Syracuse University

MEASURING THE IMPACT OF INCORRECT BALL AND STRIKE CALLS

By Matthew Penn, Tyler Bolebruch, Jayke Pastis, Nicholas Kamimoto, and Brett Gustin

Table of Contents

Introduction

- Pitch Tracking allows fans to see the exact path of the ball from the pitcher's hand to the catcher's mitt
- Umpires can be judged on ball and strike calls in real time
- Each missed call has some effect on the remainder of the game, with some potentially changing the outcome

Methodology

- Find effect of each missed call by comparing expected outcomes for the rest of the half inning in either count
- Find total effect of all missed calls in the game while each team was batting
- Add or subtract that effect from the team's final score to get an estimate of what their score should have been
- Compare corrected final score to actual final score

Weighted Run Expectancy (wRE)

Run Expectancy- An estimate on the amount of runs an average team is likely to score in a given base-out situation and count. Calculated the same way for any team at any spot in the batting order.

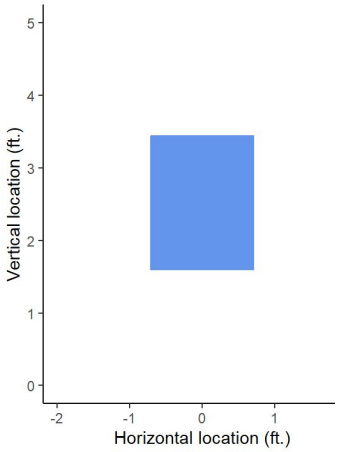
weighted Run Expectancy (wRE)- Our new metric measures expected runs from a given situation in the game based on simulating from that point to the end of the half inning, taking into account the batting order and pitcher.

wRE Difference Calculation

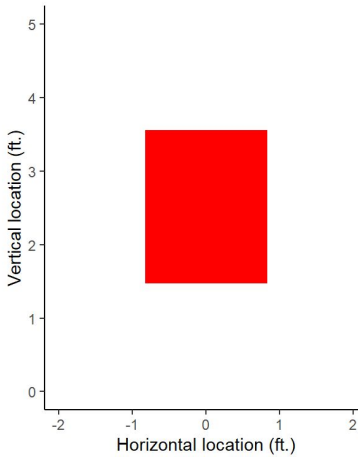
- Gathered data for pitcher and hitter tendencies in and out of the strike zone by count
- Formed probabilities of plate appearance and in play results based off these tendencies
- Using this data, we simulated half innings based on the current lineup in the game
- By running simulations, our new metric wRE is created to determine the influence of a missed call on runs scored

Defining a “Missed Call”

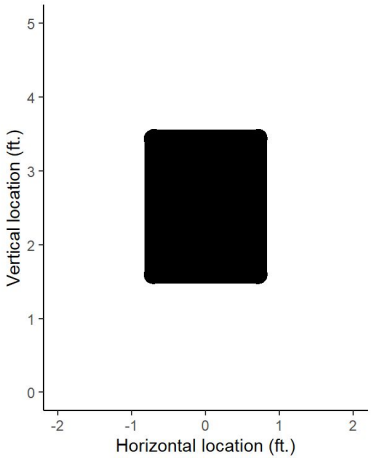
Rule Book Definition



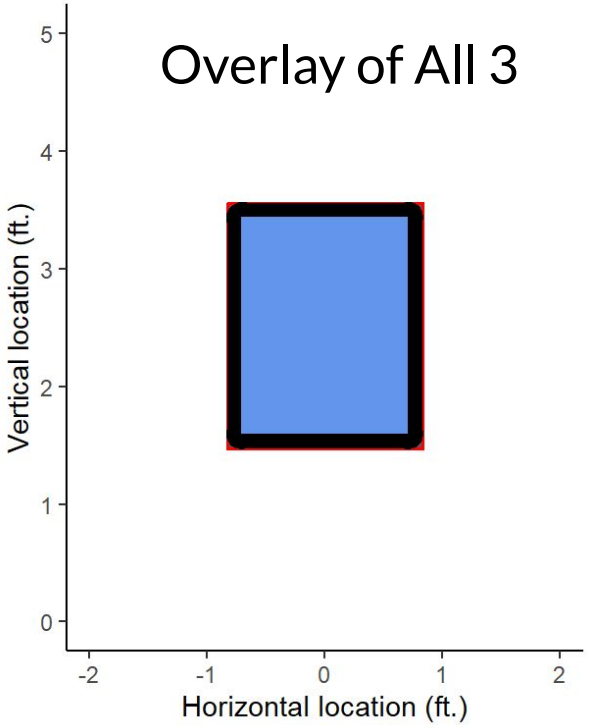
Expanded by Radius of Baseball on all 4 sides



Accounting for Over-Correction



Overlay of All 3



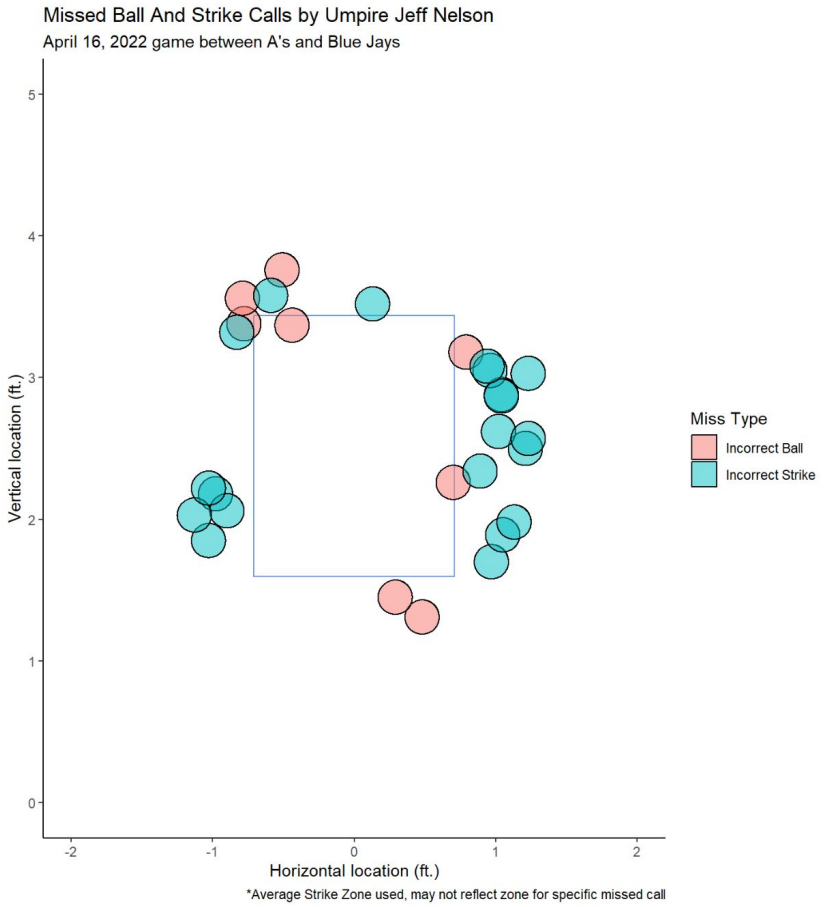
Oakland vs Toronto

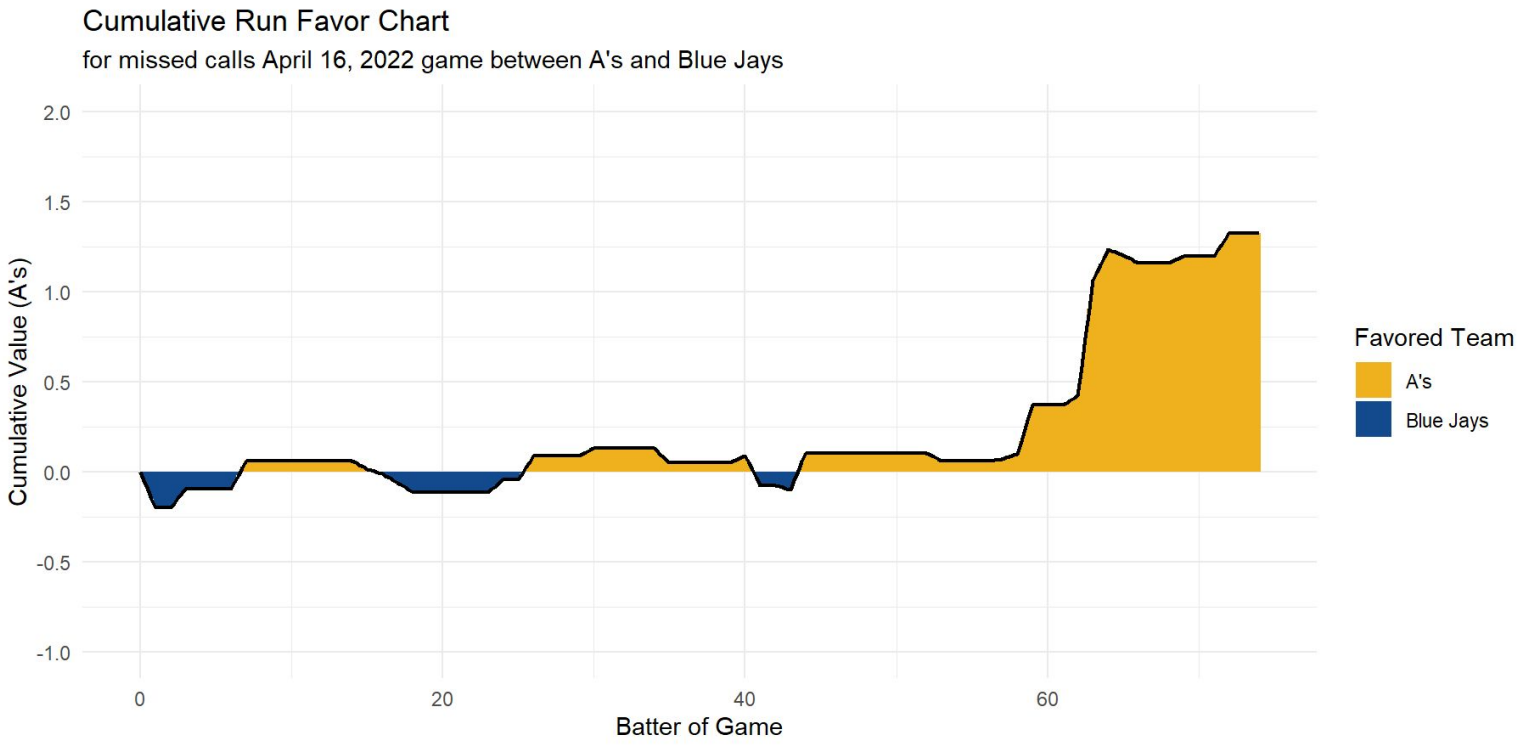
Game Summary

- April 16, 2022 (Jeff Nelson)
- Paul Blackburn vs Hyun Jin Ryu
- A's score 2 in top of the 9th inning to take lead and win
- Final Score 7-5 A's
- 28 total missed calls:
  - 16 with Blue Jays batting
  - 12 with A's batting

Overview

- Viewed each call individually with our created zone dimensions to determine the missed call
- Zone dimensions vary based off each hitter's height for that pitch
- 28 missed pitches
  - 8 incorrect ball calls
  - 20 incorrect strike calls





Total Favor: +1.325 runs for the A's

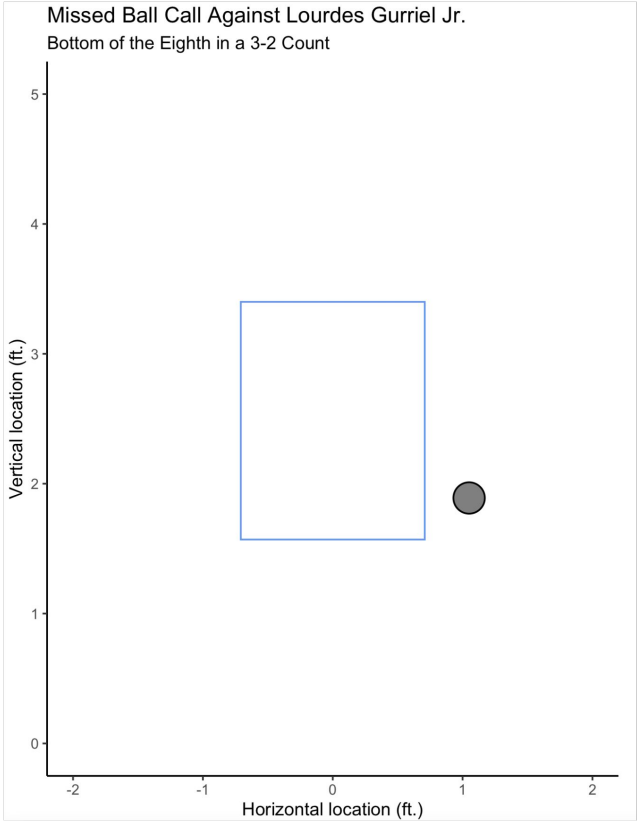
Most Impactful Calls

- Lourdes Gurriel Jr.
  - -0.630 runs
  - Bottom of the 8th
- Elvis Andrus
  - +0.270 runs
  - Top of the 8th
- Bo Bichette
  - -0.205 runs
  - Bottom of the 6th

