# Expected Swing Model: Logistic Regression Analysis

**Description:** Out of all the pitches thrown in Major League Baseball, a swing occurs about 47.5% of the time. Of the pitches that are swung at only 37% are put into the field of play. Then of the balls put into play only about a third of those result in hits. That means on average, only about 6% of all pitches thrown result in hits (Single, Double, Triple, Homerun). To put it another way, when a pitcher induces a swing, around 88% of the time he will be getting an advantageous outcome. That is either a foul ball, whiff or an out. With those odds, predicting how often a pitcher can get a swing on certain pitches is crucial to not only building a roster but also in game strategy. For that reason, I set out to build a model that takes certain variables in an attempt to predict where and when a pitcher will have the best chance of inducing a swing.

## Step 1: Data Collection

To gather data, I utilized the package *baseballr* in R studio to web scrape data from *BaseballSavant* and their public API with Statcast data. Due to the large amounts of data called, I broke up the API calls to a couple days and filled 6 dataframes with around 25,000 pitches in each. Then I used the *bind_rows* function from *dplyr* to combine all the individual dataframes into one for data cleaning.

```r
library(baseballr, dplyr)

# Scrape Data using baseballr
data1 <- scrape_statcast_savant(start_date = '2023-04-01', end_date = '2023-04-07', player_type = 'pitcher')
data2 <- scrape_statcast_savant(start_date = '2023-05-01', end_date = '2023-05-04', player_type = 'pitcher')
data3 <- scrape_statcast_savant(start_date = '2023-06-01', end_date = '2023-06-04', player_type = 'pitcher')
data4 <- scrape_statcast_savant(start_date = '2023-08-01', end_date = '2023-08-04', player_type = 'pitcher')
data5 <- scrape_statcast_savant(start_date = '2023-08-10', end_date = '2023-08-14', player_type = 'pitcher')
data6 <- scrape_statcast_savant(start_date = '2023-05-10', end_date = '2023-05-14', player_type = 'pitcher')

# Combine dataframes using dplyr
data <- data1 %>% bind_rows(., data2,data3,data4,data5,data6)
```

## Step 2: Data Preprocessing

As someone who has worked with baseball data regularly in Yakkertech or Trackman, I like to rename the columns to what I am familiar with. Additionally, I like to recode some of the values in the columns and split the *events* column into *PitchCalls* and *PlayResults*. Also, for the purposes of this study, a couple values are redundant in that they can both be called one outcome. In order to keep the code clean and have the ability to quickly edit column names or values I created 4 vectors renaming and recoding value to apply to the whole dataframe.

```r
# New Column Names
new_names <- c('Date' = 'game_date','RelSpeed' = 'release_speed','RelSide' = 'release_pos_x', 'RelHeight' = 'release_pos_z', 'PitcherName' = 'player_name',
               'Zone' = 'zone','BatterSide' = 'stand', 'PitcherThrows' = 'p_throws','Balls' = 'balls', 'Strikes' = 'strikes','HorzBreak' = 'pfx_x',
               'InducedVertBreak' = 'pfx_z','PlateLocSide' = 'plate_x', 'PlateLocHeight' = 'plate_z', 'On3B' = 'on_3b', 'On2B' = 'on_2b', 'On1B' = 'on_1b',
               'Outs' = 'outs_when_up', 'Inning' = 'inning', 'Top.Bottom' = 'inning_topbot','ExitSpeed' = 'launch_speed','Angle' = 'launch_angle',
               'SpinRate' = 'release_spin_rate','Extension' = 'release_extension','PitchofPA' = 'pitch_number','TaggedPitchType' = 'pitch_name',
               'SpinAxis' = 'spin_axis')

# New Pitch Call Names
Pitch_Calls <- c('swinging_strike' = 'StrikeSwinging', 'foul' = 'FoulBall', 'ball' = 'BallCalled', 'called_strike' = 'StrikeCalled',
  'hit_into_play' = 'InPlay', 'blocked_ball' = 'BallCalled', 'foul_tip' = 'FoulBall', 'swinging_strike_blocked' = 'StrikeSwinging',
  'hit_by_pitch' = 'HitByPitch', 'foul_bunt' = 'BuntFoul', 'missed_bunt' = 'BuntSwingingStrike', 'bunt_foul_tip' = 'BuntFoul')

# New Pitch Type Names
Pitch_Types <- c('4-Seam Fastball' = 'Fastball', 'Split-Finger' = 'Splitter')

# New Play Result Names
Play_Results <- c('Field_out' = 'Out', 'Grounded_into_double_play' = 'Out', 'Force_out' = 'Out','Home_run' = 'Homerun',
  'Hit_by_pitch' = 'HitByPitch', 'Strikeout_double_play' = 'Strikeout','Field_error' = 'Error', 'Sac_fly' = 'Sacrifice',
  'Sac_bunt' = 'Sacrifice', 'Fielders_choice' = 'FieldersChoice', 'Sac_fly_double_play' = 'Sacrifice',
  'Double_play' = 'Out', 'Fielders_choice_out' = 'FieldersChoice', 'Other_out' = 'Out')
```

