Single or Out?

James Musso

1/10/2020

Overview

Introduction

The game of baseball lends itself well to statistical analysis. Since its inception in the late 19th century, numerous observers have used statistics to better understand what contributes to a winning effort. Many but not all of the factors or features are apparent from mere observation. Statistical analysis, particularly within the last forty years, has unveiled some important contributors to winning that are not obvious to the naked eye, and are sometimes even counterintuitive.

At its core, baseball is about scoring more runs than the other team. A run is scored when a batter is able to reach first base, then second base, then third base, and finally home plate. Thus, the primary objective of a baseball team's offense is to first have a batter reach base and then eventually score a run by crossing home plate before his team makes three outs in the inning. Many events can cause a batter to reach home. One such event, the single, is the subject of this study.

A single occurs when: 1) a batter bats a ball into fair territory, 2) none of the nine players on the field (fielders) are able to catch the ball before it hits the ground, 3) none of the fielders are able to tag the batter with the ball before the batter contacts first base, and 4) none of the fielders are able to contact first base with the ball in their possession before the batter reaches first base.

A single is just one type of hit. Batted balls resulting in hits are the primary way that batters reach base. For players and fans alike, hits are among the most entertaining events in a baseball game. But singles tend to be less exciting than other types (doubles, triples, and home runs). A single only allows a batter to reach first base, so the batter, now a baserunner, is still a long way from home. Singles often aren't hit as hard and don't travel as far as other types of hits. But they are the most common way to reach base, and so they play an important role in scoring runs.

During the 2019 Major League Baseball (MLB) regular season, batters stepped to the plate with an opportunity to bat the ball about 186,000 times (https://baseballsavant.mlb.com/league). On about 126,000 of those occasions, they succeeded in hitting the ball somewhere onto the field. About 26,000 of those batted balls were singles, allowing the batter to reach first base. All the other types of hits combined amounted to only 16,000. That means about 84,000 batted balls, or two-thirds, resulted in outs. Baseball is a game of failure; most batters never make it back home.

What is it about the ubiquitous single that deserves the attention of a study? Perhaps more than any other batting event, a batted ball resulting in a single seems to be the event that's most difficult to distinguish from a field out (an out made by a fielder on a batter ball). Like field outs, singles come in a variety of shapes and sizes. Both a weakly hit ground ball tapped a few feet in front of home plate, and a screaming line drive hit off of the top of an outfield wall have been singles, and it's not that unusual to see either of these dramatically different examples.

So what is it that makes a batted ball a single rather than an out? Is it merely a matter of hitting the ball where there happens to be no fielder? Is it merely a matter of chance? Or are there identifiable features that distinguish a single from a field out, and if so, can we incorporate those features into a statistical model that reliably predicts when a batted ball will result in a single rather than an out? The goal of this effort is to answer those questions.

One point of clarification. We are not determining the probability of a particular batter hitting a single in a particular game situation. If that was the case, we would be considering features such as the past

performance of the batter, pitcher, and fielders, the altitude, dimensions and configuration of the park, the temperature and humidity, the wind direction and velocity, cloud cover and the position of the sun, the number of pitches the pitcher has already thrown in the game, the number of baserunners and outs, the inning, and so on and so on and so on. Instead, this study focuses on features the batter can conceivably control, namely, the direction he hits the ball, how hard he hits the ball, and how fast he runs to first base, as these are the most fundamental components of a single.

The Data

In 2015, MLB installed a state-of-the-art tracking technology, Statcast, in all 30 parks. It consists of a Trackman Doppler radar and high-definition Chyron Hego cameras. The radar, positioned above and behind home plate, captures just about everything that happens to the ball at 20,000 frames per second. The six stereoscopic cameras, installed down each foul line, track the movement of players on the field. In short, the technology generates data allowing the skills of players to be quantified in new ways.

Baseball Savant is MLB's public-facing repository for Statcast data. It includes, among other things, a powerful search tool that allows users to create their own custom queries. The datasets used in this study were created using Baseball Savant's various applications and search tools at BaseballSavant.MLB.com.

The primary dataset (*battedballs_2019_Raw*, or *battedballs* for short) obtained from Baseball Savant originally included 108,881 batted balls hit during the 2019 MLB season that resulted in either a single or an out (or an error, which, by definition, should have been an out). Excluded were batted balls that resulted in other types of hits (doubles, triples, and home runs), and batted balls officially ruled to be sacrifice bunts. Sacrifice bunts were excluded because, by definition, the batter is only attempting to advance a baserunner, not get a hit.

The key features or predictors in our *battedballs* dataset include Statcast measurements of how the ball comes off the bat in terms of direction and velocity. With regard to direction, a batted ball travels through three-dimensional space, but for our purposes, we're really only interested in two of those dimensions: vertical (up-down) and horizontal (left-right). The third dimension, depth, is called Hit Distance by Statcast, and it's really telling us where the ball ended up, which is where the ball landed or was touched by a fielder. And knowing where the ball landed or was touched by a fielder comes very close to directly telling us the answer to our question of whether the batted ball will be a single or an out. In other words, we're more interested in telling a batter how he has to hit the ball (the process) rather than where his batted ball needs to land on the field (the result), something that hopefully is already obvious to him.

Statcast's measure of a batted ball's vertical direction is called **Launch Angle** (*launch_angle* in the *battedballs* dataset), which represents the vertical angle at which the ball leaves the bat after being struck. A batted ball with a launch angle of 10 degrees or less is probably a ground ball, while a launch angle greater than 39 degrees is probably a pop up. But our primary purpose in using Launch Angle is not to deduce whether a batted ball is a ground ball, line drive, fly ball, or pop up. We're using it to help predict whether a batted ball will be a single or an out.

At present, Statcast surprisingly does not directly measure the horizontal direction of a batted ball (or at least doesn't publish such measurements through Baseball Savant). But it does provide a batted ball's final location coordinates (referred to as *hc_x* and *hc_y*), which reveal horizontal direction as well as distance. Like Hit Distance, *hc_x* and *hc_y* tell us where the ball ended up, so including them in our model would introduce bias. But we can use them to compute, with a little trigonometry, a very good estimate of horizontal location (commonly referred to as **Spray Angle**). These calculations will accordingly appear in the *battedballs* dataset under the variable *spray_angle*.

Another feature, this one categorical, was added to our mix in the hope it would enhance the significance of *spray_angle*. Baseball Savant provides, for each batted ball, information regarding the locations of the fielders at the time the ball was batted into play. By itself, this information can tell us little if anything about the likelihood of a batted ball resulting in a single or an out. Plus, it violates our guideline about using variables the batter can conceivably control. However, its interaction with *spray_angle* might improve

a model's ability to make such single-out distinctions for predictive purposes. So we opted to include this information in our models for the sole purpose of making *spray_angle* more meaningful.

For infielders, the variable is named *if_fielding_alignment*, and its possible values represent three somewhat broad categories intended to describe where the infielders are located with respect to second base. According to MLB's Glossary (http://m.mlb.com/glossary/statcast/shifts), a value of "Standard" indicates all four infielders are standing in their traditional spots; that is, "where, under neutral conditions (first to eighth inning, no runners on), the league average fielder was positioned 70 percent to 90 percent of the time." In mathematical terms, MLB defines these "traditional spots" based on spray angle and distance from home plate.

A value of "Infield shift" indicates at least three of the four infielders are located on one side (generally, the batter's pull side) of second base.

Lastly, a value of "Strategic" encompasses fielding alignments that are neither "Standard" nor "Infield shift". According to MLB's Glossary, an example would be a single infielder playing out of position, "like a second baseman being shifted to short right field... or a shortstop moving very close to... second base... but not quite moving to the other side of it."

Of course, knowing how the infielders are aligned can only be useful for batted balls with relatively low launch angles (Low Ground Balls and High Ground balls). Still, such batted balls comprise more than more than half of all the batted balls in our data, and for nearly one-third of those batted balls, infielders were in a non-standard alignment. So we thought it worthwhile to investigate whether the inclusion of *if_fielding_alignment* could improve our models.

Statcast's measurement of how hard the ball was hit is called **Exit Velocity** (*launch_speed* in the *battedballs* dataset). Specifically, it measures the speed of the ball (in miles per hour) immediately after the batter makes contact. It's potential relevance to our "single or out" question is apparent: a faster moving ball gives the fielders less time to react and catch the ball or stop the ball, making it less likely they will get the batter out.

A second dataset (*hometofirst_2019_Raw*, or *hometofirst* for short) from Baseball Savant includes Statcast's **Home to First** measurement. It's the length of time, in seconds, from the moment the bat hits the ball to the moment the batter touches first base. Basically, it's telling us how fast the batter can run to first base, and it's another one of our fundamental components of a single. However, it's only relevant for our study if the batted ball is a low ground ball (launch angle < 0 degrees). The batter's ability to reach first base quickly is far less likely to matter if the batted ball is a line drive, fly ball, or pop up. Still, low ground balls comprise about 37% of all the batted balls in our dataset, so the batter's ability to reach first base quickly is potentially a helpful predictor in our model.

Baseball Savant makes Home to First data available to us in the form of a season average rather than for each individual batted ball. And the season average wisely considers only batted balls that it characterizes as Topped or Weakly Hit (based on Launch Angle and Exit Velocity) because these plays are likely to require the batter to make a serious effort to run as fast as possible. Moreover, the average considers only the fastest 70 percent of these serious effort plays.

The Process

We followed a typical path toward the development of predictive models. The initial task, referred to as data wrangling, is to get the data into a form that can be used to develop algorithms. This involved importing the data files into R data frames, removing unneeded observations with missing data, removing irrelevant observations, combining data frames, adding new variables and computing their values from existing variables, converting categorical variables into numeric form, converting continuous variables into new categorical variables, and reordering the columns in the final data frame. All of these steps would facilitate our later analysis.

The next key task was to explore the data to deepen our understanding of which variables might be most helpful in a predictive model, and how they related to each other and to the outcome we were trying to

predict. Both quantitative and visual tools were used for this purpose. In the context of this project, plots of batted balls would help reveal insights, as would various types of boxplots and density plots.

Density plots proved particularly helpful in clarifying relationships between variables, showing the distribution of data values as a continuous line while the Y axis shows proportions or frequency. The message they convey is not muddied by the disparate number of observations in the groups portrayed because it shows proportions rather than counts. This was important in our case because the classes (singles and outs) of our outcome variable had significantly different numbers of observations (outs occur much more frequently than singles).

At this point, we were almost ready to begin building models, but a couple of important data preparation tasks remained to be completed. The first involved rescaling the data to meet the particular needs of each type of algorithm we anticipated testing. Because we expected to be working with a variety of algorithms, we opted to create transformed versions of our key predictor variables using three methods: centering, standardization, and normalization. These methods are discussed in some detail later in this report.

The last task in this step involved randomly partitioning our final data frame into a train set and a test set. This allowed us to use one portion of our data solely to fit an algorithm, and a completely distinct and independent portion of our data to test the predictive ability of the newly fit algorithm. This, in turn, diminishes the problem of overfitting or overtraining the model to perform very well with out data, and not nearly as well once it's used in the real world with different data.

Finally, we could begin constructing models and testing them. We initially built a simple model based on a guessing algorithm. This would serve as a baseline for comparison once we began building other models. If our later models couldn't outperform the baseline model, we would have a serious problem. We next constructed a logistic regression model. Logistic regression is essentially a form of linear regression that has been adapted for predicting a binary outcome from a set of continuous and/or categorical predictor variables, which was exactly what we were trying to do.

A K-nearest-neighbors model was built next, followed by a classification tree model and finally, a random forest model. Building each of these models involved training or fitting the model to our train set, and then testing the model by allowing it to estimate the probabilities of each outcome based on the data in the test set. The model would then make its final prediction based on the probabilities. At that point we became like a teacher grading an exam: how many predictions did the model get right?

During this model-building stage, an important decision had to be made about the criteria we would apply when assessing each model's predictions. We eventually decided to use two somewhat different methods, neither of which appears to be in mainstream use at this time, but which appeared to be well-suited for our particular challenge. The first one we labeled Maximum Truescore, which is based on maximizing the harmonic mean of sensitivity and specificity. The second method, Minimum Distance, seeks to minimize the distance between the ROC curve and the coordinate (0, 1), which is where both sensitivity and specificity will be very high.

The results of both methods can be computed using only sensitivity and specificity. The two methods were published in the same paper, the specific purpose of which was to develop better methods to address the "decision threshold shifting issue" that occurs when attempting to classify imbalanced data. [1] That was our challenge: too many outs and not enough singles would make our classification models perform less well because they cause the threshold that best differentiates the outcomes (singles and outs) to shift in favor of the majority class (outs). This is discussed in some detail later in this paper.

We ended our study by examining the prediction mistakes made by our best predictive model. It was hoped such a review might reveal ways of improving our model's performance and suggest further areas for research.

[1] Song, B., Zhang, G., Zhu, W. et al. ROC operating point selection for classification of imbalanced data with application to computer-aided polyp detection in CT colonography. Int J CARS 9, 79–89 (2014) doi:10.1007/s11548-013-0913-8 The authors' manuscript may be viewed at no cost at https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3835757/#idm140500764776720title

Methodology and Analysis

Wrangling the Data

At the outset, the required R packages were installed and loaded.

```
# Install and Load Required Packages

if(!require(tidyverse)) install.packages("tidyverse",
                                         repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret",
                                         repos = "http://cran.us.r-project.org")
if(!require(e1071)) install.packages("e1071",
                                       repos = "http://cran.us.r-project.org")
if(!require(ROCR)) install.packages("ROCR",
                                       repos = "http://cran.us.r-project.org")
if(!require(interplot)) install.packages("interplot",
                                     repos = "http://cran.us.r-project.org")
if(!require(rpart)) install.packages("rpart",
                                         repos = "http://cran.us.r-project.org")
if(!require(rpart.plot)) install.packages("rpart.plot",
                                       repos = "http://cran.us.r-project.org")
if(!require(randomForest)) install.packages("randomForest",
                                       repos = "http://cran.us.r-project.org")
if(!require(gridExtra)) install.packages("gridExtra",
                                            repos = "http://cran.us.r-project.org")
if(!require(gtable)) install.packages("gtable",
                                         repos = "http://cran.us.r-project.org")
if(!require(lemon)) install.packages("lemon",
                                        repos = "http://cran.us.r-project.org")
if(!require(knitr)) install.packages("knitr",
                                       repos = "http://cran.us.r-project.org")
if(!require(markdown)) install.packages("markdown",
                                          repos = "http://cran.us.r-project.org")
if(!require(rmarkdown)) install.packages("rmarkdown",
                                         repos = "http://cran.us.r-project.org")
if(!require(tinytex)) install.packages("tinytex",
                                      repos = "http://cran.us.r-project.org")
```

To reshape our data into a useful form for modeling, we first imported our two data files from the internet (https://github.com/jfmusso/HarvardX/) directly into R data frames known as tibbles.

```
# Data is contained in two files: battedballs_2019_Raw.csv and hometofirst_2019_Raw.csv
# located at https://github.com/jfmusso/HarvardX/

# Import the data files into R data frames.

battedballs <-
  read_csv(
    "https://raw.githubusercontent.com/jfmusso/HarvardX/master/battedballs_2019_Raw.csv",
           col_types = cols(
             game_date = col_date(format = "%m/%d/%Y"),
             batter = col_character(),
             player_name = col_character(),
```

```
            stand = col_character(),
            bb_type = col_factor(levels = c("ground_ball", "line_drive", "fly_ball",
                                            "popup")),
            events = col_factor(),
            des = col_character(),
            launch_speed = col_double(),
            launch_angle = col_double(),
            hc_x = col_double(),
            hc_y = col_double(),
            hit_location = col_factor(),
            if_fielding_alignment = col_factor(levels = c("Standard", "Strategic",
                                                          "Infield shift")),
            of_fielding_alignment = col_factor(),
            hit_distance_sc = col_integer(),
            launch_speed_angle = col_factor(),
            game_pk = col_character(),
            game_year = col_character(),
            game_type = col_character(),
            home_team = col_character(),
            away_team = col_character(),
            pitcher = col_factor(),
            p_throws = col_character(),
            description = col_character(),
            type = col_character()
          )

  )

class(battedballs[])
```

```
## [1] "tbl_df"     "tbl"        "data.frame"
```

```
hometofirst <-
  read_csv(
    "https://raw.githubusercontent.com/jfmusso/HarvardX/master/hometofirst_2019_Raw.csv",
           col_types = cols(
             last_name = col_character(),
             first_name = col_character(),
             player_id = col_character(),
             team_id = col_character(),
             team = col_character(),
             position = col_character(),
             age = col_number(),
             competitive_runs = col_integer(),
             hp_to_1b = col_double(),
             sprint_speed = col_double()
           )

  )

class(hometofirst[])
```

```
## [1] "tbl_df"     "tbl"        "data.frame"
```

We then filtered the *firsttohome* data frame to remove 58 players whose First-to-Home readings were missing (presumably because they didn't have any "serious effort" runs to first base). The *battedballs* data frame was then filtered to remove batted balls events with missing values for any of our important predictors. This resulted in removing 889 batted balls due to missing coordinate and fielder alignment data. The *battedballs* data frame now stood at 107,992 observations.

```
# Remove players from *hometofirst* data frame if they have missing values for hp_to_1b.

hometofirst <- filter(hometofirst, hp_to_1b != "NA")

# Remove batted balls from battedballs data frame if they have missing values
# for any of the following variables: hc_x, hc_y, if_fielding_alignment, or
# of_fielding_alignment.

battedballs <- filter(battedballs, hc_x != "NA", hc_y != "NA",
                      if_fielding_alignment != "NA",
                      of_fielding_alignment != "NA")
```

We then combined our *battedballs* and *hometofirst* data frames, adding each batter's Home to First average to each of his batted ball events (rows) in the *battedballs* data frame. Now, we had one dataframe, *battedballs*, containing all of our potentially relevant data in the form of 103,810 batted ball events (observations). So in the process of combining our two data frames, we lost 4,182 batted balls hit primarily by pitchers for whom there was insufficient Home to First data (pitchers don't bat nearly as often as position players).

```
# Combine battedballs and hometofirst data frames.

hometofirst <- rename(hometofirst, batter = player_id)
hometofirst <- hometofirst %>% dplyr::select(batter, team, position, age, hp_to_1b)
battedballs <- left_join(battedballs, hometofirst, by = "batter")

# Remove players from battedballs for whom there is no hp-to-1b data.

battedballs <- filter(battedballs, hp_to_1b != "NA")
```

The next step was to remove the batted balls that were hit into foul territory based on the description of the play. We were interested in distinguishing singles from outs on balls hit into fair territory. The foul balls in our data frame resulted only in outs because, by the rules of the game, fouls can't result in a hit of any kind, including singles. Excluding fouls reduced our data frame by 3226 to 100,584 batted balls.

```
# Remove batted balls hit into foul territory.

battedballs <- battedballs %>% mutate(foul = str_detect(des, "foul")) %>%
  filter(foul == FALSE)
```

It was now time to compute one of our key predictors, *spray_angle*, and add it to the *battedballs* data frame. This required using a formula to first transform Baseball Savant's coordinate data so that home plate would be located at coordinates (0, 0). [2]

[2]The formula used in this study was published in a Western Michigan University honors thesis by Tess Kolp (Kolp, Tess, "Home Run Probability Based on Hit Distance and Direction" (2018). Honors Theses. 2934. https://scholarworks.wmich.edu/honors_theses/2934). There are a number of formulas published on the internet that attempt to perform the same conversion, but after some testing, the formula published by Kolp was determined to be the most accurate. Kolp credits Tom Tango as the source for the formula. Tango works for MLB Advanced Media, helping to facilitate the development and deployment of Statcast data.

```
# Compute Spray Angle of each batted ball and add a new column containing results.

 # Transform x coordinate so that home plate is at 0.
battedballs <- battedballs %>% mutate(hc_x_Kolp = round(2.33 * (hc_x - 126), 2))

 # Transform y coordinate so that home plate is at 0.
battedballs <- battedballs %>% mutate(hc_y_Kolp = round(2.33 * (204.5 - hc_y), 2))
```

A trigonometric function was then applied to the transformed coordinates to derive a spray angle for each batted ball. Thus, the left field foul line was located at -45 degrees, and the right field foul line was located at 45 degrees. A batted ball hit directly up the middle of the field (toward the pitcher's mound) would have a spray angle of 0.

```
 # Compute spray_angle
battedballs <- battedballs %>%
  mutate(spray_angle_Kolp = round((180 / pi) * atan(hc_x_Kolp / hc_y_Kolp), 1))
```

Lastly, we adjusted the spray angle measurements for the side of the plate the batter stands on, so that batted balls hit to the pull side (the side of the field the batter stands on, with respect to home plate) will always have a negative spray angle. The final form of our Spray Angle feature was named *spray_angle_adj*.

```
 # Adjust for side of plate batter stands on, so that batted balls hit to the side of
 # the field the batter stands on (with respect to home plate) will always have a
 # negative spray angle.

battedballs <- battedballs %>%
  mutate(spray_angle_adj = ifelse(stand == "L", -spray_angle_Kolp, spray_angle_Kolp))
```

Having finalized our Spray Angle data, we turned our attention to its interactive partner, if_fielding_alignment. To facilitate the development of models, we needed to convert it from a categorical variable into a numeric variable. Numeric values of 1, 2, and 3 corresponded to the categorical values "Standard", "Strategic", and "Infield shift", respectively.

```
# Convert our categorical predictor variable, if_fielding_alignment, into a numeric
# variable.

battedballs <- battedballs %>% mutate(num_if_alignment =
                                        as.numeric(if_fielding_alignment))
```

We also added a more granular classification scheme for batted ball types. [3] This approach uses six categories, each with a unique set of features and good year-to-year correlations. The new batted ball types were added to *battedballs* under the variable name *adv_bb_type*.

```
# Add a more precise division of batted ball types (adv_bb_type) to more accurately
# categorize the values of our launch_angle predictor.

battedballs <- battedballs %>%
  mutate(adv_bb_type = cut(battedballs$launch_angle,
                           breaks = c(-90, -0.1, 10, 19, 26, 39, 90),
                           labels = c("low_ground_ball", "high_ground_ball",
                                      "low_line_drive", "high_line_drive",
                                      "fly_ball", "popup")))
```

[3] Perpetua, Andrew, "Launch Angle Derived Batted Ball Types" (2017). https://fantasy.fangraphs.com/anglebbtypes/.

To facilitate plotting, we added a categorical variable based on *launch_speed*.

```
# Add a categorical variable based on *launch_speed*.

battedballs <- battedballs %>%
  mutate(launch_speed_cat = cut(battedballs$launch_speed,
                                breaks = c(10, 69.9, 79.9, 89.9, 100, 125),
                                labels =
                                  c("<70", "70-79.9", "80-89.9", "90-100", ">100")))
```

We also added a categorical variable based on *spray_angle* to facilitate plotting.

```
# Add categorical variables based on *spray_angle_Kolp* and *spray_angle_adj*.

battedballs <- battedballs %>%
  mutate(spray_angle_Kolp_cat = cut(battedballs$spray_angle_Kolp,
                                    breaks = c(-90, -45.1, -40, -31, -22, -9, 9, 25, 35,
                                               43, 45, 90),
                                    labels = c("-90:-45.1", "LF Line (-45:-40)",
                                               "3rd Baseman (-39.9:-31)",
                                               "Hole 5-6 (-30.9:-22)",
                                               "Shortstop (-21.9:-9)",
                                               "Hole Middle (-8.9:9)",
                                               "2nd Baseman (9.1:25)",
                                               "Hole 3-4 (25.1:35)",
                                               "1st Baseman (35.1:43)",
                                               "RF Line (43.1:45)", "45.1:90")))

battedballs <- battedballs %>%
  mutate(spray_angle_adj_cat = cut(battedballs$spray_angle_adj,
                                   breaks = c(-90, -45.1, -40, -31, -22, -9, 9, 25, 35,
                                              43, 45, 90),
                                   labels = c("-90:-45.1", "Pulled Down Line (-45:-40)",
                                              "Pulled Corner Inf (-39.9:-31)",
                                              "Pulled Side Hole (-30.9:-22)",
                                              "Pulled Mid Inf (-21.9:-9)",
                                              "Middle Hole (-8.9:9)",
                                              "Oppo Mid Inf (9.1:25)",
                                              "Oppo Side Hole (25.1:35)",
                                              "Oppo Corner Inf (35.1:43)",
                                              "Oppo Down Line (43.1:45)", "45.1:90")))
```

Finally, we added a categorical variable for *hp_to_1b*.

```
# Add a categorical variable based on *hp_to_1b*.

battedballs <- battedballs %>%
  mutate(hp_to_1b_cat = cut(battedballs$hp_to_1b,
                                breaks = c(3.9, 4.2, 4.5, 4.8, 5.1),
                                labels = c("3.9-4.2", "4.21-4.5", "4.51-4.8",
                                           "4.81-5.1")))
```

We then reordered the columns in *battedballs* into a more logical sequence.

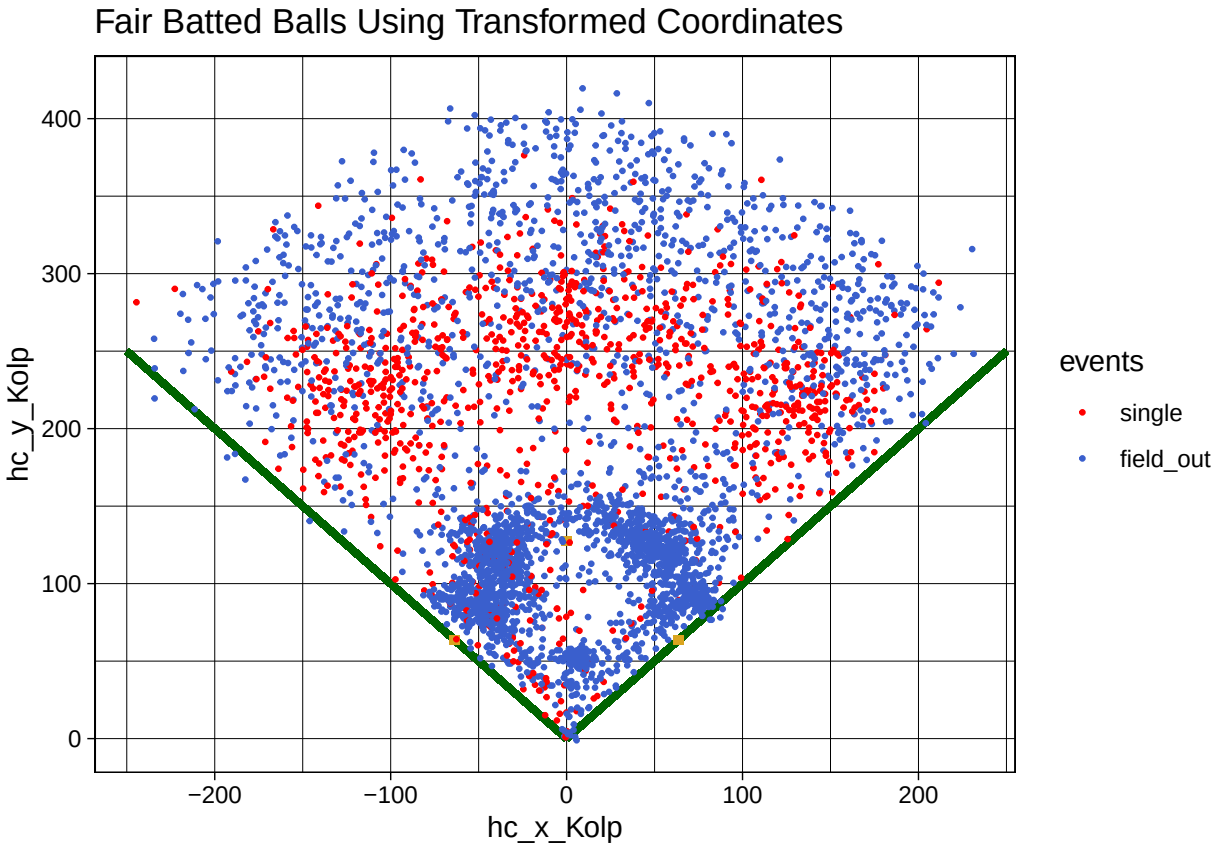
```
# Reorder the columns in battedballs.

battedballs <- battedballs %>% dplyr::select(game_date, batter, player_name, age, stand,
      position, team, bb_type, adv_bb_type, events, des, hp_to_1b, hp_to_1b_cat,
      launch_speed, launch_speed_cat, launch_angle, hc_x, hc_y, hc_x_Kolp, hc_y_Kolp,
      spray_angle_Kolp, spray_angle_Kolp_cat, spray_angle_adj, spray_angle_adj_cat,
      hit_location, hit_distance_sc, if_fielding_alignment, num_if_alignment,
      of_fielding_alignment, launch_speed_angle, game_pk, game_year, game_type,
      home_team, away_team, pitcher, p_throws, description, type, foul)
```

We were ready to take a closer look at our data.

Exploring the Data

Visualizing a random sample of 4,000 of our 100,584 batted balls confirmed the reasonableness of the transformed coordinates. Only a handful of data points fall outside of our superimposed third base and first base foul lines. In addition, batted balls resulting in outs are clustered where fielders are normally positioned, while those resulting in singles tend to end up beyond the infielders and in front of where the outfielders are normally positioned.

```
# Visualize batted balls resulting in singles or outs.

set.seed(1)
battedballs_sample <- sample_n(battedballs, 4000, replace = FALSE)

ggplot(battedballs_sample, aes(hc_x_Kolp, hc_y_Kolp, color = events)) +
  scale_color_manual(values = c("red", "royalblue3")) +
  geom_segment(x = 0, y = 0, xend = 250, yend = 250, color = "dark green", size = 1.3) +
  geom_segment(x = 0, y = 0, xend = -250, yend = 250, color = "dark green", size = 1.3) +
  geom_tile(aes(x = 63.64, y = 63.64, width = 5, height = 5), color = "goldenrod",
            fill = "goldenrod") +
  geom_tile(aes(x = -63.64, y = 63.64, width = 5, height = 5), color = "goldenrod",
            fill = "goldenrod") +
  geom_tile(aes(x = 0, y = 127.28, width = 5, height = 5), color = "goldenrod",
            fill = "goldenrod") +
  geom_point(size = 0.5) + theme_linedraw() +
  labs(title = "Fair Batted Balls Using Transformed Coordinates")
```



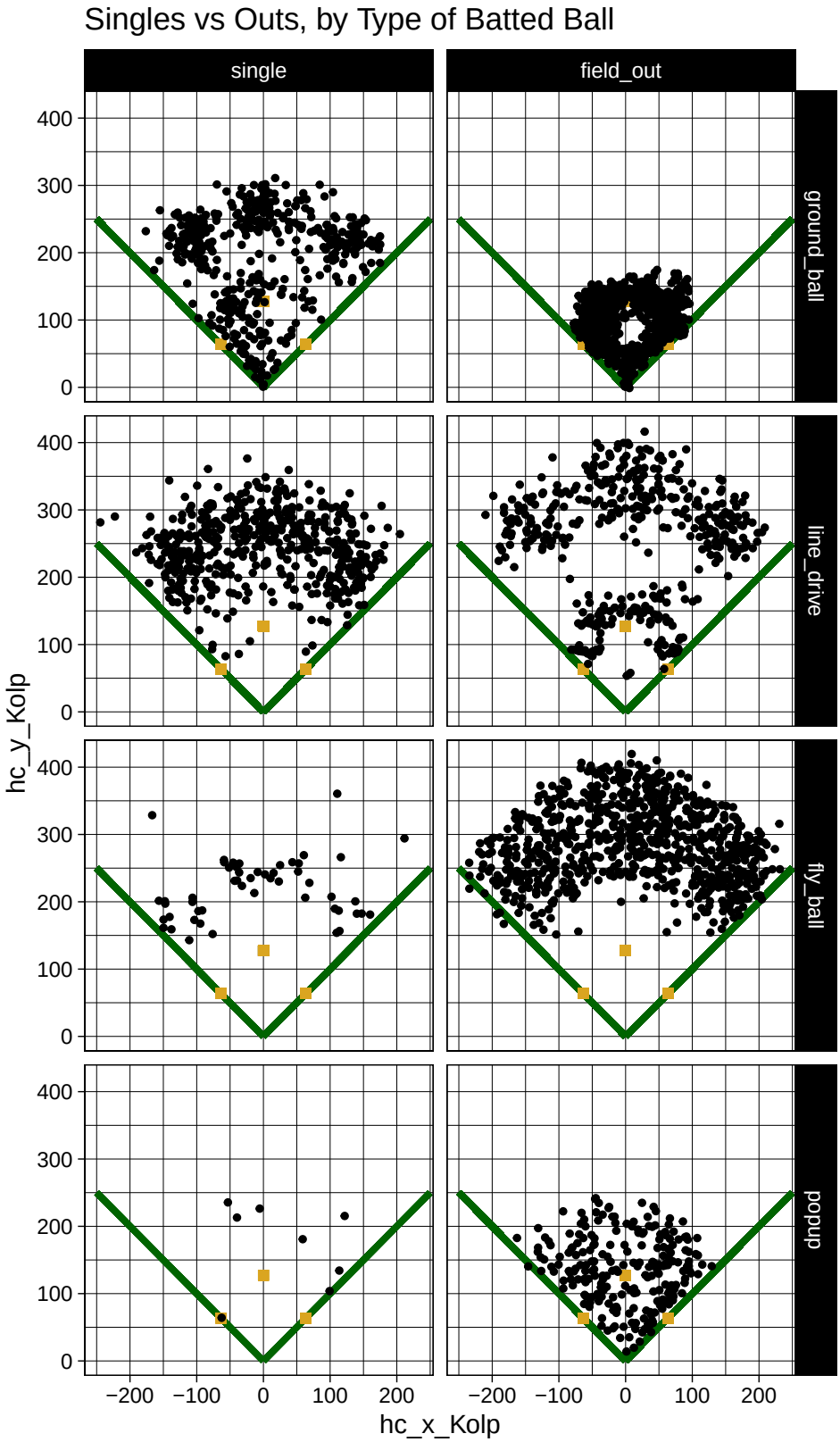
The plots of batted ball coordinates revealed distinctive patterns distinguishing singles from outs. We could see that many of the singles ended up in shallow left field, center field, and right field, where they were first touched by an outfielder. But what were the characteristics of those batted balls? More precisely, what, if anything, did the batters do differently to get those singles? Would any, some, or all of our batter-centric features (Exit Velocity, Launch Angle, Spray Angle, and Home to First) be able to distinguish singles from outs well enough to be included in a model that could reliably predict whether a batted ball would result in a single or an out? We had to begin to understand the relationships between our features and our outcomes.

Launch Angle

We first looked at singles and outs by *launch_angle*. You can view batted ball type (*bb_type* in *battedballs*) as a categorical version of our *launch_angle* feature. The traditional batted ball types are "ground_ball" (launch angle less than 10 degrees), "line_drive" (10-25 degrees), "fly ball" (25-50 degrees), and "popup" (greater than 50 degrees). Ground balls comprise 50% of *battedballs*, followed by line drives (23%), fly balls (21%), and pop ups (6%).

```
# Visualize singles and outs, by type of batted ball.

ggplot(battedballs_sample, aes(hc_x_Kolp, hc_y_Kolp)) +
  geom_segment(x = 0, y = 0, xend = 250, yend = 250, color = "dark green", size = 1.3) +
  geom_segment(x = 0, y = 0, xend = -250, yend = 250, color = "dark green", size = 1.3) +
  geom_tile(aes(x = 63.64, y = 63.64, width = 15, height = 15), color = "goldenrod",
            fill = "goldenrod") +
  geom_tile(aes(x = -63.64, y = 63.64, width = 15, height = 15), color = "goldenrod",
            fill = "goldenrod") +
  geom_tile(aes(x = 0, y = 127.28, width = 15, height = 15), color = "goldenrod",
```

```
           fill = "goldenrod") +
  geom_point(size = 1) + theme_linedraw() +
  labs(title = "Singles vs Outs, by Type of Batted Ball") +
  facet_grid(bb_type ~ events)
```



A close look at the plots reveals something that perhaps the casual baseball fan doesn't consciously notice. As expected, the ground ball plots confirm that ground ball outs are made all around the infield, though

fewer are made on balls hit up the middle past the pitcher. But the real revelation appears in the plot of ground ball singles, where we see that significantly more singles are hit to the left side than the right side of the infield. This makes sense because a fielder picking up a ground ball on the left side of the infield has a much longer throw to first base to get the batter out. Batters are able to beat the throws to first base on some of the dribblers and choppers hit down the third base line, and on grounders hit into the hole between the shortstop and third baseman. These are "infield hits"; they never make it to the outfield, yet the batter reaches first base safely. This suggest we should consider including *spray_angle_Kolp* as a predictor in our models, in addition to *spray_angle_adj*. *spray_angle_Kolp* is not adjusted for the side of the plate the batter bats on, so any ground ball hit to the left side of the infield ought to have a higher probability of resulting in a single.

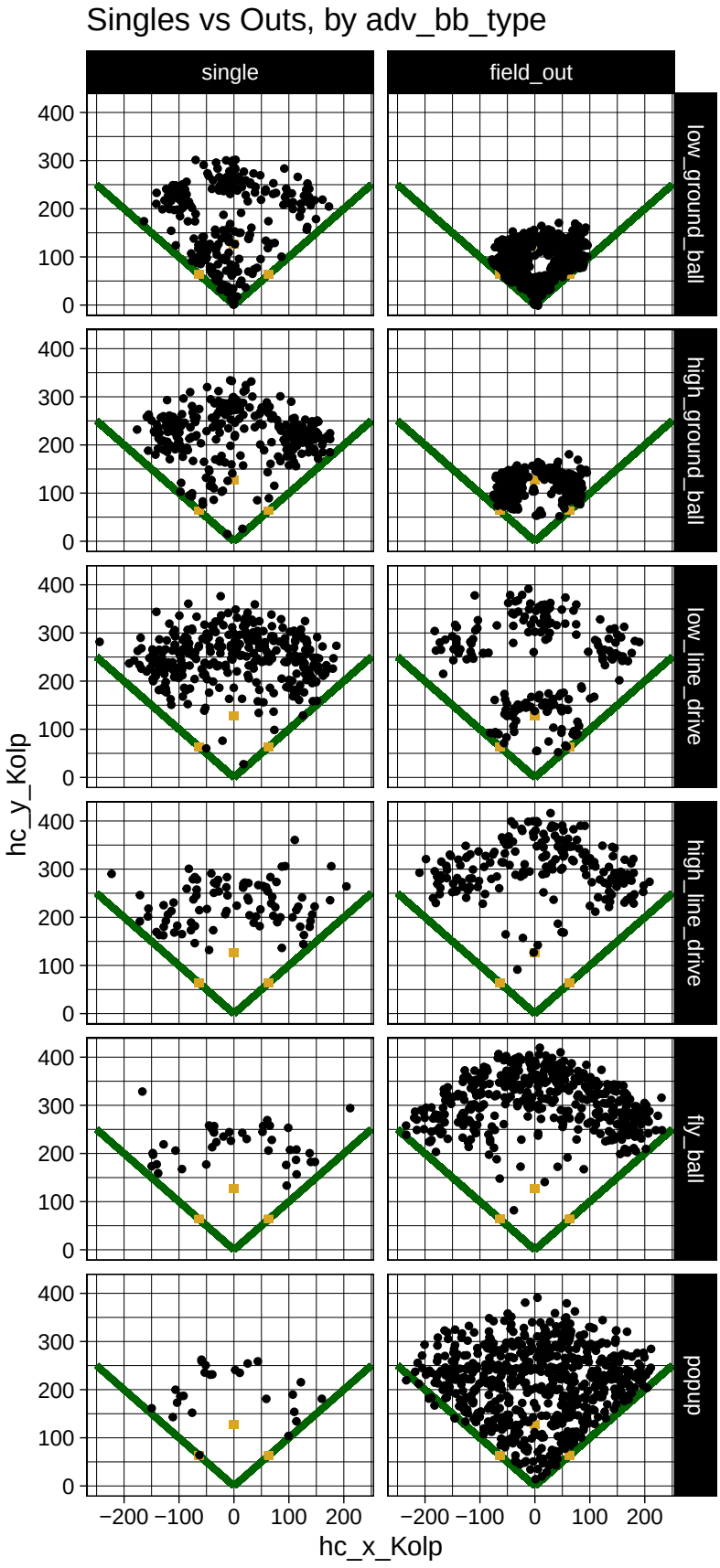
As for the ground ball singles that reach the outfield, here we would expect *launch_speed* to make a difference in our predictive models. To get to the outfield, groundballs have to be hit between the infielders. Ground balls with higher exit velocities would give infielders less time to move toward the expected path of the ball before the ball rolls past them.

With regard to line drives, the singles are hit between or above the infielders before being stopped by outfielders in shallow left field, center field, and right field. The outs tend to be hit directly where fielders are traditionally positioned. Line drives are generally hit harder than other types of batted balls, giving fielders less time to move toward the ball. So whether a line drive results in a single would appear to depend more on the intial alignment of the fielders when the ball is hit. Popups and fly balls result in relatively few singles because fielders generally have enough time to move under the ball and catch it, regardless of where they are originally positioned when the ball is hit.

We also looked at *adv_bb_type*, the more granular classification scheme for batted ball types. The categories, again defined by Launch Angle, include: Low Ground Ball (less than 0 degrees), High Ground Ball (0-10 degrees), Low Line Drive (10.1-19 degrees), High Line Drive (19.1-26 degrees), Fly Ball (26.1-39 degrees) and Popup (greater than 39 degrees). [4] Low Ground Balls comprise 37% of *battedballs*, followed by High Ground Balls (16%), Popups (15%), Low Line Drives (12%), Fly Balls (12%), and High Line Drives (7%).

```
# Visualize singles and outs, by adv_bb_type.

ggplot(battedballs_sample, aes(hc_x_Kolp, hc_y_Kolp)) +
  geom_segment(x = 0, y = 0, xend = 250, yend = 250, color = "dark green", size = 1.3) +
  geom_segment(x = 0, y = 0, xend = -250, yend = 250, color = "dark green", size = 1.3) +
  geom_tile(aes(x = 63.64, y = 63.64, width = 15, height = 15), color = "goldenrod",
            fill = "goldenrod") +
  geom_tile(aes(x = -63.64, y = 63.64, width = 15, height = 15), color = "goldenrod",
            fill = "goldenrod") +
  geom_tile(aes(x = 0, y = 127.28, width = 15, height = 15), color = "goldenrod",
            fill = "goldenrod") +
  geom_point(size = 1) + theme_linedraw() +
  labs(title = "Singles vs Outs, by adv_bb_type") +
  facet_grid(adv_bb_type ~ events)
```

[4] Perpetua, Andrew, "Launch Angle Derived Batted Ball Types" (2017). https://fantasy.fangraphs.com/anglebbtypes/.

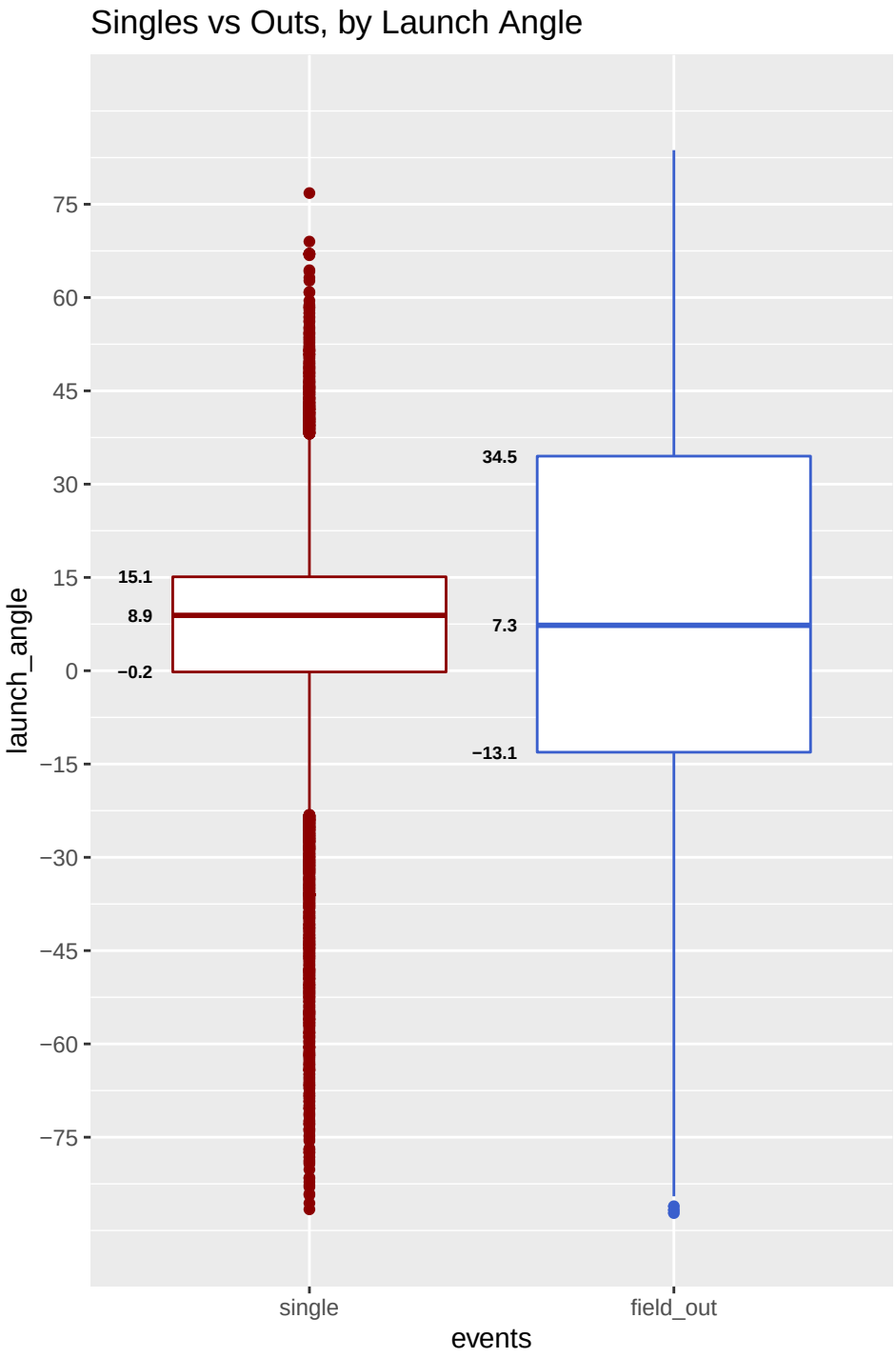

Singles vs Outs, by adv_bb_type
single
field_out
low_ground_ball
high_ground_ball
low_line_drive
high_line_drive
fly_ball
popup
hc_y_Kolp
hc_x_Kolp
400
300
200
100
0
−200
−100
0
100
200

Visually, we can now see that it's Low Ground Balls with negative launch angles that can result in singles when hit to the left side of the infield. This makes sense, since balls chopped into the ground give the batter more time to reach first base safely. We can also see that the high versions of ground balls and line drives end up somewhat further from home plate than their low counterparts. This was expected.

To better quantify what was depicted in our plots of batted ball coordinates, we created box plots showing the distribution of *launch_angle* by singles and outs. The first and third quartiles (the 25th and 75th percentiles), along with the medians were added to the plots.

```
# Create boxplots to visualize the distribution of launch_angle by singles and outs.

bb_outcome_by_launch_angle <- ggplot(battedballs,
                                     aes(x = events, y = launch_angle, color = events)) +
  geom_boxplot() +
  scale_y_continuous(breaks = seq(-75, 75, by = 15),
                     labels = c("-75", "-60", "-45", "-30", "-15", "0",  "15", "30",
                                "45", "60", "75"),
                     limits = c(-90, 90)) +
  scale_color_manual(values = c("red4", "royalblue3")) +
  ggtitle("Singles vs Outs, by Launch Angle") +
  theme(legend.position = "none")

bb_outcome_by_launch_angle_data <- layer_data(bb_outcome_by_launch_angle)

launch_angle_single_1quartile <- bb_outcome_by_launch_angle_data[1, 3]
launch_angle_single_median <- bb_outcome_by_launch_angle_data[1, 4]
launch_angle_single_3quartile <- bb_outcome_by_launch_angle_data[1, 5]
launch_angle_out_1quartile <- bb_outcome_by_launch_angle_data[2, 3]
launch_angle_out_median <- bb_outcome_by_launch_angle_data[2, 4]
launch_angle_out_3quartile <- bb_outcome_by_launch_angle_data[2, 5]
```

```
ggplot(battedballs,
       aes(x = events, y = launch_angle, color = events)) +
  geom_boxplot() +
  scale_y_continuous(breaks = seq(-75, 75, by = 15),
                     labels = c("-75", "-60", "-45", "-30", "-15", "0",  "15", "30",
                                "45", "60", "75"),
                     limits = c(-90, 90)) +
  scale_color_manual(values = c("red4", "royalblue3")) +
  ggtitle("Singles vs Outs, by Launch Angle") +
  theme(legend.position = "none") +
  annotate("text", x = c(.57, .57, .57), size = 2.5, fontface = 2, hjust = 1,
           y = c(launch_angle_single_1quartile, launch_angle_single_median,
                 launch_angle_single_3quartile),
           label = c(launch_angle_single_1quartile, launch_angle_single_median,
                     launch_angle_single_3quartile)) +
  annotate("text", x = c(1.57,1.57, 1.57), size = 2.5, fontface = 2, hjust = 1,
           y = c(launch_angle_out_1quartile, launch_angle_out_median,
                 launch_angle_out_3quartile),
           label = c(launch_angle_out_1quartile, launch_angle_out_median,
                     launch_angle_out_3quartile))
```



The boxplots revealed that singles tend to have a launch angle between 0 and 15 degrees. Many outs fell within this range as well, but outs had an interquartile range (-13.1 degrees to 34.5 degrees) more than three times as large as singles. Launch Angle was certainly going to help us distinguish singles from outs.

