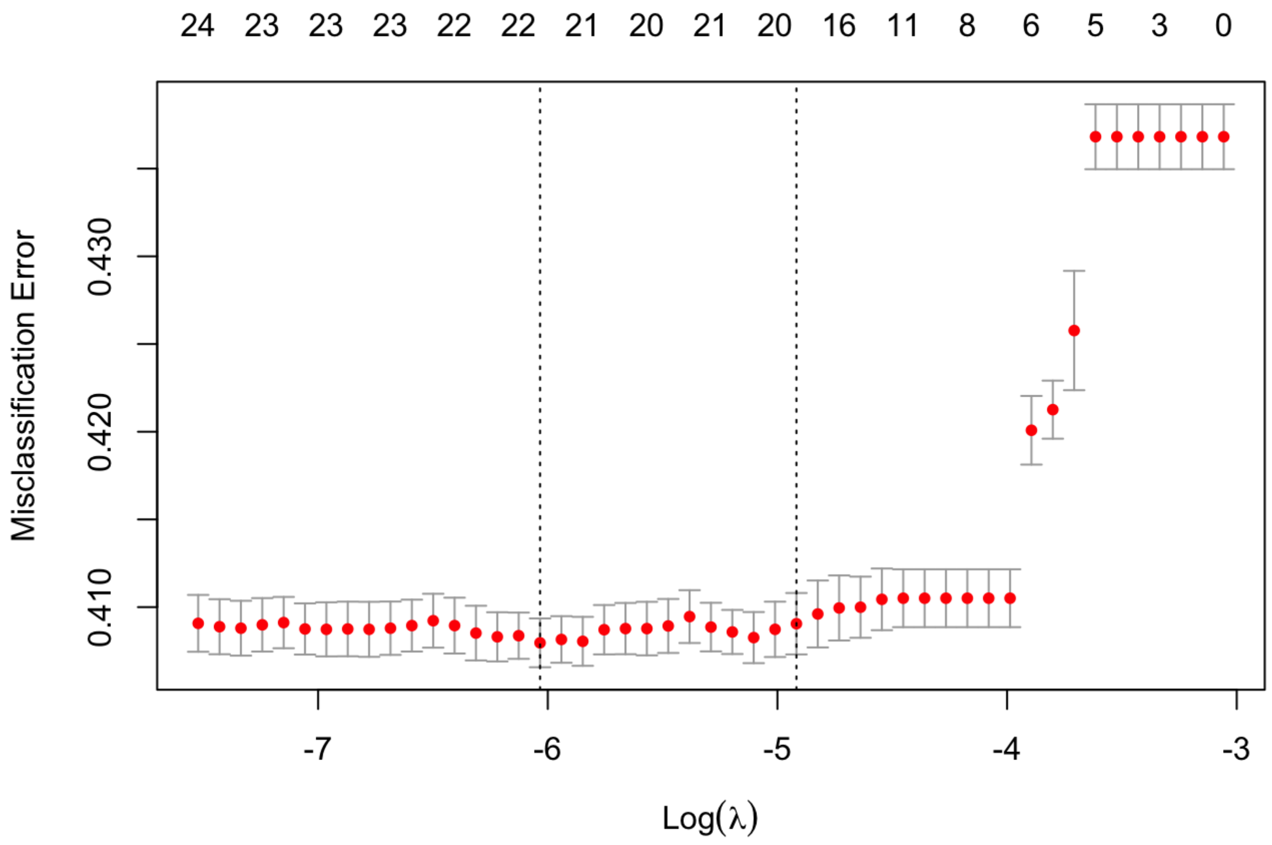
```
##
## Call:
## glm(formula = pitch_type ~ log(prev_pitch_speed) + prev_x_loc +
##     prev_z_loc + count + o + inning, family = "binomial", data = mlb_glm)
##
## Deviance Residuals:
##       Min         1Q     Median         3Q        Max
## -2.748451  -1.212436   0.833736   1.058745   1.484955
##
## Coefficients:
##                         Estimate  Std. Error   z value   Pr(>|z|)
## (Intercept)           1.09154650  0.65671471   1.66213   0.096486 .
## log(prev_pitch_speed) -0.04298339  0.15049711  -0.28561   0.775177
## prev_x_loc            -0.03381849  0.01175227  -2.87761   0.004007 **
## prev_z_loc             0.00680829  0.01143474   0.59540   0.551574
## count1                -0.33036808  0.03126427 -10.56695 < 2.22e-16 ***
## count2                -0.72902380  0.03801903 -19.17524 < 2.22e-16 ***
## count10                0.01875511  0.03811168   0.49211   0.622642
## count11               -0.31292787  0.03485685  -8.97751 < 2.22e-16 ***
## count12               -0.74730276  0.03417292 -21.86827 < 2.22e-16 ***
## count20                0.86208097  0.07228326  11.92643 < 2.22e-16 ***
## count21                0.10616439  0.04801445   2.21109   0.027029 *
## count22               -0.52133391  0.03738369 -13.94549 < 2.22e-16 ***
## count30                2.93329073  0.29398571   9.97766 < 2.22e-16 ***
## count31                1.13902939  0.09673599  11.77462 < 2.22e-16 ***
## count32                0.11160582  0.04880444   2.28680   0.022208 *
## o1                    -0.04193630  0.03515676  -1.19284   0.232933
## o2                    -0.14940100  0.03516906  -4.24808 2.1561e-05 ***
## o3                    -0.22646505  0.03645907  -6.21149 5.2486e-10 ***
## inning2               -0.21836024  0.03406398  -6.41030 1.4524e-10 ***
## inning3               -0.34143329  0.03424110  -9.97145 < 2.22e-16 ***
## inning4               -0.47503949  0.03422932 -13.87815 < 2.22e-16 ***
## inning5               -0.42341392  0.03504091 -12.08342 < 2.22e-16 ***
## inning6               -0.50331650  0.03601886 -13.97369 < 2.22e-16 ***
## inning7               -0.50388809  0.04261442 -11.82436 < 2.22e-16 ***
## inning8               -0.49294777  0.06738916  -7.31494 2.5749e-13 ***
## inning9               -0.61218646  0.13789783  -4.43942 9.0201e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 64110.111  on 46785  degrees of freedom
## Residual deviance: 61783.904  on 46760  degrees of freedom
## AIC: 61835.904
##
## Number of Fisher Scoring iterations: 5
```