I also added some columns of variables that I thought might be important to my model. Utilizing the *mutate* feature from *dplyr*, I will go through all of my changes and additions below.

1. The *rename* function allows me to rename all of my columns that the raw data provided.
2. *Count*: Using the function *paste0* I concatenated the Balls and Strikes column to return the count of the plate appearance.
3. *OutsOnPlay*: With no column detailing the number of outs on any given pitch, I wanted to make it easier on myself later by creating this column. Using the *case_when* function, I was able to use a conditional statements that output the number of outs on a certain pitch.
4. *PitcherThrows & BatterSide*: This was a preference; I prefer to have the handedness as Left or Right instead of L or R simply because that's more what I'm used to.
5. *ScoreDiff*: I thought this might be a helpful column that could play a role in my model, some hitters might be more reluctant to swing at a pitch if their team is down runs and they need to find a good pitch to hit.
6. *Swing*: Since the model aims to predict swings, instead of always having to group together the different swings in the *PitchCall* column, grouping them together is easier.
7. *Manon1B-Manon3B:* Again, I thought the base runner's situation may play a role in my model, so I wanted to clean those columns up since it returns the player ID instead of saying if there is a runner on or not.
8. *Zone:* The raw data does have a column that details which zone the pitch was thrown, but in the past I have used a different numbering system to detail the zones. Therefore, I created a function called "ZONECA" that returns the zones with my numbering system.

```
DF <- data %>% rename(any_of(new_names)) %>% mutate(Count = paste0(Balls, '-', Strikes)) %>%
  mutate(OutsOnPlay = case_when(events %in% c('strikeout', 'field_out', 'force_out', 'caught_stealing_home', 'sac_fly', 'caught_stealing_2b', 'sac_bunt', 'fielders_choice',
  'pickoff_1b', 'fielders_choice_out', 'other_out', 'pickoff_caught_stealing_2b', 'pickoff_3b', 'caught_stealing_3b')~1,
  events %in% c('double_play', 'sac_fly_double_play', 'strikeout_double_play', 'grounded_into_double_play')~2, T~0)) %>%
  mutate(PitcherThrows = recode(PitcherThrows, 'L' = 'Left', 'R' = 'Right'), BatterSide = recode(BatterSide, 'L' = 'Left', 'R' = 'Right'),
  TaggedPitchType = recode(TaggedPitchType, !!!Pitch_Types),PitchCall = recode(description, !!!Pitch_Calls),PlayResult = str_to_title(events),
  PlayResult = recode(PlayResult, !!!Play_Results)) %>%
  filter(., !(TaggedPitchType %in% c('', 'Other', 'Screwball', 'Pitch Out')) & PitchCall != 'pitchout' & RelSpeed > 0) %>%
  mutate(ScoreDiff = case_when(Top.Bottom == 'Top'~away_score-home_score, T~home_score-away_score)%>%
  mutate(Swing = case_when(PitchCall %in% c('BuntFoul', 'BuntSwingingStrike', 'StrikeSwinging', 'FoulBall', 'InPlay')~'Swing', T~'Take'))
  mutate(ManOn1 = case_when(is.na(On1B) == TRUE~'No', T~'Yes'), ManOn2 = case_when(is.na(On2B) == TRUE~'No', T~'Yes'),
  ManOn3 = case_when(is.na(On3B) == TRUE~'No', T~'Yes')) %>%
  mutate(Zone = ZONECA(PlateLocSide, PlateLocHeight))
```