This was confirmed by quickly computating the proportion of each batted ball type resulting in a single. Singles comprised about 25% of our entire *battedballs* dataset, so this served as our baseline. Approximately 63% of Low Line Drives resulted in singles, followed by 45% of High Ground Balls, 29% of High Line Drives, 17% of Low Ground Balls, 9% of Fly Balls, and 2% of Popups. Recall that the launch angles for Low Line Drives and High Ground Balls range from 0 to 19 degrees. The variability was significant; launch_angle

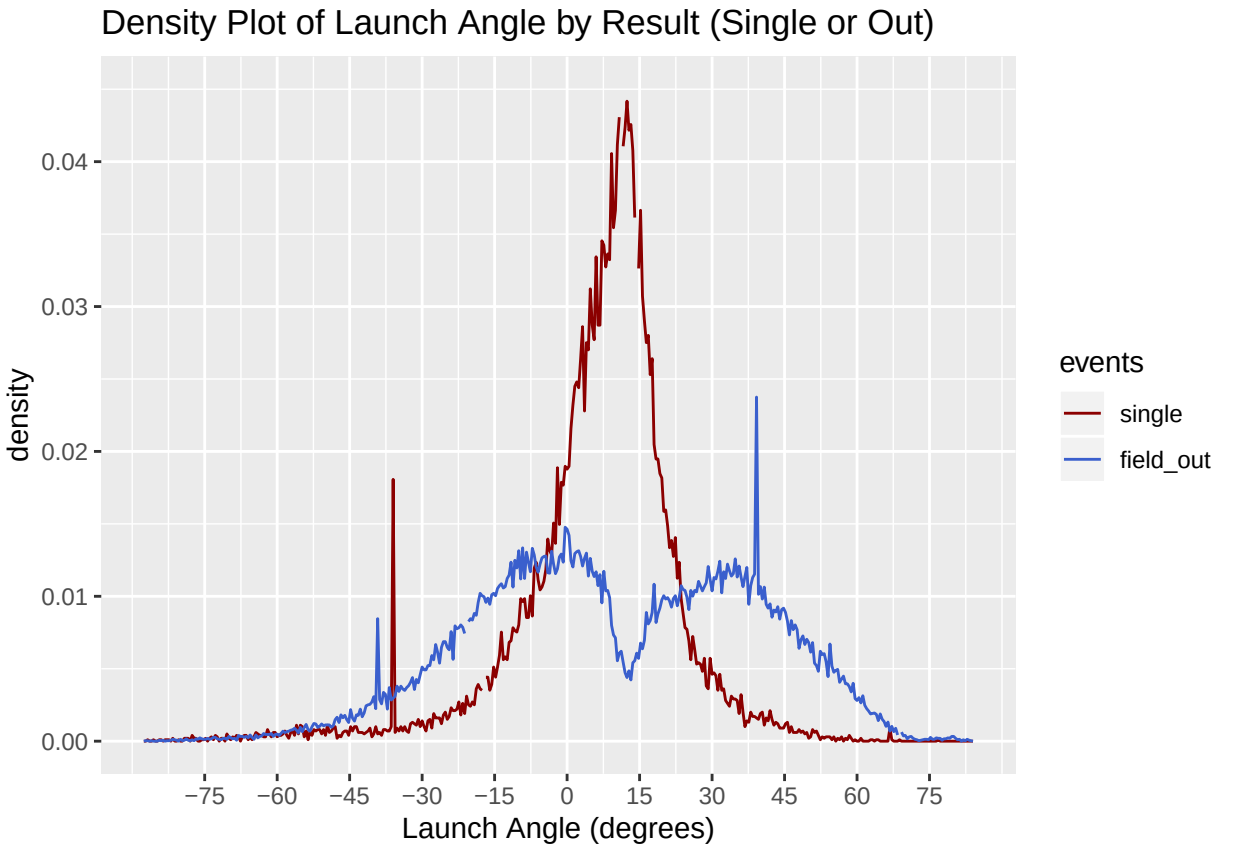
would be a very important predictor in our models.

```
# Compute the proportion of singles for each adv_bb_type.

la_prop_lgb <- (battedballs %>% filter(adv_bb_type == "low_ground_ball" &
                                       battedballs$events == "single") %>%
                  summarize(n = n())) / (battedballs %>%
                                           filter(adv_bb_type == "low_ground_ball") %>%
                                           summarize(n = n()))

la_prop_hgb <- (battedballs %>% filter(adv_bb_type == "high_ground_ball" &
                                       battedballs$events == "single") %>%
                  summarize(n = n())) / (battedballs %>%
                                           filter(adv_bb_type == "high_ground_ball") %>%
                                           summarize(n = n()))

la_prop_lld <- (battedballs %>% filter(adv_bb_type == "low_line_drive" &
                                       battedballs$events == "single") %>%
                  summarize(n = n())) / (battedballs %>%
                                           filter(adv_bb_type == "low_line_drive") %>%
                                           summarize(n = n()))

la_prop_hld <- (battedballs %>% filter(adv_bb_type == "high_line_drive" &
                                       battedballs$events == "single") %>%
                  summarize(n = n())) / (battedballs %>%
                                           filter(adv_bb_type == "high_line_drive") %>%
                                           summarize(n = n()))

la_prop_fb <- (battedballs %>% filter(adv_bb_type == "fly_ball" &
                                      battedballs$events == "single") %>%
                 summarize(n = n())) / (battedballs %>%
                                          filter(adv_bb_type == "fly_ball") %>%
                                          summarize(n = n()))

la_prop_pu <- (battedballs %>% filter(adv_bb_type == "popup" &
                                      battedballs$events == "single") %>%
                 summarize(n = n())) / (battedballs %>%
                                          filter(adv_bb_type == "popup") %>%
                                          summarize(n = n()))
```

A comprehensive visualization of these proportions was achieved by creating a density plot. The density plot showed the proportion of singles and outs at each Launch Angle. By plotting proportions (densities) rather than counts, we avoided the distortion caused by one of our outcomes (out) having far more observations than our other outcome (single). The area under a density curve always totals 1.

```
# Plot the density of our Launch Angle predictor by result (single or out).

x_scale <- scale_x_continuous(breaks = seq(-75, 75, by = 15),
                              labels = c("-75", "-60", "-45", "-30", "-15", "0",  "15",
                                         "30", "45", "60", "75"),
                              limits = NULL)
y_scale <- scale_y_continuous(limits = c(0, 0.045))
(la_density <- ggplot(battedballs, aes(x = launch_angle, y = stat(density),
                                       color = events)) +
```

```
  geom_freqpoly(binwidth = 0.4) +
  x_scale +
  y_scale +
  scale_color_manual(values = c(single = "red4", field_out = "royalblue3")) +
  labs(title = "Density Plot of Launch Angle by Result (Single or Out)",
       x = "Launch Angle (degrees)"))
```




```
la_density_data <- layer_data(la_density)
la_density_data <- filter(la_density_data, y != "NA")
```

Density rises for singles and falls for outs between launch angles of about 0 and 13 degrees. Density peaks for singles at a launch angle of 12.4. The density plot results appear to be consistent with those from our box plots and our proportions using *adv_bb_type*. [5]

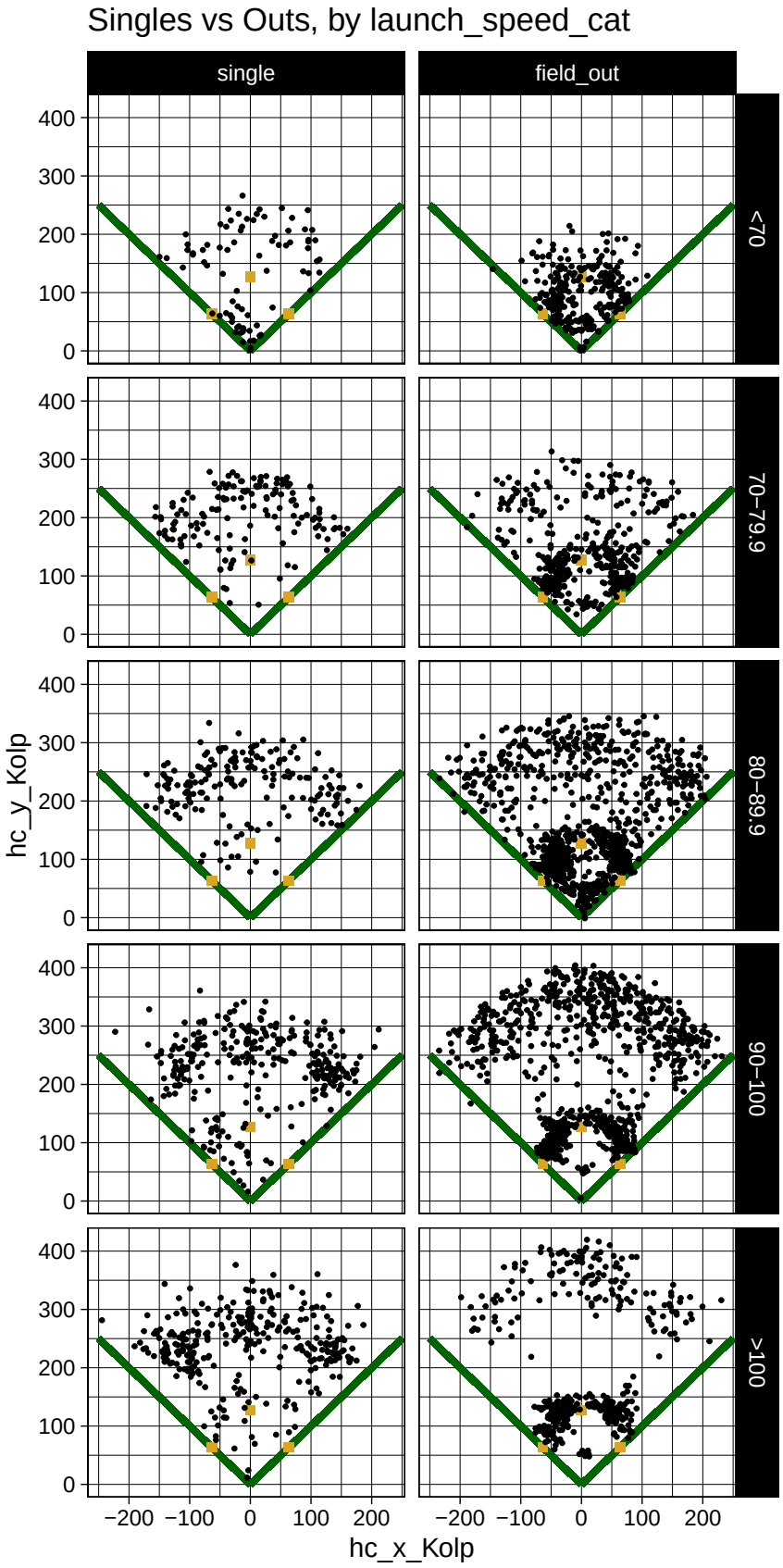
Exit Velocity

MLB believes a hard hit ball likely to result in a hit has an exit velocity of at least 95 miles per hour. But is this true for singles? [6] We again plotted the coordinates of singles and outs, this time by *launch_speed_cat*. The five categories of launch speeds were: less than 70 mph (comprised 11% of *battedballs*), 70-79.9 mph (14% of *battedballs*), 80-89.9 mph (28%), 90-100 (30%), and greater than 100 mph (17%).

[5] Note that the peripheral spikes appearing in the density plot are the result of Baseball Savant's methodology for imputing missing data caused by the Statcast technology failing to obtain measurements on certain batted balls. To provide clarity in this and other density plots, some of these spikes were removed by imposing scale limits on the axes. However, the imputed data was not removed from the *battedballs* dataset itself, as doing so would have introduced bias. Perpetua, Andrew, "Accounting for the"No Nulls" Solution" (2017). https://tht.fangraphs.com/43416-2/.

[6] http://m.mlb.com/glossary/statcast/hard-hit-rate

```r
# Visualize singles and outs, by Exit Velocity.          .

ggplot(battedballs_sample, aes(hc_x_Kolp, hc_y_Kolp)) +
  geom_segment(x = 0, y = 0, xend = 250, yend = 250, color = "dark green", size = 1.3) +
  geom_segment(x = 0, y = 0, xend = -250, yend = 250, color = "dark green", size = 1.3) +
  geom_tile(aes(x = 63.64, y = 63.64, width = 15, height = 15), color = "goldenrod",
            fill = "goldenrod") +
  geom_tile(aes(x = -63.64, y = 63.64, width = 15, height = 15), color = "goldenrod",
            fill = "goldenrod") +
  geom_tile(aes(x = 0, y = 127.28, width = 15, height = 15), color = "goldenrod",
            fill = "goldenrod") +
  geom_point(size = 0.5) + theme_linedraw() +
  labs(title = "Singles vs Outs, by launch_speed_cat") +
  facet_grid(launch_speed_cat ~ events)
```


Singles vs Outs, by launch_speed_cat
single
field_out
<70
70–79.9
80–89.9
90–100
>100
hc_y_Kolp
hc_x_Kolp
0
100
200
300
400
−200
−100

At exit velocities less than 80 mph, we see more outs being made in the infield than the outfield. The opposite appears true for balls hit at least 90 mph; these balls are reaching the outfielders on the fly, so they're being caught despite their high velocity.

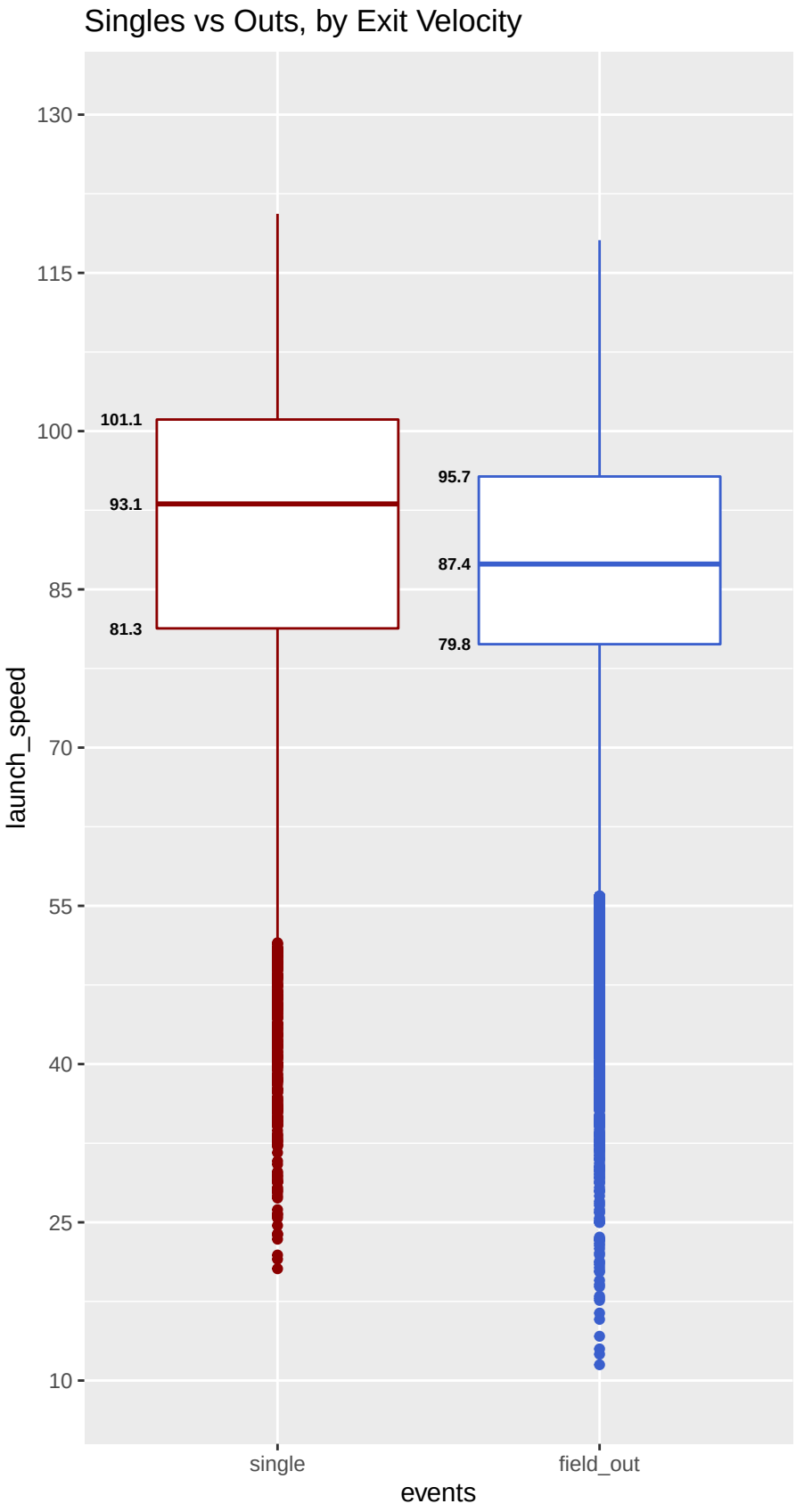
The frequency of outs appears greatest in the middle velocity category, 80 to 89.9 mph. These batted balls are hit hard enough to reach the fielders, but not so hard that fielders don't have enough time to react. By comparison, singles seem to gradually increase in density as exit velocity increases. Again, we see that most of the singles end up in the shallow portions of the outfield, indicating they were hit well enough to get between or over the infielders. Exit velocity would certainly aid the batter in this respect. The singles in the lowest launch speed category are the exception, as a good portion of them result in infield singles on the left side. We saw these infield singles earlier when we analyzed launch angles.

Following the data exploration approach we used for Launch Angle, we next created box plots showing the distribution of *launch_speed* by singles and outs. The first and third quartiles (the 25th and 75th percentiles), along with the medians were added to the plots.

```
# Create boxplots to visualize the distribution of launch_speed by singles and outs.

bb_outcome_by_launch_speed <- ggplot(battedballs,
                                     aes(x = events, y = launch_speed, color = events)) +
  geom_boxplot() +
  scale_y_continuous(breaks = seq(10, 130, by = 15),
                     labels = c("10", "25", "40", "55", "70", "85", "100",  "115", "130"),
                     limits = c(10, 130)) +
  scale_color_manual(values = c("red4", "royalblue3")) +
  ggtitle("Singles vs Outs, by Exit Velocity") +
  theme(legend.position = "none")

bb_outcome_by_launch_speed_data <- layer_data(bb_outcome_by_launch_speed)

launch_speed_single_1quartile <- bb_outcome_by_launch_speed_data[1, 3]
launch_speed_single_median <- bb_outcome_by_launch_speed_data[1, 4]
launch_speed_single_3quartile <- bb_outcome_by_launch_speed_data[1, 5]
launch_speed_out_1quartile <- bb_outcome_by_launch_speed_data[2, 3]
launch_speed_out_median <- bb_outcome_by_launch_speed_data[2, 4]
launch_speed_out_3quartile <- bb_outcome_by_launch_speed_data[2, 5]
```

```
ggplot(battedballs,
       aes(x = events, y = launch_speed, color = events)) +
  geom_boxplot() +
  scale_y_continuous(breaks = seq(10, 130, by = 15),
                     labels = c("10", "25", "40", "55", "70", "85", "100",  "115", "130"),
                     limits = c(10, 130)) +
  scale_color_manual(values = c("red4", "royalblue3")) +
  ggtitle("Singles vs Outs, by Exit Velocity") +
  theme(legend.position = "none") +
  annotate("text", size = 2.5, fontface = 2, hjust = 1, x = c(.58, .58, .58),
           y = c(launch_speed_single_1quartile, launch_speed_single_median,
                 launch_speed_single_3quartile),
           label = c(launch_speed_single_1quartile, launch_speed_single_median,
                     launch_speed_single_3quartile)) +
  annotate("text", size = 2.5, fontface = 2, hjust = 1, x = c(1.6,1.6, 1.6),
           y = c(launch_speed_out_1quartile, launch_speed_out_median,
                 launch_speed_out_3quartile),
           label = c(launch_speed_out_1quartile, launch_speed_out_median,
```

```
                launch_speed_out_3quartile))
```


Singles vs Outs, by Exit Velocity
130
115
100
85
70
55
40
25
10
launch_speed
101.1
93.1
81.3
95.7
87.4
79.8
single
field_out
events

Here, the Exit Velocity boxplots for singles and outs were more similar to each other than the Launch Angle boxplots. Still, the median and third quartile were more than five miles per hour faster for singles than outs. It looked like Exit Velocity was going to be a helpful predictor in our models, but perhaps not to the same extent as Launch Angle.

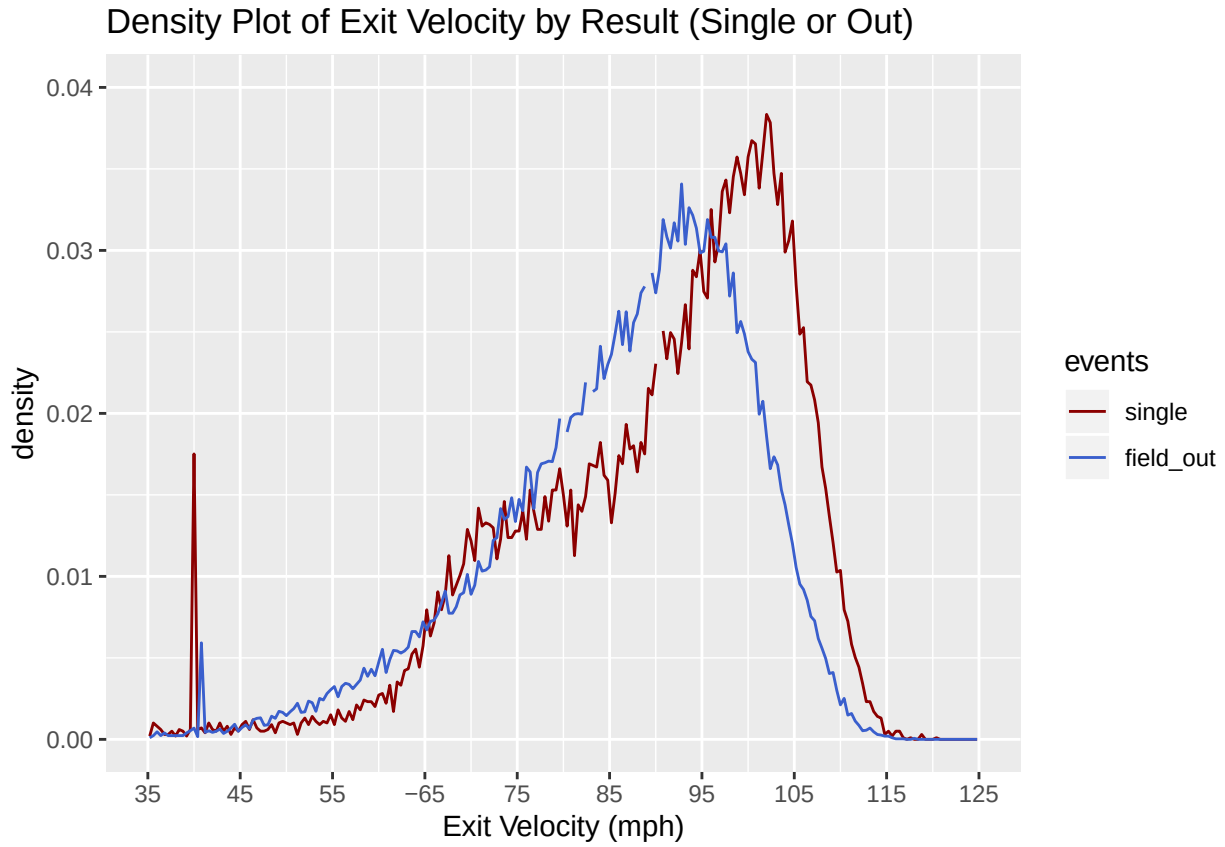
We next computed the proportion of batted balls resulting in a single for each launch speed category. Recall that singles comprise about 25% of our entire *battedballs* dataset, so that's our baseline. From the lowest to the highest velocity categories, the computed proportions (22%, 24%, 15%, 26%, and 43%) revealed an interesting pattern: the frequency of singles dropped in the mid-speed category (80 to 90 mph) and rose in the fastest category (above 100 mph), but were otherwise near the overall singles average of 25% of batted balls. That is, singles are hit with greatest frequency at the lower and higher ends of the exit velocity spectrum, while the 28% of batted balls hit between 80 and 90 miles per hour were more easily turned into outs!

That makes some sense because fielders need the ball to make a field out. If the ball is hit too slowly, the fielder can't get to the ball quickly enough to make an out. If the ball is hit too hard, the fielder isn't able to react fast enough to glove the ball. Still, to see this reflected in the numbers was a bit surprising. More importantly, the relative lack of variability in Exit Velocity across singles and outs again suggested that our *launch_speed* predictor was not going to be as important as *launch_angle* in our models.

```
# Compute the proportion of singles for each launch_speed_cat.

ls_prop_lt70 <- (battedballs %>% filter(launch_speed_cat == "<70" &
                                        battedballs$events == "single") %>%
                   summarize(n = n())) / (battedballs %>%
                                            filter(launch_speed_cat == "<70") %>%
                                            summarize(n = n()))

ls_prop_to80 <- (battedballs %>% filter(launch_speed_cat == "70-79.9" &
                                        battedballs$events == "single") %>%
                   summarize(n = n())) / (battedballs %>%
                                            filter(launch_speed_cat == "70-79.9") %>%
                                            summarize(n = n()))

ls_prop_to90 <- (battedballs %>% filter(launch_speed_cat == "80-89.9" &
                                        battedballs$events == "single") %>%
                   summarize(n = n())) / (battedballs %>%
                                            filter(launch_speed_cat == "80-89.9") %>%
                                            summarize(n = n()))

ls_prop_100 <- (battedballs %>% filter(launch_speed_cat == "90-100" &
                                       battedballs$events == "single") %>%
                  summarize(n = n())) / (battedballs %>%
                                           filter(launch_speed_cat == "90-100") %>%
                                           summarize(n = n()))

ls_prop_gt100 <- (battedballs %>% filter(launch_speed_cat == ">100"
                                       & battedballs$events == "single") %>%
                    summarize(n = n())) / (battedballs %>%
                                             filter(launch_speed_cat == ">100") %>%
                                             summarize(n = n()))
```

A density plot would provide a more thorough presentation of these proportions. The density plot we constructed showed the proportion of singles and outs at each Exit Velocity.

```
# Plot the density of our Exit Velocity predictor (*launch_speed*) by result (single or
# out).

x_scale <- scale_x_continuous(breaks = seq(35, 125, by = 10),
                              labels = c("35", "45", "55", "-65", "75", "85", 95,  "105",
                                         "115", "125"),
                              limits = c(35, 125))
y_scale <- scale_y_continuous(limits = c(0, 0.04))
(ls_density <- ggplot(battedballs, aes(x = launch_speed, y = stat(density),
                                       color = events)) +
    geom_freqpoly(binwidth = 0.4) +
    x_scale +
    y_scale +
    scale_color_manual(values = c(single = "red4", field_out = "royalblue3")) +
    labs(title = "Density Plot of Exit Velocity by Result (Single or Out)",
         x = "Exit Velocity (mph)"))
```




```
ls_density_data <- layer_data(ls_density)
ls_density_data <- filter(ls_density_data, y != "NA", x != "NA")
```

Our tentative conclusions above were largely corroborated by the density plots. The singles plot and the outs plot overlap for the most part, but the frequency of singles exceeds that of outs both at higher and lower exit velocities. Here, one wonders whether all the singles imputed around 40 miles per hour changed the lower tail of the singles plot in a significant way. In spite of the imputed data, the density plots reflect the same tendencies revealed throughout our exploration of the Exit Velocity data.

Spray Angle

Spray Angle is the most complex of our key predictors. You'll recall that we expected (or at least hoped) our Spray Angle predictor would be aided by an interactive variable, Infield Fielding Alignment (*if_fielding_alignment* in *battedballs*). Thus, Spray Angle's usefulness in our models was going to be restricted to batted balls with a launch angle of 10 degrees or less (Low Ground Balls and High Ground Balls in our *adv_bb_type* variable. Accordingly, we created a new random sample of just batted balls with launch angles of 10 degrees or less to facilitate plotting.

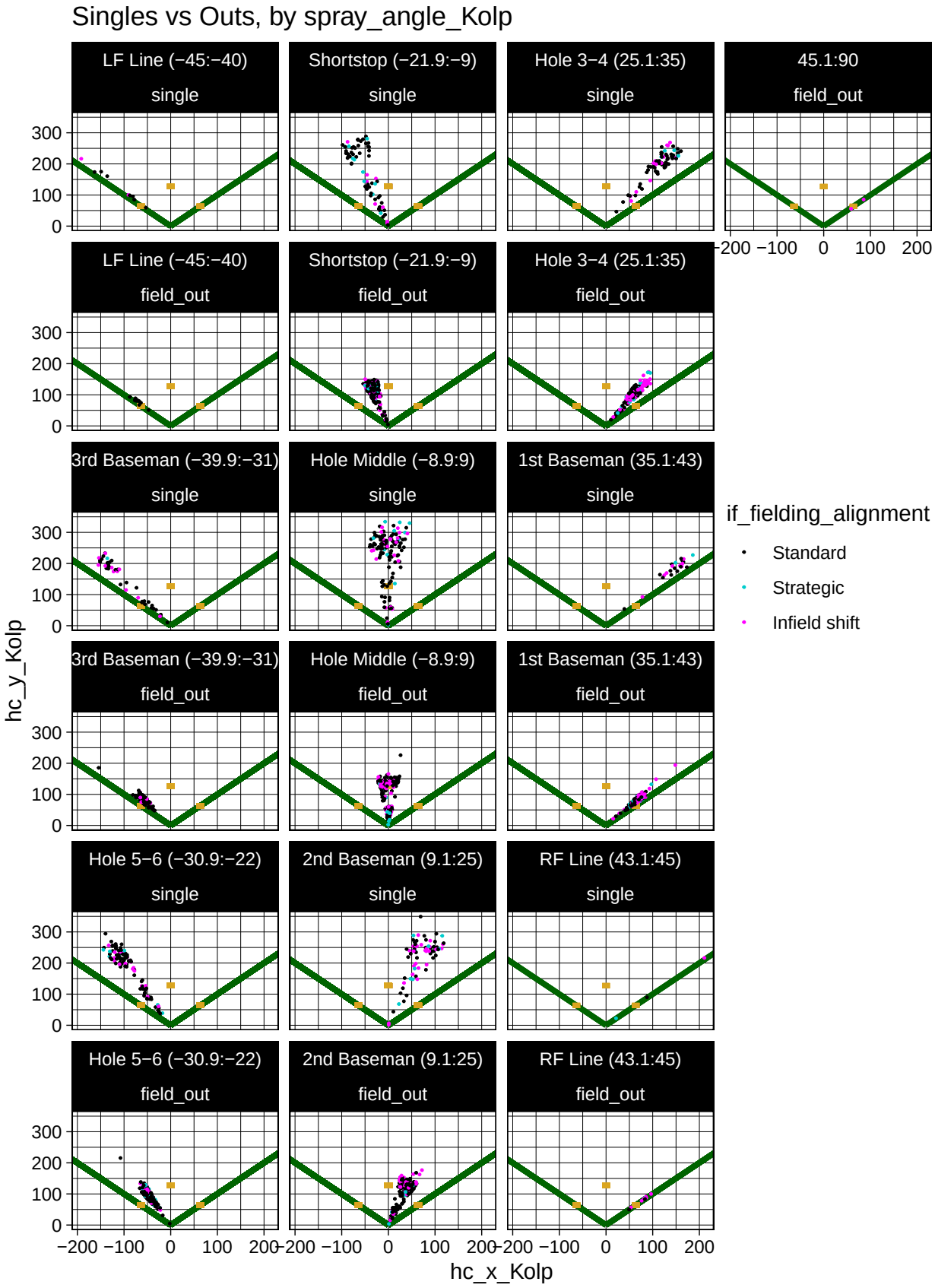
```
# Create a random sample of 2,000 ground balls.

set.seed(1)
battedballs_gb_sample <- sample_n(filter(battedballs, launch_angle <= 10), 2000,
                                  replace = FALSE)
```

This time, we would be plotting singles and outs by both a predictor (first, we'll use *spray_angle_Kolp*) and its interactive partner (*if_fielding alignment*). Kolp's version of Spray Angle is not adjusted for the side of the plate on which the batter hits, so batted balls will appear on the true left and right sides of the field.

```
# Visualize singles and outs, by Spray Angle (Kolp's version).          .

spray_angle_Kolp_plot <- ggplot(battedballs_gb_sample,
                                aes(hc_x_Kolp, hc_y_Kolp,
                                    color = if_fielding_alignment)) +
  scale_color_manual(values = c(Standard = "black", Strategic = "cyan3",
                                `Infield shift` = "magenta1")) +
  geom_segment(x = 0, y = 0, xend = 250, yend = 250, color = "dark green", size = 1.3) +
  geom_segment(x = 0, y = 0, xend = -250, yend = 250, color = "dark green", size = 1.3) +
  geom_tile(aes(x = 63.64, y = 63.64, width = 15, height = 15), color = "goldenrod",
            fill = "goldenrod") +
  geom_tile(aes(x = -63.64, y = 63.64, width = 15, height = 15), color = "goldenrod",
            fill = "goldenrod") +
  geom_tile(aes(x = 0, y = 127.28, width = 15, height = 15), color = "goldenrod",
            fill = "goldenrod") +
  geom_point(size = 0.2) + theme_linedraw() +
  labs(title = "Singles vs Outs, by spray_angle_Kolp") +
  facet_wrap(~ spray_angle_Kolp_cat + events, dir = "v", nrow = 6)
```

```
# gtable_show_names(spray_angle_Kolp_plot)
```

```
reposition_legend(spray_angle_Kolp_plot, position = 'center', panel = 'panel-1-6')
```


Singles vs Outs, by spray_angle_Kolp
LF Line (−45:−40)
single
Shortstop (−21.9:−9)
single
Hole 3−4 (25.1:35)
single
45.1:90
field_out
LF Line (−45:−40)
field_out
Shortstop (−21.9:−9)
field_out
Hole 3−4 (25.1:35)
field_out
3rd Baseman (−39.9:−31)
single
Hole Middle (−8.9:9)
single
1st Baseman (35.1:43)
single
3rd Baseman (−39.9:−31)
field_out
Hole Middle (−8.9:9)
field_out
1st Baseman (35.1:43)
field_out
Hole 5−6 (−30.9:−22)
single
2nd Baseman (9.1:25)
single
RF Line (43.1:45)
single
Hole 5−6 (−30.9:−22)
field_out
2nd Baseman (9.1:25)
field_out
RF Line (43.1:45)
field_out
hc_y_Kolp
hc_x_Kolp
if_fielding_alignment
Standard
Strategic
Infield shift

The first thing to note is that each panel displays batted balls within a certain range of spray angles. Together, the ranges cover the entire field from the left field foul line to the right field foul line. Instead of making each range the same size, an effort was made to define a range according to whether it extends into holes between infielders, or projects directly toward an infielder. For example, starting on the left side, the first panel displays a range of spray angles (-45 to -39 degress) extending into a gap down the left field foul line. A third baseman would typically have to dive to his right to stop a ball hit in this range. The next panel to the right encompasses a range (-39 to -32 degrees) projecting directly toward the standard position of the third baseman. The next panel to the right (-32 to -20 degrees) displays balls heading toward the hole between the areas typically covered by the shortstop and third baseman. The panels progress in this manner from left to right until the right field foul line is reached.

The colors of the plotted batted balls indicate the alignment of the infielders at the time the pitch was delivered. Balls hit when the infielders were in their traditional locations (about 69% of batted balls) are shown in black, while those hit when the infielders were fully shifted (about 23% of batted balls) are shown in magenta. When the infielders were strategically positioned (about 8% of batted balls), the balls are plotted in cyan.

As would be expected, the plots show most of the singles reaching the outfield (except for the spattering of infield hits we've repeatedly seen on the left side of the infield). Outs, on the other hand, were stopped in the infield. No surprise there. More interesting is the observation that singles appear to be more dense in the panels showing spray angles that extend into gaps between the infielders, such as the Hole 5-6, Hole Middle, and Hole 3-4 panels. That suggests that *spray_angle_Kolp* ought to be able to make a meaningful contribution to our predictive models.

Patterns from the infield alignment are more difficult to discern. There definitely seem to be more outs made up the middle and on the right side of the infield when the infielders are fully shifted. This didn't make sense until we realized that left-handed batters face non-standard infield alignments about half the time, while right-handed batters face non-standard infield alignments only 20% of the time.

But the key question remained as to whether infield alignment could help Spray Angle distinguish singles from outs. In theory, when infielders are in a non-standard alignment, we ought to see more singles being hit where infielders are traditionally located, and more outs being made where infield gaps are traditionally located. But this isn't readily apparent across the singles and outs panels.

It could be that the plot simply doesn't have the fine resolution to capture what's happening for the 4% of batted balls having launch angles of 10% or less, resulting in a single against a non-standard infield alignment. Perhaps infield shifts are so effective at preventing singles that we don't have enough singles in our sample to tell the story. It's also possible that the spray angle ranges simply aren't informative enough about the exact locations of the infielders. The picture may also be muddied by the fact that *spray_angle_Kolp* doesn't consider the side of the plate on which the batter stands. It seems as if we need to know whether the batter is pulling the ball into a shift, or hitting the ball away from the shift.

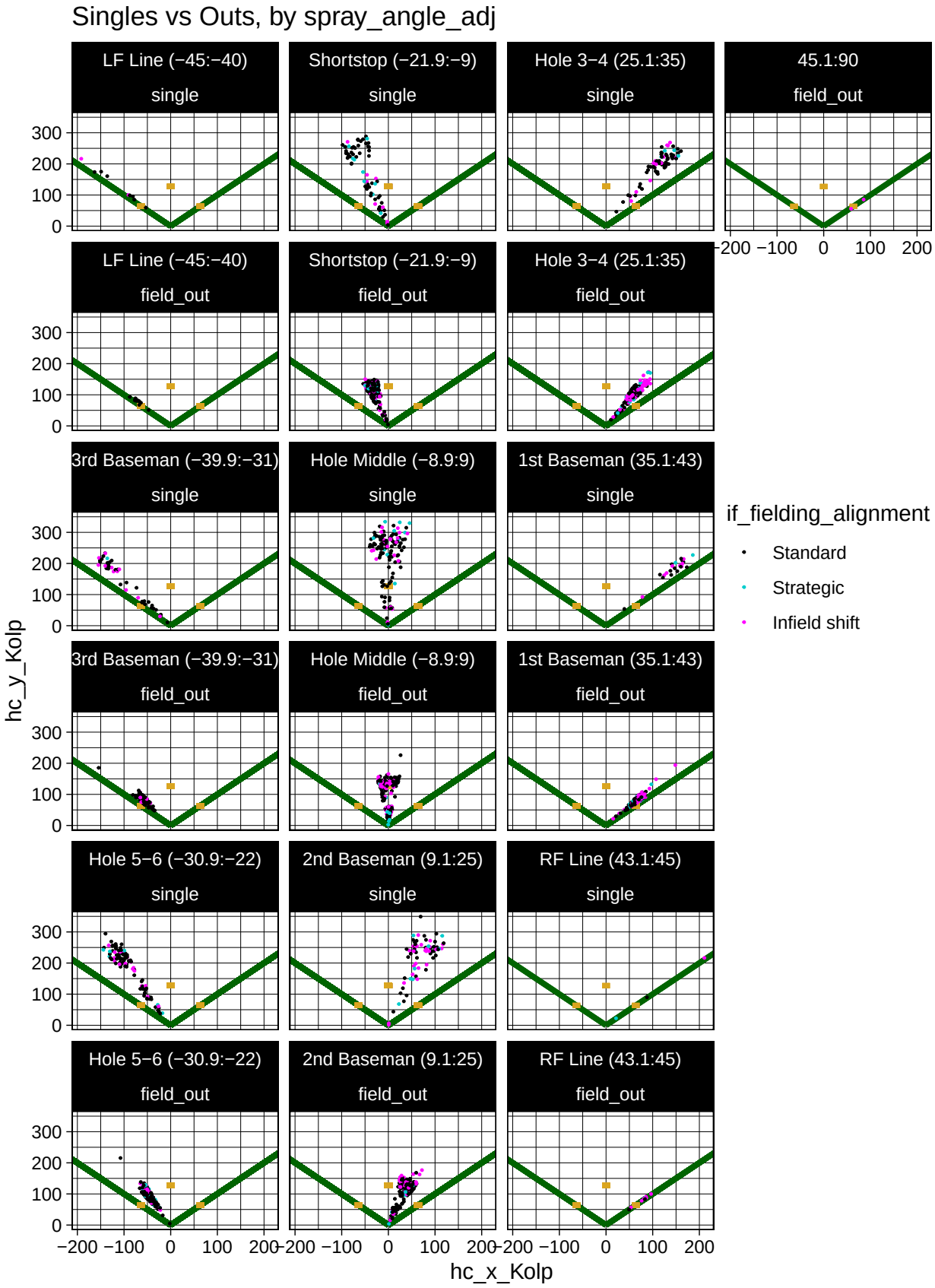
Fortunately, *spray_angle_adj* does contain this information.

```
# Visualize singles and outs, by *spray_angle_adj*

spray_angle_adj_plot <- ggplot(battedballs_gb_sample,
                               aes(hc_x_Kolp, hc_y_Kolp, color = if_fielding_alignment)) +
  scale_color_manual(values = c(Standard = "black", Strategic = "cyan3",
                                `Infield shift` = "magenta1")) +
  geom_segment(x = 0, y = 0, xend = 250, yend = 250, color = "dark green", size = 1.3) +
  geom_segment(x = 0, y = 0, xend = -250, yend = 250, color = "dark green", size = 1.3) +
  geom_tile(aes(x = 63.64, y = 63.64, width = 15, height = 15), color = "goldenrod",
            fill = "goldenrod") +
  geom_tile(aes(x = -63.64, y = 63.64, width = 15, height = 15), color = "goldenrod",
            fill = "goldenrod") +
  geom_tile(aes(x = 0, y = 127.28, width = 15, height = 15), color = "goldenrod",
```

```
            fill = "goldenrod") +
  geom_point(size = 0.2) + theme_linedraw() +
  labs(title = "Singles vs Outs, by spray_angle_adj") +
  facet_wrap(~ spray_angle_Kolp_cat + events, dir = "v", nrow = 6)
```

```
# gtable_show_names(spray_angle_adj_plot)
```

```
reposition_legend(spray_angle_adj_plot, position = 'center', panel = 'panel-1-6')
```


Singles vs Outs, by spray_angle_adj
LF Line (−45:−40)
single
Shortstop (−21.9:−9)
single
Hole 3−4 (25.1:35)
single
45.1:90
field_out
LF Line (−45:−40)
field_out
Shortstop (−21.9:−9)
field_out
Hole 3−4 (25.1:35)
field_out
3rd Baseman (−39.9:−31)
single
Hole Middle (−8.9:9)
single
1st Baseman (35.1:43)
single
3rd Baseman (−39.9:−31)
field_out
Hole Middle (−8.9:9)
field_out
1st Baseman (35.1:43)
field_out
Hole 5−6 (−30.9:−22)
single
2nd Baseman (9.1:25)
single
RF Line (43.1:45)
single
Hole 5−6 (−30.9:−22)
field_out
2nd Baseman (9.1:25)
field_out
RF Line (43.1:45)
field_out
hc_y_Kolp
hc_x_Kolp
if_fielding_alignment
Standard
Strategic
Infield shift

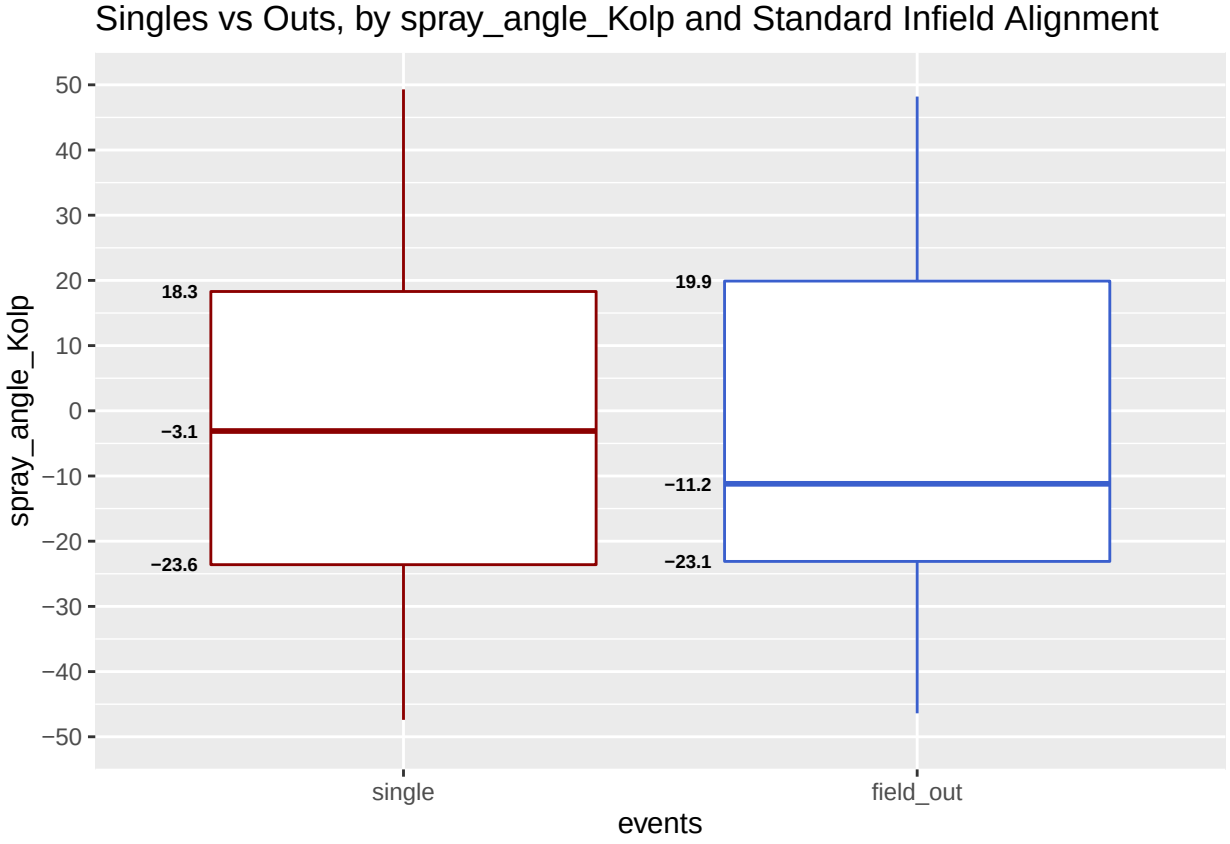
The plots of singles and outs by *spray_angle_adj* are necessarily structured quite differently than our *spray_angle_Kolb* plots. We still have the ranges of spray angles defined as they were earlier. But instead of progressing from the left side to the right side of the field, the panels progress from batted balls being pulled down either foul line to batted balls being hit toward the opposite field foul lines. Thus, you'll see two ranges of batted balls on each panel, one for a left-handed batter and one for a right-handed batter.

Recall that shifts deploy three infielders on the pull side of second base. The outs panels accordingly show (in magenta color) balls hit up the middle and into the pull side hole being turned into outs by shifted infielders. However, the singles panels still don't have enough data points to make any patterns visible. There's a nice collection of singles hit into the opposite side hole by left-handed hitters facing a shift. But that's about as much as we can see. At this point, the jury was still out on the usefulness of *if_fielding_alignment*.

We next created box plots showing the distribution of Spray Angle and Infield Alignment by singles and outs. The first and third quartiles (the 25th and 75th percentiles), along with the medians were added to the plots.

