Called Strike Prediction Model

Alex Trudel

5/28/2021

Load in the data

```
pitches <- read.csv("pitch_data_one_month.csv")
require(dplyr)
```

```
## Loading required package: dplyr
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

Question 1 - Create a Model to Predict the Probability that the Umpire Calls a Strike

I will not be running output for most of my cells or else this document would be longer than it already is.

Model Parameters

Cleaning the Data

The data set for this model included all pitch events thrown in a given month, which meant that I first had to remove all of the rows in which the batter swung, so that I would only be left with pitches that were either called balls or called strikes. Also, I created a new variable, `CS_OR_NOT`, which is simply a recoding of the simplified outcome variables. It is `YES` for a called strike and `NO` for a called ball. Finally, I removed all the rows where pitches were identified as either undefined (`UN`) or pitchout (`PO`) because, in the case of the former, predicting an undefined entity is generally not useful, and in the case of the latter, a pitchout will always be a called ball (or *maybe* a swinging strike).

```
pitches <- pitches[- which(is.na(pitches$SPIN_RATE)),]
pitches <- pitches[- which(is.na(pitches$PITCH_RESULT)),]

called_pitches <- pitches[pitches$PITCH_RESULT == "BallCalled" |
                            pitches$PITCH_RESULT == "StrikeCalled",]

called_pitches$CS_OR_NOT <- ifelse(called_pitches$PITCH_RESULT == "StrikeCalled", "YES", "NO")
called_pitches <- called_pitches %>% filter(! called_pitches$PITCH_TYPE %in% c("UN", "PO"))
```

Next, I removed all outcome-based variables from the data set, along with some other variables that I did not think would be relevant. Pitch result was recoded as CS_OR_NOT, event result, exit velo, launch angle, and spray angle were all irrelevant for this question, and pitcher side (handedness) was a worse version of the numerical variable `RELEASE_SIDE`.

```
called_pitches <- called_pitches %>% select(! c("PITCH_RESULT", "EVENT_RESULT",
                                                "PITCHER_SIDE", "EXIT_VELOCITY",
                                                "LAUNCH_ANGLE", "SPRAY_ANGLE"))
```

Creation of New Variables: Called Strike Rate for Each Third of the Zone (and out of the Zone)

Strike zones in media are generally split into 3x3 grids, so I followed that convention for this data. The more you further divide the plate, the more specific you can get with your guesses, but you also lose some interpretability. In general, I am more of a fan of interpretable parameters, especially when using nearly un-interpretable models, like I will later on.

Splitting the plate into thirds width-wise was easy, because the dimensions of the plate are standard. Once I found the width of 1/3rd of the plate, I assigned pitches into their respective zone by dividing their `PITCH_LOCATION_SIDE` attribute by the width of the plate, then rounding down to the nearest whole number. One thing to note is that I added 1/6th of the width of the plate to every pitch location, because if not, then the zones would be shifted by 1/6th of the width of the plate in the negative direction.

Splitting the plate into thirds height-wise was a little more difficult, because the height of the strike zone is not uniform. However, I was able to calculate this by subtracting the bottom of the strike zone from the top, then dividing that number by 3. Then, like the width values, I converted the measurements to zones by dividing the `PITCH_LOCATION_HEIGHT` by the 1/3rd of the height of the strike zone.

Next, I found that most values (in width and height) fell within zones 0-3, so I converted anything greater than those numbers to 3 or -3.

```
plate_width_third <- 17/12/3

called_pitches$PITCH_HEIGHT_THIRD <- (called_pitches$STRIKE_ZONE_TOP -
                                      called_pitches$STRIKE_ZONE_BOTTOM)/3

called_pitches$PITCH_HEIGHT_ZONE <- floor((called_pitches$PITCH_LOCATION_HEIGHT -
                                           mean(called_pitches$PITCH_LOCATION_HEIGHT)) /
                                              called_pitches$PITCH_HEIGHT_THIRD)

called_pitches$PITCH_WIDTH_ZONE <- floor((called_pitches$PITCH_LOCATION_SIDE +
                                         (.5 * plate_width_third)) / plate_width_third)

called_pitches[called_pitches$PITCH_HEIGHT_ZONE < -2,]$PITCH_HEIGHT_ZONE <- -3
called_pitches[called_pitches$PITCH_HEIGHT_ZONE > 2,]$PITCH_HEIGHT_ZONE <- 3

called_pitches[called_pitches$PITCH_WIDTH_ZONE < -2,]$PITCH_WIDTH_ZONE <- -3
called_pitches[called_pitches$PITCH_WIDTH_ZONE > 2,]$PITCH_WIDTH_ZONE <- 3
```

Then, I created a 7x7 matrix representing the expanded strike zone, and populated it with the called strike rate for each zone. As you can see, the assignments worked correctly because most pitches *inside* the strike zone are called strikes (>90%), and some pitches *outside* the strike zone are called strikes (0-50%). Finally, for each individual pitch's location, I assigned it the corresponding value of its zone's called strike rate.

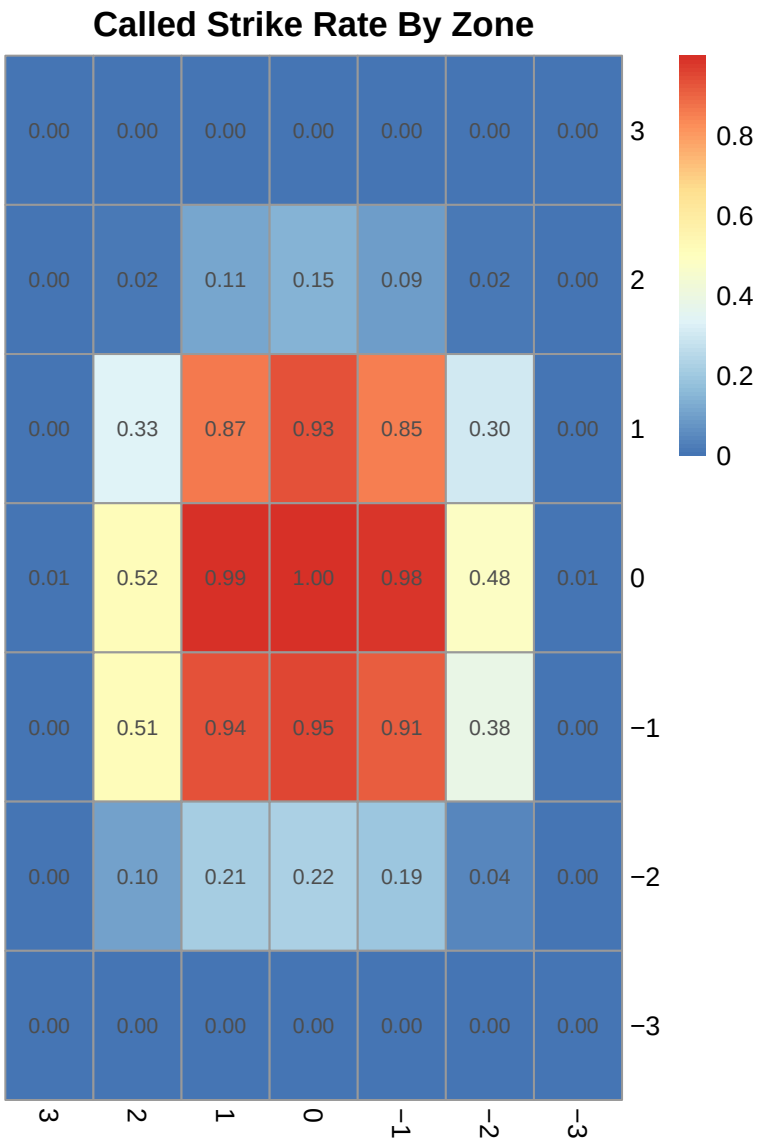
```
cs_rate_per_area <- matrix(0, nrow = 7, ncol = 7)
```

