CHAPTER 7

Catcher Framing

7.1 Introduction

In this chapter we explore the idea of *catcher framing* ability. In doing so, we will use the PITCHf/x data introduced in Section 1.5.1 and the Baseball Savant data introduced in Section 1.6.2 and discussed in greather depth in Chapter 12. We will also touch briefly upon the ability to store data in relational databases, which is discussed in greather depth in Chapter 11.

The story of catcher framing ability in sabermetrics is an interesting one. Historically, scouts and coaches insisted that certain catchers had the ability to "frame" pitches for umpires. The idea was that by holding the glove relatively still, you could trick the umpire into calling a pitch a strike even if it was techincally outside of the strike zone (see Lindbergh [2013] for a great visual explanation). Sabermetricians were generally dubious about both the existence and the impact of this skill. Most people who had studied the impact of catcher defense concluded that it was not nearly as valuable as scouts and coaches believed.

Part of the problem was that until the mid-2000s, pitch-level data was hard to come by. With the advent of PITCHf/x, more sophisticated modeling techniques became viable on these more granular data. New studies that estimated the impact of catcher framing substantiated both the existence of a persistent ability (i.e., catchers with good framing numbers stayed good over time) and the magnitude of the effect (i.e., good framers were actually really valuable) [Turkenkopf, 2008, Fast, 2011, Brooks and Pavlidis, 2014, Brooks et al., 2015, Judge, 2018a, Deshpande and Wyner, 2017].

These new findings led to changes in the baseball industry—defensive-minded catchers like José Molina starting getting multi-year contracts that were not justified by their batting skill. Minor league instruction placed greater emphasis on improving framing skills. Of course, as soon as MLB decides to let robots call balls and strikes, then this catcher framing ability will evaporate instantly.

This issue is a nice parable in that it illustrates how sabermetric thinking can (and does) change based on the availability of data and the sophistication

of modeling techniques, as well as how the game on the field can change due to sabermetric insights.

7.2 Acquiring Pitch-Level Data

First, we need to acquire pitch-level data that tells us where each pitch crossed the strike zone. The easiest way to do this is with the `pitchRx` package. In this case, since we want to store a fair amount of data, we are going to use a SQLite database to store the data. The `src_sqlite()` function from `dplyr` creates a new, local, empty SQLite database.

```
library(tidyverse)
db <- src_sqlite("data/pitchrx.sqlite", create = TRUE)
```

Next, we load `pitchRx` and use the `scrape()` function to pull down as much information as we can. In this example, we pull data from the month of May 2016.

```
library(pitchRx)
files <- c("inning/inning_all.xml", "inning/inning_hit.xml",
           "miniscoreboard.xml", "players.xml")
scrape(start = "2016-05-01", end = "2016-05-31",
       connect = db$con, suffix = files)
```

The object `db` maintains a connection to our SQLite relational database. This database now contains several tables, each of which contains different information.

```
db_list_tables(db$con)
```

```
 [1] "action" "atbat"  "coach"  "game"   "hip"    "media"
 [7] "pitch"  "player" "po"     "runner" "umpire"
```

The `pitch` table contains the PITCHf/x information. We can bring that information from SQLite into R using the `tbl()` and `collect()` functions, both of which are provided by `dplyr`.

```
my_pitches <- db %>%
  tbl("pitch") %>%
  collect()
```

The resulting object `my_pitches` is a data frame that contains 128610 observations and 50 variables.

7.3 Where Is the Strike Zone?

In order to understand the impact of catcher framing, we need a way to characterize the probability that any given pitch is called a strike. In the PITCHf/x data, each pitch has a `type`, which is simply `S` for a called strike, `B` for a ball, and `X` if the batter swings. We plot these outcomes in Figure 7.1. Note that pitches thrown in the strike zone are both more likely to be called a strike and more likely to be swung at. Note also that many pitches are called strikes even though they are technically outside of the strike zone.

```
plate_width <- 17 + 2 * (9/pi)
k_zone_plot <- ggplot(NULL, aes(x = px, y = pz)) +
  geom_rect(xmin = -(plate_width/2)/12,
            xmax = (plate_width/2)/12,
            ymin = 1.5,
            ymax = 3.6, color = crcblue, alpha = 0) +
  coord_equal() +
  scale_x_continuous("Horizontal location (ft.)",
                     limits = c(-2, 2)) +
  scale_y_continuous("Vertical location (ft.)",
                     limits = c(0, 5))
```

