

Triangle Tribulations

Sonya Lasser - Team 166 - July 2026

Table of Contents

Abstract	2
1 Introduction	3
2 Bermuda Triangles - A Definition	3
3 Proof of Concept	4
4 Identifying Triangle Plays	5
5 Triangle Catch Probability (TCP) Model	6
6 Conclusion	7
Acknowledgements	8
Bibliography	9
Appendix A: Analysis of Uncaught Bermuda Triangles	11
Appendix B: Model Evaluation	15
Appendix C: R Code	18
Data Cleaning	18
Model Creation	27
Visualizations	34

Abstract

The prototypical, repeated baseball blooper is the “Bermuda Triangle.” Three (or more) players are all in range of catching a pop up, sprinting towards the ball, and somehow the ball drops between them. MLB’s Statcast has a catch probability metric, which is used to analyze unexpected catches, but for Bermuda-Triangle-specific plays, a metric comparing players to each other would be of greater use. As a proof-of-concept animation, I used SMT’s anonymized data with a new Triangle Catch Probability (TCP) metric attached to anonymous player headshots. The TCP was created using a mixed conditional logit model, which allowed for variables to differ for each individual on the field (fielder position and direction to run in) as well as variables that are constant for everyone on a given play (the hit’s exit velocity). The main limitation of my TCP model is the lack of a probability for the ball being dropped, though the model would be equipped for that when trained on more data. With a combination of the TCP metric and replays of these miscommunications, MiLB teams can create a customizable, informative, and entertaining blooper reel to show on game-day.

1 Introduction

Everyone loves a blooper reel¹. The most sports-avoidant person and biggest die-hard fan can laugh together over a humorous collision, miscommunication, or slip-up on the field (this² baseball blooper reel has over 20 million views on YouTube). Shane Rogers of The Conversation discusses the appeal of bloopers: they are unexpected, violate the “norm” of perfection in one’s position, and make the moment relatable to the audience³. The inability to choreograph baseball to include gag plays or bloopers is part of what makes the game so special – even the Savannah Bananas⁴ do not script their gameplay.

The prototypical, repeated baseball blooper is the infamous “Bermuda Triangle⁵.” Three (or more) players are all in range of catching a pop up, sprinting towards the ball. All of a sudden, they slow down or stop entirely – “the other guy will get it, of course,” they each think. And then the ball drops. What should have been an easy out becomes a base hit. This humorous scene for fans is often an embarrassment for the defense; an embarrassment that typically cannot be recorded as an error, since no one touched the ball. While some fans ponder who is at fault on the field, others watch the replay and laugh, or breathe a sigh of relief that a runner is finally on base.

MLB’s Statcast has a catch probability metric⁶, which is used to analyze the most unexpected and impressive catches, but for Bermuda-Triangle-specific plays, an alternative probability metric, comparing players to each other, would be of greater use. With a combination of a Triangle Catch Probability (TCP) metric and replays of these miscommunications, MiLB teams can create an informative, and entertaining, blooper reel to play during games.

2 Bermuda Triangles - A Definition

For this project, I defined a Bermuda Triangle play as being one where three or more players would each need an average pace of 25 ft/sec or less to reach the ball. Additionally, with 2.35 seconds left before the ball drops, the players each need an average pace of 30 ft/sec or less, since they would be running at full speed by that point. These numbers are based on Statcast’s MLB 2016-2026 league average sprint speed⁷ of 27 ft/sec, since I wanted the necessary speed to be slower than the average sprint speed but not excessively so. While filtering the data, there were no instances of batted balls in play that were caught by a player who had to average over 25 ft/sec to reach the ball from their initial position. Since the player velocities calculated from the data are

¹ <https://www.youtube.com/watch?v=cdmKVXynX3I>

² <https://www.youtube.com/watch?v=8Ket3BFas7E>

³ <https://theconversation.com/why-are-blooper-reels-so-funny-221492>

⁴ <https://www.youtube.com/shorts/tb2j1DPomFs>

⁵ <https://www.youtube.com/watch?v=yOtrjoQoSJM>

⁶ https://baseballsavant.mlb.com/leaderboard/catch_probability

⁷ https://baseballsavant.mlb.com/leaderboard/sprint_speed

approximate by nature of the player-tracking data, I used the MLB baserunning sprint maximum of 30 ft/sec as an approximation for what an MiLB player could run while attempting to catch a ball, not a value derived from the data itself. Finally, plays in which the catcher was the closest player to the ball were disregarded, since these tended to be short pop ups or fouls, which would not pass the “eye test” of being a Bermuda Triangle play.

3 Proof of Concept

As amusing as it may be, it would be in poor taste for an instant replay during a game to include blaming an individual for the ball being dropped as a tool for entertaining the crowd. Instead, a package of blooper situations over a given week or month through the whole league could be compiled and played between innings as something for attendees to look forward to and laugh about, without present players being made fun of. As a proof-of-concept animation⁸, I used SMT’s anonymized data with a newly created “Triangle Catch Probability” (TCP) metric attached to anonymous player headshots. The animation compiles plays where the ball went entirely uncaught even though multiple players were in range of the ball. A mock-up of what this could look like on a scoreboard is shown in Figure 1. Ideally, the animation would be accompanied or preceded by a video replay, and the anonymous player headshots could be replaced with roster images, so each fielder could be easily identified. The animation can be viewed here: [Triangle Tribulations](#)



Fig. 1: Scoreboard mock-up with TCP Animation visual

⁸ <https://drive.google.com/file/d/1RVtomUJrqdjUZI9We5y02as6DlzXu9bv/view?usp=sharing>

4 Identifying Triangle Plays

Within SMT's anonymized MiLB player- and ball-tracking data, each individual player was given a unique four digit code, which will be referenced as their name. In the data, the four digit code is prefaced by a three letter team code (ex. Player 1234 could be referred to as ABC-1234, simply indicating that he plays for team ABC - but no other players on other teams would be referred to as 1234, so his number is unique to him). Additionally, play-by-play data existed in the form of "ball events," which would include what happened to the ball at a given time and the player involved. For example, "pitch" would be associated with the pitcher and "ball hit into play" would be associated with the hitter. Finally, three letter team codes were used to identify home and away teams in a game, but since all teams were anonymized, there was no information available on the dimensions of the park each game was played at.

To begin, I narrowed down the data to only plays in which the ball was put into play by the batter and then either caught or dropped to bounce on the ground. Then, a "catch" variable was determined by looking at the ball event immediately following "ball hit into play." Events of "ball bounce" were categorized as "not caught" and events of "ball acquired" were categorized as "caught". In the case of a caught ball, which player caught the ball was determined by the player position associated with the "ball acquired" event. I collected the times at which the ball was hit and then either caught or first bounced on the ground, and I looked only at ball and player positions within those two timestamps. Hang time was calculated as the time between those two timestamps. Using the first two positions the ball was recorded at following the hit, I calculated an approximate exit velocity as distance the ball moved divided by the time between the recording of those two positions. Total distance traveled by the ball was based on the initial ball position at time of contact and the catch/drop position. In order to calculate launch angle, the first three positions the ball was recorded at were used. The true launch angles of the hit balls were at least as much as those calculated, since the calculation was based only on the first three recorded positions of the ball.

From the metrics I derived, I filtered my data to only include plays where the ball was launched at greater than or equal to 50 degrees, since that is what MLB's Statcast⁹ defines as being a pop-up. Plays were filtered to only hit balls where the exit velocity was calculated to be less than or equal to 123 mph, since the fastest recorded exit velocity in MLB was 122.9, so any larger values in my data were likely data collection errors.

By taking the mean of all hang times and dividing it by two, I approximated the time at which the ball would be at its peak, in an ideal and air-resistance free world. This was estimated as 2.35 seconds, and for each play, I collected the player positions and distance from the ball this long before the ball was caught or dropped. This was used so that players who were clearly not going for the ball could be ruled out of the play and not have

⁹ <https://www.mlb.com/glossary/statcast/launch-angle>.

their catch probabilities impact the analysis. Using that 2.35-second-distance, I calculated the speed at which each player on the field would have to run to reach the ball in order to catch it. An initial necessary velocity calculation based on each fielder's starting location was also calculated.

With the data filtered and all necessary metrics calculated, there were a total of 289 plays that met the definition for being a Bermuda Triangle, of which only four went entirely uncaught.

5 Triangle Catch Probability (TCP) Model

To build the Triangle Catch Probability (TCP) metric, I looked at the factors Statcast¹⁰ takes into account, which are: (1) how far the outfielder had to go (distance needed), (2) how much time did he have to get there (opportunity time), (3) what direction he needed to go in, and (4) whether proximity to the wall was a factor. While I did not have access to information on wall location, I performed an initial data analysis on the other three factors, in addition to ball metrics such as hang time, exit velocity, and distance. An additional variable indicating whether each player was a “priority” position (shortstop or center field, since they will get priority on going for hit balls that they call) was created as well. However, after looking at multicollinearity between variables, the final model included only exit velocity, player distance 2.35 seconds before the drop/catch, and if the player had to run backwards or not. An example of the data for a single play used to train the model is shown in Table 1.

Table 1: Data from one of the 237 plays used to train TCP model

Model Training Example					
Play 282 of Game y1_d185_TDW_ARN					
Exit Velo (mph)	2.35 Distance (ft)	2.35 Speed (ft/sec)	Running Backwards?	Player Position	Made Catch?
50.41	62.90	26.72	Yes	SS	No
50.41	8.25	3.51	No	LF	Yes
50.41	58.56	24.88	No	CF	No

The TCP was created using a mixed conditional logit model. That model allowed for observational unit variables, which differ for each individual on the field (position and direction to run in), as well as group variables, which are the same for every player on a given play (exit velocity, in this case). The model was built working under the assumption that the 285 uncaught hit balls can reasonably be assumed independent, since the anonymous nature of the data precludes us from accounting for individual player skill differences. The mixed conditional logit model allowed for the player- and hit-level variables and accounted for all the unchosen

¹⁰ <https://www.mlb.com/glossary/statcast/catch-probability>

categories on each play (the players who did not catch the ball). Since there were only four uncaught Bermuda Triangle observations, I excluded them from the training data so they could be used for analysis. Currently, the model output shows a sum of 100% catch probability across the players on each play, with no option for the ball being left uncaught. Ideally, with a larger data set, each play would also include a “player position 0”, or the situation in which the ball gets dropped entirely. The testing data would also then comprise of uncaught Bermuda Triangle situations, where “blame” would only be assigned to a player if he had a higher probability of catching the ball than the probability the ball went completely uncaught. Although the main limitation of the TCP model is the lack of a probability for the ball being dropped, the model would be equipped to handle that when more training data is available. A number of other factors cannot be accounted for by the model, including situations in which a player feasibly could have caught the ball but got called off by another player, who then fumbled, leading to the ball getting dropped.