```
for (i in 1:7) {
  for (j in 1:7) {
    cs_rate_per_area[i,j] <- nrow(called_pitches[called_pitches$PITCH_HEIGHT_ZONE == i-4 &
                                                  called_pitches$PITCH_WIDTH_ZONE == j-4 &
                                                  called_pitches$CS_OR_NOT == "YES",]) /
                             nrow(called_pitches[called_pitches$PITCH_HEIGHT_ZONE == i-4 &
                                                   called_pitches$PITCH_WIDTH_ZONE == j-4,])
  }
}

called_pitches$cs_rate_per_area <- 0
for (i in 1:nrow(called_pitches)) {
  called_pitches$cs_rate_per_area[i] <- cs_rate_per_area[(called_pitches$PITCH_HEIGHT_ZONE[i] + 4),
                                                         (called_pitches$PITCH_WIDTH_ZONE[i] + 4)]
}

require(pheatmap)
```

```
## Loading required package: pheatmap
```

```
## Warning: package 'pheatmap' was built under R version 4.0.5
```

```
d <- as.data.frame(cs_rate_per_area[7:1,])

rownames(d) <- 3:-3
colnames(d) <- 3:-3
pheatmap(d, display_numbers = T, cluster_rows = F, cluster_cols = F, width = 1, height = 3,
         main = "Called Strike Rate By Zone")
```



Unsurprisingly, this was by far the best variable in the data by predicting power. This could be improved by further narrowing the strike zone, i.e. making a 6x6 strike zone instead of 3x3, but as I mentioned earlier, I left it relatively simple to keep it in a standardized format.

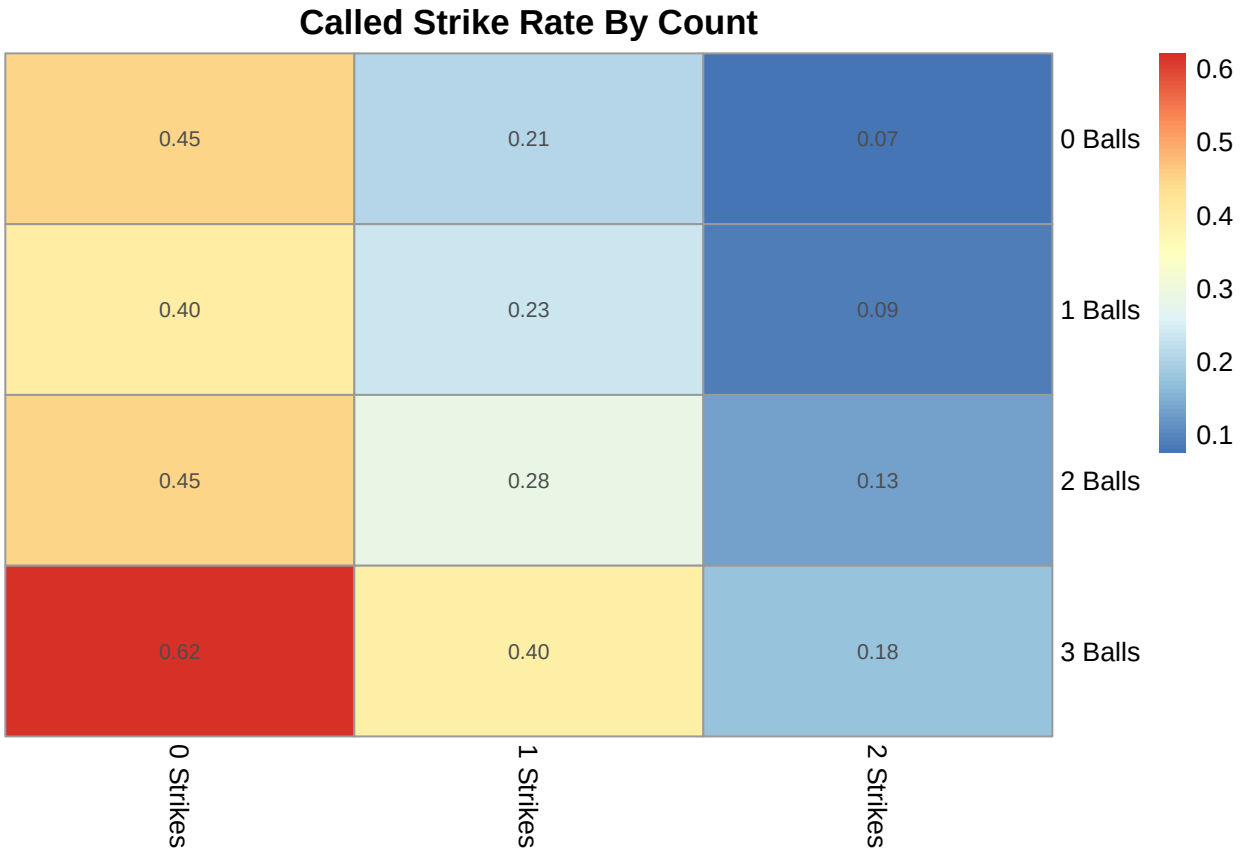
Same as above but with counts

Another important factor in called strike rate comes with counts. A batter is much less likely to watch strike 3 on an 0-2 count than they are to watch strike 1 on a 3-0 count, for example.

The procedure is mostly the same as above, except it is much simpler.

```
cs_rate_per_count <- matrix(0, nrow = 4, ncol = 3)
for (i in 1:4) {
  for (j in 1:3) {
    cs_rate_per_count[i,j] <- nrow(called_pitches[called_pitches$BALLS == i-1 &
                                                  called_pitches$STRIKES == j-1 &
                                                  called_pitches$CS_OR_NOT == "YES",]) /
                              nrow(called_pitches[called_pitches$BALLS == i-1 &
```

```
                                                    called_pitches$STRIKES == j-1,])
  }
}


called_pitches$cs_rate_per_count <- 0
for (i in 1:nrow(called_pitches)) {
  called_pitches$cs_rate_per_count[i] <- cs_rate_per_count[(called_pitches$BALLS[i] + 1),
                                                           (called_pitches$STRIKES[i] + 1)]
}

e <- as.data.frame(cs_rate_per_count)
rownames(e) <- paste(0:3, "Balls",  sep = " ")
colnames(e) <- paste(0:2, "Strikes", sep = " ")
pheatmap(e, display_numbers = T, cluster_rows = F, cluster_cols = F, width = 1, height = 3,
         main = "Called Strike Rate By Count")
```

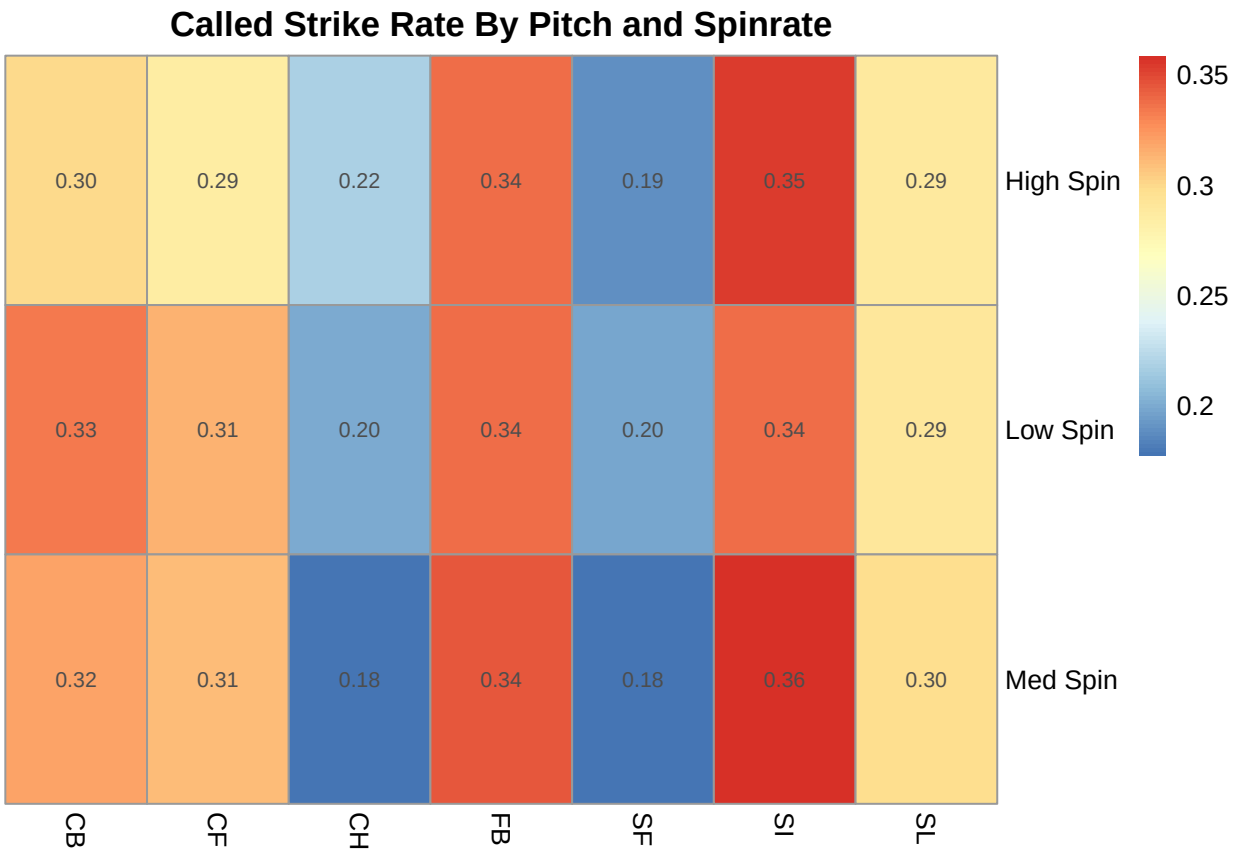


Spin Rates

I assumed that spin rates would be an important variable for borderline balls being called strikes. A slider with late movement can appear to catch a corner when in reality it just surrounds the plate, for example.

To test this, I found (for each pitch type) the 25th and 75th percentile spin rate, then classified all of the pitches in the data as either high spin, low spin, or medium spin. Then, for each subset, I found the called strike rate and assigned it as a variable to each pitch.