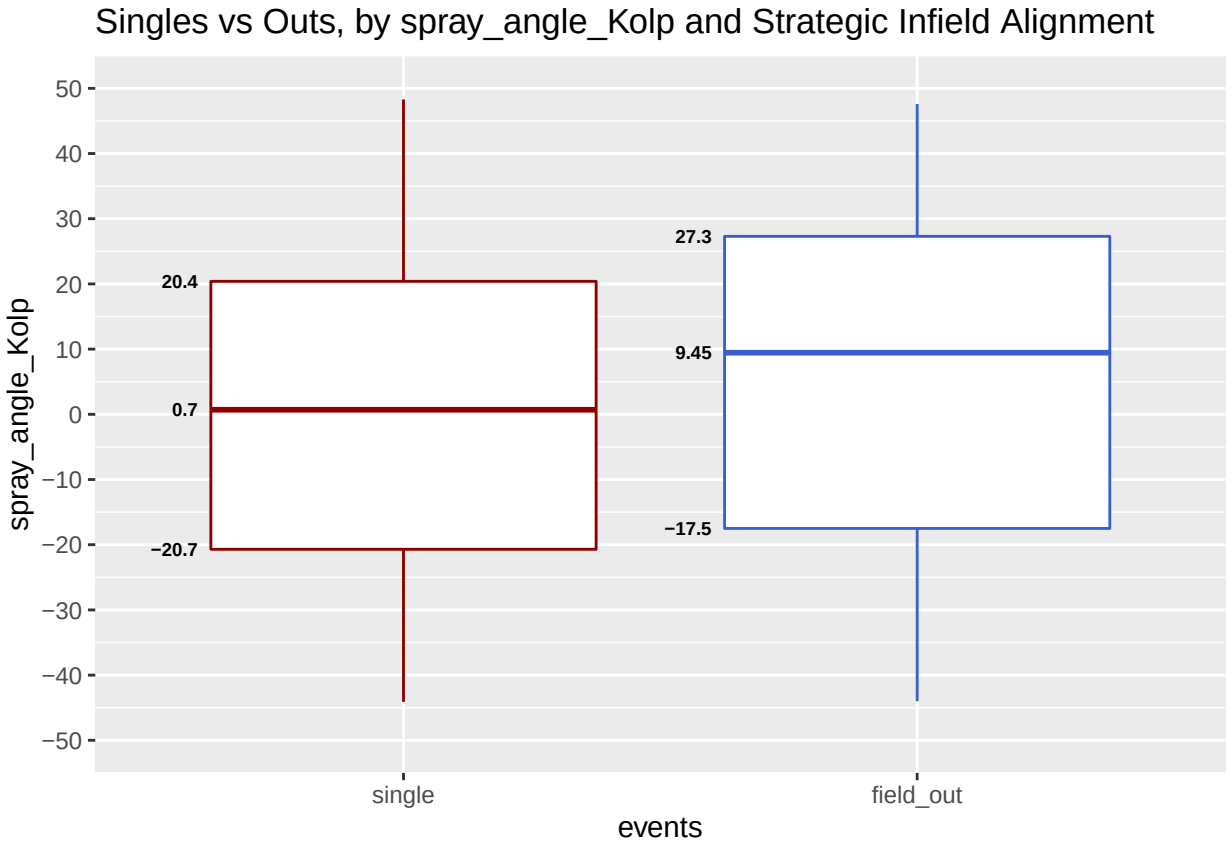
```
# Create boxplots to visualize the distribution of Spray Angle and Infield Alignment
# by singles and outs.

battedballs_gb <- battedballs %>% filter(launch_angle <= 10)
battedballs_gb_std <- battedballs %>% filter(launch_angle <= 10,
                                             if_fielding_alignment == "Standard")
battedballs_gb_strat <- battedballs %>% filter(launch_angle <= 10,
                                               if_fielding_alignment == "Strategic")
battedballs_gb_shift <- battedballs %>% filter(launch_angle <= 10,
                                               if_fielding_alignment == "Infield shift")

# Standard Infield Alignment
bb_outcome_by_spray_angle_Kolp_std <- ggplot(battedballs_gb_std,
                                     aes(x = events, y = spray_angle_Kolp,
                                         color = events)) +
  geom_boxplot() +
  scale_y_continuous(breaks = seq(-50, 50, by = 10),
                     labels = c("-50", "-40", "-30", "-20", "-10", "0", "10", "20", "30",
                                "40",  "50"),
                     limits = c(-50, 50)) +
  scale_color_manual(values = c("red4", "royalblue3")) +
  ggtitle("Singles vs Outs, by spray_angle_Kolp and Standard Infield Alignment") +
  theme(legend.position = "none")

bb_outcome_by_spray_angle_Kolp_std_data <- layer_data(bb_outcome_by_spray_angle_Kolp_std)

spray_angle_Kolp_single_std_1quartile <- bb_outcome_by_spray_angle_Kolp_std_data[1, 3]
spray_angle_Kolp_single_std_median <- bb_outcome_by_spray_angle_Kolp_std_data[1, 4]
spray_angle_Kolp_single_std_3quartile <- bb_outcome_by_spray_angle_Kolp_std_data[1, 5]
spray_angle_Kolp_out_std_1quartile <- bb_outcome_by_spray_angle_Kolp_std_data[2, 3]
spray_angle_Kolp_out_std_median <- bb_outcome_by_spray_angle_Kolp_std_data[2, 4]
spray_angle_Kolp_out_std_3quartile <- bb_outcome_by_spray_angle_Kolp_std_data[2, 5]

ggplot(battedballs_gb_std,
       aes(x = events, y = spray_angle_Kolp, color = events)) +
  geom_boxplot() +
  scale_y_continuous(breaks = seq(-50, 50, by = 10),
                     labels = c("-50", "-40", "-30", "-20", "-10", "0", "10", "20", "30",
                                "40",  "50"),
```

```
                    limits = c(-50, 50)) +
  scale_color_manual(values = c("red4", "royalblue3")) +
  ggtitle("Singles vs Outs, by spray_angle_Kolp and Standard Infield Alignment") +
  theme(legend.position = "none") +
  annotate("text", size = 2.5, fontface = 2, hjust = 1, x = c(.6, .6, .6),
           y = c(spray_angle_Kolp_single_std_1quartile,
                 spray_angle_Kolp_single_std_median,
                 spray_angle_Kolp_single_std_3quartile),
           label = c(spray_angle_Kolp_single_std_1quartile,
                     spray_angle_Kolp_single_std_median,
                     spray_angle_Kolp_single_std_3quartile)) +
  annotate("text", size = 2.5, fontface = 2, hjust = 1, x = c(1.6,1.6, 1.6),
           y = c(spray_angle_Kolp_out_std_1quartile, spray_angle_Kolp_out_std_median,
                 spray_angle_Kolp_out_std_3quartile),
           label = c(spray_angle_Kolp_out_std_1quartile, spray_angle_Kolp_out_std_median,
                     spray_angle_Kolp_out_std_3quartile))
```



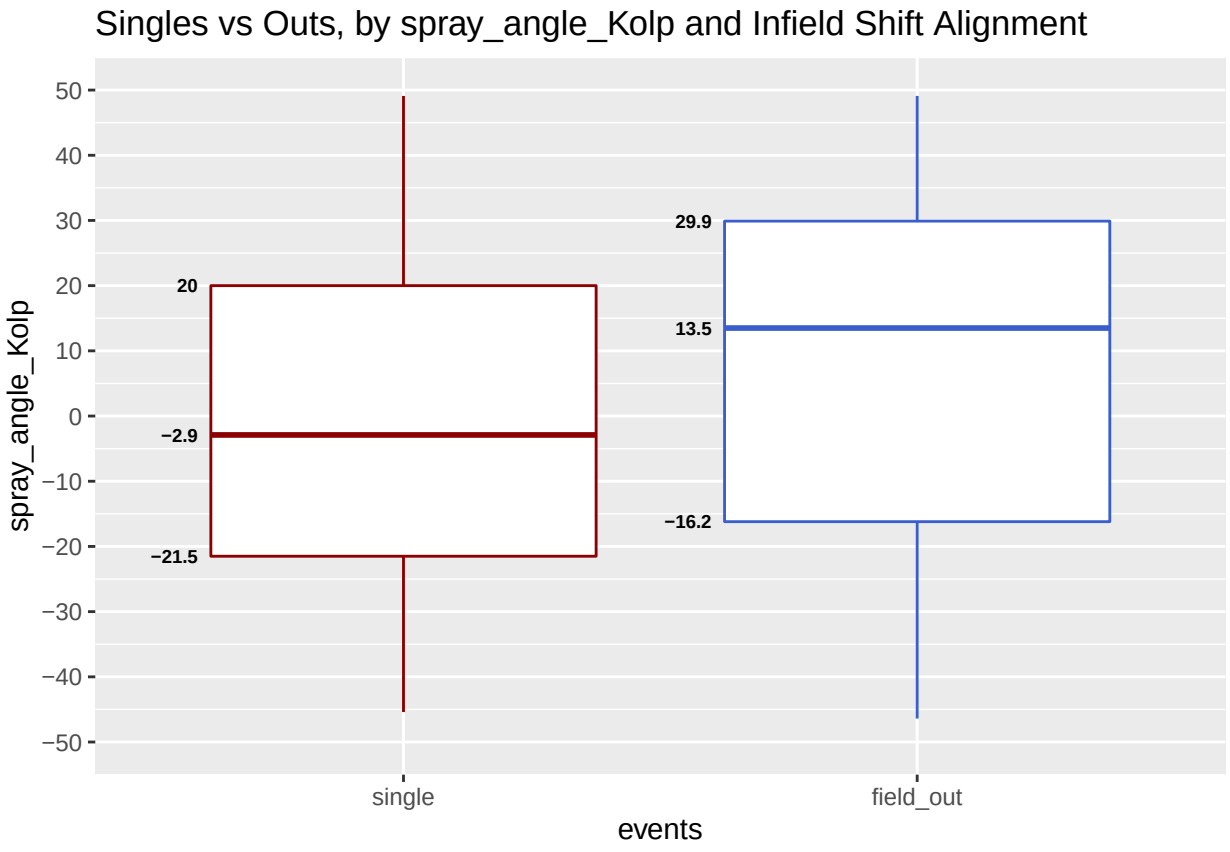

```
# Strategic Infield Alignment
bb_outcome_by_spray_angle_Kolp_strat <- ggplot(battedballs_gb_strat,
                                               aes(x = events, y = spray_angle_Kolp,
                                                   color = events)) +
  geom_boxplot() +
  scale_y_continuous(breaks = seq(-50, 50, by = 10),
                     labels = c("-50", "-40", "-30", "-20", "-10", "0", "10", "20", "30",
                                "40",  "50"),
```

```
                     limits = c(-50, 50)) +
  scale_color_manual(values = c("red4", "royalblue3")) +
  ggtitle("Singles vs Outs, by spray_angle_Kolp and Strategic Infield Alignment") +
  theme(legend.position = "none")

bb_outcome_by_spray_angle_Kolp_strat_data <-
  layer_data(bb_outcome_by_spray_angle_Kolp_strat)

spray_angle_Kolp_single_strat_1quartile <- bb_outcome_by_spray_angle_Kolp_strat_data[1, 3]
spray_angle_Kolp_single_strat_median <- bb_outcome_by_spray_angle_Kolp_strat_data[1, 4]
spray_angle_Kolp_single_strat_3quartile <- bb_outcome_by_spray_angle_Kolp_strat_data[1, 5]
spray_angle_Kolp_out_strat_1quartile <- bb_outcome_by_spray_angle_Kolp_strat_data[2, 3]
spray_angle_Kolp_out_strat_median <- bb_outcome_by_spray_angle_Kolp_strat_data[2, 4]
spray_angle_Kolp_out_strat_3quartile <- bb_outcome_by_spray_angle_Kolp_strat_data[2, 5]

ggplot(battedballs_gb_strat,
       aes(x = events, y = spray_angle_Kolp, color = events)) +
  geom_boxplot() +
  scale_y_continuous(breaks = seq(-50, 50, by = 10),
                     labels = c("-50", "-40", "-30", "-20", "-10", "0", "10", "20", "30",
                                "40",  "50"),
                     limits = c(-50, 50)) +
  scale_color_manual(values = c("red4", "royalblue3")) +
  ggtitle("Singles vs Outs, by spray_angle_Kolp and Strategic Infield Alignment") +
  theme(legend.position = "none") +
  annotate("text", size = 2.5, fontface = 2, hjust = 1, x = c(.6, .6, .6),
           y = c(spray_angle_Kolp_single_strat_1quartile,
                 spray_angle_Kolp_single_strat_median,
                 spray_angle_Kolp_single_strat_3quartile),
           label = c(spray_angle_Kolp_single_strat_1quartile,
                     spray_angle_Kolp_single_strat_median,
                     spray_angle_Kolp_single_strat_3quartile)) +
  annotate("text", size = 2.5, fontface = 2, hjust = 1, x = c(1.6,1.6, 1.6),
           y = c(spray_angle_Kolp_out_strat_1quartile,
                 spray_angle_Kolp_out_strat_median,
                 spray_angle_Kolp_out_strat_3quartile),
           label = c(spray_angle_Kolp_out_strat_1quartile,
                     spray_angle_Kolp_out_strat_median,
                     spray_angle_Kolp_out_strat_3quartile))
```




```r
# Infield Shift Alignment

bb_outcome_by_spray_angle_Kolp_shift <- ggplot(battedballs_gb_shift,
                                               aes(x = events, y = spray_angle_Kolp,
                                                   color = events)) +
  geom_boxplot() +
  scale_y_continuous(breaks = seq(-50, 50, by = 10),
                     labels = c("-50", "-40", "-30", "-20", "-10", "0", "10", "20", "30",
                                "40",  "50"),
                     limits = c(-50, 50)) +
  scale_color_manual(values = c("red4", "royalblue3")) +
  ggtitle("Singles vs Outs, by spray_angle_Kolp and Infield Shift Alignment") +
  theme(legend.position = "none")

bb_outcome_by_spray_angle_Kolp_shift_data <-
  layer_data(bb_outcome_by_spray_angle_Kolp_shift)

spray_angle_Kolp_single_shift_1quartile <- bb_outcome_by_spray_angle_Kolp_shift_data[1, 3]
spray_angle_Kolp_single_shift_median <- bb_outcome_by_spray_angle_Kolp_shift_data[1, 4]
spray_angle_Kolp_single_shift_3quartile <- bb_outcome_by_spray_angle_Kolp_shift_data[1, 5]
spray_angle_Kolp_out_shift_1quartile <- bb_outcome_by_spray_angle_Kolp_shift_data[2, 3]
spray_angle_Kolp_out_shift_median <- bb_outcome_by_spray_angle_Kolp_shift_data[2, 4]
spray_angle_Kolp_out_shift_3quartile <- bb_outcome_by_spray_angle_Kolp_shift_data[2, 5]

ggplot(battedballs_gb_shift,
       aes(x = events, y = spray_angle_Kolp, color = events)) +
```

```
  geom_boxplot() +
  scale_y_continuous(breaks = seq(-50, 50, by = 10),
                     labels = c("-50", "-40", "-30", "-20", "-10", "0", "10", "20", "30",
                                "40",  "50"),
                     limits = c(-50, 50)) +
  scale_color_manual(values = c("red4", "royalblue3")) +
  ggtitle("Singles vs Outs, by spray_angle_Kolp and Infield Shift Alignment") +
  theme(legend.position = "none") +
  annotate("text", size = 2.5, fontface = 2, hjust = 1, x = c(.6, .6, .6),
           y = c(spray_angle_Kolp_single_shift_1quartile,
                 spray_angle_Kolp_single_shift_median,
                 spray_angle_Kolp_single_shift_3quartile),
           label = c(spray_angle_Kolp_single_shift_1quartile,
                     spray_angle_Kolp_single_shift_median,
                     spray_angle_Kolp_single_shift_3quartile)) +
  annotate("text", size = 2.5, fontface = 2, hjust = 1, x = c(1.6,1.6, 1.6),
           y = c(spray_angle_Kolp_out_shift_1quartile,
                 spray_angle_Kolp_out_shift_median,
                 spray_angle_Kolp_out_shift_3quartile),
           label = c(spray_angle_Kolp_out_shift_1quartile,
                     spray_angle_Kolp_out_shift_median,
                     spray_angle_Kolp_out_shift_3quartile))
```



Our *spray_angle_Kolp* box plots produced clearer results once *if_fielding_alignment* was taken into account. When infielders were in Standard alignment, the median spray angle was significantly lower for outs (-11.2 degrees) than singles (-3.1 degrees). That is, outs were more frequent when the ball was hit toward the

shortstop on the left side of the infield. Singles were more frequent when hit up the middle. So far, so good.

When infielders were in a non-standard alignment (Strategic or Shift), the entire interquartile range of spray angles was significantly higher for outs than for singles. The difference was particularly pronounced when the infield was fully shifted; the medians were 16.4 degrees apart (13.5 degrees as compared to -2.9)! So outs were much more frequent on balls hit to the right side of the infield, which would be into the shift for left-handed batters (we already know that the vast majority of shifts are deployed against left-handed batters). Singles, again, were more frequent when hit up the middle.

Note that the interquartile ranges for singles barely changed across infield alignments, but the interquartile ranges for outs changed markedly. This likely explains why our earlier plots of batted ball coordinates depicted recognizable spray angle and infield alignment patterns for outs, but not singles.

We next computed the proportion of batted balls resulting in a single for each *spray_angle_Kolp* category. Again, singles comprise about 25% of our entire *battedballs* dataset, so that's our baseline.

```
# Compute the proportion of singles for each *spray_angle_Kolp_cat* and Infield
# Fielding Alignment.

(sa_std_prop_LFline <- (battedballs_gb_std %>%
                          filter(spray_angle_Kolp_cat == "LF Line (-45:-40)" &
                                   events == "single") %>% summarize(n = n())) /
  (battedballs_gb_std %>% filter(spray_angle_Kolp_cat == "LF Line (-45:-40)") %>%
     summarize(n = n())))
```

```
##           n
## 1 0.2821918
```

```
(sa_std_prop_3b <- (battedballs_gb_std %>%
                      filter(spray_angle_Kolp_cat == "3rd Baseman (-39.9:-31)" &
                               events == "single") %>% summarize(n = n())) /
  (battedballs_gb_std %>% filter(spray_angle_Kolp_cat == "3rd Baseman (-39.9:-31)") %>%
     summarize(n = n())))
```

```
##           n
## 1 0.1814528
```

```
(sa_std_prop_hole56 <- (battedballs_gb_std %>%
                          filter(spray_angle_Kolp_cat == "Hole 5-6 (-30.9:-22)" &
                                   events == "single") %>% summarize(n = n())) /
  (battedballs_gb_std %>% filter(spray_angle_Kolp_cat == "Hole 5-6 (-30.9:-22)") %>%
     summarize(n = n())))
```

```
##           n
## 1 0.3228307
```

```
(sa_std_prop_ss <- (battedballs_gb_std %>%
                      filter(spray_angle_Kolp_cat == "Shortstop (-21.9:-9)" &
                               events == "single") %>% summarize(n = n())) /
  (battedballs_gb_std %>% filter(spray_angle_Kolp_cat == "Shortstop (-21.9:-9)") %>%
     summarize(n = n())))
```

```
##           n
## 1 0.1495143
```

```
(sa_std_prop_holeMid <- (battedballs_gb_std %>%
                           filter(spray_angle_Kolp_cat == "Hole Middle (-8.9:9)" &
                                    events == "single") %>% summarize(n = n())) /
  (battedballs_gb_std %>% filter(spray_angle_Kolp_cat == "Hole Middle (-8.9:9)") %>%
     summarize(n = n())))
```

```
##          n
## 1 0.467631
```

```
(sa_std_prop_2b <- (battedballs_gb_std %>%
                      filter(spray_angle_Kolp_cat == "2nd Baseman (9.1:25)" &
                               events == "single") %>% summarize(n = n())) /
  (battedballs_gb_std %>% filter(spray_angle_Kolp_cat == "2nd Baseman (9.1:25)") %>%
     summarize(n = n())))
```

```
##           n
## 1 0.1710062
```

```
(sa_std_prop_hole34 <- (battedballs_gb_std %>%
                          filter(spray_angle_Kolp_cat == "Hole 3-4 (25.1:35)" &
                                   events == "single") %>% summarize(n = n())) /
  (battedballs_gb_std %>% filter(spray_angle_Kolp_cat == "Hole 3-4 (25.1:35)") %>%
     summarize(n = n())))
```

```
##           n
## 1 0.3420378
```

```
(sa_std_prop_1b <- (battedballs_gb_std %>%
                      filter(spray_angle_Kolp_cat == "1st Baseman (35.1:43)" &
                               events == "single") %>% summarize(n = n())) /
  (battedballs_gb_std %>% filter(spray_angle_Kolp_cat == "1st Baseman (35.1:43)") %>%
     summarize(n = n())))
```

```
##          n
## 1 0.161244
```

```
(sa_std_prop_RFline <- (battedballs_gb_std %>%
                          filter(spray_angle_Kolp_cat == "RF Line (43.1:45)" &
                                   events == "single") %>% summarize(n = n())) /
  (battedballs_gb_std %>% filter(spray_angle_Kolp_cat == "RF Line (43.1:45)") %>%
     summarize(n = n())))
```

```
##           n
## 1 0.2105263
```

The computed proportions of singles given a Standard infield alignment largely follow the pattern we would expect to see if spray angles did indeed help distinguish singles from outs. Proportions are lowest when balls are hit at spray angles projecting toward the traditional location of an infielder; they're highest when balls are hit at spray angles extending toward traditionally located gaps between infielders (hole56, hole34, and especially up the middle). The exception is near first base, where we have a fairly high 26% proportion of

singles toward the first baseman, and only a 13% proportion down the right field foul line. This is likely the result of placing the first baseman's location too far away from the right field foul line. First basemen, unlike third basemen, often have to stay close to first base to minimize the leadoff of a baserunner. Regardless of this misdrawn boundary, the variability in the proportions shows that **spray angle relative to infielders' locations** has a role to play in our predictive models. However, it will likely be a less significant role than that of Launch Angle and Exit Velocity simply because we have defined the scope of our spray angle predictor such that it applies only to ground balls, which comprise a little more than half of our *battedballs* dataset.

We now wanted to see what the proportions of singles looked like across spray angles when the infield alignment was fully shifted. This time, we used *spray_angle_adj* because when the infielders are shifted, knowing whether the batter is left-handed or right-handed aids our understanding of the results.

```
# Compute the proportion of singles for each *spray_angle_adj_cat* with an Infield Shift
# Alignment.

(sa_shift_prop_LFline <- (battedballs_gb_shift %>%
                            filter(spray_angle_adj_cat == "Pulled Down Line (-45:-40)" &
                                     events == "single") %>% summarize(n = n())) /
    (battedballs_gb_shift %>%
       filter(spray_angle_adj_cat == "Pulled Down Line (-45:-40)") %>%
       summarize(n = n())))
```

```
##            n
## 1 0.07430341
```

```
(sa_shift_prop_3b <- (battedballs_gb_shift %>%
                        filter(spray_angle_adj_cat == "Pulled Corner Inf (-39.9:-31)" &
                                 events == "single") %>% summarize(n = n())) /
    (battedballs_gb_shift %>%
       filter(spray_angle_adj_cat == "Pulled Corner Inf (-39.9:-31)") %>%
       summarize(n = n())))
```

```
##           n
## 1 0.1343658
```

```
(sa_shift_prop_hole56 <- (battedballs_gb_shift %>%
                            filter(spray_angle_adj_cat == "Pulled Side Hole (-30.9:-22)" &
                                     events == "single") %>% summarize(n = n())) /
    (battedballs_gb_shift %>%
       filter(spray_angle_adj_cat == "Pulled Side Hole (-30.9:-22)") %>%
       summarize(n = n())))
```

```
##           n
## 1 0.1321696
```

```
(sa_shift_prop_ss <- (battedballs_gb_shift %>%
                        filter(spray_angle_adj_cat == "Pulled Mid Inf (-21.9:-9)" &
                                 events == "single") %>% summarize(n = n())) /
    (battedballs_gb_shift %>%
       filter(spray_angle_adj_cat == "Pulled Mid Inf (-21.9:-9)") %>%
       summarize(n = n())))
```

```
##           n
## 1 0.1646972
```

```
(sa_shift_prop_holeMid <- (battedballs_gb_shift %>%
                             filter(spray_angle_adj_cat == "Middle Hole (-8.9:9)" &
                                      events == "single") %>% summarize(n = n())) /
    (battedballs_gb_shift %>%
       filter(spray_angle_adj_cat == "Middle Hole (-8.9:9)") %>%
       summarize(n = n())))
```

```
##           n
## 1 0.3205989
```

```
(sa_shift_prop_2b <- (battedballs_gb_shift %>%
                        filter(spray_angle_adj_cat == "Oppo Mid Inf (9.1:25)" &
                                 events == "single") %>% summarize(n = n())) /
    (battedballs_gb_shift %>%
       filter(spray_angle_adj_cat == "Oppo Mid Inf (9.1:25)") %>%
       summarize(n = n())))
```

```
##           n
## 1 0.4870849
```

```
(sa_shift_prop_hole34 <- (battedballs_gb_shift %>%
                            filter(spray_angle_adj_cat == "Oppo Side Hole (25.1:35)" &
                                     events == "single") %>% summarize(n = n())) /
    (battedballs_gb_shift %>%
       filter(spray_angle_adj_cat == "Oppo Side Hole (25.1:35)") %>%
       summarize(n = n())))
```

```
##           n
## 1 0.5823755
```

```
(sa_shift_prop_1b <- (battedballs_gb_shift %>%
                        filter(spray_angle_adj_cat == "Oppo Corner Inf (35.1:43)" &
                                 events == "single") %>% summarize(n = n())) /
    (battedballs_gb_shift %>%
       filter(spray_angle_adj_cat == "Oppo Corner Inf (35.1:43)") %>%
       summarize(n = n())))
```

```
##           n
## 1 0.5228758
```

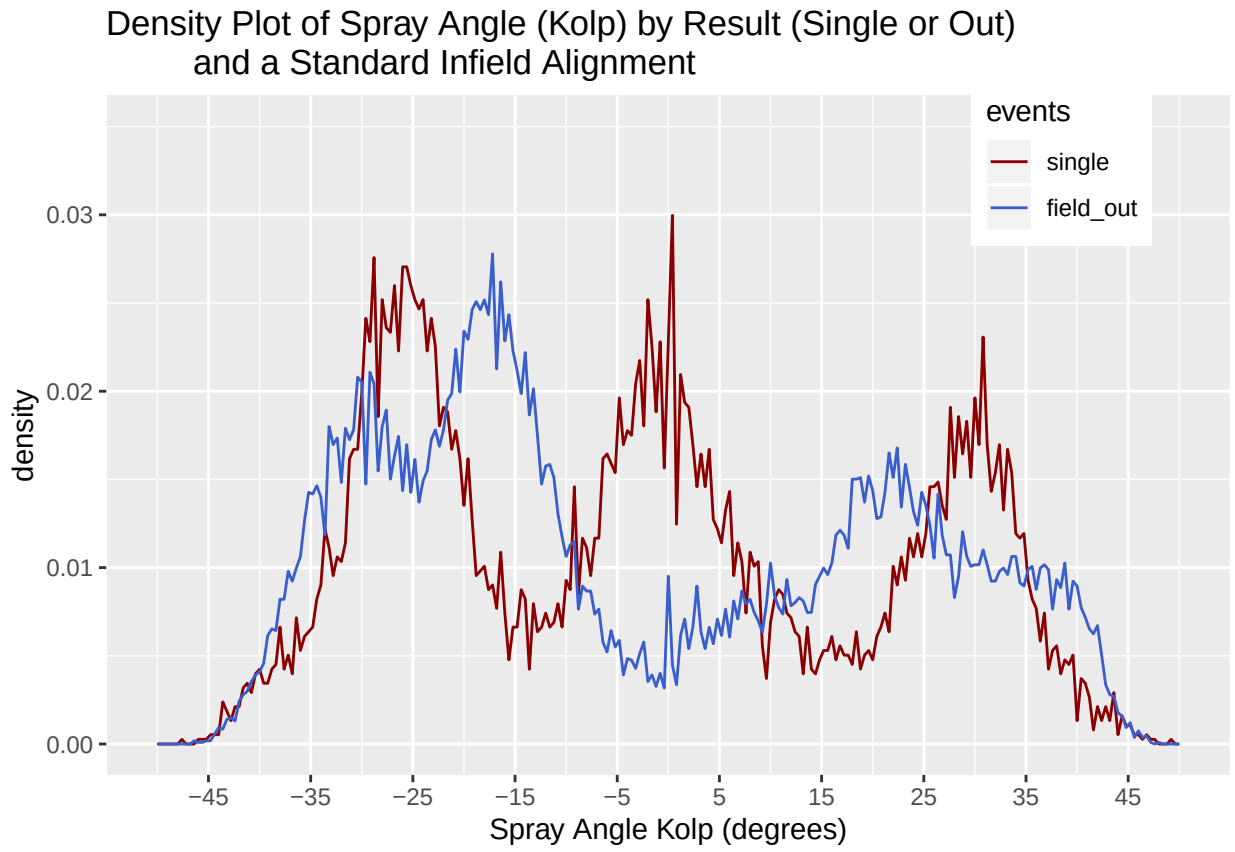
```
(sa_shift_prop_RFline <- (battedballs_gb_shift
                          %>% filter(spray_angle_adj_cat == "Oppo Down Line (43.1:45)" &
                                       events == "single") %>% summarize(n = n())) /
    (battedballs_gb_shift %>%
       filter(spray_angle_adj_cat == "Oppo Down Line (43.1:45)") %>%
       summarize(n = n())))
```

```
##           n
## 1 0.7142857
```

When a batter pulls the ball against the shift, the singles proportions never rise above 17% across the various pull spray angle ranges. Then we see a dramatic increase beginning with balls hit up the middle (32%) and progressively rising until we reach the opposite field foul line (64%). A shifted infield leaves few if any gaps on the pull side of the field, but huge gaps if the batter can manage to hit the ball to the opposite side. These numbers make credible our supposition that *if_fielding_alignment* adds useful information to our Spray Angle predictors.

Finally, we turned to the density plot to present a more comprehensive picture of singles and outs proportions without having to rely on arbitrarily constructed spray angle categories. The density plot we constructed showed the proportion of singles and outs at each Spray Angle and infield alignment.

```
# Plot the density of *spray_angle_Kolp* by result (single or out) and a Standard infield
# alignment.

x_scale <- scale_x_continuous(breaks = seq(-45, 45, by = 10),
                              labels = c("-45", "-35", "-25", "-15", "-5", "5", "15",
                                         "25", "35", "45"),
                              limits = c(-50, 50))
y_scale <- scale_y_continuous(limits = c(0, 0.035))
(sa_Kolp_density <- ggplot(battedballs_gb_std, aes(x = spray_angle_Kolp,
                                                   y = stat(density), color = events)) +
    geom_freqpoly(binwidth = 0.4) +
    x_scale +
    y_scale +
    scale_color_manual(values = c(single = "red4", field_out = "royalblue3")) +
    labs(title = "Density Plot of Spray Angle (Kolp) by Result (Single or Out)
         and a Standard Infield Alignment", x = "Spray Angle Kolp (degrees)") +
    theme(legend.position = c(.85, .9)))
```




```
sa_Kolp_density_data <- layer_data(sa_Kolp_density)
sa_Kolp_density_data <- filter(sa_Kolp_density_data, y != "NA", x != "NA")
```

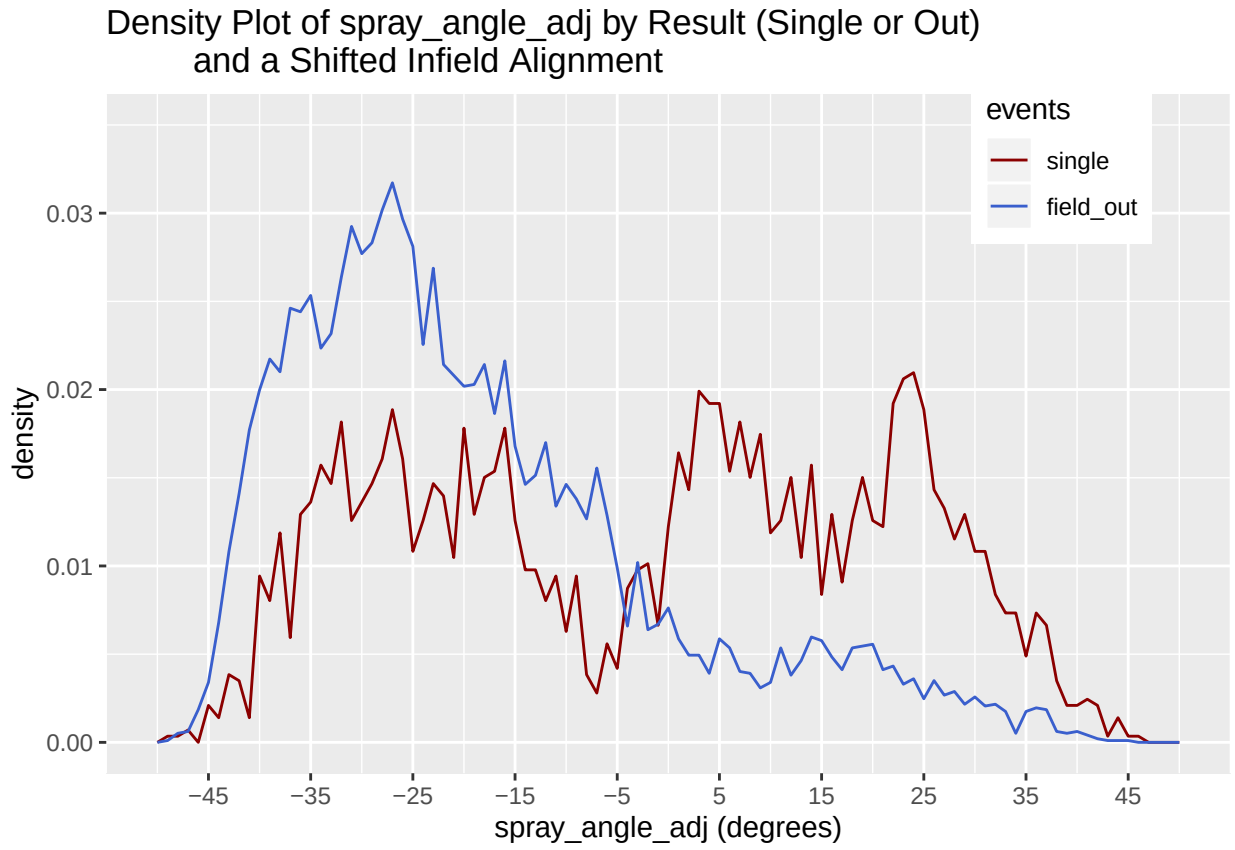
The density plot of *spray_angle_Kolp* with a Standard infield alignment quashes much of our uncertainty surrounding the locations of infield gaps, that is, spray angles at which ground balls are more likely to result in singles because they project toward locations between the infielders. We can state with more certainty now that the singles gaps (in degrees) are located at -30 to -23 (between the 3rd baseman and shortstop), -8 to 8 (up the middle, between the shortstop and second baseman), and 27 to 34 (between the second baseman and the first baseman).

The density plot reveals two other interesting aspects of spray angles. One is that there really are no singles gaps down the foul lines. When balls do get between the corner infielders and the foul lines, they are generally going to result in extra base hits, not singles. The other revelation is that some gaps are better than others for hitting singles. Specifically, the gap between the second baseman and the first baseman is not as good for singles as the gap between the 3rd baseman and shortstop. And the gap between the 3rd baseman and shortstop is not as good for singles as the gap up the middle. So we again see that the frequency of singles diminishes on the right side of the field because it's closer to first base. An infielder on the right side doesn't always have to field the ball cleanly to get the batter out at first base. If he just knocks the ball down, he's often able to recover and throw the batter out at first in time.

We now had to see if we could learn as much from a density plot of *spray_angle_adj* with a shifted infield.

```
# Plot the density of *spray_angle_adj* by result (single or out) and a shifted infield
# alignment.

x_scale <- scale_x_continuous(breaks = seq(-45, 45, by = 10),
```

```
                              labels = c("-45", "-35", "-25", "-15", "-5", "5", "15",
                                         "25", "35", "45"),
                              limits = c(-50, 50))
y_scale <- scale_y_continuous(limits = c(0, 0.035))
(sa_adj_density <- ggplot(battedballs_gb_shift, aes(x = spray_angle_adj,
                                                    y = stat(density), color = events)) +
    geom_freqpoly(binwidth = 1) +
    x_scale +
    y_scale +
    scale_color_manual(values = c(single = "red4", field_out = "royalblue3")) +
    labs(title = "Density Plot of spray_angle_adj by Result (Single or Out)
         and a Shifted Infield Alignment", x = "spray_angle_adj (degrees)") +
    theme(legend.position = c(.85, .9)))
```




```
sa_adj_density_data <- layer_data(sa_adj_density)
sa_adj_density_data <- filter(sa_adj_density_data, y != "NA", x != "NA")
```

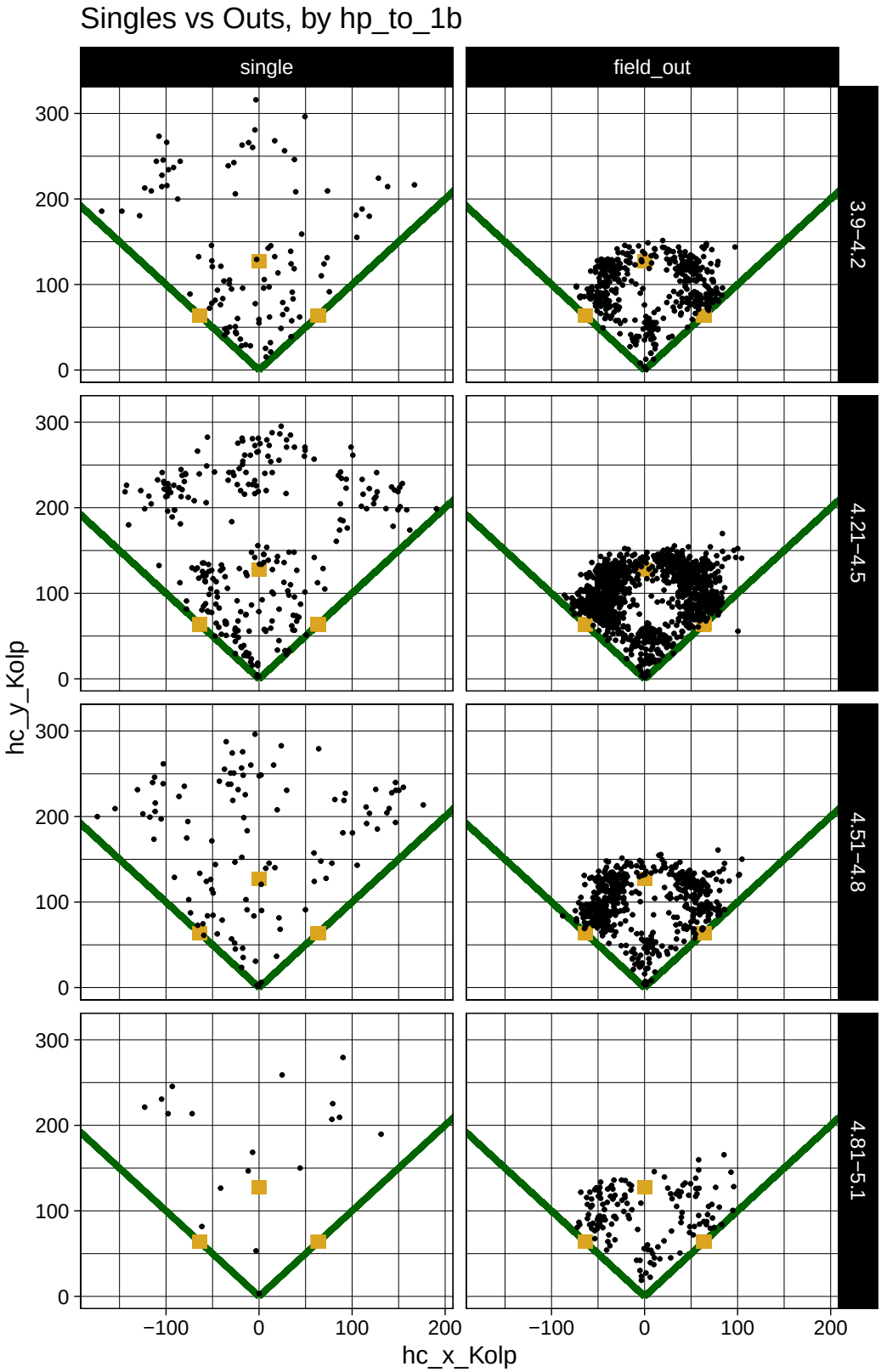
Here we see once again that when the infield is shifted, there really is no good spray angle for singles on the pull side of the field. The best sliver of hope is perhaps between -20 and -15 degrees, where the middle infielder is normally positioned. The opposite side of the field is mostly undefended, except where the middle infielder on that side of the field normally plays.

Home to First

Like Spray Angle, our last key predictor to explore, *home_to_1b*, is only going to affect the likelihood of a single on certain batted ball events. In this case, our population of batted ball events is restricted to topped and weakly hit ground balls. Topped and weak hits are two levels of an MLB metric called launch_speed_angle, which attempts to characterize the quality of a batter's contact based on the combination of Launch Angle and Exit Velocity (https://baseballsavant.mlb.com/csv-docs#launch_speed_angle). MLB considers the batter's time to first base to matter most when balls are topped or weakly hit. So we opted to use MLB's metric, with the addition of requiring the batted ball type to be a ground ball.

```
# Create a subset of *battedballs* consisting only of Topped and Weakly Hit Ground Balls.

battedballs_lowgb_topweak <- battedballs %>% filter(bb_type == "ground_ball" &
                                                      adv_bb_type == "low_ground_ball")
battedballs_lowgb_topweak <- battedballs_lowgb_topweak %>%
  filter(launch_speed_angle == "1" | launch_speed_angle == "2")

set.seed(1)
battedballs_lowgb_topweak_sample <- sample_n(battedballs_lowgb_topweak, 3000,
                                             replace = FALSE)
```

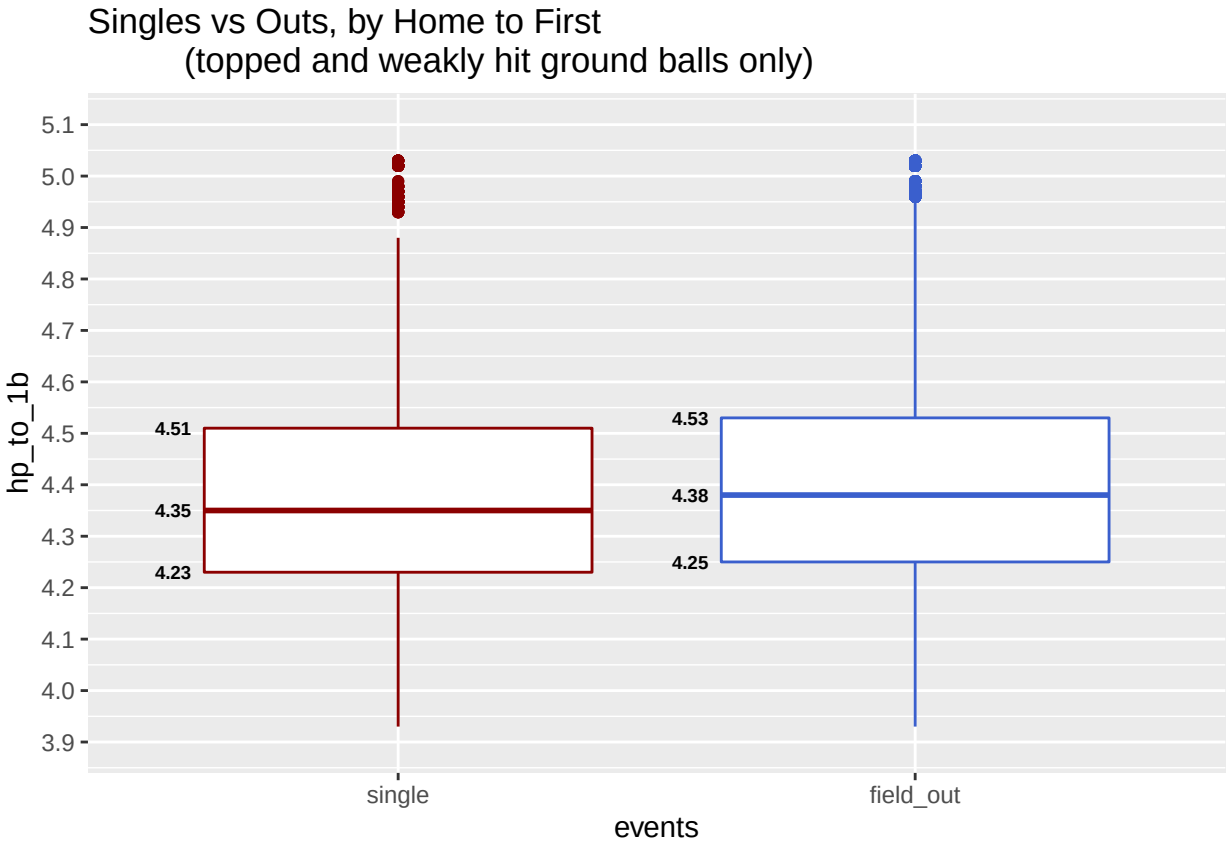
```
# Visualize singles and outs, by Home to First predictor (topped and weakly hit ground
# balls only).          .

ggplot(battedballs_lowgb_topweak_sample, aes(hc_x_Kolp, hc_y_Kolp)) +
  geom_segment(x = 0, y = 0, xend = 250, yend = 250, color = "dark green", size = 1.3) +
  geom_segment(x = 0, y = 0, xend = -250, yend = 250, color = "dark green", size = 1.3) +
  geom_tile(aes(x = 63.64, y = 63.64, width = 15, height = 15), color = "goldenrod",
            fill = "goldenrod") +
  geom_tile(aes(x = -63.64, y = 63.64, width = 15, height = 15), color = "goldenrod",
            fill = "goldenrod") +
  geom_tile(aes(x = 0, y = 127.28, width = 15, height = 15), color = "goldenrod",
            fill = "goldenrod") +
  geom_point(size = 0.5) + theme_linedraw() +
  labs(title = "Singles vs Outs, by hp_to_1b") +
  facet_grid(hp_to_1b_cat ~ events)
```



It's difficult to discern from these plots whether batters' Home to First measurements affect the likelihood of a single. The number of observations differs from row to row of the facet grid, and the proportions are

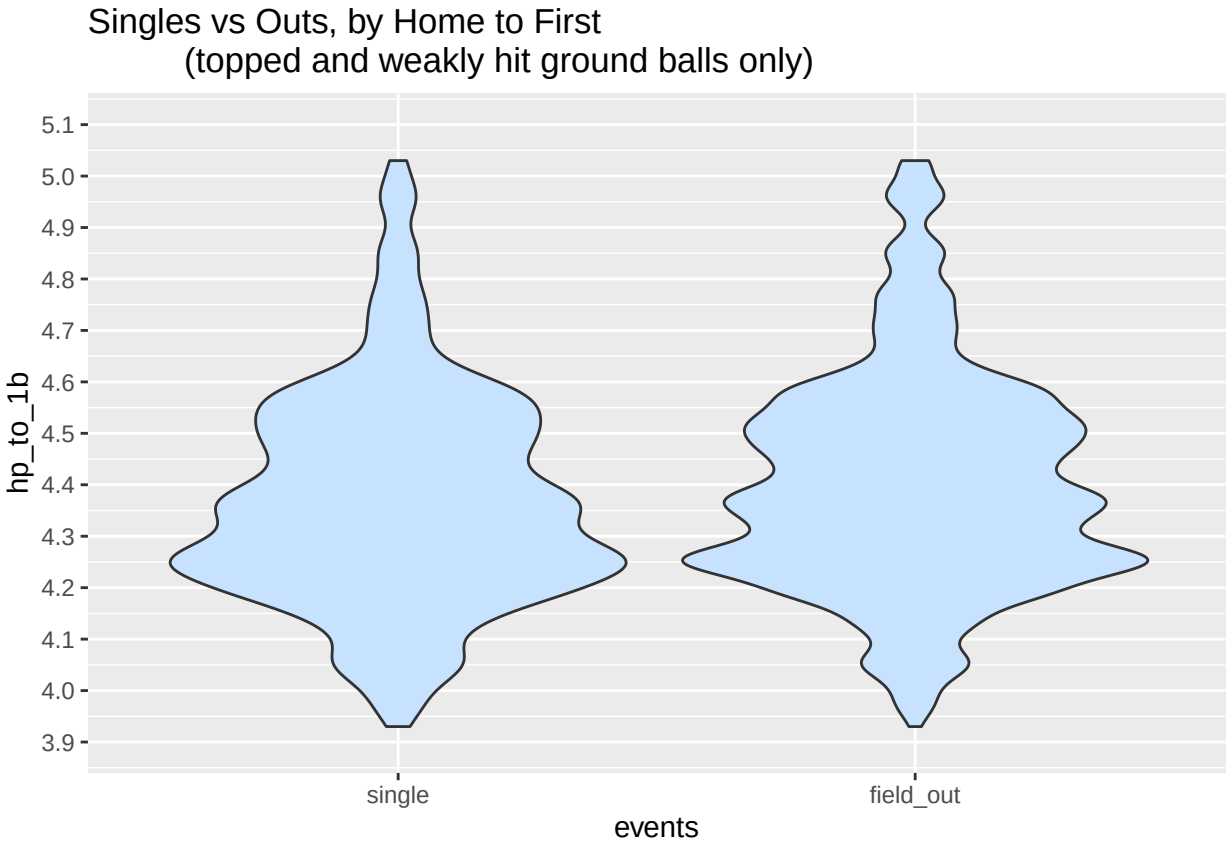
hard to gauge. But in the second row (4.21-4.5), where the observations are most plentiful, it's worth noting a denser cluster of infield singles extending down the third base line, as compared to the first base line. So that theme continues here.

```
# Create boxplots to visualize the distribution of hp_to_1b by singles and outs.

bb_outcome_by_hp_to_1b <- ggplot(battedballs_lowgb_topweak,
                                 aes(x = events, y = hp_to_1b, color = events)) +
  geom_boxplot() +
  scale_y_continuous(breaks = seq(3.9, 5.1, by = 0.1),
                     labels = c("3.9", "4.0", "4.1", "4.2", "4.3", "4.4", "4.5",  "4.6",
                                "4.7", "4.8", "4.9", "5.0", "5.1"),
                     limits = c(3.9, 5.1)) +
  scale_color_manual(values = c("red4", "royalblue3")) +
  ggtitle("Singles vs Outs, by Home to First
          (topped and weakly hit ground balls only)") +
  theme(legend.position = "none")

bb_outcome_by_hp_to_1b_data <- layer_data(bb_outcome_by_hp_to_1b)

hp_to_1b_single_1quartile <- bb_outcome_by_hp_to_1b_data[1, 3]
hp_to_1b_single_median <- bb_outcome_by_hp_to_1b_data[1, 4]
hp_to_1b_single_3quartile <- bb_outcome_by_hp_to_1b_data[1, 5]
hp_to_1b_out_1quartile <- bb_outcome_by_hp_to_1b_data[2, 3]
hp_to_1b_out_median <- bb_outcome_by_hp_to_1b_data[2, 4]
hp_to_1b_out_3quartile <- bb_outcome_by_hp_to_1b_data[2, 5]

ggplot(battedballs_lowgb_topweak,
       aes(x = events, y = hp_to_1b, color = events)) +
  geom_boxplot() +
  scale_y_continuous(breaks = seq(3.9, 5.1, by = 0.1),
                     labels = c("3.9", "4.0", "4.1", "4.2", "4.3", "4.4", "4.5",  "4.6",
                                "4.7", "4.8", "4.9", "5.0", "5.1"),
                     limits = c(3.9, 5.1)) +
  scale_color_manual(values = c("red4", "royalblue3")) +
  ggtitle("Singles vs Outs, by Home to First
          (topped and weakly hit ground balls only)") +
  theme(legend.position = "none") +
  annotate("text", size = 2.5, fontface = 2, hjust = 1, x = c(.6, .6, .6),
           y = c(hp_to_1b_single_1quartile, hp_to_1b_single_median,
                 hp_to_1b_single_3quartile),
           label = c(hp_to_1b_single_1quartile, hp_to_1b_single_median,
                     hp_to_1b_single_3quartile)) +
  annotate("text", size = 2.5, fontface = 2, hjust = 1, x = c(1.6,1.6, 1.6),
           y = c(hp_to_1b_out_1quartile, hp_to_1b_out_median, hp_to_1b_out_3quartile),
           label = c(hp_to_1b_out_1quartile, hp_to_1b_out_median, hp_to_1b_out_3quartile))
```



The boxplots aren't much more informative than our plots of batted ball coordinates. The Home to First stats barely differ between singles and outs. We decided to see whether a violin plot revealed more visual cues.

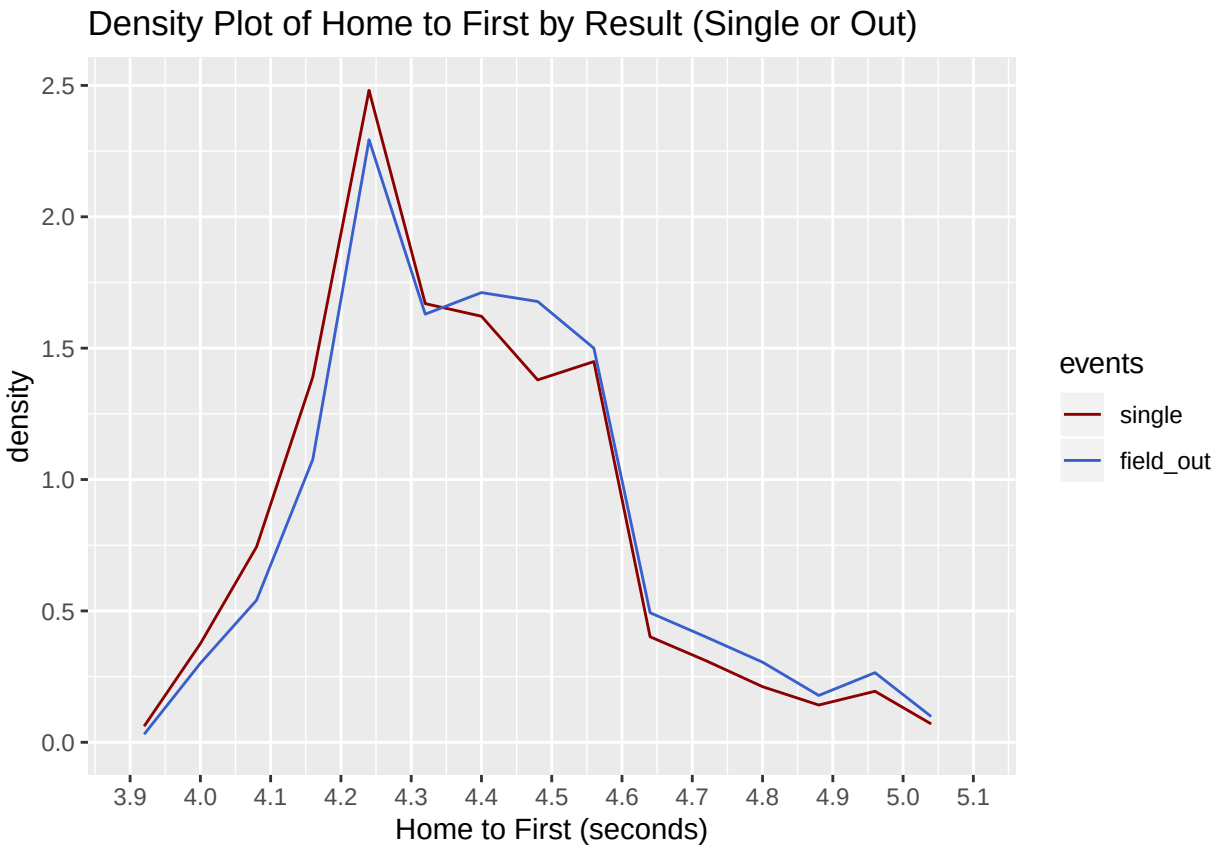
```
# Create violin plots to visualize the distribution of hp_to_1b by singles and outs.

ggplot(battedballs_lowgb_topweak,
       aes(x = events, y = hp_to_1b)) +
  geom_violin(fill = "slategray1") +
  scale_y_continuous(breaks = seq(3.9, 5.1, by = 0.1),
                     labels = c("3.9", "4.0", "4.1", "4.2", "4.3", "4.4", "4.5",  "4.6",
                                "4.7", "4.8", "4.9", "5.0", "5.1"),
                     limits = c(3.9, 5.1)) +
  ggtitle("Singles vs Outs, by Home to First
          (topped and weakly hit ground balls only)") +
  theme(legend.position = "none")
```



The answer to our question was "no." The distributions were very similar. The prospects for our *hp_to_1b* predictor were not looking good at this point.

