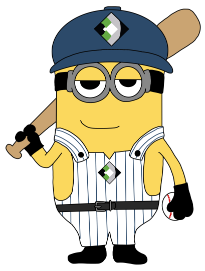
Pitcher Dynamics xPLained - Creating a Model that Determines When to Pull a Starting Pitcher

Atul Venkatesh, Levon Sarian, and Ishan Kinikar



Abstract

The decision to pull a starting pitcher is one of the most difficult in professional baseball. Managers often have to tread the fine line between not pulling a pitcher too early and not keeping them in too late. Often times, these choices are pre-determined based on a certain pitch limit, leading to ill-advised decisions that can be the difference between a win and a loss. In this paper, we use a combination of situational factors and machine learning to determine the proper time to pull a pitcher. Our ultimate product is a manager assistance tool that combines these factors with qualitative variables to provide immediate, in-game feedback on when to pull the pitcher.

1 Introduction

Let's travel back to October 27th, 2020. The Los Angeles Dodgers were up three games to two, looking to close out the Tampa Bay Rays and win their first World Series in over three decades. However, in Game Six, the Rays were beating the Dodgers 1-0 with Blake Snell throwing five scoreless innings, striking out nine, and giving up just one hit.

The sixth inning began with Snell forcing a pop-out followed by a single. This didn't seem too concerning. Snell's pitch count was only at 73, and his stuff seemed to be working.

However, the analytics told the Rays to pull the starting pitcher before the third time through the batting order. With lead-off hitter Mookie Betts up, Kevin Cash made his way to the mound.

The decision was met with immediate opposition. Snell was on fire. Why pull him out because of a seemingly random rule? These concerns were justified, as Snell's replacement, Nick Anderson, proceeded to give up a double to Betts and allow two runs. This was the ballgame, as the Dodgers ended up winning 3-1, and with that, the World Series [1].

There were certainly arguments for pulling Snell, but the *third time through the order* rule seemed too arbitrary. Staking a team's entire season on this one justification was certainly risky.

While situations like Snell's are not rare, it's more common for starters to overstay their welcome. Decisions to keep starters in late, such as with Matt Harvey in Game Five of the 2015 World Series [2][3] and Pedro Martinez in Game Seven of the 2003 ALCS [4] are prime examples of this.

Choosing the correct time to pull a starter can be the difference between a win and a loss. Currently, there is no situational metric to help managers/coaches make this loaded decision. Therefore, by taking in-game factors into account, we have created a model to tell us the appropriate time to pull a pitcher. Was the Blake Snell pulling justified? With our model, we have the answer.

2 Data

The data is provided by SportsMEDIA Technology (SMT) for the 2024 baseball analytics competition. The data includes both ball and player tracking, but we chose to focus on the pitch-by-pitch data. The data contains the following information:

Column name	**Description**
Game string	A unique game id
Home Team	A unique code for each home team. Also signifies the level of the farm system. Home1A is the first level, Home2A is the second level, Home3A is the third level, and Home4A is the fourth level.
Visiting team	A unique id for the visiting team that also identifies level in the farm system.
Day of game	Tells us the day number of the game.
Play number	Play number so far in the game.
At-bat number	At-bat number so far in the game
Inning number	Inning number in the game. Also tells us if it was the top or bottom of the inning.
Game state	Unique player ids of everyone on the field. This includes the pitcher, catcher, the seven other defensive players, the batter, and the base-runners, if any.

While this data was a solid foundation, we were missing a lot of necessary information, such as pitch count, outs, and score. We decided to create our own variables.

The first variable we created was pitch count. We operated on the assumption that each row represented a pitch and counted up with each subsequent row for a given pitcher. The other variables we constructed will be explained later.

We chose to focus on only starters for the home team since we wanted to track trends within a singular farm system. We decided to not separate the different levels in the farm system but rather included level as a factor in our model. We also assumed that the pitcher who pitched in the top of the first inning was the home team starter.

Next, we had to make the data usable. Some entries had clear problems, like displaying a "0th inning" or having no pitcher id. We decided to remove all games that contained these issues. Our dataset now contained 12899 pitches across 173 games.

3 Model preparation

Our model will consist of two smaller models: one to predict the number of pitches left in the game, and another to predict the number of pitches left in the inning. We needed to determine which variables should be included in our model.

3.1 Major factors

Listed below are the factors that were used to train the model. The independent variables are factors that could influence the dependent variables.

Factor	**Description**	**Scope**	**Source**
Pitch Count (Game)	**Independent variable**. Pitches thrown in the game by the starter. Variable derivation explained earlier.	Game	Self-produced
Pitch Count (Inning)	**Independent variable**. Pitches thrown in the inning. Calculated similarly to pitch count (Game).	Inning	Self-produced
Out counter	**Independent variable**. Since we were not given outs, we created our own metric to identify an out by looking at the change in baserunners between at-bats. The situations where we determined an out occurred are explained in detail in **Appendix A**.	Inning	Self-produced
Baserunner Situation	**Independent variable**. Who was on first, second, and third base at any given pitch.	Inning	SMT Data
Difficulty	**Independent variable**. Each situation is contextualized by outs and base-runners. We created *Difficulty*, which represents how difficult a given situation is for the pitcher. Based on the number of runs teams tend to score in that instance. 3.1.1	Inning	Self-produced + MLB Data [5]
Batting Order	**Independent variable**. Position in the batting order. We created a variable that counted up from one to nine for each unique batter. Once the variable hit nine, the counter reset to one.	Game	Self-produced
Farm System Level	**Independent variable**. We wanted the specific farm system to be an additional variable in our model. This is the ”Home Team” variable.	N/A	SMT Data
Pitches left in game	**Dependent variable**. How many pitches are left in the game. Pitch count (game) counted backwards	Game	SMT Data
Pitches left in inning	**Dependent variable**. How many pitches are left in the inning. Pitch count (inning) counted backwards	Game	SMT Data

3.1.1 *Difficulty* - Elaborated

There are 27 possible game states: nine unique combinations of baserunners for each of the three out states. We expect a certain number of runs to be scored in each game state. Since we weren't given runs scored, we had to get creative. We took data from the 2022 MLB season [5], giving us the average runs scored in each unique situation. We scaled these values to create a difficulty metric. We combined the baserunner situation from the SMT data with our *out counter* to match each situation with the scaled ”situation score”. Under our rating paradigm, bases loaded with no outs was the most difficult scenario with a *difficulty* of 100 while two outs with no base-runners was the easiest scenario with a *difficulty* of **4.01**.

4 Creating the model

4.1 Expected Pitches left in the Game (xPLG)

To figure out the expected pitches a starter has left in the game, we fitted our variables into an Extreme Gradient Boosting (XGBoost) model [6]. XGBoost is a machine learning method where weaker predictive models are iterated over each other to create one strong model that minimizes the overall residual error [7][8].

We weighed pitch observation as independent of each other rather than grouping by games, giving us 12899 entries rather than 173. If we grouped by games, the lower entries increase the probability of important values being left out. **Figure 1** displays the most influential attributes. Pitch count, unsurprisingly enough, had a feature importance of around 0.8, indicating that it contributes 80 percent of the total importance in the model.