```r
called_pitches <- called_pitches %>% group_by(PITCH_TYPE) %>% mutate(q25 = quantile(SPIN_RATE, 0.25),
                                                                     q75 = quantile(SPIN_RATE, 0.75))

called_pitches$HiLoSpin <- 0
called_pitches$HiLoSpin <- ifelse(called_pitches$SPIN_RATE < called_pitches$q25, "Low",
                           ifelse(called_pitches$SPIN_RATE > called_pitches$q75, "High", "Mid"))

cs_rate_by_pitch_type_and_spinrate <- matrix(0, nrow = 3, ncol = 7)

for (i in 1:3) {
  for (j in 1:7) {
    cs_rate_by_pitch_type_and_spinrate[i, j] <-
      nrow(called_pitches[as.numeric(as.factor(called_pitches$HiLoSpin)) == i &
                    as.numeric(as.factor(called_pitches$PITCH_TYPE)) == j &
                    called_pitches$CS_OR_NOT == "YES",]) /
                       nrow(called_pitches[as.numeric(as.factor(called_pitches$HiLoSpin)) == i &
                                           as.numeric(as.factor(called_pitches$PITCH_TYPE)) == j,])
  }
}

afspin <- as.numeric(as.factor(called_pitches$HiLoSpin))
afpt <- as.numeric(as.factor(called_pitches$PITCH_TYPE))

called_pitches$cs_rate_by_pitch_type_spinrate <- 0

for (i in 1:nrow(called_pitches)) {
  called_pitches$cs_rate_by_pitch_type_spinrate[i] <-
    cs_rate_by_pitch_type_and_spinrate[afspin[i], afpt[i]]
}

f <- as.data.frame(cs_rate_by_pitch_type_and_spinrate)
rownames(f) <- paste(c("High", "Low", "Med"), "Spin", sep = " ")
colnames(f) <- levels(as.factor(called_pitches$PITCH_TYPE))
pheatmap(f, display_numbers = T, cluster_rows = F, cluster_cols = F, width = 1, height = 3,
         main = "Called Strike Rate By Pitch and Spinrate")
```

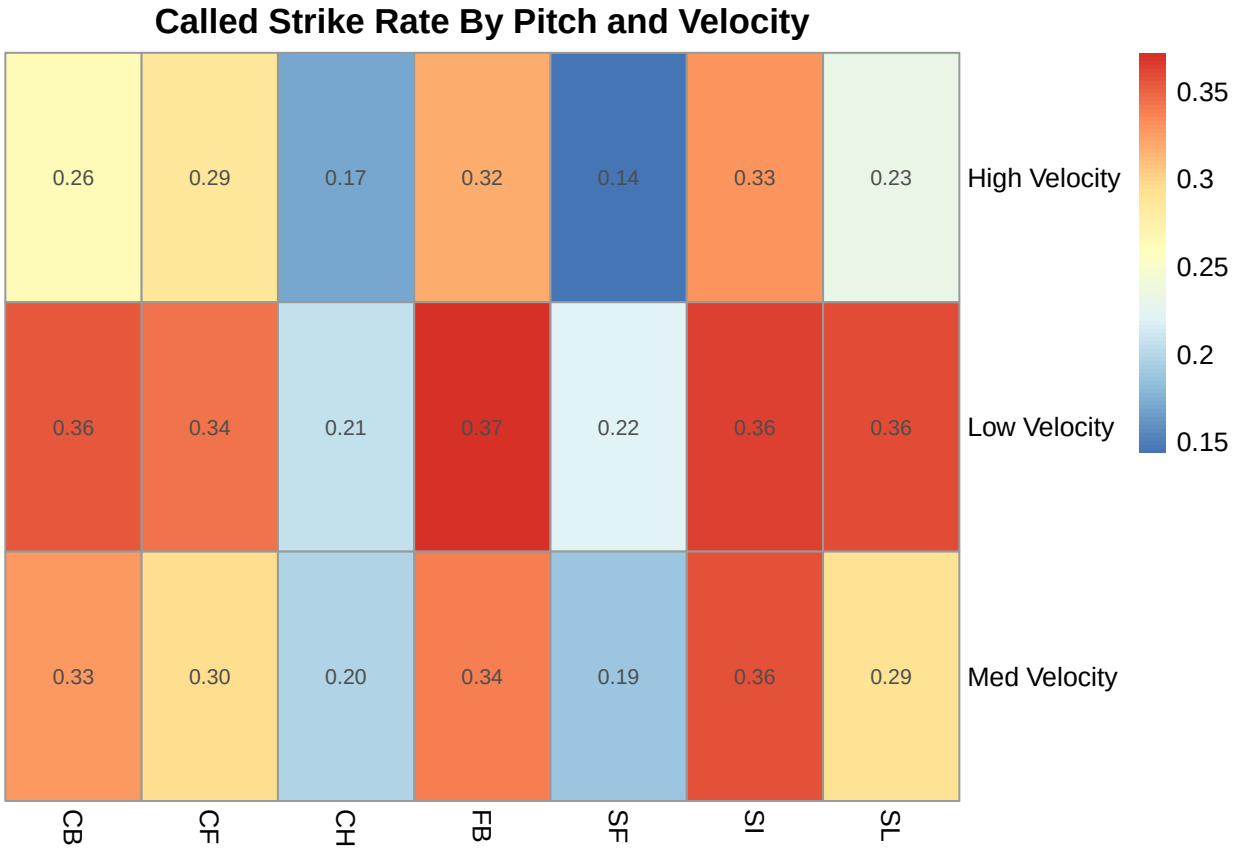


As we can see from the plot, it doesn't seem that spinrate affects called strike rate. However, we can gain some important information from this chart: Fastballs and breaking balls are taken at higher rates than changeups. I accounted for this by repeating this process with velocity, and making sure to include horizontal/vertical break into the final model.

Same as above, but with velocity

```
called_pitches <- called_pitches %>% group_by(PITCH_TYPE) %>%
  mutate(vq25 = quantile(PITCH_SPEED, 0.25), vq75 = quantile(PITCH_SPEED, 0.75))

called_pitches$HiLoVelo <- 0
called_pitches$HiLoVelo <- ifelse(called_pitches$PITCH_SPEED < called_pitches$vq25, "Low",
                           ifelse(called_pitches$PITCH_SPEED > called_pitches$vq75, "High", "Mid"))

cs_rate_by_pitch_type_and_velocity <- matrix(0, nrow = 3, ncol = 7)

for (i in 1:3) {
  for (j in 1:7) {
    cs_rate_by_pitch_type_and_velocity[i, j] <-
      nrow(called_pitches[as.numeric(as.factor(called_pitches$HiLoVelo)) == i &
                    as.numeric(as.factor(called_pitches$PITCH_TYPE)) == j &
                    called_pitches$CS_OR_NOT == "YES",]) /
                         nrow(called_pitches[as.numeric(as.factor(called_pitches$HiLoVelo)) == i &
                                             as.numeric(as.factor(called_pitches$PITCH_TYPE)) == j,])
  }
}
```

```r
afvelo <- as.numeric(as.factor(called_pitches$HiLoVelo))

called_pitches$cs_rate_by_pitch_type_velocity <- 0

for (i in 1:nrow(called_pitches)) {
  called_pitches$cs_rate_by_pitch_type_velocity[i] <-
    cs_rate_by_pitch_type_and_velocity[afvelo[i], afpt[i]]
}