Lasso Regression (Dimension Reduction)

```r
X4 <- model.matrix(glm_test2)[,-1]
y4 <- mlb_glm$pitch_type
lasso_mod4 <- glmnet(x = X4, y = y4, standardize = TRUE, family = "binomial")
plot(lasso_mod4, xvar = "dev", label = TRUE)
```



```r
lasso_cv4 <- cv.glmnet(x = X4, y = y4, family = "binomial", type.measure = "class")
plot(lasso_cv4)
```



```
best_lambda4 <- lasso_cv4$lambda.min
best_lambda4
```

```
## [1] 0.002396519362
```

```
lasso_best4 <- glmnet(x = X4, y = y4, family = "binomial", lambda = best_lambda4)
```

```
pred4 <- predict(lasso_best4, s = best_lambda4, newx = X4, type = "class")
final_pred4 <- cbind(levels(y4)[y4], pred4)
head(final_pred4, 30)
```

```
##               1
## 1  "Fastball" "Fastball"
## 2  "Fastball" "Fastball"
## 3  "Fastball" "Fastball"
## 4  "Fastball" "Fastball"
## 5  "Fastball" "Curveball"
## 6  "Fastball" "Fastball"
## 7  "Fastball" "Fastball"
## 8  "Fastball" "Fastball"
## 9  "Fastball" "Curveball"
## 10 "Fastball" "Fastball"
## 11 "Fastball" "Fastball"
## 12 "Fastball" "Fastball"
## 13 "Fastball" "Fastball"
## 14 "Fastball" "Fastball"
## 15 "Fastball" "Curveball"
## 16 "Fastball" "Fastball"
## 17 "Fastball" "Fastball"
## 18 "Fastball" "Fastball"
## 19 "Fastball" "Fastball"
## 20 "Fastball" "Fastball"
## 21 "Fastball" "Fastball"
## 22 "Fastball" "Fastball"
## 23 "Fastball" "Fastball"
## 24 "Fastball" "Fastball"
## 25 "Fastball" "Curveball"
## 26 "Fastball" "Fastball"
## 27 "Fastball" "Fastball"
## 28 "Fastball" "Fastball"
## 29 "Fastball" "Fastball"
## 30 "Fastball" "Fastball"
```

```
coef(lasso_best4)
```

```
## 26 x 1 sparse Matrix of class "dgCMatrix"
##                                    s0
## (Intercept)            0.701838984193
## log(prev_pitch_speed)  .
## prev_x_loc            -0.026015366610
## prev_z_loc             .
## count1                -0.287296985579
## count2                -0.677855770589
## count10                0.002214516172
## count11               -0.267987379499
## count12               -0.702563713650
## count20                0.790840431820
## count21                0.075559821189
## count22               -0.472799835666
## count30                2.317885731546
## count31                1.031725937658
## count32                0.086994598328
## o1                     .
## o2                    -0.083618152843
## o3                    -0.158374207656
## inning2               -0.051220945969
## inning3               -0.172803665461
## inning4               -0.303145126861
## inning5               -0.251796392903
## inning6               -0.329893180062
## inning7               -0.319928713955
## inning8               -0.282571465572
## inning9               -0.321000616827
```

```
df4 <- data.frame(levels(y4)[y4], pred4)
names(df4) <- c("Actual", "Predicted")
#df4
df_truth4 <- subset(df4, df4$Actual == df4$Predicted)
#df_truth4

Lasso_prediction_error <- 1 - (nrow(df_truth4)/ nrow(df4))
Lasso_prediction_error
```

```
## [1] 0.4072585816
```

