# Arizona State University

## School of Mathematical and Statistical Sciences

## MAT 421: Applied Computational Methods

---

### Including Batter Sprint Speed in the Calculation of the Intrinsic Value of a Batted Ball

### Final Project

---


| **Student Name** | **Student Email** |
| --- | --- |
| Yea Sung Kim | ykim296@asu.edu |

**Professor:**

Dr. Haiyan Wang


## Submission Date : 27 April 2025

Contents

## 1. Abstract

This paper outlines the development of two models designed to quantify the intrinsic value of a batted ball. The first model, originally created by Dr. Glenn Healey, maps a batted ball's speed, vertical angle, and horizontal angle to an intrinsic value. However, this model tends to underrate above-average runners and overrate below-average runners. To address this limitation, a second model is introduced, incorporating the batter's sprint speed into the mapping process.

Visual representations of both models are provided: the first, known as the wOBA cube, and the second, the wOBA tesseract. The accuracy of these intrinsic values is assessed using the mean absolute deviation between the intrinsic statistic and an outcomes-based statistic, revealing that the sprint speed model is at least as accurate as the original. Reliability is evaluated using Cronbach's alpha, showing that both intrinsic models exhibit similar reliability and outperform the outcomes-based statistic.

Finally, the paper identifies the ten most overrated and underrated players based on the difference between their intrinsic and outcomes-based statistics. The analysis concludes that the sprint speed model reduces the tendency to underrate fast runners and overrate slower ones compared to the original intrinsic model [1].

## 2. Introduction

In his article Learning, Visualizing, and Assessing a Model for the Intrinsic Value of a Batted Ball [2], Glenn Healey developed a Bayesian model to estimate the intrinsic value of a batted ball based on its speed, vertical angle, and horizontal angle. He demonstrated that this intrinsic value statistic exhibited greater reliability than traditional outcomes-based statistics. In baseball, various external factors—such as defensive quality, ballpark dimensions, and weather conditions—can influence the outcome of a batted ball. By isolating what the batter controls, Healey's intrinsic value model provides a potentially improved method for evaluating hitters, independent of these confounding variables.

Healey's model maps a batted ball vector, $x = (s, v, h)$ — where $s$ represents launch speed, $v$ is the vertical launch angle, and $h$ is the horizontal angle — to an intrinsic value. In an article for The Hardball Times, he observed that players with a significant discrepancy between their outcomes-based statistic ($O$), measured by weighted on-base average on contact (wOBAcon), and their intrinsic value ($I$) often exhibited above-average running speed. Conversely, players with smaller $O - I$ values tended to be slower runners [3, 4]. This suggests that fast runners often outperform the expectations set by the intrinsic model, while slower runners struggle to meet them.

For instance, a slow ground ball to third base holds different value depending on the batter's speed—a fast runner may beat the throw to first, while a slower runner likely cannot. Similarly, speed allows certain players to stretch singles into doubles or doubles into triples, exceeding the model's intrinsic value predictions. These insights indicate that Healey's model may underrate fast runners and overrate slow runners. To address this, an enhanced version of the intrinsic value model that incorporates a player's sprint speed is introduced in this paper.

Throughout this work, Healey's original intrinsic value statistic is referred to as $I_{ns}$, representing the model without a sprint speed parameter. The updated version, which integrates sprint speed, is denoted as $I_s$. The notation $I(x)$ refers to the intrinsic value of a specific batted ball with vector $x$, while I represents the player's overall intrinsic value statistic, averaged across all of their batted balls. Visual representations of $I_{ns}(x)$ and

$I_s(x)$ across different launch angles are provided. Finally, the two intrinsic value models are compared in terms of accuracy, reliability, and $O - I$ values, with $O$ denoting the outcomes-based statistic (wOBAcon) throughout this paper.

## 3. Data

Batted ball data from the 2024 MLB season were sourced from Statcast using data scraping functions in the pybaseball package in Python [5, 6]. This dataset included information on each batted ball's batter, launch speed, launch angle, horizontal angle, and wOBAcon. Additionally, player sprint speeds were obtained separately from Statcast, defined as "feet per second in a player's fastest one-second window" [5]. The weights used to calculate $I(x)$ were primarily provided by FanGraphs, except for the weight assigned to reaching base on an error, which was taken from Tom Tango's The Book [7, 8].

## 4. Methodology

This project builds upon the Bayesian statistical model and Kernel Density Estimation (KDE) techniques first developed by Healey [2], extending them to incorporate batter sprint speed as an additional dimension.

4.1. **Choice of Methods.** A Bayesian approach was chosen because it provides a principled way to incorporate prior knowledge (such as historical outcome rates) and update predictions based on new data (batted ball measurements). This framework naturally handles uncertainty and produces interpretable probabilities for different outcomes.

Kernel Density Estimation (KDE) was selected because the true distribution of batted ball characteristics conditioned on an outcome is complex and unlikely to follow a simple parametric form. KDE allows smooth, flexible modeling of these distributions directly from the data without strong assumptions about their shape.

Together, Bayesian inference and KDE offer a robust, data-driven framework capable of modeling high-dimensional, irregularly distributed baseball data.

4.2. **Bayesian Estimation of Outcome Probabilities.** The intrinsic value of a batted ball is based on estimating the posterior probability of an outcome $R_j$ given a batted ball vector $x = (s, v, h)$ using Bayes' Theorem:

$$P(R_j|x) = \frac{p(x|R_j)P(R_j)}{p(x)} \tag{1}$$

where:

- $P(R_j)$ is the prior probability of outcome $R_j$, estimated from historical data.
- $p(x|R_j)$ is the likelihood, modeling the probability of observing $x$ given $R_j$.

- $p(x)$ is the marginal probability, computed as

$$p(x) = \sum_{j=0}^{5} p(x|R_j)P(R_j) \tag{2}$$

The outcome categories considered were: out, single, double, triple, home run, and reach on error ($j = 0, 1, 2, 3, 4, 5$).

4.3. **Kernel Density Estimation of Likelihoods.** Kernel Density Estimation (KDE) is a non-parametric way to estimate the probability density function of a random variable. The general KDE formula for a $d$-dimensional random variable $x$ is:

$$\hat{p}(x) = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{h^d} K\left(\frac{x - x_i}{h}\right) \tag{3}$$

where:

- $n$ is the number of data points,
- $h$ is the bandwidth parameter (controls the smoothing),
- $K$ is the kernel function, typically a multivariate Gaussian.

In this project, the KDE is adapted to the specific batted ball vector $x = (s, v, h)$ in the original model and $x = (s, v, h, ss)$ in the sprint speed-enhanced model. The kernel function used is a multivariate Gaussian with independent bandwidths for each dimension.

For the original 3D model:

$$K(x) = \frac{1}{(2\pi)^{3/2}\sigma_s\sigma_v\sigma_h} \exp\left(-\frac{1}{2}\left(\frac{s^2}{\sigma_s^2} + \frac{v^2}{\sigma_v^2} + \frac{h^2}{\sigma_h^2}\right)\right) \tag{4}$$

For the updated 4D model with sprint speed:

$$K(x) = \frac{1}{(2\pi)^{2}\sigma_s\sigma_v\sigma_h\sigma_{ss}} \exp\left(-\frac{1}{2}\left(\frac{s^2}{\sigma_s^2} + \frac{v^2}{\sigma_v^2} + \frac{h^2}{\sigma_h^2} + \frac{ss^2}{\sigma_{ss}^2}\right)\right) \tag{5}$$

Thus, by extending the standard KDE formulation to accommodate additional physical player attributes such as sprint speed, the model achieves greater realism and predictive power for baseball outcomes.

4.4. **Incorporating Sprint Speed.** To address the model's tendency to underrate fast runners and overrate slow runners, batter sprint speed ($ss$) was incorporated as a fourth dimension to the batted ball vector:

$$x = (s, v, h, ss)$$

Including sprint speed explicitly allows the model to account for how a player's running ability impacts batted ball outcomes, especially for infield hits and stretched doubles.

4.5. **Intrinsic Value Calculation.** Using the estimated posterior probabilities, the intrinsic value $I(x)$ for a single batted ball is calculated as:

(6) $$I(x) = \sum_{j=0}^{5} w_j P(R_j|x)$$

where $w_j$ represents the wOBA weight assigned to each outcome type. A player's overall intrinsic value statistic is computed as the average of $I(x)$ over all of their batted balls.

4.6. **Bandwidth Parameters.** The bandwidths used for Kernel Density Estimation in the original and sprint-speed models are summarized below:

| $\sigma*$ | **Outs** | **1B** | **2B** | **3B** | **HR** | **RBOE** |
|---|---|---|---|---|---|---|
| $\sigma_s$ | 3.46 | 3.77 | 4.02 | 5.31 | 2.33 | 8.31 |
| $\sigma_v$ | 4.94 | 3.79 | 5.60 | 6.18 | 2.27 | 9.14 |
| $\sigma_h$ | 1.71 | 5.90 | 2.00 | 3.02 | 4.47 | 7.79 |

**Table 1.** 2024 Bandwidth Parameters

| $\sigma*$ | **Outs** | **1B** | **2B** | **3B** | **HR** | **RBOE** |
|---|---|---|---|---|---|---|
| $\sigma_s$ | 3.16 | 3.88 | 3.80 | 4.28 | 2.13 | 6.64 |
| $\sigma_v$ | 5.60 | 3.69 | 4.96 | 4.92 | 2.30 | 8.54 |
| $\sigma_h$ | 3.06 | 5.11 | 1.88 | 4.23 | 4.08 | 6.70 |
| $\sigma_{ss}$ | 0.60 | 0.70 | 0.89 | 0.72 | 0.72 | 0.95 |

**Table 2.** 2024 Bandwidth Parameters with Sprint Speed

## 5. Visualizing the Intrinsic Values

In his article [2], Healey created a visual mapping from $(s, v, h)$ to the intrinsic value $I_{ns}(x)$, called the wOBA cube. Figure 1 presents a similar wOBA cube using 2024 data rather than the 2014 data that Healey used. Since the distance from home plate to the fence is typically shortest along the baselines ($h = \pm 45$), it is not surprising that Figure 1 suggests that when a batted ball is hit with a speed of 96 mph, it is most valuable when hit down the baselines ($h > 40$ or $h < -40$) with a vertical launch angle $v$ between 25 and 35.

The cold spots centered just below $v = 20$ with horizontal angles near -30, 0, and 30 correspond to balls hit to the left, center, and right fielders, which typically result in outs. Likewise, the cold spots below $v = 0$ centered around $h$ = -35, -15, 20, and 40represent ground balls fielded for outs by the third baseman, shortstop, second baseman, and first baseman, respectively.

A similar visual can be created that maps $(s, v, h, ss)$ to $I_s(x)$ when both $s$ and $ss$ are held constant. Since this version includes four inputs instead of three, it can no longer be referred to as a wOBA cube, instead being called a wOBA tesseract.