6 Conclusion

With this highly customizable visualization and the TCP metric, both based on a part of baseball which every viewer can find entertaining, MiLB has an easy way to use currently available data in order to bring more enjoyment to fans. Future iterations of the TCP model can build in a predicted probability that the ball gets dropped and the possibility of assigning an error to players who had a probability of catching the ball over a certain threshold. Most importantly, in its current form the model is capable of adding entertainment value for everyone and increasing fan understanding of the game.

Acknowledgements

First and foremost, I'd like to thank Dr. Meredith Wills and Billy Fryer for their guidance, support, and assistance throughout these three months. My work would not have been possible without them, and I have learned so much from them.

For their help brainstorming, proof reading, and everything in between, I'd like to thank Conner Schupbach, Anna Thomas, and Jeevan Lasser.

For help deciding on the best model to use and how to implement it, thank you to Dr. Maria Tackett at Duke University.

Finally, I'd like to thank the entire SMT Discord Server and community for being a space to ask questions, learn from my peers, and get to know the other participants. To David Awosoga in particular, thank you for the assistance in troubleshooting the R play-by-play animation, and thank you for the base code for the animation.

Bibliography

- “Catch Probability Leaderboard.” 2026. *Baseball Savant*. https://baseballsavant.mlb.com/leaderboard/catch_probability.
- Chang, Winston. 2025. *Webshot2: Take Screenshots of Web Pages*. <https://doi.org/10.32614/CRAN.package.webshot2>.
- Elff, Martin. 2025. *Mclogit: Multinomial Logit Models for Categorical Responses and Discrete Choices*. <https://doi.org/10.32614/CRAN.package.mclogit>.
- FailArmy. 2023. “Best Sports Fails | Which Fail Wins the Playoff? - Youtube.” *YouTube*. <https://www.youtube.com/watch?v=cdmKVXynX3I>.
- Fox, John, and Sanford Weisberg. 2019. *An R Companion to Applied Regression*. Third. Thousand Oaks CA: Sage. <https://www.john-fox.ca/Companion/>.
- Garnier, Simon, Ross, Noam, Rudis, Robert, Camargo, et al. 2024. *viridis(Lite) - Colorblind-Friendly Color Maps for r*. <https://doi.org/10.5281/zenodo.4679423>.
- Heiss, Andrew. 2023. “The Ultimate Practical Guide to Multilevel Multinomial Conjoint Analysis with r.” <https://doi.org/10.59350/2mz75-rrc46>.
- Iannone, Richard, Joe Cheng, Barret Schloerke, Shannon Haughton, Ellis Hughes, Alexandra Lauer, Romain François, JooYoung Seo, Ken Brevoort, and Olivier Roy. 2026. *Gt: Easily Create Presentation-Ready Display Tables*. <https://doi.org/10.32614/CRAN.package.gt>.
- Kuhn, Max, and Hadley Wickham. 2020. *Tidymodels: A Collection of Packages for Modeling and Machine Learning Using Tidyverse Principles*. <https://www.tidymodels.org>.
- Kuhn, and Max. 2008. “Building Predictive Models in r Using the Caret Package.” *Journal of Statistical Software* 28 (5): 1–26. <https://doi.org/10.18637/jss.v028.i05>.
- “Launch Angle Glossary.” 2026. *Baseball Savant*. <https://www.mlb.com/glossary/statcast/launch-angle>.
- Mock, Thomas. 2026. *gtExtras: Extending 'Gt' for Beautiful HTML Tables*. <https://doi.org/10.32614/CRAN.package.gtExtras>.
- Ooms, Jeroen, Kornel Lesiński, and Authors of the dependency Rust crates. 2025. *Gifski: Highest Quality GIF Encoder*. <https://doi.org/10.32614/CRAN.package.gifski>.
- RAC, Coach. 2025. “Are Savannah Banana Games Scripted? #Baseball #Bananaball - YouTube.” *YouTube Shorts*. <https://www.youtube.com/shorts/tb2j1DPomFs>.
- Roback, Paul, and Julie Legler. 2021. *Beyond Multiple Linear Regression: Applied Generalized Linear Models and Multilevel Models in r*. Boca Raton: Chapman; Hall/CRC. <https://doi.org/10.1201/9780429066665>.
- Rogers, Shane. 2024. “Why Are Bloopers Reels so Funny?” *The Conversation*. <https://theconversation.com/why-are-blooper-reels-so-funny-221492>.
- Sjoberg, Daniel D., Karissa Whiting, Michael Curry, Jessica A. Lavery, and Joseph Larmarange. 2021. “Reproducible Summary Tables with the Gtsummary Package.” *The R Journal* 13: 570–80. <https://doi.org/10.32614/CRAN.package.gtsummary>.

- [//doi.org/10.32614/RJ-2021-053](https://doi.org/10.32614/RJ-2021-053).
- Sports, Baseball. 2018. “MLB | Bloopers - Oddities - YouTube.” *YouTube*. <https://www.youtube.com/watch?v=8Ket3BFas7E>.
- “Sprint Speed Leaderboard.” 2026. *Baseball Savant*. https://baseballsavant.mlb.com/leaderboard/sprint_speed.
- Videos, Sporting. 2020. “MLB Untouched Infield Popups - Youtube.” *YouTube*. <https://www.youtube.com/watch?v=yOtrjoQoSJM>.
- Wickham, Hadley. 2016. *Ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York. <https://ggplot2.tidyverse.org>.
- Wickham, Hadley, Mara Averick, Jennifer Bryan, Winston Chang, Lucy D’Agostino McGowan, Romain François, Garrett Golemund, et al. 2019. “Welcome to the tidyverse.” *Journal of Open Source Software* 4 (43): 1686. <https://doi.org/10.21105/joss.01686>.
- Wickham, Hadley, Romain François, Lionel Henry, Kirill Müller, and Davis Vaughan. 2026. *Dplyr: A Grammar of Data Manipulation*. <https://doi.org/10.32614/CRAN.package.dplyr>.
- Xie, Yihui. 2014. “Knitr: A Comprehensive Tool for Reproducible Research in R.” In *Implementing Reproducible Computational Research*, edited by Victoria Stodden, Friedrich Leisch, and Roger D. Peng. Chapman; Hall/CRC.
- Yu, Guangchuang. 2026. *Ggimage: Use Image in 'Ggplot2'*. <https://doi.org/10.32614/CRAN.package.ggimage>.

Appendix A: Analysis of Uncaught Bermuda Triangles

Of the four uncaught plays flagged as Bermuda Triangles in the data, three are truly representative of the problem we are trying to solve, and the fourth (shown last) seems to be an example of an infield fly rule.

Table 2: Variables and catch probability for play 169 of y1_d068_SQF_ARN

Game y1_d068_SQF_ARN							
Play 169							
Exit Velo (mph)	2.35 Distance (ft)	2.35 Speed (ft/sec)	Running Backwards?	Position	Unique ID	Catch Probability (%)	
42.63	69.44	29.50	Yes	2B	9323	1.07	
42.63	55.67	23.65	Yes	SS	1503	11.25	
42.63	58.33	24.78	No	RF	3382	87.68	

On the play detailed in Table and Figure 2, right fielder, 3382, makes a sprint towards the ball here while the second baseman and shortstop seem to jog. The ball does not land right in the middle of them, instead being closest to 3382 who has an 87.68% probability to catch the ball. In this situation, the blame would almost certainly fall on him, even without the TCP results. This could have been a recorded error in the scoring, though we do not know because of the anonymization.

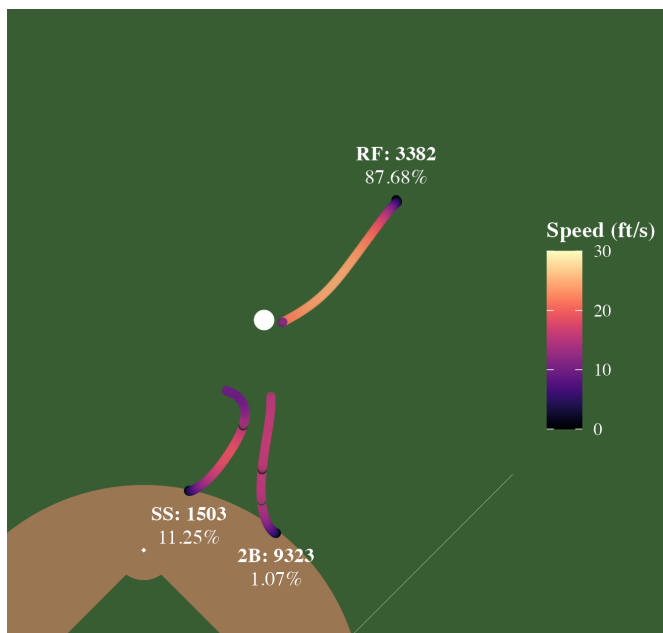


Fig. 2: Speeds of involved players for play 169 of y1_d068_SQF_ARN

Table 3: Variables and catch probability for play 113 of y1_d074_AIP_ARN

Game y1_d074_AIP_ARN						
Play 113						
Exit Velo (mph)	2.35 Distance (ft)	2.35 Speed (ft/sec)	Running Backwards?	Position	Unique ID	Catch Probability (%)
44.96	18.79	7.98	Yes	Pitcher	6151	1.65
44.96	49.83	21.17	Yes	Catcher	2385	0.05
44.96	31.89	13.55	No	1B	3323	44.26
44.96	49.32	20.95	No	2B	1650	4.49
44.96	33.45	14.21	No	3B	2415	49.55

The play in Table and Figure 3 looks more like a typical Bermuda Triangle, where three players (the 1st, 2nd, and 3rd basemen) all run from their starting positions towards the ball, none of them needing to do a full sprint. While third baseman 2415 lays off and slows down the earliest, he is also the player who the TCP model flags as having the highest likelihood of catching the ball. In this situation, the model may conflict with a viewer's opinion, as the ball was on the 1st base side of the mound. Future iterations of the model could possibly include a predictor variable based on which side of the mound the ball lands

on. Finally, the model predicts the pitcher as having only a 1.65% chance of catching the ball, even though he is the closest fielder. This confirms that the model is predicting based on player position as well, since pitchers tend to be less likely to catch the ball even when they are close to the pop up location.