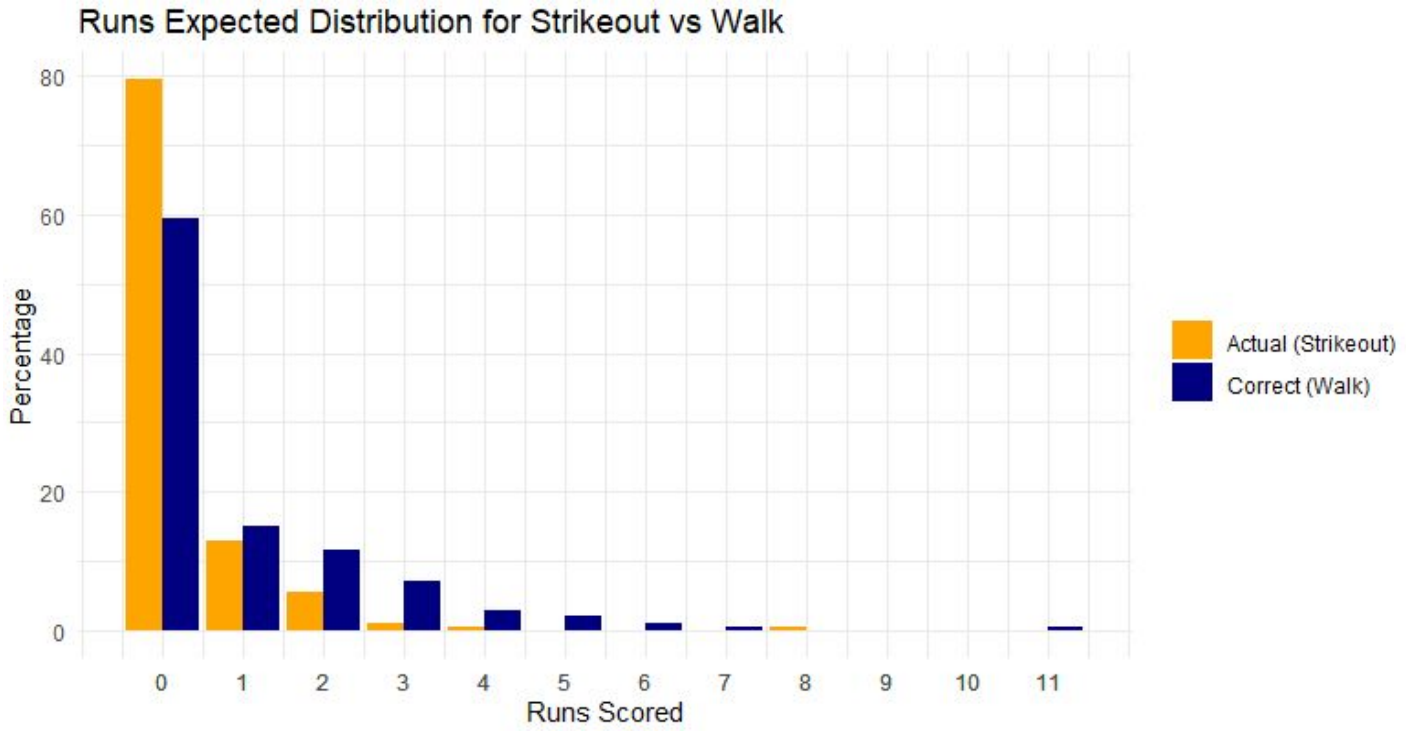
Impactful Call 1

- Inning: Bottom 8 Outs: 0
  Pitcher: Dany Jimenez
  Batter: Lourdes Gurriel Jr.
  - Actual
    - 3-3 (strikeout)
    - 0.330 wRE
  - Correct
    - 4-2 (walk)
    - 0.960 wRE

- Total effect: -0.630 runs



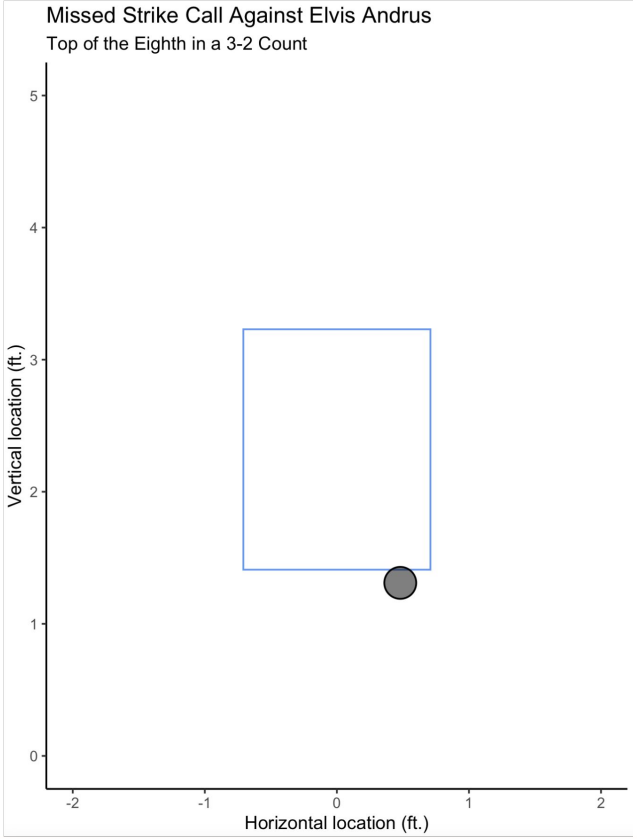
Impactful Call 1



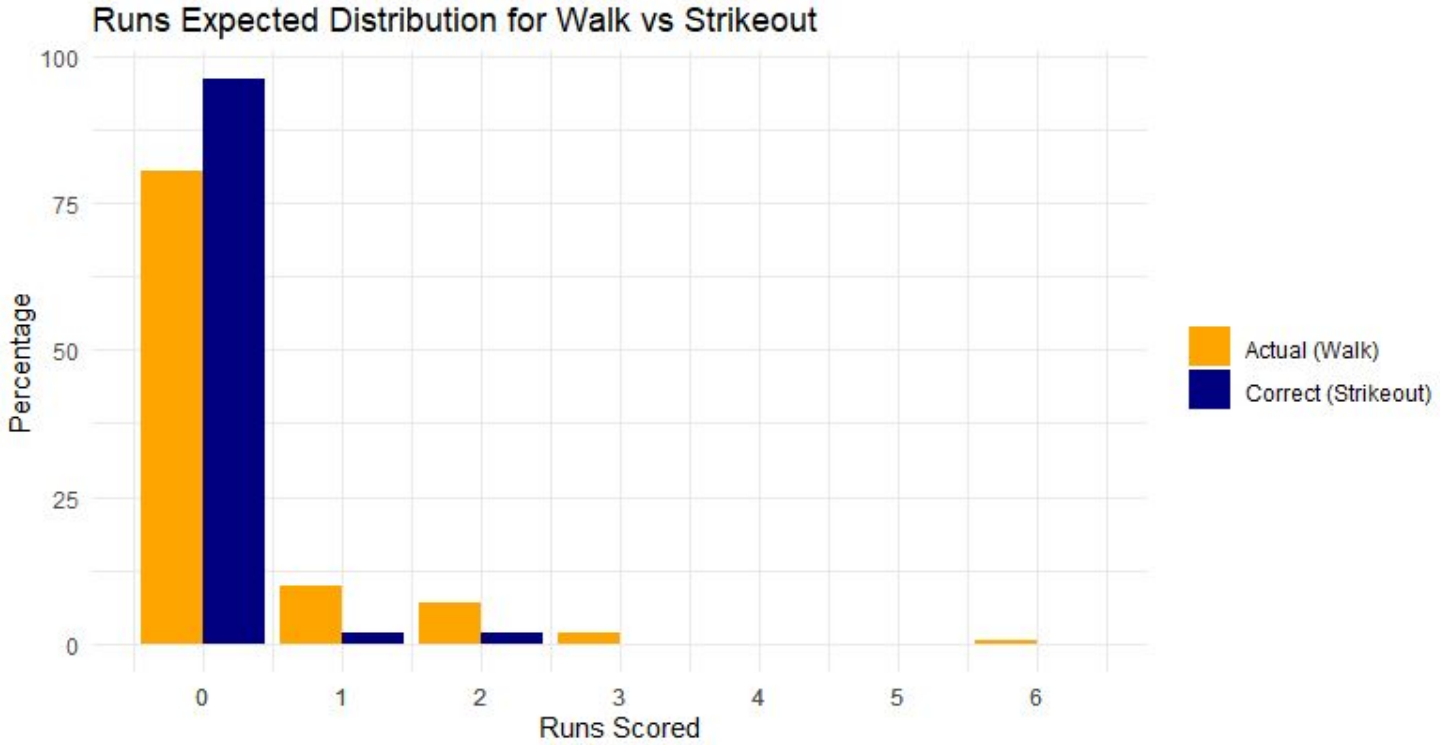
Impactful Call 2

- Inning: Top 8 Outs: 1
  Pitcher: Yimi Garcia
  Batter: Elvis Andrus
  - Actual
    - 4-2 (walk)
    - 0.330 wRE
  - Correct
    - 3-3 (strikeout)
    - 0.060 wRE

- Total effect: +0.270 runs



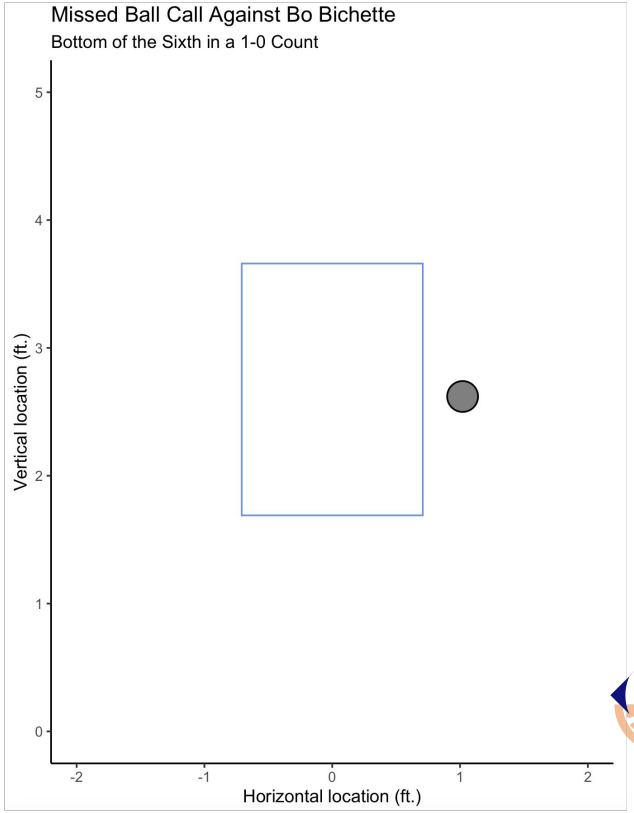
Impactful Call 2



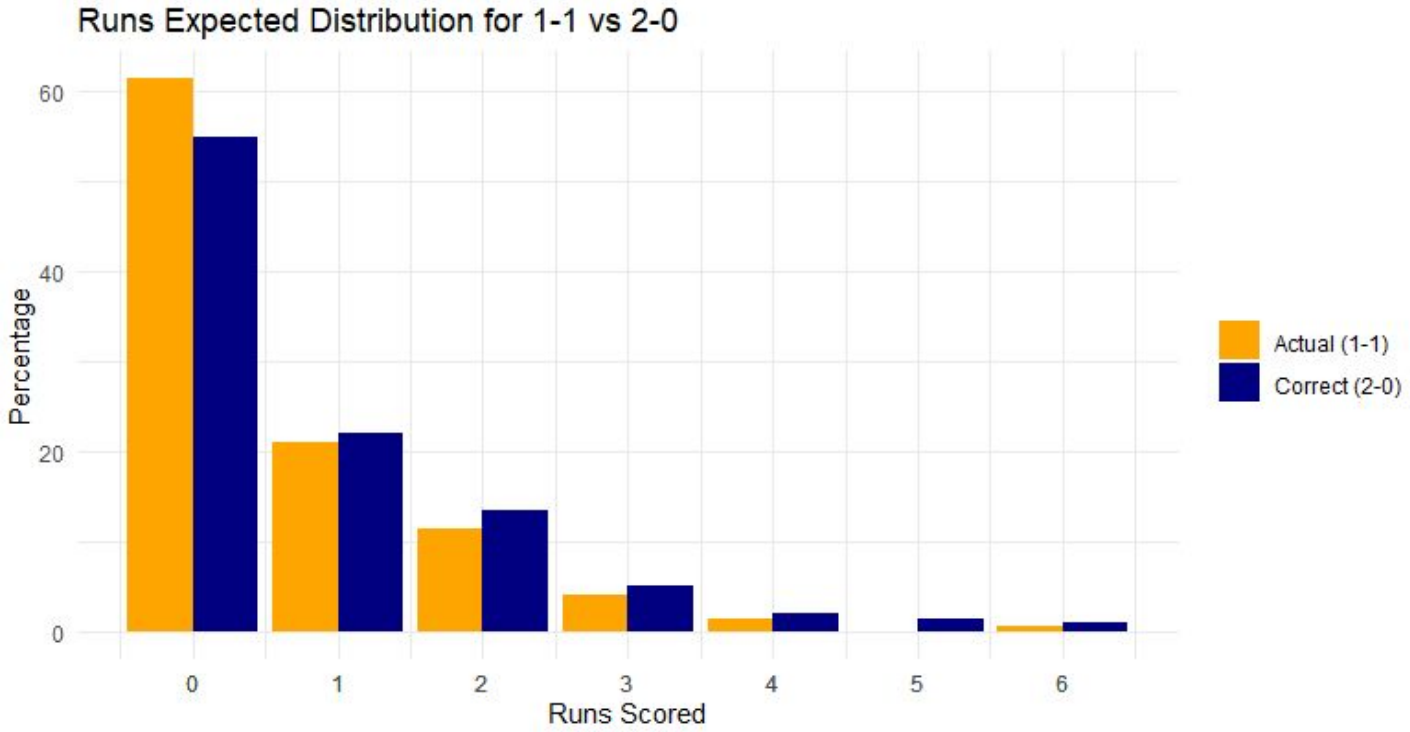
Impactful Call 3

- Inning: Bottom 6th Outs: 0
  Pitcher: Domingo Acevedo
  Batter: Bo Bichette
  - Actual
    - 1-1
    - 0.650 wRE
  - Correct
    - 2-0
    - 0.855 wRE