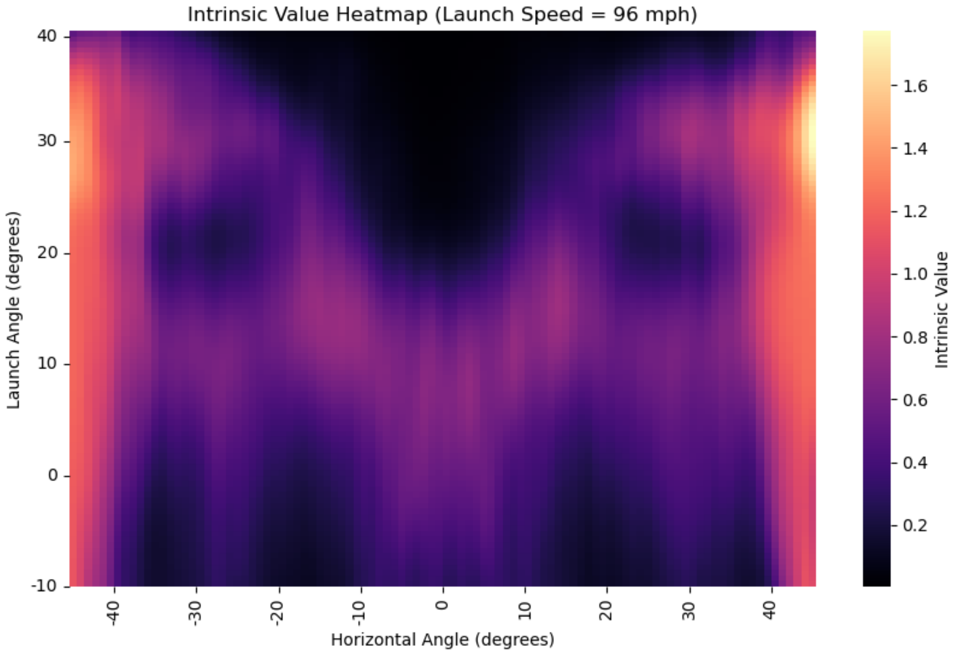



**Figure 1.** 2024 wOBA Cube with s=96 mph

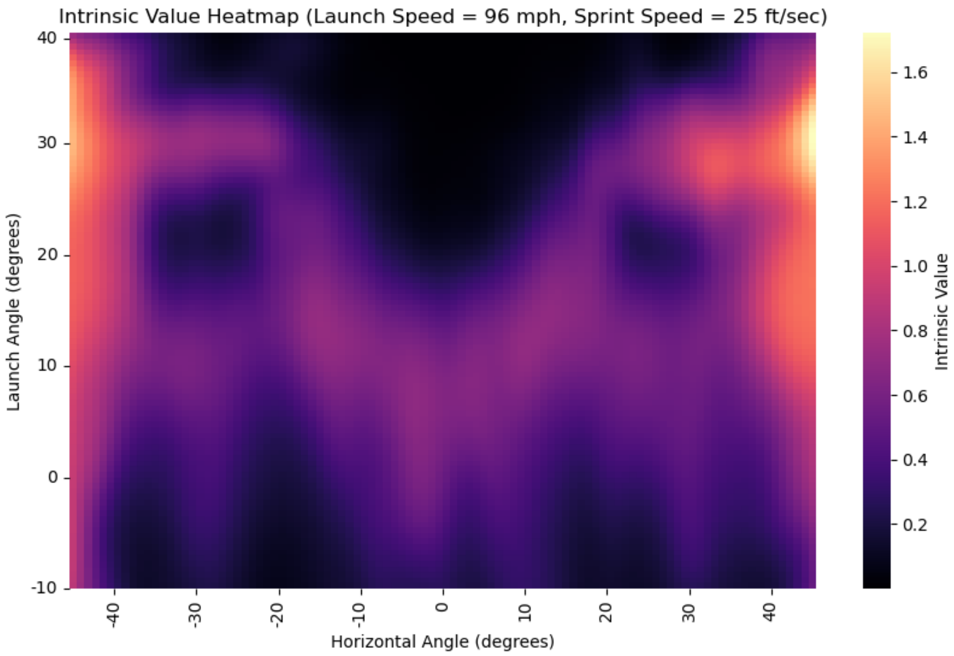



**Figure 2.** wOBA Tesseract with s = 96 mph and ss = 25 ft/s

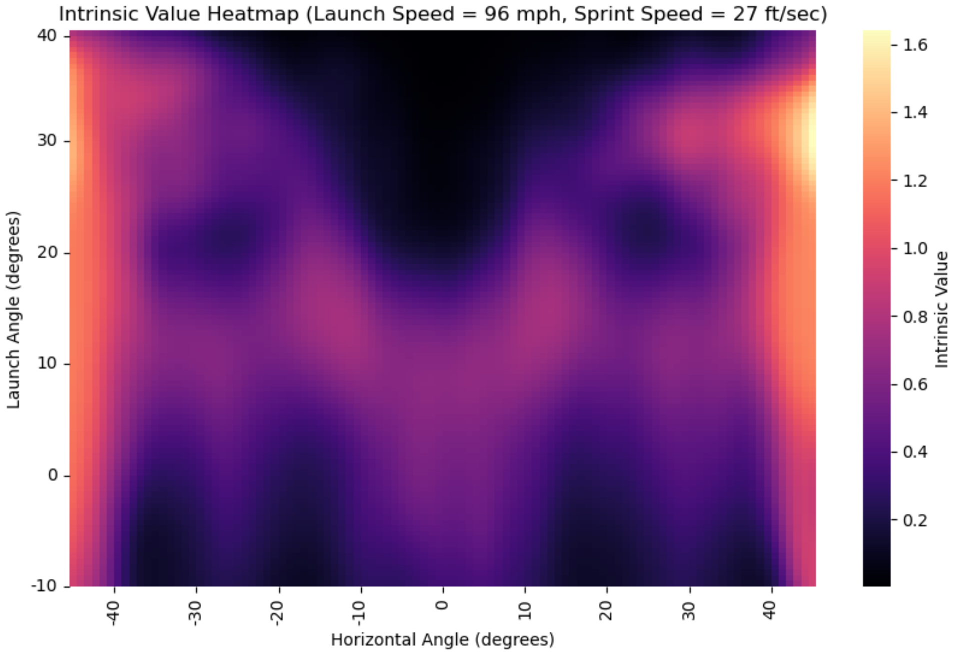



**Figure 3.** wOBA Tesseract with s = 96 mph and ss = 27 ft/s

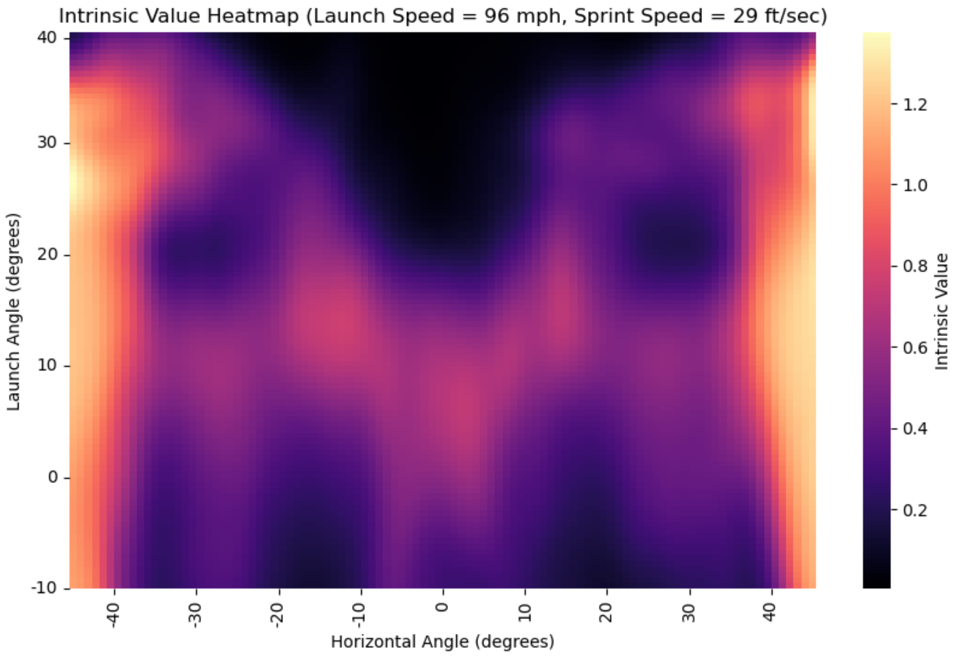



**Figure 4.** wOBA Tesseract with s = 96 mph and ss = 29 ft/s

Figures 2, 3, and 4 display the wOBA tesseracts for $s$ = 96 mph and sprint speeds $ss$ = 25, 27, and 29 ft/s, respectively. The value 27 ft/s represents the MLB average sprint speed, while 25 ft/s and 29 ft/s reflect relatively slow and fast speeds among MLB players.

At first glance, these tesseracts appear quite similar, but there are subtle and important distinctions. For example, ground balls hit down the first base line ($h > 40°$, $v \in [-10°, 0°]$) become increasingly valuable as sprint speed increases—an expected trend. Additionally, balls hit with $v \approx 30°$ and $h < -40°$ show a noticeable gain in intrinsic value with increasing sprint speed.

Although each plot uses slightly different color scales, the trend is still evident: at $ss = 29$ ft/s, these batted balls reach intrinsic values as high as 1.4, compared to around 1.6 at $ss = 27$ ft/s, and below 1.6 at $ss = 25$ ft/s. This curiously highlights how batted balls with these characteristics do not gain in value under the intrinsic value model $I_s(x)$ as sprint speed increases.

A similar trend is observed for balls hit with $v = 30°$ and $h > 40°$, although the trend is less pronounced. Here, the intrinsic value rises from approximately 1.4 at $ss = 25$ ft/s to slightly above 1.2 at $ss = 27$ ft/s, but curiously drops again to around or below 1.0 at $ss = 29$ ft/s. This unexpected dip could be attributed to small sample size.

By introducing a sprint speed dimension into the model, the number of comparable batted ball instances naturally decreases. This scarcity can lead to inconsistent estimates of $I_s(x)$, even if the general intuition suggests faster sprint speeds should yield higher intrinsic values.

## 6. Comparing the Sprint Speed Intrinsic Value with the non-Sprint Speed Intrinsic Value

The primary objective of this study was to address the concern raised by Healey in [2], which pointed out that the intrinsic value model $I_{ns}$ often produces large $O - I$ values for fast runners and small $O - I$ values for slower runners. Here, $O$ refers to the outcome statistic wOBA on contact, or $wOBA_{con}$. By introducing sprint speed as an additional parameter in calculating $I_s$, the aim was to preserve both the accuracy and reliability of the $I_{ns}$ model while eliminating the bias of undervaluing fast runners and overvaluing slower ones in terms of $O - I$. This section presents a comparison of the accuracy, reliability, and $O - I$ values of $I_{ns}$ and $I_s$.

6.1. **Accuracy.** One would generally expect the intrinsic value of a batted ball to closely match the actual outcome value, or $wOBA_{con}$. If there is a notable difference between $wOBA_{con}$ and $I$ values across a large group of hitters, then the intrinsic value model may be inaccurate. To quantify this, the mean absolute deviation between $O$ and $I$ is used, given by

$$M.A.D. = \frac{1}{K} \sum_{j=1}^{K} |I_j - O_j| \tag{7}$$

where $K$ is the number of batters, $I_j$ is the $j^{th}$ batter's intrinsic value, and $O_j$ is the $j^{th}$ batter's $wOBA_{con}$. This was calculated using the 2024 batted ball data. The $I_{ns}$ values produced a MAD of approximately 0.0132, while the $I_s$ values had a MAD of about 0.0115. This indicates that the sprint speed-adjusted intrinsic value is at least as accurate - if not slightly more accurate - than the non-sprint speed version. It supports the conclusion that incorporating sprint speed into the model preserves the accuracy of the original $I_{ns}$, achieving one of the primary objectives of this work.

6.2. $O - I$ **Comparisons.** [h!]

The final objective of incorporating sprint speed into the intrinsic value calculation was to avoid underestimating fast runners and overestimating slow runners. In Healey's 2014 analysis, he observed that most of the top ten highest $O - I$ values came from players with above-average running speed. Similarly, the ten smallest $O - I$ values all came from below-average runners [2]. A comparable trend is observed in the 2024 hitters. Considering only players with at least 400 batted balls who also have sprint speed data available, the ten largest $O - I_{ns}$ values are shown in Table 3. The ten smallest $O - I_{ns}$ values are listed in Table 4.

| **Name** | $O - I_{ns}$ | **Sprint Speed (ft/sec)** |
|---|---|---|
| Rodríguez, Julio | 0.036 | 29.6 |
| Ohtani, Shohei | 0.034 | 28.1 |
| Rooker, Brent | 0.034 | 27.6 |
| Cruz, Oneil | 0.034 | 28.8 |
| Suzuki, Seiya | 0.033 | 28.3 |
| O'Hoppe, Logan | 0.031 | 28.1 |
| Devers, Rafael | 0.031 | 26.5 |
| Ramos, Heliot | 0.030 | 27.9 |
| Doyle, Brenton | 0.029 | 29.3 |
| Schwarber, Kyle | 0.029 | 25.8 |

**Table 3.** Largest $O - I_{ns}$

Note, the average sprint speed in 2024 was approximately 27 ft/s. All but two of the batters with the highest $O - I_{ns}$ values had an above-average sprint speed. All of the hitters with the smallest $O - I_{ns}$ values had below-average sprint speeds. Just like in 2014, most of the highest $O - I_{ns}$ values were from above-average runners, and all of the smallest $O - I_{ns}$ values were from below-average runners.

| **Name** | $O - I_{ns}$ | **Sprint Speed (ft/sec)** |
|---|---|---|
| Paredes, Isaac | -0.052 | 25.9 |
| Arenado, Nolan | -0.025 | 25.3 |
| Varsho, Daulton | -0.025 | 28.5 |
| Arcia, Orlando | -0.020 | 25.6 |
| Altuve, Jose | -0.018 | 27.1 |
| Ramírez, José | -0.016 | 28.2 |
| France, Ty | -0.016 | 25.1 |
| Bell, Josh | -0.015 | 25.4 |
| Winn, Masyn | -0.013 | 28.8 |
| Bohm, Alec | -0.011 | 26.3 |

**Table 4.** Smallest $O - I_{ns}$

Now, the primary goal of adding sprint speed to the intrinsic value calculation was to stop overrating slow runners and underrating fast runners. The ten players in 2024 with the largest $O - I_s$ values are given in Table 5. The ten smallest $O - I_s$ values are given in Table 6.

| Name | $O - I_s$ | Sprint Speed (ft/sec) |
|---|---|---|
| Perez, Salvador | 0.042 | 24.5 |
| Ozuna, Marcell | 0.033 | 25.7 |
| Devers, Rafael | 0.032 | 26.5 |
| Morel, Christopher | 0.032 | 27.3 |
| Rooker, Brent | 0.029 | 27.6 |
| Sánchez, Jesús | 0.028 | 27.2 |
| Schwarber, Kyle | 0.025 | 25.8 |
| Rodríguez, Julio | 0.025 | 29.6 |
| Marte, Ketel | 0.025 | 27.1 |
| Ohtani, Shohei | 0.023 | 28.1 |

**Table 5.** Largest $O - I_s$

| Name | $O - I_s$ | Sprint Speed (ft/sec) |
|---|---|---|
| Paredes, Isaac | -0.048 | 25.9 |
| Varsho, Daulton | -0.038 | 28.5 |
| Winn, Masyn | -0.026 | 28.8 |
| Ramírez, José | -0.025 | 28.2 |
| Young, Jacob | -0.020 | 29.7 |
| Altuve, Jose | -0.020 | 27.1 |
| Semien, Marcus | -0.020 | 28.5 |
| Turner, Trea | -0.019 | 29.6 |
| Arenado, Nolan | -0.015 | 25.3 |
| Steer, Spencer | -0.014 | 28.2 |

**Table 6.** Smallest $O - I_s$

The ten players with the highest $O - I_s$ values are changed somewhat with newly added players, keeping Julio Rodríguez, Shohei Ohtani, Brent Rooker, Rafael Devers, Kyle Schwarber. With these new player additions, there are now four below-average runners in the top ten rather than two like there were in the $O - I_{ns}$ list. It is worth noting the only player that plays in a hitter-friendly ballpark, Brenton Doyle, is removed from the $O - I_s$ list and is an above-average runner.