g <- as.data.frame(cs_rate_by_pitch_type_and_velocity)
rownames(g) <- paste(c("High", "Low", "Med"), "Velocity", sep = " ")
colnames(g) <- levels(as.factor(called_pitches$PITCH_TYPE))
pheatmap(g, display_numbers = T, cluster_rows = F, cluster_cols = F, width = 1, height = 3,
         main = "Called Strike Rate By Pitch and Velocity")
```



These results actually surprised me. I can only assume that pitches with lower velocity have some combination of higher precision (on the corners) and more movement, which causes the batter to take instead of swing.

Whether or not a pitch is actually a strike

This was mostly just helpful for the next section about umpires and one later on about catcher framing. It is simply a binary variable that is "YES" if the pitch is in the strike zone, and "NO" if it isn't. The number .7083 corresponds to the width of half the plate (8.5 inches) in feet, because that is how the data is stored.

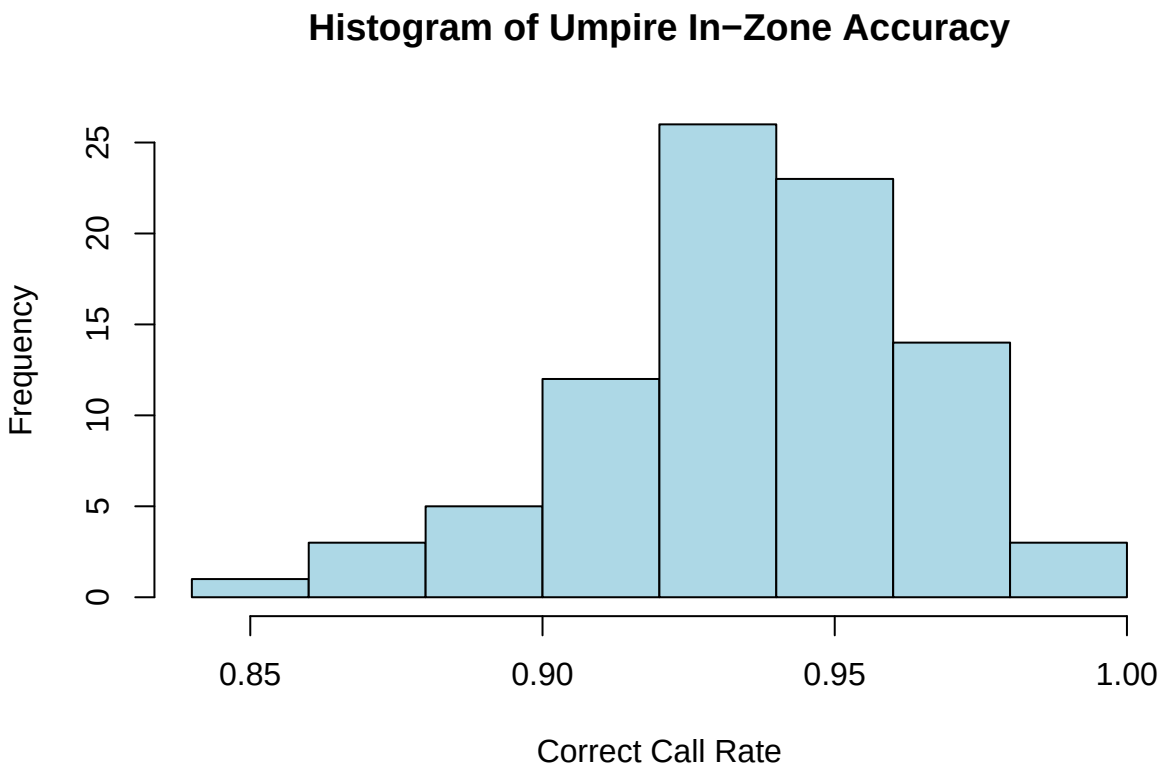
```
called_pitches$ACTUAL_STRIKE <- ifelse(
  (called_pitches$PITCH_LOCATION_HEIGHT > called_pitches$STRIKE_ZONE_BOTTOM &
      called_pitches$PITCH_LOCATION_HEIGHT < called_pitches$STRIKE_ZONE_TOP &
      called_pitches$PITCH_LOCATION_SIDE > -.7083 & called_pitches$PITCH_LOCATION_SIDE < .7083),
      "YES", "NO")
```

Umpire In-Zone CSR (Higher is Better); Out of Zone CSR (Lower is Better)

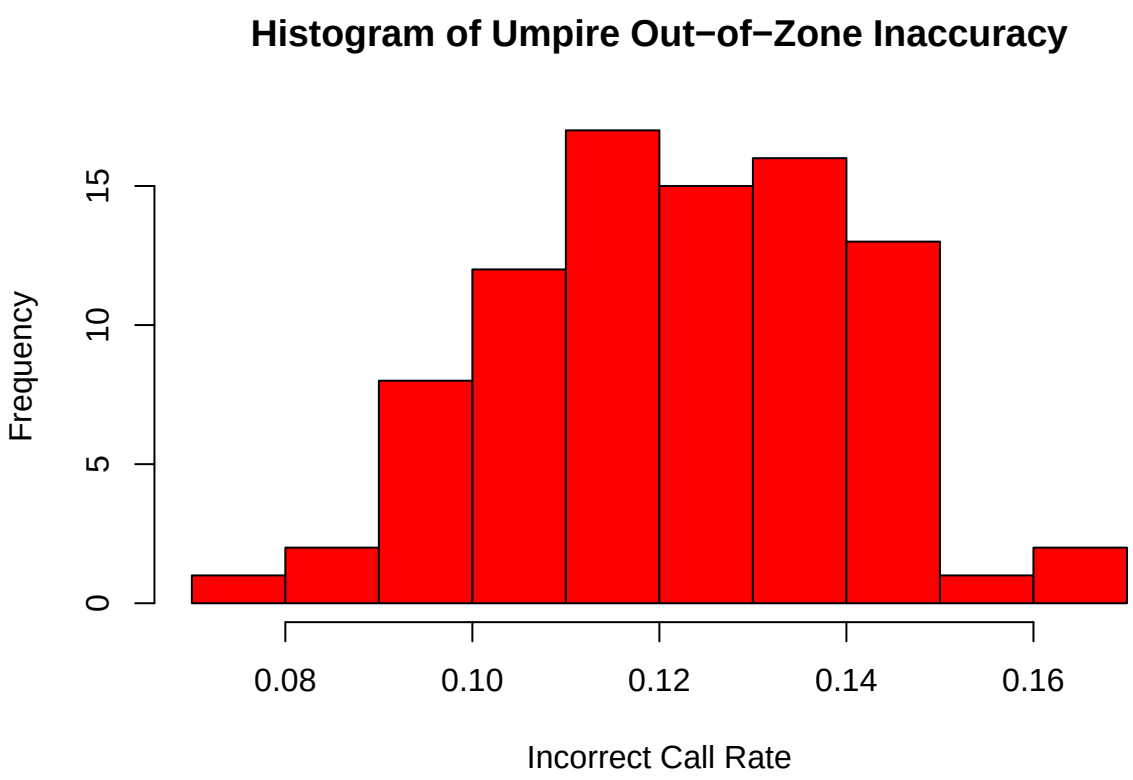
Much has been made about switching to Robo-Umps due to their inconsistency and inaccuracy. Here I try to account for this by finding each umpire's correct call rate in the zone (strikes are good) and blown call rate out of the zone (strikes are bad).

Since there are 87 umpires in the data set, I'll show histograms instead of heatmaps.

```
called_pitches$umpire_in_zone_csr <- 0
called_pitches$umpire_out_zone_csr <- 0

iz_csr <- rep(0, 87)
oz_csr <- rep(0, 87)

for (i in 1:87) {
  iz_csr[i] <- nrow(called_pitches[called_pitches$UMPIRE_ID == i &
                                   called_pitches$CS_OR_NOT == "YES" &
                                   called_pitches$ACTUAL_STRIKE == "YES",]) /
               nrow(called_pitches[called_pitches$UMPIRE_ID == i &
                                   called_pitches$ACTUAL_STRIKE == "YES",])
}

for (i in 1:87) {
  oz_csr[i] <- nrow(called_pitches[called_pitches$UMPIRE_ID == i &
                                   called_pitches$CS_OR_NOT == "YES" &
                                   called_pitches$ACTUAL_STRIKE == "NO",]) /
               nrow(called_pitches[called_pitches$UMPIRE_ID == i &
                                   called_pitches$ACTUAL_STRIKE == "NO",])
}

for (i in 1:nrow(called_pitches)) {
  called_pitches$umpire_in_zone_csr[i] <- iz_csr[called_pitches$UMPIRE_ID[i]]
}

for (i in 1:nrow(called_pitches)) {
  called_pitches$umpire_out_zone_csr[i] <- oz_csr[called_pitches$UMPIRE_ID[i]]
}

hist(iz_csr, main = "Histogram of Umpire In-Zone Accuracy", xlab = "Correct Call Rate",
     col = "light blue")
```




```
hist(oz_csr, main = "Histogram of Umpire Out-of-Zone Inaccuracy", xlab = "Incorrect Call Rate",
     col = "red")
```



Gamestate (Blowout or Not?)

It is generally assumed that an umpire's strike zone increases if one team is winning in a blowout, so I created a variable to test if this was true.

```
called_pitches$blowout <- "NO"

called_pitches$blowout <- ifelse(called_pitches$INNING >= 3 &
                           abs(called_pitches$HOME_TEAM_SCORE -
                               called_pitches$AWAY_TEAM_SCORE) > 6, "YES", "NO")

blowout_cs_rate <- nrow(called_pitches[called_pitches$blowout == "YES" &
                                       called_pitches$CS_OR_NOT == "YES",]) /
                     nrow(called_pitches[called_pitches$blowout == "YES",])

no_blowout_cs_rate <- nrow(called_pitches[called_pitches$blowout == "NO" &
                                          called_pitches$CS_OR_NOT == "YES",]) /
                       nrow(called_pitches[called_pitches$blowout == "NO",])

blowout_cs_rate
```

```
## [1] 0.3178381
```

```
no_blowout_cs_rate
```

```
## [1] 0.3121127
```

It was not.

Convert numeric factors to factors

At this point, I decided to do some more housekeeping, as I changed all numeric variables that are actually factors (Balls, Strikes, Outs, etc.) into factors. Most of these ended up being irrelevant for the model, but were necessary for creating future variables.