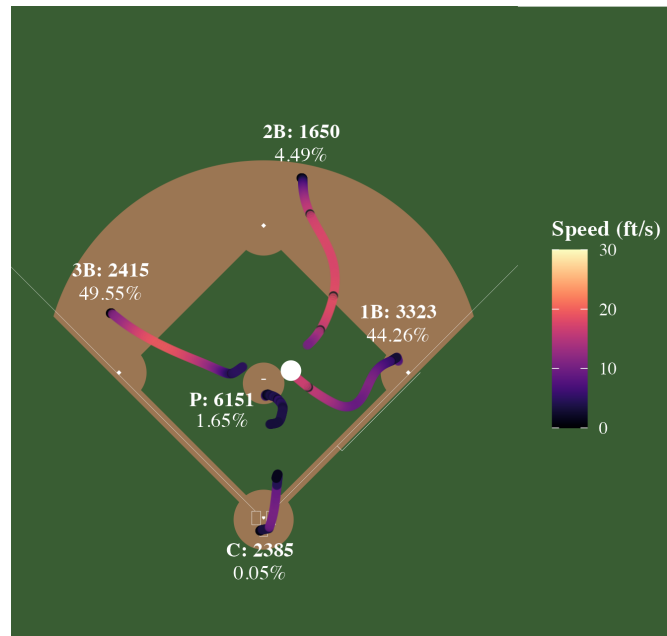


Fig. 3: Speeds of involved players for play 113 of y1_d074_AIP_ARN

Table 4: Variables and catch probability for play 56 of y1_d122_MSE_ANI

Game y1_d122_MSE_ANI						
Play 56						
Exit Velo (mph)	2.35 Distance (ft)	2.35 Speed (ft/sec)	Running Backwards?	Position	Unique ID	Catch Probability (%)
70.97	55.58	23.61	Yes	3B	1872	1.59
70.97	48.59	20.65	Yes	SS	3335	8.22
70.97	55.06	23.39	No	CF	1669	90.19

In this typical Bermuda-Triangle-type play shown in Table and Figure 4, three players all race towards the ball, with the center fielder going the fastest and the second baseman speeding up as he gets closer. In the end, the ball falls closest to the center fielder, but each player has slowed up right before reaching the ball and the ball bounces on the ground. The center fielder was still the closest to the ball and most likely to catch it, with a probability of 90.19%, so blame would be assigned to him. In a future iteration of the model where “no catch” is assigned a probability as well, the center fielder would likely have a lower catch probability considering the distance and speed he had to run at.

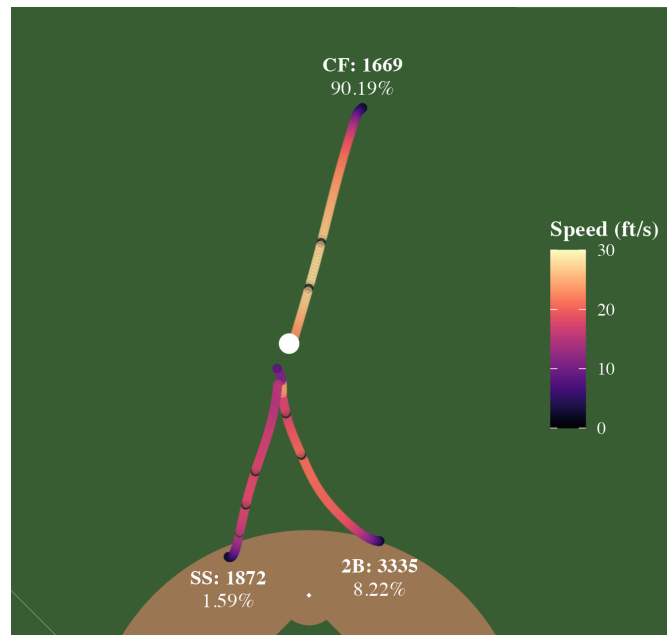


Fig. 4: Speeds of involved players for play 56 of y1_d122_MSE_ANI

Table 5: Variables and catch probability for play 128 of y1_d138_QBR_ANI

Game y1_d138_QBR_ANI						
Play 128						
Exit Velo (mph)	2.35 Distance (ft)	2.35 Speed (ft/sec)	Running Backwards?	Position	Unique ID	Catch Probability (%)
53.74	59.61	25.33	Yes	1B	2596	0.57
53.74	33.09	14.06	Yes	2B	3725	98.81
53.74	63.93	27.16	Yes	SS	2218	0.62

In the play shown in Table and Figure 5, you can see that the player locations do not follow a triangle set-up at all. The short stop is clearly jogging comfortably to cover 2nd base, while the first baseman essentially stays still. Additionally, the second baseman has a 98.81% probability of catching the ball, higher than any other catch probabilities for uncaught Bermuda Triangles. Upon viewing the player line-ups and ball locations data, this play seems to have been an example of an infield-fly rule, since there were runners at 1st and 2nd base who did not make an attempt to advance even after the ball was dropped by the second baseman. Additionally, the next pitch is thrown to the following batter in the hitting order, without the initial batter having attempted to run to first base.

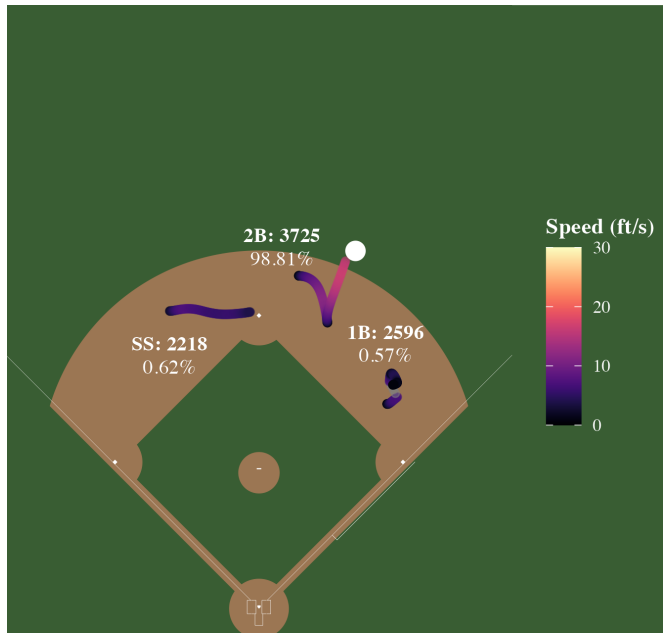


Fig. 5: Speeds of involved players for play 128 of y1_d138_QBR_ANI

Appendix B: Model Evaluation

Table 6: Output of mixed conditional logit model, showing odds ratios. Significant p-values are highlighted grey

Triangle Catch Probability Model			
Characteristic	Odds Ratio	95% CI	p-value
2.35 Sec. Distance	0.85	0.81, 0.88	<0.001
Fielder Running Backwards?	0.31	0.11, 0.83	0.020
Player Position * Exit Velocity in MPH (EV)			
Pitcher * EV	0.96	0.93, 1.00	0.038
1B * EV	1.06	1.03, 1.09	<0.001
2B * EV	1.07	1.04, 1.11	<0.001
3B * EV	1.07	1.04, 1.10	<0.001
SS * EV	1.08	1.04, 1.11	<0.001
LF * EV	1.10	1.05, 1.15	<0.001
CF * EV	1.11	1.06, 1.17	<0.001
RF * EV	1.11	1.06, 1.16	<0.001
Abbreviation: CI = Confidence Interval			

The model found all player-level variables as well as exit velocity for all positions to be significant at the 0.05 level, as shown in Table 6. The significant variables do not have confidence intervals that cross the 1.0 threshold. An interpretation of distance 2.35 seconds prior to catch/drop would be that for every one foot increase in the player's distance from where the ball ends up, that player's odds of catching the ball decrease by 15%, holding all else constant. For players who must run backwards, their odds of catching the ball decrease by 69% compared to a player who has to run forwards to catch the ball, holding all else constant. The baseline level for the player position variable was the catcher, so odds ratios for each player position as exit velocity increases by one mile per hour are in comparison to the catcher. For example, if exit velocity increases by one mile per hour, the pitcher has 4% lower odds of catching the ball as compared to the catcher. For all other positions, an increase in exit velocity was correlated with an increase in the odds of catching the ball compared to the catcher and holding all else constant, to greater degrees for outfielders than infielders.

The in-depth reasoning for using a mixed conditional logit model over any other models is as follows: a logit model allows for predicting the probability of a binary outcome (whether a given player catches the ball or not), a conditional logit model allows for modeling the probability of choosing between various alternatives (which of the given players catches the ball), and a mixed conditional logit model incorporates random effects

(coefficients to represent “preference” or “taste”, in this case, the metrics of the hit ball itself). An example for when mixed conditional logit models are used is in consumer economics: accounting for a person’s tastes, which car will they choose to purchase, given that each car has a different price, color, etc. The hit’s speed, distance, and launch angle are the equivalent of a person’s taste, and the different cars’ prices, colors, and more are the equivalent of the player’s running backwards and distance from the ball.

There are several assumptions required in using a mixed conditional logit, particularly that the 285 hit balls can be assumed independent. This is not necessarily the case, since one same defensive player could have better abilities or speed, which could not be accounted for due to the nature of the anonymization of the data, this could not be incorporated into the model.

Table 7: (a) VIF for variables in full, initial model with high GVIFs highlighted red, (b) VIF for final model

Initial Model Variance Inflation Factor			
Predictor	GVIF	Df	Adjusted GVIF
2.35 Sec. Distance	1.35	1	1.16
Priority Position	2.90×10^2	1	17.03
Backwards?	3.39	1	1.84
Player ID	1.30×10^3	36	1.10
Exit Velocity	7.60×10^{10}	9	4.02
Launch Angle	6.19×10^{15}	9	7.54
Distance	1.05×10^{11}	9	4.09
Hang Time	1.31×10^{18}	9	10.15
GVIF: Generalized Variance Inflation Factor Df: Degrees of Freedom			

Final Model Variance Inflation Factor			
Predictor	GVIF	Df	Adjusted GVIF
2.35 Sec. Distance	1.09	1	1.04
Backwards?	1.25	1	1.12
Player ID	1.35	9	1.02
Exit Velocity	1.35	9	1.02
GVIF: Generalized Variance Inflation Factor Df: Degrees of Freedom			

Two main iterations of this model were created, the first including priority positions, launch angle, distance, and hang time as variables and the second excluding them. Both models were analyzed for multicollinearity using the metric of variance inflation factor, or VIF. Specifically, general variance inflation factor, or GVIF, was used since the model included categorical variables. GVIF was adjusted using the equation $GVIF^{1/(2 \cdot Df)}$ to account for differing degrees of freedom (Df) across the predictors. The first model had issues of high multicollinearity, as shown in Table 7a, meaning that the variables that are supposed to be independent are actually highly correlated with each other, inflating the standard errors in the regression model. The second model is the one included as the output, which shows in Table 7b Adjusted GVIF values

in the acceptable range of under 2.2.