How do we know where the strike zone is? By the rulebook, only a part of the ball need pass over home plate in order for the pitch to be called a strike. Home plate is 17 inches wide, and the ball is 9 inches in circumference, so the outside edges of the strike zone from our point-of-view are about $\pm$ 0.947 feet. The top and bottom of the strike vary by batter, but are of comparatively less interest here. The object `k_zone_plot` is a blank `ggplot2` object on which we plot a random sample of the data from PITCHf/x in Figure 7.1.

```
k_zone_plot %+% sample_n(my_pitches, 10000) +
  aes(color = type) +
  geom_point(alpha = 0.1) +
  scale_color_manual(values = c(crcblue, "white", "black"))
```

Another way to think about the strike zone is in terms of zones that are pre-defined by PITCHf/x. The strike zone itself is divided into a 3×3 grid, with four additional regions defined outside of the strike zone. We first compute the observed probability of a called strike in each one of those zones, as well as its boundaries. We use the `quantile()` function to mitigate the influence of outliers.

```
taken <- my_pitches %>%
  filter(type != "X")

zones <- taken %>%
```

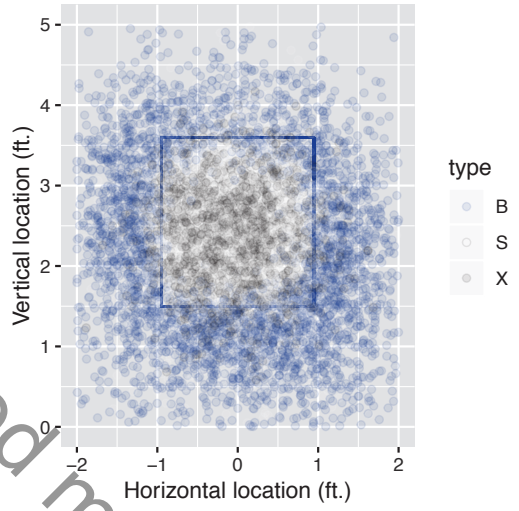

FIGURE 7.1 Scatterplot of balls, called strikes, and swings, May 2016.

```
  group_by(zone) %>%
  summarize(
    N = n(),
    right_edge = min(1.5, max(px)),
    left_edge = max(-1.5, min(px)),
    top_edge = min(5, quantile(pz, 0.95, na.rm = TRUE)),
    bottom_edge = max(0, quantile(pz, 0.05, na.rm = TRUE)),
    strike_pct = sum(type == "S") / n(),
    px = mean(px),
    pz = mean(pz))
```

In Figure 7.2 we plot each zone, along with the probability that a pitch taken in that zone will be called a strike. Note that these pre-defined zones are exclusive of those pitches "on the black".

```
library(ggrepel)
k_zone_plot %+% zones +
  geom_rect(aes(xmax = right_edge, xmin = left_edge,
                ymax = top_edge, ymin = bottom_edge,
                fill = strike_pct, alpha = strike_pct),
            color = "lightgray") +
  geom_text_repel(size = 3, aes(label = round(strike_pct, 2),
                            color = strike_pct < 0.5)) +
  scale_fill_gradient(low = "gray70", high = crcblue) +
  scale_color_manual(values = c("white", "black")) +
```