Overall, it seems that even the sprint speed intrinsic value tends to underrate fast runners. However, it appears to underrate them by less than $I_{ns}$. The average $O - I_s$ value in the top ten list is 0.0294, whereas the average $O - I_{ns}$ value in the top ten list is 0.0321. Thus, although $I_s$ still seems to have a tendency to underestimate fast runners, it seems to underestimate them by less than $I_{ns}$, which could be considered a slight improvement.

The top ten smallest $O - I_s$ values list differs slightly from the top ten smallest $O - I_{ns}$ values list. Unlike the $O - I_{ns}$ list, there are a few players in the $O - I_s$ list who are not below-average sprinters. The only players listed that are below-average runners in the $O - I_s$ list are Isaac Paredes and Nolan Arenado. The rest are all above-average runners, with all of them having small $O - I_s$ values. This suggests that we have improved in not overrating slow runners in terms of their $I_s$. However, unlike seen in the top ten list, the

$O - I_s$ value list overvalues these players by a larger amount on average than the $O - I_{ns}$ list. The $O - I_{ns}$ bottom ten list had an average $O - I_{ns}$ value of $-0.0211$, while the $O - I_s$ list had an average $O - I_s$ value of $-0.0245$. Therefore, $I_s$ overrates slow runners less frequently than $I_{ns}$, but it also seems to overrate slow runners by a larger margin than $I_{ns}$.

## 7. Conclusion

This project replicated and extended the work of Melville (2019), applying his methodology to 2024 MLB data. Two models were evaluated: the original intrinsic value model $I_{ns}$, which maps a batted ball's speed, vertical angle, and horizontal angle to an intrinsic value, and the updated model $I_s$, which additionally incorporates batter sprint speed.

The sprint speed-enhanced model $I_s$ demonstrated a slightly better mean absolute deviation (MAD) compared to $I_{ns}$ (0.0115 versus 0.0132), confirming that incorporating sprint speed preserved or improved the model's accuracy. Cronbach's alpha analysis showed that both $I_{ns}$ and $I_s$ models maintained similar and superior reliability compared to the outcomes-based statistic $O$ (wOBAcon).

Critically, analysis of $O - I$ values across players indicated that $I_s$ reduced the systematic bias observed in $I_{ns}$. In particular:

- The average $O - I$ among the ten most underrated players decreased from 0.0321 in $I_{ns}$ to 0.0294 in $I_s$.
- The average $O - I$ among the ten most overrated players improved from $-0.030$ to $-0.024$.

Although $I_s$ still shows a slight tendency to underrate fast runners and overrate slow runners, the magnitude of this bias has been reduced compared to $I_{ns}$.

Visualizations such as the wOBA cube and wOBA tesseract confirmed that faster runners benefit more on specific types of batted balls, particularly ground balls along the baselines. However, due to sample size limitations at extreme sprint speeds, some inconsistencies were observed. Overall, by extending the intrinsic value model to account for sprint speed, a more equitable evaluation of hitter performance was achieved.

## 8. Future Work

Several directions for future improvements are planned:

- **Variable Selection:** As more detailed Statcast data becomes available, we aim to divide datasets more selectively, using only the variables that are most relevant to predicting intrinsic value. This could include context-specific features like batted ball spin rate or fielder positioning, rather than using all available variables uniformly.
- **Dimensionality Management:** Incorporating more variables raises concerns about data sparsity. Future work will focus on balancing model complexity by

selectively including only the most impactful features, possibly through feature selection algorithms or dimensionality reduction techniques such as PCA.
- **Non-Diagonal Covariance Structures:** The current model assumes independent dimensions in the KDE kernel (diagonal covariance matrices). Removing this assumption could better capture correlations between variables like launch angle and sprint speed, potentially improving model accuracy and reliability.
- **Alternative Sprint Speed Metrics:** The model currently uses Statcast's sprint speed (feet per second in the fastest one-second window). In future versions, we plan to experiment with alternative, more accessible metrics like home-to-first base time, making the model easier to apply to amateur and non-MLB players.
- **Updated Datasets:** As the MLB continues to expand Statcast tracking capabilities, incorporating data from the 2025 and later seasons will be crucial to validate and further refine the model. This will allow us to test the generalizability and stability of the intrinsic value mapping over time.

By selectively managing the input variables and relaxing restrictive modeling assumptions, future iterations of this work aim to further enhance the fairness, accuracy, and applicability of intrinsic value models in baseball analytics.

## 9. References

### References


[1] William Melville. "Including Batter Sprint Speed in the Calculation of the Intrinsic Value of a Batted Ball". In: (2019).
[2] Glenn Healey. "Learning, visualizing, and assessing a model for the intrinsic value of a batted ball". In: *IEEE Access* 5 (2017), pp. 13811–13822.
[3] Glenn Healey. "Learning, Visualizing, and Exploiting a Model for the Intrinsic Value of a Batted Ball". In: *arXiv preprint arXiv:1603.00050* (2016).
[4] Steve Slowinski. "wOBA". In: *WOBA— Sabermetrics Library* (2010).
[5] *Baseball savant.* https://baseballsavant.mlb.com.
[6] James LeDoux. *pybaseball: an Open Source Package for Baseball Data Analysis.* https://github.com/jldbc/pybaseball. 2024.
[7] *wOBA Constants.* https://www.fangraphs.com/guts.aspx?type=cn.
[8] Tom M Tango, Mitchel G Lichtman, and Andrew E Dolphin. *The book: Playing the percentages in baseball.* Potomac Books, Inc., 2007.

# 10. Appendix

# MAT 421
# Final Project

[134]:
```
import pybaseball as bb
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.optimize import differential_evolution
```

## 0.1 Data Mining and Cleaning

[ ]:
```
df = bb.statcast(start_dt='2024-03-01', end_dt='2024-11-01')
```

[260]:
```
sprints = bb.statcast_sprint_speed(2024)
```

[267]:
```
df_pa = pd.read_csv('savant_data.csv')
```

[269]:
```
df.head()
```

[269]:
```
    pitch_type   game_date  release_speed  release_pos_x  release_pos_z  \
161         KC  2024-10-30           77.5          -1.11           5.65
171         KC  2024-10-30           78.7          -1.01           5.73
182         FC  2024-10-30           93.1          -1.19           5.53
192         KC  2024-10-30           78.5          -1.19            5.7
204         KC  2024-10-30           77.4          -1.23           5.78

          player_name  batter  pitcher      events              description  \
161   Buehler, Walker  657077   621111   strikeout  swinging_strike_blocked
171   Buehler, Walker  657077   621111         NaN          swinging_strike
182   Buehler, Walker  657077   621111         NaN          swinging_strike
192   Buehler, Walker  657077   621111         NaN                     ball
204   Buehler, Walker  669224   621111   strikeout          swinging_strike

    …  n_thruorder_pitcher  n_priorpa_thisgame_player_at_bat  \
161 …                    1                                 4
171 …                    1                                 4
182 …                    1                                 4
192 …                    1                                 4
204 …                    1                                 4
```

```
     pitcher_days_since_prev_game  batter_days_since_prev_game  \
161                             2                            1
171                             2                            1
182                             2                            1
192                             2                            1
204                             2                            1

     pitcher_days_until_next_game batter_days_until_next_game  \
161                          <NA>                        <NA>
171                          <NA>                        <NA>
182                          <NA>                        <NA>
192                          <NA>                        <NA>
204                          <NA>                        <NA>

    api_break_z_with_gravity api_break_x_arm api_break_x_batter_in arm_angle
161                     5.23           -1.08                  1.08      53.2
171                     5.28           -1.05                  1.05      54.2
182                     1.89           -0.53                  0.53      44.8
192                     5.16           -1.05                  1.05      51.9
204                      5.2           -1.08                  1.08      50.0

[5 rows x 113 columns]
```

[10]:
```
sprints.head()
```

[10]:
```
   last_name, first_name  player_id  team_id team position  age  \
0       Witt Jr., Bobby     677951      118   KC       SS   24
1          Rojas, Johan     679032      143  PHI       CF   23
2      De La Cruz, Elly     682829      113  CIN       SS   22
3     Fitzgerald, Tyler     666149      137   SF       SS   26
4        Clase, Jonatan     682729      141  TOR       LF   22

   competitive_runs  bolts  hp_to_1b  sprint_speed
0               298  156.0      4.10          30.5
1               176   78.0      4.24          30.1
2               249   81.0      4.21          30.0
3                99   47.0      4.30          30.0
4                20    8.0       NaN          30.0
```

[271]:
```
df_pa.head()
```

[271]:
```
   pitches  player_id        player_name  total_pitches  pitch_percent     ba  \
0      734     680776      Duran, Jarren           2844           25.8  0.285
1      721     660271     Ohtani, Shohei           2838           25.4  0.310
2      716     683002  Henderson, Gunnar           2896           24.7  0.281
3      716     543760     Semien, Marcus           2626           27.3  0.237
4      710     665742         Soto, Juan           2960           24.0  0.289
```

```
    iso  babip    slg   woba  …  batter_run_value_per_100   xobp   xslg  \
0  0.207  0.345  0.492  0.357  …                  5.521662  0.333  0.453
1  0.336  0.336  0.646  0.431  …                  9.871706  0.388  0.660
2  0.248  0.320  0.529  0.381  …                  4.344413  0.366  0.492
3  0.154  0.250  0.391  0.306  …                 -0.069832  0.319  0.391
4  0.282  0.298  0.570  0.421  …                  8.258169  0.444  0.647

   pitcher_run_value_per_100  xbadiff  xobpdiff  xslgdiff  wobadiff  \
0                  -5.521662    0.010     0.008     0.039     0.015
1                  -9.871706   -0.004    -0.006    -0.014    -0.011
2                  -4.344413   -0.002    -0.004     0.037     0.008
3                   0.069832   -0.014    -0.013     0.000    -0.007
4                  -8.258169   -0.028    -0.026    -0.077    -0.042

   swing_miss_percent  arm_angle
0                21.7       38.2
1                20.2       37.6
2                19.2       37.9
3                13.9       38.6
4                15.6       38.2

[5 rows x 70 columns]
```

[273]:
```
df.columns.values
```

[273]:
```
array(['pitch_type', 'game_date', 'release_speed', 'release_pos_x',
       'release_pos_z', 'player_name', 'batter', 'pitcher', 'events',
       'description', 'spin_dir', 'spin_rate_deprecated',
       'break_angle_deprecated', 'break_length_deprecated', 'zone', 'des',
       'game_type', 'stand', 'p_throws', 'home_team', 'away_team', 'type',
       'hit_location', 'bb_type', 'balls', 'strikes', 'game_year',
       'pfx_x', 'pfx_z', 'plate_x', 'plate_z', 'on_3b', 'on_2b', 'on_1b',
       'outs_when_up', 'inning', 'inning_topbot', 'hc_x', 'hc_y',
       'tfs_deprecated', 'tfs_zulu_deprecated', 'umpire', 'sv_id', 'vx0',
       'vy0', 'vz0', 'ax', 'ay', 'az', 'sz_top', 'sz_bot',
       'hit_distance_sc', 'launch_speed', 'launch_angle',
       'effective_speed', 'release_spin_rate', 'release_extension',
       'game_pk', 'fielder_2', 'fielder_3', 'fielder_4', 'fielder_5',
       'fielder_6', 'fielder_7', 'fielder_8', 'fielder_9',
       'release_pos_y', 'estimated_ba_using_speedangle',
       'estimated_woba_using_speedangle', 'woba_value', 'woba_denom',
       'babip_value', 'iso_value', 'launch_speed_angle', 'at_bat_number',
       'pitch_number', 'pitch_name', 'home_score', 'away_score',
       'bat_score', 'fld_score', 'post_away_score', 'post_home_score',
       'post_bat_score', 'post_fld_score', 'if_fielding_alignment',
       'of_fielding_alignment', 'spin_axis', 'delta_home_win_exp',
```

```
        'delta_run_exp', 'bat_speed', 'swing_length',
        'estimated_slg_using_speedangle', 'delta_pitcher_run_exp',
        'hyper_speed', 'home_score_diff', 'bat_score_diff', 'home_win_exp',
        'bat_win_exp', 'age_pit_legacy', 'age_bat_legacy', 'age_pit',
        'age_bat', 'n_thruorder_pitcher',
        'n_priorpa_thisgame_player_at_bat', 'pitcher_days_since_prev_game',
        'batter_days_since_prev_game', 'pitcher_days_until_next_game',
        'batter_days_until_next_game', 'api_break_z_with_gravity',
        'api_break_x_arm', 'api_break_x_batter_in', 'arm_angle'],
       dtype=object)
```