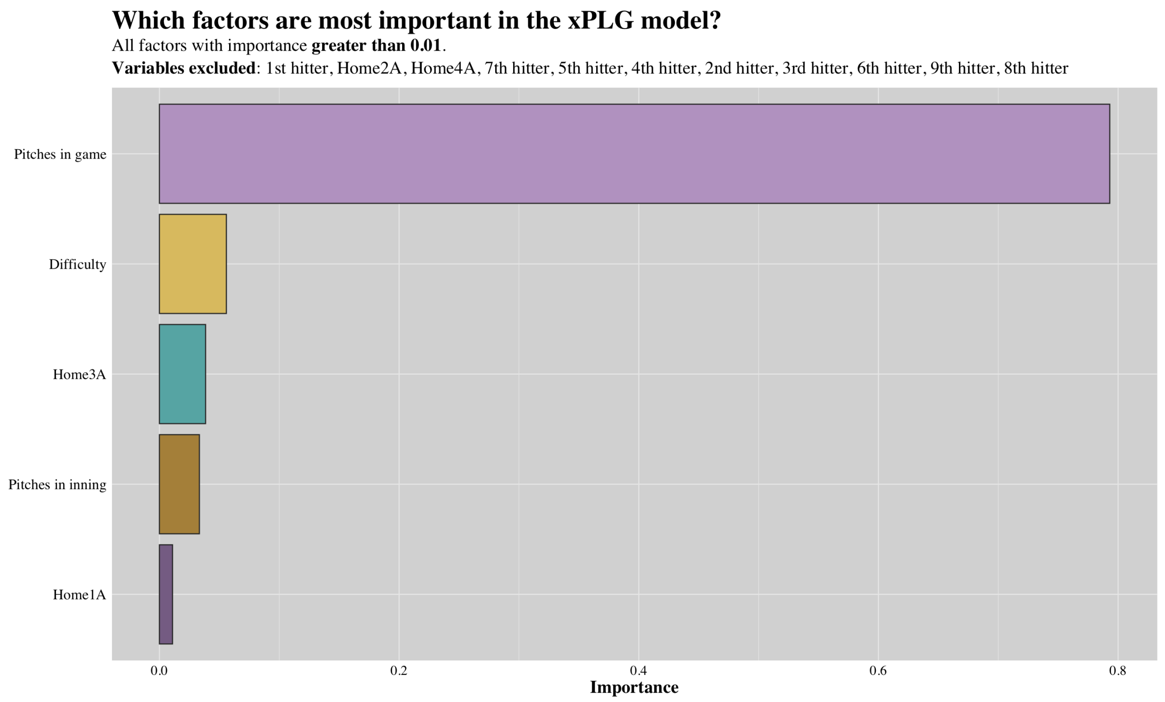


Figure 1: Most influential attributes for the xPLG model

Figure 2 displays how well the modern fared against reality. To our pleasant surprise, predictions in 2201 of the 12899 entries were within two pitches of the actual number of pitches thrown.

Due to outliers, our predictions had a root mean-squared error of 16.5. This is not a cause for concern since a deviation from what actually happened does not necessarily mean the model is incorrect.

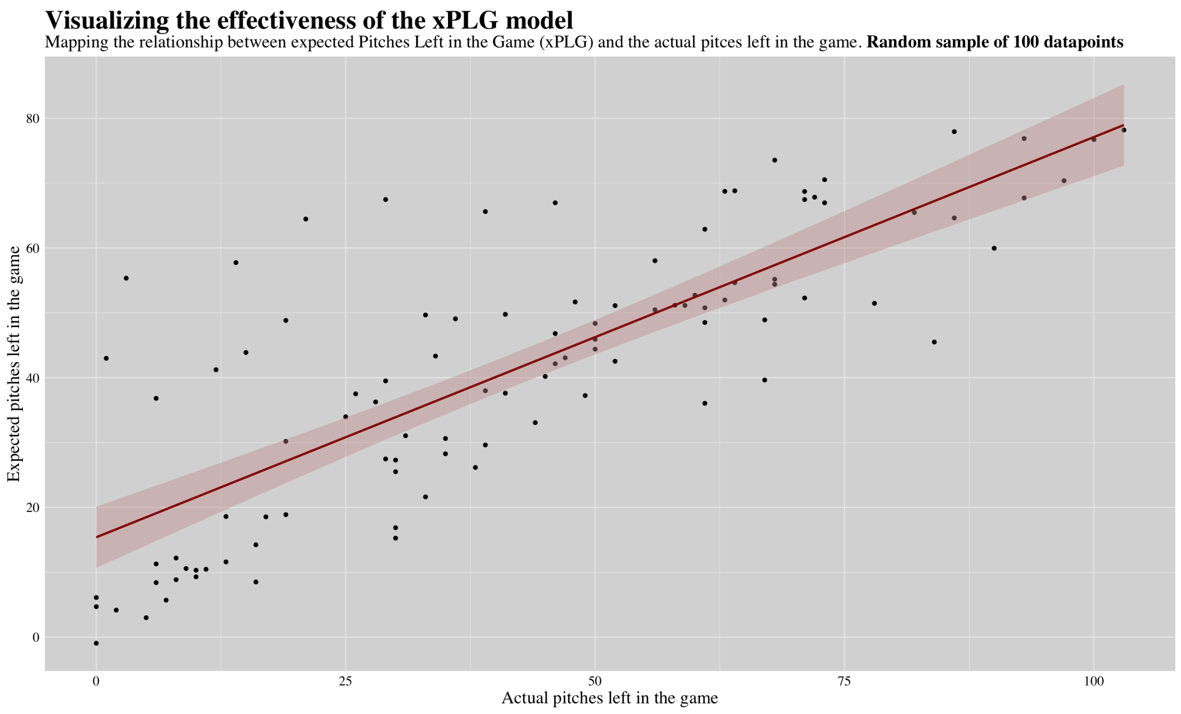


Figure 2: Plotting the accuracy of the xPLG model

4.2 Expected Pitches left in the Inning (xPLI)

Next is the xPLI model - the expected pitches left in the inning. We factored the same variables as before. **Figure 3** displays the most important factors for the xPLI model. Interestingly enough, *difficulty* was the most influential factor.