**Step 3: Data Visualization**

To get a sense of my data selection, I decided to use various graphs through *ggplot2* to get a quick glance at the contents. All the code for creating the graphs can be found in the GitHub repository, as well as the PDF of the visuals. Below is a quick summary of the data:

- 25% Left-Handed Pitchers and 75% Right-Handed Pitchers
- 1/3 Left-Handed Batters and 2/3 Right-Handed Batters
- LHP and RHP had basically the same swing rates
- Minimal difference in swing rates dependent on outs
- Variability in swing rates were present when grouped by counts.
- To be expected, swing rates declined on pitches outside of the strike zone.

**Step 4: Model Selection**

For this project, I decided to utilize a logistic regression model. Since what I was looking for was a probability of a binary output, logistic regression served my purposes well. The output of my model is to give a probability that any given batter will swing at a pitch with the respective variables.

**Step 4.1: Model 1**

To start, I used my knowledge of baseball as well as the patterns I detected in my graphs to begin choosing which variables to include in my model. Based on the graphs, I knew I had to include Balls and Strikes because of the variability seen in the bar graph. Additionally, with the strike zone plot, I can tell Plate Location will be valuable, so I included PlateLocHeight and PlateLocSide. Then, I figured the speed of the pitch and movement of the pitch will have an impact on whether or not the batter swings. Lastly, I wanted to include the number of outs in my model even though the graphs show minimal variation in swing rates because that is a prominent game situation hitters consider.

Since the output gives the predictions as a probability and considering this was the first attempt, I made the threshold for a swing or not at 50%.

```
model_data <- DFS %>% select(PlateLocHeight,PlateLocSide,RelSpeed,InducedVertBreak,HorzBreak,Balls,Strikes,Outs,Swing)

set.seed(123)

train_index <- createDataPartition(model_data$Swing, p = 0.8, list = FALSE)
train_data <- model_data[train_index, ]
test_data <- model_data[-train_index, ]

model <- glm(Swing ~ ., data = train_data, family = 'binomial')
predictions <- predict(model, newdata = test_data, type = "response")
binary_predictions <- ifelse(predictions > 0.5, 1, 0)

confusion_matrix <- table(test_data$Swing, binary_predictions)
accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix)
precision <- confusion_matrix[2, 2] / sum(confusion_matrix[, 2])
recall <- confusion_matrix[2, 2] / sum(confusion_matrix[2, ])
f1_score <- 2 * precision * recall / (precision + recall)
auc <- roc(test_data$Swing, as.numeric(predictions))$auc
```

Below are the metrics I used to determine the effectiveness of my model.

| Model 1 Eval. | |
|---|---|
| **Metric** | **Value** |
| Accuracy | 0.601 |
| Precision | 0.589 |
| Recall | 0.508 |
| F1-Score | 0.545 |
| AUC | 0.640 |

With an accuracy score of .601 and an AUC of .640, I was pleased to see my first attempt at the model was better at predicting a swing than randomly guessing. Out of all the positive predictions, around 59% turned out to be actually positive. Then out of all the actual positive instances, the model correctly identified 51% of them as positive.

With higher precision, it is good to see the model make relatively few false positive predictions. However, the low recall suggests there is a significant portion of false negatives. For the purposes of this model, I would prefer to have a higher precision because the value a swing has as discussed in the overview. Swings typically lead to a positive outcome for a pitcher, therefore a false positive could lead me to classify a pitch as swing inducing when in reality it isn't.

The F1-Score of .545 suggests a pretty good balance between Precision and Recall, however, there seems to be room for improvement.

Now that I have my first model, I want to optimize my threshold of considering the probability of a swing or not. To do that I used two different strategies and compared them.