[275]:
```
df['events'].unique()
```

[275]:
```
array(['strikeout', nan, 'field_out', 'walk', 'single', 'double',
       'sac_fly', 'catcher_interf', 'force_out', 'hit_by_pitch',
       'fielders_choice', 'field_error', 'home_run',
       'grounded_into_double_play', 'double_play',
       'strikeout_double_play', 'fielders_choice_out', 'truncated_pa',
       'sac_bunt', 'triple', 'triple_play', 'sac_fly_double_play'],
      dtype=object)
```

[277]:
```
df['description'].unique()
```

[277]:
```
array(['swinging_strike_blocked', 'swinging_strike', 'ball', 'foul',
       'called_strike', 'hit_into_play', 'blocked_ball', 'foul_tip',
       'foul_bunt', 'hit_by_pitch', 'missed_bunt', 'bunt_foul_tip',
       'pitchout'], dtype=object)
```

[279]:
```
df['woba_value'].unique()
```

[279]:
```
<FloatingArray>
[0.0, <NA>, 0.7, 0.9, 1.25, 2.0, 0.2, 1.6]
Length: 8, dtype: Float64
```

[281]:
```
df = df.dropna(subset = ['estimated_woba_using_speedangle'])
```

[282]:
```
df.head()
```

[282]:
```
    pitch_type   game_date  release_speed  release_pos_x  release_pos_z  \
161         KC  2024-10-30           77.5          -1.11           5.65
204         KC  2024-10-30           77.4          -1.23           5.78
285         KC  2024-10-30           77.6          -1.08           5.75
263         ST  2024-10-30           79.4          -1.48           5.81
276         CU  2024-10-30           75.5          -1.14           6.05

         player_name  batter  pitcher     events              description  \
161  Buehler, Walker  657077   621111  strikeout  swinging_strike_blocked
204  Buehler, Walker  669224   621111  strikeout          swinging_strike
```

```
285   Buehler, Walker  683011   621111  field_out            hit_into_play
263  Leiter Jr., Mark  669257   643410  field_out            hit_into_play
276  Leiter Jr., Mark  669242   643410  strikeout          swinging_strike

     …  n_thruorder_pitcher  n_priorpa_thisgame_player_at_bat  \
161  …                    1                                 4
204  …                    1                                 4
285  …                    1                                 4
263  …                    1                                 4
276  …                    1                                 4

     pitcher_days_since_prev_game  batter_days_since_prev_game  \
161                             2                            1
204                             2                            1
285                             2                            1
263                             1                            1
276                             1                            1

     pitcher_days_until_next_game batter_days_until_next_game  \
161                          <NA>                        <NA>
204                          <NA>                        <NA>
285                          <NA>                        <NA>
263                          <NA>                        <NA>
276                          <NA>                        <NA>

    api_break_z_with_gravity api_break_x_arm api_break_x_batter_in arm_angle
161                     5.23           -1.08                  1.08      53.2
204                      5.2           -1.08                  1.08      50.0
285                     5.33           -1.08                 -1.08      53.9
263                      4.2           -0.91                 -0.91      45.7
276                     5.52            -0.6                   0.6      54.4

[5 rows x 113 columns]
```

[ ]:
```
valid_desc = {'hit_into_play'}

df = df[df['description'].isin(valid_desc)]

df
```

[287]:
```
df['events'].unique()
```

[287]:
```
array(['field_out', 'single', 'double', 'sac_fly', 'force_out',
       'fielders_choice', 'field_error', 'home_run',
       'grounded_into_double_play', 'double_play', 'fielders_choice_out',
       'triple', 'triple_play', 'sac_fly_double_play'], dtype=object)
```

[289]:
```
df['hla'] = np.degrees(np.arctan2(df['hc_x'] - 128, 208 - df['hc_y']))
```

[291]:
```
df['hla'] = np.clip(df['hla'], -45, 45)
```

[293]:
```
df['hla'].describe()
```

[293]:
```
count     125469.0
mean     -1.327765
std      24.875745
min          -45.0
25%      -22.52902
50%        -2.5267
75%      20.565676
max           45.0
Name: hla, dtype: Float64
```

[295]:
```
df = df[['batter', 'events', 'estimated_woba_using_speedangle', 'launch_speed',␣
↪'launch_angle', 'hla']]
```

[ ]:
```
df
```

[299]:
```
df = df.dropna(subset = ['launch_speed'])
```

[ ]:
```
df
```

[303]:
```
df = df.merge(sprints[['player_id', 'sprint_speed']], left_on = 'batter',␣
↪right_on = 'player_id', how = 'left')
```

[305]:
```
df.drop(columns=['player_id'], inplace=True)
```

[ ]:
```
df
```

[311]:
```
df = df.merge(df_pa[['player_id', 'pa']], left_on = 'batter', right_on =␣
↪'player_id', how = 'left')
```

[313]:
```
df.drop(columns=['player_id'], inplace=True)
```

[ ]:
```
df
```

[ ]:
```
df = df[df['pa'] >= 500]

df
```

[321]:
```
df.rename(columns = {'estimated_woba_using_speedangle': 'wobacon'},␣
↪inplace=True)

df.head()
```

```
/var/folders/x6/yzhxgjpj4hg4fxvk0qp6p4lm0000gn/T/ipykernel_574/2966198953.py:1:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df.rename(columns = {'estimated_woba_using_speedangle': 'wobacon'},
inplace=True)
```

[321]:

```
   batter     events  wobacon  launch_speed  launch_angle        hla  \
0  683011  field_out    0.164          92.4           -13 -35.584374
1  669257  field_out    0.436         102.7             0  -27.67665
3  606192     single     0.46          99.3             1 -23.027848
5  592450     double    0.911         100.1            12 -43.872602
6  665742  field_out    0.079          65.6            -2  24.822623

   sprint_speed   pa
0          28.6  688
1          27.4  537
3          28.6  649
5          26.8  684
6          26.8  710
```

[323]:

```python
event_mapping = {
    "field_out": "out",
    "grounded_into_double_play": "out",
    "force_out": "out",
    "sac_fly": "out",
    "sac_bunt": "out",
    "single": "single",
    "double": "double",
    "triple": "triple",
    "home_run": "home run",
    "field_error": "error",
    "fielders_choice": "out",
    "fielders_choice_out": "out",
    "sac_fly_double_play": "out",
    "double_play": "out",
    "triple_play": "out"
}

df["events"] = df["events"].replace(event_mapping)

df['events'].unique()
```

```
/var/folders/x6/yzhxgjpj4hg4fxvk0qp6p4lm0000gn/T/ipykernel_574/3338060833.py:19:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
```

```
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df["events"] = df["events"].replace(event_mapping)
```

[323]:
```
array(['out', 'single', 'double', 'home run', 'error', 'triple'],
      dtype=object)
```

[325]:
```python
df = df.dropna()
```

[466]:
```python
df.head()
```

[466]:

| | batter | events | wobacon | launch_speed | launch_angle | hla |
|---|---|---|---|---|---|---|
| 0 | 683011 | out | 0.164 | 92.4 | -13 | -35.584374 |
| 1 | 669257 | out | 0.436 | 102.7 | 0 | -27.67665 |
| 3 | 606192 | single | 0.46 | 99.3 | 1 | -23.027848 |
| 5 | 592450 | double | 0.911 | 100.1 | 12 | -43.872602 |
| 6 | 665742 | out | 0.079 | 65.6 | -2 | 24.822623 |

| | sprint_speed | pa | intrinsic_value | intrinsic_sprint_value |
|---|---|---|---|---|
| 0 | 28.6 | 688 | 0.157771 | 0.164770 |
| 1 | 27.4 | 537 | 0.479763 | 0.428729 |
| 3 | 28.6 | 649 | 0.366911 | 0.395004 |
| 5 | 26.8 | 684 | 1.188269 | 1.182413 |
| 6 | 26.8 | 710 | 0.110724 | 0.092439 |

## 0.2 Methodology

[385]:
```python
features = ['launch_speed', 'launch_angle', 'hla']

outcomes = ['out', 'single', 'double', 'triple', 'home run', 'error']
outcome_data = {}
for outcome in outcomes:
    outcome_data[outcome] = df[df['events'] == outcome][features].values
```

[332]:
```python
def vectorized_gaussian_kernel(x, train_data, sigma):

    diff = (train_data - x) / sigma
    exponent = -0.5 * np.sum(diff**2, axis=1)
    norm_const = (2 * np.pi) ** (len(x)/2) * np.prod(sigma)
    return np.exp(exponent) / norm_const

def pseudolikelihood_for_sigma_vec(train_data, val_data, sigma):

    train_data = np.asarray(train_data, dtype=float)
    val_data = np.asarray(val_data, dtype=float)
```

```
    kernel_vals = np.array([vectorized_gaussian_kernel(x, train_data, sigma)␣
↪for x in val_data])
    p_x = np.mean(kernel_vals, axis=1)

    p_x[p_x <= 0] = 1e-12
    return np.sum(np.log(p_x))

def objective_sigma(sigma, train_data, val_data):

    sigma = np.array(sigma, ndmin=1, dtype=float)
    ll = pseudolikelihood_for_sigma_vec(train_data, val_data, sigma)
    return -ll

def optimize_bandwidth_de(data, bounds, n_splits=2):

    n = data.shape[0]
    indices = np.arange(n)

    if n_splits == 2:
        splits = [indices[indices % 2 == 0], indices[indices % 2 == 1]]
    else:
        splits = np.array_split(indices, n_splits)

    best_sigma_list = []

    for split in splits:
        val_data = data[split]
        train_indices = np.setdiff1d(indices, split)
        train_data = data[train_indices]

        result = differential_evolution(
            objective_sigma,
            bounds=bounds,
            args=(train_data, val_data),
            strategy='best1bin',
            tol=1e-2,
            disp=False
        )
        best_sigma_list.append(result.x)
        print(f"Optimized sigma for split: {result.x}, log-likelihood: {-result.
↪fun}")

    best_sigma_avg = np.mean(np.array(best_sigma_list), axis=0)
    return best_sigma_avg


bounds = [
```

```
    (0.5, 10.0),
    (0.5, 10.0),
    (0.5, 10.0)
]


sigma_opt_single = optimize_bandwidth_de(outcome_data["single"], bounds,␣
 ↪n_splits=2)
print("Optimized sigma for 'single':", sigma_opt_single)
```

```
Optimized sigma for split: [3.69994843 3.51688529 4.14711023], log-likelihood:
-71762.48958648695
Optimized sigma for split: [3.61331702 3.25366344 3.97090126], log-likelihood:
-71609.46373913507
Optimized sigma for 'single': [3.65663273 3.38527437 4.05900575]
```

[334]:
```
sigma_opt_double = optimize_bandwidth_de(outcome_data["double"], bounds,␣
 ↪n_splits=2)
print("Optimized sigma for 'double':", sigma_opt_double)
```

```
Optimized sigma for split: [4.33917789 4.28577445 2.0430907 ], log-likelihood:
-20644.28909520963
Optimized sigma for split: [3.35466974 4.19737166 1.78121464], log-likelihood:
-20436.571859106727
Optimized sigma for 'double': [3.84692381 4.24157305 1.91215267]
```

[336]:
```
sigma_opt_triple = optimize_bandwidth_de(outcome_data["triple"], bounds,␣
 ↪n_splits=2)
print("Optimized sigma for 'triple':", sigma_opt_triple)
```

```
Optimized sigma for split: [5.02434209 8.04272036 1.98827873], log-likelihood:
-1840.5429132696124
Optimized sigma for split: [3.74717938 8.22788185 2.58306207], log-likelihood:
-1815.1144365732446
Optimized sigma for 'triple': [4.38576074 8.1353011  2.2856704 ]
```

[338]:
```
sigma_opt_home_run = optimize_bandwidth_de(outcome_data["home run"], bounds,␣
 ↪n_splits=2)
print("Optimized sigma for 'home run':", sigma_opt_home_run)
```

```
Optimized sigma for split: [1.69297619 2.22694148 3.03688512], log-likelihood:
-14657.2508477822
Optimized sigma for split: [1.92849506 1.95393173 2.84683821], log-likelihood:
-14572.63001381495
Optimized sigma for 'home run': [1.81073562 2.0904366  2.94186166]
```