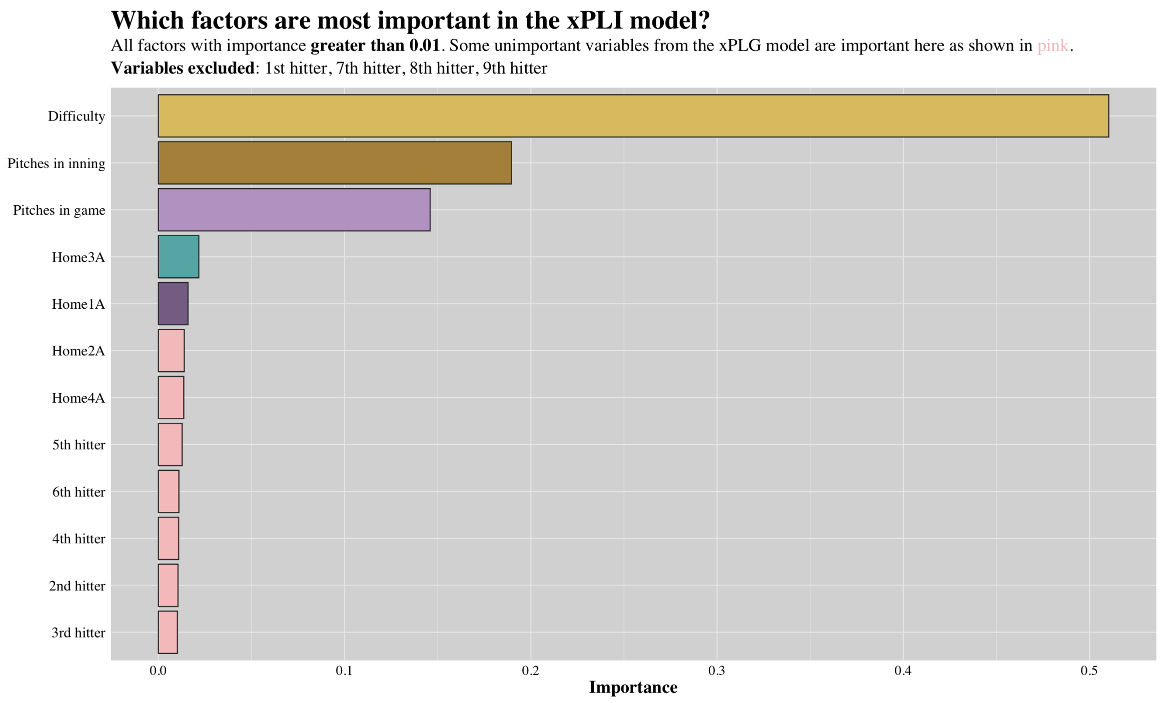


Figure 3: Most influential attributes for the xPLI model

For this model, 6784 of the 12899 observations were predicted within two pitches of the actual pitches thrown in the inning. **Figure 4** visualizes this relationship, providing a clear correlation. The xPLI model had a significantly lower root mean-squared error at 4.7. This makes sense, since pitchers throw more pitches in a game than an inning.

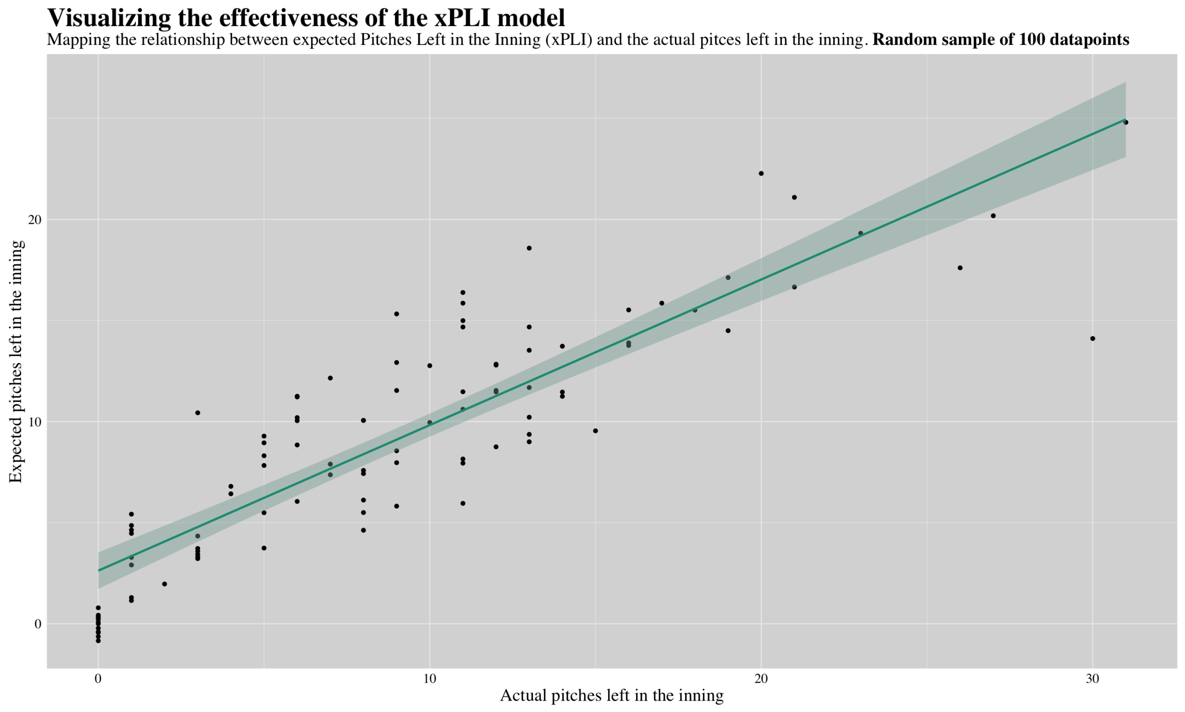


Figure 4: Plotting the accuracy of the xPLI model

We decided that a pitcher should be pulled **when the xPLI was less than the xPLG**. In other words, when the model predicted the pitcher would not make it through the rest of the inning. And with that, our final model was created.

4.3 Bringing it all together

We split the pulling criteria into three categories:

Pulling type	**Description**
Overworked:	The coaching staff decides to leave the pitcher in too long, pulling him **after** we recommended them to.
Underworked:	The coaching staff decides pull the pitcher too early, choosing to pull the pitcher **before** we recommended them to.
Appropriately used:	The coaching staff chose to pull the pitcher at the **same time** as our recommendation.

How should we evaluate the model? Given limitations with our data, we decided to use *pitch count*. In most cases, throwing more pitches means a pitcher is less successful. As a result, at the time of our recommended pulling, if the pitcher's *xPLG* was less than the *actual pitches left in the inning*, we counted this as a success.

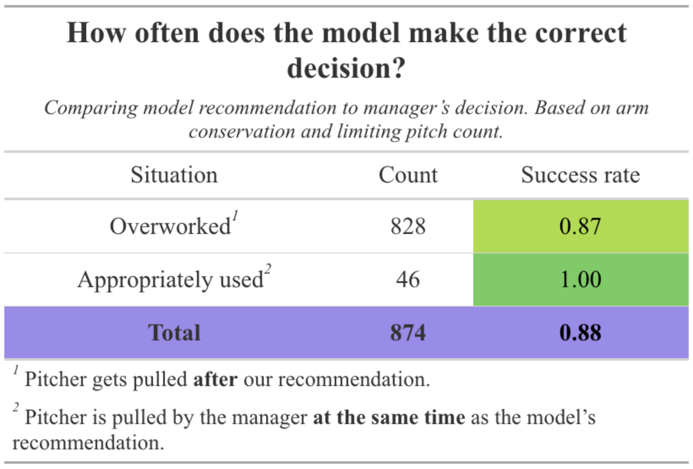
Let's say the model recommended a pitcher to be pulled, stating they only had 10 pitches left in the game. If the pitcher took 25 pitches to get out of the inning, chances are he was unsuccessful, and our model made the right decision. The opposite would be true if the pitcher performs better than expected. We automatically counted the *appropriately used* instances as a success.

