## Introduction

My co-manager and I have been running a fantasy baseball team for the last four seasons in a competitive ‘head-to-head category’ keeper league. The catch is that neither of us has watched a full baseball game, so we rely heavily on our decision-making intuition, probabilistic thinking, and collaboration. This has helped us make the playoffs three of the last four seasons (only 4/12 teams make it), including an appearance in the finals last year.

Our ultimate downfall each year though, is our consistent inability to pitch at a league average level, likely due to our lack of baseball knowledge. As we face a new team each week, we usually need to stream pitchers from the waiver wire to win categories such as strikeouts and quality starts. This is a challenging task because a high volume of difficult decisions must be made in real time with a large set of players we have mostly never heard of.

The idea for this project stems from years of painfully losing at the very end of each season, knowing exactly why we are losing, but never having a concrete solution for it. However, being able to stream pitchers with even a slightly better output each week, especially in the playoffs, would be a major competitive advantage over our opponents. This leads us to the question we will be trying to answer using data science and machine learning.

## Can we predict the likelihood an MLB pitcher will earn a ‘Quality Start’ (QS) next game?

In Major League Baseball (MLB), each team has a pitching rotation they cycle through each week. Pitchers cannot play daily because doing so, adds wear and tear to their throwing arm, so these starts are typically once a week. Of the statistics that help determine a pitcher’s success in a particular start, the QS may be the most important in fantasy. This is because a pitcher has two essential jobs: to prevent runs and force enough outs to get his teams’ batters back in a position to score. **To get a QS, a pitcher must allow three or less earned runs and play at least six innings.** The QS is a particularly important metric in fantasy baseball because of its correlation to other pitching categories we are trying to win: ERA, WHIP, and strikeouts.

## Structuring the Model

Predicting whether a pitcher records a quality start is a classification problem, but we specifically want to predict probability. This will provide us more flexibility with our decision making over the course of a season, depending on the situation.

**For our training set, I used data starting from the beginning of the 2016 regular season (April) all the way until the end of June 2019.** Months July to September 2019 were used for the validation set, while the shortened 2020 season (July to October) was used as the test set.

**Classifier**

**Did the pitcher record a QS? (Y/N)**

**All data collected prior to a pitcher’s start**

## Data Collection

Before modeling, I collected all game log data going back to 2016 for pitchers that played a game in 2020. Here is a brief list of the feature information used to predict the target:

* Raw game log statistics (last game, average of last 12 games, average to date)
* Feature engineered efficiency metrics from game log data
* Opponent team batting stats
* Pitcher’s team fielding stats

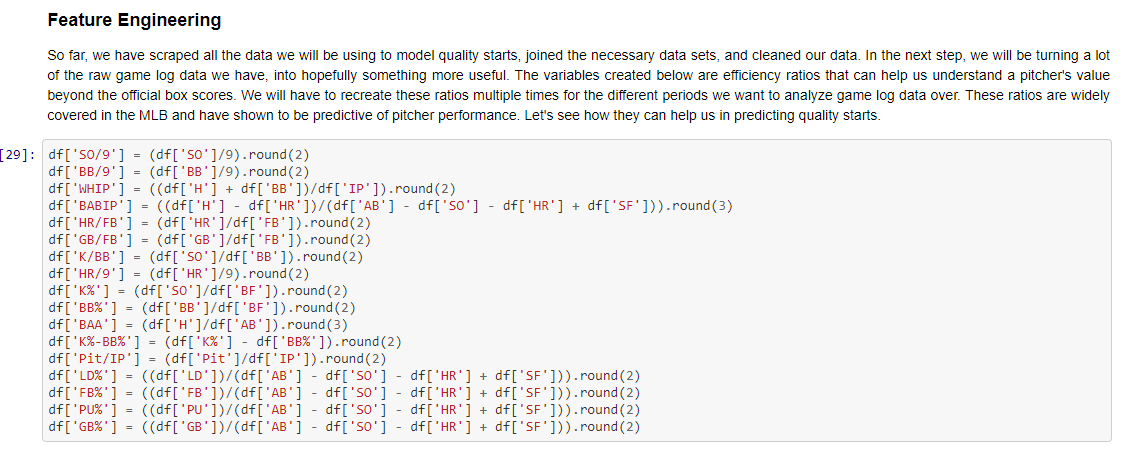
Data Sources:

* Baseball Reference – All game log stats
* ESPN – Team batting, team fielding stats, 2020 pitcher list