[340]:
```
sigma_opt_error = optimize_bandwidth_de(outcome_data["error"], bounds,␣
 ↪n_splits=2)
print("Optimized sigma for 'error':", sigma_opt_error)
```

```
Optimized sigma for split: [7.40783068 8.02977344 6.3055292 ], log-likelihood:
-2893.527496582631
Optimized sigma for split: [5.50501747 9.48335938 7.58127702], log-likelihood:
-2859.4544472087064
Optimized sigma for 'error': [6.45642407 8.75656641 6.94340311]
```

[342]:
```
sigma_opt_out = optimize_bandwidth_de(outcome_data["out"], bounds, n_splits=10)
print("Optimized sigma for 'out':", sigma_opt_out)
```

```
Optimized sigma for split: [4.20424506 4.87697046 0.5        ], log-likelihood:
-47477.836270540385
Optimized sigma for split: [3.29232318 4.89528199 0.78877496], log-likelihood:
-47416.01393978855
Optimized sigma for split: [3.09340103 4.096687   1.33279142], log-likelihood:
-47439.40545627191
Optimized sigma for split: [3.98403553 4.87759414 0.5        ], log-likelihood:
-47349.47762990925
Optimized sigma for split: [2.90306979 4.8866424  0.86564331], log-likelihood:
-47441.4707817529
Optimized sigma for split: [3.9155091  4.83453321 0.54973756], log-likelihood:
-47498.324873777194
Optimized sigma for split: [4.03764532 5.52673116 0.5        ], log-likelihood:
-47510.49417967993
Optimized sigma for split: [4.01981586 4.35589647 0.5        ], log-likelihood:
-47349.986489520605
Optimized sigma for split: [3.15444924 4.24724968 1.04218562], log-likelihood:
-47405.07402249606
Optimized sigma for split: [4.06604799 4.21837924 0.58989045], log-likelihood:
-47222.124440038984
Optimized sigma for 'out': [3.66705421 4.68159657 0.71690233]
```

[344]:
```
optimized_sigma = {
    "out": sigma_opt_out,
    "single": sigma_opt_single,
    "double": sigma_opt_double,
    "triple": sigma_opt_triple,
    "home run": sigma_opt_home_run,
    "error": sigma_opt_error
}
```

[346]:
```
total_batted_balls = df.shape[0]
priors = {}
for outcome in outcomes:
    count = df[df['events'] == outcome].shape[0]
    priors[outcome] = count / total_batted_balls

weights = {
    "out": 0.0,
```

```
    "single": 0.882,
    "double": 1.254,
    "triple": 1.590,
    "home run": 2.050,
    "error": 0.92
}
```

[348]:

```python
def compute_likelihood_vector(X_batch, data, sigma):

    X_batch = np.asarray(X_batch, dtype=float)
    data = np.asarray(data, dtype=float)
    sigma = np.asarray(sigma, dtype=float)

    diff = (X_batch[:, np.newaxis, :] - data) / sigma
    exponent = -0.5 * np.sum(diff ** 2, axis=2)
    norm_const = (2 * np.pi) ** (X_batch.shape[1] / 2) * np.prod(sigma)
    kernel_vals = np.exp(exponent) / norm_const
    return np.mean(kernel_vals, axis=1)

def compute_intrinsic_values_batch(X_batch, outcome_data, priors, optimized_sigma, weights, outcomes):
    likelihoods = {}
    for outcome in outcomes:
        data = outcome_data[outcome]
        sigma = optimized_sigma[outcome]
        if data.shape[0] == 0:
            likelihoods[outcome] = np.zeros(X_batch.shape[0])
        else:
            likelihoods[outcome] = compute_likelihood_vector(X_batch, data, sigma)

    numerators = { outcome: likelihoods[outcome] * priors[outcome] for outcome in outcomes }

    total_density = np.zeros(X_batch.shape[0])
    for outcome in outcomes:
        total_density += numerators[outcome]

    posteriors = {}
    for outcome in outcomes:
        posteriors[outcome] = np.divide(
            numerators[outcome],
            total_density,
            out=np.zeros_like(numerators[outcome]),
            where=total_density != 0
        )
```

```
    I_x_batch = np.zeros(X_batch.shape[0])
    for outcome in outcomes:
        I_x_batch += weights[outcome] * posteriors[outcome]
    return I_x_batch


features = ['launch_speed', 'launch_angle', 'hla']
X = df[features].values.astype(float)

outcomes = ["out", "single", "double", "triple", "home run", "error"]


batch_size = 1000
num_batches = int(np.ceil(X.shape[0] / batch_size))

I_x_all = np.empty(X.shape[0])

for i in range(num_batches):
    start_idx = i * batch_size
    end_idx = min((i + 1) * batch_size, X.shape[0])
    X_batch = X[start_idx:end_idx]
    I_x_batch = compute_intrinsic_values_batch(X_batch, outcome_data, priors, optimized_sigma, weights, outcomes)
    I_x_all[start_idx:end_idx] = I_x_batch
    print(f"Processed batch {i + 1}/{num_batches}")

df['intrinsic_value'] = I_x_all

print(df[['events'] + features + ['intrinsic_value']].head())
```

```
Processed batch 1/56
Processed batch 2/56
Processed batch 3/56
Processed batch 4/56
Processed batch 5/56
Processed batch 6/56
Processed batch 7/56
Processed batch 8/56
Processed batch 9/56
Processed batch 10/56
Processed batch 11/56
Processed batch 12/56
Processed batch 13/56
Processed batch 14/56
Processed batch 15/56
Processed batch 16/56
Processed batch 17/56
Processed batch 18/56
```

```
Processed batch 19/56
Processed batch 20/56
Processed batch 21/56
Processed batch 22/56
Processed batch 23/56
Processed batch 24/56
Processed batch 25/56
Processed batch 26/56
Processed batch 27/56
Processed batch 28/56
Processed batch 29/56
Processed batch 30/56
Processed batch 31/56
Processed batch 32/56
Processed batch 33/56
Processed batch 34/56
Processed batch 35/56
Processed batch 36/56
Processed batch 37/56
Processed batch 38/56
Processed batch 39/56
Processed batch 40/56
Processed batch 41/56
Processed batch 42/56
Processed batch 43/56
Processed batch 44/56
Processed batch 45/56
Processed batch 46/56
Processed batch 47/56
Processed batch 48/56
Processed batch 49/56
Processed batch 50/56
Processed batch 51/56
Processed batch 52/56
Processed batch 53/56
Processed batch 54/56
Processed batch 55/56
Processed batch 56/56
   events  launch_speed  launch_angle        hla  intrinsic_value
0     out          92.4           -13 -35.584374         0.157771
1     out         102.7             0  -27.67665         0.479763
3  single          99.3             1 -23.027848         0.366911
5  double         100.1            12 -43.872602         1.188269
6     out          65.6            -2  24.822623         0.110724

/var/folders/x6/yzhxgjpj4hg4fxvk0qp6p4lm0000gn/T/ipykernel_574/1152620938.py:88:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
```

```
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['intrinsic_value'] = I_x_all
```

[350]:
```
df.loc[:, 'intrinsic_value'] = I_x_all
```

[352]:
```
df
```

[352]:
```
        batter  events  wobacon  launch_speed  launch_angle        hla  \
0       683011     out    0.164          92.4           -13 -35.584374
1       669257     out    0.436         102.7             0  -27.67665
3       606192  single     0.46          99.3             1 -23.027848
5       592450  double    0.911         100.1            12 -43.872602
6       665742     out    0.079          65.6            -2  24.822623
…          …       …       …              …             …          …
125503  660271  single    0.921         112.3            11  24.979968
125504  605141     out    0.008          85.4            46 -36.021606
125508  592518     out    0.051          86.3           -38  -9.807328
125514  669257     out    0.034          86.6            33 -27.397377
125515  660271     out    0.059          77.9           -13 -10.978791

        sprint_speed   pa  intrinsic_value
0               28.6  688         0.157771
1               27.4  537         0.479763
3               28.6  649         0.366911
5               26.8  684         1.188269
6               26.8  710         0.110724
…                  …    …                …
125503          28.1  721         0.666474
125504          26.7  513         0.008535
125508          25.8  637         0.046517
125514          27.4  537         0.019314
125515          28.1  721         0.061887

[55392 rows x 9 columns]
```

[391]:
```
features = ['launch_speed', 'launch_angle', 'hla', 'sprint_speed']

outcomes = ['out', 'single', 'double', 'triple', 'home run', 'error']
outcome_data = {}
for outcome in outcomes:
    outcome_data[outcome] = df[df['events'] == outcome][features].values
```

[356]:
```
def vectorized_gaussian_kernel(x, train_data, sigma):

    diff = (train_data - x) / sigma
```

```
    exponent = -0.5 * np.sum(diff**2, axis=1)
    norm_const = (2 * np.pi) ** (len(x) / 2) * np.prod(sigma)
    return np.exp(exponent) / norm_const

def pseudolikelihood_for_sigma_vec(train_data, val_data, sigma):

    train_data = np.asarray(train_data, dtype=float)
    val_data = np.asarray(val_data, dtype=float)

    kernel_vals = np.array([vectorized_gaussian_kernel(x, train_data, sigma) for x in val_data])
    p_x = np.mean(kernel_vals, axis=1)

    p_x[p_x <= 0] = 1e-12
    return np.sum(np.log(p_x))

def objective_sigma(sigma, train_data, val_data):
    sigma = np.array(sigma, ndmin=1, dtype=float)
    ll = pseudolikelihood_for_sigma_vec(train_data, val_data, sigma)
    return -ll

def optimize_bandwidth_de(data, bounds, n_splits=2):

    n = data.shape[0]
    indices = np.arange(n)

    if n_splits == 2:
        splits = [indices[indices % 2 == 0], indices[indices % 2 == 1]]
    else:
        splits = np.array_split(indices, n_splits)

    best_sigma_list = []

    for split in splits:
        val_data = data[split]
        train_indices = np.setdiff1d(indices, split)
        train_data = data[train_indices]

        result = differential_evolution(
            objective_sigma,
            bounds=bounds,
            args=(train_data, val_data),
            strategy='best1bin',
            tol=1e-2,
            disp=False
        )
        best_sigma_list.append(result.x)
```

```
        print(f"Optimized sigma for split: {result.x}, log-likelihood: {-result.
 ↪fun}")

    best_sigma_avg = np.mean(np.array(best_sigma_list), axis=0)
    return best_sigma_avg

bounds = [
    (0.5, 15.0),
    (0.5, 15.0),
    (0.5, 15.0),
    (0.5, 15.0)
]
```

[358]:

```
sigma_opt_single = optimize_bandwidth_de(outcome_data["single"], bounds,␣
 ↪n_splits=10)
print("Optimized sigma for 'single' with sprint speed:", sigma_opt_single)
```

```
Optimized sigma for split: [3.54215521 4.09405216 4.43586941 0.5       ], log-
likelihood: -16307.558470646723
Optimized sigma for split: [4.04041077 3.37535937 3.62328474 0.5       ], log-
likelihood: -16273.70866651015
Optimized sigma for split: [3.4050281  3.95141553 4.45062949 0.5       ], log-
likelihood: -16392.824319696276
Optimized sigma for split: [3.93566535 2.99941778 3.91218355 0.52886941], log-
likelihood: -16310.098593578146
Optimized sigma for split: [3.66414977 3.43585823 4.28241014 0.5       ], log-
likelihood: -16350.077549968533
Optimized sigma for split: [3.96073172 3.0612706  4.25320925 0.5       ], log-
likelihood: -16355.757982025429
Optimized sigma for split: [3.16674322 3.49931532 4.26378455 0.5       ], log-
likelihood: -16184.71996401293
Optimized sigma for split: [3.83440681 3.07424611 4.31359158 0.5       ], log-
likelihood: -16296.641431812026
Optimized sigma for split: [3.61706955 3.59958329 4.43041484 0.5       ], log-
likelihood: -16250.493957249939
Optimized sigma for split: [3.5601729  3.40768817 3.4064706  0.5034722 ], log-
likelihood: -16260.583271443968
Optimized sigma for 'single' with sprint speed: [3.67265334 3.44982066
4.13718482 0.50323416]
```

[360]:

```
sigma_opt_double = optimize_bandwidth_de(outcome_data["double"], bounds,␣
 ↪n_splits=10)
print("Optimized sigma for 'double' with sprint speed:", sigma_opt_double)
```

```
Optimized sigma for split: [3.70692688 4.87650602 1.91439425 0.66904604], log-
likelihood: -4739.320553416304
Optimized sigma for split: [2.91512155 3.34914905 2.87150213 0.66177721], log-
likelihood: -4742.425522508311
```

```
Optimized sigma for split: [4.66232848 3.15637667 1.2955512  0.56407048], log-
likelihood: -4665.907555350736
Optimized sigma for split: [3.92115318 4.03648552 1.27773328 0.85228114], log-
likelihood: -4685.591113942409
Optimized sigma for split: [3.93267412 4.6452309  1.9531095  0.5       ], log-
likelihood: -4701.839746708607
Optimized sigma for split: [2.83737594 3.96480387 2.92940168 0.56673887], log-
likelihood: -4707.048137088212
Optimized sigma for split: [3.70109028 4.34472805 1.81481732 0.58237186], log-
likelihood: -4701.884993261617
Optimized sigma for split: [3.13429011 2.44950403 1.95285404 0.73696115], log-
likelihood: -4656.824005401779
Optimized sigma for split: [3.37594426 3.78182778 1.96369205 0.56305315], log-
likelihood: -4672.590869828552
Optimized sigma for split: [3.45362907 4.31208382 1.87753826 0.58749477], log-
likelihood: -4720.264452517124
Optimized sigma for 'double' with sprint speed: [3.56405339 3.89166957
1.98505937 0.62837947]
```