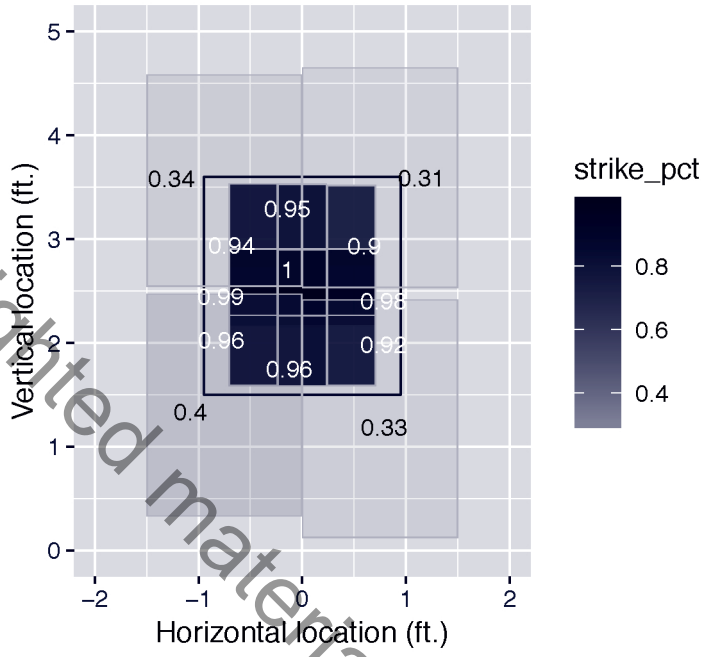

FIGURE 7.2 Strike probability for pitches taken in pre-defined areas of the strike zone.

```
  guides(color = FALSE, alpha = FALSE)
```

7.4 Modeling Called Strike Percentage

The zone-based strike probabilities in Figure 7.2 are limited by their discrete nature. What we really want is a model that will give us the estimated strike probability for any pitch based on its horizontal and vertical location. To this end, we fit a generalized additive model. This model will fit a smooth surface over the entire area, while including only the two explanatory variables for location. The `s()` function indicates over which variables the smoothing is to occur (`px` and `pz`). We set the `family` argument to `binomial`, to ensure that an appropriate link function (in this case, the logistic function) is used to model our binary response variable, which is defined by the Boolean expression `type == "S"`.

```
library(mgcv)
strike_mod <- gam(type == "S" ~ s(px, pz),
                  family = binomial, data = taken)
```

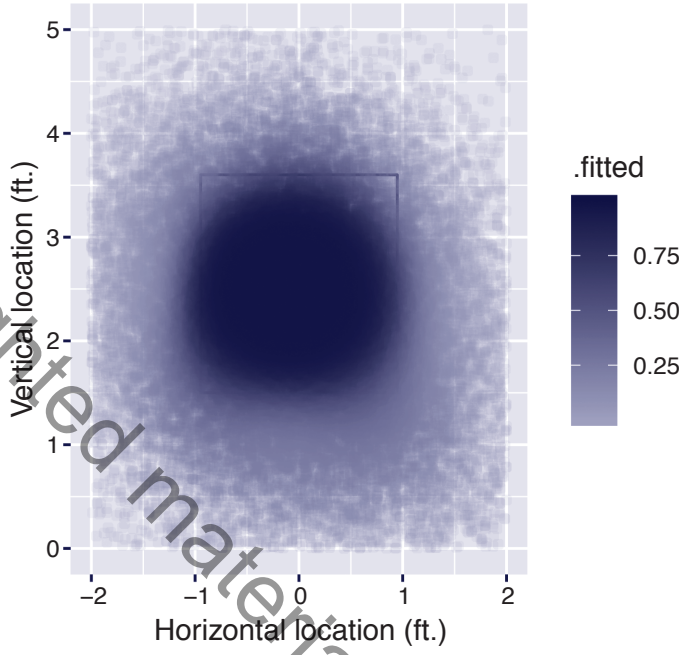

FIGURE 7.3 Estimated strike probability for taken pitches using a generalized additive model.

7.4.1 Visualizing the estimates

An easy way to visualize the estimates produced by our model is to plot the fitted values. Here we use the `augment()` function from the `broom` package to compute these fitted values and add them to our data frame. The `type.predict` argument tells R to compute the estimates on the probability scale (i.e., of the response variable).

```
library(broom)
hats <- strike_mod %>%
  augment(type.predict = "response")
```

Next, we can simply update our `k_zone_plot` object with this new data frame, add some points (`geom_point()`), and map the color aesthetic to the fitted values we just computed (`.fitted`). Figure 7.3 reveals that on these data, the GAM effectively mapped the pattern of balls and strikes.

```
k_zone_plot %+% sample_n(hats, 50000) +
  geom_point(aes(color = .fitted), alpha = 0.1) +
  scale_color_gradient(low = "gray70", high = crcblue)
```

7.4.2 Visualizing the estimated surface

Of course the GAM that we built is a continuous surface. One of the benefits of fitting such a model in the first place is that it allows us to estimate the probability of a called strike for *any* pitch whose location coordinates we know—not just the ones present in our training data set.