This solution does not count for when the pitcher was *underworked*. To truly evaluate this aspect, we would need to see how the relief pitcher fared compared to our expected metric. That requires taking the reliever's relative skill level into account which is beyond the scope of our data.

5 Statistical Applications

5.1 Model success rate

Figure 5 displays the model's success rate. Our model had a success rate of 0.88, meaning it made the correct prediction in 88 percent of the 874 highlighted instances.



How often does the model make the correct decision?

Comparing model recommendation to manager's decision. Based on arm conservation and limiting pitch count.

Situation	Count	Success rate
Overworked[1]	828	0.87
Appropriately used[2]	46	1.00
Total	**874**	**0.88**

[1] Pitcher gets pulled **after** our recommendation.

[2] Pitcher is pulled by the manager **at the same time** as the model's recommendation.

Figure 5: Visualizing the accuracy of both expected metrics

This solution is not perfect. For example, there could be instances where the pitcher was more successful while throwing more pitches. Despite these limitations, we have a good idea for how the model turned out. We further emphasized the model's success by providing situational context for several flagged games, provided in **Appendix C**.

5.2 Expectation metrics

Did a pitcher throw more or less pitches than the model expected? We can now answer this quesiton. For "overworked" instances, we took the number of pitches thrown after our recommendation. For *underworked* instances, we took the negative of the player's xPLG at the time they got pulled. We chose to look at one instance in each game: either the first time the model recommended the pitcher to get pulled or when they actually got pulled. The result is a metric called **POE** (Pitches Over Expected).

Figure 6 shows the performance of pitchers, looking at the average POE across all of their starts. As we can see, there are a larger number of pitchers that threw **more pitches than our model expected**.

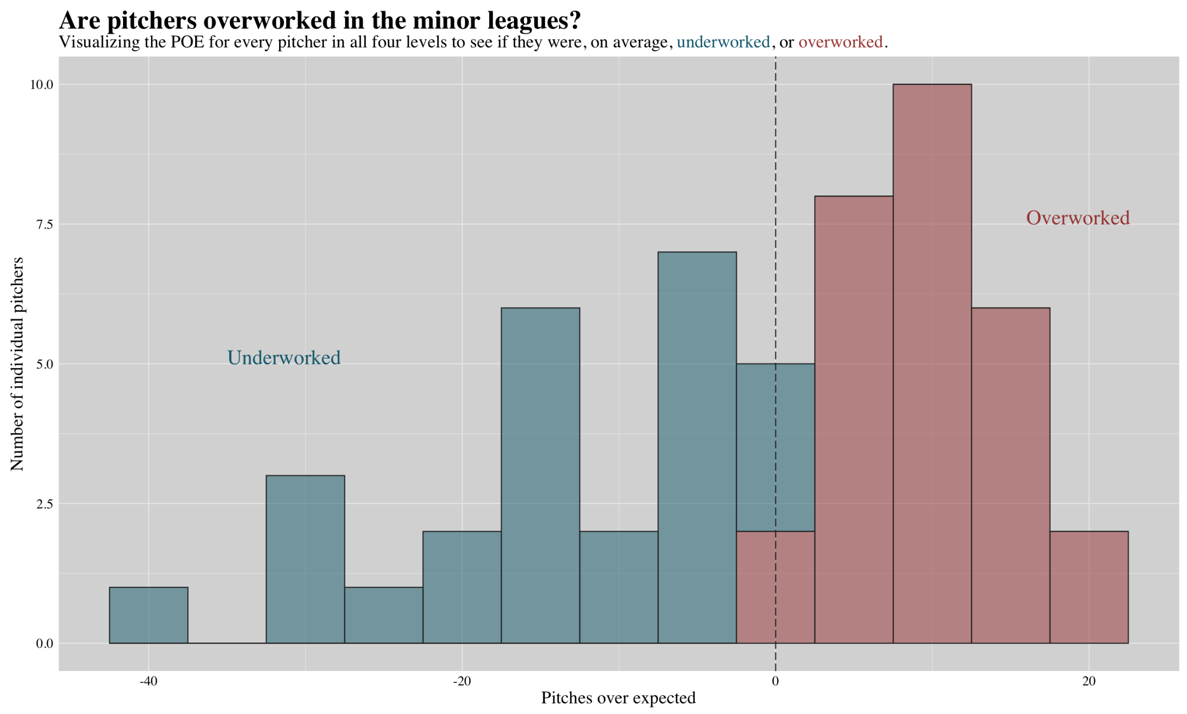


Figure 6: Visualizing the POE of pitchers

Figure 7 confirms this trend. Pitchers throw for more pitches than the model predicted, indicating there pitchers may be overworked in the minor leagues.

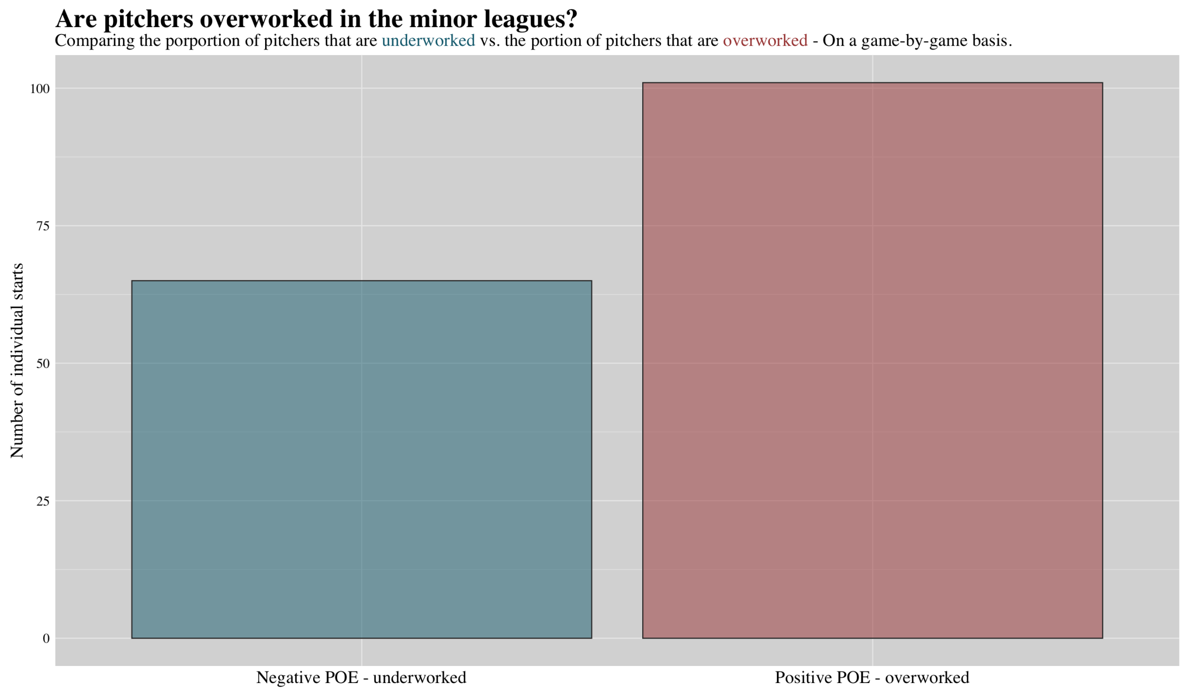
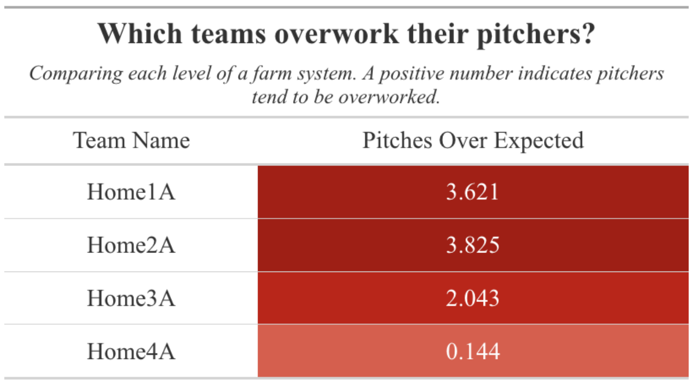