Our last best hope for *hp_to_1b* was density plotting.

```
# Plot the density of our Home to First predictor (*hp_to_1b*) by result (single or out).

x_scale <- scale_x_continuous(breaks = seq(3.9, 5.1, by = 0.1),
                              labels = c("3.9", "4.0", "4.1", "4.2", "4.3", "4.4", "4.5",
                                         "4.6", "4.7", "4.8", "4.9", "5.0", "5.1"),
                              limits = c(3.9, 5.1))
y_scale <- scale_y_continuous()
(h1b_density <- ggplot(battedballs_lowgb_topweak, aes(x = hp_to_1b, y = stat(density),
                                                      color = events)) +
    geom_freqpoly(binwidth = .08) +
    x_scale +
    y_scale +
    scale_color_manual(values = c(single = "red4", field_out = "royalblue3")) +
    labs(title = "Density Plot of Home to First by Result (Single or Out)",
         x = "Home to First (seconds)"))
```




```
h1b_density_data <- layer_data(h1b_density)
h1b_density_data <- filter(h1b_density_data, y != "NA", x != "NA")
```

Here we could at last see that singles were slightly more dense than outs for batters who could reach first base in 4.3 seconds or less (about 35% of the players for whom we have Home to First data). That makes sense. But the density plots for singles and outs were very close to each other, perhaps because there are so few batted balls for which *hp_to_1b* differences of tenths of a second are really going to turn an out into a single or vice versa. In other words, when it comes to hitting singles, Home to First speed just doesn't matter except in relatively rare circumstances.

Prepare the Data for Use in Machine Learning Algorithms

Before we could begin to develop some models using different algorithms, we needed to rescale our data. Depending on the algorithm being used and the purpose for which its being used, if the distributions of the predictor variables are significantly different, rescaling mitigates these differences so that certain predictors don't dominate the algorithm simply by virtue of their large scale. Rescaling involves adding or subtracting a constant from each predictor value, and then multiplying or dividing by a constant.

Two primary methods are commonly used to rescale predictor values. The first, Normalization (sometimes referred to as Min-Max Scaling), involves subtracting the minimum value and then dividing by the range. This has the effect of making the minimum value of the predictor zero and the maximum value one. Normalization is considered most useful when using a distance-based algorithm (such as K-Nearest Neighbors) for classification purposes.

The other primary method is Standardization (sometimes referred to as Z-Score Normalization, just to confuse things). It involves subtracting a measure of location such as the mean, and then dividing by the

standard deviation. This technique transforms the predictor values so that their mean is zero with a standard deviation of one. Standardization is considered most useful when using linear regression, logistic regression, and linear discriminant analysis.

Here, we employed both Normalization and Standardization since we expected to be testing a variety of algorithms.

Also, since our outcome (single or out) is categorical rather than continuous, the *events* variable, which contained the outcome, was coded as a factor to facilitate the process of constructing a predictive machine learning algorithm or model.

```
# Make sure battedballs$events is coded as a factor with levels
# single, followed by field_out.

battedballs$events <- factor(battedballs$events, levels = c("single", "field_out"))

# Confirm which level of battedballs$events has been assigned to 0.

contrasts(as.factor(battedballs$events))
```

```
##           field_out
## single            0
## field_out         1
```

```
# Center the predictor variables to reduce non-essential multicollinearity among the
# predictors and improve the interpretation of their coefficients.

battedballs <- battedballs %>% mutate(c_launch_angle = launch_angle - mean(launch_angle))
battedballs <- battedballs %>% mutate(c_launch_speed = launch_speed - mean(launch_speed))
battedballs <- battedballs %>%
  mutate(c_spray_angle_Kolp = spray_angle_Kolp - mean(spray_angle_Kolp))
battedballs <- battedballs %>%
  mutate(c_spray_angle_adj = spray_angle_adj - mean(spray_angle_adj))
battedballs <- battedballs %>% mutate(c_hp_to_1b = hp_to_1b - mean(hp_to_1b))
battedballs <- battedballs %>%
  mutate(c_if_alignment = as.numeric(num_if_alignment) -
           mean(as.numeric(num_if_alignment)))

# Standardize the predictor variables.

battedballs <- battedballs %>% mutate(s_launch_angle = c_launch_angle / sd(launch_angle))
battedballs <- battedballs %>% mutate(s_launch_speed = c_launch_speed / sd(launch_speed))
battedballs <- battedballs %>%
  mutate(s_spray_angle_Kolp = c_spray_angle_Kolp / sd(spray_angle_Kolp))
battedballs <- battedballs %>%
  mutate(s_spray_angle_adj = c_spray_angle_adj / sd(spray_angle_adj))
battedballs <- battedballs %>% mutate(s_hp_to_1b = c_hp_to_1b / sd(hp_to_1b))
battedballs <- battedballs %>% mutate(s_if_alignment = c_if_alignment /
                                        sd(as.numeric(num_if_alignment)))

# Normalize the predictor variables.

battedballs <- battedballs %>%
  mutate(n_launch_angle = (launch_angle - min(launch_angle)) /
```

```
          (max(launch_angle) - min(launch_angle)))
battedballs <- battedballs %>%
  mutate(n_launch_speed = (launch_speed - min(launch_speed)) /
          (max(launch_speed) - min(launch_speed)))
battedballs <- battedballs %>%
  mutate(n_spray_angle_Kolp = (spray_angle_Kolp - min(spray_angle_Kolp)) /
          (max(spray_angle_Kolp) - min(spray_angle_Kolp)))
battedballs <- battedballs %>%
  mutate(n_spray_angle_adj = (spray_angle_adj - min(spray_angle_adj)) /
          (max(spray_angle_adj) - min(spray_angle_adj)))
battedballs <- battedballs %>%
  mutate(n_hp_to_1b = (hp_to_1b - min(hp_to_1b)) /
          (max(hp_to_1b) - min(hp_to_1b)))
battedballs <- battedballs %>%
  mutate(n_if_alignment = (num_if_alignment - min(num_if_alignment)) /
          (max(num_if_alignment) - min(num_if_alignment)))
```

We also needed to randomly partition our battedballs data frame into a training set and a test set. The training set would be used to train or construct the algorithm, and the test set would be used to determine how accurately the algorithm predicted the outcome (single or out) from an independent dataset. About 20 percent of batted balls was allocated to the newly created test_set tibble, while the remainder was used for the new train_set tibble.

```
# Partion battedballs.

set.seed(1)
test_index <- createDataPartition(y = battedballs$events, times = 1, p = 0.20,
                                  list = FALSE)
train_set <- battedballs[-test_index,]
test_set <- battedballs[test_index,]
```

Build and Assess a Baseline Model

Our first model used an algorithm that randomly guessed the outcomes in *test_set*. The predictive ability of this guessing algorithm could then be used as a baseline for comparison to subsequently developed models.

The crucial decision here was how we would measure the predictive ability of the guessing algorithm and others we would construct. Our project sought to determine whether a model could reliably distinguish between a single and an out. This presented a particular machine learning problem called prevalence. We had a classification problem with a dichotomous categorical response variable (*events*), and one of the response variable's two possible values (*field_out*) occurring far more frequently than the other. Specifically, 75% of the batted balls in our dataset were field outs, while only 25% were singles. Imbalanced outcome classes cause the threshold that best differentiates the outcomes to shift in favor of the majority class. Under these circumstances, methods for measuring predictive ability will often produce very misleading results, leading to the adoption of inferior models. Thus, our *training_set* could be considered biased.

We needed to use a method that took prevalence into account. Bowen Song, Guopeng Zhang, Wei Zhu & Zhengrong Liang have proposed three new assessment methods (shortest distance, harmonic mean and anti-harmonic mean) that specifically address the challenge of classifying imbalanced data. [7] The authors found the performance of their proposed methods to be "adaptive and robust with different levels of imbalanced data." We opted to use two of these methods, harmonic mean and shortest distance) for our project.

[7] Song, B., Zhang, G., Zhu, W. et al.

The harmonic mean method computes a harmonic average of sensitivity and specificity, which we labeled **truescore** because we care about correctly identifying both singles (measured by sensitivity or the true positive rate (TPR)) and outs (measured by specificity or the true negative rate (TNR)). According to the authors, the manner in which a harmonic mean is calculated means that "the larger the difference of the elements in the pair, the smaller the harmonic mean is." Thus, "it mitigate[s] the influence of the larger value and increase[s] the influence of the small value.... It pays more attention to the balance of the pair compared to the arithmetic mean." To the extent a model's decision point (i.e., selection point, cutoff or threshold) varied from the one that maximized the harmonic average of sensitivity and specificity, it was inferior.

When it made sense, we also applied a second method, shortest distance (or minimum distance), to evaluate the predictive ability of our models. A receiver operating characteristic (ROC) curve is a widely used plot for depicting the tradeoff between a model's sensitivity (TPR) and its false positive rate (FPR, which is 1 - specificity) for a given decision point. The model's possible decision points form the actual ROC curve. The ideal coordinate on the graph, where sensitivity and specificity are both maximized, is (0, 1). Hence, the point on the ROC curve closest to (0, 1) represents the model's optimal decision point. To the extent a model's decision point varies from the one closest to (0, 1), it is inferior. The authors provide an equation for computing minimum distance using sensitivity and specificity.

In the case of our baseline model, we computed a truescore even though it employed a guessing algorithm, and did not consider the data underlying our predictor variables.

```
# Develop a baseline model that guesses the outcomes in the test_set.

set.seed(1)
bl_preds <- sample(c("single", "field_out"), size = length(test_index),
                   replace = TRUE, prob = c(0.25, 0.75)) %>%
    factor(levels = levels(test_set$events))
(bl_confusionM <- confusionMatrix(data = bl_preds, reference = test_set$events))
```

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction  single field_out
##   single      1294      3775
##   field_out   3687     11362
##
##                Accuracy : 0.6291
##                  95% CI : (0.6224, 0.6358)
##     No Information Rate : 0.7524
##     P-Value [Acc > NIR] : 1.0000
##
##                   Kappa : 0.0103
##
##  Mcnemar's Test P-Value : 0.3139
##
##             Sensitivity : 0.25979
##             Specificity : 0.75061
##          Pos Pred Value : 0.25528
##          Neg Pred Value : 0.75500
##              Prevalence : 0.24759
##          Detection Rate : 0.06432
##    Detection Prevalence : 0.25196
##       Balanced Accuracy : 0.50520
##
```

```
##        'Positive' Class : single
##
```

```
table(predicted = bl_preds, actual = test_set$events)
```

```
##            actual
## predicted   single field_out
##   single      1294      3775
##   field_out   3687     11362
```

```
(bl_tpr <- round(sensitivity(bl_preds, reference = factor(test_set$events)), 6))
```

```
## [1] 0.259787
```

```
(bl_tnr <- round(specificity(bl_preds, reference = factor(test_set$events)), 6))
```

```
## [1] 0.750611
```

```
(bl_fpr <- round(1 - bl_tnr, 6))
```

```
## [1] 0.249389
```

```
(bl_truescore <- round((2 * bl_tpr * bl_tnr) / (bl_tpr + bl_tnr) ,6))
```

```
## [1] 0.385984
```

Single is the positive outcome. With a TPR of only .26 and a TNR of .75, our guessing algorithm ended up with a Truescore of .386. Other models we develop ought to do better.

So with a categorical outcome and six continuous numeric predictors (if we include *num_if_alignment*, Spray Angle's interactive partner), we needed to select an appropriate type of model, that is, one capable of classifying each batted ball observation as either a single or an out, based on its corresponding Launch Angle, Exit Velocity, Spray Angle (two versions), Infield Alignment, and Home to First measurements.

Build and Assess a Logistic Regression Model

Generalized linear models adapt the linear model framework to cover non-normal dependent variables, such as a categorical outcome. They assume that the outcome or response variable follows an exponential rather than linear distribution. And they estimate the parameters in an algorithm using maximum likelihood rather than least squares. The end result is an estimated probability that an observation (batted ball) is in a particular class (single or out) given (conditioned on) the values of one or more predictors.

The generalized linear model often used to predict a binary (0 or 1) outcome variable from a set of numeric variables is logistic regression. It assumes the outcomes (in this context, a probability distribution) follow a binomial distribution, and it transforms the conditional probabilities into log odds to ensure the probability estimates are always between 0 and 1. The log odds represent how much more likely something will happen than not happen. Thus, probabilities will be symmetric around 0. The accuracy of the model's predictions depend on how close its probability estimates are to the actual probabilities.

So we proceeded to fit a logistic regression model to the standardized data in *train_set*.

```r
# Confirm which level of train_set$events has been assigned to 0.

contrasts(as.factor(train_set$events))
```

```
##           field_out
## single            0
## field_out         1
```

```r
# Fit a model to train_set$events using logistic regression.

fit_logit <- train_set %>% mutate(y = as.numeric(events == "field_out")) %>%
  glm(y ~ s_launch_angle + s_launch_speed + s_spray_angle_Kolp + s_spray_angle_adj +
        s_hp_to_1b + s_if_alignment, data = ., family = "binomial")

# Make sure each regression coefficient is statistically significant (p < .05).

summary(fit_logit)
```

```
##
## Call:
## glm(formula = y ~ s_launch_angle + s_launch_speed + s_spray_angle_Kolp +
##     s_spray_angle_adj + s_hp_to_1b + s_if_alignment, family = "binomial",
##     data = .)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.5845   0.3907   0.6779   0.7933   1.3146
##
## Coefficients:
##                     Estimate Std. Error z value Pr(>|z|)
## (Intercept)         1.156180   0.008492 136.147  < 2e-16 ***
## s_launch_angle      0.273571   0.009474  28.875  < 2e-16 ***
## s_launch_speed     -0.356507   0.009007 -39.580  < 2e-16 ***
## s_spray_angle_Kolp  0.095177   0.008485  11.217  < 2e-16 ***
## s_spray_angle_adj  -0.173628   0.009065 -19.154  < 2e-16 ***
## s_hp_to_1b          0.054310   0.008409   6.458 1.06e-10 ***
## s_if_alignment      0.015170   0.008447   1.796   0.0725 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 90063  on 80465  degrees of freedom
## Residual deviance: 87625  on 80459  degrees of freedom
## AIC: 87639
##
## Number of Fisher Scoring iterations: 4
```

```r
exp(coef(fit_logit))
```

```
##        (Intercept)     s_launch_angle     s_launch_speed
##          3.1777707          1.3146512          0.7001173
```

```
## s_spray_angle_Kolp  s_spray_angle_adj         s_hp_to_1b
##          1.0998539          0.8406098          1.0558123
##     s_if_alignment
##          1.0152856
```

```
exp(confint(fit_logit))
```

```
##                         2.5 %    97.5 %
## (Intercept)         3.1254016 3.2311942
## s_launch_angle      1.2904876 1.3393161
## s_launch_speed      0.6878422 0.7125628
## s_spray_angle_Kolp  1.0817216 1.1183050
## s_spray_angle_adj   0.8258031 0.8556751
## s_hp_to_1b          1.0385685 1.0733751
## s_if_alignment      0.9986340 1.0322562
```

From the p-values for the regression coefficients, we can see that all of our predictors made a significant contribution to the model except for Infield Alignment. This is no surprise since we knew that Infield Alignment was useful only as a moderator of Spray Angle. By itself, it didn't have much to say about whether a batted ball would result in a single or an out.

The function for the exponential distribution (exp()) makes the regression coefficients easier to interpret by converting the log odds back to the odds scale. Now we can see, for example, that the odds of a single are increased by a factor of 1.315 for a one unit increase in *launch_angle*, assuming the other predictors remain constant. The confidence interval function (confint()) generates the confidence intervals of the coefficients.

We then used our logistic regression model to estimate the probability of a single for each batted ball in *test_set*.

```
# Using our logistic regression model (fit_logit), estimate the probability
# of a single for each observation (batted ball) in test_set.

p_hat_logit <- predict(fit_logit, newdata = test_set, type = "response")
test_set <- add_column(test_set, p_hat_logit)
```

We next needed to apply a decision rule to the probabilities so that the model could actually predict whether each batted ball in the test_set will result in a single. This first required determining the optimal cutoff (probability) that best differentiated between singles and outs. First, we defined "optimal cutoff" as the probability that maximizes the Truescore.

```
# Determine the cutoff in the decision rule by identifying the cutoff (probability)
  # that maximizes the truescore.

cutoffs <- test_set$p_hat_logit
truescores <- map_dbl(cutoffs, function(x){
  truescore_preds <- ifelse(test_set$p_hat_logit >= x, "single", "field_out") %>%
    factor(levels = levels(test_set$events))
  truescore_tprs <- round(sensitivity(truescore_preds,
                                      reference = factor(test_set$events)), 6)
  truescore_tnrs <- round(specificity(truescore_preds,
                                      reference = factor(test_set$events)), 6)
  truescores <- round((2 * truescore_tprs * truescore_tnrs) /
                        (truescore_tprs + truescore_tnrs) ,6)
```

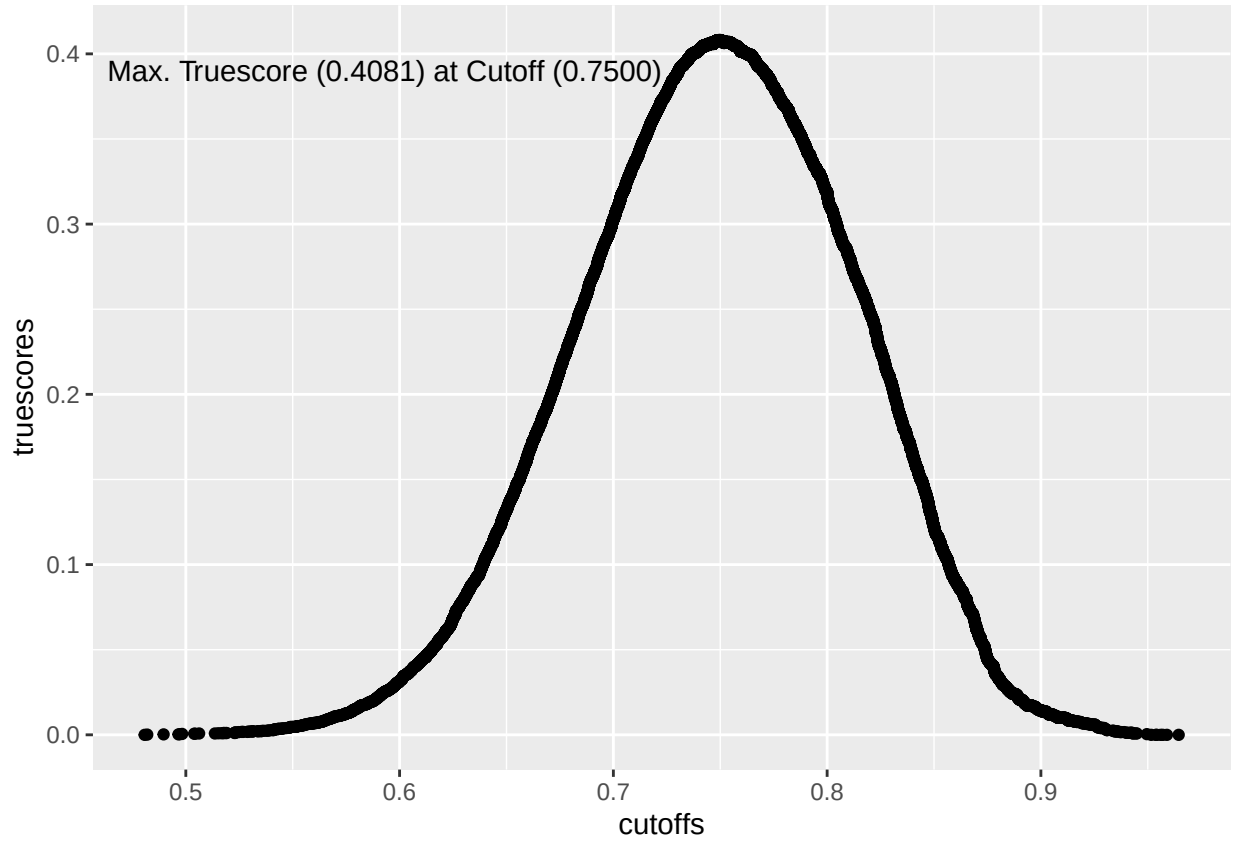
```
})
test_set <- mutate(test_set, truescores = truescores)
(max_truescore <- max(truescores))
```

```
## [1] 0.408139
```

```
(max_truescore_cutoff <- round(cutoffs[which.max(truescores)], 6))
```

```
## [1] 0.749984
```

```
qplot(x = cutoffs, y = truescores) + annotate("text", x = 0.593, y = 0.39,
                                    label = "Max. Truescore (0.4081) at Cutoff (0.7500)")
```



Now we could apply the decision rule based on the cutoff that maximized the Truescore. Once we obtained the model's predictions against the *test_set*, we could compute the model's TPR, TNR, and FPR.

```
# Apply a decision rule based on the cutoff that maximizes the truescore.

max_truescore_preds <- ifelse(p_hat_logit >= max_truescore_cutoff, "single",
                              "field_out") %>%
  factor(levels = levels(test_set$events))
test_set <- add_column(test_set, max_truescore_preds)
(max_truescore_tpr <- round(sensitivity(max_truescore_preds,
                                        reference = factor(test_set$events)), 6))
```

```
## [1] 0.383256
```

```
(max_truescore_tnr <- round(specificity(max_truescore_preds,
                                        reference = factor(test_set$events)), 6))
```

```
## [1] 0.436216
```

```
(max_truescore_fpr <- round(1 - max_truescore_tnr, 6))
```

```
## [1] 0.563784
```

Next, we used our minimum distance method to determine the optimal cutoff point, used that cutoff point to make predictions against *test_set*, and computed the model's TPR, TNR, and FPR.

```
  # Determine the cutoff in the decision rule by identifying the probability that
  # minimizes the distance from the model's ROC curve to the coordinate (0, 1).

pred <- prediction(test_set$p_hat_logit, test_set$events, label.ordering = c("field_out",
                                                                             "single"))
perf <- performance(pred, measure="tpr", x.measure="fpr")
rocr_perf_elements <- tibble(cutoffs = perf@alpha.values[[1]],
                             FPR = round(perf@x.values[[1]], 6),
                             TPR = round(perf@y.values[[1]], 6))
rocr_perf_elements <- rocr_perf_elements %>% mutate(distance =
                                                      sqrt((1 - TPR)^2 + (FPR)^2))

(min_distance <- round(min(rocr_perf_elements$distance), 6))
```

```
## [1] 0.831776
```

```
(min_distance_cutoff <-
    round(rocr_perf_elements$cutoffs[which.min(rocr_perf_elements$distance)], 6))
```

```
## [1] 0.764628
```

```
(min_distance_tpr <-
    round(rocr_perf_elements$TPR[which.min(rocr_perf_elements$distance)], 6))
```

```
## [1] 0.328448
```

```
(min_distance_fpr <-
    round(rocr_perf_elements$FPR[which.min(rocr_perf_elements$distance)], 6))
```

```
## [1] 0.490784
```

```
(min_distance_tnr <- round(1 - min_distance_fpr, 6))
```

```
## [1] 0.509216
```

```r
# Compute the truescore of the optimal cutoff point identified by minimizing the distance.

(min_distance_truescore <- round((2 * min_distance_tpr * min_distance_tnr) /
                                   (min_distance_tpr + min_distance_tnr), 6))
```

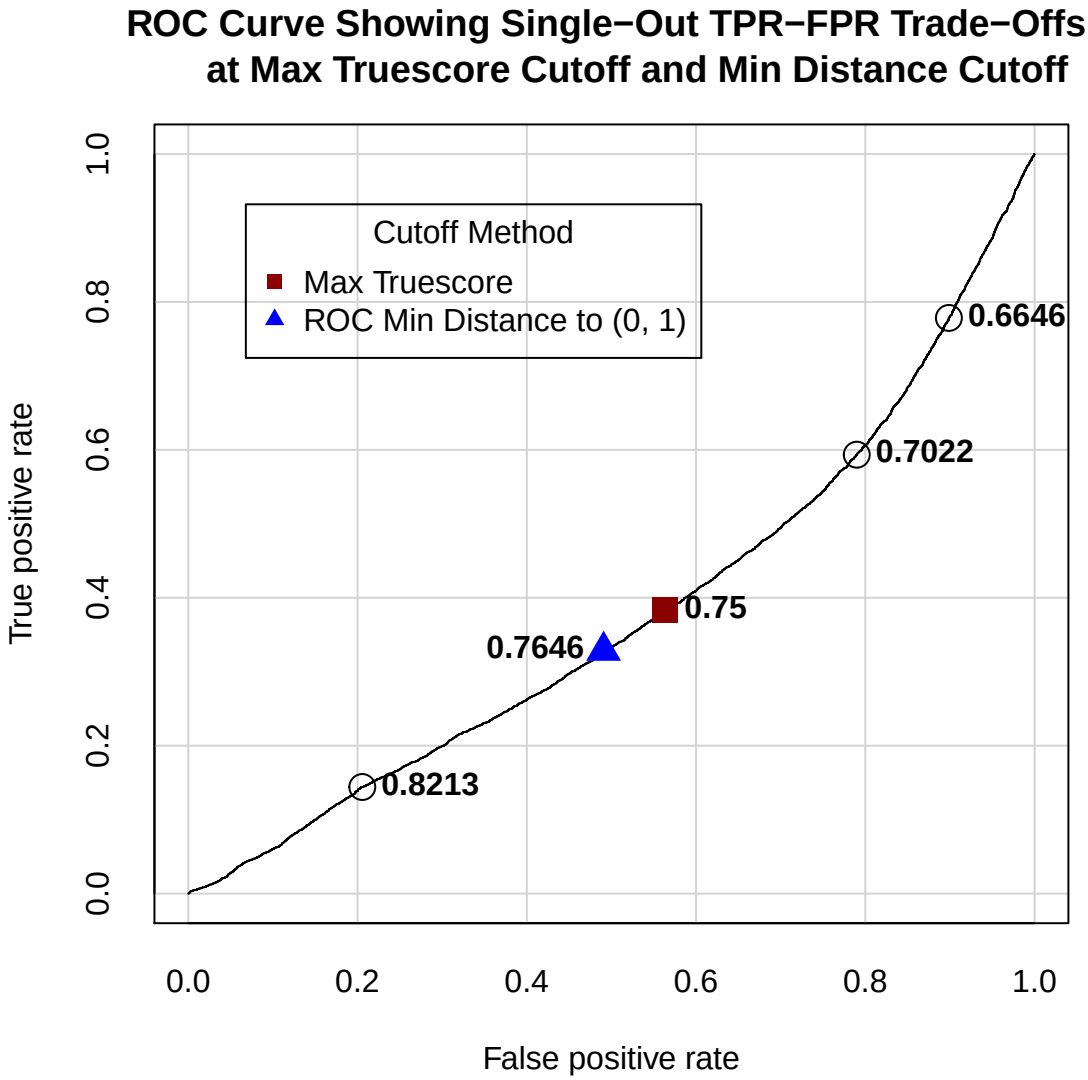
```
## [1] 0.399327
```

```r
# Compute the distance of the optimal cutoff point identified by maximizing the truescore.

(max_truescore_distance <-
    round(sqrt((1 - max_truescore_tpr)^2 + (max_truescore_fpr)^2), 6))
```

```
## [1] 0.835599
```

A plot of the ROC curve and the two decision points generated by our cutoff methods helped us visualize the difference between the two approaches. The other labeled decision points are for reference only.

```r
# Create a ROC curve visualizing the optimal cutoff points identified
# by our two methods: Max Truescore and Min Distance.

plot(perf, main = "ROC Curve Showing Single-Out TPR-FPR Trade-Offs
     at Max Truescore Cutoff and Min Distance Cutoff",
     print.cutoffs.at = c(0.749984, 0.764628, 0.8213, 0.7022, 0.6646),
     cutoff.label.function = function(x) { round(x, 4) },
     points.pch = c(15, 17, 21, 21, 21),
     points.col = c("dark red", "blue", "black", "black", "black"),
     points.cex = c(1.8, 1.8, 1.8, 1.8, 1.8), text.font = c(2, 2, 2, 2, 2),
     text.pos = c(4, 2, 4, 4, 4),
     cutoff.label.text = c("Max Truescore", "Min Distance", "ref", "ref", "ref"),
     xaxis.tck = 1, yaxis.tck = 1, xaxis.col.tick = "light gray",
     yaxis.col.tick = "light gray")
legend("topleft", inset = 0.1, title = "Cutoff Method",
       legend = c("Max Truescore", "ROC Min Distance to (0, 1)"),
       pch = c(15, 17), col = c("dark red", "blue"))
```



For purposes of comparing our two logistic regression models, we computed the Truescore of the Minimum Distance Model, and the distance of the maximum Truescore Model.

```
# Compute the truescore of the optimal cutoff point identified by minimizing the distance.

(min_distance_truescore <- round((2 * min_distance_tpr * min_distance_tnr) /
                                 (min_distance_tpr + min_distance_tnr), 6))
```

```
## [1] 0.399327
```

```
# Compute the distance of the optimal cutoff point identified by maximizing the truescore.

(max_truescore_distance <-
    round(sqrt((1 - max_truescore_tpr)^2 + (max_truescore_fpr)^2), 6))
```

```
## [1] 0.835599
```

Finally, we created a table displaying the assessment results obtained for the models tested thus far.

```
# Create a table displaying the assessment results obtained thus far.

assessment_results <- tibble(Model = c("Baseline", "Log. Regression",
                                       "Log. Regression"), `Cutoff Method` =
                               c(NA, "Max Truescore", "ROC Min Distance"),
                             `Truescore` =
                               c(bl_truescore, max_truescore, min_distance_truescore),
                             TPR = c(bl_tpr, max_truescore_tpr, min_distance_tpr),
                             TNR = c(bl_tnr, max_truescore_tnr, min_distance_tnr),
                             ROC_Distance =
                               c(NA, max_truescore_distance, min_distance),
                             FPR = c((1 - bl_tnr), max_truescore_fpr, min_distance_fpr),
                             `Best Cutoff` =
                               c(NA, max_truescore_cutoff, min_distance_cutoff))
knitr::kable(assessment_results[1:3, ],
             caption = "Assessment Results - First Logistic Regression Model")
```

Table 1: Assessment Results - First Logistic Regression Model

Model	Cutoff Method	Truescore	TPR	TNR	ROC_Distance	FPR	Best Cutoff
Baseline	NA	0.385984	0.259787	0.750611	NA	0.249389	NA
Log. Regression	Max Truescore	0.408139	0.383256	0.436216	0.835599	0.563784	0.749984
Log. Regression	ROC Min Distance	0.399327	0.328448	0.509216	0.831776	0.490784	0.764628

Shockingly, our two logistic regression models performed only marginally better than our guessing algorithm. What to do? The function for fitting generalized linear models allows a user to specify an interactive term by inserting a colon between two interactive variables in the formula argument. So here was an opportunity to see if including infield alignment in our model contributed to the model's predictive ability.

```
# Fit a refined logistic regression model (r_fit_logit) with interaction terms to
# train_set$events.

r_fit_logit <- train_set %>% mutate(y = as.numeric(events == "field_out")) %>%
  glm(y ~ s_launch_angle + s_launch_speed + s_spray_angle_Kolp +
        s_spray_angle_adj + s_hp_to_1b + s_if_alignment +
        s_spray_angle_Kolp:s_if_alignment + s_spray_angle_adj:s_if_alignment,
      data = ., family = "binomial")

# Make sure each regression coefficient is statistically significant (p < .05).

summary(r_fit_logit)
```

```
##
## Call:
## glm(formula = y ~ s_launch_angle + s_launch_speed + s_spray_angle_Kolp +
##     s_spray_angle_adj + s_hp_to_1b + s_if_alignment + s_spray_angle_Kolp:s_if_alignment +
##     s_spray_angle_adj:s_if_alignment, family = "binomial", data = .)
##
## Deviance Residuals:
```

```
##     Min       1Q   Median       3Q      Max
## -2.6177   0.3812   0.6754   0.7955   1.3129
##
## Coefficients:
##                                   Estimate Std. Error z value Pr(>|z|)
## (Intercept)                       1.152070   0.008505 135.452  < 2e-16
## s_launch_angle                    0.274971   0.009485  28.990  < 2e-16
## s_launch_speed                   -0.358486   0.009016 -39.762  < 2e-16
## s_spray_angle_Kolp                0.068066   0.008914   7.636 2.25e-14
## s_spray_angle_adj                -0.158948   0.009455 -16.811  < 2e-16
## s_hp_to_1b                        0.053108   0.008418   6.309 2.81e-10
## s_if_alignment                    0.017145   0.008463   2.026   0.0428
## s_spray_angle_Kolp:s_if_alignment  0.040482   0.008953   4.522 6.13e-06
## s_spray_angle_adj:s_if_alignment  -0.081680   0.008923  -9.154  < 2e-16
##
## (Intercept)                       ***
## s_launch_angle                    ***
## s_launch_speed                    ***
## s_spray_angle_Kolp                ***
## s_spray_angle_adj                 ***
## s_hp_to_1b                        ***
## s_if_alignment                    *
## s_spray_angle_Kolp:s_if_alignment ***
## s_spray_angle_adj:s_if_alignment  ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 90063  on 80465  degrees of freedom
## Residual deviance: 87508  on 80457  degrees of freedom
## AIC: 87526
##
## Number of Fisher Scoring iterations: 4
```

```
exp(coef(r_fit_logit))
```

```
##                       (Intercept)                    s_launch_angle
##                         3.1647378                         1.3164923
##                    s_launch_speed                s_spray_angle_Kolp
##                         0.6987334                         1.0704363
##                 s_spray_angle_adj                         s_hp_to_1b
##                         0.8530406                         1.0545436
##                    s_if_alignment s_spray_angle_Kolp:s_if_alignment
##                         1.0172931                         1.0413125
##  s_spray_angle_adj:s_if_alignment
##                         0.9215669
```

```
exp(confint(r_fit_logit))
```

```
##                                       2.5 %    97.5 %
## (Intercept)                       3.1125027 3.2180253
## s_launch_angle                    1.2922674 1.3412206
```

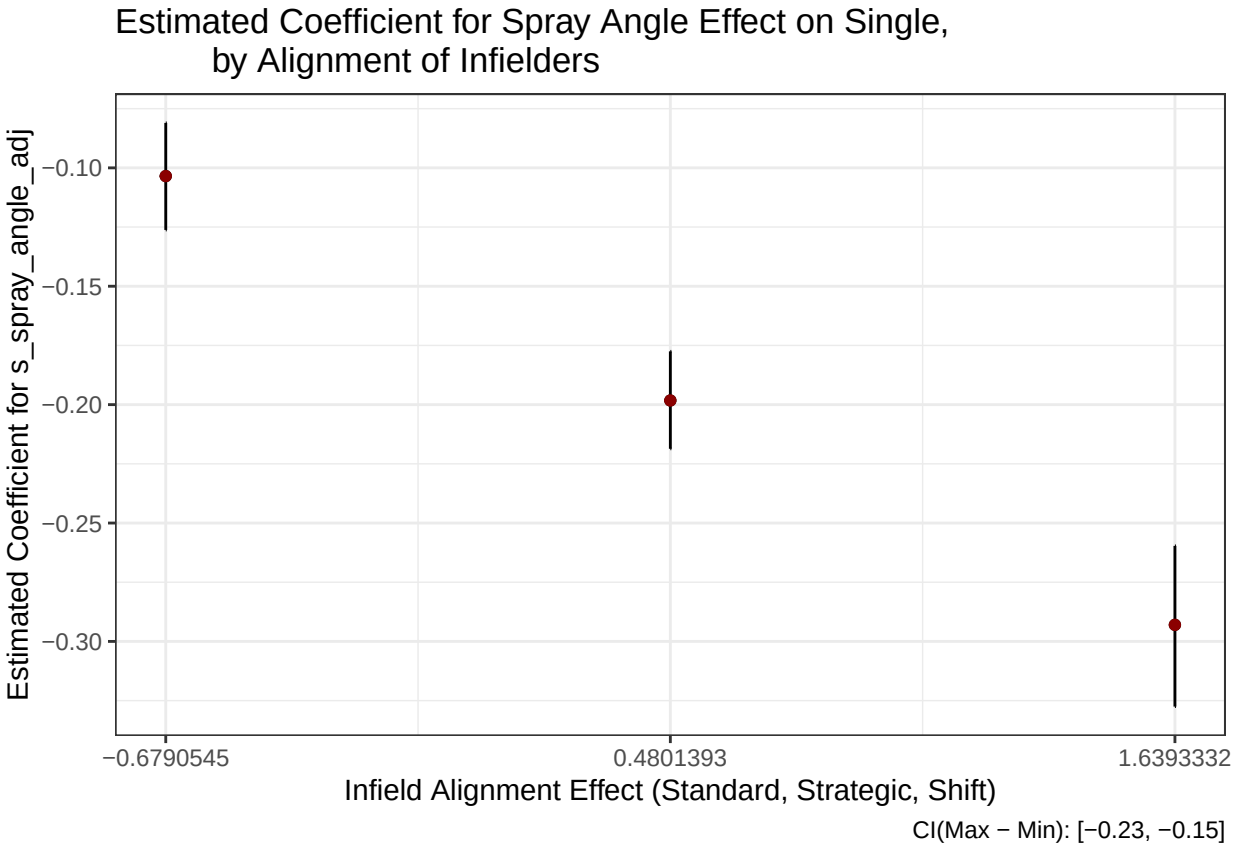
```
## s_launch_speed                     0.6864713 0.7111658
## s_spray_angle_Kolp                 1.0519014 1.0893089
## s_spray_angle_adj                  0.8373737 0.8689921
## s_hp_to_1b                         1.0373024 1.0721038
## s_if_alignment                     1.0005776 1.0343301
## s_spray_angle_Kolp:s_if_alignment 1.0231970 1.0597435
## s_spray_angle_adj:s_if_alignment  0.9055892 0.9378245
```

As can be seen from the return of the summary() function, the p-values of all of our predictors (even infield alignment!) were statistically significant. The interactive terms' exponentiated coefficients appeared modest but not miniscule in impact. In any case, researchers appear to agree that the nonlinear nature of interaction means that "the statistical estimate of an interactive effect cannot be interpreted as straightforward as the coefficient of a regular regression parameter." [8] In other words, the statistical significance of an interactive term's regression coefficient doesn't tell us much.

Researchers have turned to other techniques to assess the effect of a conditional variable (Infield Alignment) on the contribution of another variable (Spray Angle) to the dependent variable (Event (single or out)). Wondering whether either variety of our logistic regression model would perform better than previously as a result of the interactive term, we decided to use the interplot package to construct a conditional effect plot that would help us visualize the effect of infield alignment on the coefficient for Spray Angle.

```
# Create a conditional effect plot to visualize how infield alignment affects
# the coefficient for s_spray_angle_adj.

interplot(m = r_fit_logit, var1 = "s_spray_angle_adj", var2 = "s_if_alignment") +
  geom_point(color = "dark red") +
  ggtitle("Estimated Coefficient for Spray Angle Effect on Single,
          by Alignment of Infielders") +
  theme(plot.title = element_text(face = "bold")) +
  xlab("Infield Alignment Effect (Standard, Strategic, Shift)") +
  ylab("Estimated Coefficient for s_spray_angle_adj") +
  theme_bw()
```

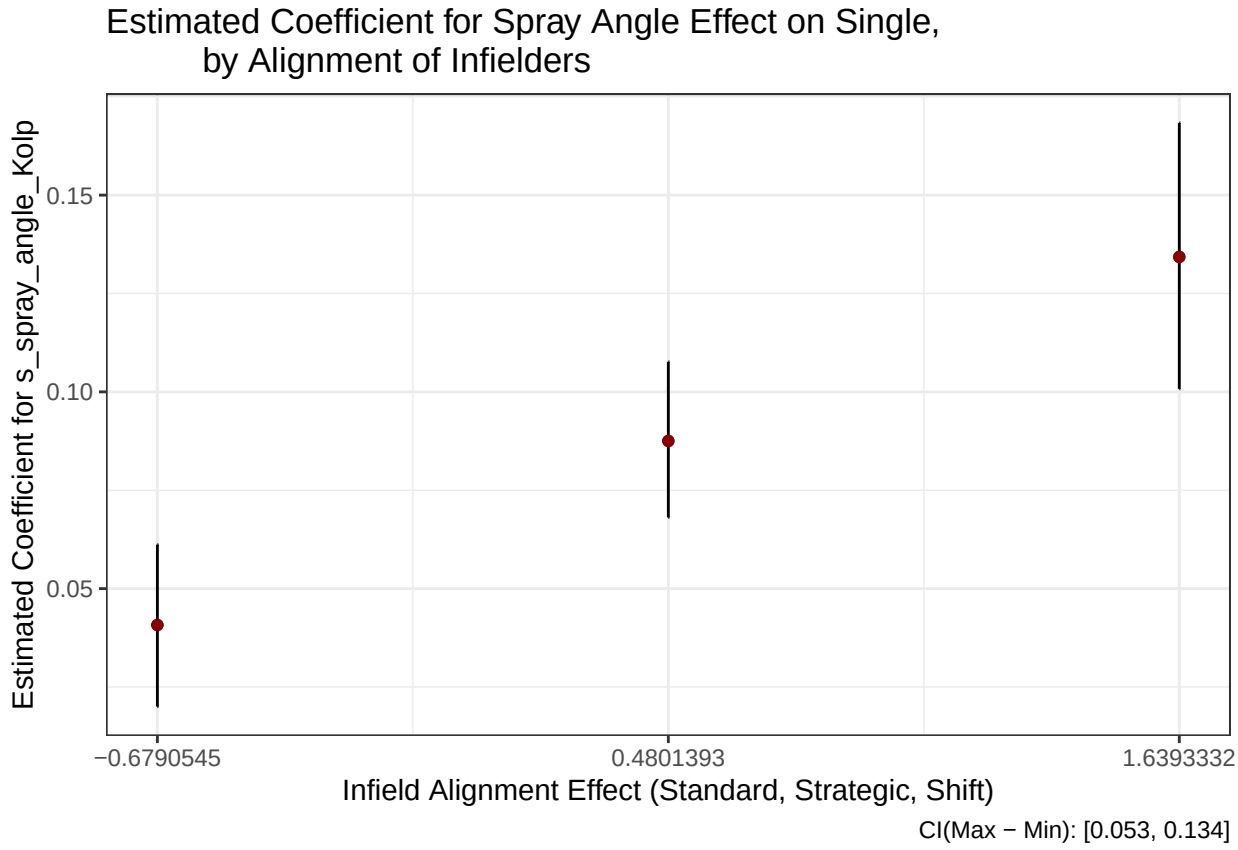
[8] Solt, Frederick, Hu, Yue, "interplot: Plot the Effects of Variables in Interaction Terms" (2019). https://cran.r-project.org/web/packages/interplot/vignettes/interplot-vignette.html



The above conditional effect plot shows how the estimated coefficient for *s_spray_angle_adj* varies depending on the level of the conditional factor *s_infield_alignment*. The lines extending from each point represent the 95% confidence interval. In this case, the intervals do not overlap, indicating genuinely distinct coefficients for each level. Moreover, the nature of the effect makes sense. When the infielders are shifted, *s_spray_angle_adj* has a greater impact on the response variable. We saw during data exploration the dramatic density plots of *spray_angle_adj* differentiating singles from outs when a shift was deployed.

We expected *s_if_alignment* to have a smaller effect on *s_spray_angle_Kolp*, but we created a separate conditional effect plot for this moderator as well.

```
# Create a conditional effect plot to visualize how infield alignment affects
# the coefficient for s_spray_angle_Kolp.

interplot(m = r_fit_logit, var1 = "s_spray_angle_Kolp", var2 = "s_if_alignment") +
  geom_point(color = "dark red") +
  ggtitle("Estimated Coefficient for Spray Angle Effect on Single,
          by Alignment of Infielders") +
  theme(plot.title = element_text(face = "bold")) +
  xlab("Infield Alignment Effect (Standard, Strategic, Shift)") +
  ylab("Estimated Coefficient for s_spray_angle_Kolp") +
  theme_bw()
```



The plot confirmed our expectations. This time, the estimated coefficients were closer to each other, with confidence intervals overlapping or nearly overlapping. Yet, the plot still depicted an interactive relationship between infield alignment and *spray_angle_Kolp*.

With our hopes buoyed, we proceeded to test the new logistic regression model using *test_set*.

```
# Using our refined logistic regression model (r_fit_logit), estimate the probability
# of a single for each observation (batted ball) in test_set.

r_p_hat_logit <- predict(r_fit_logit, newdata = test_set, type = "response")
test_set <- add_column(test_set, r_p_hat_logit)
```

Using the same steps we used earlier, we next determined the optimal cutoff (probability) using the maximum Truescore method. The optimal cutoff was then used to apply a decision rule that predicted whether each batted ball in the test_set will result in a single.

```
# Determine the cutoff in the decision rule by identifying the cutoff (probability)
  # that maximizes the truescore.

r_cutoffs <- test_set$r_p_hat_logit
r_truescores <- map_dbl(r_cutoffs, function(x){
  r_truescore_preds <- ifelse(test_set$r_p_hat_logit >= x, "single", "field_out") %>%
    factor(levels = levels(test_set$events))
  r_truescore_tprs <- round(sensitivity(r_truescore_preds,
                                        reference = factor(test_set$events)), 6)
  r_truescore_tnrs <- round(specificity(r_truescore_preds,
```

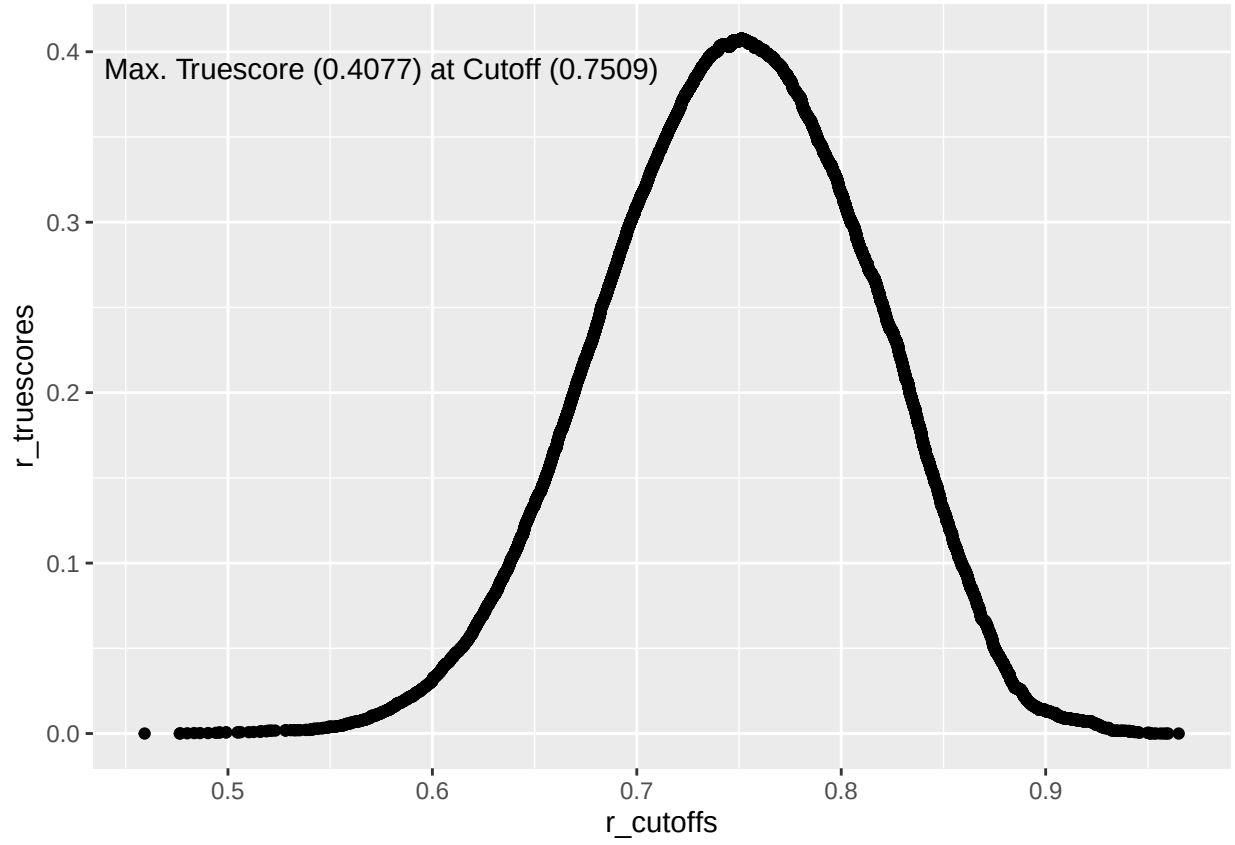
```
                                        reference = factor(test_set$events)), 6)
  r_truescores <- round((2 * r_truescore_tprs * r_truescore_tnrs) /
                          (r_truescore_tprs + r_truescore_tnrs) ,6)
})
test_set <- mutate(test_set, r_truescores = r_truescores)
(r_max_truescore <- max(r_truescores))
```

```
## [1] 0.407656
```

```
(r_max_truescore_cutoff <- round(r_cutoffs[which.max(r_truescores)], 6))
```

```
## [1] 0.750887
```

```
qplot(x = r_cutoffs, y = r_truescores) +
  annotate("text", x = 0.575, y = 0.39,
           label = "Max. Truescore (0.4077) at Cutoff (0.7509)")
```




```
  # Apply a decision rule based on the cutoff that maximizes the truescore.

r_max_truescore_preds <- ifelse(r_p_hat_logit >= r_max_truescore_cutoff, "single",
                                "field_out") %>%
  factor(levels = levels(test_set$events))
test_set <- add_column(test_set, r_max_truescore_preds)
(r_max_truescore_tpr <- round(sensitivity(r_max_truescore_preds,
                                          reference = factor(test_set$events)), 6))
```

```
## [1] 0.376832
```

```
(r_max_truescore_tnr <- round(specificity(r_max_truescore_preds,
                                          reference = factor(test_set$events)), 6))
```

```
## [1] 0.443417
```

```
(r_max_truescore_fpr <- round(1 - r_max_truescore_tnr, 6))
```

```
## [1] 0.556583
```

We repeated the same steps again, this time using the Minimum Distance method.