*Method 1. Calculate F1-Score for each threshold.*

To do this, I created a function called *calculate_f1_score* which we'll use when we run every threshold possible (0-1). Then I will combine all the F1-Scores with each threshold and find which threshold output the highest F1-Score. Below is a snippet of the code I used.

```
calculate_f1_score <- function(tp, fp, fn) {
  precision <- tp / (tp + fp)
  recall <- tp / (tp + fn)
  f1_score <- 2 * precision * recall / (precision + recall)
  return(f1_score)
}
f1_scores <- data.frame(threshold = numeric(length = 101), f1 = numeric(length = 101))

# Calculate F1 scores for each threshold
for (i in 1:101) {
  threshold <- (i - 1) / 100
  binary_predictions <- ifelse(predictions > threshold, 1, 0)
  confusion_matrix <- table(test_data$Swing, binary_predictions)

  # Ensure that confusion matrix has all needed values
  if (ncol(confusion_matrix) < 2 || nrow(confusion_matrix) < 2) {
    # Not enough values in the confusion matrix, skip
    next
  }

  tp <- confusion_matrix[2, 2]
  fp <- confusion_matrix[1, 2]
  fn <- confusion_matrix[2, 1]

  f1_scores[i, "threshold"] <- threshold
  f1_scores[i, "f1"] <- calculate_f1_score(tp, fp, fn)
}

# Find the threshold that maximizes the F1 score
optimal_threshold <- f1_scores$threshold[which.max(f1_scores$f1)]
```

This method told me that the optimal threshold for my binary predictions is .33. So, I re-ran the binary predictions with a threshold of .33 and below is the evaluation for my model.

| Model 1 Eval. | |
|---|---|
| **Metric** | **Value** |
| Accuracy | 0.525 |
| Precision | 0.498 |
| Recall | 0.944 |
| F1-Score | 0.652 |
| AUC | 0.640 |

While the F1-Score has increased, the Precision declined by .10. By using a threshold of .33, it seems to be leading to a much higher number of false positives which is not helpful for my study.

*Method 2: Youden's J Statistic*

Here I am again iterating over every possible threshold but calculating the Youden J Statistic in hopes that this threshold will prove to serve my purposes better.

```
# Threshold Determination Attempt 2
thresholds <- seq(0, 1, by = 0.01)
metrics <- data.frame(threshold = thresholds, sensitivity = numeric(length(thresholds)), specificity = numeric(length(thresholds)), youden_j = numeric(length(thresholds)))

# Calculate sensitivity, specificity, and Youden's J for each threshold
for (i in 1:length(thresholds)) {
  threshold <- thresholds[i]
  binary_predictions <- ifelse(predictions > threshold, 1, 0)
  confusion_matrix <- table(test_data$Swing, binary_predictions)
  # Ensure that confusion matrix has all needed values
  if (ncol(confusion_matrix) < 2 || nrow(confusion_matrix) < 2) {
    # Not enough values in the confusion matrix, skip
    next
  }
  # Calculate sensitivity and specificity
  sensitivity <- confusion_matrix[2, 2] / sum(confusion_matrix[2, ])
  specificity <- confusion_matrix[1, 1] / sum(confusion_matrix[1, ])

  # Calculate Youden's J statistic
  youden_j <- sensitivity + specificity - 1

  # Store metrics in the data frame
  metrics[i, c("sensitivity", "specificity", "youden_j")] <- c(sensitivity, specificity, youden_j)
}

# Find the threshold that maximizes Youden's J statistic
optimal_threshold <- metrics$threshold[which.max(metrics$youden_j)]
```

This method told me that the optimal threshold would be .44. So below is an evaluation of the model using .44 as my threshold.

| Model 1 Eval. | |
|---|---|
| **Metric** | **Value** |
| Accuracy | 0.607 |
| Precision | 0.571 |
| Recall | 0.670 |
| F1-Score | 0.616 |
| AUC | 0.640 |

With slightly higher accuracy and only minimally sacrificing the precision, I feel .44 is a better threshold than .33 and provides a better balance between precision and recall than .5 does.

Finally, I want to look at a summary of the model to determine which variables were significant and if there is any room for improvement in my variable selection.

```
Call:
glm(formula = Swing ~ ., family = "binomial", data = train_data)

Deviance Residuals:
   Min      1Q  Median      3Q     Max
-1.827  -1.063  -0.846   1.152   1.659

Coefficients:
                 Estimate Std. Error z value Pr(>|z|)
(Intercept)      -1.276039   0.150606  -8.473  < 2e-16 ***
PlateLocHeight    0.130609   0.008043  16.239  < 2e-16 ***
PlateLocSide     -0.065778   0.008975  -7.329 2.32e-13 ***
RelSpeed          0.003447   0.001760   1.959   0.0501 .
InducedVertBreak -0.002847   0.014825  -0.192   0.8477
HorzBreak         0.016115   0.008558   1.883   0.0597 .
Balls             0.171240   0.007898  21.681  < 2e-16 ***
Strikes           0.480025   0.009295  51.644  < 2e-16 ***
Outs             -0.007202   0.008632  -0.834   0.4041
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 118628  on 85715  degrees of freedom
Residual deviance: 113541  on 85707  degrees of freedom
AIC: 113559

Number of Fisher Scoring iterations: 4
```