We can visualize our model as a surface by plotting the estimated probability across a fine grid of horizontal and vertical coordinate pairs. The `modelr` package has several functions, including `data_grid()` and `seq_range()` that help us create a grid of values relevant for our data.

```
library(modelr)
grid <- taken %>%
  data_grid(px = seq_range(px, n = 100),
            pz = seq_range(pz, n = 100))
```

Next, use the `augment()` function just as before, except this time, we specify the `newdata` argument to be the data frame of grid points that we just created. This results in a 10000 row data frame that contains the estimated called strike probability for each coordinate pair.

```
grid_hats <- strike_mod %>%
  augment(type.predict = "response", newdata = grid)
```

Once again, we update our `k_zone_plot` with these new data. The `geom_tile()` function in Figure 7.4 offers a nice alternative to `geom_contour()`.

```
tile_plot <- k_zone_plot %+% grid_hats +
  geom_tile(aes(fill = .fitted), alpha = 0.7) +
  scale_fill_gradient(low = "gray92", high = crcblue)
tile_plot
```

7.4.3 Controlling for handedness

Contrary to what the rulebook states, it stands to reason that the effective strike zone may depend on with which hand the pitcher throws, and on which side of the plate the batter stands. Unfortunately, these data are not present in the `pitch` table. Rather, these data are present in the `atbat` table. To get that information aligned with our pitch-level data, we use an `inner_join()` between the two tables, using the variables `num` and `gameday_link` as matching keys.

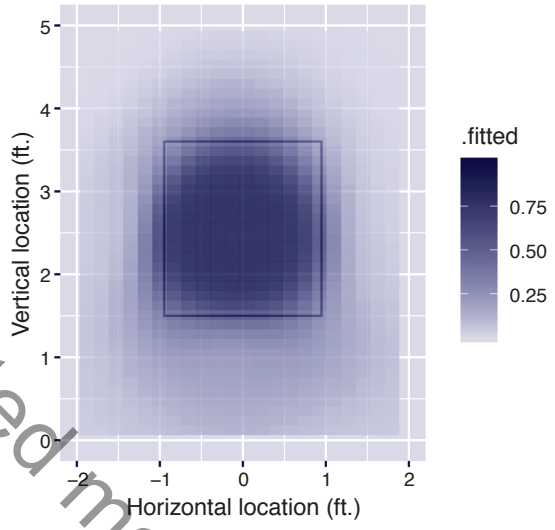

FIGURE 7.4 Estimated strike probability over a grid for taken pitches using a generalized additive model.

```
more_taken <- db %>%
  tbl("pitch") %>%
  filter(type != "X") %>%
  inner_join(tbl(db, "atbat"),
             by = c("num", "gameday_link")) %>%
  collect()
```

The resulting data frame has variables for `p_throws` and `stand` in addition to the location data encoded in `px` and `pz`. We can now fit another GAM across these four variables. Note that the binary variables `p_throws` and `stand` are not smoothed, and are thus outside of the `s()` function in the model specification formula.

```
hand_mod <- gam(type == "S" ~ p_throws + stand + s(px, pz),
                family = binomial, data = more_taken)
```

We must now recompute our grid of values such that they include the two additional binary variables.

```
hand_grid <- more_taken %>%
  data_grid(px = seq_range(px, n = 100),
            pz = seq_range(pz, n = 100),
            p_throws, stand)
```