- Total effect: -0.205 runs



Impactful Call 3



Corrected Final Score

- Original
  - 7 - 5 A's win
- Adjustment
  - A's favored by -0.060
  - Blue Jays favored by -1.385
- Corrected Final Score
  - 7.06 - 6.39 A's win

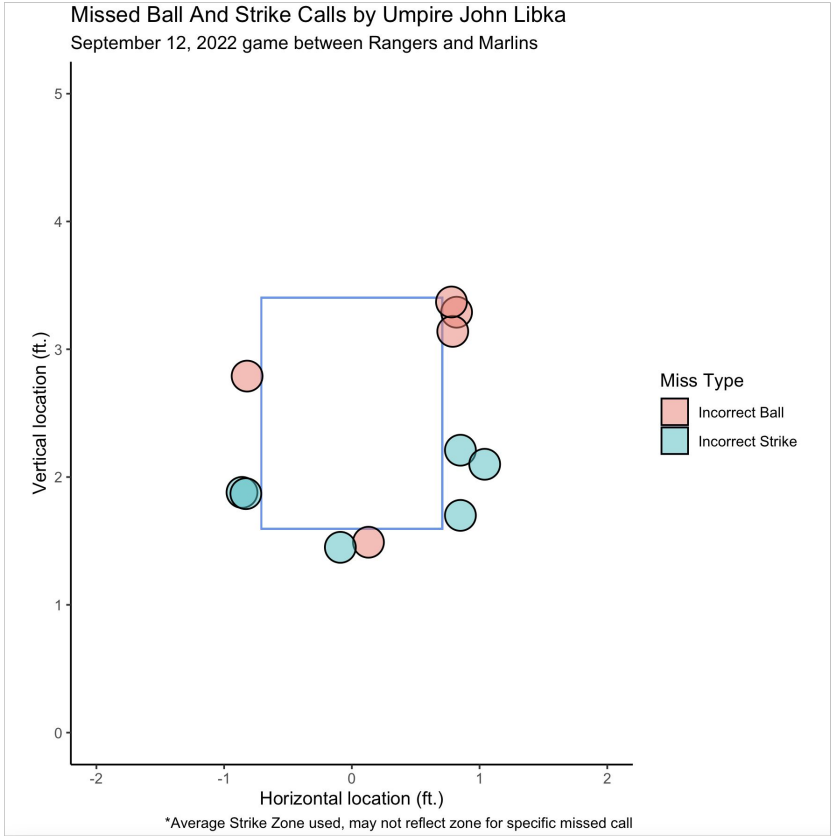
Texas vs Miami

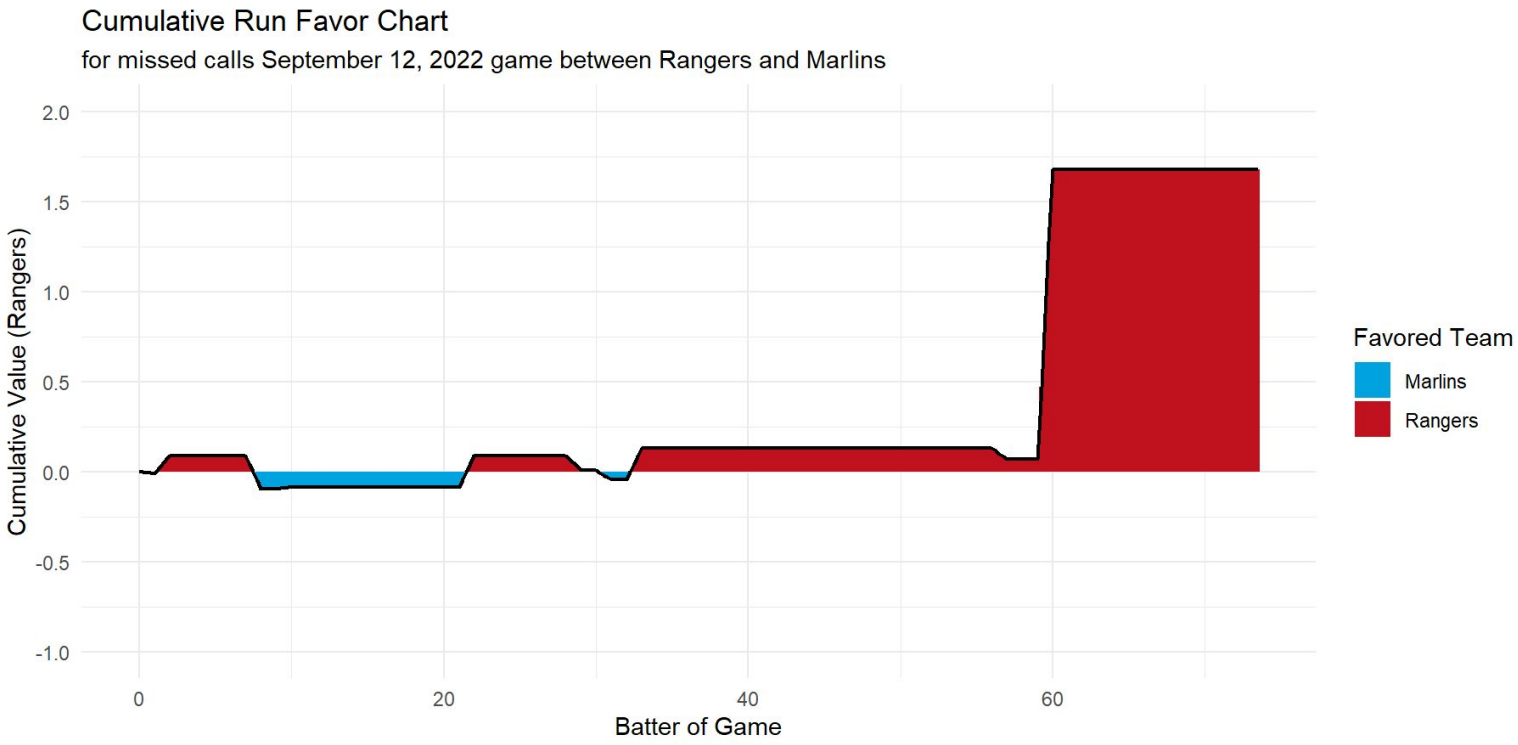
Game Summary

- September 12, 2022 (John Libka)
- Glenn Otto vs Trevor Rogers
- Rangers score 2 in top of the 7th inning and 1 in the top of the 8th to take lead and win
- Final Score 3-2 Rangers
- 11 total missed calls:
  - 10 with Rangers batting
  - 1 with Marlins batting

Overview

- Viewed each call individually with our created zone dimensions to determine the missed call
- Zone dimensions vary based off each hitter's height for that pitch
- 11 missed calls
  - 6 incorrect strikes
  - 5 incorrect balls





Total Favor: +1.680 runs for the Rangers

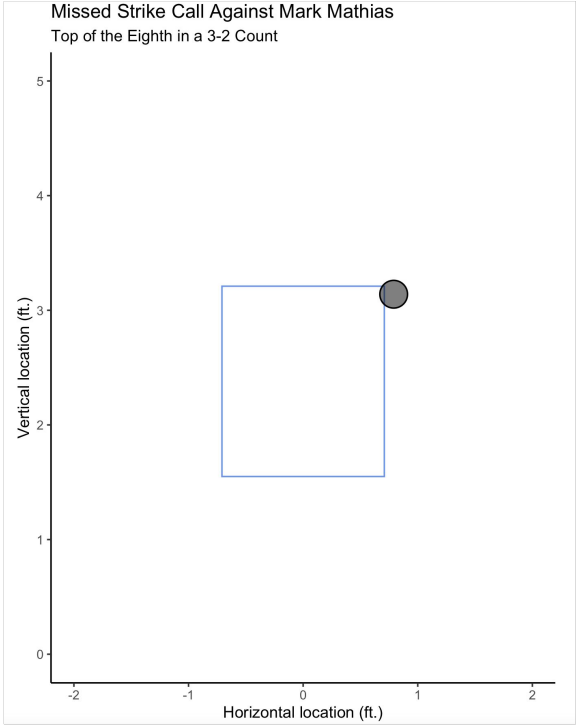
Most Impactful Calls

- Mark Mathias
  - +1.610 runs
  - Top of the 8th
- Mark Mathias
  - +0.185 runs
  - Top of the 2nd
- Nathaniel Lowe
  - -0.175 runs
  - Top of the 4th

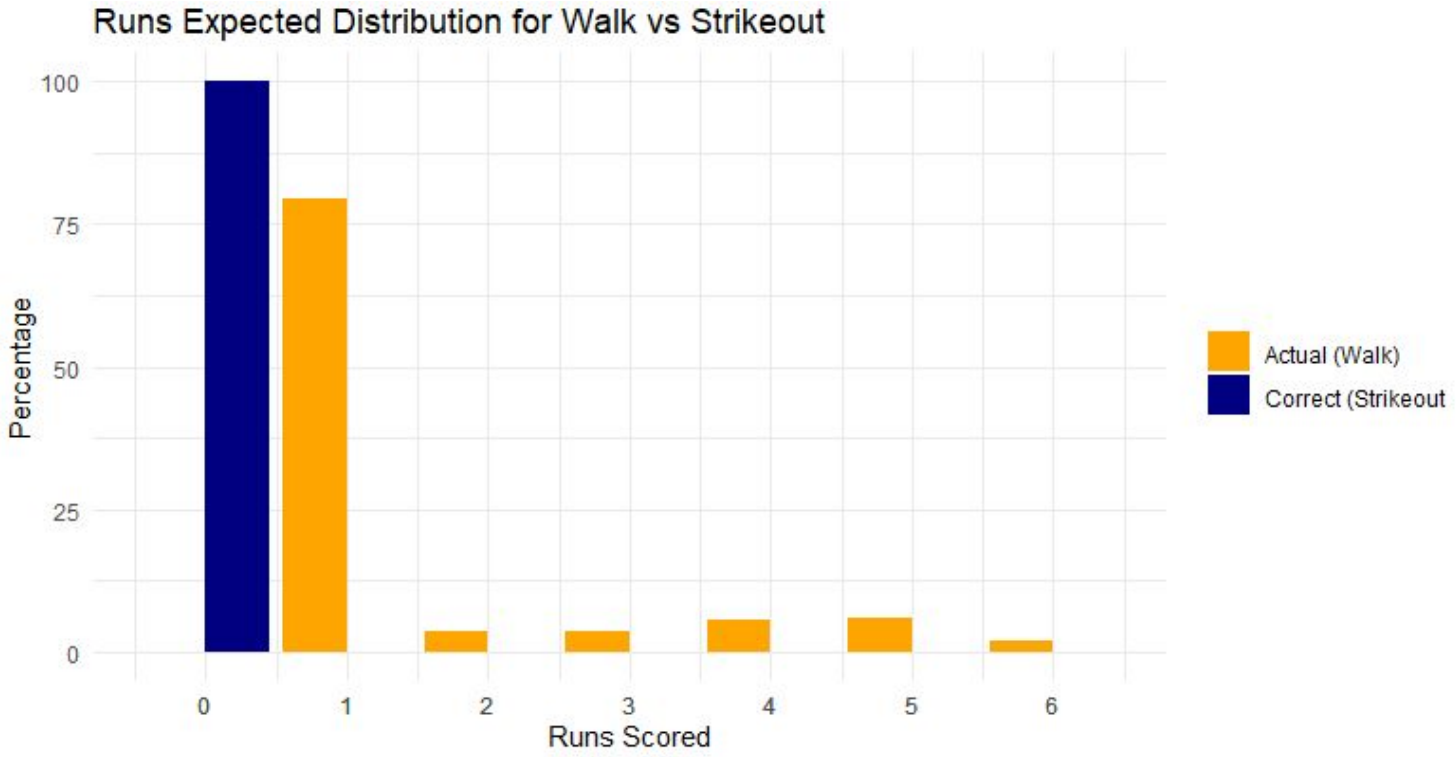
Impactful Call 1

- Inning: Top 8th Outs: 2
  Pitcher: Steven Okert
  Batter: Mark Mathias
  Bases Loaded
  - Actual
    - 4-2 (walk)
    - 1.610 wRE
  - Correct
    - 3-3 (strikeout)
    - 0 wRE