```
called_pitches$BALLS <- as.factor(called_pitches$BALLS)
called_pitches$STRIKES <- as.factor(called_pitches$STRIKES)
called_pitches$OUTS <- as.factor(called_pitches$OUTS)

called_pitches[called_pitches$PITCH_IN_PA >= 7,]$PITCH_IN_PA <- 7
called_pitches$PITCH_IN_PA <- as.factor(called_pitches$PITCH_IN_PA)
levels(called_pitches$PITCH_IN_PA) <- c("1", "2", "3", "4", "5", "6", "7+")

called_pitches$INNING <- as.factor(called_pitches$INNING)
called_pitches$RUNNER_ON_1B <- as.factor(called_pitches$RUNNER_ON_1B)
called_pitches$RUNNER_ON_2B <- as.factor(called_pitches$RUNNER_ON_2B)
called_pitches$RUNNER_ON_3B <- as.factor(called_pitches$RUNNER_ON_3B)
called_pitches$CS_OR_NOT <- as.factor(called_pitches$CS_OR_NOT)
called_pitches$BATTER_SIDE <- as.factor(called_pitches$BATTER_SIDE)
```

Absolute Value of `Pitch Location Side`

Just the absolute value of the `PITCH_LOCATION_SIDE` variable. It effectively tells how far off the plate any given pitch landed, and has high predictive power in my models. It is somewhat correlated with `cs_rate_per_area`, but not enough to make me want to consider eliminating either one.

```
called_pitches$apls <- abs(called_pitches$PITCH_LOCATION_SIDE)
```

Catcher Framing

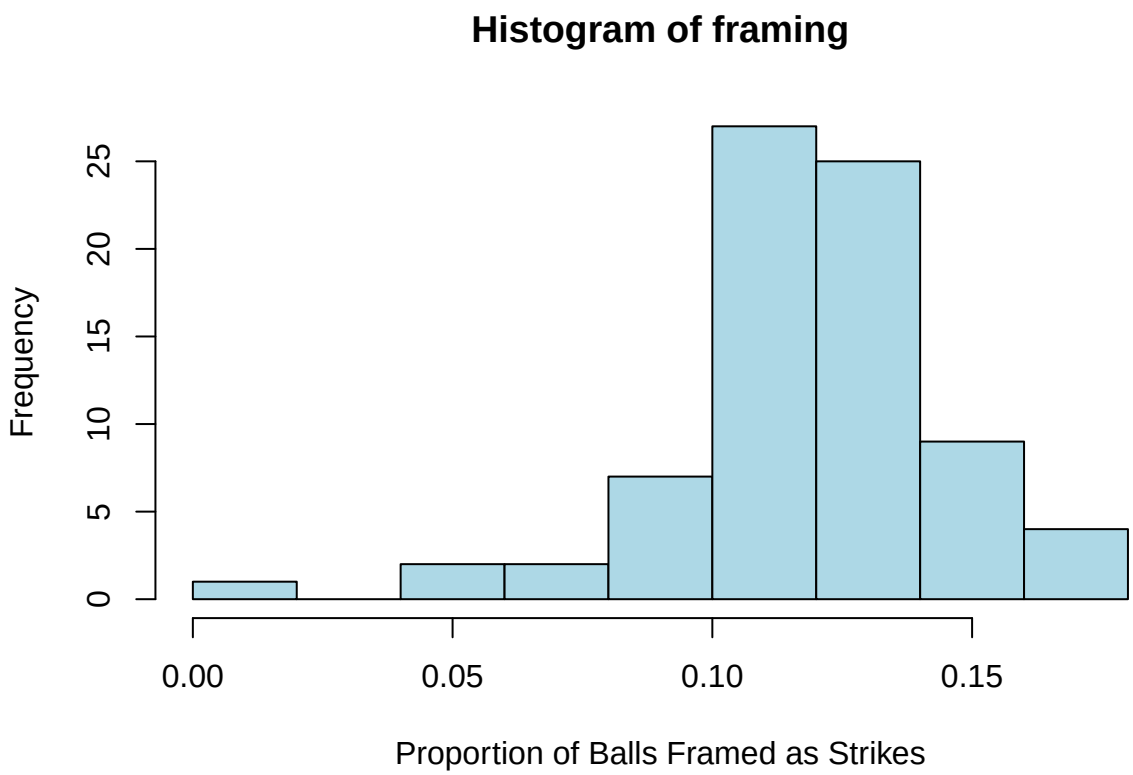
While framing is more important than just stealing strikes, this is what this variable calculates. For each catcher in the data set, it calculates percentage of pitches they caught that *were* balls but *called* strikes. Admittedly, there is no guarantee that this variable is a metric of catcher framing as it is of umpire incompetence, but any catcher with a rate $> 12\%$ is probably a passable framer.

```
frame <- called_pitches %>% group_by(CATCHER_ID, CS_OR_NOT, ACTUAL_STRIKE, .drop = FALSE) %>%
  summarise(count = n())
```

```
## `summarise()` regrouping output by 'CATCHER_ID', 'CS_OR_NOT' (override with `.groups` argument)
```

```
frame$CS_OR_NOT <- as.character(frame$CS_OR_NOT)

framing <- rep(0, 77)

for (i in 1:77) {
  framing[i] <- nrow(called_pitches[called_pitches$CATCHER_ID == i &
                  called_pitches$CS_OR_NOT == "YES" & called_pitches$ACTUAL_STRIKE == "NO",]) /
                    nrow(called_pitches[called_pitches$CATCHER_ID == i &
                                  called_pitches$ACTUAL_STRIKE == "NO",])
}

called_pitches$catcher_framing <- 0

for (i in 1:nrow(called_pitches)) {
  called_pitches$catcher_framing[i] <- framing[called_pitches$CATCHER_ID[i]]
}
```

```
hist(framing, col = "light blue", xlab = "Proportion of Balls Framed as Strikes")
```



Pitcher CSR

Another player-specific variable, this one calculates the called strike rate for each pitcher in the data. While there is risk for overfitting, I believe that since the dataset is universal, and we have past/minor league data for all players, the creation and usage of this variable (and the next one for the hitters) is warranted.

With that said, there were two pitchers who did not have a "called" pitch in this data set, and a fair amount with less than 20 total events. I don't know when called strike rate stabilizes, but I doubt it's at 20 pitches. For these players, I simply combined them into two players – a right-handed pitcher and a left-handed one. While all pitchers have different skillsets, I felt confident grouping these players together because they are all most likely minor-league call-ups who pitched an inning or two before being sent back down. With that in mind, creating an aggregate "AAA+" pitcher out of many other "AAA+" pitchers felt appropriate.

This process involved removing and shifting many of the player ID numbers, but no data was lost. For the purposes of Part 4 of this question, the right-handed pitcher is ID #483, and the lefty is #511.

```
called_pitches[called_pitches$PITCHER_ID > 182,]$PITCHER_ID <-
  called_pitches[called_pitches$PITCHER_ID > 182,]$PITCHER_ID - 1
called_pitches[called_pitches$PITCHER_ID > 225,]$PITCHER_ID <-
  called_pitches[called_pitches$PITCHER_ID > 225,]$PITCHER_ID - 1

lowpitches <- called_pitches %>% group_by(PITCHER_ID) %>% summarise(count = n()) %>% filter(count < 20)
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
lownums <- lowpitches$PITCHER_ID

combined_lownums <- called_pitches[called_pitches$PITCHER_ID %in% lownums,]
lownum_lefties <- combined_lownums[combined_lownums$RELEASE_SIDE > 0,]
lownum_righties <- combined_lownums[combined_lownums$RELEASE_SIDE < 0,]

ll_id <- unique(lownum_lefties$PITCHER_ID)
rr_id <- unique(lownum_righties$PITCHER_ID)

called_pitches[called_pitches$PITCHER_ID %in% ll_id,]$PITCHER_ID <- 553
called_pitches[called_pitches$PITCHER_ID %in% rr_id,]$PITCHER_ID <- 525

called_pitches$PITCHER_ID <- as.factor(called_pitches$PITCHER_ID)
levels(called_pitches$PITCHER_ID) <- sort(as.numeric(as.factor(unique(called_pitches$PITCHER_ID))))


pcsr <- rep(0, 512)

for (i in c(1:512)) {
  pcsr[i] <- nrow(called_pitches[called_pitches$PITCHER_ID == i &
                                   called_pitches$CS_OR_NOT == "YES",]) /
              nrow(called_pitches[called_pitches$PITCHER_ID == i,])

  if(is.infinite(pcsr[i])){
    pcsr[i] <- 0
  }

  if(is.nan(pcsr[i])) {
    pcsr[i] <- 0
  }
}


called_pitches$pitcher_called_strike <- 0

for (i in 1:nrow(called_pitches)) {
  called_pitches$pitcher_called_strike[i] <- pcsr[called_pitches$PITCHER_ID[i]]
}
```

```
called_pitches_backup <- called_pitches
```

Hitter Tendencies

This is effectively the same as above.

Hitters IDs that are associated with less than 20 pitches are aggregated then randomly combined based on handedness. There were significantly more hitters with few events than pitchers, so 10 aggregate players were created (leaving 438 real ones) with about 140 (righties) and 180(lefties) events apiece.