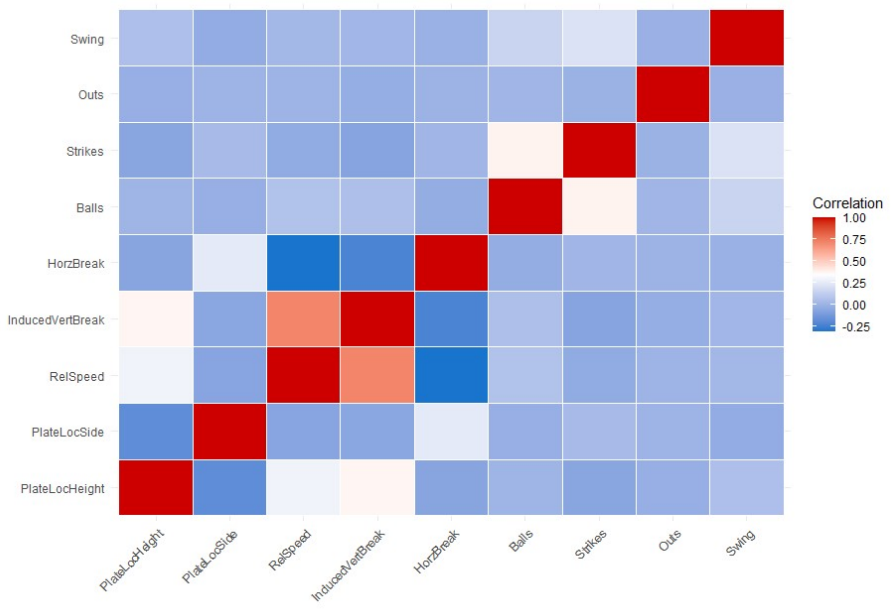
As I had expected through my visualizations, Balls and Strikes had high significance, while Outs did not. To my surprise, Release Speed, Induced Vertical Break, and Horizontal Break did not play a major role in my model. While I know release speed has to have an effect, I wonder if these metrics have any collinearity with PlateLocHeight and PlateLocSide (the location of the pitch).

For that reason, before I proceed with model specification, I wanted to run a couple tests on collinearity in the data. Below were the strategies I used to test for multicollinearity:

```
355 cor_matrix <- cor(model_data)
356
357 vif_model <- lm(Swing~., data = model_data)
358 vif_results <- vif(vif_model)
359
360 eigen_values <- eigen(cor_matrix)$values
361
362 tolerance <- 1 / vif_results
363
364 corrplot(cor_matrix, method = 'circle')
```

The correlation matrix with the help of the correlation plot showed that there was a relatively high relationship between Release Speed and Induced Vertical break. Also, when using the tolerance method, mostly all other variables weren't correlated but release speed and induced vertical break both had a tolerance of .451 and .443 respectively. For these reasons, the high p-value and based on my knowledge of baseball, I chose to remove Induced Vertical Break from the model and keep release speed. Also, with little significance as seen in the summary I will be removing Outs from the model.

Here is the output for the correlation matrix I described above:



**Step 4.2: Model 2**

After removing Induced Vertical Break and Outs as predictor variables from the model, I saw very little difference in terms of all my performance metrics I have been using. Below is a table from the second model displaying the performance metrics.

| Model 2 Eval. | |
|---|---|
| **Metric** | **Value** |
| Accuracy | 0.606 |
| Precision | 0.570 |
| Recall | 0.669 |
| F1-Score | 0.616 |
| AUC | 0.640 |

Everything about the second model was the same (threshold and logistic regression) and since I did not see any noticeable improvements, for my third attempt I decided to add another predictor variable.