- Total effect: +1.610 runs



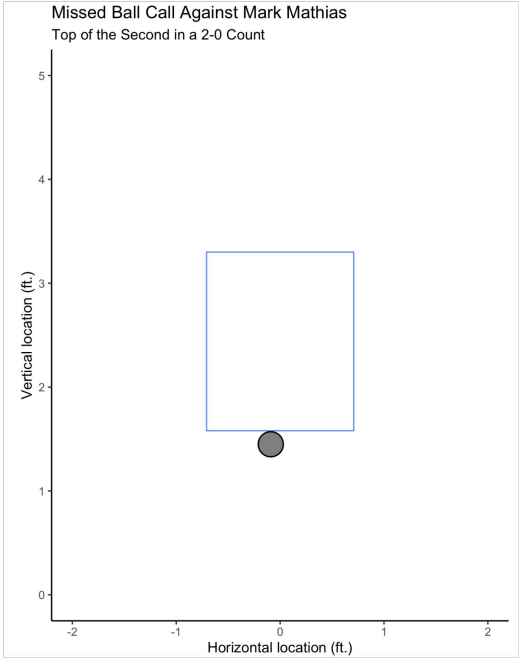
Impactful Call 1



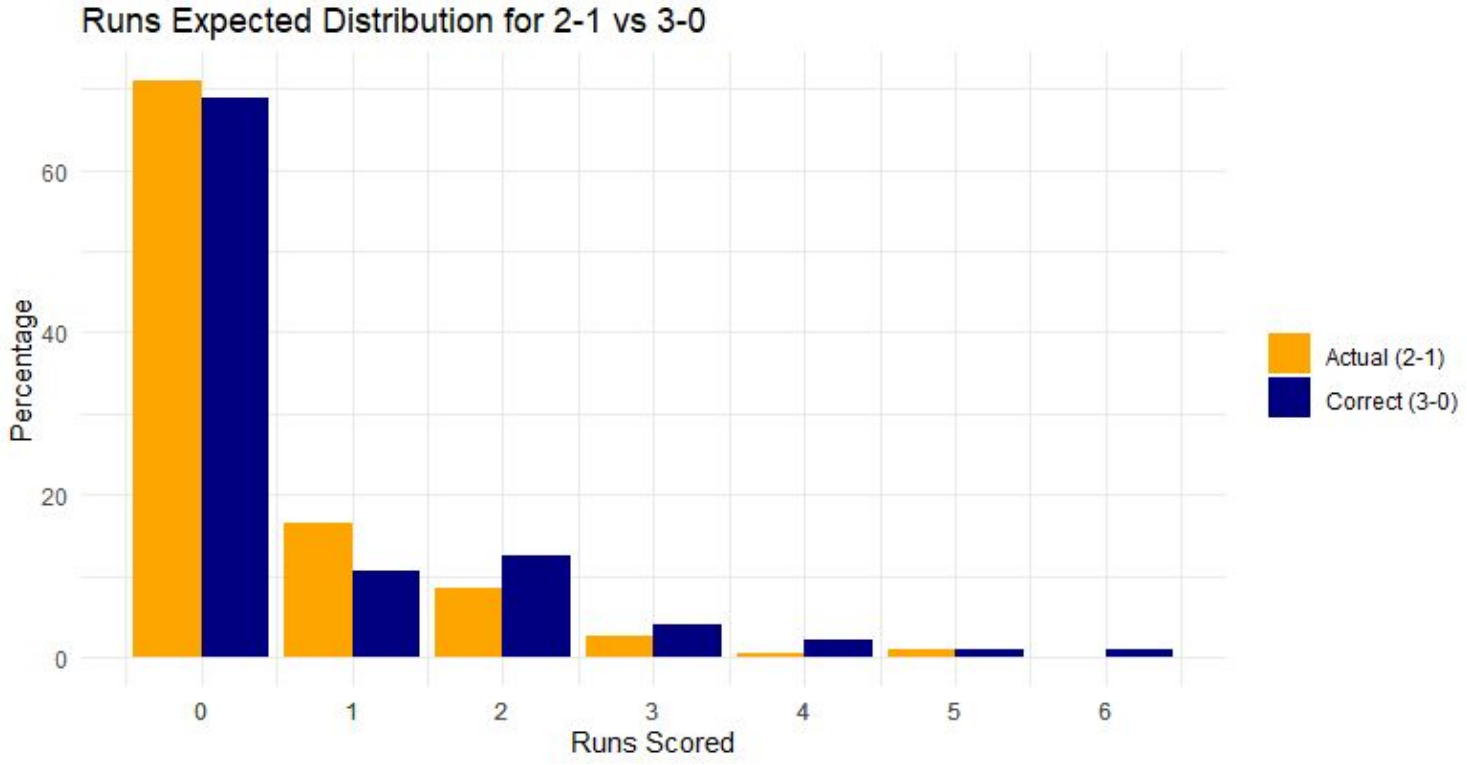
Impactful Call 2

- Inning: Top 2nd Outs: 0
  Pitcher: Trevor Rogers
  Batter: Mark Mathias
  - Actual
    - 2-1
    - 0.480 wRE
  - Correct
    - 3-0
    - 0.665 wRE

- Total effect: +0.185 runs



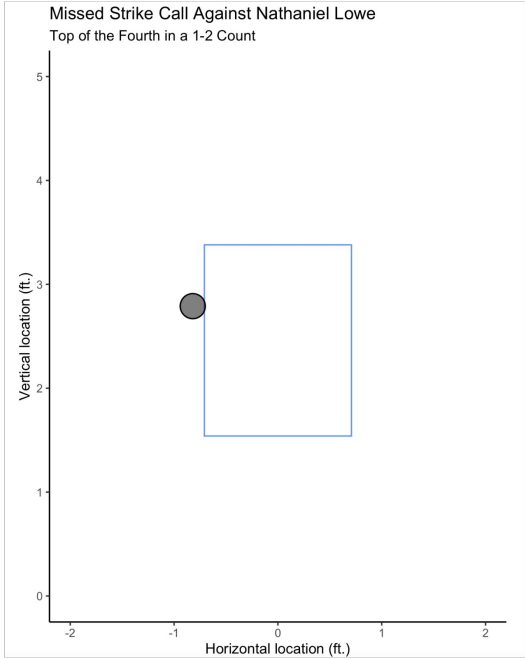
Impactful Call 2



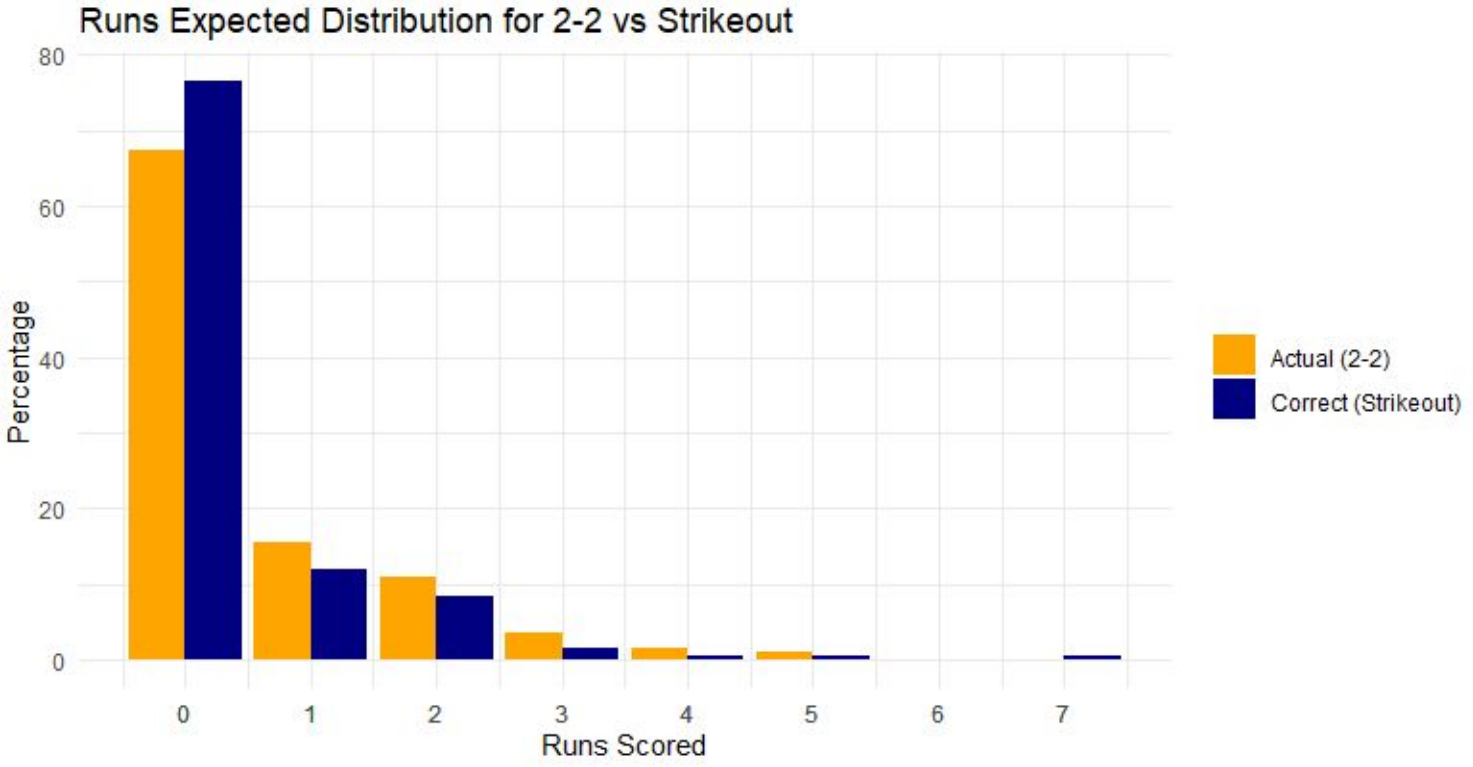
Impactful Call 3

- Inning: Top 4th Outs: 0
  Pitcher: Trevor Rogers
  Batter: Nathaniel Lowe
  - Actual
    - 2-2
    - 0.590 wRE
  - Correct
    - 1-3
    - 0.415 wRE

- Total effect: -0.175 runs



Impactful Call 3



Corrected Final Score

- Original
  - 3 - 2 Rangers win
- Adjustment
  - Rangers favored by +1.505
  - Marlins favored by -0.175
- Corrected Final Score
  - 2.18 - 1.50 Marlins win

Conclusions

Future Considerations

- Given more time, we could've ran more simulations and provided more accurate results
- More data to be able to go more in depth with the simulations
- Larger sample size of games to compare across games
- We could've evaluated umpire performance for all games, not just the ones provided

Conclusions

- Missed calls can change the outcome of a game
- The situation and magnitude of the missed call had more of an impact than the number of missed calls
- The most impactful calls were later in the games
  - Umpires might feel the pressure more later in the game
- Umpires called too big of a zone
  - May lead batters to expand zone

Questions?

Offensive Production Analysis in English Premier League

Tyler Bolebruch, Brett Gustin, Ryan Kamper, Marissa Schneider

Table of Contents

Research Question

What factors contribute most to a team's offensive production?