Fitting the Reduced-GLM and Prediction

```
train$pitcher_name <- as.factor(train$pitcher_name)
test$pitcher_name <- as.factor(test$pitcher_name)

# New GLM Model with Updated Predictors
glm_model <- glm(pitch_type ~ prev_x_loc + log10(prev_pitch_speed) + count + factor(o, e
xclude = c("1")) + inning, family = "binomial", data = train)
#summary(glm_model)
pred <- predict(glm_model, test, type = "response")
pred <- na.omit(round(pred))
pred1 <- mean(pred)

# subtract 1 because in test data, curveball level = 1 and fastball = 2 so it is on 1-2
 scale, not 0-1 scale like our prediction model.
actual <- mean(as.numeric(test$pitch_type)) - 1
bias_pred <- pred1 - actual
pred_df <- data.frame("Mean", pred1, actual)
names(pred_df) <- c("","Prediction", "Actual")
bias_df <- data.frame("Bias", abs(bias_pred), "NA")
names(bias_df) <- c("", "Prediction", "Actual")
pred_act_df <- rbind(pred_df, bias_df)
knitr::kable(pred_act_df)
```

	Prediction	Actual
Mean	0.7138143283	0.552912083226742
Bias	0.1609022451	NA

Monte Carlo Simulation #1

```
set.seed(3343229)
runs <- 23454
mc_func <- function() {
    # 0 = curveball, 1 = fastball
    z <- sample(c(0,1),46786, prob = c(0.44, 0.56), replace = TRUE)
    return(mean(z))
}
mc.sim <- replicate(runs, mc_func())
```