```
  # Determine the cutoff in the decision rule by identifying the probability that
  # minimizes the distance from the model's ROC curve to the coordinate (0, 1). Then apply
  # a decision rule using that probability (cutoff).

r_pred <- prediction(test_set$r_p_hat_logit, test_set$events,
                     label.ordering = c("field_out", "single"))
r_perf <- performance(r_pred, measure="tpr", x.measure="fpr")
r_rocr_perf_elements <- tibble(r_cutoffs = r_perf@alpha.values[[1]],
                               r_FPR = round(r_perf@x.values[[1]], 6),
                               r_TPR = round(r_perf@y.values[[1]], 6))
r_rocr_perf_elements <- r_rocr_perf_elements %>%
  mutate(r_distance = sqrt((1 - r_TPR)^2 + (r_FPR)^2))

(r_min_distance <- round(min(r_rocr_perf_elements$r_distance), 6))
```

```
## [1] 0.832194
```

```
(r_min_distance_cutoff <-
    round(r_rocr_perf_elements$r_cutoffs[which.min(r_rocr_perf_elements$r_distance)], 6))
```

```
## [1] 0.766355
```

```
(r_min_distance_tpr <-
    round(r_rocr_perf_elements$r_TPR[which.min(r_rocr_perf_elements$r_distance)], 6))
```

```
## [1] 0.322024
```

```
(r_min_distance_fpr <-
    round(r_rocr_perf_elements$r_FPR[which.min(r_rocr_perf_elements$r_distance)], 6))
```

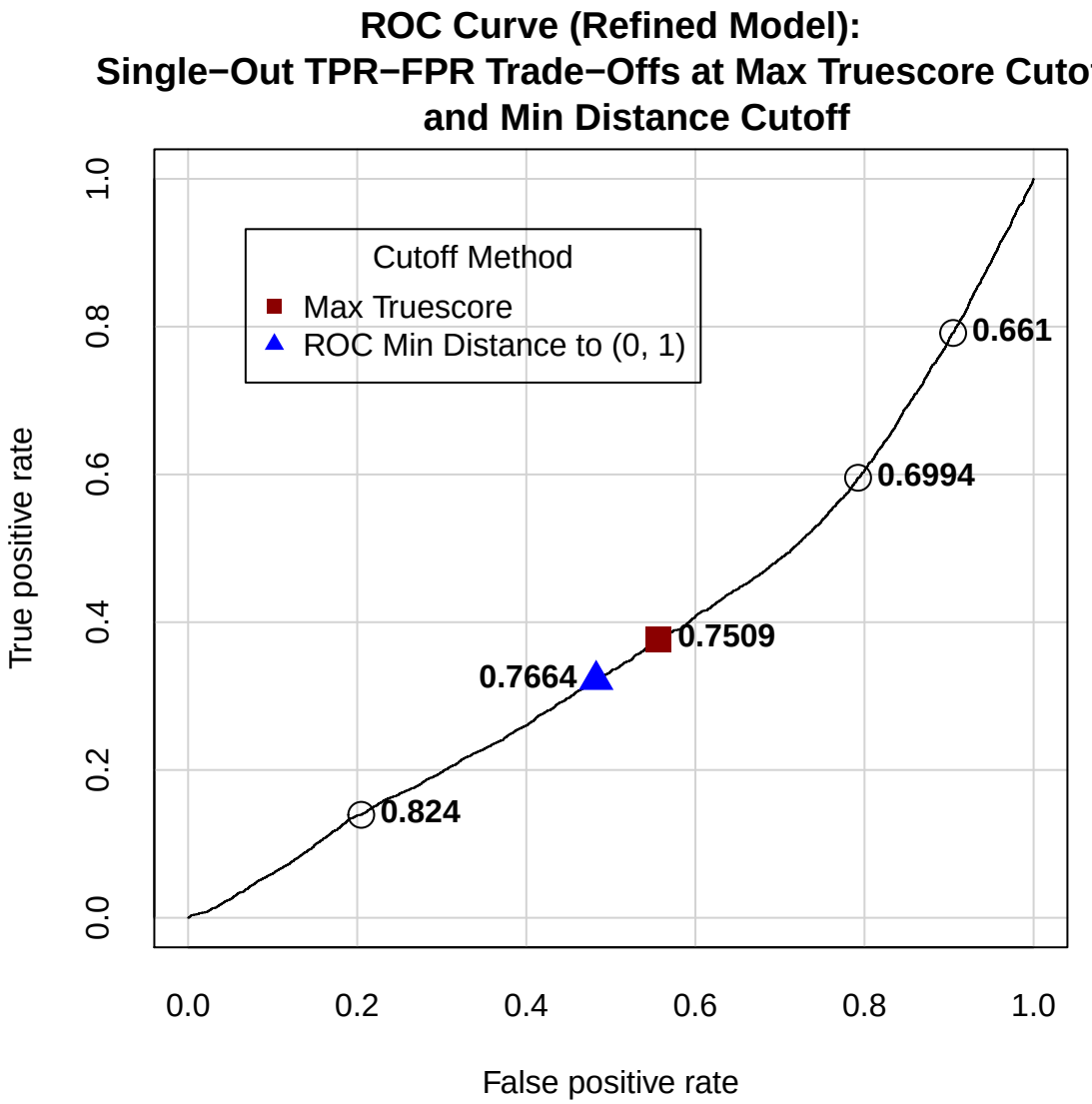
```
## [1] 0.482592
```

```
(r_min_distance_tnr <- round(1 - r_min_distance_fpr, 6))
```

```
## [1] 0.517408
```

We plotted the new ROC curve and the two decision points generated by our cutoff methods.

```r
# Create a ROC curve allowing us to visualize the optimal cutoff points identified
# by our two methods: Max Truescore and Min Distance.

plot(r_perf, main = "ROC Curve (Refined Model):
Single-Out TPR-FPR Trade-Offs at Max Truescore Cutoff
     and Min Distance Cutoff",
     print.cutoffs.at = c(0.750887, 0.766355, 0.6610, 0.6994, 0.8240),
     cutoff.label.function = function(x) { round(x, 4) },
     points.pch = c(15, 17, 21, 21, 21),
     points.col = c("dark red", "blue", "black", "black", "black"),
     points.cex = c(1.8, 1.8, 1.8, 1.8, 1.8), text.font = c(2, 2, 2, 2, 2),
     text.pos = c(4, 2, 4, 4, 4),
     cutoff.label.text = c("Max Truescore", "Min Distance", "ref", "ref", "ref"),
     xaxis.tck = 1, yaxis.tck = 1, xaxis.col.tick = "light gray",
     yaxis.col.tick = "light gray")
legend("topleft", inset = 0.1, title = "Cutoff Method",
       legend = c("Max Truescore", "ROC Min Distance to (0, 1)"),
       pch = c(15, 17), col = c("dark red", "blue"))
```



We again computed the Truescore of the Minimum Distance Model, and the distance of the maximum Truescore Model to facilitate comparison of our two refined logistic regression models.

```
# Compute the truescore of the optimal cutoff point identified by minimizing the distance.

(r_min_distance_truescore <- round((2 * r_min_distance_tpr * r_min_distance_tnr) /
                                   (r_min_distance_tpr + r_min_distance_tnr), 6))
```

```
## [1] 0.396977
```

```
# Compute the distance of the optimal point identified by maximizing the weighted
# truescore.

(r_max_truescore_distance <-
    round(sqrt((1 - r_max_truescore_tpr)^2 + (r_max_truescore_fpr)^2), 6))
```

```
## [1] 0.835538
```

The updated table of results for all of our models followed.

```
# Create a table displaying all of the assessment results.

r_assessment_results <- tibble(Model = c("Baseline", "Log. Regression",
                                         "Log. Regression", "Log. Regression",
                                         "Log. Regression"),
                               `Cutoff Method` = c("na", "Max Truescore",
                                                   "ROC Min Distance.",
                                                   "Max Truescore (refined)",
                                                   "ROC Min Distance (ref.)"),
                               `Truescore` = c(bl_truescore, max_truescore,
                                               min_distance_truescore, r_max_truescore,
                                               r_min_distance_truescore),
                               TPR = c(bl_tpr, max_truescore_tpr, min_distance_tpr,
                                       r_max_truescore_tpr, r_min_distance_tpr),
                               TNR = c(bl_tnr, max_truescore_tnr, min_distance_tnr,
                                       r_max_truescore_tnr, r_min_distance_tnr),
                               `ROC
                               Distance` = c("na", max_truescore_distance,
                                             min_distance,
                                             r_max_truescore_distance,
                                             r_min_distance),
                               FPR = c((1 - bl_tnr), max_truescore_fpr,
                                       min_distance_fpr, r_max_truescore_fpr,
                                       r_min_distance_fpr),
                               `Best Cutoff` = c("na", max_truescore_cutoff,
                                                 min_distance_cutoff,
                                                 r_max_truescore_cutoff,
                                                 r_min_distance_cutoff))
knitr::kable(r_assessment_results[1:5, ],
             caption = "Assessment Results - Refined Logistic Regression Model")
```

Table: Assessment Results - Refined Logistic Regression Model

Model Cutoff Method Truescore TPR TNR ROC Distance FPR Best Cutoff ———- ——— ——— ——— ——— ——— ——— Baseline na 0.385984 0.259787 0.750611 na 0.249389 na
Log. Regression Max Truescore 0.408139 0.383256 0.436216 0.835599 0.563784 0.749984
Log. Regression ROC Min Distance. 0.399327 0.328448 0.509216 0.831776 0.490784 0.764628
Log. Regression Max Truescore (refined) 0.407656 0.376832 0.443417 0.835538 0.556583 0.750887
Log. Regression ROC Min Distance (ref.) 0.396977 0.322024 0.517408 0.832194 0.482592 0.766355

Our so called "refined" logistic regression model with an interactive term actually performed worse than our base logistic regression model without any interactive terms. This was true under both the Maximum Truescore method and the Minimum Distance method of measuring predictive ability. In actuality, the results were quite similar. What did stand out, however, was that Maximum Truescore seemed to strike a better balance between sensitivity and specificity than Minimum Distance.

In the end, the predictive ability of all of our logistic regression models was highly disappointing. It could be that the linear nature of logistic regression was simply not flexible enough to capture the complex relationships between our features and our dependent response variable.

Build and Assess a Model with the k-Nearest Neighbors (knn) Algorithm

The knn algorithm estimates conditional probability by averaging the k nearest points (a neighborhood). The parameter k specifies the size of the neighborhood. A larger k results in a smoother estimate, while a smaller k results in a more flexible estimate.

To quickly see whether knn could potentially provide a better performing model than logistic regression, we first used the default value of k, which is 5. We also used normalized data for knn.

```
# Build a knn model with k=5. Prepare the train and test sets, define the response vector
# and matrix of predictors, and fit the model using the knn3 function.

train_set2 <- dplyr::select(train_set, -(hit_distance_sc))
test_set2 <- dplyr::select(test_set, -(hit_distance_sc), -(p_hat_logit), -(truescores),
                           -(max_truescore_preds), -(r_p_hat_logit), -(r_truescores),
                           -(r_max_truescore_preds))
x <- as.matrix(train_set2[, c("n_launch_angle", "n_launch_speed", "n_spray_angle_Kolp",
                              "n_spray_angle_adj", "n_hp_to_1b", "n_if_alignment")])
y <- factor(train_set2$events)
knn_fit_k5 <- knn3(x, y, k = 5)

# Test the knn_fit_k5 model. Define the matrix of new data (test_set) and use it to
# predict the outcomes.

z <- as.matrix(test_set2[, c("n_launch_angle", "n_launch_speed", "n_spray_angle_Kolp",
                             "n_spray_angle_adj", "n_hp_to_1b", "n_if_alignment")])
knn_y_hat <- predict(knn_fit_k5, z, type = "class")
knn_y_hat_prob <- predict(knn_fit_k5, z, type = "prob")

# Assess the predictions using both the Truescore method and the Minimum Distance method.
# Compare the performance of knn_fit_k5 to that of the logistic regression models.

confusionMatrix(data = knn_y_hat, reference = test_set2$events)
```

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction  single field_out
##   single      3162      1437
##   field_out   1819     13700
##
##                Accuracy : 0.8382
##                  95% CI : (0.833, 0.8432)
##     No Information Rate : 0.7524
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5541
##
##  Mcnemar's Test P-Value : 2.438e-11
##
##             Sensitivity : 0.6348
##             Specificity : 0.9051
##          Pos Pred Value : 0.6875
##          Neg Pred Value : 0.8828
```

```
##              Prevalence : 0.2476
##          Detection Rate : 0.1572
##    Detection Prevalence : 0.2286
##       Balanced Accuracy : 0.7699
##
##        'Positive' Class : single
##
```

```
(knn_tpr <- round(sensitivity(knn_y_hat, reference = factor(test_set2$events)), 6))
```

```
## [1] 0.634812
```

```
(knn_tnr <- round(specificity(knn_y_hat, reference = factor(test_set2$events)), 6))
```

```
## [1] 0.905067
```

```
(knn_fpr <- round(1 - knn_tnr, 6))
```

```
## [1] 0.094933
```

```
(knn_truescore <- round((2 * knn_tpr * knn_tnr) / (knn_tpr + knn_tnr) , 6))
```

```
## [1] 0.746224
```

```
(knn_distance <- round(sqrt((1 - knn_tpr)^2 + (knn_fpr)^2), 6))
```

```
## [1] 0.377326
```

```
assessment_results <- tibble(Model = c("Baseline", "Log. Regression",
                                       "Log. Regression", "KNN"),
                             `Cutoff Method` = c(NA, "Truescore",
                                                 "ROC Distance", NA),
                             `Truescore` = c(bl_truescore, max_truescore,
                                             min_distance_truescore, knn_truescore),
                             TPR = c(bl_tpr, max_truescore_tpr, min_distance_tpr,
                                     knn_tpr),
                             TNR = c(bl_tnr, max_truescore_tnr, min_distance_tnr,
                                     knn_tnr),
                             ROC_Distance = c(NA, max_truescore_distance,
                                              min_distance, knn_distance),
                             FPR = c((1 - bl_tnr), max_truescore_fpr,
                                     min_distance_fpr, knn_fpr),
                             `Best Cutoff` = c(NA, max_truescore_cutoff,
                                               min_distance_cutoff, NA))
knitr::kable(assessment_results[1:4, ], caption = "Assessment Results")
```

Table 2: Assessment Results

Model	Cutoff Method	Truescore	TPR	TNR	ROC_Distance	FPR	Best Cutoff
Baseline	NA	0.385984	0.259787	0.750611	NA	0.249389	NA
Log. Regression	Truescore	0.408139	0.383256	0.436216	0.835599	0.563784	0.749984
Log. Regression	ROC Distance	0.399327	0.328448	0.509216	0.831776	0.490784	0.764628
KNN	NA	0.746224	0.634812	0.905067	0.377326	0.094933	NA

The knn model with k=5 performed well, especially in comparison to the logistic regression models. But could its performance be even better with an optimized value for k? We turned to the caret package because of its ability to perform cross-validation to identify optimal k. In this case, we used 10-fold cross-validation, with 20% of the train_set serving as a validation set for cross-validation purposes only. Twenty different k values, ranging from 1 to 39, were subjected to cross_validation. With each of the twenty different k values being tested 10 times, there was a total of 200 tests, so this took some time to complete. We first defined optimal k as the value of k that maximizes the Truescore of the model, and then as the value of k that minimizes the distance to (0, 1).

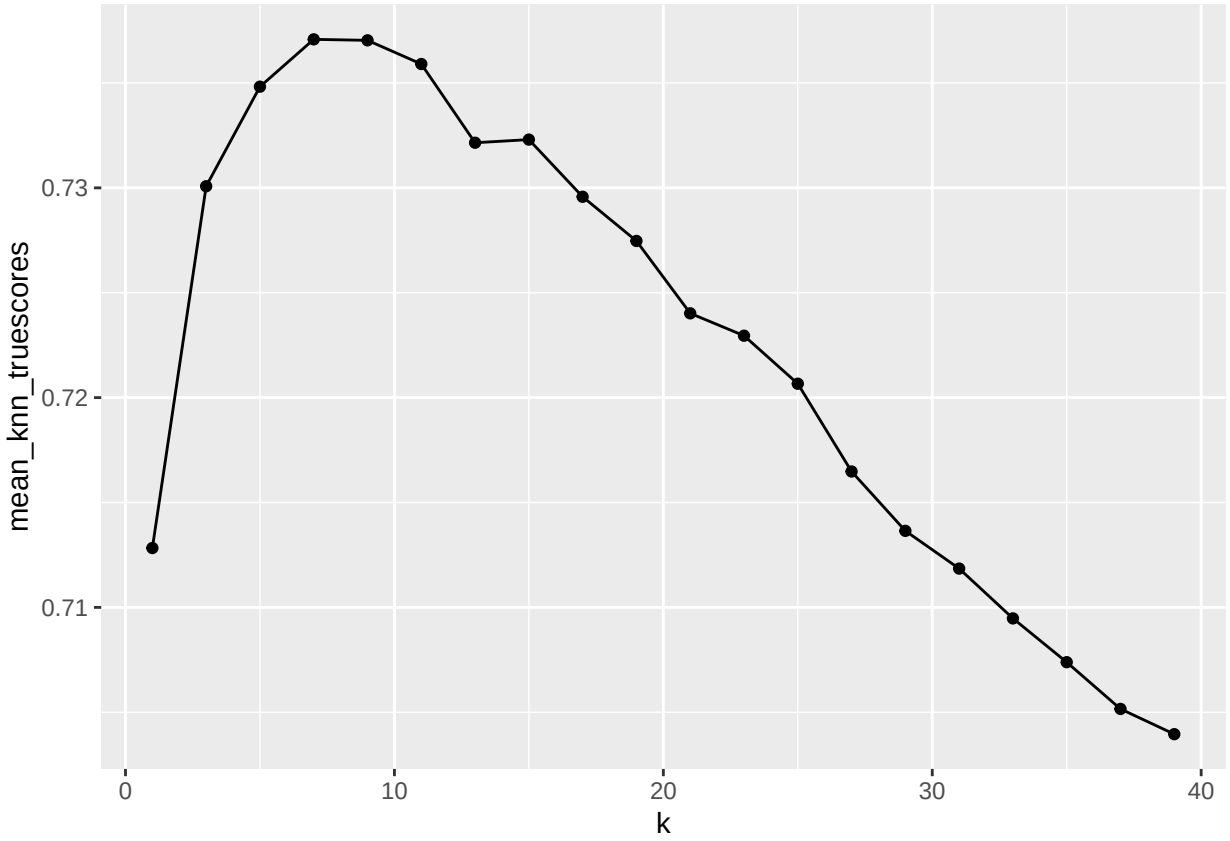
```
  # Determine the value of k that maximizes the truescore of the model.

fitControl <- trainControl(method = "cv", number = 10, p = 0.8, returnData = TRUE,
                           returnResamp = "all", savePredictions = "all",
                           summaryFunction = twoClassSummary, classProbs = TRUE)
set.seed(1)
knn_train <- train(x, y, method = "knn",
                   tuneGrid = data.frame(k = seq(1, 39, 2)),
                   trControl = fitControl)
```

One nice feature of caret's train() function is that we can use an argument (returnResamp) to return the computed sensitivity and specificity for each of the 200 tests. This is invaluable, because the train() function isn't capable of using anything like Maximum Truescore or Minimum Distance (not widely used at present) to select the optimal value of k. We're going to have to do that ourselves, and we can do it because we have all the sensitivity and specificity scores.

```
knn_train_sample_results <- knn_train[["resample"]]

knn_train_sample_results <- knn_train_sample_results %>%
  mutate(knn_truescores = (2 * Sens * Spec) / (Sens + Spec))
knn_train_sample_results <- knn_train_sample_results %>%
  mutate(knn_distances = sqrt((1 - Sens)^2 + (1 - Spec)^2))
```

Now it's simply a matter a finding the value of k with the highest average Truescore, and the value of k with the shortest average distance to (0, 1).

```
knn_train_sample_mean_truescores <- knn_train_sample_results %>% group_by(k) %>%
  summarize(mean_knn_truescores = mean(knn_truescores))
ggplot(knn_train_sample_mean_truescores, aes(x = k, y = mean_knn_truescores)) +
  geom_point() + geom_line()
```



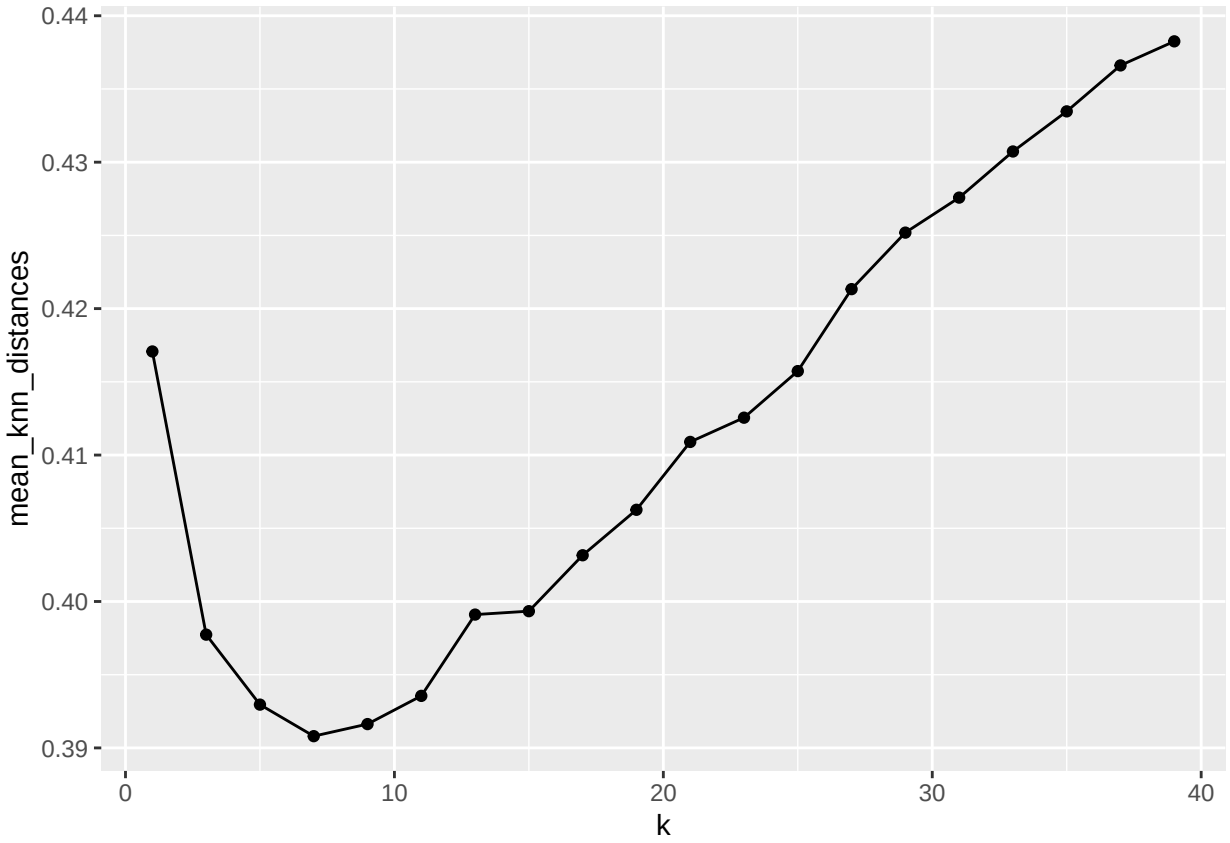

```
max(knn_train_sample_mean_truescores$mean_knn_truescores)
```

```
## [1] 0.7370802
```

```
knn_train_sample_mean_truescores$
  k[which.max(knn_train_sample_mean_truescores$mean_knn_truescores)]
```

```
## [1] 7
```

```
knn_train_sample_mean_distances <- knn_train_sample_results %>% group_by(k) %>%
  summarize(mean_knn_distances = mean(knn_distances))
ggplot(knn_train_sample_mean_distances, aes(x = k, y = mean_knn_distances)) +
  geom_point() + geom_line()
```




```
min(knn_train_sample_mean_distances$mean_knn_distances)
```

```
## [1] 0.3908031
```

```
knn_train_sample_mean_distances$
  k[which.min(knn_train_sample_mean_distances$mean_knn_distances)]
```

```
## [1] 7
```

In this case, both Maximum Truescore and Minimum Distance agreed that the optimal value of k is 7. Armed with optimal k, we could now go back and fit a new model with k=7, and use that model to predict outcomes based on the data in test_set2.

```
  # Fit an optimized knn model (k = 7) to the entire predictor matrix and outcome vector.

knn_fit_k7 <- knn3(x, y, k = 7)

# Apply our final knn model (knn_fit_k7) to predict outcomes based on the predictors in
# test_set2.

knn_y_hat <- predict(knn_fit_k7, z, type = "class")
knn_y_hat_prob <- predict(knn_fit_k7, z, type = "prob")
```

We were finally in a position to assess the predictive ability of our latest knn model.

```r
# Assess the predictive ability of our final knn model (knn_fit_k7).

confusionMatrix(data = knn_y_hat, reference = test_set2$events)
```

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction  single field_out
##   single      3166      1329
##   field_out   1815     13808
##
##                Accuracy : 0.8437
##                  95% CI : (0.8386, 0.8487)
##     No Information Rate : 0.7524
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5664
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.6356
##             Specificity : 0.9122
##          Pos Pred Value : 0.7043
##          Neg Pred Value : 0.8838
##              Prevalence : 0.2476
##          Detection Rate : 0.1574
##    Detection Prevalence : 0.2234
##       Balanced Accuracy : 0.7739
##
##        'Positive' Class : single
##
```

```r
(knn_tpr <- round(sensitivity(knn_y_hat, reference = factor(test_set2$events)), 6))
```

```
## [1] 0.635615
```

```r
(knn_tnr <- round(specificity(knn_y_hat, reference = factor(test_set2$events)), 6))
```

```
## [1] 0.912202
```

```r
(knn_fpr <- round(1 - knn_tnr, 6))
```

```
## [1] 0.087798
```

```r
(knn_truescore <- round((2 * knn_tpr * knn_tnr) / (knn_tpr + knn_tnr) , 6))
```

```
## [1] 0.749196
```

```
(knn_distance <- round(sqrt((1 - knn_tpr)^2 + (knn_fpr)^2), 6))
```

```
## [1] 0.374813
```

```
assessment_results <- tibble(Model = c("Baseline", "Log. Regression",
                                       "Log. Regression", "KNN"),
                             `Cutoff Method` =
                               c(NA, "Truescore", "ROC Distance", NA),
                             `Truescore` =
                               c(bl_truescore, max_truescore, min_distance_truescore,
                                 knn_truescore),
                             TPR = c(bl_tpr, max_truescore_tpr, min_distance_tpr,
                                     knn_tpr),
                             TNR = c(bl_tnr, max_truescore_tnr, min_distance_tnr,
                                     knn_tnr),
                             ROC_Distance = c(NA, max_truescore_distance,
                                              min_distance, knn_distance),
                             FPR = c((1 - bl_tnr), max_truescore_fpr,
                                     min_distance_fpr, knn_fpr),
                             `Best Cutoff` = c(NA, max_truescore_cutoff,
                                               min_distance_cutoff, NA))
knitr::kable(assessment_results[1:4, ], caption = "Assessment Results")
```

Table 3: Assessment Results

Model	Cutoff Method	Truescore	TPR	TNR	ROC_Distance	FPR	Best Cutoff
Baseline	NA	0.385984	0.259787	0.750611	NA	0.249389	NA
Log. Regression	Truescore	0.408139	0.383256	0.436216	0.835599	0.563784	0.749984
Log. Regression	ROC Distance	0.399327	0.328448	0.509216	0.831776	0.490784	0.764628
KNN	NA	0.749196	0.635615	0.912202	0.374813	0.087798	NA

With k=7, our model's predictive ability did indeed improve. Finally, we had a decent Truescore of .7492 and a much shorter distance to (0, 1) of .3748. Could we do better with a different algorithm?

Build and Assess a Model with a Decision (Classification) Tree Algorithm

We next sought to develop a decision or classification tree with an optimized complexity parameter (cp).

One appealing characteristic of classification trees is that they mimic the human decision making process. A classification tree makes a decision (prediction) based on the answers to a series of questions. Each question further reduces the scope of possibilities until a satisfactory level of certainty is reached.

In R, a classification tree model can be built with the rpart function (Recursive Partitioning and Regression Trees). The algorithm "split[s] the data recursively, which means that the subsets that arise from a split are further split until a predetermined termination criterion is reached. At each step, the split is made based on the independent variable that results in the largest possible reduction in heterogeneity of the dependent (predicted) variable." [9]

[9] Awati, Kailash, "A gentle introduction to decision trees using R" (2016). https://eight2late.wordpress.com/2016/02/16/a-gentle-introduction-to-decision-trees-using-r/

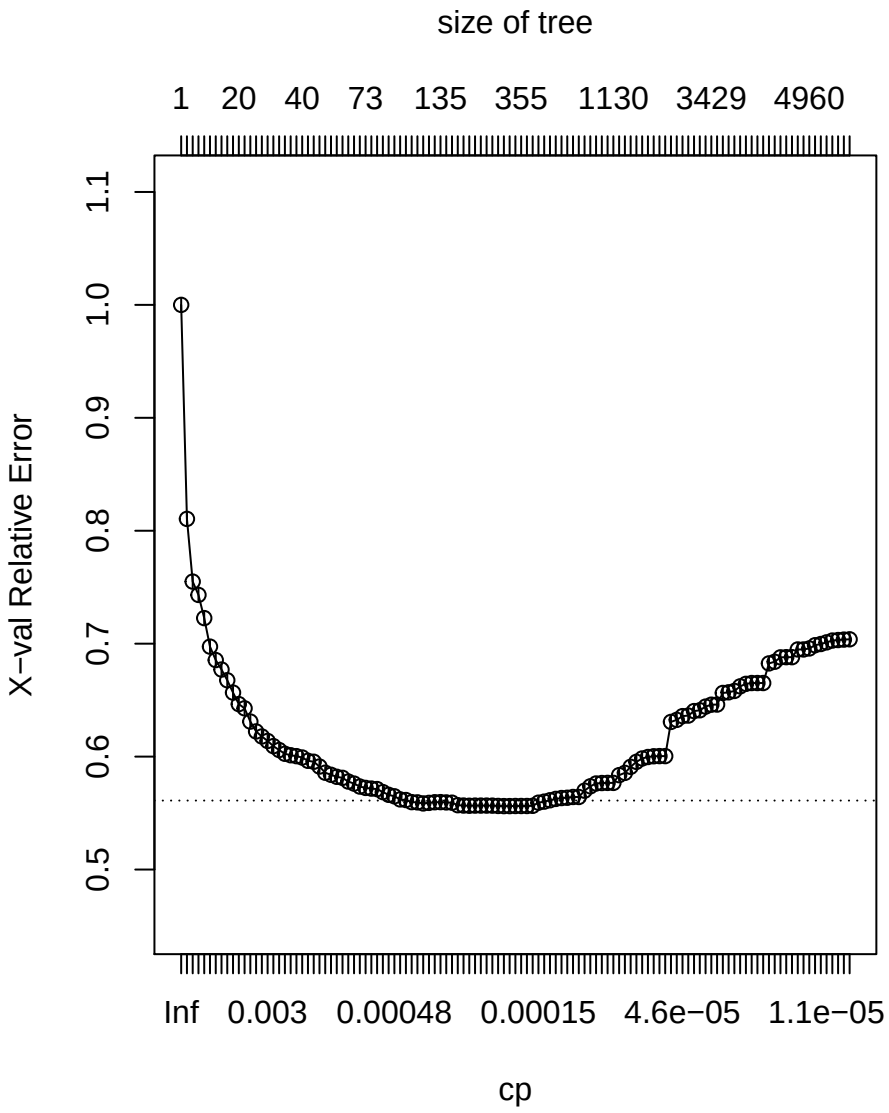
There are different methods for splitting the data to achieve homogeneity. We opted for the Gini Index, which is rpart's default method. A pure table consisting of a single class has a Gini Index of 0. When all classes have an equal probability, the Gini Index is 1.

The initial tree was built using 10-fold cross validation. We also specified the minimum number of observations in any terminal node to be 2, with a complexity parameter (cp) of 0. The cp allows the user to specify the extent to which a split must improve the fit to avoid being pruned off during cross-validation. Our parameters were set to permit the construction of a complete or near-complete tree that could be pruned later.

```
  # Build a complete tree with cp and minsplit set to 0.

train_set3 <- dplyr::select(train_set2, events, launch_angle, launch_speed,
                            spray_angle_Kolp, spray_angle_adj, if_fielding_alignment,
                            hp_to_1b)

set.seed(1)
ctree_train <- rpart(events ~ ., method = "class", data = train_set3,
                     control = rpart.control(xval = 10, minbucket = 2, cp = 0))
```

Having constructed a complete tree, we followed a multi-step process to identify the optimal value of cp for our final model. In the final step of the process, we used what is referred to as "Breiman's one standard error rule" to identify optimal cp. In the context of classification trees, the rule directs the selection of the smallest tree whose error is no more than one standard error above the error of the best tree. Following this rule, the smaller tree would not be worse than the best tree in a statistically significant way.

```
plotcp(ctree_train)
```




```
ctree_train$cptable
```

```
    # Find the lowest estimated cross-validated error (xerror), along with
    # its standard error (xstd).

(cp_min_xerror_index <- which.min(ctree_train$cptable[,"xerror"]))
```

```
## 57
## 57
```

```
(cp_min_xerror <- ctree_train$cptable[cp_min_xerror_index, "xerror"])
```

```
## [1] 0.5561747
```

```
(cp_min_xerror_xstd <- ctree_train$cptable[cp_min_xerror_index, "xstd"])
```

```
## [1] 0.004906748
```

```
    # Identify the highest CP (the smallest tree) whose xerror is still within one
    # standard error of the lowest xerror (Breiman's "one standard error rule").

(cp_max_range_xerror <- cp_min_xerror + cp_min_xerror_xstd)
```

```
## [1] 0.5610814
```

```
(cp_min_range_xerror <- cp_min_xerror - cp_min_xerror_xstd)
```

```
## [1] 0.551268
```

```
cptable <- as.data.frame(ctree_train$cptable)
cptable1 <- cptable %>% filter(xerror <= cp_max_range_xerror &
                                 xerror >= cp_min_range_xerror)
(opt_cp <- max(cptable1[, 1]))
```

```
## [1] 0.000376506
```

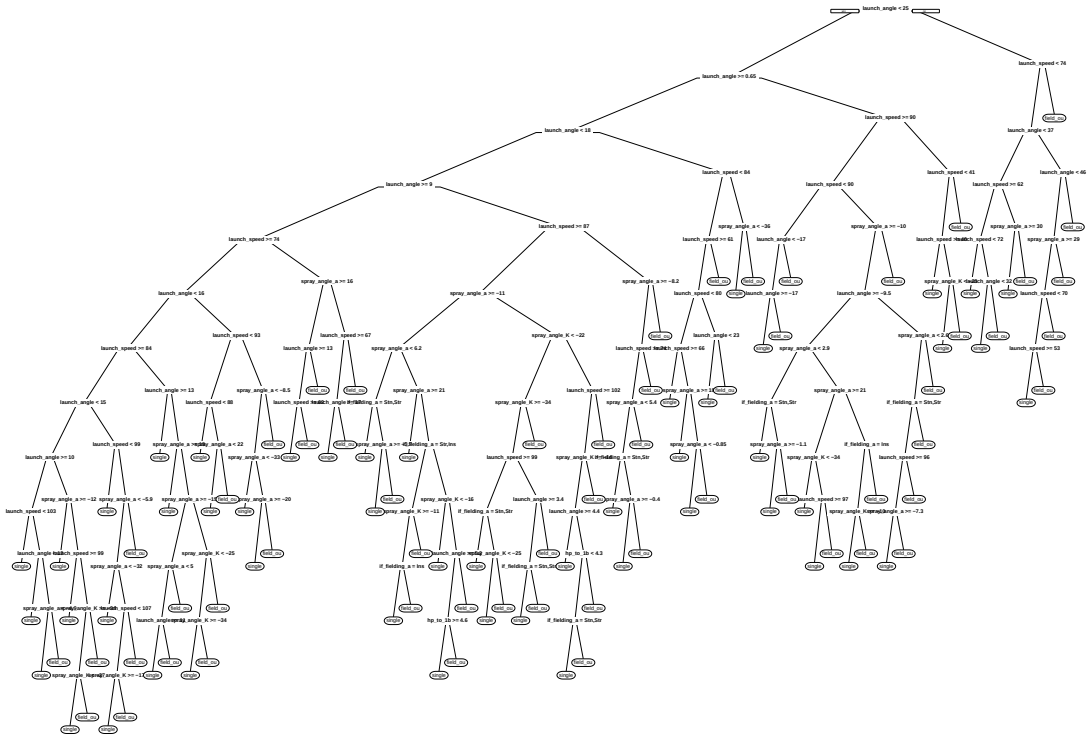
We now used our optimal cp of 0.000376506 to prune our tree. The prune() function recursively snips off the least important splits, based on the complexity parameter (cp). This helps avoid overfitting the model.

```
  # Use opt_cp to prune our classification tree.

ctree_pruned_train <- prune(ctree_train, opt_cp)
```

```
  # Review and visualize our pruned classification tree object.

print(ctree_pruned_train)
View(ctree_pruned_train)
```

```
## Error in as.data.frame.default(x): cannot coerce class '"rpart"' to a data.frame
```

```
prp(ctree_pruned_train)
```



```
# If the above plot is generated in Rstudio, the text can be made
# readable by exporting it and saving it as a pdf file that can be enlarged.
```

We then used our pruned classification tree (*ctree_pruned_train*) to predict outcomes based on our test set predictors.

```
test_set3 <- dplyr::select(test_set2, events, launch_angle, launch_speed,
                           spray_angle_Kolp, spray_angle_adj, if_fielding_alignment,
                           hp_to_1b)

ctree_pruned_predict <- predict(ctree_pruned_train, newdata = test_set3[, -1],
                                type = "class")

# Assess the performance of the pruned classification tree.

table(predicted = ctree_pruned_predict, actual = test_set3$events)
```

```
##            actual
## predicted   single field_out
##   single      3171       950
##   field_out   1810     14187
```

```
(ctree_pruned_tpr <- round(sensitivity(ctree_pruned_predict,
                                       reference = factor(test_set3$events)), 6))
```

```
## [1] 0.636619
```

```
(ctree_pruned_tnr <- round(specificity(ctree_pruned_predict,
                                       reference = factor(test_set3$events)), 6))
```

```
## [1] 0.93724
```

```
(ctree_pruned_fpr <- round(1 - ctree_pruned_tnr, 6))
```

```
## [1] 0.06276
```

```
(ctree_pruned_truescore <- round((2 * ctree_pruned_tpr * ctree_pruned_tnr) /
                                   (ctree_pruned_tpr + ctree_pruned_tnr) ,6))
```

```
## [1] 0.758219
```

```
(ctree_pruned_distance <- round(sqrt((1 - ctree_pruned_tpr)^2 +
                                       (ctree_pruned_fpr)^2), 6))
```

```
## [1] 0.368761
```

```
assessment_results <- tibble(Model = c("Baseline", "Log. Regression",
                                       "Log. Regression", "KNN",
                                       "Class. Tree"),
                             `Cutoff Method` = c(NA, "Truescore", "ROC Distance",
                                                 NA, NA),
                             `Truescore` = c(bl_truescore, max_truescore,
                                             min_distance_truescore,
                                             knn_truescore, ctree_pruned_truescore),
                             TPR = c(bl_tpr, max_truescore_tpr, min_distance_tpr,
                                     knn_tpr, ctree_pruned_tpr),
                             TNR = c(bl_tnr, max_truescore_tnr, min_distance_tnr,
                                     knn_tnr, ctree_pruned_tnr),
                             ROC_Distance = c(NA, max_truescore_distance, min_distance,
                                              knn_distance, ctree_pruned_distance),
                             FPR = c((1 - bl_tnr), max_truescore_fpr, min_distance_fpr,
                                     knn_fpr, ctree_pruned_fpr),
                             `Best Cutoff` = c(NA, max_truescore_cutoff,
                                               min_distance_cutoff, NA, NA))
knitr::kable(assessment_results[1:5, ], caption = "Assessment Results")
```

Table 4: Assessment Results

Model	Cutoff Method	Truescore	TPR	TNR	ROC_Distance	FPR	Best Cutoff
Baseline	NA	0.385984	0.259787	0.750611	NA	0.249389	NA
Log. Regression	Truescore	0.408139	0.383256	0.436216	0.835599	0.563784	0.749984
Log. Regression	ROC Distance	0.399327	0.328448	0.509216	0.831776	0.490784	0.764628
KNN	NA	0.749196	0.635615	0.912202	0.374813	0.087798	NA
Class. Tree	NA	0.758219	0.636619	0.937240	0.368761	0.062760	NA

Our Classification Tree Model marginally outperformed our Knn model by both measures of predictive ability, Maximum Truescore and Minimum Distance! We finally had some momentum.

One nice feature of rpart is that it explicitly computes the importance of each predictor variable, and then scales the values to sum to 100. This scaled version of variable importance can be obtained by using rpart's summary function; unfortunately, the output is so lengthy that it's best to save it to a file in order to review it. The file argument in the summary function allows you to specify that the output should be saved to file.

```
  # Review the Variable Importance computations from the summary() function, scaled to
  # sum to 100.

ctree_pruned_train_var_imp <- tibble(Predictor = c("launch_angle", "launch_speed",
                                                   "spray_angle_Kolp", "spray_angle_adj",
                                                   "if_fielding_alignment", "hp_to_1b"),
                                     `ctree Importance` = c(48, 32, 4, 13, 2, "< 1"))
knitr::kable(ctree_pruned_train_var_imp[1:6, ],
             caption = "Variable Importance: Pruned Classification Tree")
```

Table 5: Variable Importance: Pruned Classification Tree

Predictor	ctree Importance
launch_angle	48
launch_speed	32
spray_angle_Kolp	4
spray_angle_adj	13
if_fielding_alignment	2
hp_to_1b	< 1

Our data exploration suggested the same order of importance, but it was a bit surprising to see launch_angle and launch_speed accounting for so much as 80% of the model's predictive ability. Still, the Spray Angle predictors combined to account for nearly all of the remaining 20%, not an insignificant portion. We had to do a great deal of data transformation to get the Spray Angle concept quantified in such a way that it could be incorporated in a model. We see the full integration of Spray Angle in the Baseball Savant arena as a worthwhile area for improvement.

Build and Assess a Random Forest Model

A random forest algorithm creates many classification trees. Each tree is constructed using a random sample (with replacement) of N observations from the original data; this sample serves as the training set. In addition, a random selection of m predictor variables is used to build each tree. N is typically the number of observations in the original training set, and m is a constant that's less than the total number of predictor variables. At the end of the process, each tree essentially votes for a particular class. The forest's prediction is the class receiving the most votes.

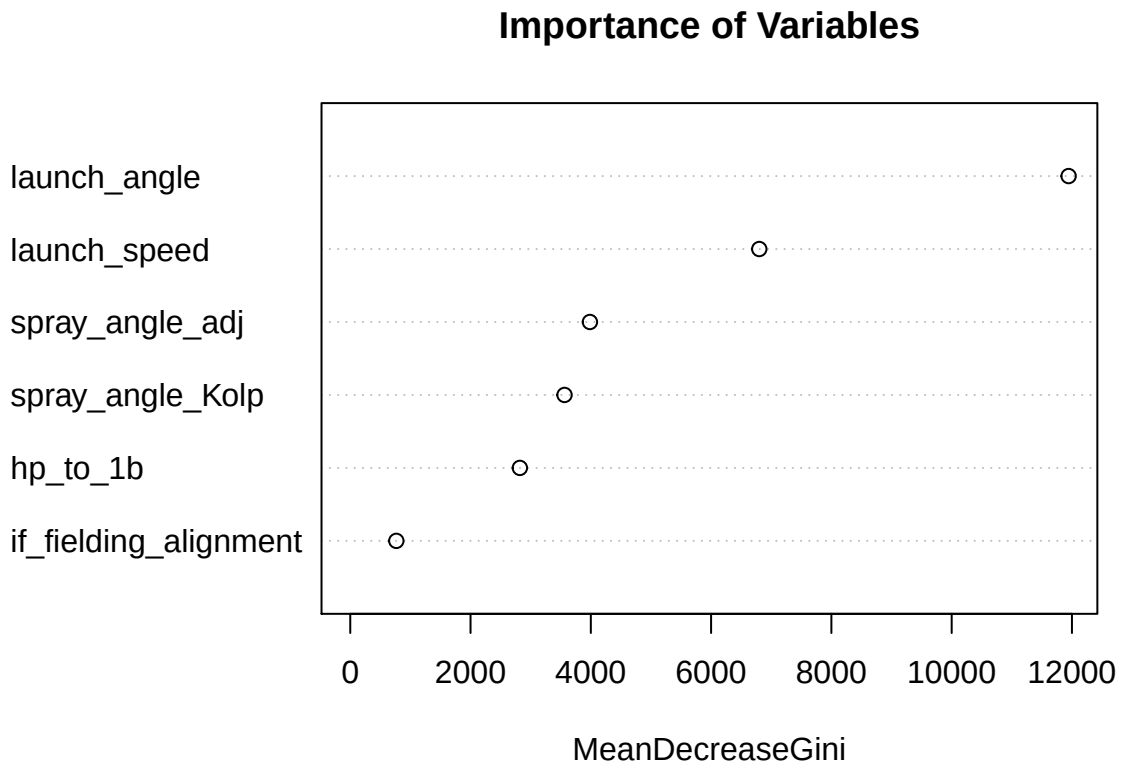
Given its design, a random forest algorithm ought to outperform a single classification tree. To see whether this was true, we first trained a random forest model.

```
# Fit a random forest model to our training data. Use the randomForest function in
# the randomForest package.

library(randomForest)
```

```r
set.seed(1)
(rforest_train <- randomForest(events ~ ., data = train_set3, importance = TRUE))
```

```
##
## Call:
##  randomForest(formula = events ~ ., data = train_set3, importance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 13.09%
## Confusion matrix:
##           single field_out class.error
## single     13496      6424  0.32248996
## field_out   4105     56441  0.06779969
```

```r
# Which variables did the randomForest algorithm consider to be the most important?

varImpPlot(rforest_train, sort = TRUE, type = 2, main = "Importance of Variables")
```




```r
importance(rforest_train, type = 2)
```

```
##                      MeanDecreaseGini
## launch_angle                11944.7235
```

```
## launch_speed                6800.9150
## spray_angle_Kolp            3562.7599
## spray_angle_adj             3984.9930
## if_fielding_alignment        766.0971
## hp_to_1b                    2820.0616
```

```
# Compare variable importance as estimated by our classification tree model and
# our random forest model.

rforest_train_var_imp <- as_tibble(importance(rforest_train, type = 2))
rforest_train_var_imp <- mutate(rforest_train_var_imp, rforest_imp =
         round((MeanDecreaseGini / sum(rforest_train_var_imp$MeanDecreaseGini)) * 100))

variable_imp <- tibble(Predictor = c("launch_angle", "launch_speed", "spray_angle_Kolp",
                                     "spray_angle_adj", "if_fielding_alignment",
                                     "hp_to_1b"),
                       `ctree` = c(48, 32, 4, 13, 2, "< 1"),
                       `rforest` = rforest_train_var_imp$rforest_imp)

knitr::kable(variable_imp[1:6, ], caption = "Variable Importance by Model")
```

Table 6: Variable Importance by Model

Predictor	ctree	rforest
launch_angle	48	40
launch_speed	32	23
spray_angle_Kolp	4	12
spray_angle_adj	13	13
if_fielding_alignment	2	3
hp_to_1b	< 1	9

Comparing the two models in terms of variable importance, our new Random Forest Model accorded more importance to *spray_angle_Kolp* and *hp_to_1b*, and less importance to *launch_angle* and *launch_speed*. The differences are fairly significant. We considered the proportions computed by Random Forest to be more in line with what we would have ideally hoped for, with all the predictors making more equal and meaningful contributions. *launch_angle* is still the most influential variable for predicting singles, as it should be. And *hp_to_1b* being accorded 9% significance justifies its inclusion in the study.

It was time to let our Random Forest Model make some predictions with the test set.