Table 8: (a) Metrics from test classifications of model, (b) confusion matrix with predicted and actual classifications

Evaluation of Test Classifications		Testing Data Confusion Matrix		
For Mixed Conditional Logit Model		For Mixed Conditional Logit Model		
Metric	Value	Actual (Reference) Values		
Sensitivity	0.963	Predicted Values	No Catch	Catch
Specificity	0.900	No Catch	183	8
Positive Prediction Value	0.958	Catch	7	72
Negative Prediction Value	0.911			
Accuracy	0.944			
Kappa	0.866			
Accuracy Null	0.704			
Accuracy p-Value	0.000			

By evaluating the final model using the testing data set, we can see that the model accurately classified 255 player-specific outcomes (catch or no catch) while only incorrectly predicting 15, as shown in Table 8b. This accuracy rate of 94.4% indicates a strong model performance without being so high as to cause concern for over-fitting of the data. As shown in Table 8a, the sensitivity, specificity, positive prediction value, and negative prediction value all being 0.9 or greater indicate strong model performance as well. The kappa value of 0.866, which is on the scale from -1 to 1, also indicates a strong level of prediction as compared to randomly deciding, and the accuracy null value of 0.704 indicates that if the model were to have just predicted based on the most common outcome (“no catch”), it would have been right 70.4% of the time. The accuracy p-value being near 0 indicates that the model is significantly better than predicting each player outcome as being “no catch”.

Appendix C: R Code

Data Cleaning

Packages and Loading Data

```
library(dplyr)
library(tidyverse)
library(ggplot2)
library(gifski)
library(gt)
library(gtExtras)

# substitute necessary working directory here as needed
source(
  "../Code/SMT_Data_Starter.R"
)

ball_events_df <- ball_events |>
  collect() |>
  # create column with unique classifier for each game's individual plays
  mutate(game_str_ppg = paste(game_string, play_per_game, sep = "_"))

# BBIP (batted balls in play) occurrences
bbip_plays <- ball_events_df |>
  # remove observations missing game strings
  filter(!is.na(game_string)) |>
  # filter to only when ball was hit and bounced or got caught
  filter(ball_eventcode == 4 &
    (lead(ball_eventcode == 2) |
     lead(ball_eventcode == 16))) |>
  pull(game_str_ppg)

# find duplicates of same event on unique play and game combination
dup_plays <- ball_events_df |>
  # remove timestamp to track repeats of the same events on a play
  dplyr::select(-timestamp) |>
  group_by_all() |>
  # do not filter out multiple bounces or deflections on the same play
  filter((n() > 1 &
    ball_eventcode != 16 &
    ball_eventcode != 10 &
    ball_eventcode != 9)) |>
  ungroup() |>
  pull(game_str_ppg)

# compile data for balls hit into play but NOT duplicated
int_ball_events_df <- ball_events_df |>
  filter(game_str_ppg %in% bbip_plays &
```

```

      !(game_str_ppg %in% dup_plays))

# create vector of all BBIP plays (not duplicates)

hit_plays <- int_ball_events_df |>
  pull(game_str_ppg)

# loading game lineups
play_lineups <- lineups |>
  mutate(game_str_ppg = paste(game_string, play_per_game, sep = "_")) |>
  filter(game_str_ppg %in% hit_plays)

# loading ball positions
ball_pos_df <- ball_positions |>
  mutate(game_str_ppg = paste(game_string, play_per_game, sep = "_"),
         timestamp = as.numeric(timestamp),
         play_per_game = as.numeric(play_per_game)) |>
  filter(game_str_ppg %in% hit_plays) |>
  collect()

# if ball was caught (did not bounce), mark "1" in "catch" column

catch_plays <- int_ball_events_df |>
  mutate(catch = if_else((lag(ball_eventcode) == 4 & ball_eventcode == 2),
                        1, 0)) |>
  filter(catch == 1) |>
  # select the unique play per game and position of player who made catch
  dplyr::select(game_str_ppg, catch, player_id)

catch_plays_str <- catch_plays |> pull(game_str_ppg)

```

Ball Hit & Drop Times

```

# ball hit time & location ----
ball_hit_time <- int_ball_events_df |>
  # only the rows recording the hit
  filter(ball_eventcode == 4) |>
  mutate(timestamp = as.numeric(timestamp)) |>
  dplyr::select(game_str_ppg,
               hit_timestamp = timestamp, game_string, play_per_game)

# find position of ball at time of hit
ball_pos_hit_time <- ball_hit_time |>
  left_join(ball_pos_df,
           by = join_by(hit_timestamp == timestamp,
                        game_str_ppg, game_string, play_per_game)) |>
  # rename position at time of hit as "contact" x, y, or z
  dplyr::select(game_string, play_per_game, game_str_ppg,
               contact_x = ball_position_x,
               contact_y = ball_position_y,

```

```

        contact_z = ball_position_z,
        hit_timestamp)

# ball end event time & location ----
ball_drop_time <- int_ball_events_df |>
  # only include rows immediately after ball being hit
  filter(lag(ball_eventcode) == 4) |>
  mutate(timestamp = as.numeric(timestamp)) |>
  dplyr::select(game_str_ppg,
    drop_timestamp = timestamp, game_string, play_per_game)

# find closest timestamp to event after the hit (catch or drop)
ball_drop_pos <- ball_drop_time |>
  left_join(ball_pos_df,
    by = join_by(game_str_ppg, game_string, play_per_game)) |>
  dplyr::select(drop_timestamp, timestamp, game_string,
    play_per_game, game_str_ppg,
    landed_x = ball_position_x,
    landed_y = ball_position_y,
    landed_z = ball_position_z) |>
  group_by(game_str_ppg) |>
  # find time of ball position most similar to time dropped/caught
  mutate(near_timestamp =
    timestamp[which.min(abs(timestamp - drop_timestamp))]) |>
  filter(near_timestamp == timestamp) |>
  ungroup() |>
  dplyr::select(game_string, play_per_game, game_str_ppg,
    landed_x, landed_y, landed_z, drop_timestamp = timestamp)

```

Ball Positions

```

# find tracked ball positions between when ball was hit and caught/dropped
ball_pos_after_contact <- ball_pos_df |>
  mutate(play_per_game = as.numeric(play_per_game)) |>
  # add time ball was hit as a column
  left_join(ball_hit_time,
    by = join_by(game_string, play_per_game, game_str_ppg)) |>
  # add time ball was dropped/caught as a column
  left_join(ball_drop_time,
    by = join_by(game_string, play_per_game, game_str_ppg)) |>
  filter(timestamp >= hit_timestamp & timestamp <= drop_timestamp)

# take first 3 tracked ball positions on each play
ball_pos_first_3 <- ball_pos_after_contact |>
  arrange(game_string, play_per_game, timestamp) |>
  group_by(game_str_ppg) |>
  slice_head(n = 3) |>
  mutate(row_num = row_number()) |>
  ungroup() |>
  dplyr::select(game_string, play_per_game, game_str_ppg,
    timestamp, starts_with("ball_position"), row_num)

```

```
# find first position of the ball (position at time of hit)
ball_pos_hit_1 <- ball_pos_first_3 |>
  filter(row_num == 1) |>
  dplyr::select(game_string : game_str_ppg,
    time_1 = timestamp,
    x_1 = ball_position_x,
    y_1 = ball_position_y,
    z_1 = ball_position_z)

# find second position of the ball (position immediately after hit)
ball_pos_hit_2 <- ball_pos_first_3 |>
  filter(row_num == 2) |>
  dplyr::select(game_string : game_str_ppg,
    time_2 = timestamp,
    x_2 = ball_position_x,
    y_2 = ball_position_y,
    z_2 = ball_position_z)

# create data frame with 1st and 2nd locations in one row for each ball
ball_pos_first_2 <- full_join(ball_pos_hit_1,
  ball_pos_hit_2,
  by = join_by(game_string, play_per_game,
    game_str_ppg)) |>
  mutate(across(4:11, as.numeric))
```

Calculating Ball Hit Distance, Angle, and Velocity

```
# Hit Distances ----

# join tables with ball's start and end positions
ball_distances <- left_join(ball_pos_hit_time, ball_drop_pos,
  by = join_by(game_string, play_per_game,
    game_str_ppg)) |>

# calculate total distance the ball traveled (in feet)
mutate(across(4:11, as.numeric),
  change_x = landed_x - contact_x,
  change_y = landed_y - contact_y,
  change_z = landed_z - contact_z,
  distance = sqrt(change_x^2+change_y^2+change_z^2),
  hangtime = drop_timestamp - hit_timestamp)

# Launch Angle ----

# calculate the angle at which the ball was launched
launch_angles_pop <- ball_pos_first_3 |>
  mutate(across(5:7, as.numeric)) |>
  arrange(game_string, play_per_game, timestamp) |>
  group_by(game_str_ppg) |>
  # find minimum and maximum x, y, and z ball positions within the first 3
  # observations in order to calculate the change in ball position
  mutate(change_x = max(ball_position_x) - min(ball_position_x),
    change_y = max(ball_position_y) - min(ball_position_y),
```

```

    change_z = max(ball_position_z) - min(ball_position_z),
    # calculate distance moved over the ground
    ground_distance = sqrt(change_x ^2 + change_y ^2),
    # calculate the radians of the launch angle
    launch_angle_rad = atan(change_z / ground_distance),
    # convert radians to degrees
    launch_angle_deg = launch_angle_rad * 180 / pi) |>
ungroup() |>
dplyr::select(game_str_ppg, launch_angle_deg) |>
distinct() |>
# remove observations that did not have a measurable launch angle
filter(!is.na(launch_angle_deg)) |>
# only want pop ups (launched at 50 degrees or more)
filter(launch_angle_deg >= 50)

# Exit Velocity ----

# calculate the exit velocity the ball had
exit_velos <- ball_pos_first_2 |>
  # find change in ball position and time between the first two observations
  mutate(change_x = x_2 - x_1,
         change_y = y_2 - y_1,
         change_z = z_2 - z_1,
         change_time = time_2 - time_1,
         # calculate distance and exit velocity
         dist = sqrt(change_x^2 + change_y^2 + change_z^2),
         exit_velo_ft_ms = dist/change_time,
         # convert exit velocity to mph from ft/ms
         exit_velo_mph = exit_velo_ft_ms / 5280 * (1000*60*60)) |>
dplyr::select(game_string, play_per_game, game_str_ppg, exit_velo_mph)