To pull game log information for every pitcher that started in 2020, data had to be scraped from baseball reference using the **BeautifulSoup** and **requests** packages in Python. I wrote a simple for loop to do this for every pitcher in our list using a base URL and iterating through each season. This resulted in over 32,000 rows of game log data. I also had to pull team fielding and batting data from ESPN using a similar method. This was much easier since I was only using yearly information. Of course, bringing all this data together in a clean format was not easy. The fielding/batting data needed to be pulled apart and then joined again to get the team names on the same rows as their respective rows. And the game log data had features we did not need and many blank rows that we had to identify and remove. Some column headers and values were also confusing and needed to be changed. **I also filtered only on pitchers that started a game, removing any relievers from the equation and leaving close to 14,000 rows ready to be used for feature engineering and modeling.**

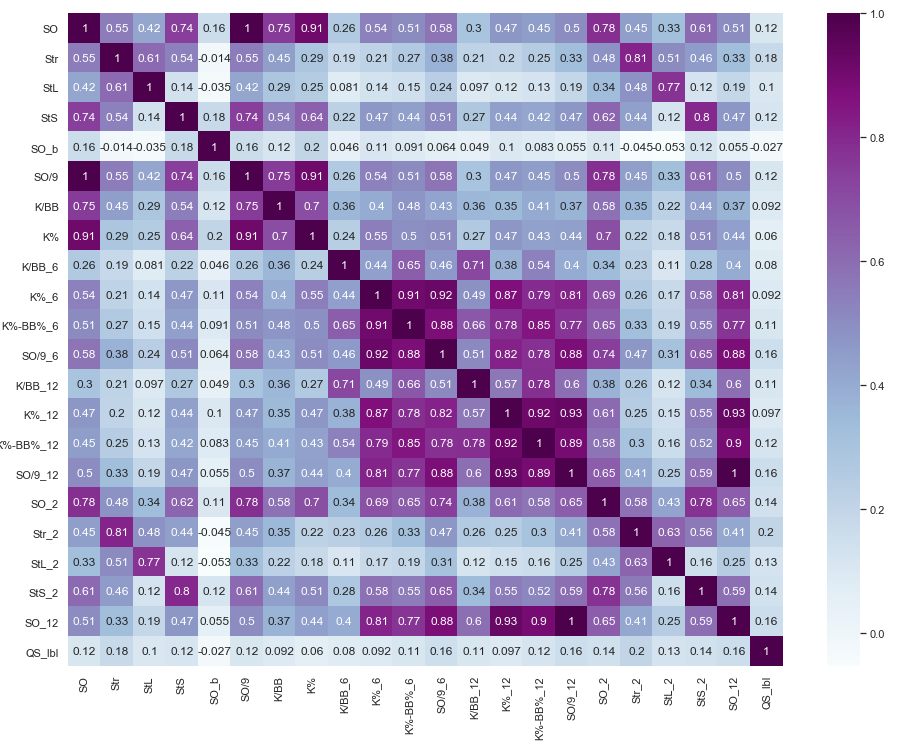
## Feature Engineering and EDA

The raw game log data on its own is valuable, especially when looking at it on a rolling or cumulative basis. But pitchers also have efficiency ratios that are used widely within the league. I implemented these using the game log information. This resulted in 17 new variables derived from game log data alone, before counting the variations from using this on a rolling or cumulative basis.



Many different rolling windows were tested for the game log features, starting with the average of game log stats over 3 games, 5, 15, and 30 games. After testing these windows, I landed on taking the average of game log features over the last 12 games of a pitcher (Example: *R\_12* was the feature for the average runs given up by a pitcher over his last 12 starts). Model performance showed a tendency to improve over longer timelines (> 8 starts), so using a rolling 12 game average was a good way of incorporating recency and track record. The final variants of game log data used in the model were the rolling *average* of raw game log numbers over the past 12 games, rolling six game *average* of the efficiency ratios, rolling two game *sum* of strike related features, and the cumulative *average* game log stats of each pitcher.

After building in these new features, I started the EDA process by creating correlation matrices across different groups of features. A correlation matrix is one of the easiest ways to quickly summarize a lot of information from different features.