Figure 7: Visualizing the distribution of starts in terms of POE

Figure 8 looks at the POE distribution by team. In this specific farm system, all teams tend to overwork their pitchers.



Which teams overwork their pitchers?

Comparing each level of a farm system. A positive number indicates pitchers tend to be overworked.

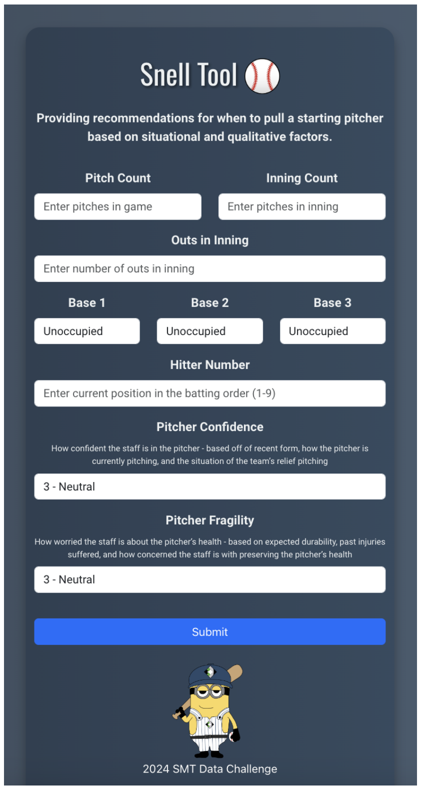
Team Name	Pitches Over Expected
Home1A	3.621
Home2A	3.825
Home3A	2.043
Home4A	0.144

Figure 8: POE distribution by team

6 The ***Snell Tool*** (UI)

The main limitation of our model is that it does not account for situational factors such as confidence in the pitcher and pitcher injury history. These qualitative variables would be imperative in determining when to pull a pitcher. To account for this limitation, we have created an app called the ***Snell Tool***.

Built using React.js, it provides a comprehensive and intuitive user interface (UI) that helps the coaching staff decide when to pull a starting pitcher. **Figure 9** illustrates the ***Snell Tool***'s design. Users input game-state information and two qualitative variables: *pitcher confidence* and *pitcher fragility* (both on a scale of 1-5). These variables are based on the user's confidence in the pitcher and how much they want to push the pitcher physically and mentally. The ***Snell Tool*** is open to the public, residing at snelltool.com. Note that the UI is mobile-friendly, so any device will display it properly.



Snell Tool

Providing recommendations for when to pull a starting pitcher based on situational and qualitative factors.

Pitch Count: Enter pitches in game

Inning Count: Enter pitches in inning

Outs in Inning: Enter number of outs in inning

Base 1: Unoccupied

Base 2: Unoccupied

Base 3: Unoccupied

Hitter Number: Enter current position in the batting order (1-9)

Pitcher Confidence

How confident the staff is in the pitcher - based off of recent form, how the pitcher is currently pitching, and the situation of the team's relief pitching

3 - Neutral

Pitcher Fragility

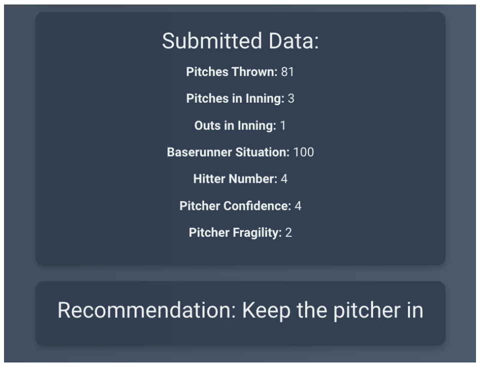
How worried the staff is about the pitcher's health - based on expected durability, past injuries suffered, and how concerned the staff is with preserving the pitcher's health

3 - Neutral

2024 SMT Data Challenge

Figure 9: ***Snell Tool*** user interface. *Click on image to be redirected to our site.*

By fitting a linear regression model on top of our final data, the tool calculates an xPLG and xPLI based on the user's input before spitting out a recommendation for the manager: keep the pitcher in or pull them out. A sample output is shown in **Figure 10**.


Submitted Data:

Pitches Thrown: 81

Pitches in Inning: 3

Outs in Inning: 1

Baserunner Situation: 100

Hitter Number: 4

Pitcher Confidence: 4

Pitcher Fragility: 2

Recommendation: Keep the pitcher in

Figure 10: ***Snell Tool*** example output

The technical mechanisms behind the UI, including state management and real-time rendering, play a pivotal role in maintaining the app's user experience. Moreover, the qualitative variables have allowed the pulling decision to not be based solely on statistics, but rather taking the pitching coach's judgement into account. For a more detailed explanation, please refer to **Appendix D**. An anonymized GitHub repository with our code can be found here.

7 Discussion

7.1 Pitcher injury prevention

According to our model's analysis, teams tend to overuse pitchers. Over the course of a season, these excess pitches add to workload stress. This is especially true for older pitchers [9]. The consequences of overworked pitchers hurt both teams and players. At the MLB level, fatigued pitchers are more likely to sustain arm injuries. For younger pitchers, excessive workload and lack of regulation are the major factors behind rising UCL injury rates [10]. This emphasizes the urgency for the implementation of the ***Snell Tool*** at all levels.