```
called_pitches <- called_pitches_backup
called_pitches$BATTER_ID <- as.factor(called_pitches$BATTER_ID)
levels(called_pitches$BATTER_ID) <- sort(as.numeric(as.factor(unique(called_pitches$BATTER_ID))))

called_pitches$BATTER_ID <- as.numeric(called_pitches$BATTER_ID)
```

```
lowhitters <- called_pitches %>% group_by(BATTER_ID) %>% summarise(count = n()) %>%
  filter(count < 20)
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
lownums_h <- lowhitters$BATTER_ID
combined_h_lownums <- called_pitches[ called_pitches$BATTER_ID %in% lownums_h,]
called_pitches <- called_pitches[! called_pitches$BATTER_ID %in% lownums_h,]

h_lownum_lefties <- combined_h_lownums[combined_h_lownums$BATTER_SIDE == "L",]
h_lownum_righties <- combined_h_lownums[combined_h_lownums$BATTER_SIDE == "R",]

h_lownum_lefties[1:180,]$BATTER_ID <- 602
h_lownum_lefties[181:361,]$BATTER_ID <- 588

h_lownum_righties[1:141,]$BATTER_ID <- 609
h_lownum_righties[142:283,]$BATTER_ID <- 611
h_lownum_righties[284:425,]$BATTER_ID <- 594
h_lownum_righties[426:567,]$BATTER_ID <- 590
h_lownum_righties[568:709,]$BATTER_ID <- 626
h_lownum_righties[710:851,]$BATTER_ID <- 616
h_lownum_righties[852:993,]$BATTER_ID <- 597
h_lownum_righties[994:1131,]$BATTER_ID <- 444

combined_h_lownums <- rbind(h_lownum_lefties, h_lownum_righties)

called_pitches <- rbind(called_pitches, combined_h_lownums)

called_pitches$BATTER_ID <- as.factor(called_pitches$BATTER_ID)
levels(called_pitches$BATTER_ID) <- sort(as.numeric(as.factor(unique(called_pitches$BATTER_ID))))

bcsr <- rep(0, 448)

for (i in c(1:448)) {
  bcsr[i] <- nrow(called_pitches[called_pitches$BATTER_ID == i &
                                   called_pitches$CS_OR_NOT == "YES",]) /
             nrow(called_pitches[called_pitches$BATTER_ID == i,])

  if(is.infinite(bcsr[i])){
    bcsr[i] <- 0
  }

  if(is.nan(pcsr[i])) {
    bcsr[i] <- 0
  }
}


called_pitches$batter_called_strike <- 0

for (i in 1:nrow(called_pitches)) {
  called_pitches$batter_called_strike[i] <- bcsr[called_pitches$BATTER_ID[i]]
}
```

Make sure numeric data is Normally Distributed

For the numeric features that I like to make sure that they are at least approximately Normally distributed. While it wasn't relevant for the models I used in this exploration, I think it is good practice to do anyway. There is, however, a significant loss of interpretability.

```
require(car)
```

```
## Loading required package: car
```

```
## Loading required package: carData
```

```
##
## Attaching package: 'car'
```

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

```
powerTransform(called_pitches$PITCH_SPEED)
```

```
## Estimated transformation parameter
## called_pitches$PITCH_SPEED
##                   5.105865
```

```
called_pitches$PITCH_SPEED <- called_pitches$PITCH_SPEED^5

powerTransform((called_pitches$INDUCED_VERTICAL_BREAK + 25))
```

```
## Estimated transformation parameter
## (called_pitches$INDUCED_VERTICAL_BREAK + 25)
##                                    2.116303
```

```
called_pitches$INDUCED_VERTICAL_BREAK <- (called_pitches$INDUCED_VERTICAL_BREAK + 25)^2.1

powerTransform(abs(called_pitches$HORIZONTAL_BREAK))
```

```
## Estimated transformation parameter
## abs(called_pitches$HORIZONTAL_BREAK)
##                             0.6715747
```

```
called_pitches$abshorizontalbreak <- (abs(called_pitches$HORIZONTAL_BREAK))^(2/3)

powerTransform(called_pitches$RELEASE_HEIGHT)
```

```
## Estimated transformation parameter
## called_pitches$RELEASE_HEIGHT
##                      3.768623
```

```
called_pitches$RELEASE_HEIGHT <- called_pitches$RELEASE_HEIGHT^3.75
```

Training and Testing Data

I normally at the very least split my data into a training and testing data set, but since the question asks for the pitcher who lost the most strikes relative to expectation, I will be using the full model.

```
cols <- c("cs_rate_per_area", "cs_rate_per_count", "cs_rate_by_pitch_type_velocity",
          "umpire_out_zone_csr", "umpire_in_zone_csr", "batter_called_strike", "apls",
          "abshorizontalbreak", "INDUCED_VERTICAL_BREAK", "pitcher_called_strike",
          "VERTICAL_APPROACH_ANGLE", "ACTUAL_STRIKE", "CS_OR_NOT")
```

```
cps <- called_pitches[,cols]
```

Choosing a Model

Logistic Regression

While it is usually not as accurate as more powerful predictive techniques, it is fast, versatile, and accurate. For this data, logistic regression was marginally worse in terms of accuracy, but the ability to tune parameters allowed it to be quite competent in terms of specificity (TN/TN+FP), at a cost of precision and accuracy.

```
require(glmnet)
```

```
## Loading required package: glmnet
```

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

```
## Loaded glmnet 4.0-2
```

```
m2 <- glm(CS_OR_NOT ~ ., data = cps, family = "binomial")

summary(m2)
```

```
##
## Call:
## glm(formula = CS_OR_NOT ~ ., family = "binomial", data = cps)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.1108  -0.3327  -0.1648   0.1850   3.1382
##
## Coefficients:
##                                  Estimate Std. Error z value Pr(>|z|)
## (Intercept)                    -1.027e+01  6.153e-01 -16.690  < 2e-16 ***
## cs_rate_per_area                6.409e+00  7.647e-02  83.817  < 2e-16 ***
## cs_rate_per_count               3.280e+00  1.163e-01  28.215  < 2e-16 ***
## cs_rate_by_pitch_type_velocity  1.637e+00  3.038e-01   5.387 7.18e-08 ***
## umpire_out_zone_csr             1.023e+01  9.025e-01  11.337  < 2e-16 ***
## umpire_in_zone_csr              3.543e+00  5.871e-01   6.034 1.60e-09 ***
## batter_called_strike            3.507e+00  2.838e-01  12.356  < 2e-16 ***
## apls                           -1.236e+00  4.340e-02 -28.491  < 2e-16 ***
## abshorizontalbreak             -2.636e-02  8.764e-03  -3.008 0.002627 **
## INDUCED_VERTICAL_BREAK         -1.092e-04  2.908e-05  -3.757 0.000172 ***
## pitcher_called_strike           3.728e+00  3.408e-01  10.939  < 2e-16 ***
## VERTICAL_APPROACH_ANGLE         3.834e-02  1.278e-02   3.000 0.002703 **
## ACTUAL_STRIKEYES                1.962e-01  6.284e-02   3.122 0.001795 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 80161  on 64531  degrees of freedom
## Residual deviance: 29850  on 64519  degrees of freedom
## AIC: 29876
##
## Number of Fisher Scoring iterations: 6
```

The summary of the model shows that all of these predictor variables are significant in predicting whether a pitch will be called or strike. Unsurprisingly, the most important are called strike rate per area, called strike rate per count, and the absolute value of the pitch's final horizontal location. This is shown by them having the largest Z values and lowest p-values.

```
m2.probs <- predict(m2, cps, type = "response")

#maximize accuracy (90.7%), specificity is about 87.7%
pred_acc <- ifelse(m2.probs > 0.415, "YES", "NO")
table(pred_acc, cps$CS_OR_NOT)
```

```
##
## pred_acc    NO   YES
##      NO  42085  3932
##      YES  2280 16235
```

As shown by the confusion matrix, this logistic model is roughly 90.7% accurate and 87.7% specific.

(42085+16235) / 64532 = .907

16235 / (16235+2280) = .877

Most of the following models will be using this type of analysis to show their performance.

Random Forest

Random Forests are a machine learning technique where massive amounts of decision trees consisting of permutations of many predictor variables are created, and the best model wins out. They are generally very powerful, albeit computationally expensive, to run. The model gives a perfect score here because the model is compared to the data it was trained on.

```
require(randomForest)
```

```
## Loading required package: randomForest
## randomForest 4.6-14
## Type rfNews() to see new features/changes/bug fixes.
##
## Attaching package: 'randomForest'
## The following object is masked from 'package:dplyr':
##
##     combine
```

```
rf2 <- randomForest(CS_OR_NOT ~ ., data = cps, importance = T)
pred_rf2 <- predict(rf2, cps, type = "response")

table(pred_rf2, cps$CS_OR_NOT)
```

```
##
## pred_rf2    NO   YES
##      NO  44365     0
##      YES     0 20167
```

Random Forest with Training/Testing Data

I am using 90% of the data to create a model to test on the remaining 10%.

The model is 92% accurate, and has a specificity of 89.6%.