[362]:

```
sigma_opt_triple = optimize_bandwidth_de(outcome_data["triple"], bounds,␣
↪n_splits=10)
print("Optimized sigma for 'triple' with sprint speed:", sigma_opt_triple)
```

```
Optimized sigma for split: [4.91040057 6.0045607  1.61952574 0.57718948], log-
likelihood: -417.08028098339884
Optimized sigma for split: [2.85446605 3.21137997 3.95067218 0.6837418 ], log-
likelihood: -410.6723343123922
Optimized sigma for split: [4.74065536 5.71920455 1.56378782 0.5       ], log-
likelihood: -416.00543016121867
Optimized sigma for split: [4.28873496 4.62250985 3.41756332 0.5       ], log-
likelihood: -398.11677124430616
Optimized sigma for split: [3.6057936  5.26030657 4.59462015 0.68542245], log-
likelihood: -410.6259946603892
Optimized sigma for split: [2.08148515 6.88462585 2.44401914 1.07139346], log-
likelihood: -407.6531874590729
Optimized sigma for split: [3.36705797 9.94777424 3.10024772 0.65600494], log-
likelihood: -425.042326902802
Optimized sigma for split: [3.57332372 5.17746034 3.15585836 0.66395538], log-
likelihood: -402.37238297140993
Optimized sigma for split: [5.01759889 3.04590193 2.25283074 0.75323328], log-
likelihood: -400.99692015847836
Optimized sigma for split: [2.3890763  6.78798664 3.58088611 0.59864744], log-
likelihood: -407.29447665380314
Optimized sigma for 'triple' with sprint speed: [3.68285926 5.66617106
2.96800113 0.66895882]
```

[364]:

```
sigma_opt_home_run = optimize_bandwidth_de(outcome_data["home run"], bounds,␣
↪n_splits=10)
```

```
print("Optimized sigma for 'home run' with sprint speed:", sigma_opt_home_run)
```

```
Optimized sigma for split: [1.59879571 2.37756433 3.66908802 0.5       ], log-
likelihood: -3403.8091545249818
Optimized sigma for split: [1.80198724 1.91475375 3.68804927 0.5       ], log-
likelihood: -3381.66904474657
Optimized sigma for split: [1.70367083 2.04342026 2.43508024 0.579562  ], log-
likelihood: -3392.7277557011994
Optimized sigma for split: [1.88251266 2.30846424 4.42190617 0.5       ], log-
likelihood: -3416.945445098091
Optimized sigma for split: [2.09188902 1.92932161 3.21563347 0.5       ], log-
likelihood: -3391.289543542679
Optimized sigma for split: [1.47996632 2.50120229 3.9834139  0.5       ], log-
likelihood: -3410.4604464064178
Optimized sigma for split: [1.72977538 2.05221816 2.48917978 0.5       ], log-
likelihood: -3368.2129907010003
Optimized sigma for split: [1.53990196 1.80066536 4.72897237 0.5       ], log-
likelihood: -3366.368473397035
Optimized sigma for split: [1.88524758 2.15812401 3.29934142 0.5       ], log-
likelihood: -3362.311398659784
Optimized sigma for split: [1.25544185 2.39535767 4.49888966 0.5       ], log-
likelihood: -3378.9440315637244
Optimized sigma for 'home run' with sprint speed: [1.69691886 2.14810917
3.64295543 0.5079562 ]
```

[366]:
```
sigma_opt_error = optimize_bandwidth_de(outcome_data["error"], bounds,␣
↪n_splits=10)
print("Optimized sigma for 'error' with sprint speed:", sigma_opt_error)
```

```
Optimized sigma for split: [7.24034991 5.3589389  4.47278235 0.77324769], log-
likelihood: -638.5718661798749
Optimized sigma for split: [8.77808921 3.28021332 8.79874456 1.03545601], log-
likelihood: -656.1746254557886
Optimized sigma for split: [ 8.36814909 11.28627858  9.15105686  0.5       ],
log-likelihood: -658.6840118307925
Optimized sigma for split: [9.39323877 6.14354917 6.82474889 0.5       ], log-
likelihood: -649.7999710869435
Optimized sigma for split: [5.08216109 6.51429418 5.01111869 1.03207619], log-
likelihood: -642.7221619927184
Optimized sigma for split: [10.17186144  4.57392265  6.90554103  0.5       ],
log-likelihood: -656.8540655075122
Optimized sigma for split: [ 7.82756083 11.63655899  3.72102398  0.61629833],
log-likelihood: -654.5056239300696
Optimized sigma for split: [6.07041946 5.05280632 9.05229689 0.90416072], log-
likelihood: -650.2685901257121
Optimized sigma for split: [5.52317642 9.57660188 8.12060228 0.71701124], log-
likelihood: -642.7030046589424
Optimized sigma for split: [ 5.65510267 10.35905777  1.35421886  0.9605762 ],
```

```
log-likelihood: -630.6959970996093
Optimized sigma for 'error' with sprint speed: [7.41101089 7.37822218 6.34121344
0.75388264]
```

[ ]:
```
sigma_opt_out = optimize_bandwidth_de(outcome_data["out"], bounds, n_splits=20)
print("Optimized sigma for 'out' with sprint speed:", sigma_opt_out)
```

[372]:
```
optimized_sigma_sprints = {
    "out": np.array([3.52, 4.71, 1.83, 0.59558604]),
    "single": sigma_opt_single,
    "double": sigma_opt_double,
    "triple": sigma_opt_triple,
    "home run": sigma_opt_home_run,
    "error": sigma_opt_error
}
```

[374]:
```
total_batted_balls = df.shape[0]
priors = {}
for outcome in outcomes:
    count = df[df['events'] == outcome].shape[0]
    priors[outcome] = count / total_batted_balls

weights = {
    "out": 0.0,
    "single": 0.882,
    "double": 1.254,
    "triple": 1.590,
    "home run": 2.050,
    "error": 0.92
}
```

[376]:
```
def compute_likelihood_vector(X_batch, data, sigma):
    X_batch = np.asarray(X_batch, dtype=float)
    data = np.asarray(data, dtype=float)
    sigma = np.asarray(sigma, dtype=float)

    diff = (X_batch[:, None, :] - data[None, :, :]) / sigma
    exponent = -0.5 * np.sum(diff ** 2, axis=2)
    norm_const = (2 * np.pi) ** (X_batch.shape[1] / 2) * np.prod(sigma)
    kernel_vals = np.exp(exponent) / norm_const
    return np.mean(kernel_vals, axis=1)

def compute_intrinsic_values_batch(X_batch, outcome_data, priors,␣
 ↪optimized_sigma, weights, outcomes):
    likelihoods = {}

    for outcome in outcomes:
```

```
        data = outcome_data.get(outcome)
        sigma = optimized_sigma.get(outcome)

        if data is None or len(data) == 0:
            likelihoods[outcome] = np.zeros(X_batch.shape[0])
        else:
            likelihoods[outcome] = compute_likelihood_vector(X_batch, data,␣
↪sigma)

    numerators = {outcome: likelihoods[outcome] * priors[outcome] for outcome␣
↪in outcomes}

    px = np.sum(list(numerators.values()), axis=0)
    px[px <= 0] = 1e-12

    posteriors = {outcome: numerators[outcome] / px for outcome in outcomes}

    I_sx = np.sum([weights[outcome] * posteriors[outcome] for outcome in␣
↪outcomes], axis=0)

    return I_sx

features = ['launch_speed', 'launch_angle', 'hla', 'sprint_speed']
X = df[features].values.astype(float)

outcomes = ["out", "single", "double", "triple", "home run", "error"]

batch_size = 1000
num_batches = int(np.ceil(X.shape[0] / batch_size))

I_sx_all = np.empty(X.shape[0])

for i in range(num_batches):
    start_idx = i * batch_size
    end_idx = min((i + 1) * batch_size, X.shape[0])
    X_batch = X[start_idx:end_idx]

    I_sx_batch = compute_intrinsic_values_batch(X_batch, outcome_data, priors,␣
↪optimized_sigma_sprints, weights, outcomes)
    I_sx_all[start_idx:end_idx] = I_sx_batch

    print(f"Processed batch {i + 1}/{num_batches}")

df['intrinsic_sprint_value'] = I_sx_all

print(df[['events'] + features + ['intrinsic_sprint_value']].head())
```

```
Processed batch 1/56
Processed batch 2/56
Processed batch 3/56
Processed batch 4/56
Processed batch 5/56
Processed batch 6/56
Processed batch 7/56
Processed batch 8/56
Processed batch 9/56
Processed batch 10/56
Processed batch 11/56
Processed batch 12/56
Processed batch 13/56
Processed batch 14/56
Processed batch 15/56
Processed batch 16/56
Processed batch 17/56
Processed batch 18/56
Processed batch 19/56
Processed batch 20/56
Processed batch 21/56
Processed batch 22/56
Processed batch 23/56
Processed batch 24/56
Processed batch 25/56
Processed batch 26/56
Processed batch 27/56
Processed batch 28/56
Processed batch 29/56
Processed batch 30/56
Processed batch 31/56
Processed batch 32/56
Processed batch 33/56
Processed batch 34/56
Processed batch 35/56
Processed batch 36/56
Processed batch 37/56
Processed batch 38/56
Processed batch 39/56
Processed batch 40/56
Processed batch 41/56
Processed batch 42/56
Processed batch 43/56
Processed batch 44/56
Processed batch 45/56
Processed batch 46/56
Processed batch 47/56
Processed batch 48/56
```

```
Processed batch 49/56
Processed batch 50/56
Processed batch 51/56
Processed batch 52/56
Processed batch 53/56
Processed batch 54/56
Processed batch 55/56
Processed batch 56/56
   events  launch_speed  launch_angle        hla  sprint_speed  \
0     out          92.4           -13 -35.584374          28.6
1     out         102.7             0  -27.67665          27.4
3  single          99.3             1 -23.027848          28.6
5  double         100.1            12 -43.872602          26.8
6     out          65.6            -2  24.822623          26.8

   intrinsic_sprint_value
0                0.164770
1                0.428729
3                0.395004
5                1.182413
6                0.092439
/var/folders/x6/yzhxgjpj4hg4fxvk0qp6p4lm0000gn/T/ipykernel_574/1833385578.py:67:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['intrinsic_sprint_value'] = I_sx_all
```

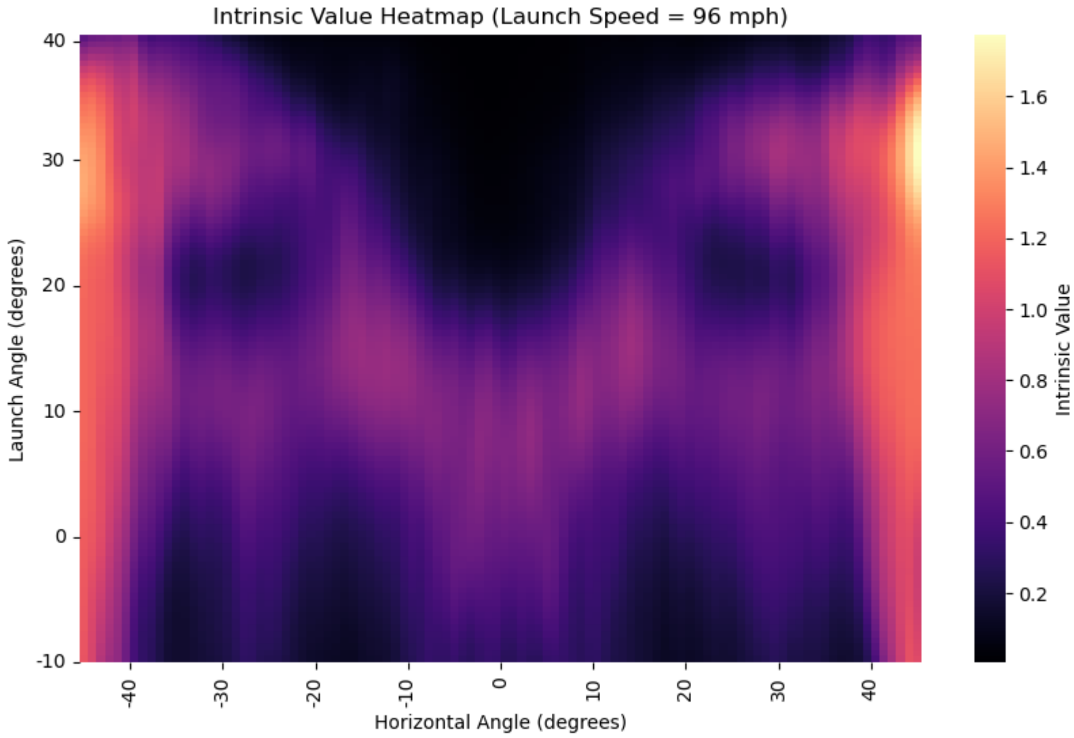
[378]:
```
df.loc[:, 'intrinsic_sprint_value'] = I_sx_all

df.head()
```