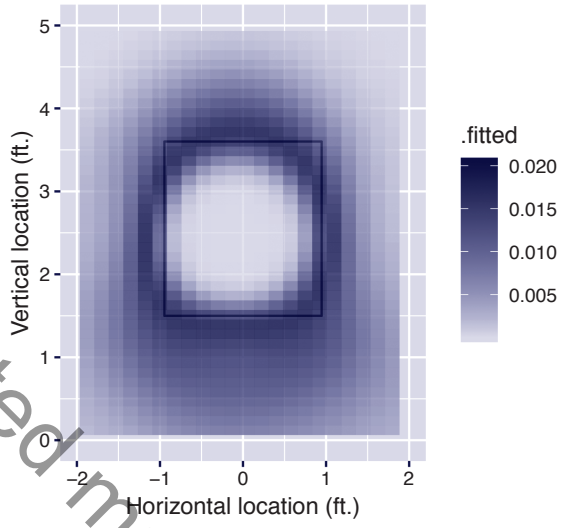

FIGURE 7.5 Standard deviation of estimated called strike probability across all four pitcher-batter handedness combinations.

```
hand_grid_hats <- hand_mod %>%
  augment(type.predict = "response", newdata = hand_grid)
```

The following code will produce a faceted plot across the four combinations of batter and pitcher handedness. However, as it is difficult to perceive marked difference across these four facets, we omit the plot here.

```
tile_plot %+% hand_grid_hats +
  facet_grid(p_throws ~ stand)
```

Instead, we plot the standard deviation across the four handedness combinations in Figure 7.5. In the heart of the strike zone, we see no differences due to handedness. However, the standard deviation of called strike probability is as large as 2 percentage points in some area around the perimeter of the strike zone.

```
diffs <- hand_grid_hats %>%
  group_by(px, pz) %>%
  summarize(N = n(), .fitted = sd(.fitted))
tile_plot %+% diffs
```

7.5 Modeling Catcher Framing

In order to estimate the framing ability of catchers, we need to know who the catcher is during every pitch. Unfortunately, that information is difficult to extract from the data we get from `pitchRx`. Instead, we use the Statcast summary data provided by Baseball Savant and accessible through the `baseballr` package (see Chapter 12).

Previously, we downloaded these data for the 2017 season. We now load those data from a CSV file.

```
sc_2017 <- read_csv("data/statcast2017.csv")
```

Preparing these data for modeling will take some work. First, we are only interested in those pitches where there was no swing. Second, because we want to use the called strike GAM we built earlier (`strike_mod`), we need to rename the location variables from their Statcast names (`plate_x`, `plate_z`) to their PITCHf/x equivalents (`px`, `pz`). Third, we will evaluate our GAM for called strike probability on each pitch. This helps us control for the location of each pitch. It is relatively slow to fit the mixed model of this section on the full dataset, so we use the `sample_n` function to take a sample of 10,000 pitches for this exercise.

```
set.seed(111653)
sc_taken <- sc_2017 %>%
  filter(type != "X") %>%
  rename(px = plate_x, pz = plate_z) %>%
  sample_n(10000) %>%
  mutate(strike_prob = predict(strike_mod, newdata = .,
                               type = "response"))
```

Next, we follow Brooks et al. [2015] in fitting a generalized linear mixed model. The response variable is whether the pitch was called a strike or a ball. Let p denote the probability that a called pitch is a strike. Our first mixed model writes the logit of the probability of a strike p as the sum

$$\log \frac{p}{1-p} = \beta_0 + \beta_1 strike\_prob + catcher.$$

In this model, `strike_prob` is the "fixed effect" for the estimated called strike probability based on its location computed from the previous model. So we are essentially controlling for the pitch location in this model. In addition, it is assumed that the individual catchers have "random" parameters, called $catcher_1, \ldots, catcher_C$, with mean 0 and standard deviation of s_c.