Methodology

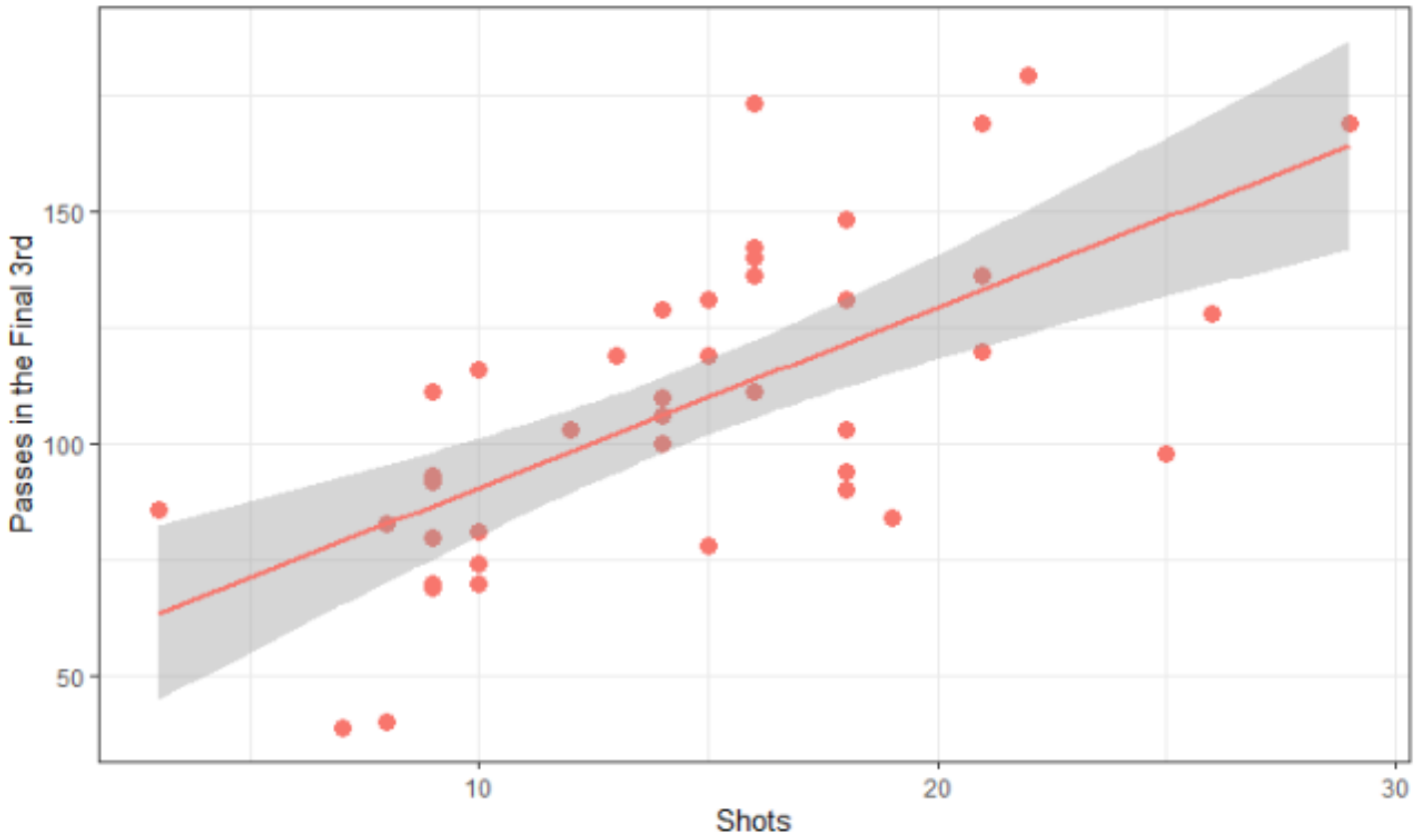
- Match data from English Premier League
- Multivariate Regression
- **<u>Response Variables</u>**

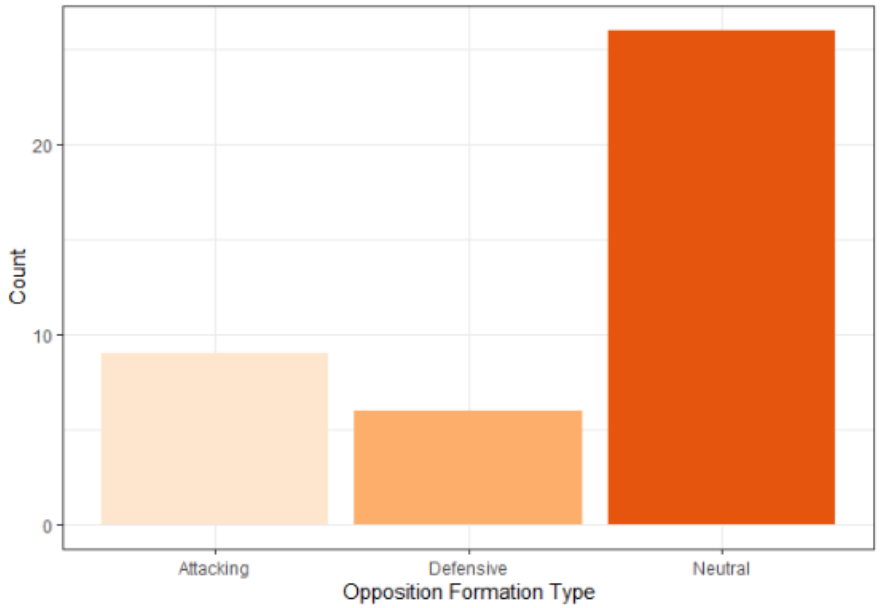
  -Shots

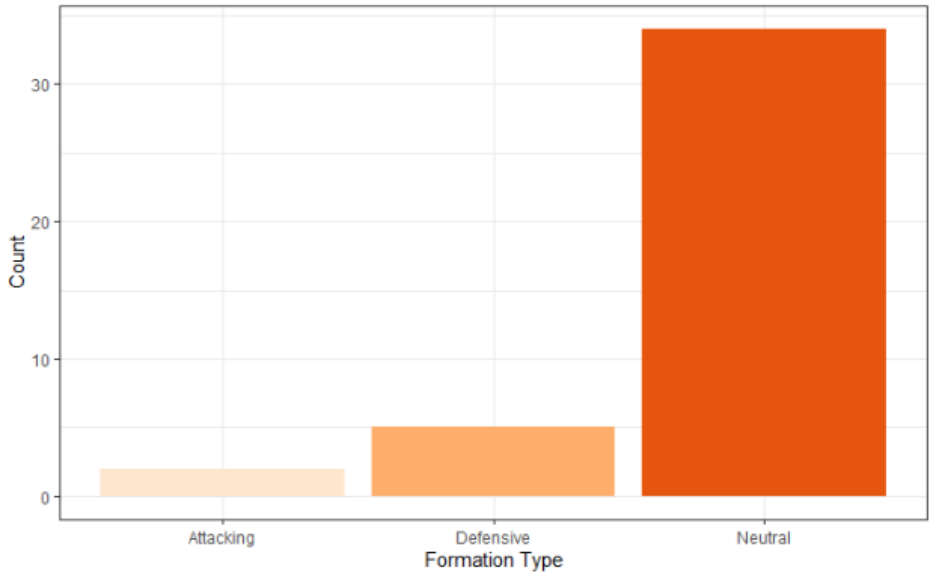
  -Passes in final third
- Dummy variable categorizing all formations in data

Multivariate Regression

- A regression that presents the same predictors that effect two different response variables
    - Appropriate when response variables are correlated





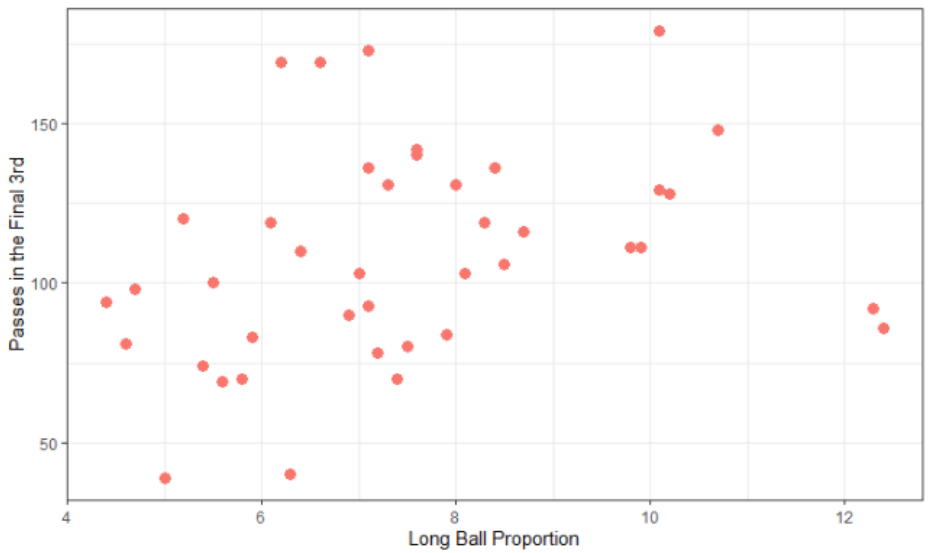


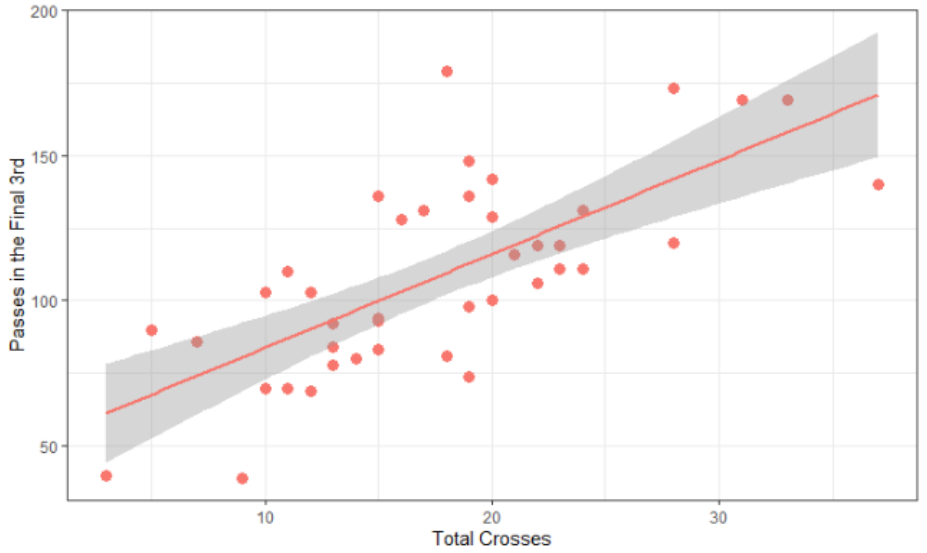
Formation	Formation Type
4-3-3	Attacking
3-4-2-1	Attacking
3-5-2	Attacking
4-4-2	Defensive
5-3-2	Defensive
5-4-1	Defensive
4-4-1-1	Defensive
4-2-3-1	Neutral
4-1-4-1	Neutral
4-2-2-2	Neutral
3-4-1-2	Neutral

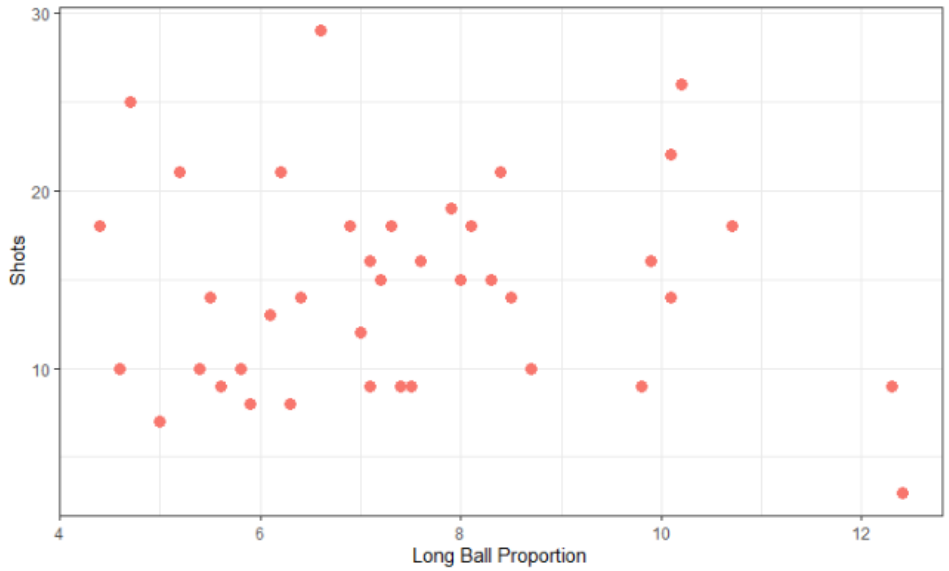
Initial Model

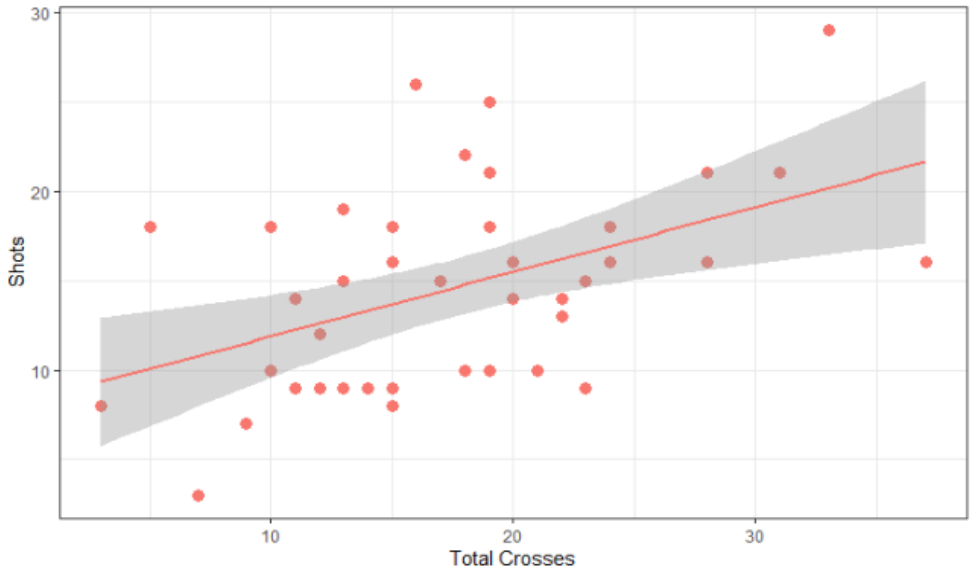
Passes in the Final 3rd & Shots = β_0 + β_1Opponent Shots + β_2Clearances+ β_3 Percent Corner Passes Short + β_4Total Crosses + β_5Long Ball Proportion + β_6Interception + β_7Tackle Success Rate + β_8Duel Success Rate + β_9Formation Type + β_{10}Opposition Formation Type + β_{11}(Formation Type x Opposition Formation Type)