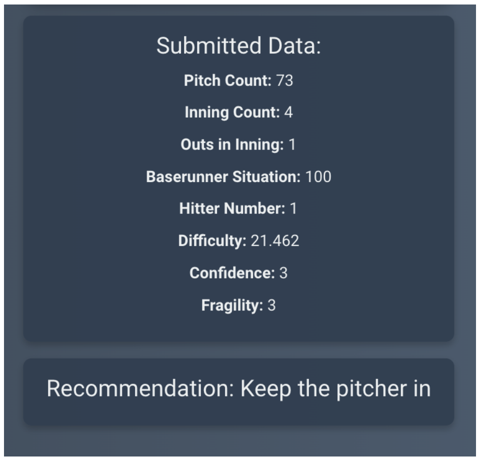
Accounting for training pitches, injury history, and rest days creates a complicated equation that can be difficult to streamline. In future, the ***Snell Tool*** could use data collection from training pitches and bio-mechanics to help produce workload management strategies that prolong the careers of many pitchers.

7.2 Decision-making evaluation tool

As mentioned previously, the ***Snell tool*** requires users to import qualitative information. As such, it is possible to use this success of these inputs to evaluate coaching decisions. Let's say the pitching coach uses our model to decide on keeping the starter in relatively late. If this starter loses steam and lets up several runs, the coaching staff could be blamed for this occurrence. After all, if the coaching staff had inputted different entries for the qualitative aspects of the model, the outcome might be different.

8 Conclusion

So did the Rays make the right decision? We inputted the game-state information at the moment Blake Snell was pulled. Setting both qualitative variables at *neutral*, our model strongly recommended to **keep Snell in**, as **Figure 11** emphasizes.



Submitted Data:

Pitch Count: 73

Inning Count: 4

Outs in Inning: 1

Baserunner Situation: 100

Hitter Number: 1

Difficulty: 21.462

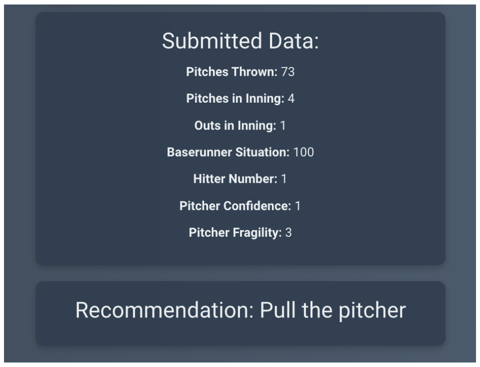
Confidence: 3

Fragility: 3

Recommendation: Keep the pitcher in

Figure 11: Model recommendation - Neutral *confidence* and *fragility*

However, if we assumed the Rays were extremely unconfident in their starting pitcher, then the model recommended to **pull Snell**, shown in **Figure 12**.



Submitted Data:

Pitches Thrown: 73

Pitches in Inning: 4

Outs in Inning: 1

Baserunner Situation: 100

Hitter Number: 1

Pitcher Confidence: 1

Pitcher Fragility: 3

Recommendation: Pull the pitcher

Figure 12: Model recommendation - low *confidence* and holding *fragility* constant.

This example illustrates the complexity of the issue we've chosen to address. We hope that our model can be a first-of-its-kind decision-making tool, providing a gut-check for managers and pitching coaches at all levels of professional baseball. Maybe next time your favorite pitcher gets pulled after seven no-hit innings, our model can tell you if that decision made sense.

9 References

1. https://www.youtube.com/watch?v=DLnJ3ih5xQ0
   Jomboy Media. (2021). *Kevin Cash taking Blake Snell out of the game was one of the worst moves in postseason history*. YouTube. https://www.youtube.com/watch?v=DLnJ3ih5xQ0

2. https://nypost.com/2020/06/17/matt-harvey-was-one-inning-from-becoming-a-mets-world-series-legend/
   Braziller, Z. (2020, June 18). *Matt Harvey was one inning from becoming a mets world series legend*. New York Post. https://nypost.com/2020/06/17/matt-harvey-was-one-inning-from-becoming-a-mets-world-series-legend/

3. https://www.youtube.com/watch?v=of3CSLtR88A
   And That's Baseball. (2023). *The Heartbreaking Tragedy of Matt Harvey*. YouTube.
   https://www.youtube.com/watch?v=of3CSLtR88A

4. https://www.nbcsportsboston.com/mlb/boston-red-sox/grady-little-pulling-pedro-martinez-in-game-7-couldve-changed-history/181268/

Leger, J. (2021, June 4). *Grady Little pulling Pedro Martinez in Game 7 could've changed history*. NBC Sports Boston. https://www.nbcsportsboston.com/mlb/boston-red-sox/grady-little-pulling-pedro-martinez-in-game-7-couldve-changed-history/181268/

5. https://pitcherlist.com/adjusting-for-the-current-run-expectancy-matrix/
Drummey, D. (2022, November 17). *Adjusting for the current run expectancy matrix*. Pitcher List. https://pitcherlist.com/adjusting-for-the-current-run-expectancy-matrix/

6. https://www.youtube.com/watch?v=p5LxuIjh6Z8
MFANS. (2022, November 21). *M-FANS: Making a Random Forest and XGBoost Model with NFL Data*. YouTube. https://www.youtube.com/watch?v=p5LxuIjh6Z8

7. https://www.nvidia.com/en-us/glossary/xgboost/
NVIDIA. (2024). *XGBOOST*. NVIDIA. https://www.nvidia.com/en-us/glossary/xgboost/

8. https://dl.acm.org/doi/10.1145/2939672.2939785
Chen, T., & Guestrin, C. (2016, August 13). *XGBoost: A Scalable Tree Boosting System*. ACM Conferences. https://dl.acm.org/doi/10.1145/2939672.2939785

9. https://pubmed.ncbi.nlm.nih.gov/27132773/
Riff, A., Chalmers, P., Sgroi, T., Lesniak, M., Sayegh, E., Verma, N., Cole, B., & Romeo, A. (2016, August). *Epidemiologic comparison of pitching mechanics, pitch type, and pitch counts among healthy pitchers at various levels of youth competition*. PubMed.
https://pubmed.ncbi.nlm.nih.gov/27132773/

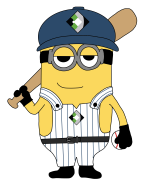
10. https://www.utsouthwestern.edu/newsroom/articles/year-2023/april-elbow-injuries-among-young-baseball-players.html
The University of Texas Southwestern Medical Center. (2023, April 10). *UT southwestern surgeon offers insight on elbow injuries among young baseball players*. UT Southwestern Medical Center. https://www.utsouthwestern.edu/newsroom/articles/year-2023/april-elbow-injuries-among-young-baseball-players.html

11. https://www.snelltool.com/
The Snell Tool. User Interface for this paper. SMT Data Challenge 2024. Github repo found here.

10 Acknowledgements

We would like to give a huge thank you to Dr. Meredith Wills, Andrew Steenkamer, Kenny Bores, Cameron Adams, and the rest of the SMT staff for their feedback and support throughout this entire project.

A special thanks to my little brother for designing our team's logo :)



Appendix A - Outs model explained

Note that of the 4086 innings that we examined, 3889 of them displayed three outs in the inning, which we assumed meant that the outs model worked accurately. We decided to remove the games that contained an incorrect number of outs.