```r
set.seed(9)
tvec <- sample(1:nrow(cps), nrow(cps)/10)
train <- cps[-tvec,]
test <- cps[tvec,]

rf <- randomForest(CS_OR_NOT ~ ., data = train, importance = T)
pred_rf <- predict(rf, test, type = "response")
table(pred_rf, test$CS_OR_NOT)
```

```
##
## pred_rf   NO  YES
##     NO  4254  266
##     YES  203 1730
```

LDA and QDA

Linear and Quadratic Discriminant Analysis try to draw decision boundaries that decide which data regions are balls, and which are strikes. QDA performed better than LDA due to the fact that this data can't really be split by a single line. The decision boundary should surround the plate, which is why QDA performs better.

The LDA model actually had the best specificity of all the models I tested, coming in at 93.7% and an accuracy of 89.2%. The QDA model is 90.6% accurate and 85.8% specific – roughly in line with the other non-ML models.

```r
require(MASS)
```

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

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

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

```r
pitch_lda <- lda(CS_OR_NOT ~ ., data = cps)

pred_lda <- predict(pitch_lda, cps, type = "response")
table(pred_lda$class, cps$CS_OR_NOT)
```

```
##
##          NO   YES
##   NO  43406  5993
##   YES   959 14174
```

```r
pitch_qda <- qda(CS_OR_NOT ~ ., data = cps)

pred_qda <- predict(pitch_qda, cps, type = "response")
table(pred_qda$class, cps$CS_OR_NOT)
```

```
##
##          NO   YES
##   NO  41149  2865
##   YES  3216 17302
```

SVM

SVM - Support Vector Machines, are a machine learning technique that tries to find the optimal hyperplane between the two classifications. It is very complicated, computationally expensive, and difficult to interpret, but it is very good at predictions. In fact, SVM had the highest accuracy out of any of the non-perfect Random Forest models with an accuracy of 93.3% and a specificity of 90.7%.

```
require(e1071)
```

```
## Loading required package: e1071
```

```
pitch_svm20 <- svm(CS_OR_NOT ~ ., data = cps, kernel = "radial", cost = 20)

svm_pred20 <- predict(pitch_svm20, cps, decision.values = F)
table(svm_pred20, cps$CS_OR_NOT)
```

```
##
## svm_pred20    NO   YES
##        NO  42545  2478
##        YES  1820 17689
```

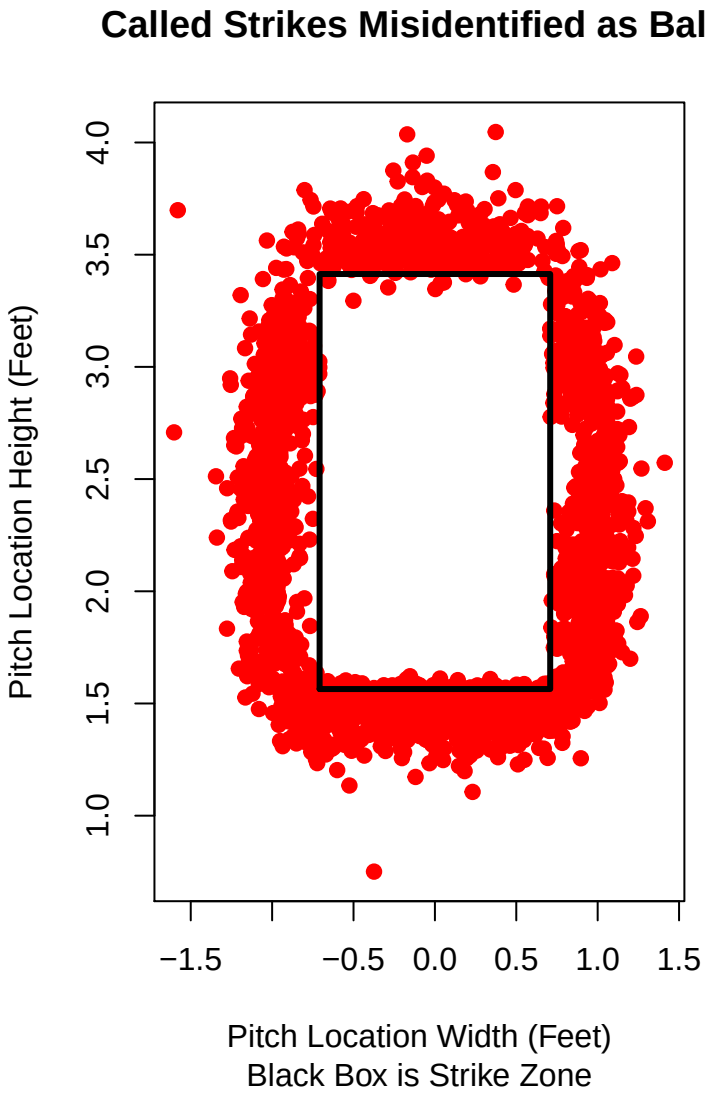
Choosing the Model

After going through all the choices, the SVM model is the best-performing model I made, so I will continue the question using that model's predictions as the basis for my analysis.

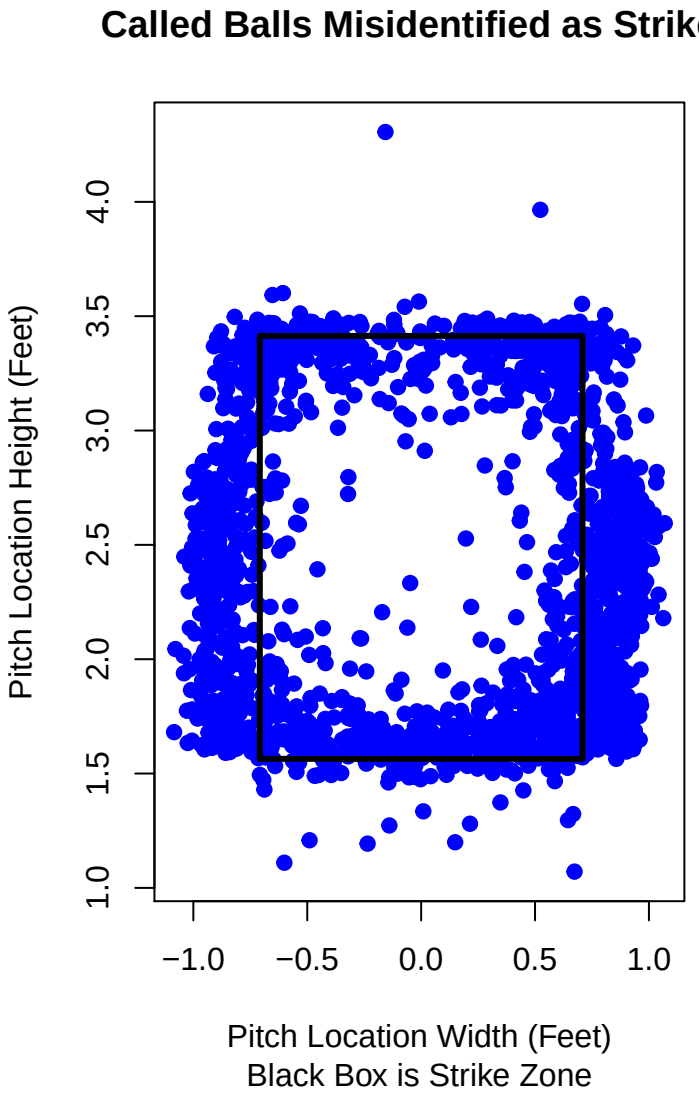
Model Performance Visualization

Shown below are the locations where the model predicted the pitch would be called a ball, but it was actually called a strike (red), and all the locations where the model predicted the pitch would be called a strike, but was actually called a ball (blue).

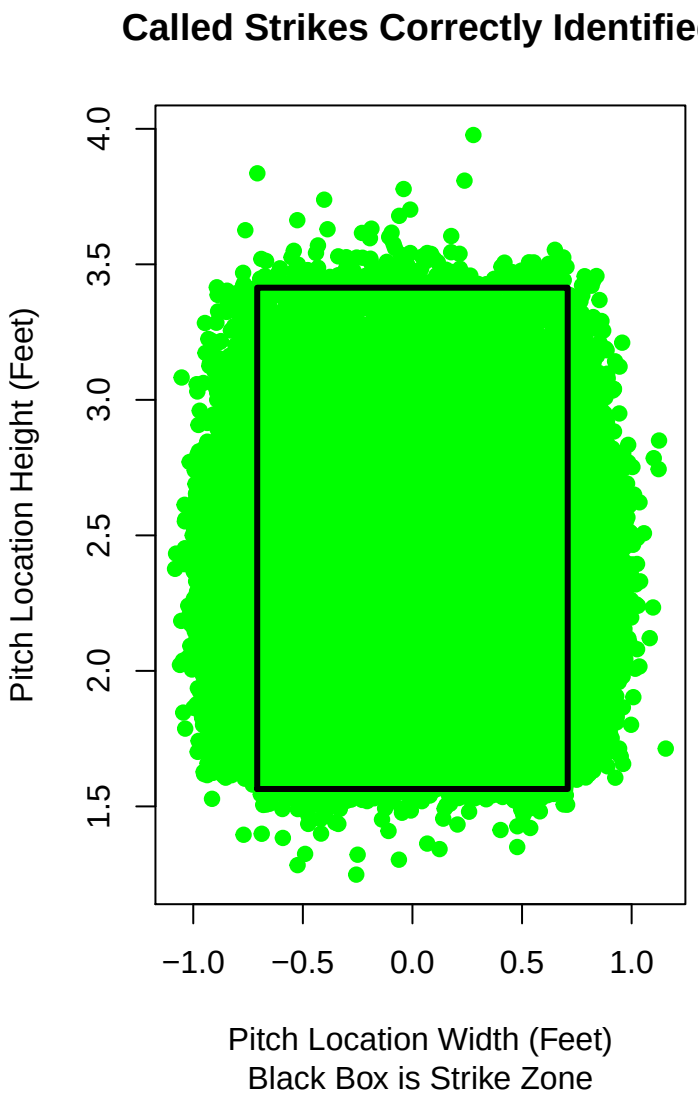
```
predball_callstrike <- called_pitches[called_pitches$CS_OR_NOT == "YES" & svm_pred20 == "NO",]
predstrike_callball <- called_pitches[called_pitches$CS_OR_NOT == "NO" & svm_pred20 == "YES",]
predball_callball <- called_pitches[called_pitches$CS_OR_NOT == "NO" & svm_pred20 == "NO",]
predstrike_callstrike <- called_pitches[called_pitches$CS_OR_NOT == "YES" & svm_pred20 == "YES",]