Assumptions - Linearity









Assumptions- No Multicollinearity

Variance Inflation Factor(VIF)

	GVIF
Shots...Opposition	1.716775
Clearances...Team	1.904052
Corner.Short.Proportion	1.605559
Crosses...Team	1.795334
Long.Passes.Proportion.....Team	67.370586
I(Long.Passes.Proportion.....Team^2)	68.107352
Interceptions...Team	1.514417
Tackle.Success.Rate.....Team	1.705223
Duels.Success.Rate.....Team	1.138525
as.factor(Formation.Type)	1.907121
as.factor(Opposition.Formation.Type)	2.019890

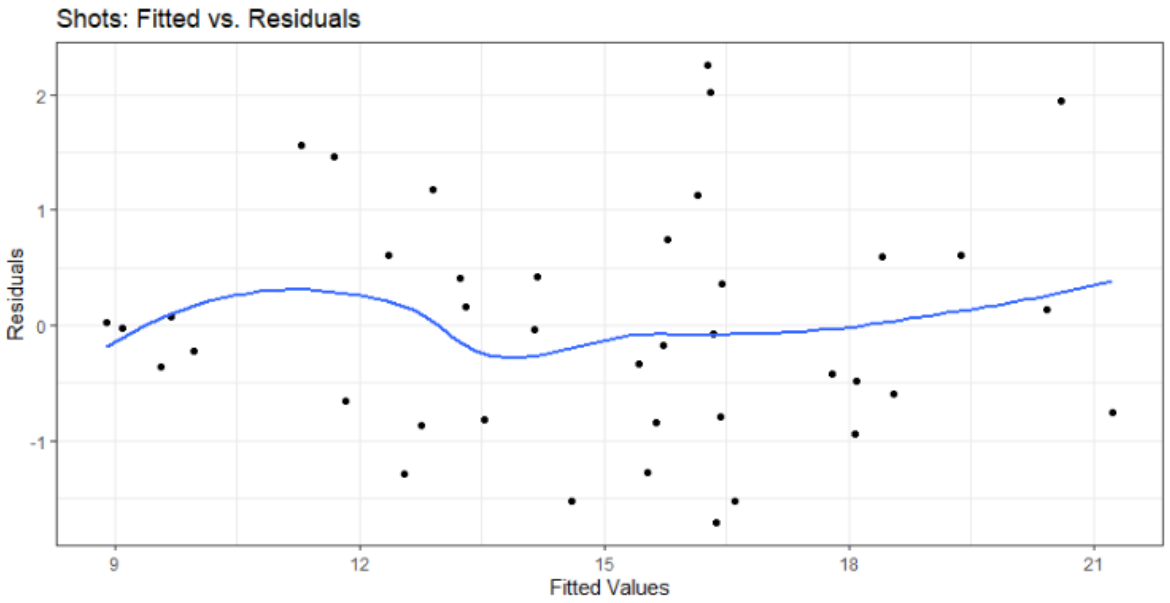
Assumptions- Independence

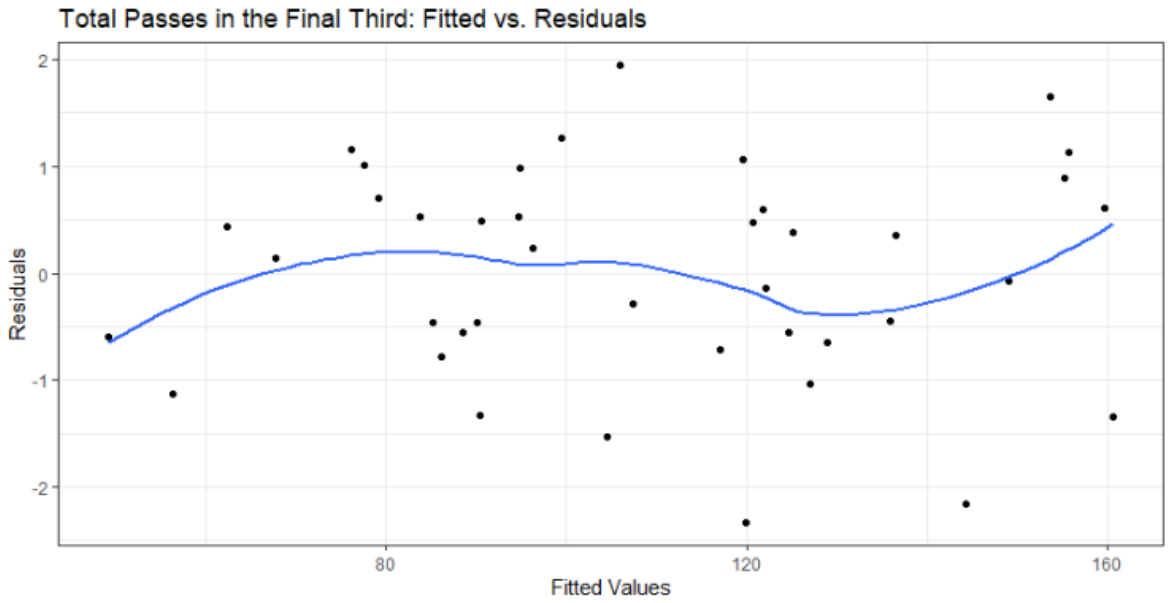
Durbin-Watson Test

```
> dwtest(Model2)

        Durbin-Watson test

data:  Model2
DW = 2.0378, p-value = 0.5206
alternative hypothesis: true autocorrelation is greater than 0
```

Assumptions- Constant Variance

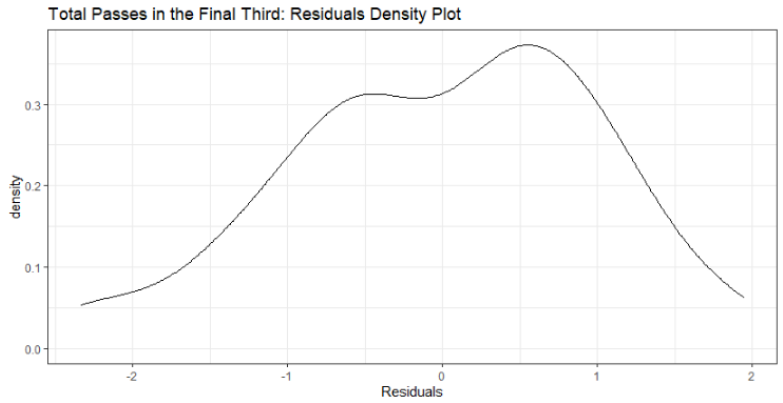
Fitted vs. Residuals

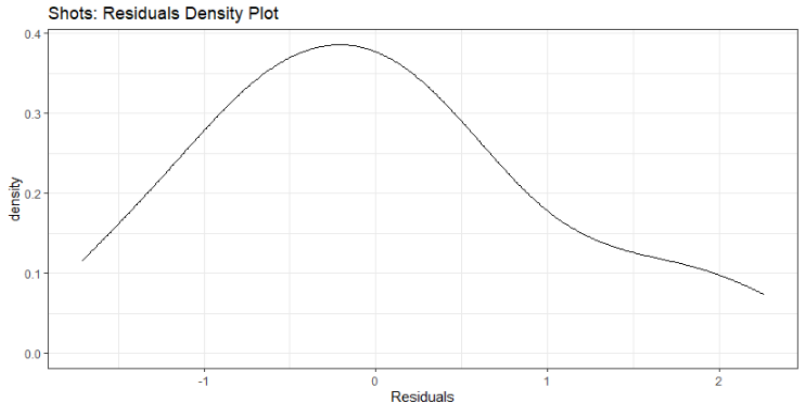




Assumptions- Normality in Error Term

```
        Shapiro-Wilk normality test

data:  Model2$residuals
W = 0.96502, p-value = 0.02749
```






```
        Shapiro-Wilk normality test

data:  Model3$residuals
W = 0.98676, p-value = 0.5827
```

Final Model

$$\textbf{Box Cox(Passes in the Final 3}^{\textbf{rd}}\textbf{ \& Shots)} = \beta_0 + \beta_1 \text{Opponent Shots} + \beta_2 \text{Clearances} + \beta_3 \text{ Percent Corner Passes Short} + \beta_4 \text{Total Crosses} + \beta_5 \text{Long Ball Proportion} + \beta_6 \text{Long Ball Proportion}^2 + \beta_7 \text{Interception} + \beta_8 \text{Tackle Success Rate} + \beta_9 \text{Duel Success Rate} + \beta_{10} \text{Formation Type} + \beta_{11} \text{Opposition Formation Type}$$

Final Model Analysis

	Dependent variable:	*Dependent variable:*
	Passes in the Final Third	Shots
Opponent Shots	-0.041	-0.040
	(0.043)	(0.041)
Clearances	-0.084**	-0.041
	(0.033)	(0.032)
Percent of Corners Passed Short	0.006	0.004
	(0.008)	(0.008)
Total Crosses	0.094***	0.016
	(0.027)	(0.026)
Long Ball Percentage	1.749**	0.830
	(0.657)	(0.625)
Long Ball Percentage Squared	-0.097**	-0.054
	(0.041)	(0.039)
Interceptions	-0.012	-0.010
	(0.031)	(0.030)
Tackle Success Rate	-0.017	-0.016
	(0.018)	(0.017)
Duel Success Rate	0.031	0.016
	(0.033)	(0.032)
Defensive Formation	-0.144	0.393
	(0.928)	(0.883)
Neutral Formation	1.027	0.559
	(0.789)	(0.751)
Opponent Defensive Formation	0.154	0.268
	(0.576)	(0.548)
Opponent Neutral Formation	0.160	-0.172
	(0.419)	(0.398)
Constant	2.748	1.892
	(3.736)	(3.554)
Observations	40	40
R^2	0.786	0.381
Adjusted R^2	0.679	0.072
Residual Std. Error	0.920 (df = 26)	0.875 (df = 26)
F Statistic	7.360*** (df = 13; 26)	1.231 (df = 13; 26)
Note:	*p<0.1; **p<0.05; ***p<0.01	p<0.1; **p<0.05; ***p<0.01

Limitations & Further Inquiry

- Not a large enough sample size
  - Increases significance in our coefficients
  - Be able to include interaction term that we had to remove
- In the future:
  - Provide more in-depth categorization of formation types
  - Some variable to measure the strength of the goalie/defense
  - Some metric that combine our two response variables rather than running a multivariate regression

Introduction to RLang

Will Cave, Ryan Kamper, Brett Gustin, & Eric Ducret

Frameworks

tidy eval:

- A programmable framework used in tidyverse packages like dplyr and ggplot2. Evaluates expressions in calling environments for complex functions through data-masking.

rlang errors:

- A set of tools to signal and display errors. Messages including errors and warnings make it easier to follow and create certain conditions.

rlang is mostly used by advanced programmers aiming to develop tools, however, there are practical applications for other users as well.

Falk College

Dataset

The dataset used in this presentation is from the 2019 MLB season, scraped via baseballr.