**Step 4.3: Model 3**

For the third model I decided to incorporate the run expectancy matrix as a predictor variable. The run expectancy matrix attempts to place run values on each at-bat depending on game state situations. The matrix gives an average of how many runs are scored by the end of an inning for any given situation. For example, when there are no runners on, and 2 outs teams can expect to score .095 runs by the end of the inning.

Now, expecting batters to go through the run expectancy matrix each time they are at-bat and determine if they want to swing is unrealistic. However, game state situations do have an effect on hitters and by using the run expectancy numbers we can place more importance on certain situations. For example, if there are runners on first and second with 1 out, the team can expect to score .908 runs. Players might be more reluctant to swing at any given pitch because they want to score a run and if they get out their run expectancy goes down to .343 runs. Now since there are 2 outs players might feel more inclined to take a risk and swing rather than sit back and wait for their favorite pitch.

To incorporate the RE Matrix, I first created a new column in my main dataframe called *Runners*. By concatenating the *On1B, On2B,* and *On3B* columns together. Then separately I created a new

dataframe called *REMATRIX* with a *Runners* column and columns called *Zero, One* and *Two.* In those 3 respective columns I entered the expected runs based on the game state situation. After that, with some data manipulation, I was able to produce a dataframe that will seamlessly join the main dataframe. Below is the code I wrote to perform this task.

```
REMATRIX <- data.frame(Runners = c('No_No_No','Yes_No_No','No_Yes_No','Yes_Yes_No','No_No_Yes','Yes_No_Yes','No_Yes_Yes','Yes_Yes_Yes' ),
  Zero = c(.461,.831,1.068,1.373,1.426,1.798,1.920,2.282),
  One = c(.243,.489,.644,.908,.865,1.140,1.352,1.520),
  Two = c(.095,.214,.305,.343,.413,.471,.570,.736)) %>% reshape2::melt(., id.vars = c('Runners')) %>%
  mutate(variable = recode(variable, 'Zero' = 0, 'One' = 1, 'Two' = 2)) %>%
  mutate(Situation = paste0(Runners,'_', variable)) %>% select(Situation, RE = value)

DFS <- DF %>% mutate(Swing = case_when(PitchCall %in% c('BuntFoul', 'BuntSwingingStrike', 'StrikeSwinging', 'FoulBall', 'InPlay')~1, T~0)) %>%
  mutate(Situation = paste0(ManOn1, '_', ManOn2, '_', ManOn3, '_', Outs)) %>% left_join(., REMATRIX, by = c('Situation'))

```

After running the model with the newly created variable, *RE* I was surprised to see there was not much of a difference. Below are the performance metrics of the third model.

| Model 3 Eval. | |
|---|---|
| **Metric** | **Value** |
| Accuracy | 0.604 |
| Precision | 0.569 |
| Recall | 0.667 |
| F1-Score | 0.614 |
| AUC | 0.641 |

Even though the run expectancy column proved to be an important variable with a p-value close to zero, the all the performance metrics are approximately the same as the other models. After going through the summaries for each model, I decided to move forward with the 1st model as the accuracy was the highest, but also a high precision score. I talked about this early that having a higher precision score was important for this model since a swing tends to lead to better outcomes and therefore I want the risk of predicting a swing when it wasn't an actual swing to be low.

**Step 5: Visualization and Analysis**

As the final part of this study, I will bring the model to life by visualizing different aspects through the use of *ggplot2* in R studio. Working as a data analyst in baseball, I have several different strike zone and heat map templates that I use to display pitch locations. For this study I will be using data from 2023 on three individual pitchers to see whether expected swing rates can be a good predictor of success.

**Step 5.1: Pitcher Selection**

In order to properly value a pitcher, one must look at numerous statistics across all facets of their game. However, for this study I will be selecting 3 pitchers with varying WARs (wins above replacement) and run their pitches through my model.