Here are the specific situations where we concluded that an out occurred:

- If the game state **remains unchanged** from one at-bat to another, i.e. the next baserunners are the same as the current baserunners, then we are confident an out occured. The only exception is a solo home run, but we will correct for that later.

- If the batter does not show up on base in the next at-bat, unless he hit a home run, then we know he got out.
- **Coding for simple double play:** If there is a man on first and no one on second and third and there are no baserunners in the next at-bat, then we know a double play occurred.
- **Coding for simple fielder's choice:** If there is only a man on first and only a man on first in the next at-bat, but the ID of the next first baserunner **matches the ID of the hitter**, then an out occured.
- **Coding for complex double plays**:
  - Runner on first and second, the second baserunner advances to third base, but first and second base are empty in the next at-bat. We know a double play occurred here.
  - Runner on first and third and the next at-bat, the bases are empty. Here we would assume that the runner on third scored, and the first baserunner and hitter were thrown out. We can be reasonably confident that this is the case.
  - Bases are loaded and the next at-bat, the runner on second is now at third, and the bases are empty. Here we can also assume that the runner on third scored, and the first baserunner and hitter were thrown out.
- **When the home team pitches, the last play of the inning is always an out.** If there is one out before the last play of the inning, we know an inning-ending double play occurred. If there are no outs before the last play of the inning, we know a triple play occurred. This provides some more accuracy to the model.
- **Adding in home runs:** SMT provides data that tells which at-bats were home runs. We merged this information with our existing data. In situations where a home run occurred, we made sure our model could recognize that there was **no out** on the play.

Appendix B - xPLG and xPLI explained

XGBoost algorithms serve as a method to understand the relationships within data and use those relationships to inform future predictions. This is commonly used to creat expected metrics across all sports. With this in mind, we created the xPLI and xPLG metrics. While these two metrics are separate, we created them in a similar fashion.

The first step is wrangling the data. We isolate all of the variables that we think are important. In this scenario, our variables were the level in the farm system, the player's current pitch count, the number of pitches the player has thrown in an inning, the situation difficulty, and the position in the batting order. For the xPLG metric, our outcome variable was the actual number of pitches left in the game, so we had to calculate that as our "label variable". Similarly, for the xPLI metric, the actual pitches left in the inning was our label variable.

Next, we made sure all non-numeric variables would be accepted by the algorithm. This means converting our character variables like level of farm system and hitter number to "factor" variables. This ensures that values are split into categories rather than numbers. Take batting order for example. While this is technically a number, we do not want to categorize it as such. No one can bat 3.5th in the order and we want each spot in the batting order to be treated as an **independent value**.

Next, we split our data into training and testing. We decided to do an 80/20 split respectively. We would include the respective label variable in the training datasets and removed the variable for the test datasets to see how our model performed.

Conducting the actual XGBoost algorithm was fairly straightforward. We used the *xgboost* package in R and specified the dimensions. With a combination of five-fold cross validation and trial and error, we determined the best possible dimensions to reduce root mean-squared error. For the xPLG model, the number of rounds was 134, the number of early stopping rounds was three, the max depth was six, and the eta was 0.18. For the xPLI model, the number of rounds was 230, and the number of stopping rounds, max depth, and eta were all the same as above.

With that, the two XGBoost models were created. We were able to evaluate the success through statistical measures like root mean-squared error and R-squared.

Appendix C - A micro-level analysis of our model

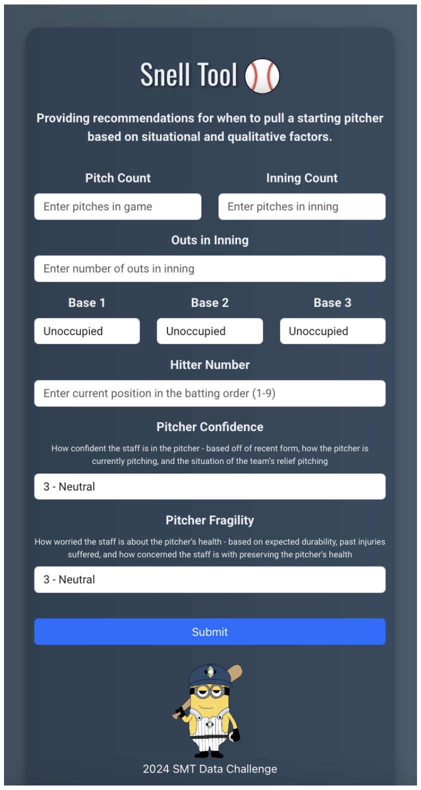
If managers chose to keep their pitchers in after our recommendation, how did the pitchers fare? According to the model, we expect them to struggle. To test the hypothesis, we looked at all instances of a ”overworked” and highlighted what happened in some of the notable games. We included the home team (level of farm system), away team, and day of the game.

- **Home 1A vs. Vis1AL, Day 18** Pitcher pitched three full innings. We recommended to pull the pitcher out before the beginning of the 4th inning. What happened? The pitcher got one out and then proceeded to load the bases. The manager eventually pulled the pitcher at 1 out with the bases loaded.
- **Home 1A vs. Vis1AU, Day 77** The pitcher is near the beginning of the fourth inning. Man on first, no outs. Already gave up one run in the innings. We say pull the pitcher. What happens? Next batter goes on base. Allows another base hit, but the runner on first gets out stretching to third. Ends up getting out of the inning. But some damage has been done. Interestingly enough, two pitches after our recommendation, **we no longer recommended the pitcher would be pulled**. However, five or so pitches after our first recommendation, we once again recommended the pitcher to be pulled.
- **Home1A vs. Vis1AY, Day 109 -** The pitcher has pitched four full innings. We recommend the pitcher to be pulled now. The pitcher proceeds to two base runners on base then gives up a double, allowing for a run to score. While the pitcher eventually gets two outs, he also allows another base hit and then another run. Eventually gets pulled out with two outs and men on first and third.
- **Home 2A vs. Vis2AF, Day 11 -** Pitcher goes 6 full innings. We recommend to pull him before the first pitch in the 7th inning. The pitcher is facing the cleanup hitter. The pitcher gets an out before letting the next two hitters get on base. Now the manager decides to pull him out.
- **Home2A vs. Vis2AG, Day 15 -** Pitcher pitched five full innings. We recommended the pitcher be pulled before the beginning of the 6th inning. The pitcher proceeded to record one out, give up a double, and get pulled mid-inning.
- **Home2A vs. Vis2AI, Day 28 -** Pitcher goes 5 full innings. We recommend pulling him before the 6th inning. Proceeds to allow the first three batters to reach base without getting an out. **Gets pulled with the bases loaded and zero outs.**
- **Home2A vs. Vis2AT, Day 118 -** The pitcher is midway through the fourth inning, man on first and second with no outs. We recommend the pitcher be pulled. The pitcher proceeds to give up a double and a single, letting up two runs before finally getting 1 out and getting pulled with men on second and third. After our recommendation, the pitcher let in multiple runs, left with baserunners on, and only got one out.
- **Home 3A vs. Vis3AK, Day 38 -** The pitcher pitched six full innings. We said to pull the pitcher 8 pitches into the 7th inning. The pitcher proceeds to give up two doubles and a single, letting up one run and only getting one out before he is pulled out of the game.
- **Home 3A vs. Vis3AO, Day 78 -** The pitcher is seven pitches into the fifth inning with a man on first and one out. We recommend pulling the pitcher. The pitcher proceeds to get one more out before allowing the next two hitters to reach base. With the bases loaded, the pitcher allows a bases clearing triple before getting pulled out.
- **Home3A vs. Vis3AV, Day 140 -** The pitcher is midway through the second inning and in a bit of a jam. Man on first and second with one out in the inning. We say pull the pitcher. What happens next? The next baserunner gets on base so the bases become loaded. The last batter the pitcher faces also gets on base, driving in a run. After our recommendation, the pitcher let two batters on base, giving up a run, and not getting an out.
- **Home4A vs. Vis4AO), Day 27 -** The pitcher pitched five full innings before giving up a leadoff single. We then recommended the picher be pulled. The pitcher proceeds to **let up a two run home run**.