All other data that is used will be specified before usage.

```
── MLB Schedule data from MLB.com ──────────────
ℹ Data updated: 2022-11-28 15:51:58 EST
# A tibble: 3,010 × 70
   date       total_i…¹ total…² total…³ total…⁴
   <chr>          <int>   <int>   <int>   <int>
 1 2019-02-21         1       0       1       0
 2 2019-02-22         4       0       4       0
 3 2019-02-22         4       0       4       0
 4 2019-02-22         4       0       4       0
 5 2019-02-22         4       0       4       0
 6 2019-02-23        16       0      16       0
 7 2019-02-23        16       0      16       0
 8 2019-02-23        16       0      16       0
 9 2019-02-23        16       0      16       0
```

Falk College

Defusing function arguments with !!enquo()

Rlang's !!enquo() and base R substitute() can view the calling argument for a function to identify expressions

substitute() does this only once, while !!enquo() can identify the calling argument for nested functions

eval_tidy() allows for evaluation of quosures in their stored environment

Shorthand for !!enquo is now available with {{ as seen in the next slide

```
> f <- function(data) {
+   eval(substitute(data),
+   list(a = 2, b = 3))
+ }
> f(a+b)
[1] 5
> g <- function(data) {
+   val <- f(substitute(data))
+   val
+ }
> g(a+b)
data
```

```
> f2 <- function(data) {
+   eval_tidy(enquo(data),
+   list(a = 2, b = 3))
+ }
> g2 <- function(data) {
+   val <- f2(!!enquo(data))
+   val
+ }
> g2(a+b)
[1] 5
```

Falk College

Data Masking

Certain functions will mask the code to prevent operation.

Using an embracing operator will inject the code supplied to the function and prevent the code from not being operated on.

This is useful for functions that involve other elements of the tidyverse.

```
> my_mean_1 <- function(data, var1, var2) {
+   dplyr::summarise(data, mean(var1 + var2))
+ }
> my_mean_1(season, total_items, game_number)
Error in `dplyr::summarise()`:
! Problem while computing `..1 = mean(var1 + var2)`.
Caused by error in `mean()`:
! object 'total_items' not found
Run `rlang::last_error()` to see where the error occurred.
```

```
> my_mean_2 <- function(data, var1, var2) {
+   dplyr::summarise(data, mean({{ var1 }} + {{ var2 }}))
+ }
> my_mean_2(season, total_items, game_number)
# A tibble: 1 × 1
  `mean(total_items + game_number)`
                              <dbl>
1                              15.2
```

```
> my_mean_3 <- function(var1, var2) {
+   mean(var1 + var2)
+ }
> my_mean_3(season$total_items, season$game_number)
[1] 15.21528
```

Falk College

Data Masking

This is another example of injecting code into the function.

Without the double brackets, the function would not run.

This is because the tidyverse functions defuse the code and make it inoperable.

```
> wpct_z_score <- function(data, var1, var2, var3) {
+   data %>% group_by({{var1}}) %>%
+     mutate(z_score = ({{var2}} - mean({{var2}}))/sd({{var2}})) %>%
+     select({{var1}}, {{var2}}, z_score) %>% filter({{var1}} > {{var3}})
+ }
> wpct_z_score(season, date, teams_away_league_record_pct, '2019-07-01')
# A tibble: 1,220 × 3
# Groups:   date [114]
   date       teams_away_league_record_pct z_score
   <chr>                             <dbl>   <dbl>
 1 2019-07-02                        0.523  0.229
 2 2019-07-02                        0.386 -1.02
 3 2019-07-02                        0.529  0.283
 4 2019-07-02                        0.282 -1.96
```

Falk College

rlang Functions for Function Manipulation

new_function()

Used to create a function within an environment.

Gives the ability to set default arguments for the function.

You can also specify the environment that the function will be in.

new_function(args, body, env)

```
> func_1 <- new_function(
+   list(x = 1, y = NULL),
+   quote(x + y))
> func_1(y = 4)
[1] 5
> func_1(3, 4)
[1] 7
```

Falk College

as_function()

Turns a formula into a function with specified arguments.

Tilda is needed at the beginning of the arguments.

A dot is placed in front of each variable, if there is only one variable, it does not need to be specified, but a dot is still necessary.

When working within the tidyverse, specify rlang before calling as_function.

```
> func_2 <- rlang::as_function(~ 4 + .)
> func_2(12)
[1] 16
> func_3 <- rlang::as_function(~ 4 + .x)
> func_3(12)
[1] 16
> func_4 <- rlang::as_function(~ .x * .y)
> func_4(6, 7)
[1] 42
```

Falk College

is_function()

Evaluates whether or not the argument is a function.

Works with closures, eager primitive functions, and lazy primitive functions.

```
> is_function(func_4)
[1] TRUE
> is_closure(func_4)
[1] TRUE
> is_primitive(c)
[1] TRUE
> is_primitive_eager(c)
[1] TRUE
> is_primitive_lazy(quote)
[1] TRUE
```

Falk College

Types of functions

Closures

- Take formal arguments
- Creates a new environment where the function is evaluated.

Primitives

- Do not take formal arguments
- Written in low level code

Eager Primitives

- Arguments are evaluated when they are passed through

Lazy Primitives

- Primitive functions that operate on expressions
- Arguments are not evaluated when passed through

Example of a lazy function

```
> lazy_func1 <- quote(5 + 6 + 7)
> lazy_func1
5 + 6 + 7
```

Falk College

fn_fmls()

Returns the formal arguments of a function.

Variations:

fn_fmls_names(): Returns the names of each argument.

fn_fmls_syms(): Returns the arguments as a named list of symbols.

```
> fn_fmls(wpct_z_score)
$data


$var1


$var2


$var3
```

```
> fn_fmls_names(wpct_z_score)
[1] "data" "var1" "var2" "var3"
```

```
> fn_fmls_syms(wpct_z_score)
$data
data

$var1
var1

$var2
var2

$var3
var3
```

Falk College

`fn_fmls<-`()

```
> fn_fmls(func_1) <-
+   list(x = 3, y = 5)
> func_1()
[1] 8
```

Changes the default arguments or the names of the function.

`fn_fmls_names<-`() changes the names of the arguments.

```
> fn_fmls_names(wpct_z_score) <-
+   c("dataset", "date_column", "wpct", "start_date")
> fn_fmls_names(wpct_z_score)
[1] "dataset"     "date_column" "wpct"        "start_date"
```

Falk College

fn_body()

Returns the body of a function.

Can be used to set the body with “<-”

```
> fn_body(func_4)
{
    .x * .y
}
```

fn_env()

Returns the environment of a function.

Can be used to set the environment with “<-”

```
> fn_env(func_4)
<environment: R_GlobalEnv>
```

Falk College

as_closure()

Wraps primitive functions inside closures

Runs the function as a closure instead of a primitive.

```
> as_closure(c)
function (...)
c(...)
<bytecode: 0x104cd0960>
<environment: 0x11f1574d8>
```

Falk College

rlang Errors

Signaling Errors

abort(), warn(), and inform() display errors, warnings, or messages.

Useful for customizing errors in package or function development.

cnd() signals a condition.

```
> abort("Error")
Error:
! Error
Run `rlang::last_error()` to see where the error occurred.
> warn("Warning Message")
Warning message:
Warning Message
> inform("Information")
Information
```

```
> cnd <- warning_cnd("my_warning_class", message = "Custom Warning Message")
> cnd_signal(cnd)
Warning message:
Custom Warning Message
```

Falk College

Handling Errors

global_handle() - Sets up default error message configurations

global_entrace() - Enable rlang errors in the global environment

global_prompt_install() - Prompts the user to install a missing package without stopping the current program.

```
global_handle(entrace = TRUE,
              prompt_install = TRUE)

rlang::global_entrace()
global_entrace(enable = TRUE,
               class = c("error",
                         "warning",
                         "message"))

global_prompt_install(enable = TRUE)
```

Handling Errors

try_fetch() takes an expression as an argument and can also take functions like abort() to display error messages.

```
> try_fetch(1 + "", error = function(cnd)
+   abort("Expression could not be evaluated.", parent = cnd))
Error:
! Expression could not be evaluated.
Caused by error in `1 + ""`:
! non-numeric argument to binary operator
Run `rlang::last_error()` to see where the error occurred.
```

catch_cnd() displays the condition of the arguments in the function.

```
> catch_cnd(abort("Error Message"))
<error/rlang_error>
Error:
! Error Message
---
Backtrace:
 1. rlang::catch_cnd(abort("Error Message"))
 8. rlang::abort("Error Message")
```

Falk College

Backtracking Errors

Backtracking errors allows the user to find what the specific error in the code is.

Functions include:

- last_error() and last_trace() display the last error recorded within the environment.
- last_warnings() and last_messages() display the last warnings.
- Other functions include global_entrace(), trace_back(), and trace_length()

```
> my_mean_1 <- function(data, var1, var2) {
+   dplyr::summarise(data, mean(var1 + var2))
+ }
> my_mean_1(season, total_items, game_number)
Error in `dplyr::summarise()`:
! Problem while computing `..1 = mean(var1 + var2)`.
Caused by error in `mean()`:
! object 'total_items' not found
Run `rlang::last_error()` to see where the error occurred.
> last_error()
<error/rlang_error>
Error in `dplyr::summarise()`:
! Problem while computing `..1 = mean(var1 + var2)`.
Caused by error in `mean()`:
! object 'total_items' not found
---
Backtrace:
  1. global my_mean_1(season, total_items, game_number)
 10. base::mean(var1 + var2)
Run `rlang::last_trace()` to see the full context.
```

Falk College

Error Conditions

format_error_bullets() can change the format of a bullet point within an output.

```
> writeLines(format_error_bullets
+            (set_names(c("Line One",
+                         "Line Two",
+                         "Line Three"),
+                       "i")))
i Line One
i Line Two
i Line Three
```

These can also be assigned individually.

```
> writeLines(format_error_bullets(c(x = "Red X",
+         v = "Green Check", "*" = "Blue Dot")))
✖ Red X
✔ Green Check
● Blue Dot
```

Falk College

Other Functions in rlang

Other Functions in rlang

flatten: Flatten/squash a list of lists into a simple vector

```
> x <- list(1, 2, 3)
> flatten(x)
[[1]]
[1] 1

[[2]]
[1] 2

[[3]]
[1] 3
```

set_names: Sets name of a vector

```
> set_names(1:4, letters[23:26])
w x y z
1 2 3 4
```

Falk College

Working in the R session

is_installed: Checks if packages are installed in the specified environment.

```
> is_installed("baseballr", call = caller_env())
[1] TRUE
> is_installed("bayestestR", call = caller_env())
[1] TRUE
> is_installed("notapackage", call = caller_env())
[1] FALSE
```

env: Creates an environment with specified elements.

```
> env <- env(a = 4, b = TRUE, c = rlang::as_function(~ .x + .y))
> env$a
[1] 4
> env$b
[1] TRUE
> env$c(4, 5)
[1] 9
```

Falk College

Checking an Object: is_

is_"" is useful to test to see what the object created is defined as.

is_function: is object a function?

is_formula: is object a formula?

is_call: is object a call?

```
> my_function <- function(data) {
+   x + y
+ }
> is_function(my_function)
[1] TRUE
> is_formula(my_function)
[1] FALSE
> is_call(my_function)
[1] FALSE
```

is_callable: is an object callable?

is_conditon: is object a condition?

is_environment: is object an environment?

Is_true: Is object identical to TRUE or FALSE?

```
> is_callable(my_function)
[1] TRUE
> is_condition(my_function)
[1] FALSE
> is_environment(my_function)
[1] FALSE
> is_true(my_function)
[1] FALSE
```

Falk College

Creating vectors

rlang has functions similar to c() that create vectors. The functions specify which type of data is inside the vector.

lgl(): Creates a logical vector.

```
> lgl(TRUE, FALSE, FALSE, TRUE, FALSE)
[1]  TRUE FALSE FALSE  TRUE FALSE
```

int(): Creates an integer vector.

```
> int(1, 2, 3, 4, 5)
[1] 1 2 3 4 5
```

TRUE and FALSE are coerced to 1 and 0, respectively.

dbl(): Creates a double vector.

```
> dbl(1.5, 6.7, 9.4, 13.2, TRUE)
[1]  1.5  6.7  9.4 13.2  1.0
```

cpl(): Creates a complex vector.

```
> cpl(1, 4.5, 8.9, TRUE, 23)
[1]  1.0+0i  4.5+0i  8.9+0i  1.0+0i 23.0+0i
```

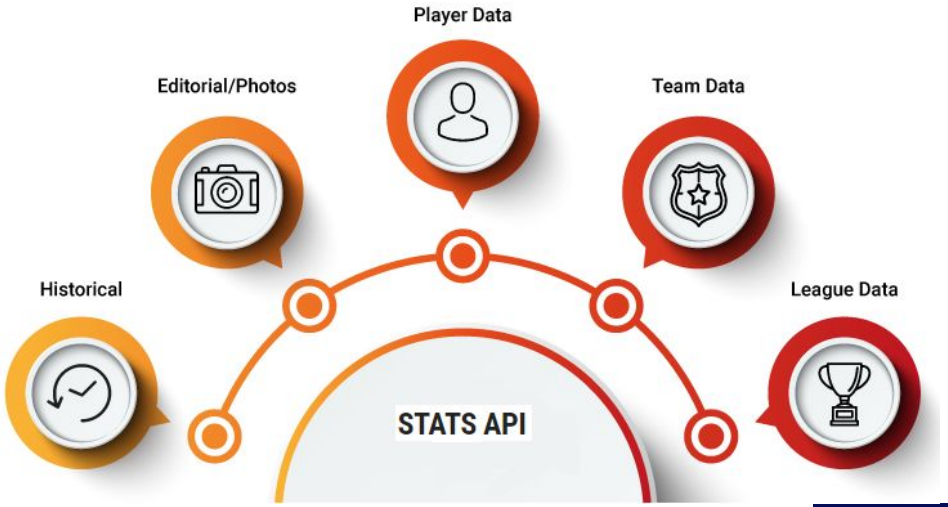
chr(): Creates a character vector.

```
> chr("1", "Syracuse", "UNC")
[1] "1"        "Syracuse" "UNC"
```