```

Combined Ball Metrics Data Frame

```

# create data frame combining the distances, hang times, and launch angles
hit_metrics <- launch_angles_pop |>
  left_join(ball_distances,
           by = join_by(game_str_ppg)) |>
  left_join(exit_velos,
           by = join_by(game_string, play_per_game, game_str_ppg)) |>
# filter since balls have not been hit in the MLB faster than 122.9 mph
filter(exit_velo_mph <= 123)

hit_metrics_9 <- hit_metrics |>
  # create nine rows for each hit ball, one for each fielder
  group_by(game_str_ppg) |>
  uncount(9) |>
  ungroup() |>
  mutate(player_id = rep(seq(9), (nrow(hit_metrics)))) |>
  # join earlier data frame marking which fielder caught the ball
  left_join(catch_plays, by = join_by(game_str_ppg, player_id)) |>
  # mark "1" in "catch" for player who caught it, "0" for all others
  mutate(catch = if_else(!is.na(catch), 1, 0),

```

```

# convert number to named position
player_id = case_when(
  player_id == 1 ~ "Pitcher",
  player_id == 2 ~ "Catcher",
  player_id == 3 ~ "1B",
  player_id == 4 ~ "2B",
  player_id == 5 ~ "3B",
  player_id == 6 ~ "SS",
  player_id == 7 ~ "LF",
  player_id == 8 ~ "CF",
  player_id == 9 ~ "RF"),
player_id = as.factor(player_id))

# pull the game strings for all balls that were pop ups
pop_up_plays <- hit_metrics_9 |>
  pull(game_str_ppg)

# find the mid-point time of the average hangtime of popups
middle_avg_hang_time <- mean(hit_metrics$hangtime, na.rm = TRUE)/2

# find time on each play that is 2.35 sec. prior to ball drop/catch

ball_time_mid <- ball_drop_time |>
  mutate(time_mid = drop_timestamp - middle_avg_hangtime) |>
  dplyr::select(-drop_timestamp)

```

Fielder Positions

```

# find player positions at all times for identified pop-up plays
player_positions_df <- player_positions |>
  mutate(game_str_ppg = paste(game_string, play_per_game, sep = "_"),
         player_id = as.numeric(player_id),
         play_per_game = as.numeric(play_per_game),
         timestamp = as.numeric(timestamp)) |>
  filter(game_str_ppg %in% pop_up_plays) |>
  dplyr::select(-c(home_team, away_team, year, day)) |>
  collect() |>
  # only defense, remove observations where location not measured
  filter(player_id <= 9,
         !is.na(field_x) &
         !is.na(field_y) &
         field_x != "NA" &
         field_y != "NA") |>
  # add position names to player locations
  mutate(player_id = case_when(
    player_id == 1 ~ "Pitcher",
    player_id == 2 ~ "Catcher",
    player_id == 3 ~ "1B",
    player_id == 4 ~ "2B",
    player_id == 5 ~ "3B",
    player_id == 6 ~ "SS",
    player_id == 7 ~ "LF",

```

```

    player_id == 8 ~ "CF",
    player_id == 9 ~ "RF"),
    player_id = as.factor(player_id))

# find player positions at the moment the ball is hit
init_player_positions <- ball_hit_time |>
  left_join(player_positions_df,
            by = join_by(game_str_ppg, game_string,
                          play_per_game, hit_timestamp == timestamp)) |>
# rename these positions as their initial positions
  rename(init_field_x = field_x,
          init_field_y = field_y) |>
  dplyr::select(-hit_timestamp)

# remove repeats recordings of player location on a play
unique_player_positions <- player_positions_df |>
  dplyr::select(game_str_ppg, timestamp) |>
  unique()

# find time of player's position closest to 2.35 sec. before catch/drop
mid_player_positions <- ball_time_mid |>
  left_join(unique_player_positions, by = join_by(game_str_ppg)) |>
  mutate(timestamp = as.numeric(timestamp),
          time_mid = as.numeric(time_mid)) |>
  filter(!is.na(timestamp)) |>
  group_by(game_str_ppg, game_string, play_per_game) |>
  # add time closest to 2.35 sec. before catch/drop
  mutate(int_timestamp = timestamp[which.min(abs(timestamp - time_mid))]) |>
  # only include player positions at that timestamp of interest
  filter(timestamp == int_timestamp) |>
  ungroup() |>
  dplyr::select(-time_mid, -int_timestamp) |>
  left_join(player_positions_df,
            by = join_by(game_string, play_per_game,
                          game_str_ppg, timestamp)) |>
  dplyr::select(-timestamp)

```

Combined Fielder & Ball Metrics Data Frame

```

# merge the data frames with ball metrics and fielder locations
hit_and_fielder_data <- hit_metrics_9 |>
  left_join(init_player_positions,
            by = join_by(game_string, play_per_game,
                          game_str_ppg, player_id)) |>
  filter(!is.na(init_field_x) &
         !is.na(init_field_y) &
         init_field_x != "NA" &
         init_field_y != "NA") |>
# determine distance between fielder and ball's catch/drop location
  mutate(across(c(5:18, 21, 22), as.numeric),
          init_dist =
            sqrt((landed_x - init_field_x)^2 + (landed_y - init_field_y)^2)

```

```

) |>
# include player positions 2.35 seconds before ball was caught/dropped
left_join(mid_player_positions,
          by = join_by(game_str_ppg, game_string,
                       play_per_game, player_id)) |>
rename(mid_field_x = field_x,
       mid_field_y = field_y) |>
filter(!is.na(mid_field_x) &
       !is.na(mid_field_y) &
       mid_field_x != "NA" &
       mid_field_y != "NA") |>
mutate(across(24:25, as.numeric),
       # determine if they had to run backwards to reach ball
       backwards = if_else(landed_y > init_field_y, 1, 0),
       # necessary speed to reach ball total & 2.35 seconds out
       mid_dist =
         sqrt((landed_x - mid_field_x)^2 + (landed_y - mid_field_y)^2),
       init_nec_speed = init_dist / (hangtime / 1000),
       mid_nec_speed = mid_dist / (middle_avg_hangtime / 1000)
)

```

Bermuda Triangle Plays

```

# find closest player to ball at 2.35 sec. out, not including catcher
not_catcher_str <- hit_and_fielder_data |>
  group_by(game_str_ppg) |>
  filter(mid_dist == min(mid_dist)) |>
  ungroup() |>
  filter(player_id != 2) |>
# pull game strings of these plays
pull(game_str_ppg)

# strings for the plays in which 3+ players could reach ball
triangle_play_str <- hit_and_fielder_data |>
  filter(game_str_ppg %in% not_catcher_str) |>
  filter(mid_dist <= 75 &
         init_nec_speed <= 25 &
         mid_nec_speed <= 30) |>
# count how many players match the above criteria for each play
count(game_str_ppg) |>
# include only plays where three or more players match the criteria
filter(n >= 3) |>
pull(game_str_ppg)

# data for the players in the above found triangle play strings
triangle_df <- hit_and_fielder_data |>
  # filter to only the players in range on each caught triangle play
  filter(game_str_ppg %in% triangle_play_str &
         game_str_ppg %in% catch_plays_str &
         mid_dist <= 75 &
         init_nec_speed <= 25 &

```

```

        mid_nec_speed <= 30) |>
# keep only necessary variables
dplyr::select(game_string, play_per_game, game_str_ppg,
              distance, launch_angle_deg, exit_velo_mph,
              hangtime, init_dist, init_nec_speed,
              mid_dist, mid_nec_speed, backwards,
              player_id, catch) |>
# mark cf and ss as "priority" positions
mutate(game_str_ppg = as.factor(game_str_ppg),
       priority =
         as.factor(if_else(player_id %in% c("SS", "CF"), "Yes", "No")),
       backwards =
         as.factor(if_else(backwards == 1, "Yes", "No")),
       player_id = factor(player_id,
                           levels = c("Pitcher", "1B", "2B",
                                       "3B", "SS", "LF", "CF",
                                       "RF", "Catcher")))

# save the plays for the Bermuda Triangle situations that did get caught
caught_tri_str <- triangle_df |>
mutate(game_str_ppg = as.character(game_str_ppg)) |>
distinct(game_str_ppg) |>
dplyr::select(game_str_ppg)

# save player data for the Bermuda Triangle situations NOT caught
uncaught_triangle_df <- hit_and_fielder_data |>
# include only the players on each uncaught play in range
filter(game_str_ppg %in% triangle_play_str &
       !(game_str_ppg %in% catch_plays_str) &
       mid_dist <= 75 &
       init_nec_speed <= 25 &
       mid_nec_speed <= 30) |>
dplyr::select(game_string, play_per_game, game_str_ppg,
              distance, launch_angle_deg, exit_velo_mph,
              hangtime, init_dist, init_nec_speed,
              mid_dist, mid_nec_speed, backwards,
              player_id, catch) |>
# mark cf and ss as the "priority" positions
mutate(game_str_ppg = as.factor(game_str_ppg),
       priority =
         as.factor(if_else(player_id %in% c("SS", "CF"), "Yes", "No")),
       backwards =
         as.factor(if_else(backwards == 1, "Yes", "No")),
       player_id =
         factor(player_id, levels = levels(triangle_df$player_id)))

# save the plays for the Bermuda Triangle situations NOT caught
uncaught_tri_plays <- uncaught_triangle_df |>
distinct(game_str_ppg) |>
pull(game_str_ppg)