Mean, Bias, Variance, and MSE of Simulation 1

```
mean(mc.sim)
```

```
## [1] 0.5600144556
```

```
var(mc.sim)
```

```
## [1] 5.304022694e-06
```

```
test_df <- as.data.frame(mc.sim)
test_df2 <- mutate(
    test_df,
    test_df$pitch_type <- ifelse(test_df$mc.sim < 0.56, "Curveball", "Fastball")
)
names(test_df2) <- c("MC.Sim Value", "Pitch Type")
#test_df2
theta <- mean(as.numeric(test$pitch_type))
theta_hat <- mean(as.numeric(as.factor(test_df2$`Pitch Type`)))
bias <- theta_hat - theta
bias
```

```
## [1] -0.05146243711
```

```
MSE <- var(mc.sim) + bias^2
MSE
```

```
## [1] 0.002653686456
```

Monte Carlo Simulation #2

```
set.seed(3343229)
mean_prev_xloc <- mean(mlb_glm$prev_x_loc)
sd_xloc <- sqrt(var(mlb_glm$prev_x_loc))

mean_pitch_speed <- mean(mlb_glm$prev_pitch_speed)
sd_pitch_speed <- sqrt(var(mlb_glm$prev_pitch_speed))

runs <- 23454
mc_func2 <- function(){
    # count variables where 0 = "0",....,3 = "10",....12 = "32"
    count <- sample(c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12), 23454, replace = TRUE)
    # innings
    innings <- sample(c(1, 2, 3, 4, 5, 6, 7, 8, 9), 23454, replace = TRUE)

    outs <- sample(c(0, 1, 2, 3), 23454, replace = TRUE)

    x_loc <- rnorm(23454, mean = mean_prev_xloc, sd = sd_xloc)

    prev_speed <- rnorm(23454, mean = mean_pitch_speed, sd = sd_pitch_speed)

    pitch_class <- 0.229996*count + -0.139265*outs + -0.446322*innings + -0.033818*x_loc
+ -0.042988339*prev_speed

    if (mean(pitch_class) <= -4.44335) {
        pitch_class <- 0
    }
    else {
        pitch_class <- 1
    }
    return(mean(pitch_class))
}

mc.sim2 <- replicate(runs, mc_func2())
```

Mean, Variance, Bias, MSE of Simulation 2

```
mean(mc.sim2)
```

```
## [1] 0.5033256587
```

```
var(mc.sim2)
```

```
## [1] 0.2499995991
```

```
test_df3 <- as.data.frame(mc.sim2)
#test_df3
test_df4 <- mutate(
    test_df3,
    test_df3$pitch_type <- ifelse(test_df3$mc.sim2 == 0, "Curveball", "Fastball")
)
names(test_df4) <- c("MC.Sim Value", "Pitch Type")
#test_df2
theta <- mean(as.numeric(test$pitch_type))
theta_hat2 <- mean(as.numeric(as.factor(test_df4$`Pitch Type`)))
bias2 <- theta_hat2 - theta
bias2
```

```
## [1] -0.04958642449
```

```
MSE2 <- var(mc.sim2) + bias2^2
MSE2
```

```
## [1] 0.2524584126
```

Simulation Results

```
MC_df <- data.frame("Simulation 1", mean(mc.sim), var(mc.sim), abs(bias), MSE)
names(MC_df) <- c("", "Mean of Simulation","Variance of Simulation", "Bias of Simulatio
n", "Mean Squared Error")
MC_df2 <- data.frame("Simulation 2", mean(mc.sim2), var(mc.sim2), abs(bias2), MSE2)
names(MC_df2) <- c("","Mean of Simulation","Variance of Simulation", "Bias of Simulatio
n", "Mean Squared Error")
MC <- rbind(MC_df, MC_df2)
knitr::kable(MC, main = "Table of Mean and Variance of Monte-Carlo Simulations")
```

	Mean of Simulation	Variance of Simulation	Bias of Simulation	Mean Squared Error
Simulation 1	0.5600144556	0.0000053040	0.0514624371	0.0026536865
Simulation 2	0.5033256587	0.2499995991	0.0495864245	0.2524584126