```
# Use the randomForest algorithm to predict "single" or "field_out" based on our test
# data.

rforest_predict <- predict(rforest_train, newdata = test_set3, type = "response")
rforest_predict_prob <- predict(rforest_train, newdata = test_set3, type = "prob")

# Assess the predictive ability of the randomForest algorithm.

confusionMatrix(rforest_predict, test_set3$events)
```

```
## Confusion Matrix and Statistics
##
```

```
##            Reference
## Prediction  single field_out
##   single      3445       977
##   field_out   1536     14160
##
##                Accuracy : 0.8751
##                  95% CI : (0.8704, 0.8796)
##     No Information Rate : 0.7524
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6516
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.6916
##             Specificity : 0.9355
##          Pos Pred Value : 0.7791
##          Neg Pred Value : 0.9021
##              Prevalence : 0.2476
##          Detection Rate : 0.1712
##    Detection Prevalence : 0.2198
##       Balanced Accuracy : 0.8135
##
##        'Positive' Class : single
##
```

```
(rforest_tpr <- round(sensitivity(rforest_predict,
                                  reference = factor(test_set3$events)), 6))
```

```
## [1] 0.691628
```

```
(rforest_tnr <- round(specificity(rforest_predict,
                                  reference = factor(test_set3$events)), 6))
```

```
## [1] 0.935456
```

```
(rforest_fpr <- round(1 - rforest_tnr, 6))
```

```
## [1] 0.064544
```

```
(rforest_truescore <- round((2 * rforest_tpr * rforest_tnr) /
                              (rforest_tpr + rforest_tnr) , 6))
```

```
## [1] 0.795272
```

```
(rforest_distance <- round(sqrt((1 - rforest_tpr)^2 + (rforest_fpr)^2), 6))
```

```
## [1] 0.315054
```

```
assessment_results <- tibble(Model = c("Baseline", "Log. Regression", "Log.
                                       Regression", "KNN",
                                       "Class. Tree", "Random Forest"),
                             `Cutoff Method` = c(NA, "Truescore", "ROC Distance",
                                                 NA, NA, NA),
                             `Truescore` = c(bl_truescore, max_truescore,
                                             min_distance_truescore,
                                             knn_truescore, ctree_pruned_truescore,
                                             rforest_truescore),
                             TPR = c(bl_tpr, max_truescore_tpr, min_distance_tpr, knn_tpr,
                                     ctree_pruned_tpr, rforest_tpr),
                             TNR = c(bl_tnr, max_truescore_tnr, min_distance_tnr, knn_tnr,
                                     ctree_pruned_tnr, rforest_tnr),
                             ROC_Distance = c(NA, max_truescore_distance, min_distance,
                                              knn_distance, ctree_pruned_distance,
                                              rforest_distance),
                             FPR = c((1 - bl_tnr), max_truescore_fpr, min_distance_fpr,
                                     knn_fpr, ctree_pruned_fpr, rforest_fpr),
                             `Best Cutoff` = c(NA, max_truescore_cutoff,
                                               min_distance_cutoff, NA, NA, NA))
knitr::kable(assessment_results[1:6, ], caption = "Assessment Results")
```

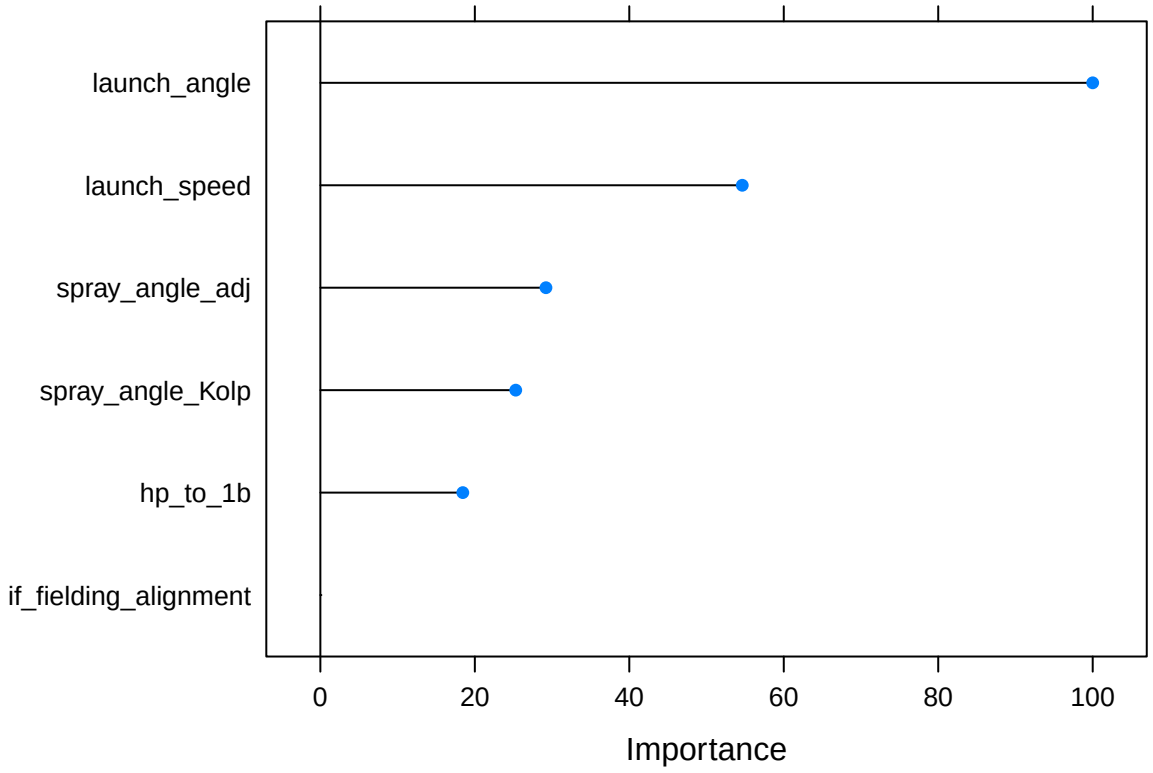
Table 7: Assessment Results

Model	Cutoff Method	Truescore	TPR	TNR	ROC_Distance	FPR	Best Cutoff
Baseline	NA	0.385984	0.259787	0.750611	NA	0.249389	NA
Log. Regression	Truescore	0.408139	0.383256	0.436216	0.835599	0.563784	0.749984
Log. Regression ROC Di	stance 0.39	9327 0.328	448 0.509	216 0	.831776 0.490	784 0.	764628
KNN	NA	0.749196	0.635615	0.912202	0.374813	0.087798	NA
Class. Tree	NA	0.758219	0.636619	0.937240	0.368761	0.062760	NA
Random Forest	NA	0.795272	0.691628	0.935456	0.315054	0.064544	NA

The randomForest algorithm clearly outperformed our classification tree, with a TPR of nearly 70% and a TNR of 94%, resulting in a Truescore of .795 and a distance of .315.

We also trained a random forest algorithm using the caret package's train function with method = "rf".

```
# Fit a random forest model to our training data. Use the train function in
# the caret package.

set.seed(200)
fitControl <- trainControl(method = "repeatedcv", number = 5, repeats = 5,
                           returnData = TRUE, returnResamp = "all",
                           savePredictions = "all",
                           summaryFunction = twoClassSummary, classProbs = TRUE)
rforest_train2 <- train(train_set3[, -1], train_set3$events, method = "rf",
                        trControl = fitControl, metric = "ROC")

# Which variables did the rf method (train function) consider to be the most important?

plot(varImp(rforest_train2))
```




```
# Use the rf method (train function) to predict "single" or "field_out" based on our test
# data.

rforest_predict2 <- predict(rforest_train2, newdata = test_set3, type = "raw")
rforest_predict2_prob <- predict(rforest_train2, newdata = test_set3, type = "prob")

# Assess the predictive ability of the rf method (train function).

confusionMatrix(rforest_predict2, test_set3$events)
```

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction  single field_out
##   single      3420       972
##   field_out   1561     14165
##
##                Accuracy : 0.8741
##                  95% CI : (0.8694, 0.8786)
##     No Information Rate : 0.7524
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6481
##
##  Mcnemar's Test P-Value : < 2.2e-16
```

```
##
##             Sensitivity : 0.6866
##             Specificity : 0.9358
##          Pos Pred Value : 0.7787
##          Neg Pred Value : 0.9007
##              Prevalence : 0.2476
##          Detection Rate : 0.1700
##    Detection Prevalence : 0.2183
##       Balanced Accuracy : 0.8112
##
##        'Positive' Class : single
##
```

```
(rforest2_tpr <- round(sensitivity(rforest_predict2,
                                   reference = factor(test_set3$events)), 6))
```

```
## [1] 0.686609
```

```
(rforest2_tnr <- round(specificity(rforest_predict2,
                                   reference = factor(test_set3$events)), 6))
```

```
## [1] 0.935786
```

```
(rforest2_fpr <- round(1 - rforest2_tnr, 6))
```

```
## [1] 0.064214
```

```
(rforest2_truescore <- round((2 * rforest2_tpr * rforest2_tnr) /
                               (rforest2_tpr + rforest2_tnr) , 6))
```

```
## [1] 0.792062
```

```
(rforest2_distance <- round(sqrt((1 - rforest2_tpr)^2 + (rforest2_fpr)^2), 6))
```

```
## [1] 0.319902
```

The random forest algorithm in the caret package produced very similar results (including variable importance) to those produced by the randomForest package.

Since random forest was our top-performing model, we chose to analyze its prediction mistakes to see if we could identify ways to improve our ability to predict singles.

```
# Create a data frame showing all erroneous predictions by our rforest_train2 algorithm,
# along with their potentially relevant characteristics.

 # Combine all predictions with actual outcomes.

rforest_predict2_results <- bind_cols(enframe(rforest_predict2),
                                      as_tibble(rforest_predict2_prob))
rforest_predict2_results <- rforest_predict2_results %>%
```

```
  dplyr::mutate(obs_nmbr = name, prob_single = single, prob_out = field_out,
                pred = value) %>%
  dplyr::select(obs_nmbr, prob_single, prob_out, pred)
rforest_predict2_v_actual <- bind_cols(rforest_predict2_results,
                                       as_tibble(test_set3$events))
rforest_predict2_v_actual$actual_events <- rforest_predict2_v_actual$value
rforest_predict2_v_actual <- rforest_predict2_v_actual %>% dplyr::select(-value)

 # Identify the mistaken predictions.
rforest_predict2_v_actual <- rforest_predict2_v_actual %>%
  mutate(eval = ifelse(rforest_predict2_v_actual$pred !=
                         rforest_predict2_v_actual$actual_events, "mistake", "-"))

 # Quantify the magnitude of each prediction mistake.

rforest_predict2_v_actual <- rforest_predict2_v_actual %>%
  mutate(prob_diff = abs(prob_single - prob_out))

 # Add additional information about each batted ball event.

rforest_predict2_v_actual <- bind_cols(rforest_predict2_v_actual, test_set[, 1:52])
rforest_predict2_v_actual <- rforest_predict2_v_actual %>%
  dplyr::select(obs_nmbr, prob_single, prob_out, pred, actual_events,   eval, prob_diff,
                game_date, batter, player_name, age, stand, position, team, events,
                des,  bb_type, adv_bb_type, launch_angle, launch_speed, launch_speed_cat,
                hc_x_Kolp, hc_y_Kolp, spray_angle_Kolp, spray_angle_Kolp_cat,
                spray_angle_adj, spray_angle_adj_cat, if_fielding_alignment,
                num_if_alignment, hp_to_1b, hp_to_1b_cat, of_fielding_alignment,
                hit_location, hit_distance_sc, launch_speed_angle, game_pk, game_year,
                game_type, home_team, away_team, pitcher, p_throws, description)

# Create a new data frame with just the mistaken predictions.

rforest_predict2_mistakes <- rforest_predict2_v_actual %>% dplyr::filter(eval == "mistake")

# Create a new data frame with just the correct predictions.

rforest_predict2_correct <- rforest_predict2_v_actual %>% dplyr::filter(eval != "mistake")
```

We had a total of 2,533 mistaken predictions out of the 20,118 batted ball events in test_set3, a 12.6% error rate.

```
# Was it more common to mistakenly predict singles or outs?

mean(rforest_predict2_mistakes$pred == "single")
```

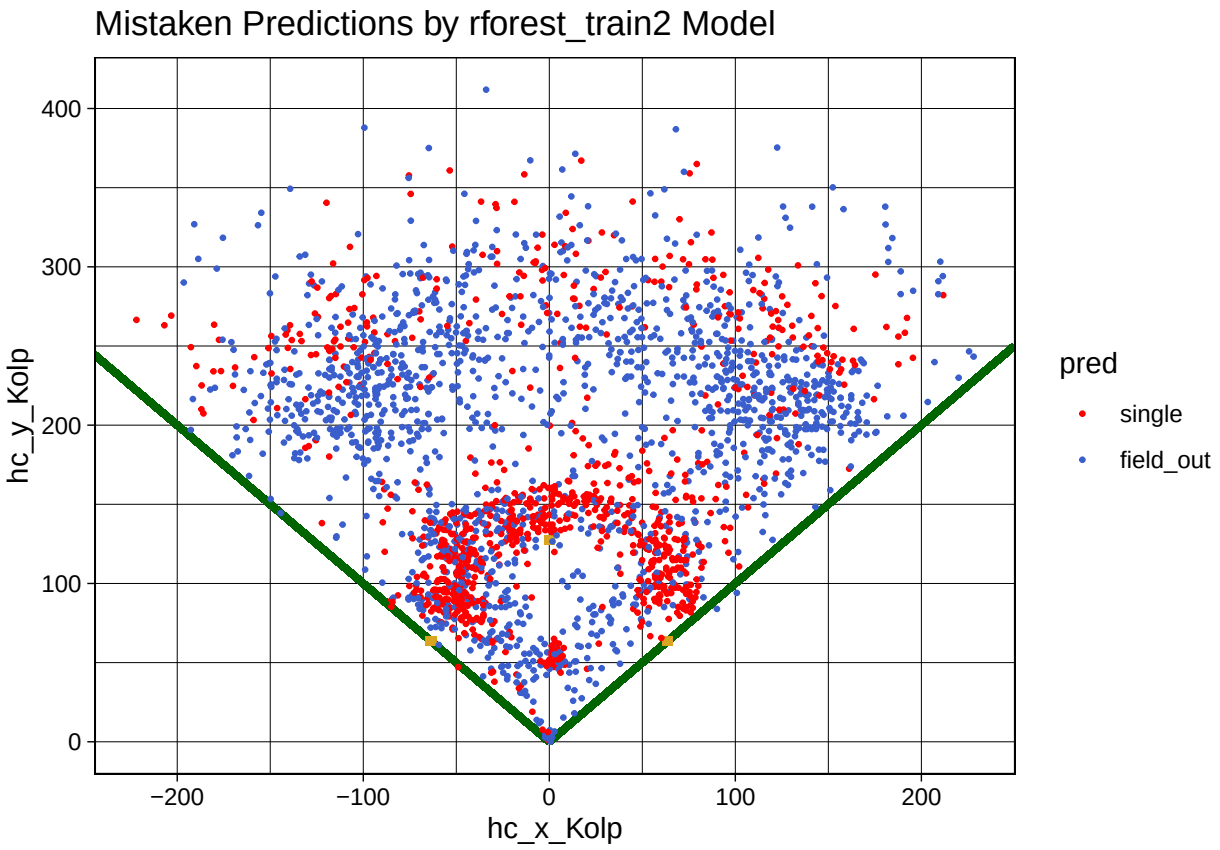
```
## [1] 0.3837347
```

```
mean(rforest_predict2_mistakes$pred == "field_out")
```

```
## [1] 0.6162653
```

Erroneous out predictions (false negatives) comprised 62% of the mistakes made by our rforest_train2 algorithm. These were batted balls that actually resulted in singles when the model predicted they were outs. They're represented by the blue points in the following plot of mistaken predictions. As you'll see, they're clustered in moderately shallow left field and right field, as well as on the left side of the infield toward the third basemand and the hole between the shortstop and the third baseman.

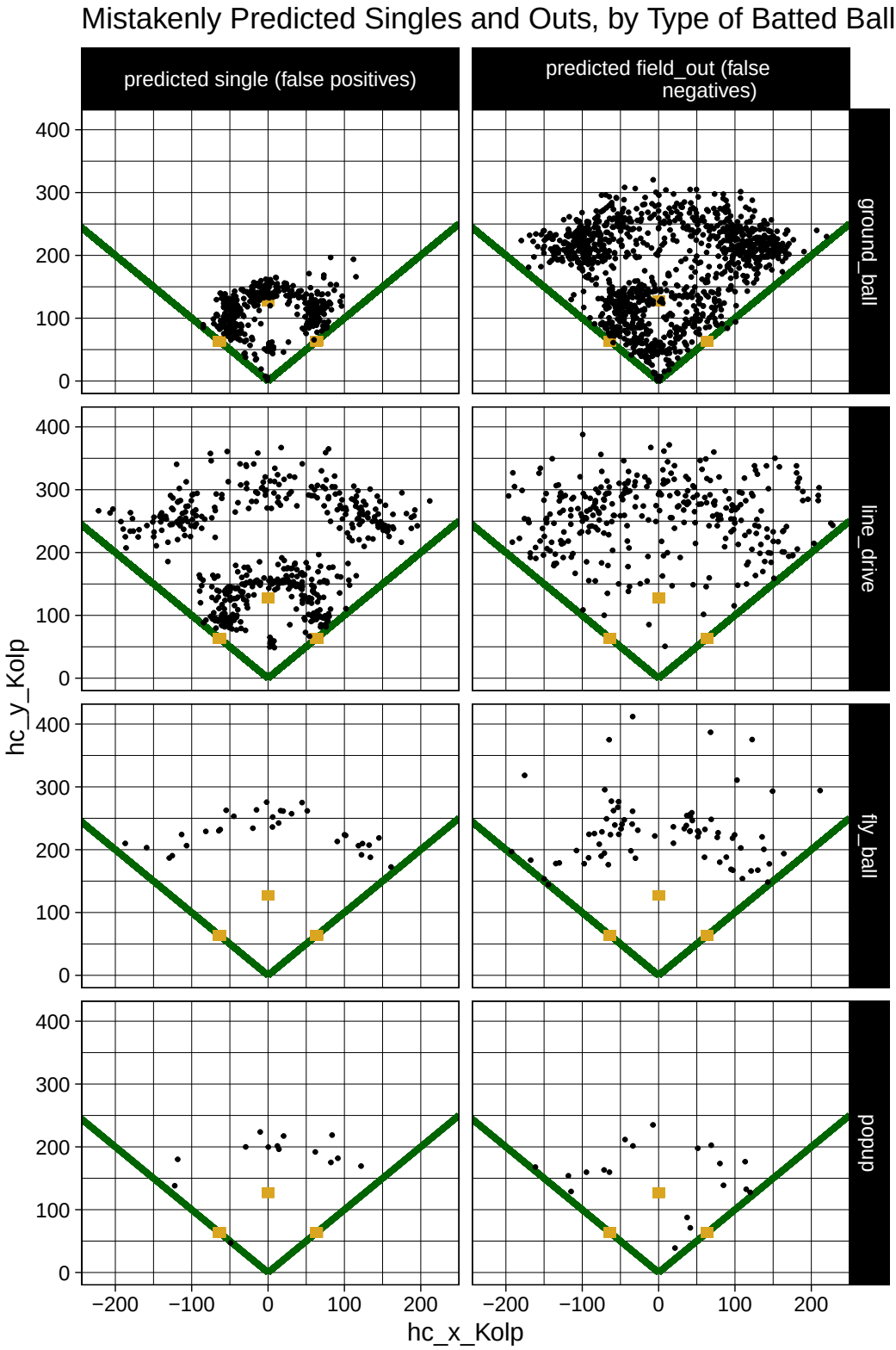
```
# Visualize erroneous predictions by rforest_train2 model.

ggplot(rforest_predict2_mistakes, aes(hc_x_Kolp, hc_y_Kolp, color = pred)) +
  scale_color_manual(values = c("red", "royalblue3")) +
  geom_segment(x = 0, y = 0, xend = 250, yend = 250, color = "dark green", size = 1.3) +
  geom_segment(x = 0, y = 0, xend = -250, yend = 250, color = "dark green", size = 1.3) +
  geom_tile(aes(x = 63.64, y = 63.64, width = 5, height = 5), color = "goldenrod",
            fill = "goldenrod") +
  geom_tile(aes(x = -63.64, y = 63.64, width = 5, height = 5), color = "goldenrod",
            fill = "goldenrod") +
  geom_tile(aes(x = 0, y = 127.28, width = 5, height = 5), color = "goldenrod",
            fill = "goldenrod") +
  geom_point(size = 0.5) + theme_linedraw() +
  labs(title = "Mistaken Predictions by rforest_train2 Model")
```



The mistaken singles predictions (false positives), less common but still significant, are represented by the red points, which are clustered all around the infield, but a bit more densely on the left side of the infield.

We wanted to find out more about the mistaken predictions, and we began by looking at launch_angle and its categorical versions, *bb_type* and *adv_bb_type*.

```r
# Visualize erroneous predictions, by type of batted ball.

pred_labels <- c("predicted single (false positives)", "predicted field_out (false
                 negatives)")
names(pred_labels) <- c("single", "field_out")

ggplot(rforest_predict2_mistakes, aes(hc_x_Kolp, hc_y_Kolp)) +
  geom_segment(x = 0, y = 0, xend = 250, yend = 250, color = "dark green", size = 1.3) +
  geom_segment(x = 0, y = 0, xend = -250, yend = 250, color = "dark green", size = 1.3) +
  geom_tile(aes(x = 63.64, y = 63.64, width = 15, height = 15), color = "goldenrod",
            fill = "goldenrod") +
  geom_tile(aes(x = -63.64, y = 63.64, width = 15, height = 15), color = "goldenrod",
            fill = "goldenrod") +
  geom_tile(aes(x = 0, y = 127.28, width = 15, height = 15), color = "goldenrod",
            fill = "goldenrod") +
  geom_point(size = 0.5) + theme_linedraw() +
  labs(title = "Mistakenly Predicted Singles and Outs, by Type of Batted Ball") +
  facet_grid(bb_type ~ pred, labeller = labeller(pred = pred_labels))
```



Most of the false negatives were ground balls. The false positives (outs that were predicted to be singles) appear a little more evenly divided between ground balls and line drives.

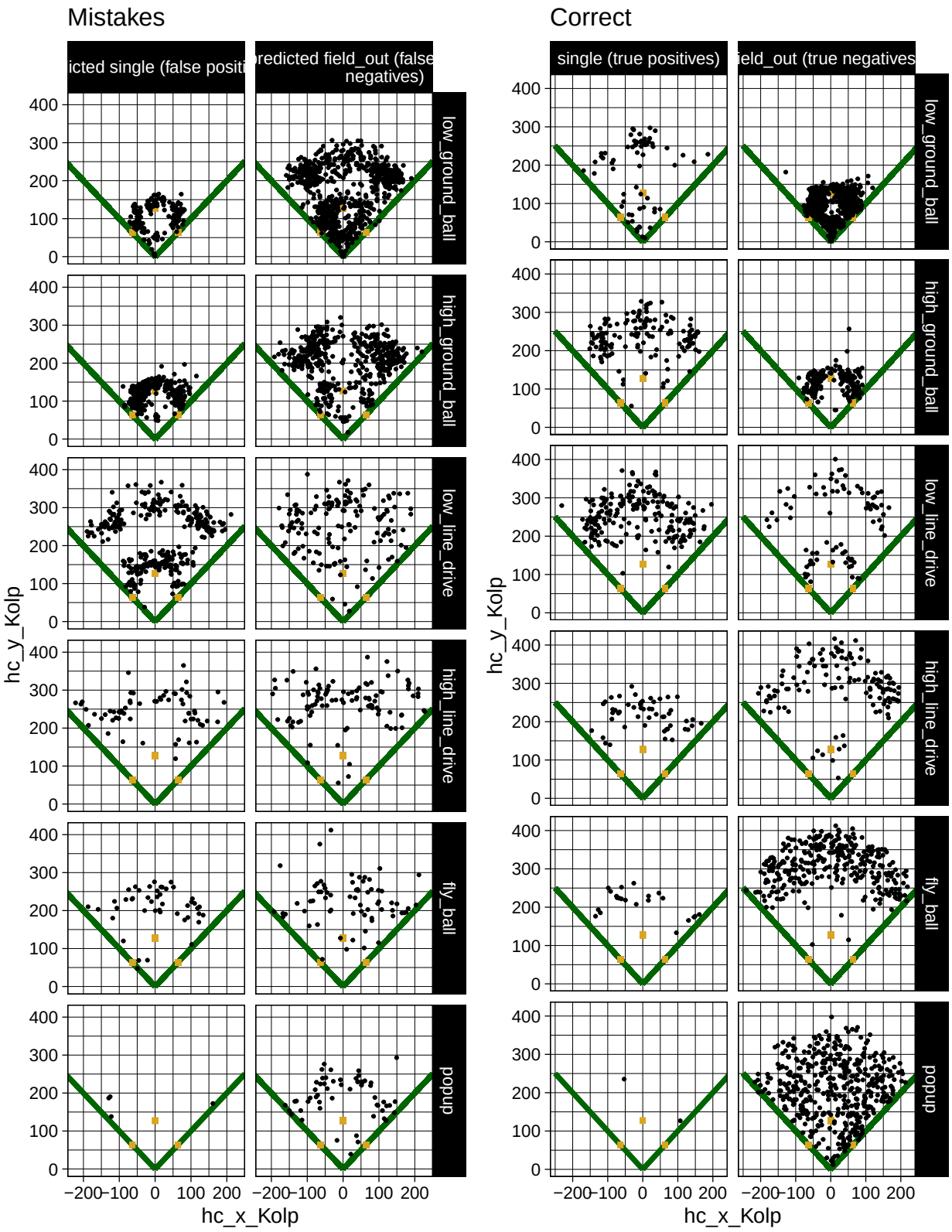
```
# Visualize erroneous predictions, by adv_bb_type.

pred_labels <- c("predicted single (false positives)", "predicted field_out (false
                 negatives)")
names(pred_labels) <- c("single", "field_out")

erroneous_predictions_by_adv_bb_type <- ggplot(rforest_predict2_mistakes,
                                               aes(hc_x_Kolp, hc_y_Kolp)) +
  geom_segment(x = 0, y = 0, xend = 250, yend = 250, color = "dark green", size = 1.3) +
  geom_segment(x = 0, y = 0, xend = -250, yend = 250, color = "dark green", size = 1.3) +
  geom_tile(aes(x = 63.64, y = 63.64, width = 15, height = 15), color = "goldenrod",
            fill = "goldenrod") +
  geom_tile(aes(x = -63.64, y = 63.64, width = 15, height = 15), color = "goldenrod",
            fill = "goldenrod") +
  geom_tile(aes(x = 0, y = 127.28, width = 15, height = 15), color = "goldenrod",
            fill = "goldenrod") +
  geom_point(size = 0.5) + theme_linedraw() +
  labs(title = "Mistakes") +
  facet_grid(adv_bb_type ~ pred, labeller = labeller(pred = pred_labels))
```

Our *adv_bb_type* variable allowed us to take a more detailed look. Here we could see that most of the false negatives were low ground balls, but there were also a fair number of high ground balls. As for the false positives, these tended to fall in the high ground ball and low line drive categories.

To determine whether any of these pictorial proportions were out of the ordinary, we compared them to a random sample of 2,533 batted balls whose outcomes were correctly predicted.

```
# Visualize correct predictions, by adv_bb_type.

set.seed(1)
rforest_predict2_correct_sample_2533 <- sample_n(rforest_predict2_correct, 2533,
                                                 replace = FALSE)

pred_labels <- c("single (true positives)", "field_out (true negatives)")
names(pred_labels) <- c("single", "field_out")

correct_predictions_by_adv_bb_type <- ggplot(rforest_predict2_correct_sample_2533,
                                             aes(hc_x_Kolp, hc_y_Kolp)) +
  geom_segment(x = 0, y = 0, xend = 250, yend = 250, color = "dark green", size = 1.3) +
  geom_segment(x = 0, y = 0, xend = -250, yend = 250, color = "dark green", size = 1.3) +
  geom_tile(aes(x = 63.64, y = 63.64, width = 15, height = 15), color = "goldenrod",
            fill = "goldenrod") +
  geom_tile(aes(x = -63.64, y = 63.64, width = 15, height = 15), color = "goldenrod",
            fill = "goldenrod") +
  geom_tile(aes(x = 0, y = 127.28, width = 15, height = 15), color = "goldenrod",
            fill = "goldenrod") +
  geom_point(size = 0.5) + theme_linedraw() +
  labs(title = "Correct") +
  facet_grid(adv_bb_type ~ pred, labeller = labeller(pred = pred_labels))
```

```
grid.arrange(erroneous_predictions_by_adv_bb_type, correct_predictions_by_adv_bb_type,
             ncol = 2)
```



As interesting as they were to look at, we found it difficult to judge the numbers and proportions from the batted ball coordinates. A different approach was called for.

We made one data frame consisting of our the 2,533 mistaken predictions by the rforest_train2 model, and

another data frame comprised of an equal number of randomly sampled correct predictions by the same model. We then joined the two data frames to facilitate our analysis.

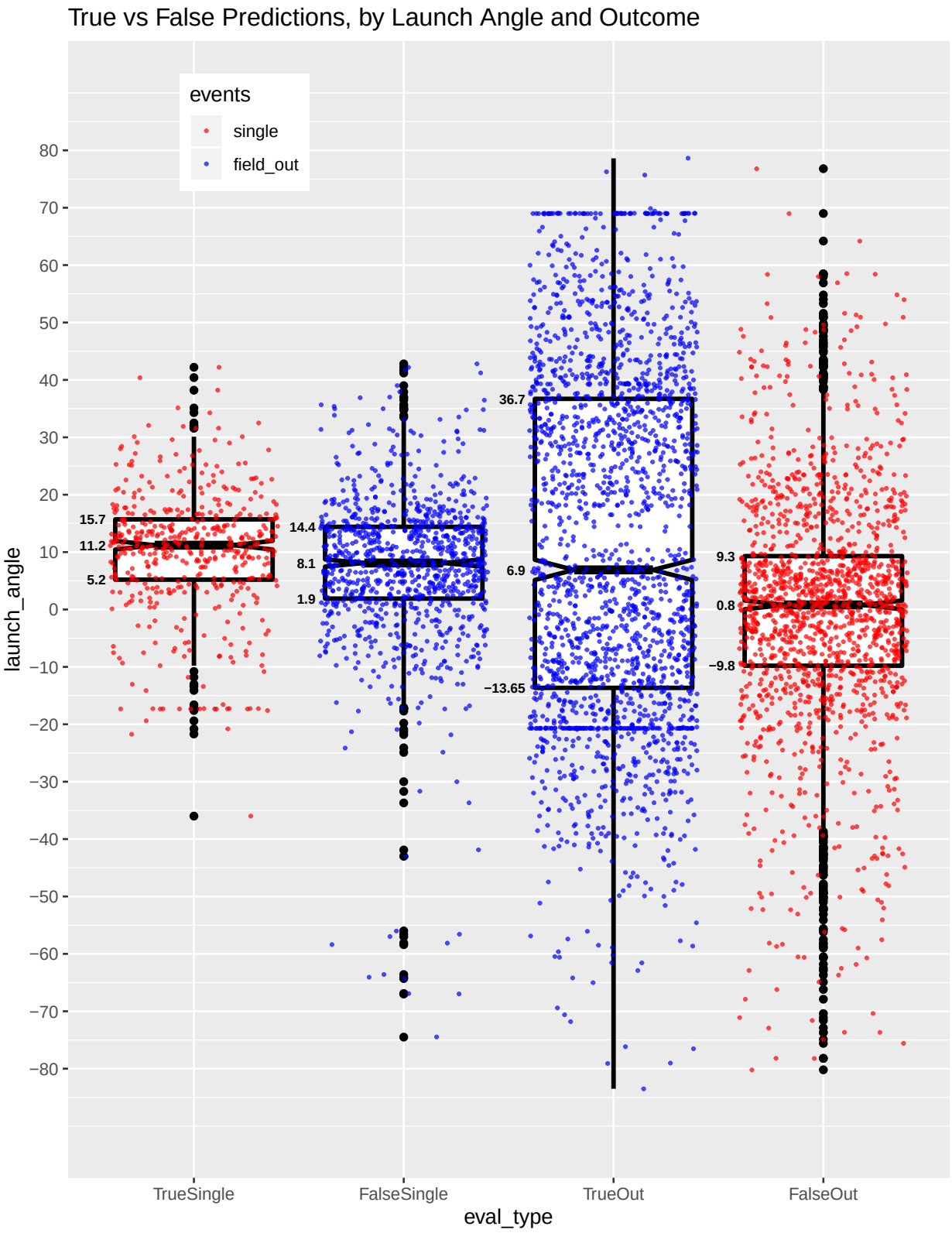
```
rforest_predict2_mistakes <- rforest_predict2_mistakes %>%
  mutate(eval_type = ifelse(pred == "single", "FalseSingle", "FalseOut"))
rforest_predict2_mistakes <- rforest_predict2_mistakes %>%
  dplyr::select(1:6, eval_type, everything())

rforest_predict2_correct_sample_2533 <- rforest_predict2_correct_sample_2533 %>%
  mutate(eval_type = ifelse(pred == "single", "TrueSingle", "TrueOut"))
rforest_predict2_correct_sample_2533 <- rforest_predict2_correct_sample_2533 %>%
  dplyr::select(1:6, eval_type, everything())

predictions_5066 <- bind_rows(rforest_predict2_mistakes,
                              rforest_predict2_correct_sample_2533)
predictions_5066$eval_type <- factor(predictions_5066$eval_type,
                                     levels = c("TrueSingle", "FalseSingle",
                                                "TrueOut", "FalseOut"))
```

The idea was to compare, for each of the model's features, the distributions of the true positives and false positives (or true singles and false singles), and the distributions of true negatives and false negatives (or true outs and false outs). This would be done with a visualization of Tukey's five number summary, in the form of box plots. We were hoping to see patterns showing the kinds of batted balls the model mistakenly took for outs, and what kinds of batted balls the model mistakenly took for singles.

```
# Compare, by launch_angle, our population of 2,533 mistaken predictions to a random
#population of 2,533 correct predictions.

truefalse_by_launch_angle <- ggplot(predictions_5066,
                                    aes(x = eval_type, y = launch_angle,
                                        color = events)) +
  geom_boxplot(color = "black", notch = TRUE, size = 1.1) +
  geom_point(position = "jitter", alpha = .7, size = 0.5) +
  scale_y_continuous(breaks = seq(-80, 80, by = 10),
                     labels = c("-80", "-70", "-60", "-50", "-40", "-30",  "-20",
                                "-10", "0", "10", "20", "30", "40", "50", "60",
                                "70", "80"),
                     limits = c(-90, 90)) +
  scale_color_manual(values = c("Red", "Blue")) +
  ggtitle("True vs False Predictions, by Launch Angle and Outcome") +
  theme(legend.position = c(.2, .92))

truefalse_by_launch_angle_data <- layer_data(truefalse_by_launch_angle)

launch_angle_TrueSingle_1quartile <- truefalse_by_launch_angle_data[1, 2]
launch_angle_TrueSingle_median <- truefalse_by_launch_angle_data[1, 3]
launch_angle_TrueSingle_3quartile <- truefalse_by_launch_angle_data[1, 4]
launch_angle_FalseSingle_1quartile <- truefalse_by_launch_angle_data[2, 2]
launch_angle_FalseSingle_median <- truefalse_by_launch_angle_data[2, 3]
launch_angle_FalseSingle_3quartile <- truefalse_by_launch_angle_data[2, 4]
launch_angle_TrueOut_1quartile <- truefalse_by_launch_angle_data[3, 2]
launch_angle_TrueOut_median <- truefalse_by_launch_angle_data[3, 3]
launch_angle_TrueOut_3quartile <- truefalse_by_launch_angle_data[3, 4]
launch_angle_FalseOut_1quartile <- truefalse_by_launch_angle_data[4, 2]
```

```
launch_angle_FalseOut_median <- truefalse_by_launch_angle_data[4, 3]
launch_angle_FalseOut_3quartile <- truefalse_by_launch_angle_data[4, 4]
```

```
(truefalse_by_launch_angle <- ggplot(predictions_5066,
                                     aes(x = eval_type, y = launch_angle,
                                         color = events)) +
  geom_boxplot(color = "black", notch = TRUE, size = 1.1) +
  geom_point(position = "jitter", alpha = .7, size = 0.5) +
  scale_y_continuous(breaks = seq(-80, 80, by = 10),
                     labels = c("-80", "-70", "-60", "-50", "-40", "-30",  "-20",
                                "-10", "0", "10", "20", "30", "40", "50", "60",
                                "70", "80"),
                     limits = c(-90, 90)) +
  scale_color_manual(values = c("Red", "Blue")) +
  ggtitle("True vs False Predictions, by Launch Angle and Outcome") +
  theme(legend.position = c(.2, .92)) +
  annotate("text", x = c(.58, .58, .58), size = 2.5, fontface = 2, hjust = 1,
           y = c(launch_angle_TrueSingle_1quartile, launch_angle_TrueSingle_median,
                 launch_angle_TrueSingle_3quartile),
           label = c(launch_angle_TrueSingle_1quartile, launch_angle_TrueSingle_median,
                     launch_angle_TrueSingle_3quartile)) +
  annotate("text", x = c(1.58,1.58, 1.58), size = 2.5, fontface = 2, hjust = 1,
           y = c(launch_angle_FalseSingle_1quartile, launch_angle_FalseSingle_median,
                 launch_angle_FalseSingle_3quartile),
           label = c(launch_angle_FalseSingle_1quartile,
                     launch_angle_FalseSingle_median,
                     launch_angle_FalseSingle_3quartile)) +
  annotate("text", x = c(2.58, 2.58, 2.58), size = 2.5, fontface = 2, hjust = 1,
           y = c(launch_angle_TrueOut_1quartile, launch_angle_TrueOut_median,
                 launch_angle_TrueOut_3quartile),
           label = c(launch_angle_TrueOut_1quartile, launch_angle_TrueOut_median,
                     launch_angle_TrueOut_3quartile)) +
  annotate("text", x = c(3.58, 3.58, 3.58), size = 2.5, fontface = 2, hjust = 1,
           y = c(launch_angle_FalseOut_1quartile, launch_angle_FalseOut_median,
                 launch_angle_FalseOut_3quartile),
           label = c(launch_angle_FalseOut_1quartile, launch_angle_FalseOut_median,
                     launch_angle_FalseOut_3quartile)))
```



False outs were the most common mistakes made by the model, so we focused on those.

As expected, the outs correctly predicted by the model had a very dispersed interquartile range extemdomg from -13.7 degrees to 36.7. But the singles that it incorrectly predicted were outs appeared in a more compact

range essentially centered at 0 and extending 10 degrees up and down. These were low ground balls and high ground balls according to our adv_bb_type variable. It's easy to see how any model would have a more difficult time distinguishing ground ball singles from ground ball outs. Our exit velocity feature should have helped in this regard, but the location of the infielders also has a significant influence. Our features that attempted to capture infielders' locations were not as granular and accurate as they could have been with better data.

Instead of just looking at all the prediction mistakes (which included a lot of close calls), we also decided to zero in on just the missed outs and singles that the model predicted had a 30% or less probability of occurring; in other words, the 1,106 worst prediction mistakes. These comprised about 44% of all the model's mistakes, and we hoped that by focusing on them, we might gain further insight. Again, we would be comparing them to an equal number of correct predictions.

```
# Analyze the worst prediction mistakes (1,106 outcomes with a predicted probability of
# 30% or less).

rforest_predict2_worstMistakes <- rforest_predict2_mistakes %>%
  dplyr::filter(prob_diff >= 0.4)

rforest_predict2_worstMistakes <- rforest_predict2_worstMistakes %>%
  mutate(eval_type = ifelse(pred == "single", "FalseSingle", "FalseOut"))
rforest_predict2_worstMistakes <- rforest_predict2_worstMistakes %>%
  dplyr::select(1:6, eval_type, everything())

set.seed(1)
rforest_predict2_correct_sample_1106 <- sample_n(rforest_predict2_correct, 1106,
                                                 replace = FALSE)

rforest_predict2_correct_sample_1106 <- rforest_predict2_correct_sample_1106 %>%
  mutate(eval_type = ifelse(pred == "single", "TrueSingle", "TrueOut"))
rforest_predict2_correct_sample_1106 <- rforest_predict2_correct_sample_1106 %>%
  dplyr::select(1:6, eval_type, everything())

predictions_2212 <- bind_rows(rforest_predict2_worstMistakes,
                              rforest_predict2_correct_sample_1106)
predictions_2212$eval_type <- factor(predictions_2212$eval_type,
                                     levels = c("TrueSingle", "FalseSingle",
                                                "TrueOut", "FalseOut"))
```

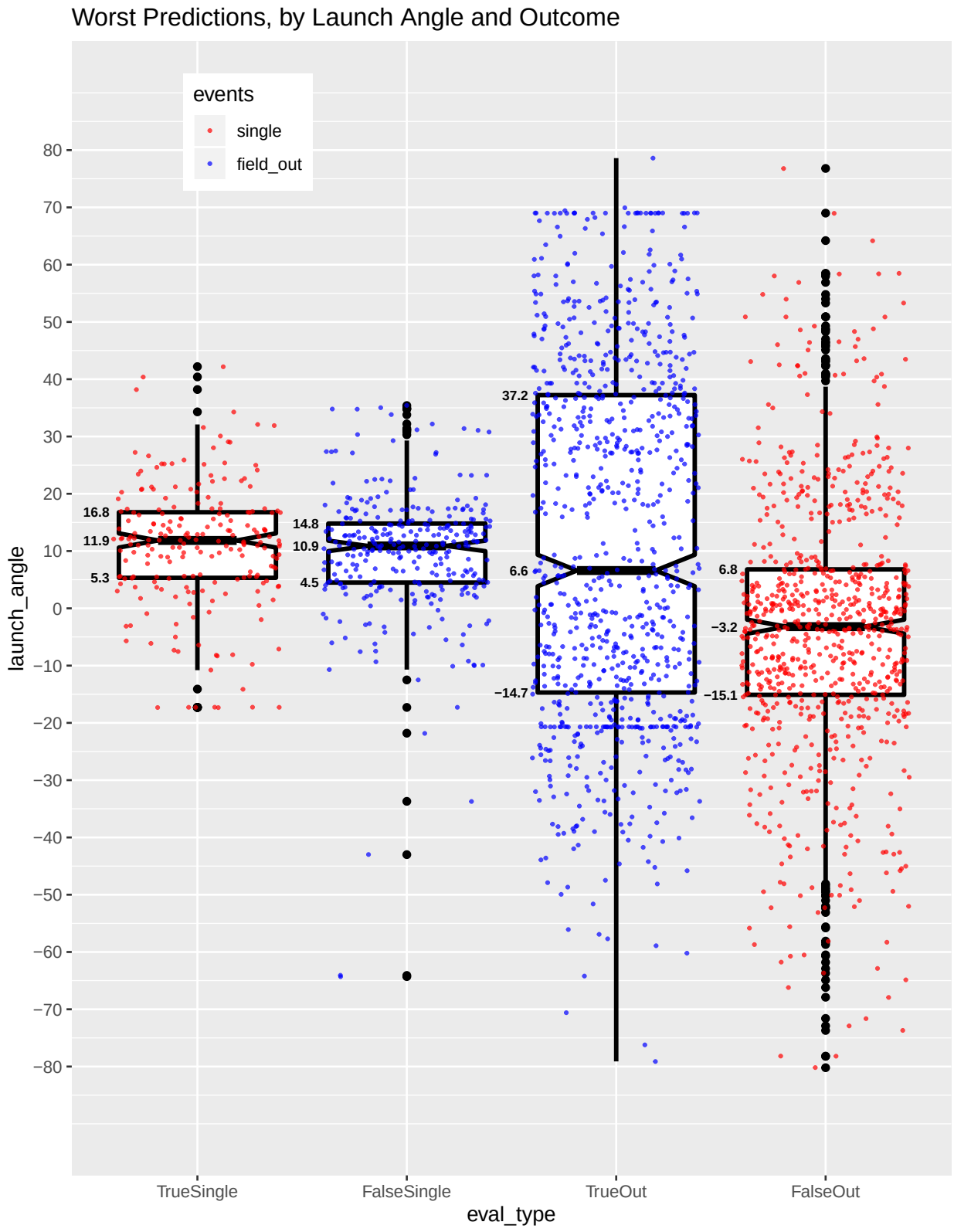
What kinds of launch angles did the worst mistakes have?

```
# Compare, by launch_angle, our population of the worst 1,106 prediction mistakes to a
# random population of 1,106 correct predictions.

truefalse_by_launch_angle <- ggplot(predictions_2212,
                                    aes(x = eval_type, y = launch_angle,
                                        color = events)) +
  geom_boxplot(color = "black", notch = TRUE, size = 1.1) +
  geom_point(position = "jitter", alpha = .7, size = 0.5) +
  scale_y_continuous(breaks = seq(-80, 80, by = 10),
                     labels = c("-80", "-70", "-60", "-50", "-40", "-30",  "-20",
                                "-10", "0", "10", "20", "30", "40", "50", "60",
                                "70", "80"),
                     limits = c(-90, 90)) +
```

```
  scale_color_manual(values = c("Red", "Blue")) +
  ggtitle("Worst Predictions, by Launch Angle and Outcome") +
  theme(legend.position = c(.2, .92))

truefalse_by_launch_angle_data <- layer_data(truefalse_by_launch_angle)

launch_angle_TrueSingle_1quartile <- round(truefalse_by_launch_angle_data[1, 2], 1)
launch_angle_TrueSingle_median <- round(truefalse_by_launch_angle_data[1, 3], 1)
launch_angle_TrueSingle_3quartile <- round(truefalse_by_launch_angle_data[1, 4], 1)
launch_angle_FalseSingle_1quartile <- round(truefalse_by_launch_angle_data[2, 2], 1)
launch_angle_FalseSingle_median <- round(truefalse_by_launch_angle_data[2, 3], 1)
launch_angle_FalseSingle_3quartile <- round(truefalse_by_launch_angle_data[2, 4], 1)
launch_angle_TrueOut_1quartile <- round(truefalse_by_launch_angle_data[3, 2], 1)
launch_angle_TrueOut_median <- round(truefalse_by_launch_angle_data[3, 3], 1)
launch_angle_TrueOut_3quartile <- round(truefalse_by_launch_angle_data[3, 4], 1)
launch_angle_FalseOut_1quartile <- round(truefalse_by_launch_angle_data[4, 2], 1)
launch_angle_FalseOut_median <- round(truefalse_by_launch_angle_data[4, 3], 1)
launch_angle_FalseOut_3quartile <- round(truefalse_by_launch_angle_data[4, 4], 1)
```

```
(truefalse_by_launch_angle <- ggplot(predictions_2212,
                                     aes(x = eval_type, y = launch_angle,
                                         color = events)) +
    geom_boxplot(color = "black", notch = TRUE, size = 1.1) +
    geom_point(position = "jitter", alpha = .7, size = 0.5) +
    scale_y_continuous(breaks = seq(-80, 80, by = 10),
                       labels = c("-80", "-70", "-60", "-50", "-40", "-30",
                                  "-20", "-10", "0", "10", "20", "30", "40", "50", "60",
                                  "70", "80"),
                       limits = c(-90, 90)) +
    scale_color_manual(values = c("Red", "Blue")) +
    ggtitle("Worst Predictions, by Launch Angle and Outcome") +
    theme(legend.position = c(.2, .92)) +
    annotate("text", x = c(.58, .58, .58), size = 2.5, fontface = 2, hjust = 1,
             y = c(launch_angle_TrueSingle_1quartile, launch_angle_TrueSingle_median,
                   launch_angle_TrueSingle_3quartile),
             label = c(launch_angle_TrueSingle_1quartile,
                       launch_angle_TrueSingle_median,
                       launch_angle_TrueSingle_3quartile)) +
    annotate("text", x = c(1.58,1.58, 1.58), size = 2.5, fontface = 2, hjust = 1,
             y = c(launch_angle_FalseSingle_1quartile, launch_angle_FalseSingle_median,
                   launch_angle_FalseSingle_3quartile),
             label = c(launch_angle_FalseSingle_1quartile,
                       launch_angle_FalseSingle_median,
                       launch_angle_FalseSingle_3quartile)) +
    annotate("text", x = c(2.58, 2.58, 2.58), size = 2.5, fontface = 2, hjust = 1,
             y = c(launch_angle_TrueOut_1quartile, launch_angle_TrueOut_median,
                   launch_angle_TrueOut_3quartile),
             label = c(launch_angle_TrueOut_1quartile, launch_angle_TrueOut_median,
                       launch_angle_TrueOut_3quartile)) +
    annotate("text", x = c(3.58, 3.58, 3.58), size = 2.5, fontface = 2, hjust = 1,
             y = c(launch_angle_FalseOut_1quartile, launch_angle_FalseOut_median,
                   launch_angle_FalseOut_3quartile),
             label = c(launch_angle_FalseOut_1quartile, launch_angle_FalseOut_median,
```

```
launch_angle_FalseOut_3quartile)))
```

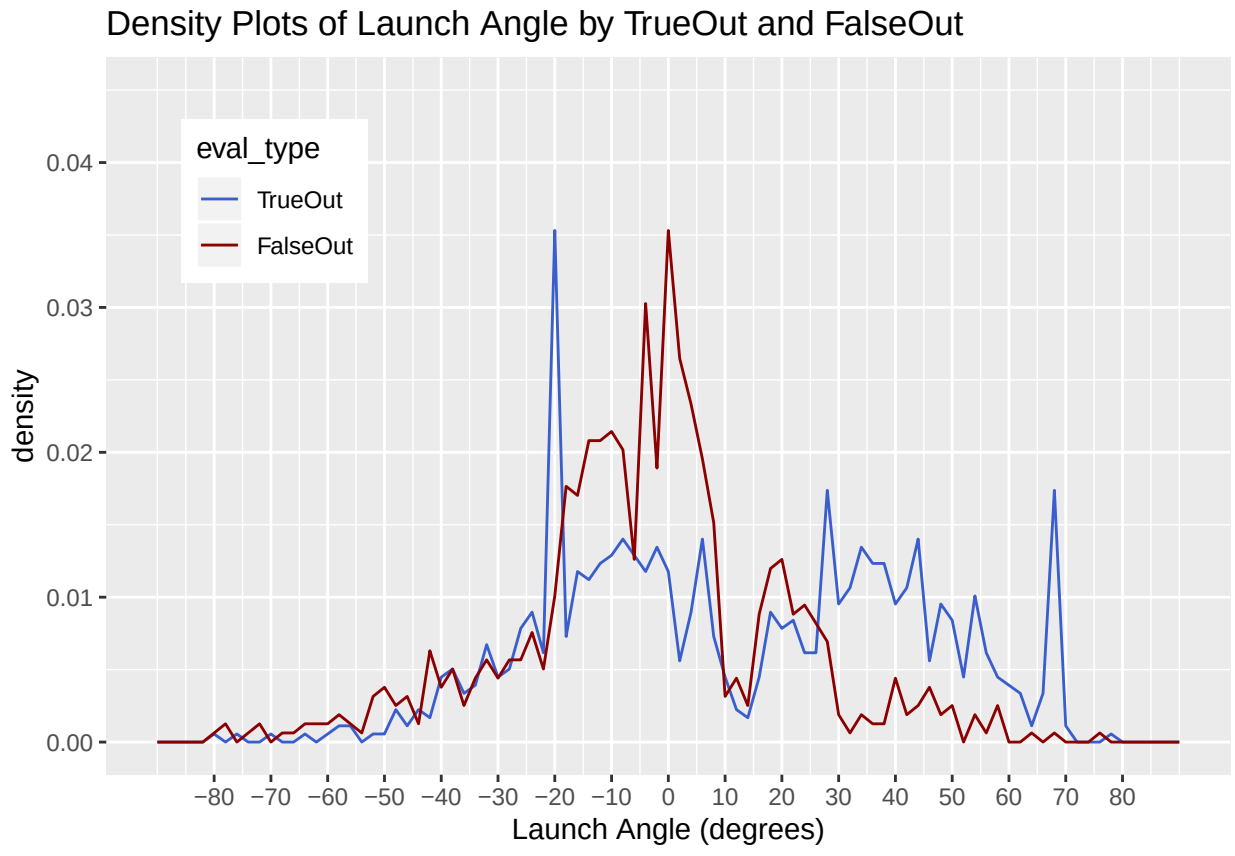


Here, the worst false outs fell into an even lower range of launch angles, which meant the mistakes skewed

more toward low ground balls than high ground balls. So the model had an even more difficult time making single-out predictions for low ground balls having launch angles as low as -15 degrees.