xx <- c(-.7083, -.7083, .7083, .7083)
yy <- c(1.5643, 3.4138, 3.4138, 1.5643)

plot(predball_callstrike$PITCH_LOCATION_SIDE, predball_callstrike$PITCH_LOCATION_HEIGHT,
     col = "red", pch = 19,
     main = "Called Strikes Misidentified as Balls", xlab = "Pitch Location Width (Feet)",
     ylab = "Pitch Location Height (Feet)", sub = "Black Box is Strike Zone")
polygon(xx, yy, density = 0, col = "black", lwd = 3)
```



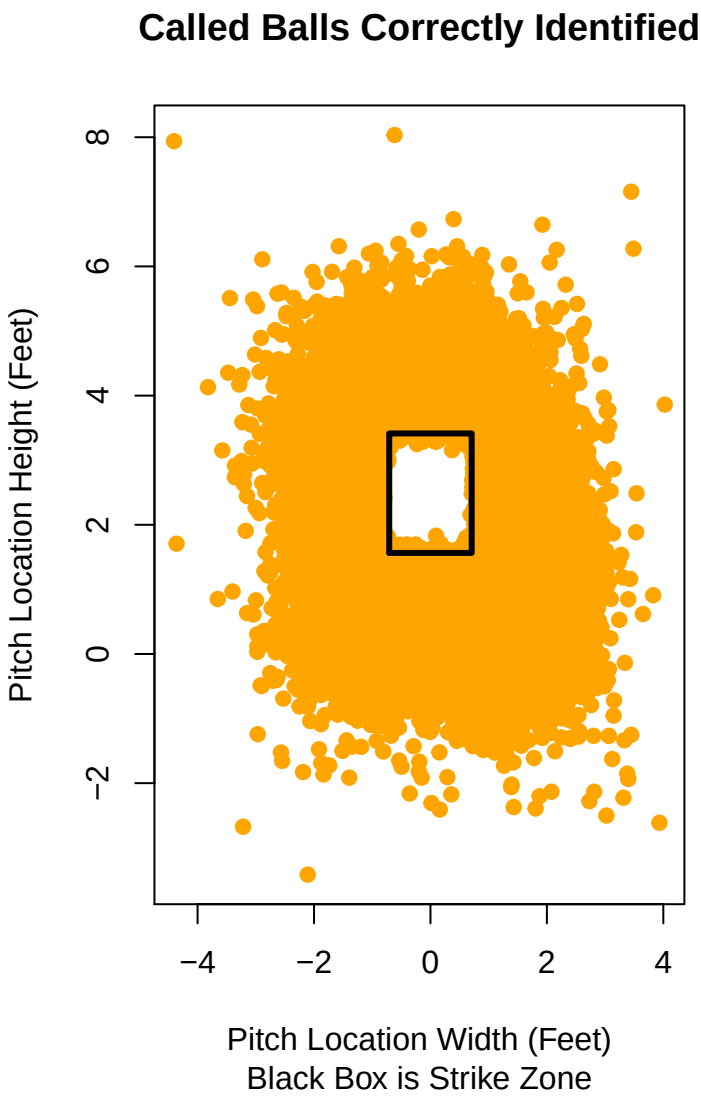

```
plot(predstrike_callball$PITCH_LOCATION_SIDE, predstrike_callball$PITCH_LOCATION_HEIGHT,
     col = "blue", pch = 19,
     main = "Called Balls Misidentified as Strikes", xlab = "Pitch Location Width (Feet)",
     ylab = "Pitch Location Height (Feet)", sub = "Black Box is Strike Zone")
polygon(xx, yy, density = 0, col = "black",  lwd = 3)
```




```
plot(predstrike_callstrike$PITCH_LOCATION_SIDE, predstrike_callstrike$PITCH_LOCATION_HEIGHT,
     col = "green", pch = 19,
     main = "Called Strikes Correctly Identified", xlab = "Pitch Location Width (Feet)",
     ylab = "Pitch Location Height (Feet)", sub = "Black Box is Strike Zone")
polygon(xx, yy, density = 0, col = "black", lwd = 3)
```




```
plot(predball_callball$PITCH_LOCATION_SIDE, predball_callball$PITCH_LOCATION_HEIGHT,
     col = "orange", pch = 19,
     main = "Called Balls Correctly Identified", xlab = "Pitch Location Width (Feet)",
     ylab = "Pitch Location Height (Feet)", sub = "Black Box is Strike Zone")
polygon(xx, yy, density = 0, col = "black",  lwd = 3)
```



The previous visualizations prove that truly predicting every called strike is nearly impossible, because a significant amount of called strikes are actually balls, and a significant amount of called balls are actually strikes.

My model performed admirably, only misidentifying a few actual strikes in the strike zone and misidentifying mostly strikes and borderline calls as balls. Most of the pitches where misidentifications occurs are due to catcher framing, umpire incompetence, or some other factor that I don't have access to.

Which Pitcher Lost the Most Strikes Relative to Expectation?

For this question, I will be finding the pitcher who lost the most strikes, as well as the pitcher who lost the highest percentage of strikes. Expectation will be considered everything the model guessed was a strike but was actually called a ball.

I can simply use the `predstrike_callball` data frame from the plots I just made as a base.

```
most_predicted_strikes_called_balls <- predstrike_callball %>% group_by(PITCHER_ID) %>%
                                       summarise(count = n()) %>% arrange(desc(count))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
most_predicted_strikes_called_balls
```

```
## # A tibble: 443 x 2
##    PITCHER_ID count
##    <fct>      <int>
##  1 266           20
##  2 240           16
##  3 369           16
##  4 53            15
##  5 4             14
##  6 17            13
##  7 43            12
##  8 151           12
##  9 176           12
## 10 194           12
## # ... with 433 more rows
```

```
head(most_predicted_strikes_called_balls)
```

```
## # A tibble: 6 x 2
##   PITCHER_ID count
##   <fct>      <int>
## 1 266           20
## 2 240           16
## 3 369           16
## 4 53            15
## 5 4             14
## 6 17            13
```

The top 5 pitchers who lost strikes in terms of amount are pitchers #266, #240, #369, #53, and #4.

```
ids <- most_predicted_strikes_called_balls$PITCHER_ID
pitcher_apps <- called_pitches[called_pitches$PITCHER_ID %in% ids,] %>% group_by(PITCHER_ID) %>%
  summarise(count_apps = n())
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
combined <- most_predicted_strikes_called_balls %>% full_join(pitcher_apps)
```

```
## Joining, by = "PITCHER_ID"
```

```
lost_strikes_rate <- combined %>% group_by(PITCHER_ID) %>% filter(count_apps > 100) %>%
                      summarise(lost_strikes_rate = count/count_apps) %>%
                      arrange(desc(lost_strikes_rate))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
lost_strikes_rate
```

```
## # A tibble: 242 x 2
##    PITCHER_ID lost_strikes_rate
##    <fct>                  <dbl>
##  1 266                   0.0873
##  2 263                   0.0678
##  3 176                   0.0670
##  4 158                   0.0650
##  5 415                   0.0621
```

```
##  6 369                   0.0608
##  7 33                    0.0569
##  8 319                   0.0566
##  9 240                   0.0557
## 10 423                   0.0556
## # ... with 232 more rows
```

For pitchers that have had at least 100 events in this data set, pitcher #266 was also the worst, losing expected called strikes nearly 9% of the time. I think it is safe to say that pitcher #266 lost the most strikes relative to expectation.