Gerrit Cole – WAR: 5.2
Dylan Cease – WAR: 3.7
Dean Kremer – WAR: 1.5

*Also, as a disclaimer, these 3 pitcher's data from 2023 was not used in the training of the swing model.*

**Step 5.2: Pitcher Analysis**

I ran 3 different calculations on each of the pitchers for the first table below. The first was using the binary predictions with a threshold of .44 for if it was a swing or not. The second was I took the predicted probabilities from the logistic regression model and took the average of those probabilities. Then, the third I calculated the actual swing rate of each of the pitchers.

| | PitcherName | ExpectedSwingRate_Binary | SwingProbability | ActualSwingRate |
|---|---|---|---|---|
| 1 | Cease, Dylan | 56.0% | 47.5% | 46.2% |
| 2 | Cole, Gerrit | 59.3% | 48.6% | 50.0% |
| 3 | Kremer, Dean | 56.8% | 48.2% | 47.5% |

The expected binary swing rates actually produced higher rates than the actual swing rate which as I explained earlier is not helpful for this study. The swing probabilities actually produced expected swing rates fairly close to the actual rates. Overall, this output is very good confirmation that this model could be useful.

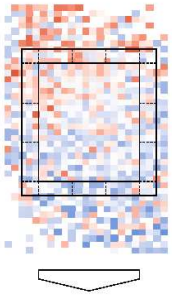
My takeaway from the expected swing rate binary being higher than the actual swing rate is that many of the pitches should have been swung at but hitters were able to restrain from swinging. Additionally, both the binary and probabilities all ranked the pitchers correctly in who had the highest swing rate compared to their actual swing rate.

Below is a table looking at the same calculations but on each of the pitcher's individual pitch types. One caveat for this model is that it appears to be consistently predicting pitchers to have higher swing rates than what actually occurred. Again, the Swing Probability is still relatively close to the actual swing rates.

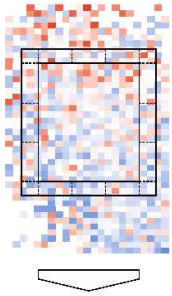
| Expected Swing Rates | | | | |
|---|---|---|---|---|
| **Pitcher** | **Pitch** | **xSwing Binary** | **xSwing Prob.** | **Actual Swing** |
| **Cease, Dylan** | Changeup | 49.5% | 45.8% | 26.3% |
| | Fastball | 58.2% | 49.3% | 48.1% |
| | Knuckle Curve | 42.4% | 43.4% | 37.2% |
| | Slider | 59.3% | 47.3% | 49.3% |
| **Cole, Gerrit** | Changeup | 58.8% | 47.5% | 48.1% |
| | Cutter | 55.2% | 46.8% | 56.5% |
| | Fastball | 62.3% | 50.3% | 51.2% |
| | Knuckle Curve | 53.0% | 45.8% | 42.4% |
| | Slider | 57.1% | 46.9% | 49.8% |
| **Kremer, Dean** | Changeup | 57.0% | 47.7% | 42.2% |
| | Curveball | 41.4% | 42.7% | 37.9% |
| | Cutter | 54.3% | 47.3% | 47.9% |
| | Fastball | 66.1% | 51.5% | 53.1% |
| | Sinker | 47.4% | 45.4% | 43.2% |
| | Sweeper | 54.2% | 46.3% | 45.8% |

The last visual I created were strike zones displaying the averages of the swing probabilities. Since the swing probabilities were close to the actual swing rates, I thought it would be helpful to display them with the pitch locations. The results of this test were interesting as many of the higher expected swing rates were up in the zone. My takeaway from the strike zones is that pitchers can expect hitters to chase more pitches up in the zone rather than any other parts of the zone. So, if a pitcher were to miss, it would be best to miss their location high rather than anywhere else.

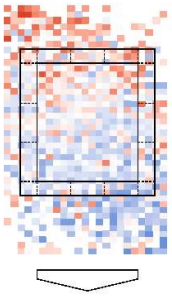
Gerrit Cole Swing Probability



Dylan Cease Swing Probability



Dean Kremer Swing Probability



**Step 6: Conclusion**

While the results of the model in terms of accuracy were not as high as I would have expected, the overall generalizations can still be helpful. The model predicted that hitters were more likely to swing than what actually occurred, which is telling. In my opinion, this means that pitchers have the upper hand, and hitters are more prone to swing than what actually occurs. Additionally, we learned that pitches up in the zone are more likely to be swung at than anywhere else outside of the strike zone. In conclusion, there were many good takeaways from this study, and I believe there are definitely spots that could be improved upon in a different study with other techniques.