```

Model Creation

Catch Probability Model

```
library(mclgit)
library(tidymodels)
library(knitr)
library(car)
library(caret)
library(gtsummary)

# set seed for reproducibility
set.seed(062026)

# standard testing/training data split of plays
triangle_str_split <- initial_split(caught_tri_str)

triangle_str_train <- training(triangle_str_split) |>
  pull(game_str_ppg)
triangle_str_test <- testing(triangle_str_split) |>
  pull(game_str_ppg)

# use the split plays to filter for test/train data
triangle_train <- triangle_df |>
  filter(game_str_ppg %in% triangle_str_train)
triangle_test <- triangle_df |>
  filter(game_str_ppg %in% triangle_str_test)

# final mclgit model creation
model_triangle_mclgit <- mclgit(
  cbind(catch, game_str_ppg) ~ mid_dist + backwards +
    player_id:exit_velo_mph,
  data = triangle_train)

# mclgit regression model output

tbl_regression(model_triangle_mclgit, exponentiate = TRUE,
  show_single_row = c(backwards),
  label = list(
    `mid_dist` = "2.35 Sec. Distance",
    priority = "Priority Position?",
    backwards = "Fielder Running Backwards?",
    `player_id:exit_velo_mph` =
      "Player Position * Exit Velocity in MPH (EV)"
  )) |>
as_gt() |>
cols_label(estimate = "Odds Ratio") |>
tab_header(
  title = "Triangle Catch Probability Model"
) |>
```

```

tab_style(
  style = cell_fill(color = "lightgrey"),
  locations = cells_body(
    rows = (p.value < 0.05),
    columns = p.value)
) |>
text_replace(pattern = "exit_velo_mph", replacement = "EV") |>
tab_style(style = cell_borders(
  sides = "bottom",
  color = "darkgrey",
  weight = px(2)),
locations = cells_body(rows = 3)
) |>
tab_options(
  table.width = pct(60),
  row_group.font.weight = "bold",
  column_labels.font.weight = "bold",
  table.font.names = "serif"
) |>
gtsave("../tcp_summary.png")

```

Model Analysis

```

# model testing for fit, using testing data set
test_predictions <- predict(model_triangle_mclgit,
                             newdata = triangle_test, type = "response")

# data frame with predictions and actual outcomes
augmented_test <- cbind(triangle_test, as.data.frame(test_predictions)) |>
  # cbind is used here since no additional steps are being done on frame
  # in future, a full_join or left_join may better ensure reproducibility
  mutate(probability = format(test_predictions, scientific = FALSE),
         test_predictions = as.factor(round(test_predictions)),
         catch = as.factor(catch))

# confusion matrix for final model analysis
cm <- confusionMatrix(data = augmented_test$test_predictions,
                      reference = augmented_test$catch)

# pull class metrics out of confusion matrix
byclass <- cm$byClass |>
  as.data.frame() |>
  rename(Value = `cm$byClass`) |>
  rownames_to_column(var = "Metric") |>
  slice(c(1:4))

# pull overall metrics out of confusion matrix
overall <- cm$overall |>
  as.data.frame() |>
  rename(Value = `cm$overall`) |>
  rownames_to_column(var = "Metric") |>
  slice(c(1,2,5:6))

```

```

# create readable view of metrics from confusion matrix
cm_view <- rbind(byclass, overall) |>
gt() |>
tab_header(title = "Evaluation of Test Classifications",
           subtitle = "For Mixed Conditional Logit Model") |>
tab_options(
  row_group.font.weight = "bold",
  column_labels.font.weight = "bold",
  table.font.names = "serif"
) |>
fmt_number(
  columns = Value,
  decimals = 3
) |>
text_replace(pattern = "Pos", replacement = "Positive") |>
text_replace(pattern = "Neg", replacement = "Negative") |>
text_replace(pattern = "Pred", replacement = "Prediction") |>
text_replace(pattern = "McNemar", replacement = "McNemar ") |>
text_replace(pattern = "Accuracy", replacement = "Accuracy ") |>
text_replace(pattern = "PValue", replacement = "p-Value") |>
gtsave("../cm-view.png")

# show actual confusion matrix table
cm_table <- cm$table |>
as.data.frame.matrix() |>
rename("No Catch" = "0",
       "Catch" = "1") |>
rownames_to_column(var = "Predicted Values") |>
mutate("Predicted Values" = case_when(
  `Predicted Values` == "1" ~ "Catch",
  `Predicted Values` == "0" ~ "No Catch"
)) |>
gt() |>
tab_header(
  title = "Testing Data Confusion Matrix",
  subtitle = "For Mixed Conditional Logit Model"
) |>
tab_spanner(
  label = "Actual (Reference) Values",
  columns = c(2:3)
) |>
tab_options(
  row_group.font.weight = "bold",
  column_labels.font.weight = "bold",
  table.font.names = "serif"
) |>
tab_style(
  style = cell_text(weight = "bold"),
  locations = cells_body(columns = "Predicted Values")
) |>
gtsave("../cm-table.png")

```

```

knitr::include_graphics("../cm-table.png")

# initial variables as a linear model for VIF calculations
init_model <- lm(catch ~ mid_dist + priority + backwards +
                 player_id:exit_velo_mph +
                 player_id:launch_angle_deg +
                 player_id:distance +
                 player_id:hangtime,
                 data = triangle_train)

# VIF table for initial model variables
initial_vif <- vif(init_model, type = "predictor") |>
  as.data.frame() |>
  rownames_to_column(var = "Predictor") |>
  rename(
    GVIF = `GVIF`,
    Df = `Df`,
    `Adjusted GVIF` = `GVIF^(1/(2*Df))`
  ) |>
  mutate(Predictor = case_when(
    Predictor == "mid_dist" ~ "2.35 Sec. Distance",
    Predictor == "priority" ~ "Priority Position",
    Predictor == "backwards" ~ "Backwards?",
    Predictor == "player_id" ~ "Player ID",
    Predictor == "distance" ~ "Distance",
    Predictor == "launch_angle_deg" ~ "Launch Angle",
    Predictor == "exit_velo_mph" ~ "Exit Velocity",
    Predictor == "hangtime" ~ "Hang Time")) |>

select(1:4) |>
gt() |>
fmt_number(columns = c(2,4), decimals = 2) |>
tab_header(title = "Initial Model Variance Inflation Factor") |>
fmt_scientific(columns = (GVIF)) |>
tab_options(
  row_group.font.weight = "bold",
  column_labels.font.weight = "bold",
  table.font.names = "serif"
) |>
tab_style(
  style = cell_fill(color = "#fbcbc7"),
  locations = cells_body(
    rows = (`Adjusted GVIF` > 2.2),
    columns = `Adjusted GVIF`
  )
) |>
tab_footnote(footnote =
  html("GVIF: Generalized Variance Inflation Factor<br>
        Df: Degrees of Freedom")) |>
gtsave("../initial_tcp_vif.png")

```

```

# final variables as linear model for VIF calculations
final_model <- lm(catch ~ mid_dist + backwards +
                  player_id:exit_velo_mph,
                  data = triangle_train)

# VIF table for final model variables
final_vif <- vif(final_model, type = "predictor") |>
  as.data.frame() |>
  rownames_to_column(var = "Predictor") |>
  rename(
    GVIF = `GVIF`,
    Df = `Df`,
    `Adjusted GVIF` = `GVIF^(1/(2*Df))`
  ) |>
  mutate(Predictor = case_when(
    Predictor == "mid_dist" ~ "2.35 Sec. Distance",
    Predictor == "priority" ~ "Priority Position",
    Predictor == "backwards" ~ "Backwards?",
    Predictor == "player_id" ~ "Player ID",
    Predictor == "distance" ~ "Distance",
    Predictor == "launch_angle_deg" ~ "Launch Angle",
    Predictor == "exit_velo_mph" ~ "Exit Velocity",
    Predictor == "hangtime" ~ "Hang Time")) |>
  select(1:4) |>
  gt() |>
  fmt_number(columns = c(2,4), decimals = 2) |>
  tab_header(title = "Final Model Variance Inflation Factor") |>
  tab_options(
    row_group.font.weight = "bold",
    column_labels.font.weight = "bold",
    table.font.names = "serif"
  ) |>
  tab_style(
    style = cell_fill(color = "#fbcbc7"),
    locations = cells_body(
      rows = (`Adjusted GVIF` > 2.2),
      columns = `Adjusted GVIF`
    ) |>
    tab_footnote(footnote =
      html("GVIF: Generalized Variance Inflation Factor<br>
          Df: Degrees of Freedom")) |>
    gtsave("../final_tcp_vif.png")

```

Uncaught Triangle Predictions

```

# run model on uncaught Bermuda Triangle Plays
catch_probability <- predict(model_triangle_mclogit,
                             newdata = uncaught_triangle_df,
                             type = "response")

# find player positions and names for uncaught Bermuda Triangles

```

```

uncaught_tri_lineups <- play_lineups |>
  filter(game_str_ppg %in% uncaught_tri_plays) |>
  pivot_longer(cols = c("pitcher", "catcher",
                        "first_base", "second_base", "third_base",
                        "shortstop", "left_field", "center_field",
                        "right_field"),
               names_to = "player_id",
               values_to = "name") |>
  dplyr::select(game_str_ppg, player_id, name) |>
  mutate(
    player_id = case_when(
      player_id == "pitcher" ~ "Pitcher",
      player_id == "catcher" ~ "Catcher",
      player_id == "first_base" ~ "1B",
      player_id == "second_base" ~ "2B",
      player_id == "third_base" ~ "3B",
      player_id == "shortstop" ~ "SS",
      player_id == "left_field" ~ "LF",
      player_id == "center_field" ~ "CF",
      player_id == "right_field" ~ "RF"
    ),
    player_id = as.factor(player_id),
    player_id = factor(player_id, levels = levels(triangle_df$player_id))
  )

# merge catch probabilities & names with hit metrics
uncaught_probabilities <- cbind(uncaught_triangle_df,
                                as.data.frame(catch_probability)) |>
# cbind is used here since no additional steps are being done on the frame
# in future, a full_join or left_join may better ensure reproducibility
mutate(catch_probability =
  as.numeric(format(catch_probability, scientific = FALSE))) |>
mutate(catch_probability = 100 * round(catch_probability, 4)) |>
left_join(uncaught_tri_lineups,
          by = join_by(game_str_ppg, player_id)) |>
mutate(name_short = str_remove_all(name, "[a-zA-Z-]"))

```

Uncaught Triangle Player Positions

```

uncaught_tri_players <- uncaught_probabilities |>
  mutate(game_str_ppg_player_id =
    paste(game_str_ppg, player_id, sep = "_")) |>
  pull(game_str_ppg_player_id)

uncaught_tri_plays <- uncaught_probabilities |>
  distinct(game_str_ppg) |>
  pull(game_str_ppg)

uncaught_tri_player_pos <- player_positions_df |>
  mutate(game_str_ppg_player_id =
    paste(game_str_ppg, player_id, sep = "_")) |>