- **Home 4A vs. Vis4AV, Day 78** The pitcher is 11 pitches into the inning. No outs, man on second and third. We say pull the pitcher, but they wait one at bat. The batter ends up getting on base. With the bases loaded and nobody out, the starter gets pulled.
- **Home 4A vs. Vis4AY, Day 104 -** The pitcher has pitched six full innings. we recommend pulling the pitcher out before the 7th inning. What happened? Pitcher got two outs, let the third hitter get on base, and then gave up a hit to the fourth hitter. Gets pulled out, crossing 100 pitches. Leaves a man on first and third for the reliever.
- **Home 4A vs. Vis4BD, Day 136** Pitcher pitched five full innings. We recommended puling the pitcher before the start of the 6th inning. The pitcher had a rough 6th inning, facing 5 batters. While the pitcher did not give up any runs, the inning took 30 pitches, putting his final pitch count over 100

Appendix D - The *Snell Tool*: explained

The app UI consists of several key components, each serving a specific function to collect data, display results, and guide the user through the decision-making process.



Snell Tool

Providing recommendations for when to pull a starting pitcher based on situational and qualitative factors.

Pitch Count

Enter pitches in game

Inning Count

Enter pitches in inning

Outs in Inning

Enter number of outs in inning

Base 1

Unoccupied

Base 2

Unoccupied

Base 3

Unoccupied

Hitter Number

Enter current position in the batting order (1-9)

Pitcher Confidence

How confident the staff is in the pitcher - based off of recent form, how the pitcher is currently pitching, and the situation of the team's relief pitching

3 - Neutral

Pitcher Fragility

How worried the staff is about the pitcher's health - based on expected durability, past injuries suffered, and how concerned the staff is with preserving the pitcher's health

3 - Neutral

Submit

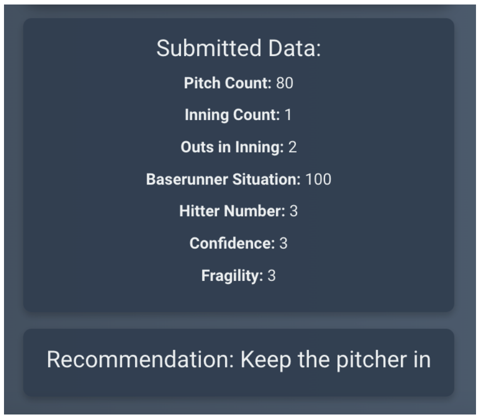
2024 SMT Data Challenge

The primary components include:

1. **Data Input Form**:
    - **Pitch Count, Inning Count, and Outs in Inning**: These input fields allow users to enter numerical data regarding the pitcher's current game statistics. The UI utilizes standard HTML ¡input¿ elements with appropriate validation to ensure accurate data entry.
    - **Base-runner Situation**: A set of dropdown menus representing the bases (first, second, and third) enables users to specify the presence of runners. This feature dynamically updates the game state and contributes to the difficulty calculation.
    - **Hitter Number**: An input field allows the user to enter the position of the current batter in the lineup, which influences the expected performance metrics.

- **Qualitative Variables (Confidence and Fragility)**: Two dropdown menus provide a range of options from 1 to 5, representing the pitcher's subjective attributes. The confidence metric is based off recent form, how the pitcher is currently pitching, and the situation of the team's relief pitching. The fragility metric is based on expected durability, past injuries suffered, and how concerned the staff is with preserving the pitcher's health. The UI includes descriptive text under each dropdown, helping users understand the impact of these selections on the decision-making process.

2. **Real-time feedback and recommendations**:



Submitted Data:

Pitch Count: 80

Inning Count: 1

Outs in Inning: 2

Baserunner Situation: 100

Hitter Number: 3

Confidence: 3

Fragility: 3

Recommendation: Keep the pitcher in

- **Dynamic Display of Difficulty**: Once the user inputs the relevant data, the app calculates the difficulty of the current game situation based on the outs and base-runner configuration. This value is immediately displayed on the UI, providing real-time feedback to the user.
- **Expected Metrics and Decision Output**: The app calculates the values for expected pitches left in the game (xPLG) and expected pitches left in the inning (xPLI). Depending on these metrics, a recommendation is dynamically generated and shown to the user. This recommendation—either to pull the pitcher or keep them in—appears in a dedicated results section, highlighted to draw attention.

3. **Interactive design elements**:

- **Responsive Layout**: The UI is designed to be responsive, adapting to various screen sizes and devices. This ensures that users can access the app on desktops, tablets, and smartphones without compromising functionality.
- **User Guidance and Error Handling**: The UI incorporates tooltips, placeholder text, and validation messages to guide users through the data entry process. For example, if a user inputs an invalid value or leaves a required field empty, the app provides instant feedback to correct the error.

Technical Mechanisms:

1. **State Management with React Hooks**:

- The app leverages React's useState and useEffect hooks to manage and update the state of the application. Each input field updates the corresponding state variable, which in turn triggers recalculations and UI updates. This mechanism ensures that all changes are reflected in real-time, providing an interactive experience.

2. **Form Handling and Data Submission**:

- Upon submission, the form data is collected and passed to the parent component (App.js). This data includes both the numerical inputs and qualitative assessments, which are then used to compute the xPLG and xPLI metrics. The app uses JavaScript's onSubmit event handler to prevent the default form submission behavior and instead process the data asynchronously.

3. **Real-time Calculation and Rendering**:

    - The calculation of xPLG and xPLI is handled within the App.js component. As the user interacts with the form, the app recalculates these metrics using the most up-to-date data. React's virtual DOM efficiently updates only the necessary parts of the UI, ensuring smooth performance even with frequent updates.

4. **Visualization and Feedback Loop**:

    - The UI provides a visual representation of the decision-making process. The calculated difficulty, expected metrics, and final recommendation are presented in a clear and concise manner. This feedback loop not only informs the user of the current situation but also educates them on the factors influencing the recommendation.