Falk College

Applications in Sport

- Rlang provides a consistent structure for API's (application programming interface)
- **API's:** sets of protocols used to construct application software used to communicate with other computer programs
- Large betting corporations implement their own API's to constantly connecting with other computer systems to retrieving lists of live scores and game data
- Database systems within Major and Minor league teams need API's to connect system storage of data



Falk College

Citations

https://rlang.r-lib.org/

https://www.rdocumentation.org/packages/rlang/versions/1.0.6

https://github.com/r-lib/rlang

https://www.tidyverse.org/blog/2019/06/rlang-0-4-0/

Falk College

NBA Attendance Regression

Brett Gustin, Jordan Jones, Tyler Bolebruch, Ryan Kamper, Nicholas Kamimoto

Introduction

- This model uses the NBA schedule from the 2016-2017, 2017-2018, 2018-2019 seasons.
- The regression demonstrates the effects of our selected variables on the attendance over these seasons
- We ran two regressions
  - In game player/team data
  - External variables unrelated to in game

Regression 1

Variables:

- Home/Away Simple Rating System (SRS
- Stars
- Opening Point Spread (Quadratic Term)
- Inter-Conference
- Game Importance

$$\text{Attendance} = \alpha + \beta_1 \text{Inter-Conference Matchup} + \beta_2 \text{TotalStars} + \beta_3 \text{PointSpread} + \beta_4 \text{PointsSpread}^2 + \beta_5 \text{HomeSRS} + \beta_6 \text{AwaySRS} + \beta_7 \text{Top10vsTop10} + \beta_8 \text{Top10vsNonTop10} + \varepsilon$$

REGRESSION 1 RESULTS

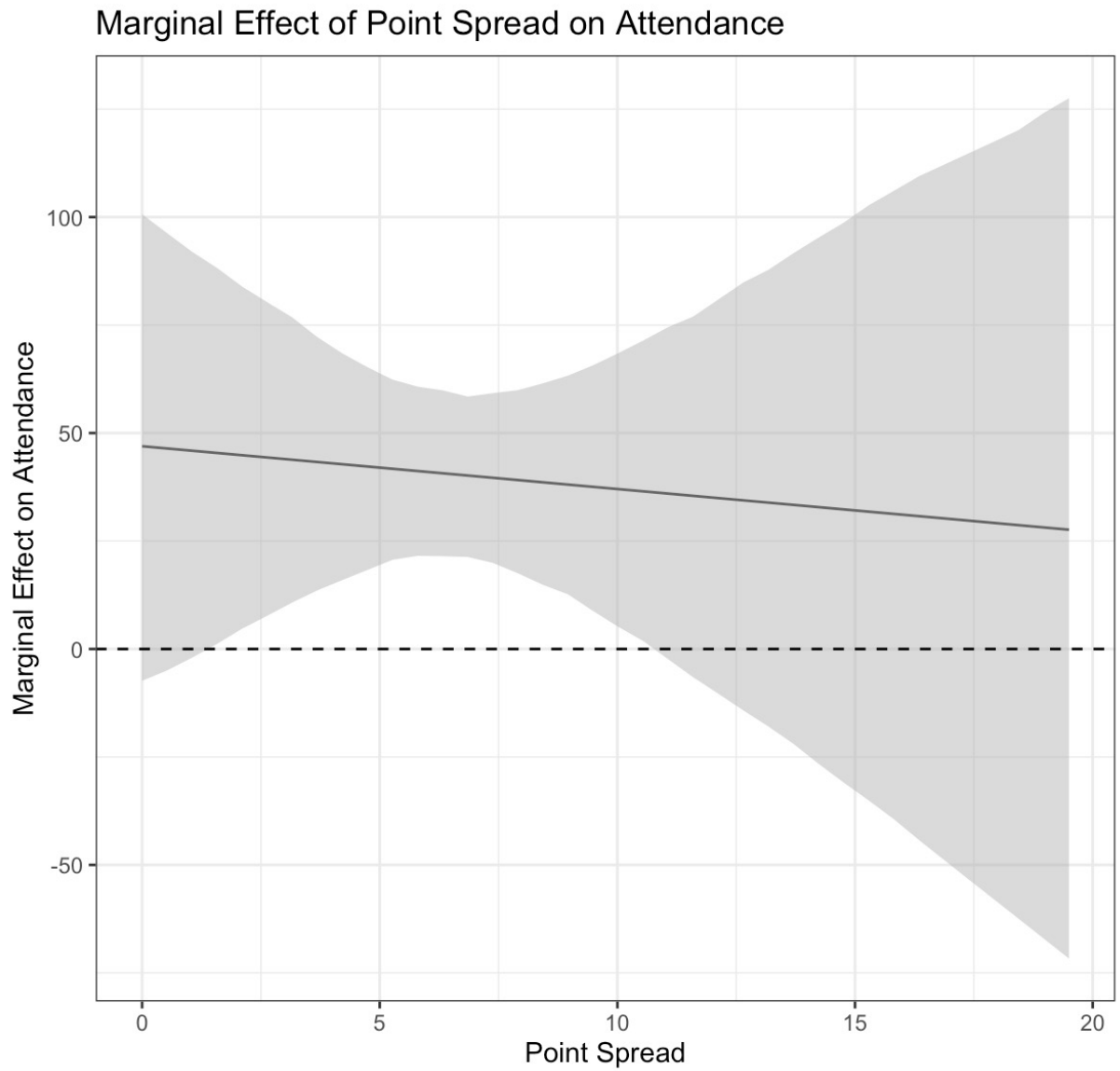
	Dependent variable:
	Attendance
Interconference Mathup	-125.421
	(77.639)
Total Stars	659.972***
	(39.925)
Point Spread	47.575
	(33.743)
Home Simple Rating System	23.434**
	(9.610)
Away Simple Rating System	-9.642
	(9.754)
Point Spread Squared	-0.546
	(2.382)
Top 10 vs Top 10	580.762***
	(117.262)
Top 10 vs Non Top 10	387.081***
	(115.051)
Constant	16,939.890***
	(114.070)
Observations	2,502
R^2	0.160
Adjusted R^2	0.157
Residual Std. Error	1,865.357 (df = 2493)
F Statistic	59.156*** (df = 8; 2493)
Note:	*p<0.1; **p<0.05; ***p<0.01

SUMMARY STATISTICS

NBA Attendance Summary Statistics

Statistic	N	Mean	St. Dev.	Min	Pctl(25)	Pctl(75)	Max
Average Temperature	2,502	47.775	15.490	-22.300	37.500	59.000	86.900
Precipitation (in)	2,502	0.085	0.270	0.000	0.000	0.010	3.770
Point Spread	2,502	5.753	3.708	0	3	8.5	20
Home Simple Rating System	2,502	0.005	4.397	-9.390	-2.418	3.230	11.350
Away Simple Rating System	2,502	0.016	4.401	-9	-2.4	3.2	11

MARGINAL EFFECTS PLOT



Regression 2

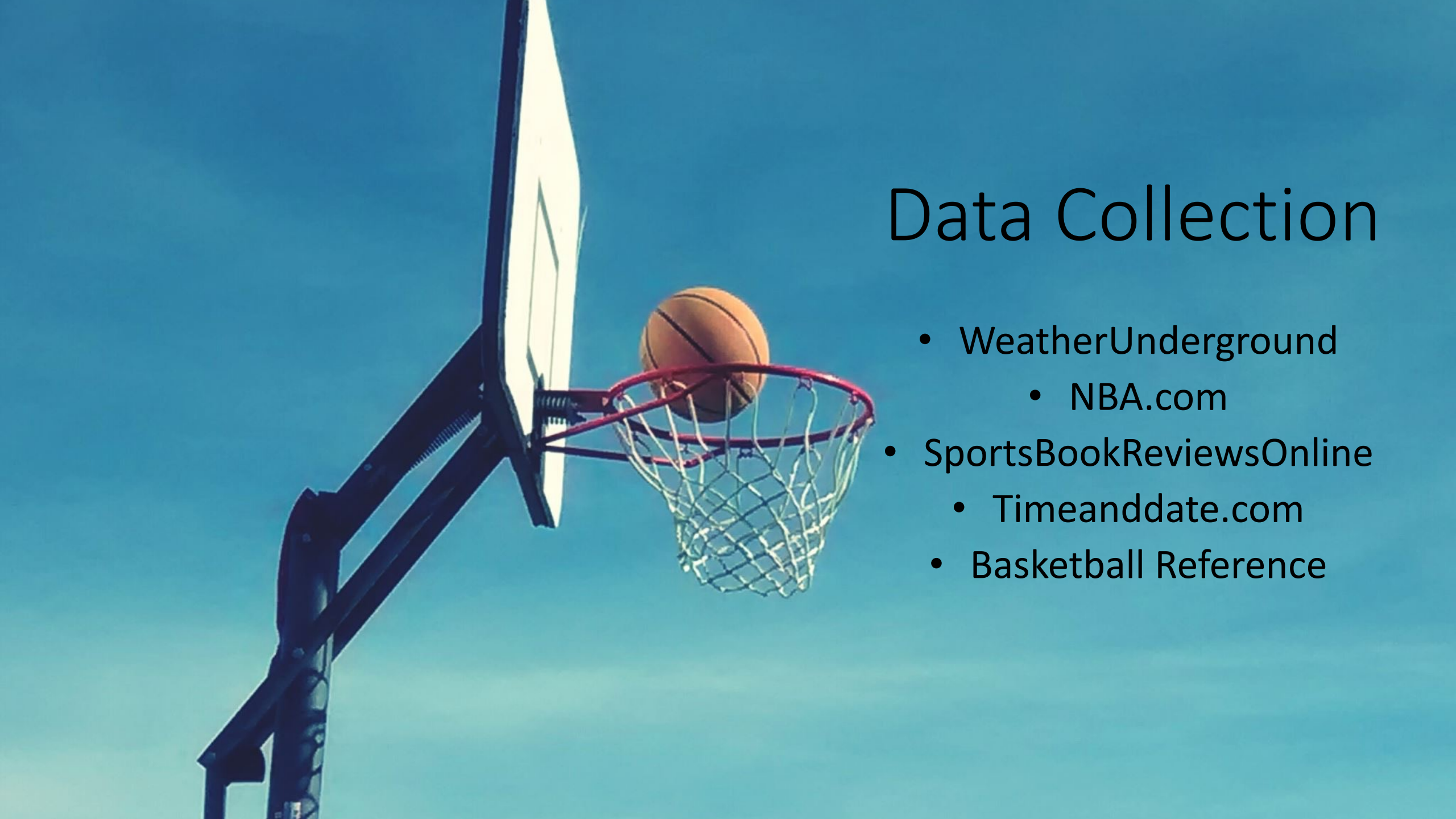
Variables:

- Weather(Temperature, Precipitation)
- Start Time
- Season
- Holidays
- Day of Week

$$\text{Attendance} = \alpha + \beta_1 2017\text{-}18\text{Season} + \beta_2 2018\text{-}19\text{Season} + \beta_3 \text{Temperature} + \beta_4 \text{Precipitation} + \beta_5 \text{Holiday} + \beta_6 \text{Monday} + \beta_7 \text{Tuesday} + \beta_8 \text{Thursday} + \beta_9 \text{Friday} + \beta_{10} \text{Saturday} + \beta_{11} \text{Sunday} + \beta_{12} \text{Afternoon} + \beta_{13} \text{Evening} + \varepsilon$$

REGRESSION 2 RESULTS

	Attendance
2017-2018 Season	-455.977
	(306.191)
2018-2019 Season	-584.546*
	(306.283)
Average Temperature	-3.012
	(2.610)
Precipitation (in)	-187.892
	(148.480)
Holiday	0.568
	(219.970)
Monday	-253.440*
	(140.310)
Tuesday	304.497**
	(148.402)
Thursday	552.515***
	(163.088)
Friday	581.505***
	(131.258)
Saturday	871.605***
	(135.435)
Sunday	344.377**
	(166.353)
Afternoon Game	223.734
	(185.288)
Evening Game	220.250
	(152.699)
Constant	18,232.710***
	(349.809)
Observations	2,502
R^2	0.038
Adjusted R^2	0.033
Residual Std. Error	1,998.173 (df = 2488)
F Statistic	7.463*** (df = 13; 2488)
Note:	*p<0.1; **p<0.05; ***p<0.01



Data Collection

- WeatherUnderground
- NBA.com
- SportsBookReviewsOnline
- Timeanddate.com
- Basketball Reference

Results

- Day of the week was expected
- Holiday a little surprising
- Could include ticket prices
- Could improve game importance