This model can be fit using the `glmer()` function in the `lme4` package. The code indicates the response variable is `type == "S"`, `strike_prob` is the fixed effect and `pos2_person_id` (the catcher id) represents the random effect.

```
library(lme4)
mod_a <- glmer(type == "S" ~ strike_prob + (1|pos2_person_id),
               data = sc_taken, family = binomial)
```

We recover information about the fixed effects using the `tidy()` function from the `broom` package.

```
tidy(mod_a, effects = "fixed")

# A tibble: 2 x 5
  term        estimate std.error statistic p.value
  <chr>          <dbl>     <dbl>     <dbl>   <dbl>
1 (Intercept)    -2.82    0.0576     -49.0       0
2 strike_prob     5.96    0.106       56.3       0
```

Certainly different catchers will have different impacts on the probability of a called strike. The variability in these impacts is measured by the standard deviation of these random catcher effects s_c that we display by the `tidy()` function by setting the `effects` paramter to `ran_pars`.

```
tidy(mod_a, effects = "ran_pars")

# A tibble: 1 x 3
  term                          group         estimate
  <chr>                         <chr>            <dbl>
1 sd_(Intercept).pos2_person_id pos2_person_id   0.0437
```

This model also provides estimates of the catcher random effects $\{catcher_j\}$ that one extracts with the `ranef()` function. We put the estimates together with the catcher ids in the data frame `c_effects`.

```
c_effects <- mod_a %>%
  ranef() %>%
  as_tibble() %>%
  transmute(id = as.numeric(levels(grp)),
            effect = condval)
```

The names of the catchers are missing, but we include a separate file `masterid.csv` that provides a table for these ids and names. We merge the name information with the data frame `c_effects` and display the names of the catchers with the largest and smallest random effect estimates below.

```
master_id <- read_csv("data/masterid.csv")
c_effects <- c_effects %>%
  left_join(select(master_id, mlb_id, mlb_name),
```

```
            by = c("id" = "mlb_id")) %>%
  arrange(desc(effect))

c_effects %>% head()

# A tibble: 6 x 3
     id effect mlb_name
  <dbl>  <dbl> <chr>
1 592663 0.0243 J.T. Realmuto
2 543877 0.0152 Christian Vazquez
3 553869 0.0148 Elias Diaz
4 471083 0.0112 Miguel Montero
5 543432 0.0111 Ryan Lavarnway
6 455139 0.0103 Robinson Chirinos

c_effects %>% tail()

# A tibble: 6 x 3
     id  effect mlb_name
  <dbl>   <dbl> <chr>
1 465041 -0.0108 Francisco Cervelli
2 488912 -0.0113 Tuffy Gosewisch
3 502570 -0.0125 Rocky Gale
4 596119 -0.0132 Blake Swihart
5 572033 -0.0152 Josh Phegley
6 457454 -0.0180 Jarrod Saltalamacchia
```

From this output, we see that J.T. Realmuto was most effective in getting a called strike and Jarrod Saltalamacchia was least effective.

One criticism of this first model is that no allowances were made for the pitcher or batter, and it is believed that both people have an impact on the probability of a called strike. We can extend the above model to include random effects for both the pitcher and the batter. We write this model as

$$\log \frac{p}{1-p} = \beta_0 + \beta_1 strike\_prob + catcher + pitcher + batter$$

Here the individual pitchers are assigned parameters $p_1, \ldots, p_P$ that are assumed to be random from a distribution with standard deviation s_p. In addition, the individual batters are assigned parameters $b_1, \ldots, b_B$ that come from a distribution with standard deviation s_b.

This larger model is fit with a second application of the `glmer()` function, adding `batter` and `pitcher` as inputs in the regression expression.

```
mod_b <- glmer(type == "S" ~ strike_prob + (1|pos2_person_id) +
                 (1|batter) + (1|pitcher),
               data = sc_taken, family = binomial)
```

Using the `effects` argument in the `tidy()` function, we display estimates of the three standard deviations s_c, s_p, and s_b. Note that the value of s_c is slightly different than it was in the previous model.

```
tidy(mod_b, effects = "ran_pars")

# A tibble: 3 x 3
  term                           group          estimate
  <chr>                          <chr>             <dbl>
1 sd_(Intercept).batter          batter           0.266
2 sd_(Intercept).pitcher         pitcher          0.187
3 sd_(Intercept).pos2_person_id pos2_person_id   0.0319
```

This table is helpful in identifying the components that contribute most to the total variability in called strikes. The largest standard deviation is s_b =0.266 which indicates that called strikes are most influenced by the identity of the batter, followed by the identity of the pitcher, and last by the identity of the catcher.