[378]:
```
   batter  events  wobacon  launch_speed  launch_angle        hla  \
0  683011     out    0.164          92.4           -13 -35.584374
1  669257     out    0.436         102.7             0  -27.67665
3  606192  single     0.46          99.3             1 -23.027848
5  592450  double    0.911         100.1            12 -43.872602
6  665742     out    0.079          65.6            -2  24.822623

   sprint_speed   pa  intrinsic_value  intrinsic_sprint_value
0          28.6  688         0.157771                0.164770
1          27.4  537         0.479763                0.428729
3          28.6  649         0.366911                0.395004
5          26.8  684         1.188269                1.182413
6          26.8  710         0.110724                0.092439
```

## 0.3 Visualizing

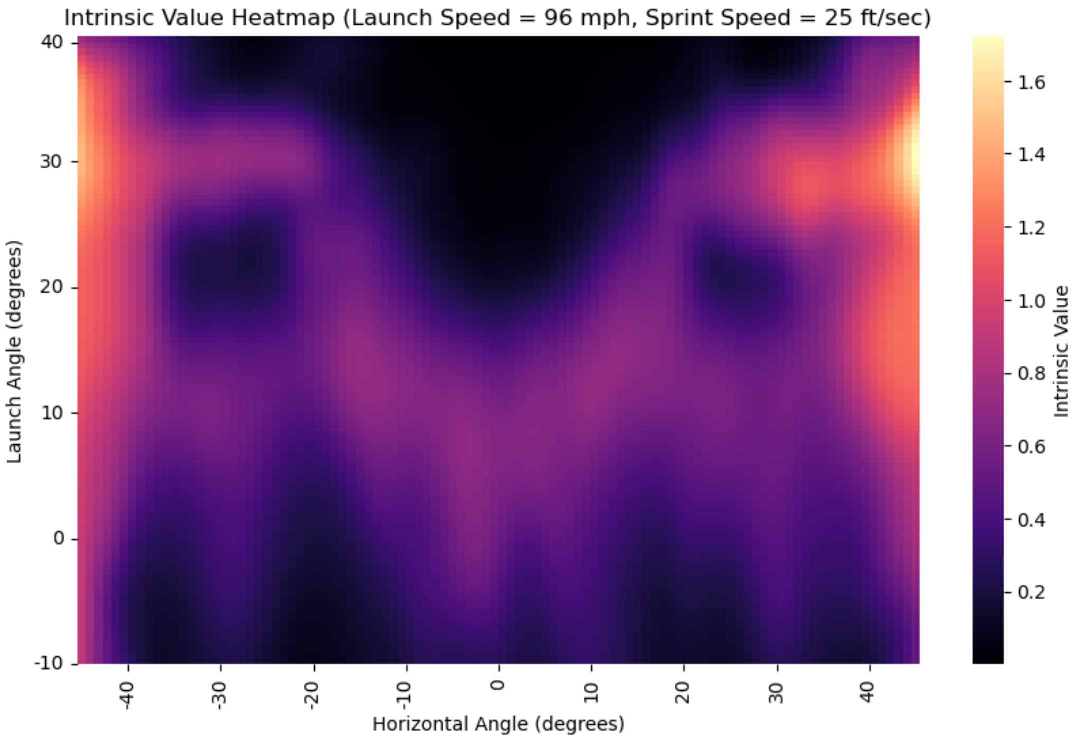
[381]:
```
launch_speed_fixed = 96
```

[387]:
```
horizontal_angles = np.linspace(-45, 45, 100)
launch_angles = np.linspace(-10, 40, 100)
launch_speed_fixed = 96

H, L = np.meshgrid(horizontal_angles, launch_angles)

grid_points = np.column_stack((np.full(H.shape, launch_speed_fixed).ravel(),
                               L.ravel(),
                               H.ravel()))

I_s_grid = np.array([
    compute_intrinsic_values_batch(x.reshape(1, -1), outcome_data, priors,␣
 ↪optimized_sigma, weights, outcomes)
    for x in grid_points
])

I_s_grid = I_s_grid.reshape(H.shape)

plt.figure(figsize=(10, 6))
ax = sns.heatmap(I_s_grid, cmap="magma", cbar_kws={'label': 'Intrinsic Value'},
                 xticklabels=np.round(horizontal_angles, 1), yticklabels=np.
 ↪round(launch_angles, 1))

ax.set_xticks([np.where(horizontal_angles >= x)[0][0] for x in [-40, -30, -20,␣
 ↪-10, 0, 10, 20, 30, 40]])
ax.set_xticklabels([-40, -30, -20, -10, 0, 10, 20, 30, 40])

ax.set_yticks([np.where(launch_angles >= y)[0][0] for y in [-10, 0, 10, 20, 30,␣
 ↪40]])
ax.set_yticklabels([-10, 0, 10, 20, 30, 40])

plt.gca().invert_yaxis()

plt.xlabel("Horizontal Angle (degrees)")
plt.ylabel("Launch Angle (degrees)")
plt.title("Intrinsic Value Heatmap (Launch Speed = 96 mph)")
plt.show()
```



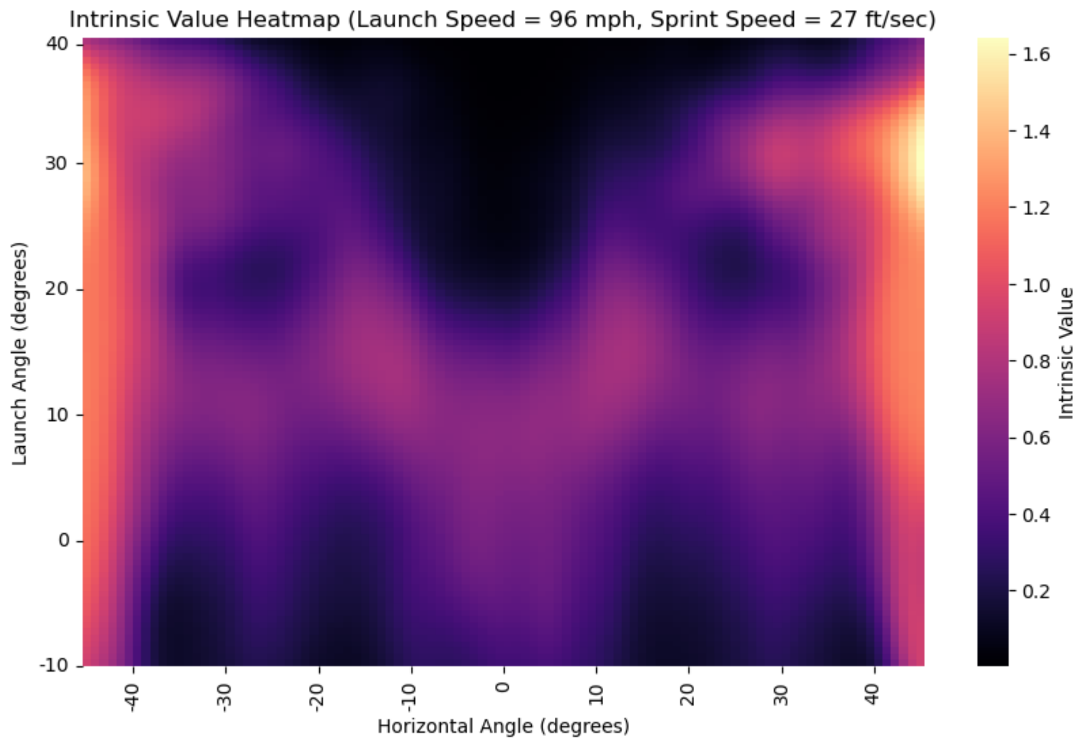

[393]:
```python
fixed_sprint_speed = 25

H, L = np.meshgrid(horizontal_angles, launch_angles)
grid_points = np.column_stack((np.full(H.shape, launch_speed_fixed).ravel(),
                               L.ravel(), H.ravel(),
                               np.full(H.shape, fixed_sprint_speed).ravel()))

I_s_grid = np.array([compute_intrinsic_values_batch(x.reshape(1, -1),␣
 ↪outcome_data, priors, optimized_sigma_sprints, weights, outcomes)
                     for x in grid_points])
I_s_grid = I_s_grid.reshape(H.shape)

plt.figure(figsize=(10, 6))
ax = sns.heatmap(I_s_grid, cmap="magma", cbar_kws={'label': 'Intrinsic Value'},
                 xticklabels=np.round(horizontal_angles, 1), yticklabels=np.
 ↪round(launch_angles, 1))

ax.set_xticks([np.where(horizontal_angles >= x)[0][0] for x in [-40, -30, -20,␣
 ↪-10, 0, 10, 20, 30, 40]])
ax.set_xticklabels([-40, -30, -20, -10, 0, 10, 20, 30, 40])
```

```
ax.set_yticks([np.where(launch_angles >= y)[0][0] for y in [-10, 0, 10, 20, 30,␣
↪40]])
ax.set_yticklabels([-10, 0, 10, 20, 30, 40])

plt.gca().invert_yaxis()

plt.xlabel("Horizontal Angle (degrees)")
plt.ylabel("Launch Angle (degrees)")
plt.title("Intrinsic Value Heatmap (Launch Speed = 96 mph, Sprint Speed = 25 ft/
↪sec)")
plt.show()
```

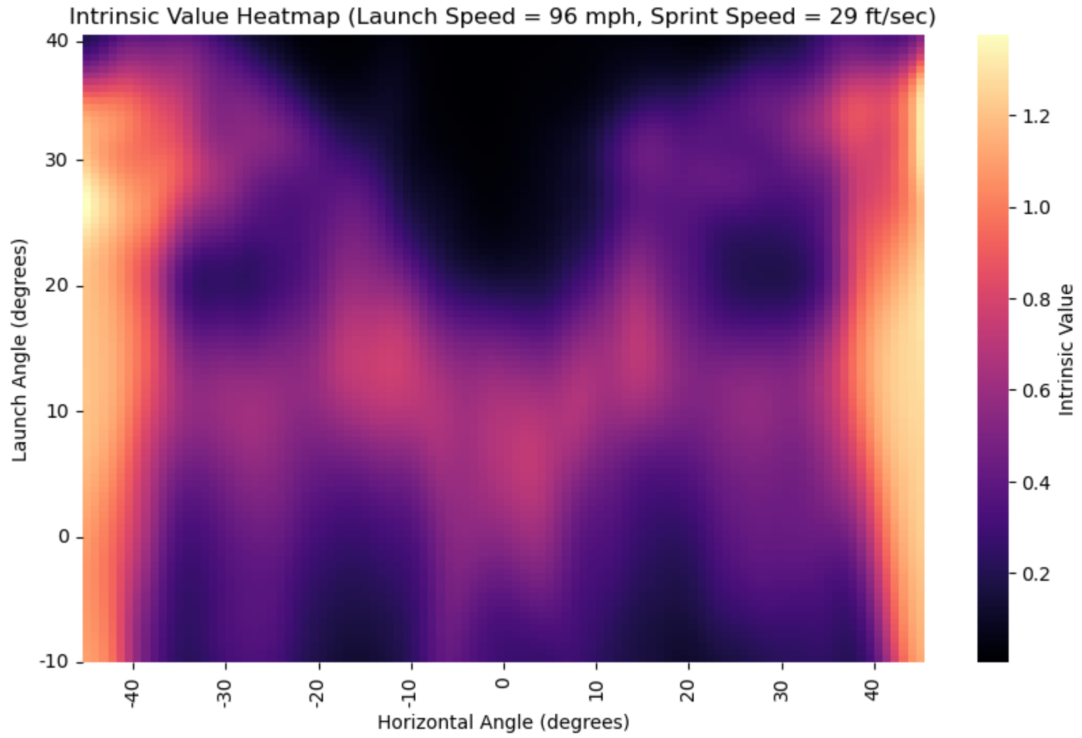



[395]:
```
fixed_sprint_speed = 27

H, L = np.meshgrid(horizontal_angles, launch_angles)
grid_points = np.column_stack((np.full(H.shape, launch_speed_fixed).ravel(),
                               L.ravel(), H.ravel(),
                               np.full(H.shape, fixed_sprint_speed).ravel()))

I_s_grid = np.array([compute_intrinsic_values_batch(x.reshape(1, -1),␣
↪outcome_data, priors, optimized_sigma_sprints, weights, outcomes)
                     for x in grid_points])
```

```
I_s_grid = I_s_grid.reshape(H.shape)

plt.figure(figsize=(10, 6))
ax = sns.heatmap(I_s_grid, cmap="magma", cbar_kws={'label': 'Intrinsic Value'},
                 xticklabels=np.round(horizontal_angles, 1), yticklabels=np.
 ↪round(launch_angles, 1))

ax.set_xticks([np.where(horizontal_angles >= x)[0][0] for x in [-40, -30, -20,␣
 ↪-10, 0, 10, 20, 30, 40]])
ax.set_xticklabels([-40, -30, -20, -10, 0, 10, 20, 30, 40])

ax.set_yticks([np.where(launch_angles >= y)[0][0] for y in [-10, 0, 10, 20, 30,␣
 ↪40]])
ax.set_yticklabels([-10, 0, 10, 20, 30, 40])

plt.gca().invert_yaxis()

plt.xlabel("Horizontal Angle (degrees)")
plt.ylabel("Launch Angle (degrees)")
plt.title("Intrinsic Value Heatmap (Launch Speed = 96 mph, Sprint Speed = 27 ft/
 ↪sec)")
plt.show()
```