```

```
filter(game_str_ppg_player_id %in% uncaught_tri_players) |>
dplyr::select(-game_str_ppg_player_id)

uncaught_tri_player_pos <- uncaught_tri_player_pos |>
  mutate(field_x = as.numeric(field_x),
         field_y = as.numeric(field_y))

uncaught_tri_player_speeds <- player_positions_df |>
  mutate(game_str_ppg_player_id =
    paste(game_str_ppg, player_id, sep = "_")) |>
  filter(game_str_ppg_player_id %in% uncaught_tri_players) |>
  group_by(game_str_ppg_player_id) |>
  mutate(field_x = as.numeric(field_x),
         field_y = as.numeric(field_y),
         timestamp = as.numeric(timestamp),
         change_x = field_x - lag(field_x),
         change_y = field_y - lag(field_y),
         change_time = timestamp - lag(timestamp),
         # calculate distance and exit velocity
         dist = sqrt(change_x^2 + change_y^2),
         speed_ft_ms = dist/change_time,
         # convert exit velocity to mph from ft/ms
         speed = speed_ft_ms * 1000,
         ) |>
  ungroup() |>
  left_join(ball_drop_time,
    by = join_by(game_str_ppg, game_string, play_per_game)) |>
  filter(timestamp <= drop_timestamp) |>
  left_join(ball_hit_time,
    by = join_by(game_str_ppg, game_string, play_per_game)) |>
  filter(timestamp >= hit_timestamp) |>
  dplyr::select(game_string, play_per_game, game_str_ppg_player_id,
    game_str_ppg, player_id, timestamp,
    field_x, field_y, speed) |>
  left_join(uncaught_probabilities, by = join_by(game_str_ppg, player_id))
```

Visualizations

Visualization Set-Up

```
library(viridis)
library(ggimage)

anonymous_image <- "https://images.zoogletools.com/s:bzglfiles/u/1152029
/46e005131ea378b4d510b8d3a1acf8511b72b36d/original/mlb-player.png/!!
/b%3AW1sicmVzaXplIiwuNTZdLFsibWF4I10sWyJ3ZSJdXQ%3D%3D
/meta%3AeyJzcmNCdWnrZXQiOiJiemdsZmlsZXMifQ%3D%3D.png"

player_images <- data.frame(
  player_id = c(1:9),
  photo_url = anonymous_image) |>
mutate(player_id = case_when(
  player_id == 1 ~ "Pitcher",
  player_id == 2 ~ "Catcher",
  player_id == 3 ~ "1B",
  player_id == 4 ~ "2B",
  player_id == 5 ~ "3B",
  player_id == 6 ~ "SS",
  player_id == 7 ~ "LF",
  player_id == 8 ~ "CF",
  player_id == 9 ~ "RF"
),
  player_id = as.factor(player_id),
  player_id = factor(player_id, levels = levels(triangle_df$player_id)))

catch_probability_images <-
  left_join(uncaught_probabilities, player_images,
    by = join_by(player_id))

theme_field <- function() {
  theme(
    plot.title = element_text(color = "white", margin = margin(t= 5)),
    legend.background = element_rect(fill = "#395d33"),
    legend.text = element_text(color = "white", family = "serif"),
    legend.title = element_text(color = "white", family = "serif",
      face = "bold"))
}

uncaught_tri_ball_pos <- ball_pos_df |>
  filter(game_str_ppg %in% uncaught_tri_plays) |>
  mutate(ball_position_x = as.numeric(ball_position_x),
    ball_position_y = as.numeric(ball_position_y)) |>
  left_join(ball_drop_time,
    by = join_by(game_str_ppg, game_string, play_per_game)) |>
  filter(timestamp <= drop_timestamp) |>
  left_join(ball_hit_time,
    by = join_by(game_str_ppg, game_string, play_per_game)) |>
```

```
filter(timestamp >= hit_timestamp)
```

Animation Code

```
animate_triangle_play <- function(
  game_string_input = "y1_d068_SQF_ARN",
  play_per_game_input = 169,
  x1 = 0, x2 = 100, y1 = 0, y2 = 100)
{
  # Set the specs for the gif
  options(gganimate.dev_args =
    list(width = 3, height = 3, units = 'in', res = 350))

  fps <- 50

  # Find the time of the pitch
  time_of_pitch <- ball_events %>%
    filter(game_string == game_string_input &
      play_per_game == play_per_game_input &
      ball_eventcode == 1) %>%
    collect() %>%
    pull(timestamp)

  # Get the Ball Tracking Data
  ball_tracking_data <- uncaught_tri_ball_pos %>%
    ## Filter to correct game
    filter(game_string == game_string_input &
      play_per_game == play_per_game_input) %>%
    ## Add on a type and player_id column to match with player_tracking_data
    mutate(type = "ball",
      player_id = NA) %>%
    ## Reorder and Rename Columns
    dplyr::select(game_string:timestamp, player_id, type,
      position_x = ball_position_x,
      position_y = ball_position_y,
      position_z = ball_position_z, everything())

  # Get the Player Tracking Data
  player_tracking_data <- uncaught_tri_player_pos %>%
    ## Filter to the Correct Game
    filter(game_string == game_string_input &
      play_per_game == play_per_game_input) %>%
    ## Calculate type and put position_z as NA
    mutate(type = "defense",
      position_z = NA
    ) %>%
    ## Reorder and Rename Columns
    dplyr::select(game_string:timestamp, player_id,
      type, position_x = field_x,
      position_y = field_y, position_z,
      everything())
```

```

# Combine all tracking data into 1 data frame
tracking_data <- bind_rows(player_tracking_data, ball_tracking_data) %>%
  ## Convert timestamps and positions to numeric
  mutate(across(c(timestamp, position_x, position_y, position_z),
                 as.numeric)) %>%
  ## Order data chronologically
  arrange(timestamp) %>%
  ## Align timestamps to account for mechanical measurement error
  mutate(timestamp_adj = plyr::round_any(timestamp, fps)) %>%
  ## Start the animation to start when the pitch is thrown
  filter(timestamp >= time_of_pitch) %>%
  ## Create a frame_id for animation
  mutate(frame_id = match(timestamp_adj, unique(timestamp_adj))) |>
  left_join(catch_probability_images,
            by = join_by(game_str_ppg, player_id))

# Make Field and Plot Points
p <- geom_baseball(league = "MiLB", xlims = c(x1, x2),
                  ylims = c(y1, y2)) +
  ## Plot all people as dots
  geom_image(data = tracking_data %>% filter(type != "ball"),
             aes(x = position_x, y = position_y, image = photo_url),
             size = 0.10) +
  ## Label on top of the people dots
  geom_label(data = tracking_data %>% filter(type == "defense"),
             aes(x = position_x, y = position_y - 15,
                  label = paste(catch_probability, "%"),
                  fill = catch_probability),
             color = "white", size = 2, family = "serif",
             show.legend = F) +
  scale_fill_continuous(low = "#FF8888", high = "#B22222") +
  ## Plot the ball
  geom_point(data = tracking_data %>%
             filter(type == "ball"),
             aes(x = position_x, y = position_y,
                  size = (position_z/3)),
             fill = "white", color = "darkred",
             shape = 21,
             show.legend = F) +
  ## Specify when to transition
  transition_time(frame_id) +
  ## Add Shadows
  shadow_wake(0.1, exclude_layer = c(1:16)) +
  annotate("text", x = c(x1, x1), y = c(y2, y2 - 10), color = "white",
          label = c(paste("Game: ", game_string_input),
                    paste("Play: ", play_per_game_input)),
          hjust = "left",
          family = "serif", fontface = "bold", size = 2)
theme_field()

# Find the number of frames

```

```

number_of_frames <- max(tracking_data$frame_id)

# Animate
p2 <- animate(p, fps = fps, nframes = number_of_frames,
              renderer = magick_renderer())

return(p2)
}

```

Model Play Table

```

library(webshot2)

triangle_train |>
  filter(game_str_ppg == "y1_d185_TDW_ARN_282") |>
  dplyr::select(exit_velo_mph, mid_dist, mid_nec_speed,
                backwards, player_id, catch) |>
  mutate(backwards = if_else(backwards == "Yes", "Yes", "No"),
         catch = if_else(catch == 1, "Yes", "No")) |>
  gt() |>
  tab_header(
    subtitle = "Play 282 of Game y1_d185_TDW_ARN",
    title = "Model Training Example"
  ) |>
  cols_label(
    exit_velo_mph = "Exit Velo (mph)",
    mid_dist = "2.35 Distance (ft)",
    mid_nec_speed = "2.35 Speed (ft/sec)",
    backwards = "Running Backwards?",
    player_id = "Player Position",
    catch = "Made Catch?"
  ) |>
  fmt_number(
    columns = c(exit_velo_mph:mid_nec_speed),
    decimals = 2
  ) |>
  tab_options(
    row_group.font.weight = "bold",
    column_labels.font.weight = "bold",
    table.font.names = "serif"
  ) |>
  tab_style(style = cell_fill(color = "lightgrey"),
            locations = cells_body(rows = 2)) |>
  gt_add_divider(
    columns = mid_dist,
    sides = "left",
    color = "darkgrey",
    weight = px(3),
    style = "solid"
  ) |>
  gt_add_divider(

```

```

    columns = c(mid_nec_speed, backwards, player_id, catch),
    sides = "left",
    color = "darkgrey",
    weight = px(2),
    style = "dashed"
) |>
cols_align(
  align = "center",
  columns = c(1:6)
) |>
gtsave("../y1_d211_QHX_ANI_186.png")

```

Individual Play Visualizations

```

player_speeds_A <- uncaught_tri_player_speeds |>
  filter(game_str_ppg == "y1_d068_SQF_ARN_169")
ball_A <- uncaught_tri_ball_pos |>
  filter(game_str_ppg == "y1_d068_SQF_ARN_169") |>
  filter(timestamp == max(timestamp)) |>
  dplyr::select(ball_position_x, ball_position_y)

geom_baseball(league = "MiLB",
  xlims = c(-50, 150),
  ylims = c(100, 350)) +
geom_point(data = player_speeds_A,
  aes(x= field_x, y= field_y, color = speed),
  alpha = 0.5) +
scale_color_viridis(name = "Speed (ft/s)",
  limits = c(0, 30),
  option = "A") +

theme_field() +
annotate("text",
  x = c(57.39, 19.59, 109.48),
  y = c(115, 133, 289),
  color = "white",
  label = c("1.07%", "11.25%", "87.68%"),
  family = "serif",
  size = 3.5) +
annotate("text",
  x = c(57.39, 19.59, 109.48),
  y = c(125, 143, 299),
  color = "white",
  label = c("2B: 9323", "SS: 1503", "RF: 3382"),
  family = "serif",
  fontface = "bold",
  size = 3.5) +
geom_point(data = ball_A,
  aes(x= ball_position_x, y= ball_position_y),
  color = "white",
  size = 4)

ggsave("../imageA.png")