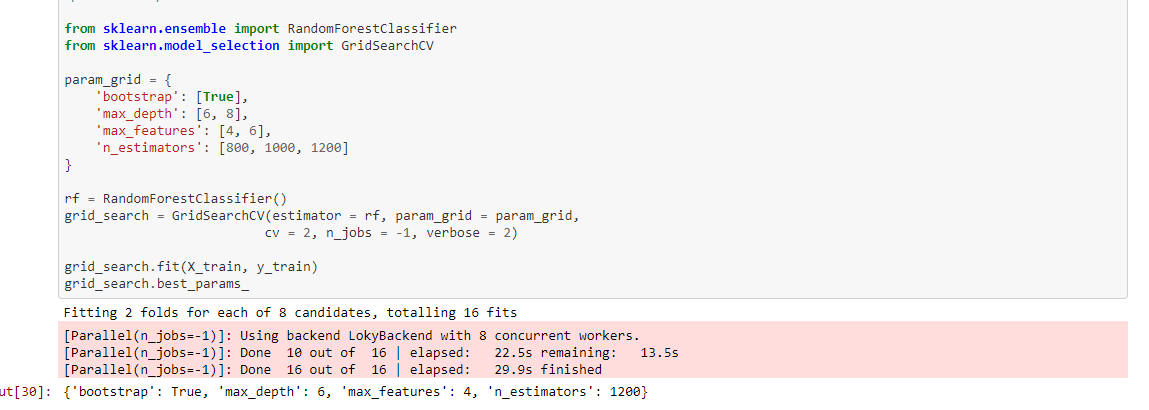
Going through each of the matrices, provided me with a lot of understanding of the data. The highest correlation was only 20%, indicating the challenges that the models would face later in predicting a one game sample. **I also learned that runs, despite being an input of quality starts, had no correlation with quality starts.** This was fascinating because it was always something my co-manager and I looked at in the past when streaming pitchers. Instead, the visualizations during our EDA process seemed to indicate that **variants of strikes and time played had a stronger relationship with quality starts**.

## Modeling

Before I started modeling the data, I needed to narrow down the feature set. I turned to a method called **Mutual Information Classification**, which measures dependency between two random variables. Higher dependency means that an independent variable provides more information regarding the target. I used this approach to select 50 features (out of over 200) for modeling. I also chose **AUC (Area Under the Curve)** as the primary performance metric because it measures how well the model’s predictions rank regardless of scale or threshold. **In this problem, we are less concerned about minimizing one kind of classification error and more concerned with overall model performance.**

To model this data, I used a Decision Tree, Random Forest, and Support Vector Machine. Because I wanted to build a classifier that would output the probability by pitcher, I ended up deciding between the Random Forest and Support Vector Machine. This is because a Decision Tree only outputs classes of probabilities, which is not ideal for ranking pitchers during a season.

To build the Random Forest Classifier, I used a grid search to select the most optimal parameters.



Using these parameters and building a random forest classifier, **resulted in an AUC of 59.9%.** The feature importance table showed average strikes over last 12 games, innings pitched over last 12 games, and the rolling sum of strikes over the last two games as the most important features. These features were similar to the ones picked out by the Decision Tree as well. After building the Random Forest, I also tested a Support Vector Machine. This model performed slightly better than the Random Forest on both the validation and test set and was chosen as the final model, scoring an AUC of just over 60%.

## Conclusion

The SVM model selected will be put into production in April 2021 (start of the next MLB season), where we will get a real time opportunity to see how it performs on new data. While the model will likely struggle to classify pitchers into the correct bucket consistently because it is predicting a one game outcome, it should work well as a decision-making tool during the season. Having the probability by pitcher allows us to make real time informed decisions on pitchers we have never seen play. The probabilities are dynamic too, meaning they will change for each pitcher depending on a given matchup. As we move through the season, I will continuously look at ways the model can be used better and keep an eye out for other features to include to possibly enhance performance.

## Future Work

I only had a month of part time work to complete this project. However, as we test out this model in production in April and I get more time in the future, many improvements can be made. Here are a few of the potential improvements:

**Pitcher-Catcher Data:** I could build a scraper to grab each catcher from a pitcher’s team and include his stats as well. Then we can test the impact of a good catcher on the pitcher’s performance.

**Pitcher vs Hitter Matchup:** One of the most important factors in a pitcher’s performance is likely to be individual batters. Some batters are especially good against certain pitchers.

**Sabermetrics:** Analytics in the MLB is very advanced. We haven’t even scratched the surface of the possible advanced metrics that can still be incorporated for each pitcher.

**More Feature Engineering:** This is one that I will continue to think about, but there is likely to be many more opportunities to create more robust features using the data we already have from the raw game log data.