[397]:
```
fixed_sprint_speed = 29

H, L = np.meshgrid(horizontal_angles, launch_angles)
grid_points = np.column_stack((np.full(H.shape, launch_speed_fixed).ravel(),
                               L.ravel(), H.ravel(),
                               np.full(H.shape, fixed_sprint_speed).ravel()))

I_s_grid = np.array([compute_intrinsic_values_batch(x.reshape(1, -1),␣
↪outcome_data, priors, optimized_sigma_sprints, weights, outcomes)
                     for x in grid_points])
I_s_grid = I_s_grid.reshape(H.shape)

plt.figure(figsize=(10, 6))
ax = sns.heatmap(I_s_grid, cmap="magma", cbar_kws={'label': 'Intrinsic Value'},
                 xticklabels=np.round(horizontal_angles, 1), yticklabels=np.
↪round(launch_angles, 1))

ax.set_xticks([np.where(horizontal_angles >= x)[0][0] for x in [-40, -30, -20,␣
↪-10, 0, 10, 20, 30, 40]])
ax.set_xticklabels([-40, -30, -20, -10, 0, 10, 20, 30, 40])

ax.set_yticks([np.where(launch_angles >= y)[0][0] for y in [-10, 0, 10, 20, 30,␣
↪40]])
ax.set_yticklabels([-10, 0, 10, 20, 30, 40])

plt.gca().invert_yaxis()

plt.xlabel("Horizontal Angle (degrees)")
plt.ylabel("Launch Angle (degrees)")
plt.title("Intrinsic Value Heatmap (Launch Speed = 96 mph, Sprint Speed = 29 ft/
↪sec)")
plt.show()
```




## 0.4 O - I Comparisons

[456]:
```
batter_avg_stats = df.groupby('batter').agg({
    'wobacon': 'mean',
    'intrinsic_value': 'mean',
    'intrinsic_sprint_value': 'mean'
}).reset_index()

batter_avg_stats.rename(columns={
    'wobacon': 'avg_wobacon',
    'intrinsic_value': 'avg_intrinsic_no_sprint',
    'intrinsic_sprint_value': 'avg_intrinsic_with_sprint'
}, inplace=True)

batter_avg_stats.head()
```

[456]:

| | batter | avg_wobacon | avg_intrinsic_no_sprint | avg_intrinsic_with_sprint |
|---|---|---|---|---|
| 0 | 457705 | 0.389332 | 0.376817 | 0.374776 |
| 1 | 457759 | 0.343304 | 0.341312 | 0.336370 |
| 2 | 467793 | 0.348392 | 0.344153 | 0.336967 |
| 3 | 502671 | 0.422585 | 0.417799 | 0.415931 |
| 4 | 514888 | 0.350911 | 0.369539 | 0.371067 |

[458]:
```
batter_avg_stats = batter_avg_stats.merge(sprints[['player_id', 'sprint_speed',␣
↪'last_name, first_name']], left_on = 'batter', right_on = 'player_id', how =␣
↪'left')

batter_avg_stats.drop(columns=['player_id'], inplace=True)

batter_avg_stats.head()
```

[458]:
```
   batter  avg_wobacon  avg_intrinsic_no_sprint  avg_intrinsic_with_sprint  \
0  457705     0.389332                 0.376817                   0.374776
1  457759     0.343304                 0.341312                   0.336370
2  467793     0.348392                 0.344153                   0.336967
3  502671     0.422585                 0.417799                   0.415931
4  514888     0.350911                 0.369539                   0.371067

   sprint_speed last_name, first_name
0          26.8      McCutchen, Andrew
1          25.4         Turner, Justin
2          25.9        Santana, Carlos
3          26.3      Goldschmidt, Paul
4          27.1           Altuve, Jose
```

[460]:
```
batter_avg_stats['O - Ins'] = (
    batter_avg_stats['avg_wobacon'] -␣
↪batter_avg_stats['avg_intrinsic_no_sprint']
)

batter_avg_stats['O - Is'] = (
    batter_avg_stats['avg_wobacon'] -␣
↪batter_avg_stats['avg_intrinsic_with_sprint']
)

batter_avg_stats.head()
```

[460]:
```
   batter  avg_wobacon  avg_intrinsic_no_sprint  avg_intrinsic_with_sprint  \
0  457705     0.389332                 0.376817                   0.374776
1  457759     0.343304                 0.341312                   0.336370
2  467793     0.348392                 0.344153                   0.336967
3  502671     0.422585                 0.417799                   0.415931
4  514888     0.350911                 0.369539                   0.371067

   sprint_speed last_name, first_name    O - Ins     O - Is
0          26.8      McCutchen, Andrew  0.012515   0.014556
1          25.4         Turner, Justin  0.001991   0.006934
2          25.9        Santana, Carlos   0.00424   0.011425
3          26.3      Goldschmidt, Paul  0.004786   0.006653
4          27.1           Altuve, Jose -0.018629  -0.020157
```

```
[464]: batter_avg_stats.nlargest(10, 'O - Ins')
```

[464]:

| | batter | avg_wobacon | avg_intrinsic_no_sprint | avg_intrinsic_with_sprint |
|---|---|---|---|---|
| 98 | 677594 | 0.437361 | 0.401149 | 0.411661 |
| 53 | 660271 | 0.549981 | 0.515068 | 0.526303 |
| 78 | 667670 | 0.510167 | 0.475674 | 0.480757 |
| 72 | 665833 | 0.457889 | 0.423814 | 0.440115 |
| 95 | 673548 | 0.437248 | 0.404069 | 0.413643 |
| 108 | 681351 | 0.442262 | 0.410368 | 0.422478 |
| 41 | 646240 | 0.456199 | 0.425153 | 0.423284 |
| 91 | 671218 | 0.43368 | 0.402767 | 0.414896 |
| 119 | 686668 | 0.411528 | 0.381968 | 0.395627 |
| 50 | 656941 | 0.489728 | 0.460323 | 0.464017 |

| | sprint_speed | last_name, first_name | O - Ins | O - Is |
|---|---|---|---|---|
| 98 | 29.6 | Rodríguez, Julio | 0.036211 | 0.0257 |
| 53 | 28.1 | Ohtani, Shohei | 0.034913 | 0.023677 |
| 78 | 27.6 | Rooker, Brent | 0.034493 | 0.02941 |
| 72 | 28.8 | Cruz, Oneil | 0.034076 | 0.017775 |
| 95 | 28.3 | Suzuki, Seiya | 0.033179 | 0.023604 |
| 108 | 28.1 | O'Hoppe, Logan | 0.031893 | 0.019784 |
| 41 | 26.5 | Devers, Rafael | 0.031046 | 0.032915 |
| 91 | 27.9 | Ramos, Heliot | 0.030913 | 0.018784 |
| 119 | 29.3 | Doyle, Brenton | 0.02956 | 0.015901 |
| 50 | 25.8 | Schwarber, Kyle | 0.029406 | 0.025711 |

```
[408]: batter_avg_stats.nlargest(10, 'O - Is')
```

[408]:

| | batter | avg_wobacon | avg_intrinsic_no_sprint | avg_intrinsic_with_sprint |
|---|---|---|---|---|
| 6 | 521692 | 0.426331 | 0.410997 | 0.383469 |
| 7 | 542303 | 0.506937 | 0.482676 | 0.473796 |
| 41 | 646240 | 0.456199 | 0.425153 | 0.423284 |
| 75 | 666624 | 0.380939 | 0.353002 | 0.348151 |
| 78 | 667670 | 0.510167 | 0.475674 | 0.480757 |
| 54 | 660821 | 0.432917 | 0.407973 | 0.404122 |
| 50 | 656941 | 0.489728 | 0.460323 | 0.464017 |
| 98 | 677594 | 0.437361 | 0.401149 | 0.411661 |
| 23 | 606466 | 0.444378 | 0.424790 | 0.419031 |
| 53 | 660271 | 0.549981 | 0.515068 | 0.526303 |

| | sprint_speed | last_name, first_name | O - Ins | O - Is |
|---|---|---|---|---|
| 6 | 24.5 | Perez, Salvador | 0.015333 | 0.042862 |
| 7 | 25.7 | Ozuna, Marcell | 0.02426 | 0.033141 |
| 41 | 26.5 | Devers, Rafael | 0.031046 | 0.032915 |
| 75 | 27.3 | Morel, Christopher | 0.027937 | 0.032788 |
| 78 | 27.6 | Rooker, Brent | 0.034493 | 0.02941 |
| 54 | 27.2 | Sánchez, Jesús | 0.024944 | 0.028794 |

```
50          25.8      Schwarber, Kyle  0.029406  0.025711
98          29.6    Rodríguez, Julio  0.036211    0.0257
23          27.1        Marte, Ketel  0.019588  0.025346
53          28.1       Ohtani, Shohei  0.034913  0.023677
```

[410]:
```
batter_avg_stats.nsmallest(10, 'O - Ins')
```

[410]:
```
     batter  avg_wobacon  avg_intrinsic_no_sprint  avg_intrinsic_with_sprint  \
90   670623     0.296002                 0.348771                   0.344013
12   571448     0.313148                 0.338826                   0.328170
56   662139     0.304056                 0.329242                   0.343043
21   606115     0.297009                 0.317394                   0.305619
4    514888     0.350911                 0.369539                   0.371067
26   608070     0.344644                 0.361111                   0.370269
66   664034     0.342565                 0.357708                   0.342357
19   605137     0.348239                 0.361713                   0.351387
123  691026     0.326021                 0.339067                   0.352644
67   664761     0.366324                 0.377912                   0.373824

     sprint_speed last_name, first_name   O - Ins    O - Is
90           25.9        Paredes, Isaac -0.052768 -0.048011
12           25.3        Arenado, Nolan -0.025677 -0.015021
56           28.5       Varsho, Daulton -0.025187 -0.038988
21           25.6        Arcia, Orlando -0.020385  -0.00861
4            27.1          Altuve, Jose -0.018629 -0.020157
26           28.2         Ramírez, José -0.016467 -0.025625
66           25.1            France, Ty -0.015143  0.000208
19           25.4            Bell, Josh -0.013474 -0.003148
123          28.8           Winn, Masyn -0.013046 -0.026623
67           26.3            Bohm, Alec -0.011588   -0.0075
```

[412]:
```
batter_avg_stats.nsmallest(10, 'O - Is')
```

[412]:
```
     batter  avg_wobacon  avg_intrinsic_no_sprint  avg_intrinsic_with_sprint  \
90   670623     0.296002                 0.348771                   0.344013
56   662139     0.304056                 0.329242                   0.343043
123  691026     0.326021                 0.339067                   0.352644
26   608070     0.344644                 0.361111                   0.370269
127  696285     0.319209                 0.323381                   0.339911
4    514888     0.350911                 0.369539                   0.371067
8    543760      0.32846                 0.338720                   0.348582
25   607208     0.361938                 0.356920                   0.381630
12   571448     0.313148                 0.338826                   0.328170
81   668715     0.345176                 0.352073                   0.359591

     sprint_speed last_name, first_name   O - Ins    O - Is
90           25.9        Paredes, Isaac -0.052768 -0.048011
```

```
56            28.5       Varsho, Daulton -0.025187 -0.038988
123           28.8           Winn, Masyn -0.013046 -0.026623
26            28.2         Ramírez, José -0.016467 -0.025625
127           29.7          Young, Jacob -0.004173 -0.020702
4             27.1          Altuve, Jose -0.018629 -0.020157
8             28.5        Semien, Marcus -0.010259 -0.020122
25            29.6          Turner, Trea  0.005018 -0.019692
12            25.3        Arenado, Nolan -0.025677 -0.015021
81            28.2        Steer, Spencer -0.006897 -0.014415
```

## 0.5 Accuracy

[419]:
```
MAD_no_sprint = np.mean(np.abs(batter_avg_stats['avg_wobacon'] -
  batter_avg_stats['avg_intrinsic_no_sprint']))

MAD_with_sprint = np.mean(np.abs(batter_avg_stats['avg_wobacon'] -
  batter_avg_stats['avg_intrinsic_with_sprint']))

print(f"MAD (Intrinsic Value without sprint speed): {MAD_no_sprint:.4f}")
print(f"MAD (Intrinsic Value with sprint speed): {MAD_with_sprint:.4f}")
```

```
MAD (Intrinsic Value without sprint speed): 0.0132
MAD (Intrinsic Value with sprint speed): 0.0115
```