As before, we extract the catcher effect estimates by the `ranef()` function, create a data frame of ids, names, and estimates for all catchers, and then display the best and worst catchers with respect to framing. These lists are not similar to the lists prepared with the simpler random effects model, suggesting these catchers worked with different pitchers and batters who impacted the called strikes.

```
c_effects <- mod_b %>%
  ranef() %>%
  as_tibble() %>%
  filter(grpvar == "pos2_person_id") %>%
  transmute(id = as.numeric(as.character(grp)),
            effect = condval)
c_effects <- c_effects %>%
  left_join(select(master_id, mlb_id, mlb_name),
            by = c("id" = "mlb_id")) %>%
  arrange(desc(effect))

c_effects %>% head()

# A tibble: 6 x 3
      id  effect mlb_name
   <dbl>   <dbl> <chr>
1 425784 0.0133  Rene Rivera
2 452095 0.00789 Tyler Flowers
3 431145 0.00758 Russell Martin
4 455117 0.00542 Martin Maldonado
```

```
5 425772 0.00527 Jeff Mathis
6 519222 0.00520 Austin Romine

c_effects %>% tail()

# A tibble: 6 x 3
     id   effect mlb_name
  <dbl>    <dbl> <chr>
1 519237 -0.00549 Cameron Rupp
2 446308 -0.00614 Matt Wieters
3 543877 -0.00658 Christian Vazquez
4 518960 -0.00658 Jonathan Lucroy
5 592663 -0.00765 J.T. Realmuto
6 547172 -0.00925 Tony Wolters
```

This is clearly not a thorough analysis since we only used a small dataset and did not include other effects such as umpires that could impact the called strike probability. But these mixed models with inclusion of fixed and random effects are very useful for obtaining estimates of player abilities making adjustments for other relevant inputs.

7.6 Further Reading

The first study of catcher framing using PITCHf/x was Turkenkopf [2008]. See Fast [2011] for a follow-up piece. Lindbergh [2013] provides a highly-readable lay overview of the evolution of thinking on catcher framing. More sophisticated models for catcher framing include Brooks and Pavlidis [2014], Brooks et al. [2015], Judge [2018a], Deshpande and Wyner [2017].

7.7 Exercises

1. **Strike Probabilities on a Grid**

   (a) Divide the zone region into bins by use of the following script.

```
seq_x <- seq(-1.4, 1.4, by = 0.4)
seq_z <- seq(1.1, 3.9, by = 0.4)
taken %>%
    mutate(px = cut(plate_x, seq_x),
           pz = cut(plate_z, seq_z)) -> taken
```

   (b) By use of the `group_by()` and `summarize()` functions, find the number of strikes and balls among called pitches in each bin.

   (c) Find the percentage of strikes in each bin. Comment on any interesting patterns in these strike percentages across bins.

2. **Strike Probability Batter Effects**

   In the first exercise, the strike probability percentages were found for different zones. By tabulating the balls and strikes across bins and for the variable `stand`, explore how the strike probabilities vary by the side of the batter.

3. **Strike Probability Pitcher Effects**

   In the first exercise, the strike probability percentages were found for different zones. By tabulating the balls and strikes across bins and for the variable `p_throws`, explore how the strike probabilities vary by the throwing arm of the pitcher.

4. **Count Effects**

   One way to explore the effect of the count on a strike probability is to fit the logistic model using the `glm()` function:

```
fit <- glm(type == "S" ~ Count,
           data = taken, family = binomial)
```

   In this expression, `Count` is a new variable derived from the `balls` and `strikes` variables in the `taken` data frame. From the output of this fit, interpret how the strike probability depends on the count.

5. **Home/Away Effects**

   One way to explore the effect of home field on a strike probability is to fit the logistic model using the `glm()` function:

```
fit <- glm(type == "S" ~ Home,
           data = taken, family = binomial)
```

   In this expression, `Home` is a new variable that is equal to one if the batter is from the home team, and equal to zero otherwise. From the output of this fit, interpret how the strike varies among home and away batters.