Lastly, we wanted to see what density plots would show.

```
# Compare the densities of launch_angle, using our population of the worst 1,106
# prediction mistakes and a random population of 1,106 correct predictions.

x_scale <- scale_x_continuous(breaks = seq(-80, 80, by = 10),
                              labels = c("-80", "-70", "-60", "-50", "-40", "-30",
                                         "-20", "-10", "0", "10", "20", "30", "40",
                                         "50", "60", "70", "80"),
                              limits = c(-90, 90))
y_scale <- scale_y_continuous(limits = c(0, 0.045))
(la_density <- ggplot(dplyr::filter(predictions_2212, eval_type == "TrueOut" |
                                    eval_type == "FalseOut"),
                      aes(x = launch_angle, y = stat(density), color = eval_type)) +
    geom_freqpoly(binwidth = 2) +
    x_scale +
    y_scale +
    scale_color_manual(values = c(FalseOut = "red4", TrueOut = "royalblue3")) +
    labs(title = "Density Plots of Launch Angle by TrueOut and FalseOut",
         x = "Launch Angle (degrees)")) +
  theme(legend.position = c(.15, .8))
```



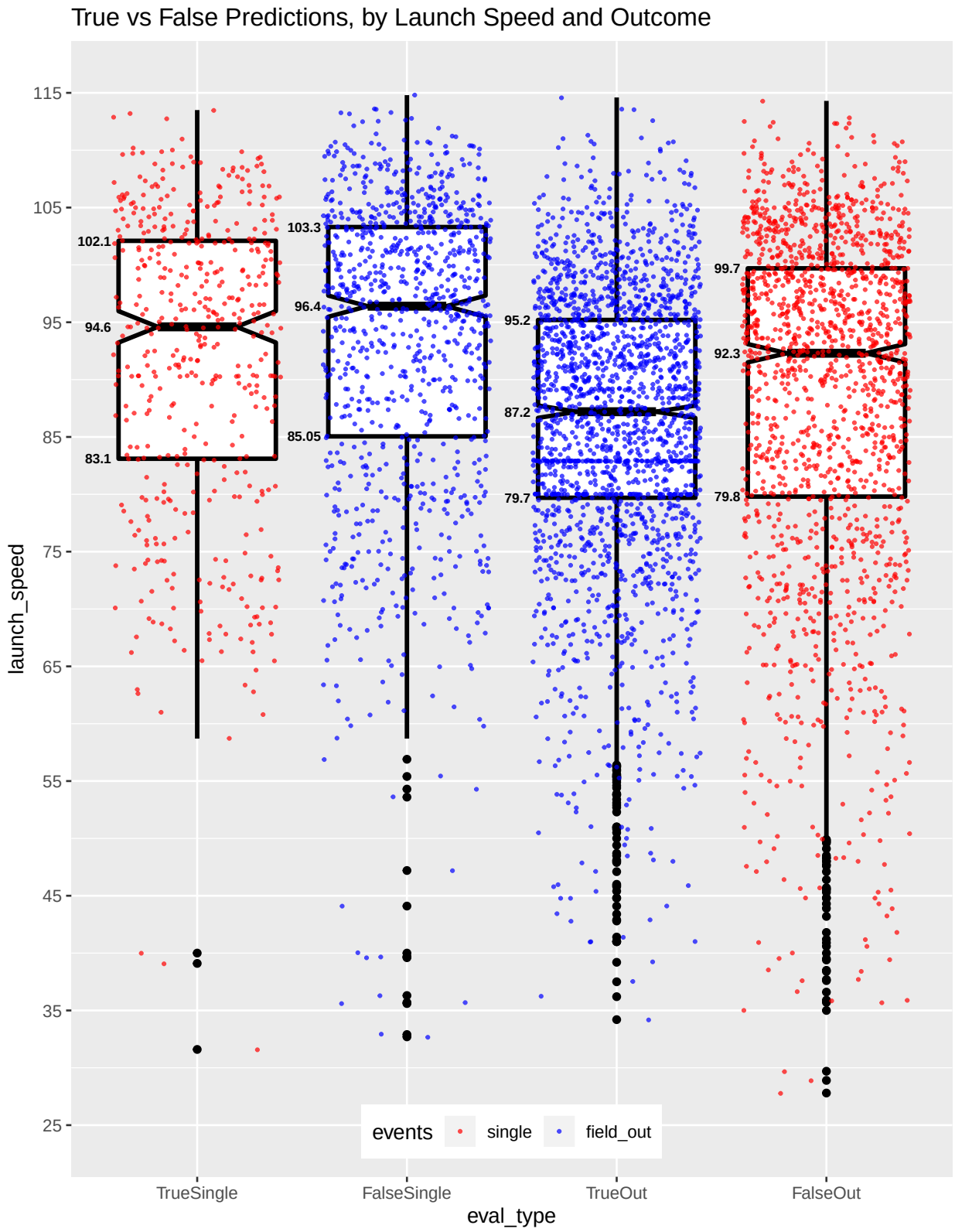
The density plots show the model having the same difficulty with launch angles between -15 and 7 degrees.

We moved on to the next feature, Launch Speed (Exit Velocity).

```
# Compare, by launch_speed, our population of 2,533 mistaken predictions to a random
# population of 2,533 correct predictions.

truefalse_by_launch_speed <- ggplot(predictions_5066,
                                    aes(x = eval_type, y = launch_speed,
                                        color = events)) +
  geom_boxplot(color = "black", notch = TRUE, size = 1.1) +
  geom_point(position = "jitter", alpha = .7, size = 0.5) +
  scale_y_continuous(breaks = seq(25, 115, by = 10),
                     labels = c("25", "35", "45", "55", "65", "75",  "85", "95", "105",
                                "115"),
                     limits = c(25, 115)) +
  scale_color_manual(values = c("Red", "Blue")) +
  ggtitle("True vs False Predictions, by Launch Speed and Outcome") +
  theme(legend.position = c(.5, .04), legend.direction = "horizontal")

truefalse_by_launch_speed_data <- layer_data(truefalse_by_launch_speed)

launch_speed_TrueSingle_1quartile <- truefalse_by_launch_speed_data[1, 2]
launch_speed_TrueSingle_median <- truefalse_by_launch_speed_data[1, 3]
launch_speed_TrueSingle_3quartile <- truefalse_by_launch_speed_data[1, 4]
launch_speed_FalseSingle_1quartile <- truefalse_by_launch_speed_data[2, 2]
launch_speed_FalseSingle_median <- truefalse_by_launch_speed_data[2, 3]
launch_speed_FalseSingle_3quartile <- truefalse_by_launch_speed_data[2, 4]
launch_speed_TrueOut_1quartile <- truefalse_by_launch_speed_data[3, 2]
launch_speed_TrueOut_median <- truefalse_by_launch_speed_data[3, 3]
launch_speed_TrueOut_3quartile <- truefalse_by_launch_speed_data[3, 4]
launch_speed_FalseOut_1quartile <- truefalse_by_launch_speed_data[4, 2]
launch_speed_FalseOut_median <- truefalse_by_launch_speed_data[4, 3]
launch_speed_FalseOut_3quartile <- truefalse_by_launch_speed_data[4, 4]
```

```
(truefalse_by_launch_speed <- ggplot(predictions_5066,
                                     aes(x = eval_type, y = launch_speed,
                                         color = events)) +
    geom_boxplot(color = "black", notch = TRUE, size = 1.1) +
    geom_point(position = "jitter", alpha = .7, size = 0.5) +
    scale_y_continuous(breaks = seq(25, 115, by = 10),
                       labels = c("25", "35", "45", "55", "65", "75",  "85", "95",
                                  "105", "115"),
                       limits = c(25, 115)) +
    scale_color_manual(values = c("Red", "Blue")) +
    ggtitle("True vs False Predictions, by Launch Speed and Outcome") +
    theme(legend.position = c(.5, .04), legend.direction = "horizontal") +
    annotate("text", x = c(.59, .59, .59), size = 2.5, fontface = 2, hjust = 1,
             y = c(launch_speed_TrueSingle_1quartile, launch_speed_TrueSingle_median,
                   launch_speed_TrueSingle_3quartile),
             label = c(launch_speed_TrueSingle_1quartile, launch_speed_TrueSingle_median,
                       launch_speed_TrueSingle_3quartile)) +
    annotate("text", x = c(1.59,1.59, 1.59), size = 2.5, fontface = 2, hjust = 1,
             y = c(launch_speed_FalseSingle_1quartile, launch_speed_FalseSingle_median,
                   launch_speed_FalseSingle_3quartile),
             label = c(launch_speed_FalseSingle_1quartile,
                       launch_speed_FalseSingle_median,
```

```
                    launch_speed_FalseSingle_3quartile)) +
  annotate("text", x = c(2.59, 2.59, 2.59), size = 2.5, fontface = 2, hjust = 1,
           y = c(launch_speed_TrueOut_1quartile, launch_speed_TrueOut_median,
                 launch_speed_TrueOut_3quartile),
           label = c(launch_speed_TrueOut_1quartile, launch_speed_TrueOut_median,
                     launch_speed_TrueOut_3quartile)) +
  annotate("text", x = c(3.59, 3.59, 3.59), size = 2.5, fontface = 2, hjust = 1,
           y = c(launch_speed_FalseOut_1quartile, launch_speed_FalseOut_median,
                 launch_speed_FalseOut_3quartile),
           label = c(launch_speed_FalseOut_1quartile, launch_speed_FalseOut_median,
                     launch_speed_FalseOut_3quartile)))
```



With regard to launch_speed, the singles that were mistakenly predicted to be outs had significantly higher exit velocities than the correctly predicted outs. The median launch speed for these false outs was 92.3, as compared to 87.2 for the true outs. MLB considers hard hit balls to have exit velocities of 95 miler per hour

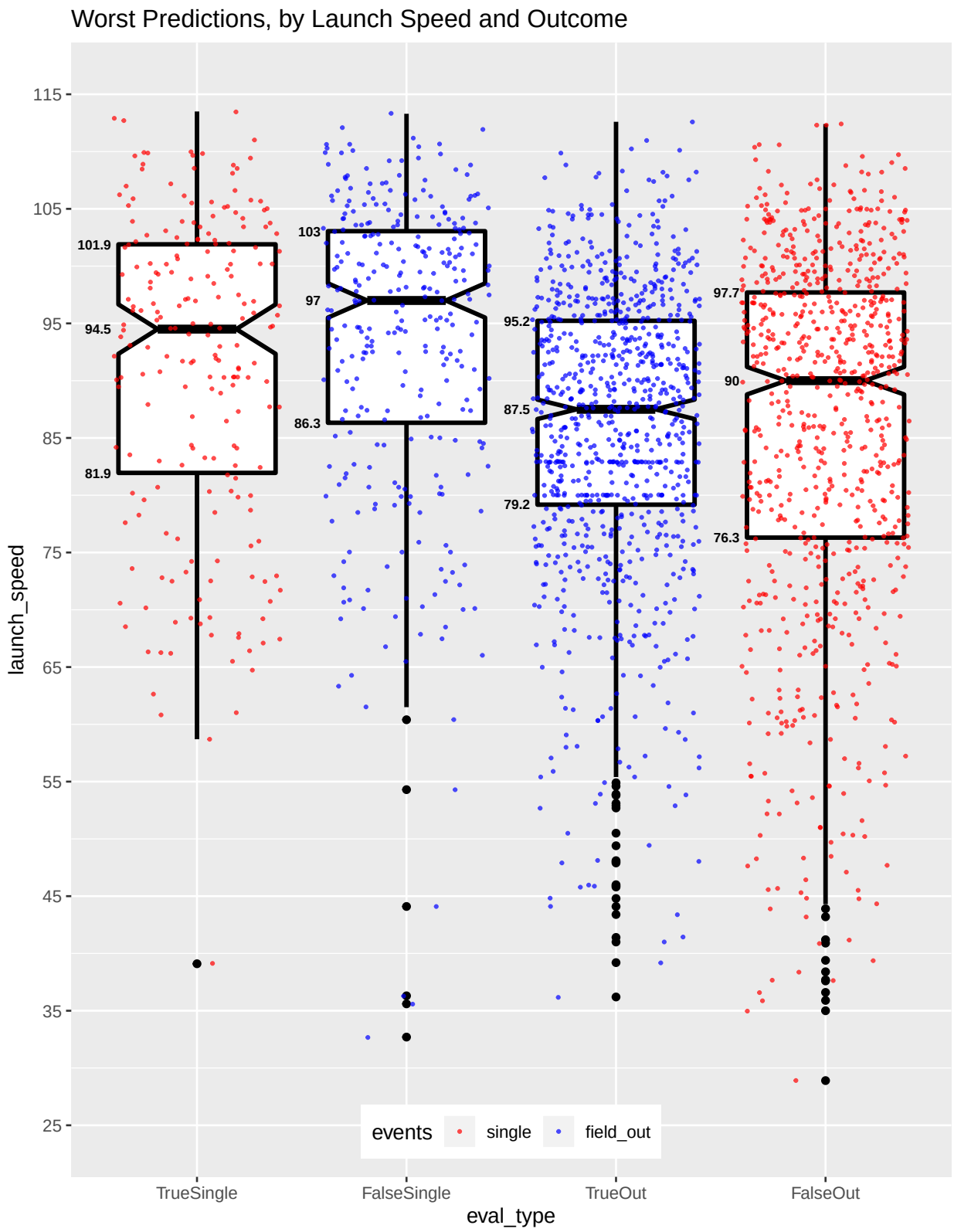
or more. So it's curious that the model missed on these predictions, unless there was some other feature that was screaming "out!".

What kinds of launch speeds did we see with the worst predictions?

```
# Compare, by launch_speed, our population of the worst 1,106 predictions to a random
# population of 1,106 correct predictions.

truefalse_by_launch_speed <- ggplot(predictions_2212,
                                    aes(x = eval_type, y = launch_speed,
                                        color = events)) +
  geom_boxplot(color = "black", notch = TRUE, size = 1.1) +
  geom_point(position = "jitter", alpha = .7, size = 0.5) +
  scale_y_continuous(breaks = seq(25, 115, by = 10),
                     labels = c("25", "35", "45", "55", "65", "75",  "85", "95", "105",
                                "115"),
                     limits = c(25, 115)) +
  scale_color_manual(values = c("Red", "Blue")) +
  ggtitle("Worst Predictions, by Launch Speed and Outcome") +
  theme(legend.position = c(.5, .04), legend.direction = "horizontal")

truefalse_by_launch_speed_data <- layer_data(truefalse_by_launch_speed)

launch_speed_TrueSingle_1quartile <- round(truefalse_by_launch_speed_data[1, 2], 1)
launch_speed_TrueSingle_median <- round(truefalse_by_launch_speed_data[1, 3], 1)
launch_speed_TrueSingle_3quartile <- round(truefalse_by_launch_speed_data[1, 4], 1)
launch_speed_FalseSingle_1quartile <- round(truefalse_by_launch_speed_data[2, 2], 1)
launch_speed_FalseSingle_median <- round(truefalse_by_launch_speed_data[2, 3], 1)
launch_speed_FalseSingle_3quartile <- round(truefalse_by_launch_speed_data[2, 4], 1)
launch_speed_TrueOut_1quartile <- round(truefalse_by_launch_speed_data[3, 2], 1)
launch_speed_TrueOut_median <- round(truefalse_by_launch_speed_data[3, 3], 1)
launch_speed_TrueOut_3quartile <- round(truefalse_by_launch_speed_data[3, 4], 1)
launch_speed_FalseOut_1quartile <- round(truefalse_by_launch_speed_data[4, 2], 1)
launch_speed_FalseOut_median <- round(truefalse_by_launch_speed_data[4, 3], 1)
launch_speed_FalseOut_3quartile <- round(truefalse_by_launch_speed_data[4, 4], 1)
```

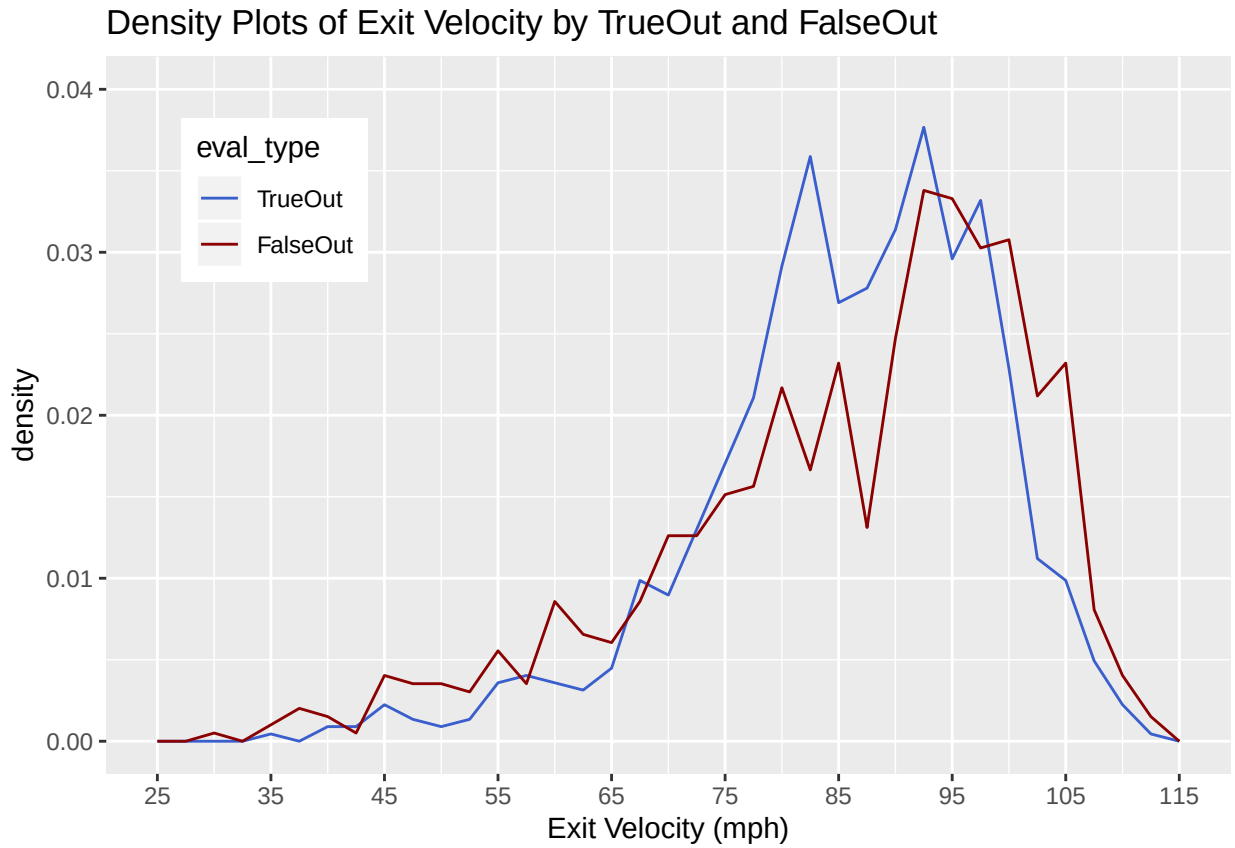
```
(truefalse_by_launch_speed <- ggplot(predictions_2212,
                                     aes(x = eval_type, y = launch_speed,
                                         color = events)) +
    geom_boxplot(color = "black", notch = TRUE, size = 1.1) +
    geom_point(position = "jitter", alpha = .7, size = 0.5) +
    scale_y_continuous(breaks = seq(25, 115, by = 10),
                       labels = c("25", "35", "45", "55", "65", "75",  "85", "95", "105",
                                  "115"),
                       limits = c(25, 115)) +
    scale_color_manual(values = c("Red", "Blue")) +
    ggtitle("Worst Predictions, by Launch Speed and Outcome") +
    theme(legend.position = c(.5, .04), legend.direction = "horizontal") +
    annotate("text", x = c(.59, .59, .59), size = 2.5, fontface = 2, hjust = 1,
             y = c(launch_speed_TrueSingle_1quartile, launch_speed_TrueSingle_median,
                   launch_speed_TrueSingle_3quartile),
             label = c(launch_speed_TrueSingle_1quartile, launch_speed_TrueSingle_median,
                       launch_speed_TrueSingle_3quartile)) +
    annotate("text", x = c(1.59,1.59, 1.59), size = 2.5, fontface = 2, hjust = 1,
```

```
          y = c(launch_speed_FalseSingle_1quartile, launch_speed_FalseSingle_median,
                launch_speed_FalseSingle_3quartile),
          label = c(launch_speed_FalseSingle_1quartile,
                    launch_speed_FalseSingle_median,
                    launch_speed_FalseSingle_3quartile)) +
  annotate("text", x = c(2.59, 2.59, 2.59), size = 2.5, fontface = 2, hjust = 1,
          y = c(launch_speed_TrueOut_1quartile, launch_speed_TrueOut_median,
                launch_speed_TrueOut_3quartile),
          label = c(launch_speed_TrueOut_1quartile, launch_speed_TrueOut_median,
                    launch_speed_TrueOut_3quartile)) +
  annotate("text", x = c(3.59, 3.59, 3.59), size = 2.5, fontface = 2, hjust = 1,
          y = c(launch_speed_FalseOut_1quartile, launch_speed_FalseOut_median,
                launch_speed_FalseOut_3quartile),
          label = c(launch_speed_FalseOut_1quartile, launch_speed_FalseOut_median,
                    launch_speed_FalseOut_3quartile)))
```



The entire interquartile range of launch speeds dropped a couple miles per hour for the worst out predictions, bringing the mistakes a bit more in line with the exit velocities of the true outs. We took a look at the density plots next.

```
# Compare the densities of launch_speed, using our population of the worst 1,106
# prediction mistakes and a random population of 1,106 correct predictions.

x_scale <- scale_x_continuous(breaks = seq(25, 115, by = 10),
                              labels = c("25", "35", "45", "55", "65", "75",  "85", "95",
                                         "105", "115"),
                              limits = c(25, 115))
y_scale <- scale_y_continuous(limits = c(0, 0.04))
(ls_density <- ggplot(dplyr::filter(predictions_2212, eval_type == "TrueOut" |
                                      eval_type == "FalseOut"),
                      aes(x = launch_speed, y = stat(density), color = eval_type)) +
    geom_freqpoly(binwidth = 2.5) +
    x_scale +
    y_scale +
    scale_color_manual(values = c(FalseOut = "red4", TrueOut = "royalblue3")) +
    labs(title = "Density Plots of Exit Velocity by TrueOut and FalseOut",
         x = "Exit Velocity (mph)")) +
  theme(legend.position = c(.15, .8))
```



They showed the model having some modest difficulty recognizing singles hit between 45 and 65 miles per hour, and between 100 and 107 miles per hour.

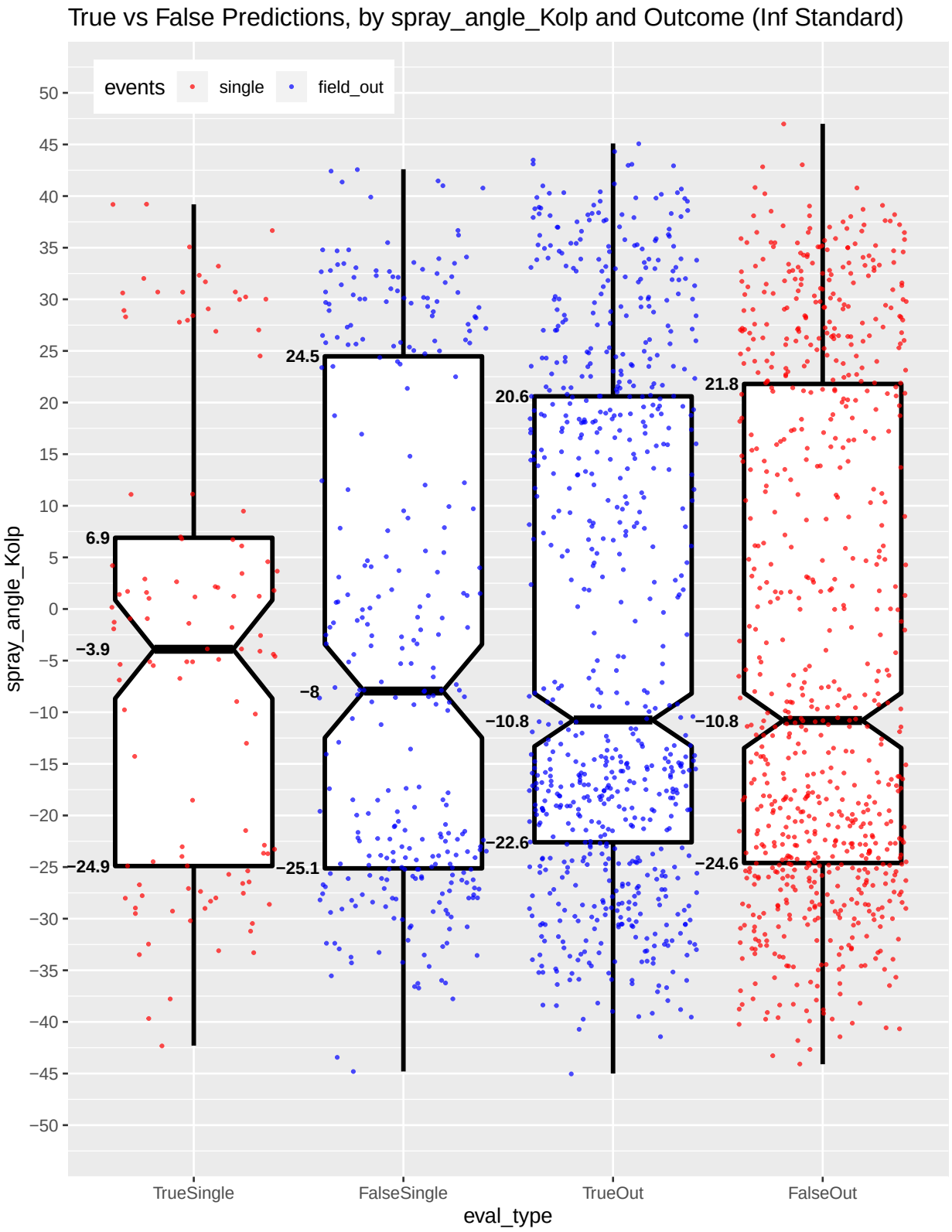
We examined spray_angle_Kolp next.

```
# Compare, by spray_angle_Kolp, our population of 2,533 mistaken predictions to a random
# population of 2,533 correct predictions.
```

```
predictions_5066_std_gb <-
  dplyr::filter(predictions_5066, bb_type == "ground_ball" &
                  if_fielding_alignment == "Standard")
predictions_5066_std_gb <- predictions_5066_std_gb %>%
  dplyr::filter(adv_bb_type == "low_ground_ball" | adv_bb_type == "high_ground_ball")

truefalse_by_spray_angle_Kolp <- ggplot(predictions_5066_std_gb,
                                        aes(x = eval_type, y = spray_angle_Kolp,
                                            color = events)) +
    geom_boxplot(color = "black", notch = TRUE, size = 1.1) +
    geom_point(position = "jitter", alpha = .7, size = 0.5) +
    scale_y_continuous(breaks = seq(-50, 50, by = 5),
                       labels = c("-50", "-45", "-40", "-35", "-30", "-25", "-20", "-15",
                                  "-10", "-5",  "0", "5", "10", "15", "20", "25", "30",
                                  "35", "40", "45", "50"),
                       limits = c(-50, 50)) +
    scale_color_manual(values = c("Red", "Blue")) +
    ggtitle("True vs False Predictions, by spray_angle_Kolp and Outcome (Inf Standard)") +
    theme(legend.position = c(.5, .96), legend.direction = "horizontal")

truefalse_by_spray_angle_Kolp_data <- layer_data(truefalse_by_spray_angle_Kolp)

spray_angle_Kolp_TrueSingle_1quartile <-
  round(truefalse_by_spray_angle_Kolp_data[1, 2], 1)
spray_angle_Kolp_TrueSingle_median <-
  round(truefalse_by_spray_angle_Kolp_data[1, 3], 1)
spray_angle_Kolp_TrueSingle_3quartile <-
  round(truefalse_by_spray_angle_Kolp_data[1, 4], 1)
spray_angle_Kolp_FalseSingle_1quartile <-
  round(truefalse_by_spray_angle_Kolp_data[2, 2], 1)
spray_angle_Kolp_FalseSingle_median <-
  round(truefalse_by_spray_angle_Kolp_data[2, 3], 1)
spray_angle_Kolp_FalseSingle_3quartile <-
  round(truefalse_by_spray_angle_Kolp_data[2, 4], 1)
spray_angle_Kolp_TrueOut_1quartile <-
  round(truefalse_by_spray_angle_Kolp_data[3, 2], 1)
spray_angle_Kolp_TrueOut_median <-
  round(truefalse_by_spray_angle_Kolp_data[3, 3], 1)
spray_angle_Kolp_TrueOut_3quartile <-
  round(truefalse_by_spray_angle_Kolp_data[3, 4], 1)
spray_angle_Kolp_FalseOut_1quartile <-
  round(truefalse_by_spray_angle_Kolp_data[4, 2], 1)
spray_angle_Kolp_FalseOut_median <-
  round(truefalse_by_spray_angle_Kolp_data[4, 3], 1)
spray_angle_Kolp_FalseOut_3quartile <-
  round(truefalse_by_spray_angle_Kolp_data[4, 4], 1)
```

```
(truefalse_by_spray_angle_Kolp <- ggplot(predictions_5066_std_gb,
                                         aes(x = eval_type, y = spray_angle_Kolp,
                                             color = events)) +
  geom_boxplot(color = "black", notch = TRUE, size = 1.1) +
  geom_point(position = "jitter", alpha = .7, size = 0.5) +
  scale_y_continuous(breaks = seq(-50, 50, by = 5),
```

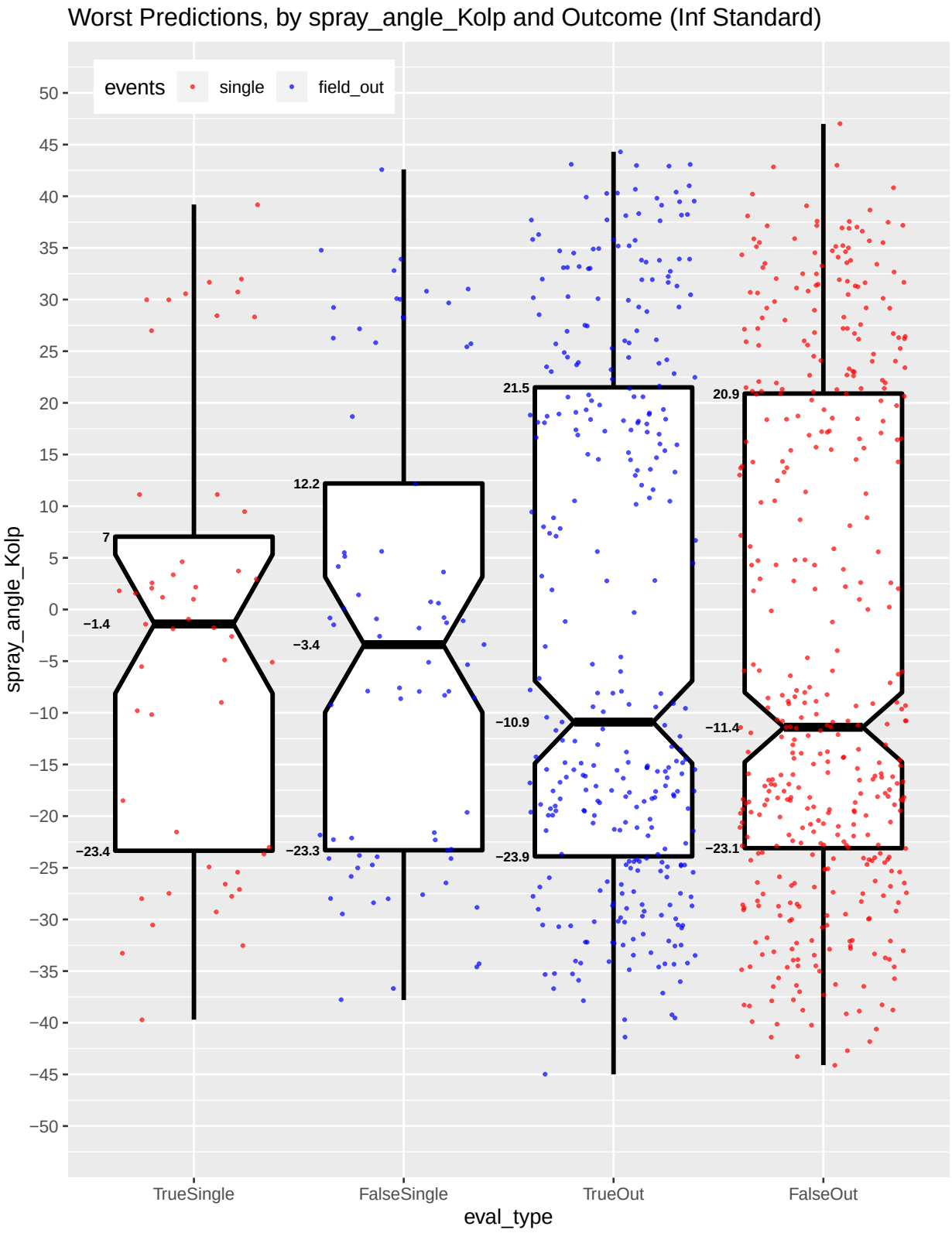
```
                    labels = c("-50", "-45", "-40", "-35", "-30", "-25", "-20", "-15",
                               "-10", "-5",  "0", "5", "10", "15", "20", "25", "30",
                               "35", "40", "45", "50"),
                    limits = c(-50, 50)) +
scale_color_manual(values = c("Red", "Blue")) +
ggtitle("True vs False Predictions, by spray_angle_Kolp and Outcome (Inf Standard)") +
theme(legend.position = c(.2, .96), legend.direction = "horizontal") +
annotate("text", x = c(.60, .60, .60), size = 3, fontface = 2, hjust = 1,
         y = c(spray_angle_Kolp_TrueSingle_1quartile,
               spray_angle_Kolp_TrueSingle_median,
               spray_angle_Kolp_TrueSingle_3quartile),
         label = c(spray_angle_Kolp_TrueSingle_1quartile,
                   spray_angle_Kolp_TrueSingle_median,
                   spray_angle_Kolp_TrueSingle_3quartile)) +
annotate("text", x = c(1.60,1.60, 1.60), size = 3, fontface = 2, hjust = 1,
         y = c(spray_angle_Kolp_FalseSingle_1quartile,
               spray_angle_Kolp_FalseSingle_median,
               spray_angle_Kolp_FalseSingle_3quartile),
         label = c(spray_angle_Kolp_FalseSingle_1quartile,
                   spray_angle_Kolp_FalseSingle_median,
                   spray_angle_Kolp_FalseSingle_3quartile)) +
annotate("text", x = c(2.60, 2.60, 2.60), size = 3, fontface = 2, hjust = 1,
         y = c(spray_angle_Kolp_TrueOut_1quartile,
               spray_angle_Kolp_TrueOut_median,
               spray_angle_Kolp_TrueOut_3quartile),
         label = c(spray_angle_Kolp_TrueOut_1quartile,
                   spray_angle_Kolp_TrueOut_median,
                   spray_angle_Kolp_TrueOut_3quartile)) +
annotate("text", x = c(3.60, 3.60, 3.60), size = 3, fontface = 2, hjust = 1,
         y = c(spray_angle_Kolp_FalseOut_1quartile,
               spray_angle_Kolp_FalseOut_median,
               spray_angle_Kolp_FalseOut_3quartile),
         label = c(spray_angle_Kolp_FalseOut_1quartile,
                   spray_angle_Kolp_FalseOut_median,
                   spray_angle_Kolp_FalseOut_3quartile)))
```



When looking at all the false outs, the distribution is very similar to that of the true outs. No pattern is apparent that would tell us the model makes more mistakes with certain unadjusted spray angles. Next, we looked at the worst out predictions.

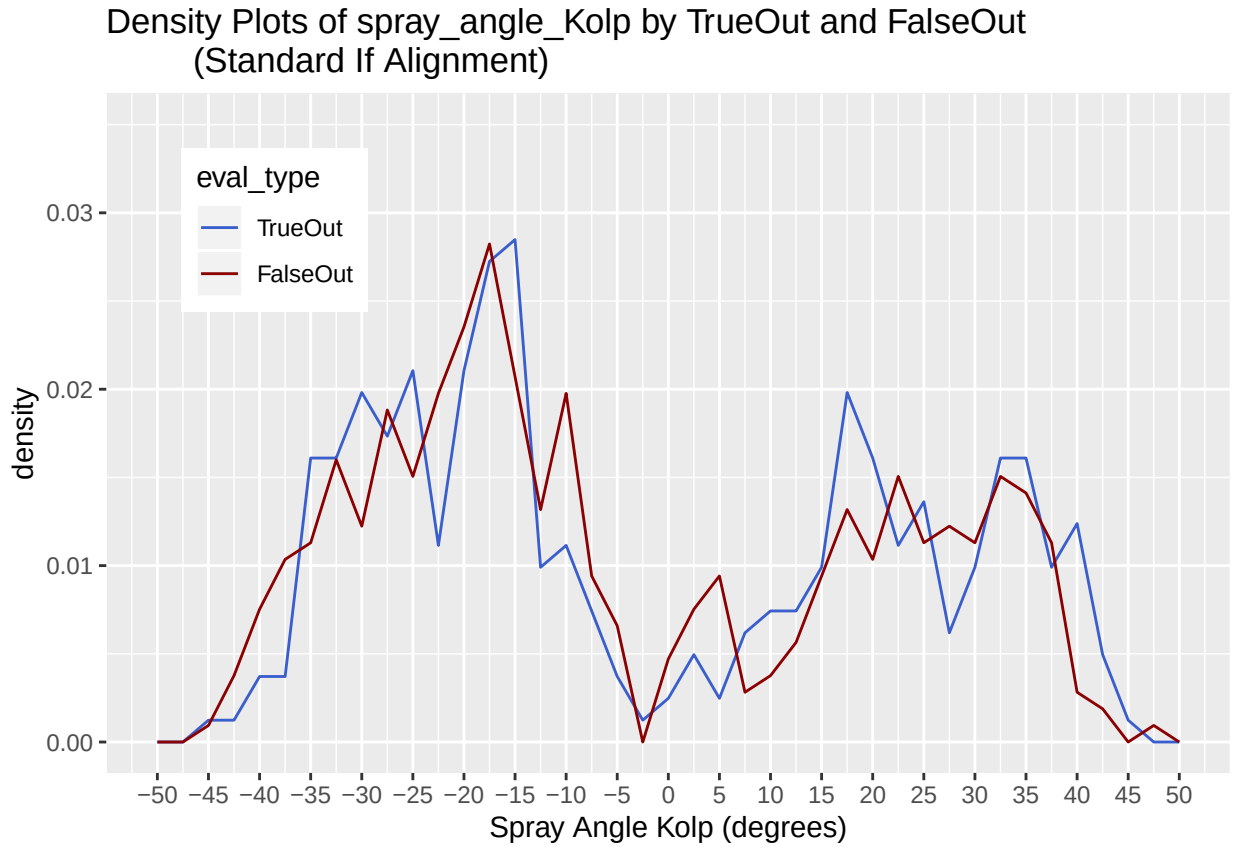
```r
# Compare, by spray_angle_Kolp, our population of 1,106 mistaken predictions to a random
# population of 1,106 correct predictions.

predictions_2212_std_gb <-
  dplyr::filter(predictions_2212,
                bb_type == "ground_ball" & if_fielding_alignment == "Standard")
predictions_2212_std_gb <- predictions_2212_std_gb %>%
  dplyr::filter(adv_bb_type == "low_ground_ball" | adv_bb_type == "high_ground_ball")


truefalse_by_spray_angle_Kolp <- ggplot(predictions_2212_std_gb,
                                        aes(x = eval_type, y = spray_angle_Kolp,
                                            color = events)) +
  geom_boxplot(color = "black", notch = TRUE, size = 1.1) +
  geom_point(position = "jitter", alpha = .7, size = 0.5) +
  scale_y_continuous(breaks = seq(-50, 50, by = 5),
                     labels = c("-50", "-45", "-40", "-35", "-30", "-25", "-20", "-15",
                                "-10", "-5",  "0",
                                "5", "10", "15", "20", "25", "30", "35", "40", "45",
                                "50"),
                     limits = c(-50, 50)) +
  scale_color_manual(values = c("Red", "Blue")) +
  ggtitle("Worst Predictions, by spray_angle_Kolp and Outcome (Inf Standard)") +
  theme(legend.position = c(.5, .96), legend.direction = "horizontal")

truefalse_by_spray_angle_Kolp_data <- layer_data(truefalse_by_spray_angle_Kolp)

spray_angle_Kolp_TrueSingle_1quartile <-
  round(truefalse_by_spray_angle_Kolp_data[1, 2], 1)
spray_angle_Kolp_TrueSingle_median <-
  round(truefalse_by_spray_angle_Kolp_data[1, 3], 1)
spray_angle_Kolp_TrueSingle_3quartile <-
  round(truefalse_by_spray_angle_Kolp_data[1, 4], 1)
spray_angle_Kolp_FalseSingle_1quartile <-
  round(truefalse_by_spray_angle_Kolp_data[2, 2], 1)
spray_angle_Kolp_FalseSingle_median <-
  round(truefalse_by_spray_angle_Kolp_data[2, 3], 1)
spray_angle_Kolp_FalseSingle_3quartile <-
  round(truefalse_by_spray_angle_Kolp_data[2, 4], 1)
spray_angle_Kolp_TrueOut_1quartile <-
  round(truefalse_by_spray_angle_Kolp_data[3, 2], 1)
spray_angle_Kolp_TrueOut_median <-
  round(truefalse_by_spray_angle_Kolp_data[3, 3], 1)
spray_angle_Kolp_TrueOut_3quartile <-
  round(truefalse_by_spray_angle_Kolp_data[3, 4], 1)
spray_angle_Kolp_FalseOut_1quartile <-
  round(truefalse_by_spray_angle_Kolp_data[4, 2], 1)
spray_angle_Kolp_FalseOut_median <-
  round(truefalse_by_spray_angle_Kolp_data[4, 3], 1)
spray_angle_Kolp_FalseOut_3quartile <-
  round(truefalse_by_spray_angle_Kolp_data[4, 4], 1)
```

```
(truefalse_by_spray_angle_Kolp <- ggplot(predictions_2212_std_gb,
                                         aes(x = eval_type, y = spray_angle_Kolp,
                                             color = events)) +
    geom_boxplot(color = "black", notch = TRUE, size = 1.1) +
    geom_point(position = "jitter", alpha = .7, size = 0.5) +
    scale_y_continuous(breaks = seq(-50, 50, by = 5),
                       labels = c("-50", "-45", "-40", "-35", "-30", "-25", "-20", "-15",
                                  "-10", "-5",  "0",
                                  "5", "10", "15", "20", "25", "30", "35", "40", "45",
                                  "50"),
                       limits = c(-50, 50)) +
    scale_color_manual(values = c("Red", "Blue")) +
    ggtitle("Worst Predictions, by spray_angle_Kolp and Outcome (Inf Standard)") +
    theme(legend.position = c(.2, .96), legend.direction = "horizontal") +
    annotate("text", x = c(.60, .60, .60), size = 2.5, fontface = 2, hjust = 1,
             y = c(spray_angle_Kolp_TrueSingle_1quartile,
                   spray_angle_Kolp_TrueSingle_median,
                   spray_angle_Kolp_TrueSingle_3quartile),
             label = c(spray_angle_Kolp_TrueSingle_1quartile,
                       spray_angle_Kolp_TrueSingle_median,
                       spray_angle_Kolp_TrueSingle_3quartile)) +
    annotate("text", x = c(1.60,1.60, 1.60), size = 2.5, fontface = 2, hjust = 1,
             y = c(spray_angle_Kolp_FalseSingle_1quartile,
                   spray_angle_Kolp_FalseSingle_median,
                   spray_angle_Kolp_FalseSingle_3quartile),
             label = c(spray_angle_Kolp_FalseSingle_1quartile,
                       spray_angle_Kolp_FalseSingle_median,
                       spray_angle_Kolp_FalseSingle_3quartile)) +
    annotate("text", x = c(2.60, 2.60, 2.60), size = 2.5, fontface = 2, hjust = 1,
             y = c(spray_angle_Kolp_TrueOut_1quartile,
                   spray_angle_Kolp_TrueOut_median,
                   spray_angle_Kolp_TrueOut_3quartile),
             label = c(spray_angle_Kolp_TrueOut_1quartile,
                       spray_angle_Kolp_TrueOut_median,
                       spray_angle_Kolp_TrueOut_3quartile)) +
    annotate("text", x = c(3.60, 3.60, 3.60), size = 2.5, fontface = 2, hjust = 1,
             y = c(spray_angle_Kolp_FalseOut_1quartile,
                   spray_angle_Kolp_FalseOut_median,
                   spray_angle_Kolp_FalseOut_3quartile),
             label = c(spray_angle_Kolp_FalseOut_1quartile,
                       spray_angle_Kolp_FalseOut_median,
                       spray_angle_Kolp_FalseOut_3quartile)))
```



Here, it looks like the false outs are a bit more dense from -12 to about 7 degrees, almost directly up the middle of the field. We needed to see if that was borne out by the density plots.

```r
# Compare the densities of spray_angle_Kolp (Standard If Alignment), using our population
# of the worst 1,106 prediction mistakes and a random population of 1,106 correct
# predictions.

predictions_2212_std_gb_TFOuts <- dplyr::filter(predictions_2212,
                bb_type == "ground_ball" & if_fielding_alignment == "Standard")
predictions_2212_std_gb_TFOuts <- predictions_2212_std_gb_TFOuts %>%
  dplyr::filter(adv_bb_type == "low_ground_ball" | adv_bb_type == "high_ground_ball")
predictions_2212_std_gb_TFOuts <- predictions_2212_std_gb_TFOuts %>%
  dplyr::filter(eval_type == "TrueOut" | eval_type == "FalseOut")

x_scale <- scale_x_continuous(breaks = seq(-50, 50, by = 5),
                              labels = c("-50", "-45", "-40", "-35", "-30", "-25", "-20",
                                         "-15", "-10", "-5",  "0", "5", "10", "15", "20",
                                         "25", "30", "35", "40", "45", "50"),
                              limits = c(-50, 50))
y_scale <- scale_y_continuous(limits = c(0, 0.035))

(sa_Kolp_density <- ggplot(predictions_2212_std_gb_TFOuts,
                           aes(x = spray_angle_Kolp, y = stat(density),
                               color = eval_type)) +
    geom_freqpoly(binwidth = 2.5) +
    x_scale +
    y_scale +
    scale_color_manual(name = "eval_type", labels = c("TrueOut", "FalseOut"),
                       values = c("royalblue3", "red4")) +
    labs(title = "Density Plots of spray_angle_Kolp by TrueOut and FalseOut
         (Standard If Alignment)", x = "Spray Angle Kolp (degrees)")) +
  theme(legend.position = c(.15, .80))
```



Here, we can see the model likes to predict outs a bit too frequently on balls hit down the third base line (-38 to -44 degrees), balls hit toward the hole between the shortstop and third baseman (-22 to -17 degrees), balls hit toward the middle of the field (-12 to 7 degrees), and balls hit toward the hole between the second baseman and the first baseman (21 to 30 degrees). These, in fact, are all of the infield gaps when infielders are in a standard alignment. The model appears to recognize the impact of these gaps to some degree, just not to their full extent.