```

```
play_A <- uncaught_probabilities |>
  filter(game_str_ppg == "y1_d068_SQF_ARN_169") |>
  select(exit_velo_mph, mid_dist, mid_nec_speed,
         backwards, player_id, name_short, catch_probability) |>
  gt() |>
  tab_header(
    title = "Game y1_d068_SQF_ARN",
    subtitle = "Play 169"
  ) |>
  cols_label(
    exit_velo_mph = "Exit Velo (mph)",
    mid_dist = "2.35 Distance (ft)",
    mid_nec_speed = "2.35 Speed (ft/sec)",
    backwards = "Running Backwards?",
    player_id = "Position",
    name_short = "Unique ID",
    catch_probability = "Catch Probability (%)"
  ) |>
  fmt_number(
    columns = c(1:3),
    decimals = 2
  ) |>
  tab_options(
    row_group.font.weight = "bold",
    column_labels.font.weight = "bold",
    table.font.names = "serif"
  ) |>
  gt_add_divider(
    columns = 2,
    sides = "left",
    color = "darkgrey",
    weight = px(2),
    style = "dashed"
  ) |>
  cols_align(
    align = "center",
    columns = c(1:6)
  ) |>
  data_color(
    columns = catch_probability,
    fn = scales::col_bin(
      palette = c("transparent", "#fbcbc7"),
      domain = c(0, 100),
      bins = c(0, 40, 100)
    ),
    alpha = 0.7
  ) |>
  gtsave("../playA.png")
```

```

player_speeds_B <- uncaught_tri_player_speeds |>
  filter(game_str_ppg == "y1_d074_AIP_ARN_113")
ball_B <- uncaught_tri_ball_pos |>
  filter(game_str_ppg == "y1_d074_AIP_ARN_113") |>
  filter(timestamp == max(timestamp)) |>
  dplyr::select(ball_position_x, ball_position_y)

geom_baseball(league = "MiLB",
              xlims = c(-100, 100),
              ylims = c(-40, 210)) +
  geom_point(data = player_speeds_B,
            aes(x= field_x, y= field_y,color = speed),
            alpha = 0.5) +
  scale_color_viridis(name = "Speed (ft/s)",
                    limits = c(0, 30),
                    option = "A") +
  theme_field() +
  annotate("text",
        x = c(-18, -1.56, 58.25, 16.27, -66.19),
        y = c(43, -23, 80, 158, 98),
        color = "white",
        label = c("1.65%", "0.05%", "44.26%",
                  "4.49%", "49.55%"),
        family = "serif",
        size = 3.5) +
  annotate("text",
        x = c(-18, -1.56, 58.25, 16.27, -66.19),
        y = c(52, -13, 90, 168, 108),
        color = "white",
        label = c("P: 6151", "C: 2385", "1B: 3323",
                  "2B: 1650", "3B: 2415"),
        family = "serif",
        fontface = "bold",
        size = 3.5) +
  geom_point(data = ball_B,
            aes(x= ball_position_x, y= ball_position_y),
            color = "white",
            size = 4)

ggsave("../imageB.png")

play_B <- uncaught_probabilities |>
  filter(game_str_ppg == "y1_d074_AIP_ARN_113") |>
  select(exit_velo_mph, mid_dist, mid_nec_speed,
        backwards, player_id, name_short, catch_probability) |>
  gt() |>
  tab_header(
    title = "Game y1_d074_AIP_ARN",
    subtitle = "Play 113"
  ) |>

```

```

cols_label(
  exit_velo_mph = "Exit Velo (mph)",
  mid_dist = "2.35 Distance (ft)",
  mid_nec_speed = "2.35 Speed (ft/sec)",
  backwards = "Running Backwards?",
  player_id = "Position",
  name_short = "Unique ID",
  catch_probability = "Catch Probability (%)"
) |>
fmt_number(
  columns = c(1:3),
  decimals = 2
) |>
tab_options(
  row_group.font.weight = "bold",
  column_labels.font.weight = "bold",
  table.font.names = "serif"
) |>
gt_add_divider(
  columns = 2,
  sides = "left",
  color = "darkgrey",
  weight = px(2),
  style = "dashed"
) |>
cols_align(
  align = "center",
  columns = c(1:6)
) |>
data_color(
  columns = catch_probability,
  fn = scales::col_bin(
    palette = c("transparent", "#fbcbc7"),
    domain = c(0, 100),
    bins = c(0, 48, 100)
  ),
  alpha = 0.7
) |>
gtsave("../playB.png")

```

```

player_speeds_C <- uncaught_tri_player_speeds |>
  filter(game_str_ppg == "y1_d122_MSE_ANI_56")
ball_C <- uncaught_tri_ball_pos |>
  filter(game_str_ppg == "y1_d122_MSE_ANI_56") |>
  filter(timestamp == max(timestamp)) |>
  dplyr::select(ball_position_x, ball_position_y)

geom_baseball(league = "MiLB",
  xlims = c(-120, 80),
  ylims = c(120, 370)) +

```

```

geom_point(data = player_speeds_C,
           aes(x= field_x, y= field_y, color = speed),
           alpha = 0.5) +
scale_color_viridis(name = "Speed (ft/s)",
                   limits = c(0, 30),
                   option = "A") +
theme_field() +
annotate("text",
        x = c(-35, 30.74, 23.29),
        y = c(125, 130, 347),
        color = "white",
        label = c("1.59%", "8.22%", "90.19%"),
        family = "serif", size = 3.5) +
annotate("text",
        x = c(-35, 30.74, 23.29),
        y = c(135, 140, 357),
        color = "white",
        label = c("SS: 1872", "2B: 3335", "CF: 1669"),
        family = "serif",
        fontface = "bold",
        size = 3.5) +
geom_point(data = ball_C,
           aes(x= ball_position_x, y= ball_position_y),
           color = "white",
           size = 4)

ggsave("../imageC.png")

play_C <- uncaught_probabilities |>
  filter(game_str_ppg == "y1_d122_MSE_ANI_56") |>
  select(exit_velo_mph, mid_dist, mid_nec_speed,
         backwards, player_id, name_short, catch_probability) |>
  gt() |>
  tab_header(
    title = "Game y1_d122_MSE_ANI",
    subtitle = "Play 56"
  ) |>
  cols_label(
    exit_velo_mph = "Exit Velo (mph)",
    mid_dist = "2.35 Distance (ft)",
    mid_nec_speed = "2.35 Speed (ft/sec)",
    backwards = "Running Backwards?",
    player_id = "Position",
    name_short = "Unique ID",
    catch_probability = "Catch Probability (%)"
  ) |>
  fmt_number(
    columns = c(1:3),
    decimals = 2
  ) |>
  tab_options(

```

```

    row_group.font.weight = "bold",
    column_labels.font.weight = "bold",
    table.font.names = "serif"
  ) |>
  gt_add_divider(
    columns = 2,
    sides = "left",
    color = "darkgrey",
    weight = px(2),
    style = "dashed"
  ) |>
  cols_align(
    align = "center",
    columns = c(1:6)
  ) |>
  data_color(
    columns = catch_probability,
    fn = scales::col_bin(
      palette = c("transparent", "#fbcbc7"),
      domain = c(0, 100),
      bins = c(0, 40, 100)
    ),
    alpha = 0.7
  ) |>
  gtsave("../playC.png")

```

```

player_speeds_D <- uncaught_tri_player_speeds |>
  filter(game_str_ppg == "y1_d138_QBR_ANI_128")
ball_D <- uncaught_tri_ball_pos |>
  filter(game_str_ppg == "y1_d138_QBR_ANI_128") |>
  filter(timestamp == max(timestamp)) |>
  dplyr::select(ball_position_x, ball_position_y)

geom_baseball(league = "MiLB",
  xlims = c(-100, 100),
  ylims = c(0, 250)) +
  geom_point(data = player_speeds_D,
    aes(x= field_x, y= field_y, color = speed),
    alpha = 0.5) +
  scale_color_viridis(name = "Speed (ft/s)",
    limits = c(0, 30),
    option = "A") +
  theme_field() +
  annotate("text",
    x = c(55, 10, -38.82),
    y = c(110, 152, 105),
    color = "white",
    label = c("0.57%", "98.81%", "0.62%"),
    family = "serif",
    size = 3.5) +
  annotate("text",

```

```

      x = c(55, 10, -38.82),
      y = c(120, 162, 115),
      color = "white",
      label = c("1B: 2596", "2B: 3725", "SS: 2218"),
      family = "serif",
      fontface = "bold",
      size = 3.5) +
geom_point(data = ball_D,
           aes(x= ball_position_x, y= ball_position_y),
           color = "white",
           size = 4)

ggsave("../imageD.png")

play_D <- uncaught_probabilities |>
  filter(game_str_ppg == "y1_d138_QBR_ANI_128") |>
  select(exit_velo_mph, mid_dist, mid_nec_speed,
         backwards, player_id, name_short, catch_probability) |>
  gt() |>
  tab_header(
    title = "Game y1_d138_QBR_ANI",
    subtitle = "Play 128"
  ) |>
  cols_label(
    exit_velo_mph = "Exit Velo (mph)",
    mid_dist = "2.35 Distance (ft)",
    mid_nec_speed = "2.35 Speed (ft/sec)",
    backwards = "Running Backwards?",
    player_id = "Position",
    name_short = "Unique ID",
    catch_probability = "Catch Probability (%)"
  ) |>
  fmt_number(
    columns = c(1:3),
    decimals = 2
  ) |>
  tab_options(
    row_group.font.weight = "bold",
    column_labels.font.weight = "bold",
    table.font.names = "serif"
  ) |>
  gt_add_divider(
    columns = 2,
    sides = "left",
    color = "darkgrey",
    weight = px(2),
    style = "dashed"
  ) |>
  cols_align(
    align = "center",
    columns = c(1:6)
  )

```

```
) |>
data_color(
  columns = catch_probability,
  fn = scales::col_bin(
    palette = c("transparent", "#fbcbc7"),
    domain = c(0, 100),
    bins = c(0, 40, 100)
  ),
  alpha = 0.7
) |>
gtssave("../playD.png")
```