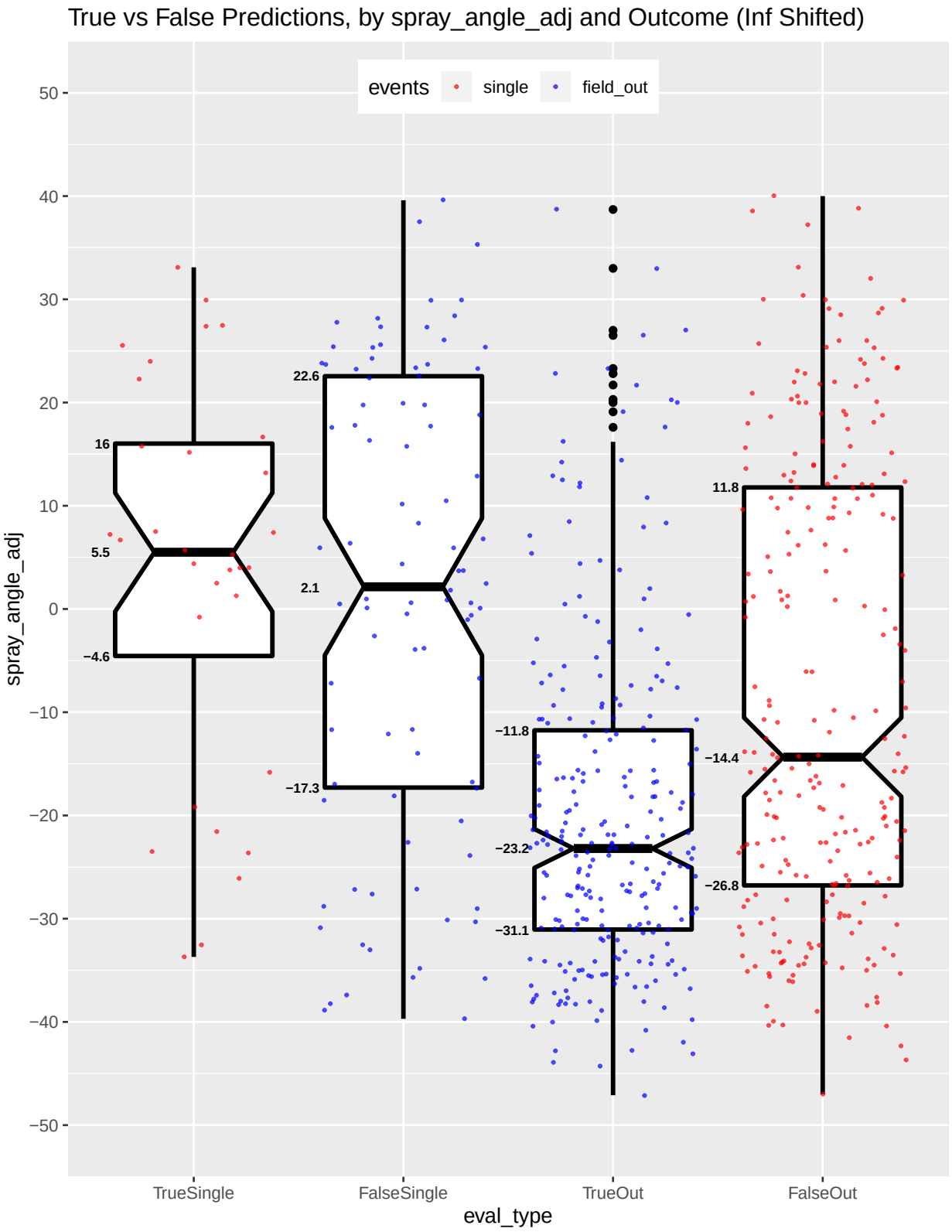
The next feature up for study was spray_angle_adj. You'll recall that negative values for this variety of Spray Angle indicate the batter has pulled the ball, while positive values indicate the ball has been hit to the opposite side from where the batter stands in relation to home plate.

```
# Compare, by spray_angle_adj, our population of 2,533 mistaken predictions to a random
# population of 2,533 correct predictions.

predictions_5066_shift_gb <-
  dplyr::filter(predictions_5066, bb_type == "ground_ball" &
                  if_fielding_alignment == "Infield shift")
predictions_5066_shift_gb <- predictions_5066_shift_gb %>%
  dplyr::filter(adv_bb_type == "low_ground_ball" | adv_bb_type == "high_ground_ball")

truefalse_by_spray_angle_adj <- ggplot(predictions_5066_shift_gb,
                                       aes(x = eval_type, y = spray_angle_adj,
                                           color = events)) +
  geom_boxplot(color = "black", notch = TRUE, size = 1.1) +
  geom_point(position = "jitter", alpha = .7, size = 0.5) +
  scale_y_continuous(breaks = seq(-50, 50, by = 10),
                     labels = c("-50", "-40", "-30", "-20", "-10", "0", "10", "20", "30",
```

```
                                "40",  "50"),
                     limits = c(-50, 50)) +
  scale_color_manual(values = c("Red", "Blue")) +
  ggtitle("True vs False Predictions, by spray_angle_adj and Outcome (Inf Shifted)") +
  theme(legend.position = c(.5, .96), legend.direction = "horizontal")

truefalse_by_spray_angle_adj_data <- layer_data(truefalse_by_spray_angle_adj)

spray_angle_adj_TrueSingle_1quartile <- round(truefalse_by_spray_angle_adj_data[1, 2], 1)
spray_angle_adj_TrueSingle_median <- round(truefalse_by_spray_angle_adj_data[1, 3], 1)
spray_angle_adj_TrueSingle_3quartile <- round(truefalse_by_spray_angle_adj_data[1, 4], 1)
spray_angle_adj_FalseSingle_1quartile <- round(truefalse_by_spray_angle_adj_data[2, 2], 1)
spray_angle_adj_FalseSingle_median <- round(truefalse_by_spray_angle_adj_data[2, 3], 1)
spray_angle_adj_FalseSingle_3quartile <- round(truefalse_by_spray_angle_adj_data[2, 4], 1)
spray_angle_adj_TrueOut_1quartile <- round(truefalse_by_spray_angle_adj_data[3, 2], 1)
spray_angle_adj_TrueOut_median <- round(truefalse_by_spray_angle_adj_data[3, 3], 1)
spray_angle_adj_TrueOut_3quartile <- round(truefalse_by_spray_angle_adj_data[3, 4], 1)
spray_angle_adj_FalseOut_1quartile <- round(truefalse_by_spray_angle_adj_data[4, 2], 1)
spray_angle_adj_FalseOut_median <- round(truefalse_by_spray_angle_adj_data[4, 3], 1)
spray_angle_adj_FalseOut_3quartile <- round(truefalse_by_spray_angle_adj_data[4, 4], 1)
```

```
(truefalse_by_spray_angle_adj <- ggplot(predictions_5066_shift_gb,
                                        aes(x = eval_type, y = spray_angle_adj,
                                            color = events)) +
  geom_boxplot(color = "black", notch = TRUE, size = 1.1) +
  geom_point(position = "jitter", alpha = .7, size = 0.5) +
  scale_y_continuous(breaks = seq(-50, 50, by = 10),
                     labels = c("-50", "-40", "-30", "-20", "-10", "0", "10", "20", "30",
                                "40",  "50"),
                     limits = c(-50, 50)) +
  scale_color_manual(values = c("Red", "Blue")) +
  ggtitle("True vs False Predictions, by spray_angle_adj and Outcome (Inf Shifted)") +
  theme(legend.position = c(.5, .96), legend.direction = "horizontal") +
  annotate("text", x = c(.60, .60, .60), size = 2.5, fontface = 2, hjust = 1,
           y = c(spray_angle_adj_TrueSingle_1quartile,
                 spray_angle_adj_TrueSingle_median,
                 spray_angle_adj_TrueSingle_3quartile),
           label = c(spray_angle_adj_TrueSingle_1quartile,
                     spray_angle_adj_TrueSingle_median,
                     spray_angle_adj_TrueSingle_3quartile)) +
  annotate("text", x = c(1.60,1.60, 1.60), size = 2.5, fontface = 2, hjust = 1,
           y = c(spray_angle_adj_FalseSingle_1quartile,
                 spray_angle_adj_FalseSingle_median,
                 spray_angle_adj_FalseSingle_3quartile),
           label = c(spray_angle_adj_FalseSingle_1quartile,
                     spray_angle_adj_FalseSingle_median,
                     spray_angle_adj_FalseSingle_3quartile)) +
  annotate("text", x = c(2.60, 2.60, 2.60), size = 2.5, fontface = 2, hjust = 1,
           y = c(spray_angle_adj_TrueOut_1quartile,
                 spray_angle_adj_TrueOut_median,
                 spray_angle_adj_TrueOut_3quartile),
           label = c(spray_angle_adj_TrueOut_1quartile,
                     spray_angle_adj_TrueOut_median,
```

```
                    spray_angle_adj_TrueOut_3quartile)) +
annotate("text", x = c(3.60, 3.60, 3.60), size = 2.5, fontface = 2, hjust = 1,
         y = c(spray_angle_adj_FalseOut_1quartile,
               spray_angle_adj_FalseOut_median,
               spray_angle_adj_FalseOut_3quartile),
         label = c(spray_angle_adj_FalseOut_1quartile,
                   spray_angle_adj_FalseOut_median,
                   spray_angle_adj_FalseOut_3quartile)))
```



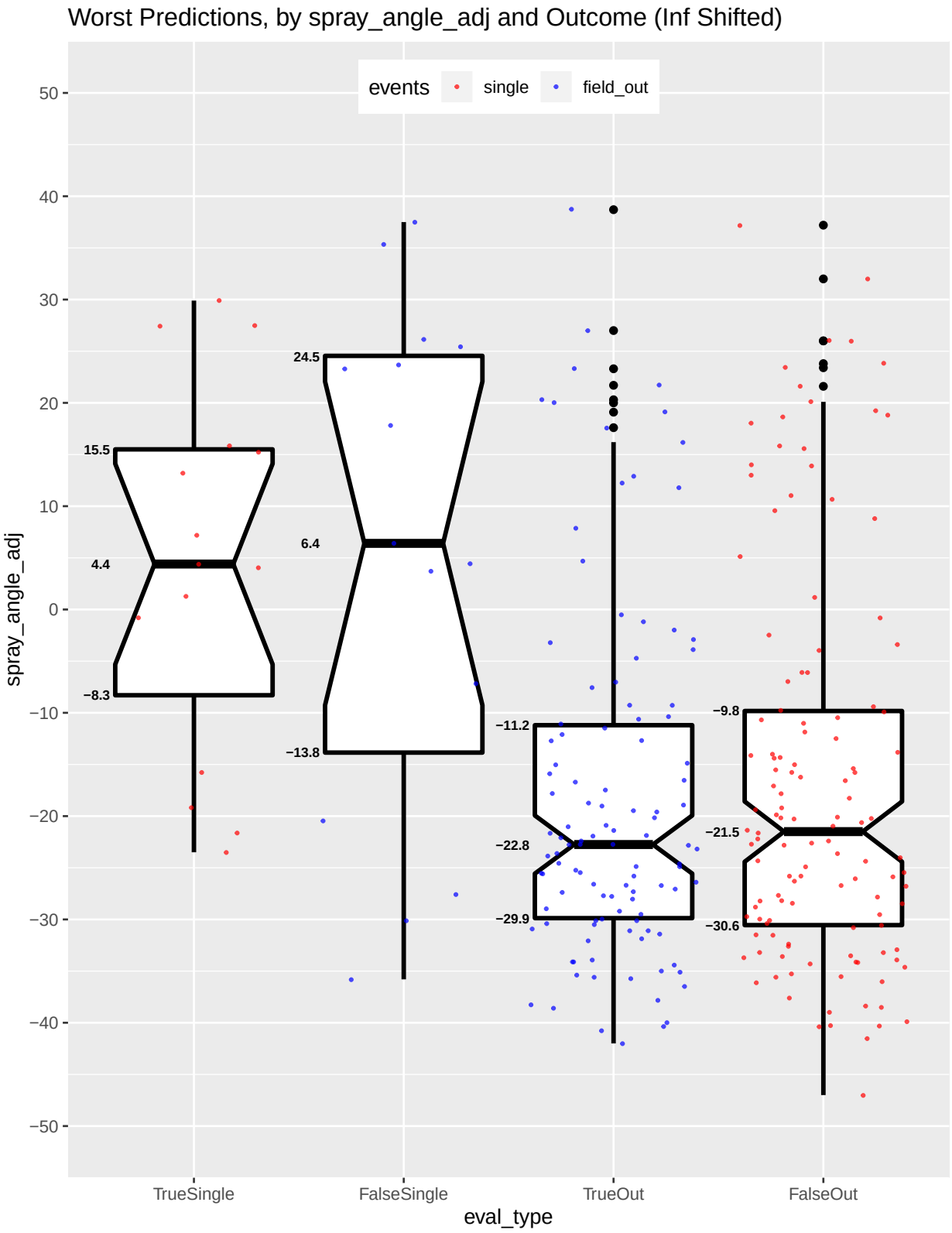
A clear pattern is apparent for *spray_angle_adj* when the infielders are shifted. The interquartile range for true outs covers -31 to -12 degrees. This makes sense because the pull side of the infield has three infielders, providing almost no gaps through which the ball can be hit. The interquartile range for false outs, on the

other hand, is more than twice as large, extending into the almost undefended opposite side of the field to 12 degrees.

```
# Compare, by spray_angle_adj, our population of 1,106 mistaken predictions to a
# random population of 1,106 correct predictions.

predictions_2212_shift_gb <-
  dplyr::filter(predictions_2212, bb_type == "ground_ball" &
                  if_fielding_alignment == "Infield shift")
predictions_2212_shift_gb <- predictions_2212_shift_gb %>%
  dplyr::filter(adv_bb_type == "low_ground_ball" |
                  adv_bb_type == "high_ground_ball")


truefalse_by_spray_angle_adj <- ggplot(predictions_2212_shift_gb,
                                       aes(x = eval_type, y = spray_angle_adj,
                                           color = events)) +
  geom_boxplot(color = "black", notch = TRUE, size = 1.1) +
  geom_point(position = "jitter", alpha = .7, size = 0.5) +
  scale_y_continuous(breaks = seq(-50, 50, by = 10),
                     labels = c("-50", "-40", "-30", "-20", "-10", "0", "10", "20", "30",
                                "40",  "50"),
                     limits = c(-50, 50)) +
  scale_color_manual(values = c("Red", "Blue")) +
  ggtitle("Worst Predictions, by spray_angle_adj and Outcome (Inf Shifted)") +
  theme(legend.position = c(.5, .96), legend.direction = "horizontal")

truefalse_by_spray_angle_adj_data <- layer_data(truefalse_by_spray_angle_adj)

spray_angle_adj_TrueSingle_1quartile <- round(truefalse_by_spray_angle_adj_data[1, 2], 1)
spray_angle_adj_TrueSingle_median <- round(truefalse_by_spray_angle_adj_data[1, 3], 1)
spray_angle_adj_TrueSingle_3quartile <- round(truefalse_by_spray_angle_adj_data[1, 4], 1)
spray_angle_adj_FalseSingle_1quartile <- round(truefalse_by_spray_angle_adj_data[2, 2], 1)
spray_angle_adj_FalseSingle_median <- round(truefalse_by_spray_angle_adj_data[2, 3], 1)
spray_angle_adj_FalseSingle_3quartile <- round(truefalse_by_spray_angle_adj_data[2, 4], 1)
spray_angle_adj_TrueOut_1quartile <- round(truefalse_by_spray_angle_adj_data[3, 2], 1)
spray_angle_adj_TrueOut_median <- round(truefalse_by_spray_angle_adj_data[3, 3], 1)
spray_angle_adj_TrueOut_3quartile <- round(truefalse_by_spray_angle_adj_data[3, 4], 1)
spray_angle_adj_FalseOut_1quartile <- round(truefalse_by_spray_angle_adj_data[4, 2], 1)
spray_angle_adj_FalseOut_median <- round(truefalse_by_spray_angle_adj_data[4, 3], 1)
spray_angle_adj_FalseOut_3quartile <- round(truefalse_by_spray_angle_adj_data[4, 4], 1)
```

```
(truefalse_by_spray_angle_adj <- ggplot(predictions_2212_shift_gb,
                                        aes(x = eval_type, y = spray_angle_adj,
                                            color = events)) +
    geom_boxplot(color = "black", notch = TRUE, size = 1.1) +
    geom_point(position = "jitter", alpha = .7, size = 0.5) +
    scale_y_continuous(breaks = seq(-50, 50, by = 10),
                       labels = c("-50", "-40", "-30", "-20", "-10", "0", "10", "20",
                                  "30", "40",  "50"),
                       limits = c(-50, 50)) +
    scale_color_manual(values = c("Red", "Blue")) +
    ggtitle("Worst Predictions, by spray_angle_adj and Outcome (Inf Shifted)") +
    theme(legend.position = c(.5, .96), legend.direction = "horizontal") +
```

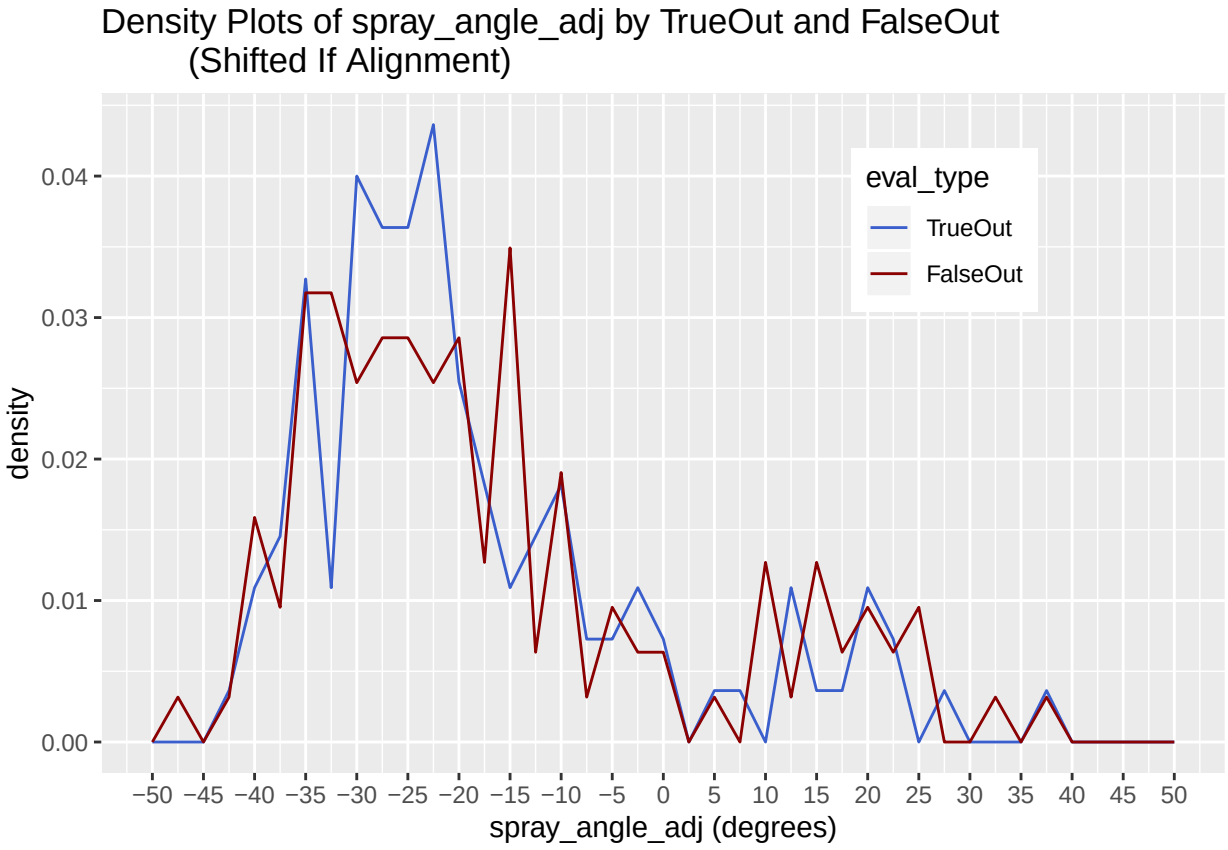
```
  annotate("text", x = c(.60, .60, .60), size = 2.5, fontface = 2, hjust = 1,
           y = c(spray_angle_adj_TrueSingle_1quartile,
                 spray_angle_adj_TrueSingle_median,
                 spray_angle_adj_TrueSingle_3quartile),
           label = c(spray_angle_adj_TrueSingle_1quartile,
                     spray_angle_adj_TrueSingle_median,
                     spray_angle_adj_TrueSingle_3quartile)) +
  annotate("text", x = c(1.60,1.60, 1.60), size = 2.5, fontface = 2, hjust = 1,
           y = c(spray_angle_adj_FalseSingle_1quartile,
                 spray_angle_adj_FalseSingle_median,
                 spray_angle_adj_FalseSingle_3quartile),
           label = c(spray_angle_adj_FalseSingle_1quartile,
                     spray_angle_adj_FalseSingle_median,
                     spray_angle_adj_FalseSingle_3quartile)) +
  annotate("text", x = c(2.60, 2.60, 2.60), size = 2.5, fontface = 2, hjust = 1,
           y = c(spray_angle_adj_TrueOut_1quartile,
                 spray_angle_adj_TrueOut_median,
                 spray_angle_adj_TrueOut_3quartile),
           label = c(spray_angle_adj_TrueOut_1quartile,
                     spray_angle_adj_TrueOut_median,
                     spray_angle_adj_TrueOut_3quartile)) +
  annotate("text", x = c(3.60, 3.60, 3.60), size = 2.5, fontface = 2, hjust = 1,
           y = c(spray_angle_adj_FalseOut_1quartile,
                 spray_angle_adj_FalseOut_median,
                 spray_angle_adj_FalseOut_3quartile),
           label = c(spray_angle_adj_FalseOut_1quartile,
                     spray_angle_adj_FalseOut_median,
                     spray_angle_adj_FalseOut_3quartile)))
```



The distribution of the worst false outs was very similar to that of the true outs. That would explain why the model computed such a low probability of a single in these cases. These batted balls looked more like outs. If this interpretation was true, the density plots for the worst false outs and the true outs ought to

track each other very closely.

```
# Compare the densities of spray_angle_adj (Shifted If Alignment), using our population
# of the worst 1,106 prediction mistakes and a random population of 1,106 correct
# predictions.

predictions_2212_shift_gb_TFOuts <-
  dplyr::filter(predictions_2212, bb_type == "ground_ball" &
                  if_fielding_alignment == "Infield shift")
predictions_2212_shift_gb_TFOuts <- predictions_2212_shift_gb_TFOuts %>%
  dplyr::filter(adv_bb_type == "low_ground_ball" | adv_bb_type == "high_ground_ball")
predictions_2212_shift_gb_TFOuts <- predictions_2212_shift_gb_TFOuts %>%
  dplyr::filter(eval_type == "TrueOut" | eval_type == "FalseOut")

x_scale <- scale_x_continuous(breaks = seq(-50, 50, by = 5),
                              labels = c("-50", "-45", "-40", "-35", "-30", "-25", "-20",
                                         "-15", "-10", "-5",  "0", "5", "10", "15", "20",
                                         "25", "30", "35", "40", "45", "50"),
                              limits = c(-50, 50))
#y_scale <- scale_y_continuous(limits = c(0, 0.035))

(sa_adj_density <- ggplot(predictions_2212_shift_gb_TFOuts,
                          aes(x = spray_angle_adj,
                              y = stat(density), color = eval_type)) +
    geom_freqpoly(binwidth = 2.5) +
    x_scale +
    #    y_scale +
    scale_color_manual(name = "eval_type", labels = c("TrueOut", "FalseOut"),
                       values = c("royalblue3", "red4")) +
    labs(title = "Density Plots of spray_angle_adj by TrueOut and FalseOut
         (Shifted If Alignment)", x = "spray_angle_adj (degrees)") +
    theme(legend.position = c(.75, .80)))
```



The density plots were sensitive enough to detect two quite small gaps on the pull side of the infield. Some batted balls snuck through these gaps for singles, and the model couldn't quite recognize this to the full extent that it happens. The gaps were -34 to -31 degrees, and -17 to -13 degrees. On the opposite side of the infield, batted balls were getting through the infield at 8 to 12 degrees, 14 to 18 degrees, 23 to 27 degrees, and 31 to 34 degrees.

If you add up the undefended areas on each side of the infield, you get 7 degrees of gaps on the pull side, and 15 degrees of gaps on the opposite side. This is the effect of a shifted infield. Yet, there are far fewer balls hit to the opposite side against a shift. One possible explanation is that the batters against whom shifts are deployed are the most extreme pull hitters who will find it most difficult to hit to the opposite side against a shift. But are they unable or unwilling? How much control do batters really have with respect to Spray Angle? Or do these extreme pull hitters **choose** not to try to hit to the opposite side because they (or their teams) believe the advantages of pulling the ball (e.g. the increased probability of an extra base hit) outweigh the negative consequences of hitting a ground ball into the shift?

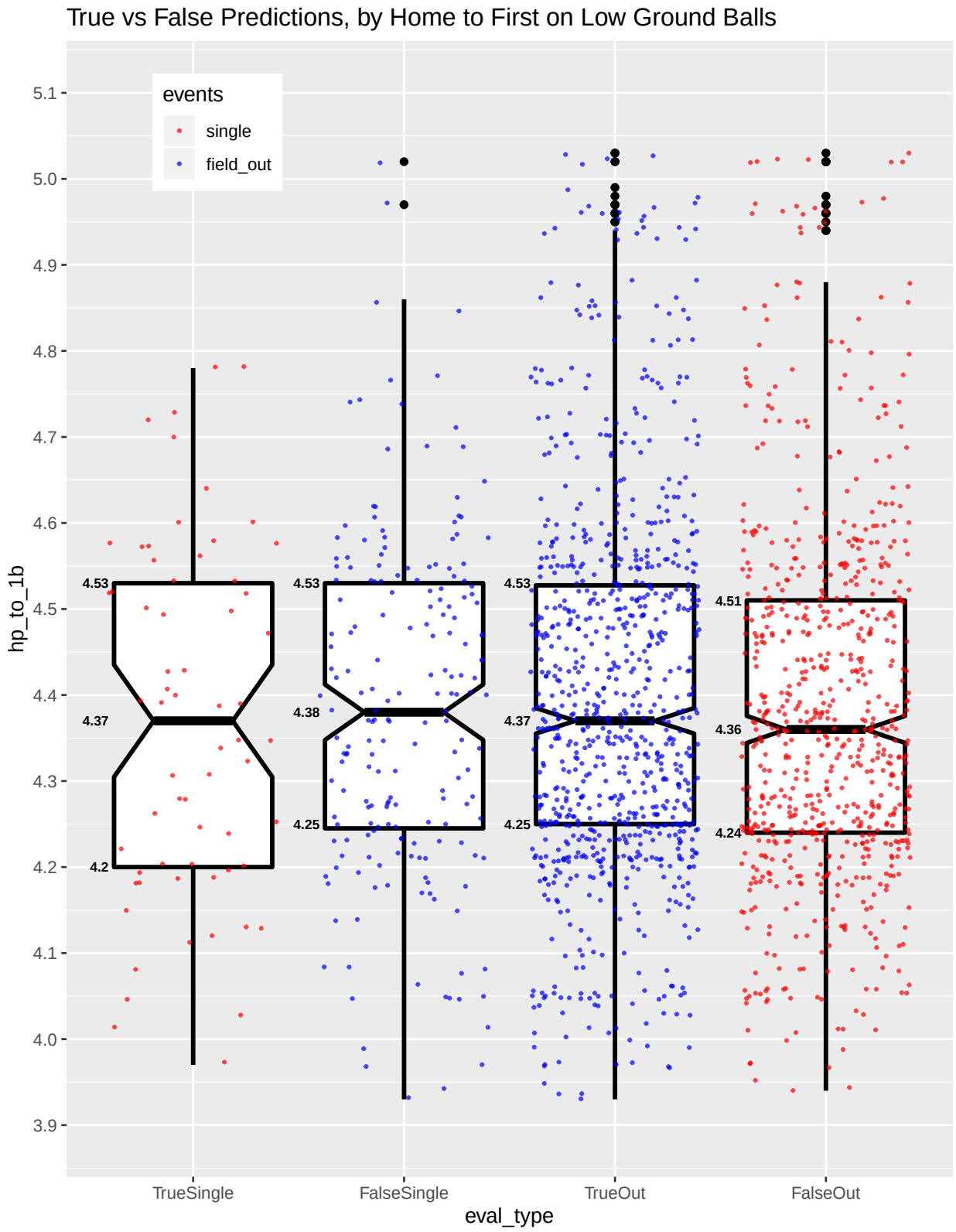
It was time to move on to our last feature, Home to First. Would we be able to see the predictive capacity of *hp_to_1b*, our weakest predictor according to our Classification Tree and Random Forest Models?

```
# Compare, by hp_to_1b, our population of 2,533 mistaken predictions to a random
# population of 2,533 correct predictions.

predictions_5066_gb <-
  dplyr::filter(predictions_5066, bb_type == "ground_ball" &
                  adv_bb_type == "low_ground_ball")

truefalse_by_hp_to_1b <- ggplot(predictions_5066_gb,
                                aes(x = eval_type, y = hp_to_1b, color = events)) +
```

```
  geom_boxplot(color = "black", notch = TRUE, size = 1.1) +
  geom_point(position = "jitter", alpha = .7, size = 0.5) +
  scale_y_continuous(breaks = seq(3.9, 5.1, by = 0.1),
                     labels = c("3.9", "4.0", "4.1", "4.2", "4.3", "4.4", "4.5",
                                "4.6", "4.7", "4.8",  "4.9", "5.0", "5.1"),
                     limits = c(3.9, 5.1)) +
  scale_color_manual(values = c("Red", "Blue")) +
  ggtitle("True vs False Predictions, by Home to First on Low Ground Balls") +
  theme(legend.position = c(.17, .92))

truefalse_by_hp_to_1b_data <- layer_data(truefalse_by_hp_to_1b)

hp_to_1b_TrueSingle_1quartile <- round(truefalse_by_hp_to_1b_data[1, 2], 2)
hp_to_1b_TrueSingle_median <- round(truefalse_by_hp_to_1b_data[1, 3], 2)
hp_to_1b_TrueSingle_3quartile <- round(truefalse_by_hp_to_1b_data[1, 4], 2)
hp_to_1b_FalseSingle_1quartile <- round(truefalse_by_hp_to_1b_data[2, 2], 2)
hp_to_1b_FalseSingle_median <- round(truefalse_by_hp_to_1b_data[2, 3], 2)
hp_to_1b_FalseSingle_3quartile <- round(truefalse_by_hp_to_1b_data[2, 4], 2)
hp_to_1b_TrueOut_1quartile <- round(truefalse_by_hp_to_1b_data[3, 2], 2)
hp_to_1b_TrueOut_median <- round(truefalse_by_hp_to_1b_data[3, 3], 2)
hp_to_1b_TrueOut_3quartile <- round(truefalse_by_hp_to_1b_data[3, 4], 2)
hp_to_1b_FalseOut_1quartile <- round(truefalse_by_hp_to_1b_data[4, 2], 2)
hp_to_1b_FalseOut_median <- round(truefalse_by_hp_to_1b_data[4, 3], 2)
hp_to_1b_FalseOut_3quartile <- round(truefalse_by_hp_to_1b_data[4, 4], 2)
```

```
(truefalse_by_hp_to_1b <- ggplot(predictions_5066_gb,
                                 aes(x = eval_type, y = hp_to_1b, color = events)) +
  geom_boxplot(color = "black", notch = TRUE, size = 1.1) +
  geom_point(position = "jitter", alpha = .7, size = 0.5) +
  scale_y_continuous(breaks = seq(3.9, 5.1, by = 0.1),
                     labels = c("3.9", "4.0", "4.1", "4.2", "4.3", "4.4", "4.5", "4.6",
                                "4.7", "4.8",  "4.9", "5.0", "5.1"),
                     limits = c(3.9, 5.1)) +
  scale_color_manual(values = c("Red", "Blue")) +
  ggtitle("True vs False Predictions, by Home to First on Low Ground Balls") +
  theme(legend.position = c(.17, .92)) +
  annotate("text", x = c(.60, .60, .60), size = 2.5, fontface = 2, hjust = 1,
           y = c(hp_to_1b_TrueSingle_1quartile, hp_to_1b_TrueSingle_median,
                 hp_to_1b_TrueSingle_3quartile),
           label = c(hp_to_1b_TrueSingle_1quartile, hp_to_1b_TrueSingle_median,
                     hp_to_1b_TrueSingle_3quartile)) +
  annotate("text", x = c(1.60,1.60, 1.60), size = 2.5, fontface = 2, hjust = 1,
           y = c(hp_to_1b_FalseSingle_1quartile, hp_to_1b_FalseSingle_median,
                 hp_to_1b_FalseSingle_3quartile),
           label = c(hp_to_1b_FalseSingle_1quartile, hp_to_1b_FalseSingle_median,
                     hp_to_1b_FalseSingle_3quartile)) +
  annotate("text", x = c(2.60, 2.60, 2.60), size = 2.5, fontface = 2, hjust = 1,
           y = c(hp_to_1b_TrueOut_1quartile, hp_to_1b_TrueOut_median,
                 hp_to_1b_TrueOut_3quartile),
           label = c(hp_to_1b_TrueOut_1quartile, hp_to_1b_TrueOut_median,
                     hp_to_1b_TrueOut_3quartile)) +
  annotate("text", x = c(3.60, 3.60, 3.60), size = 2.5, fontface = 2, hjust = 1,
           y = c(hp_to_1b_FalseOut_1quartile, hp_to_1b_FalseOut_median,
```

```
            hp_to_1b_FalseOut_3quartile),
        label = c(hp_to_1b_FalseOut_1quartile, hp_to_1b_FalseOut_median,
                  hp_to_1b_FalseOut_3quartile)))
```

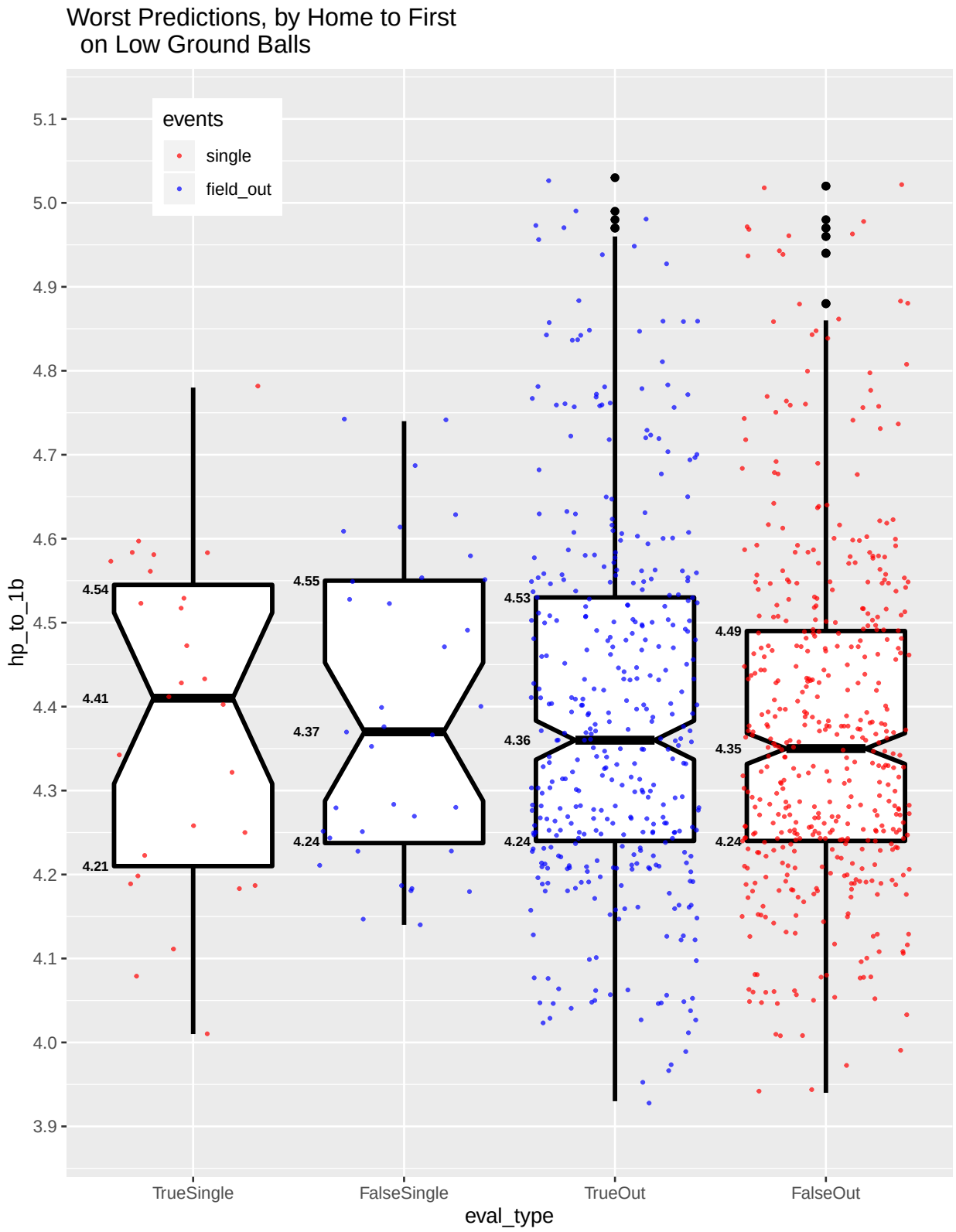


The distribution of false outs is virtually indistinguishable from that of true outs. We tried again, this time looking at only the worst false outs.

```
# Compare, by hp_to_1b, our population of 1,106 mistaken predictions to a random
# population of 1,106 correct predictions.

predictions_2212_gb <-
  dplyr::filter(predictions_2212, bb_type == "ground_ball" &
                  adv_bb_type == "low_ground_ball")

truefalse_by_hp_to_1b <- ggplot(predictions_2212_gb,
                                aes(x = eval_type, y = hp_to_1b, color = events)) +
  geom_boxplot(color = "black", notch = TRUE, size = 1.1) +
  geom_point(position = "jitter", alpha = .7, size = 0.5) +
  scale_y_continuous(breaks = seq(3.9, 5.1, by = 0.1),
                     labels = c("3.9", "4.0", "4.1", "4.2", "4.3", "4.4", "4.5", "4.6",
                                "4.7", "4.8",  "4.9", "5.0", "5.1"),
                     limits = c(3.9, 5.1)) +
  scale_color_manual(values = c("Red", "Blue")) +
  ggtitle("Worst Predictions, by Home to First
  on Low Ground Balls") +
  theme(legend.position = c(.17, .92))

truefalse_by_hp_to_1b_data <- layer_data(truefalse_by_hp_to_1b)

hp_to_1b_TrueSingle_1quartile <- round(truefalse_by_hp_to_1b_data[1, 2], 2)
hp_to_1b_TrueSingle_median <- round(truefalse_by_hp_to_1b_data[1, 3], 2)
hp_to_1b_TrueSingle_3quartile <- round(truefalse_by_hp_to_1b_data[1, 4], 2)
hp_to_1b_FalseSingle_1quartile <- round(truefalse_by_hp_to_1b_data[2, 2], 2)
hp_to_1b_FalseSingle_median <- round(truefalse_by_hp_to_1b_data[2, 3], 2)
hp_to_1b_FalseSingle_3quartile <- round(truefalse_by_hp_to_1b_data[2, 4], 2)
hp_to_1b_TrueOut_1quartile <- round(truefalse_by_hp_to_1b_data[3, 2], 2)
hp_to_1b_TrueOut_median <- round(truefalse_by_hp_to_1b_data[3, 3], 2)
hp_to_1b_TrueOut_3quartile <- round(truefalse_by_hp_to_1b_data[3, 4], 2)
hp_to_1b_FalseOut_1quartile <- round(truefalse_by_hp_to_1b_data[4, 2], 2)
hp_to_1b_FalseOut_median <- round(truefalse_by_hp_to_1b_data[4, 3], 2)
hp_to_1b_FalseOut_3quartile <- round(truefalse_by_hp_to_1b_data[4, 4], 2)
```

```
(truefalse_by_hp_to_1b <- ggplot(predictions_2212_gb,
                                 aes(x = eval_type, y = hp_to_1b, color = events)) +
    geom_boxplot(color = "black", notch = TRUE, size = 1.1) +
    geom_point(position = "jitter", alpha = .7, size = 0.5) +
    scale_y_continuous(breaks = seq(3.9, 5.1, by = 0.1),
                       labels = c("3.9", "4.0", "4.1", "4.2", "4.3", "4.4", "4.5", "4.6",
                                  "4.7", "4.8",  "4.9", "5.0", "5.1"),
                       limits = c(3.9, 5.1)) +
    scale_color_manual(values = c("Red", "Blue")) +
    ggtitle("Worst Predictions, by Home to First
  on Low Ground Balls") +
    theme(legend.position = c(.17, .92)) +
    annotate("text", x = c(.60, .60, .60), size = 2.5, fontface = 2, hjust = 1,
             y = c(hp_to_1b_TrueSingle_1quartile, hp_to_1b_TrueSingle_median,
                   hp_to_1b_TrueSingle_3quartile),
             label = c(hp_to_1b_TrueSingle_1quartile, hp_to_1b_TrueSingle_median,
                       hp_to_1b_TrueSingle_3quartile)) +
    annotate("text", x = c(1.60,1.60, 1.60), size = 2.5, fontface = 2, hjust = 1,
```

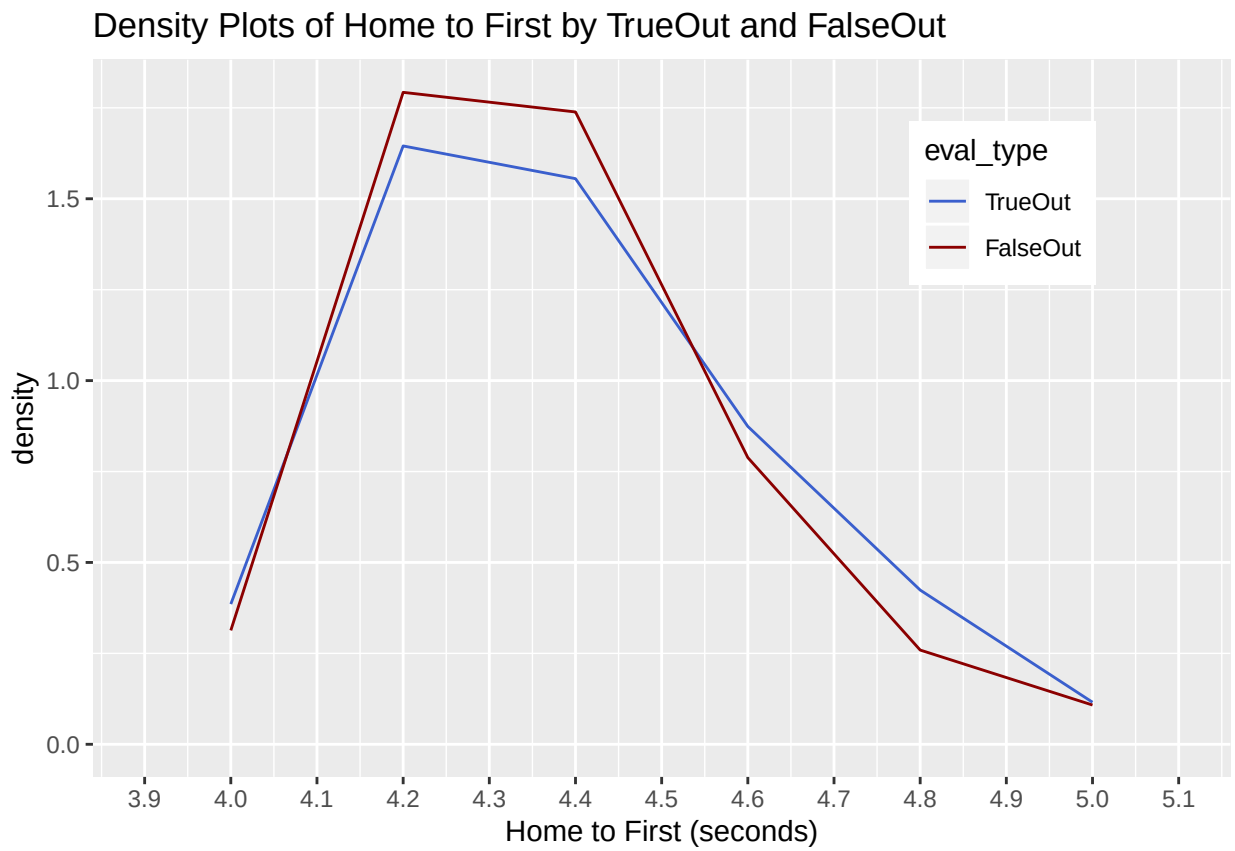
```
          y = c(hp_to_1b_FalseSingle_1quartile, hp_to_1b_FalseSingle_median,
                hp_to_1b_FalseSingle_3quartile),
          label = c(hp_to_1b_FalseSingle_1quartile, hp_to_1b_FalseSingle_median,
                    hp_to_1b_FalseSingle_3quartile)) +
annotate("text", x = c(2.60, 2.60, 2.60), size = 2.5, fontface = 2, hjust = 1,
          y = c(hp_to_1b_TrueOut_1quartile, hp_to_1b_TrueOut_median,
                hp_to_1b_TrueOut_3quartile),
          label = c(hp_to_1b_TrueOut_1quartile, hp_to_1b_TrueOut_median,
                    hp_to_1b_TrueOut_3quartile)) +
annotate("text", x = c(3.60, 3.60, 3.60), size = 2.5, fontface = 2, hjust = 1,
          y = c(hp_to_1b_FalseOut_1quartile, hp_to_1b_FalseOut_median,
                hp_to_1b_FalseOut_3quartile),
          label = c(hp_to_1b_FalseOut_1quartile, hp_to_1b_FalseOut_median,
                    hp_to_1b_FalseOut_3quartile)))
```



Here, the upper end of the interquartile range for false outs skewed a bit more toward faster times than the true outs, but the overall differences between the distributions appeared to be very small. Part of the problem may be the small scale of this predictor, with so many batters having hp_to_1b measurements

within a few tenths of a second from each other.

```
# Compare the densities of Home to First using our population of the worst
# 1,106 prediction mistakes and a random population of 1,106 correct predictions.

predictions_2212_lgb_TFOuts <-
  dplyr::filter(predictions_2212, bb_type == "ground_ball" &
                  adv_bb_type == "low_ground_ball")
predictions_2212_lgb_TFOuts <- predictions_2212_lgb_TFOuts %>%
  dplyr::filter(eval_type == "TrueOut" | eval_type == "FalseOut")

x_scale <- scale_x_continuous(breaks = seq(3.9, 5.1, by = 0.1),
                              labels = c("3.9", "4.0", "4.1", "4.2", "4.3", "4.4", "4.5",
                                         "4.6", "4.7", "4.8",  "4.9", "5.0", "5.1"),
                              limits = c(3.9, 5.1))
y_scale <- scale_y_continuous()

(h1b_density <- ggplot(predictions_2212_lgb_TFOuts,
                       aes(x = hp_to_1b, y = stat(density), color = eval_type)) +
    geom_freqpoly(binwidth = 0.2) +
    x_scale +
    y_scale +
    scale_color_manual(name = "eval_type", labels = c("TrueOut", "FalseOut"),
                       values = c("royalblue3", "red4")) +
    labs(title = "Density Plots of Home to First by TrueOut and FalseOut",
         x = "Home to First (seconds)") +
    theme(legend.position = c(.80, .80)))
```



The density plots seemed to magnify the tiny differences between hp_to_1b times for false outs and true outs. Here, we could see that the model had a more difficult time distinguishing singles from outs for the intermediate Home to First times, ranging from 4.2 to 4.5 seconds. This range coincides closely with the interquartile ranges of Home to First we just viewed. But the model was at least able to make use of the very fastest and very slowest Home to First measurements to enhance its predictive ability.

Results

We were satisfied with the results of our initial effort to develop a model capable of distinguishing singles from outs. There were two primary reasons for our satisfaction: 1) we were trying to predict a minority class (singles), which is a notorious challenge in the area of machine learning; and 2) our models intentionally considered only predictors that a batter could conceivably control, and thus didn't take into account the many external factors that can cause a batted ball to result in a single instead of an out.

In light of this, the rforest_train2 algorithm's true positive rate of 0.69, resulting in a Truescore of 0.80 and a distance of 0.315, seemed to us to be an accomplishment of sorts, creating a solid foundation for further research.

```
# Assess the predictive ability of the randomForest algorithm.

confusionMatrix(rforest_predict, test_set3$events)
```

```
## Confusion Matrix and Statistics
##
##           Reference
```

```
## Prediction  single field_out
##   single      3445       977
##   field_out   1536     14160
##
##                Accuracy : 0.8751
##                  95% CI : (0.8704, 0.8796)
##     No Information Rate : 0.7524
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6516
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.6916
##             Specificity : 0.9355
##          Pos Pred Value : 0.7791
##          Neg Pred Value : 0.9021
##              Prevalence : 0.2476
##          Detection Rate : 0.1712
##    Detection Prevalence : 0.2198
##       Balanced Accuracy : 0.8135
##
##        'Positive' Class : single
##
```

```
(rforest_tpr <- round(sensitivity(rforest_predict,
                                  reference = factor(test_set3$events)), 6))
```

```
## [1] 0.691628
```

```
(rforest_tnr <- round(specificity(rforest_predict,
                                  reference = factor(test_set3$events)), 6))
```

```
## [1] 0.935456
```

```
(rforest_fpr <- round(1 - rforest_tnr, 6))
```

```
## [1] 0.064544
```

```
(rforest_truescore <- round((2 * rforest_tpr * rforest_tnr) /
                              (rforest_tpr + rforest_tnr) , 6))
```

```
## [1] 0.795272
```

```
(rforest_distance <- round(sqrt((1 - rforest_tpr)^2 + (rforest_fpr)^2), 6))
```

```
## [1] 0.315054
```

```
assessment_results <- tibble(Model = c("Baseline", "Log. Regression",
                                       "Log. Regression", "KNN",
                                       "Class. Tree", "Random Forest"),
                             `Cutoff Method` = c(NA, "Truescore", "ROC Distance",
                                                 NA, NA, NA),
                             `Truescore` = c(bl_truescore, max_truescore,
                                             min_distance_truescore, knn_truescore,
                                             ctree_pruned_truescore, rforest_truescore),
                             TPR = c(bl_tpr, max_truescore_tpr, min_distance_tpr, knn_tpr,
                                     ctree_pruned_tpr, rforest_tpr),
                             TNR = c(bl_tnr, max_truescore_tnr, min_distance_tnr, knn_tnr,
                                     ctree_pruned_tnr, rforest_tnr),
                             ROC_Distance = c(NA, max_truescore_distance, min_distance,
                                              knn_distance, ctree_pruned_distance,
                                              rforest_distance),
                             FPR = c((1 - bl_tnr), max_truescore_fpr, min_distance_fpr,
                                     knn_fpr, ctree_pruned_fpr, rforest_fpr),
                             `Best Cutoff` = c(NA, max_truescore_cutoff,
                                               min_distance_cutoff, NA, NA, NA))
knitr::kable(assessment_results[1:6, ], caption = "Assessment Results")
```

Table 8: Assessment Results

Model	Cutoff Method	Truescore	TPR	TNR	ROC_Distance	FPR	Best Cutoff
Baseline	NA	0.385984	0.259787	0.750611	NA	0.249389	NA
Log. Regression	Truescore	0.408139	0.383256	0.436216	0.835599	0.563784	0.749984
Log. Regression	ROC Distance	0.399327	0.328448	0.509216	0.831776	0.490784	0.764628
KNN	NA	0.749196	0.635615	0.912202	0.374813	0.087798	NA
Class. Tree	NA	0.758219	0.636619	0.937240	0.368761	0.062760	NA
Random Forest	NA	0.795272	0.691628	0.935456	0.315054	0.064544	NA

As can be seen, our K-Nearest Neighbors Model and Classification Tree also performed well, but the Random Forest Model clearly distinguished itself.

Launch Angle repeatedly showed itself to be the most important of our predictors, followed by Launch Speed. With the help of Infield Alignment, Spray Angle also proved to be meaningful as it finished a respectable third in importance. With better quality Spray Angle data, we have no doubt Spray Angle can be even more informative. Home to First consistently appeared to be the least informative predictor, though the measure of its significance was significantly higher by the Random Forest Model (9%) than the Classification Tree Model (less than 1%). Ultimately, all of our chosen predictors contributed to the predictive ability of our best performing Random Forest Model.

Conclusion

Our journey was complete, yet it had hardly begun. From finding quality data and transforming it, to using it to make predictions about whether a batted ball was going to fall for a hit or ring up another out, every step of the way proved to be a valuable learning experience.

Of course, this study would not have been possible had MLB not invested in the advanced technology that could turn so much of what goes on on a baseball field into a pile of numbers. And still, it seems so much more can be done with regard to collecting data and making it available to the public. We have to keep reminding ourselves that Baseball Savant does not exist to furnish data scientists with sustenance for research. But

in the course of performing its real mission of providing common fans with fun and interesting metrics, it comes so darn close to feeding us analysts that we can taste it!

Understandably, Baseball Savant has focused on Launch Angle and Exit Velocity as the primary predictors of batting results. But these two predictors don't come close to explaining all that we would like to know. It's hoped that someday, corporate priorities will turn their attention toward the admittedly more complex predictor, Spray Angle. For this study, we were able to scrape enough pieces of data together and manipulate it in enough ways to paint a hazy picture of Spray Angle's impact on the outcome of a batted ball. But this is an area where the quality of data could be significantly improved.

In the context of this study, the random forest algorithm demonstrated superior performance. It recognized a single to be a single 69% of the time. It found some features that allowed it to distinguish a single from an out. But faced with the challenge of a binary outcome with unbalanced classes, we bumped into an obstacle that, with further study, might be overcome. And that is how the black box algorithms such as knn and the tree-based algorithms, after going through all the trouble of computing precise probability estimates, seem to blindly rest their final decision on a simple majority vote. Circumstances, such as those presented by unbalanced outcomes, warrant more flexibility in selecting decision points and criteria for evaluation. Had we more time, we would have searched for techniques that could have overcome at least some of the rigidity of these types of algorithms. They likely exist, but we simply ran out of time.

There are numerous machine-learning algorithms that were not tested during this study. It's likely that one of these untested algorithms can outperform our application of random forest. This would be another area for further research.