Predicting Fantasy Points

ISyE 7406

April 23, 2019

Cameron Bradley
Dakota Hauesler
Darcy Johnston
Yuqi Li
Anyu Lu
Xiaoxun Liu
Reuben Tate
Chin-Yuan Tseng
James Wiggins

Abstract

This is the final project for ISyE 7406 Data Mining and Statistical Learning taught by Dr. Jye-Chyi Lu. For this project, we created an ensemble model for predicting DraftKings fantasy points for baseball batters in a single game. We derived the best possible model for a number of different models such as nonlinear regression, nonparametric regression, decision trees, nonlinear support vector regression, and neural networks. We then combined the results of each model in a final ensemble model such that the results made sense intuitively and produced fantasy point predictions that minimized the squared error of the estimate.

Table of Contents

Workload Distribution

Project Idea and Proposal	Cameron
Literature Review	Everyone
Data Collection and Cleaning	Cameron
Model Building/Tuning	
Mixture Model	Cameron
Nonparametric Methods	Cameron and Dakota
Decision Tree	Anyu and Yuqi
Nonlinear SVR	Reuben and Xiaoxun
Artificial Neural Network	Howard, Darcy, and James
Composite Model	Dakota

Introduction

Fantasy baseball is an online game where people assemble imaginary teams of real Major League Baseball players. These imaginary rosters then compete on the statistical performance of the actual players from real games. Daily fantasy baseball is a subset of fantasy baseball where people assemble imaginary rosters of players who will be playing games within the next twenty-four hours. Rosters must be assembled given player positional constraints and a budget constraint. The value of each player is assigned by the site hosting the game, and the individual creating a roster must stay under the salary cap (budget). DraftKings is the most popular site in the world that hosts daily fantasy baseball.

One of the largest reasons DraftKings is popular is that it allows players to bet their lineups against other players. There are many different types of games in which a person can compete with varying entry fees and payouts. If a person's imaginary roster earns more fantasy points (a measure of statistical performance) relative to other players, he wins cash payouts.

The objective of this project is to see if we can use metrics and factors known prior to a game to predict the number of DraftKings fantasy points a batter will earn during the game. If we can accurately predict fantasy points and optimize a lineup (which is outside the scope of this project), this project can increase the probability that we enter a lineup that is better than other entries. Unlike projecting fantasy points for batters over the course of an entire season, predicting fantasy points for batters on a game to game basis is more difficult due to a high level of variability in daily batting performance. Despite this challenge, we desire to build a model or mixture of models that predicts a batter's fantasy points better than the average fantasy points per game of the batter would predict.

In this paper, we first provide a literature review that explores what experts have said about predicting variability in daily human performance. We then research expert opinion about which variables contribute to predicting a batter's performance in fantasy baseball. Our project is novel in the sense that there currently are no papers that discuss how to predict a batter's fantasy points on a game to game basis. While many claim to have researched this topic, all fail to share their work or produce any substantial results. Many sources talk about the factors that go into predicting how well a player will perform, but few of them reveal the relationship between those data points. We plan to incorporate some of the variables these sources mention in our analysis.

After the literature review, we describe how we collected our data, and then discuss how we used the following models to predict a batter's fantasy points per game:

1) Mixture Model: Logistic Regression and Negative Binomial Regression
2) Multivariate Adaptive Regression Splines

3) Local Polynomial
4) Additive Model
5) K-Nearest Neighbor
6) Regression Tree
7) Nonlinear SVR
8) Neural Network

After exploring each model and their properties, we then create a composite model that uses the predictions of the individual models to improve our predictive accuracy of daily fantasy points for batters. Finally, we discuss the limitations of our analysis and potential future opportunities for this field of research.

Literature Review

This literature review is divided into two parts. The first part is more general and reviews literature about important factors included and methods used to predict daily variation in human performance. The second section of the literature review is more specific to predicting fantasy points for batters in DraftKings lineups. After reviewing the literature, we discuss how we used the information in this literature review for our project.

Literature Pertaining to Analyzing and Predicting Variance in Human Performance

Beyond considerations that are specific to the game of baseball, there are many factors that influence human performance on a day to day basis that are important to consider for our analysis. Kevin Wheeler, author of Predicting NBA Player Performance, says one reason that modeling an athlete's performance on a day to day basis is difficult is that there are many behavioral, psychological, philosophical, and social factors that are not captured by the standard statistical categories measured in sports. For example, these factors can include confidence, anxiety, and the type of social relationships the athlete has with his teammates and coaches. Wheeler suggests exploring data from social media and press conferences to incorporate some of these factors that are difficult to model.

In their paper, Team Dynamics: A Social Network Perspective, Warner et al. explore how incorporating social data into models can provide valuable insights on athletic performance. In their paper, the authors use social network analysis to research how team cohesiveness of women's collegiate basketball teams is correlated with team performance. In their study, collegiate team members completed online roster-based surveys related to different types of structural cohesion levels. Their study revealed that higher performing teams are correlated with improved cohesion levels over time.

A second paper called The Home Advantage in Major League Baseball explains how factors that appear more sport specific at first may be more closely related to social phenomena. When brainstorming possible independent variables for building a model to predict player performance, the concept of home field advantage surfaced. A study performed by Marshall B. Jones at The Pennsylvania State University titled The Home Advantage in Major League Baseball states that baseball as a professional sport experiences little to no tendencies of home field advantage. The study found that a home advantage is when a team experiences better teamwork and synergy and that the more a sport is based on individual performance, the less home advantage exists for that sport. It intuitively makes sense that since baseball is more of an individual sport (batter and pitcher matchups), it would experience less home field advantage than another sport like soccer or basketball. We therefore include this factor in our analysis and validate whether the variable is important or not. Both of the above papers imply that social dynamics and data are important to include when analyzing human performance. Data from baseball player's social media accounts may be useful for our research.

In addition to social factors, psychological factors provide valuable insights for evaluating human performance. Predicting Athletic Performance Using the Five-Factor Model of Personality is a study that looks at the five-factor model for describing personalities and applies it to predicting performance in sports. The five factors are neuroticism, extraversion, openness, agreeableness, and conscientiousness. For this study, the performance of 79 female soccer players was collected in two areas, game statistics and coaches' ratings. The game statistics collected were goals, assists, games played, and shots taken on goal. Coaches also rated the players performance on coachability, athletic ability, game performance, team playerness, and work ethic on a 1-7 Likert scale. To measure the five-factor personality traits, players were subjected to an 80 item bipolar adjective scale. Both the coaches' ratings and the personality tests were taken at the end of the season. The study found through a regression analysis that neuroticism and conscientiousness explained approximately 23% of the variation in coaches' ratings and that conscientiousness explaining about 8% of the variance in game statistics. For the purposes of our project, we could use these findings if we had personality metrics on each of the players, but unfortunately we do not.

A second study called Self-Consciousness and Trait Anxiety as Predictors of Choking in Sport in the *Journal of Science and Medicine in Sports* focuses on 66 basketball players who were evaluated pre-performance on various emotional and psychological characteristics to see which could be used to predict whether an athlete would perform better or worse under immense pressure. In this study, the basketball players completed twenty free throws under low and high pressure, and the results were used in multiple regression analysis to evaluate what traits could predict performance. Using self-consciousness and self proclaimed anxiety resulted in an R-squared value of 0.35, which is very high for studies related to human behavior. In relation to this project, if we could label certain games as high stress or low stress, we could leverage this study to determine beforehand which player would perform better or worse than their personal average. Unfortunately, in order to leverage this information for our project, we would need to evaluate players before each game to determine if they were experiencing anxiety or

self-consciousness that day, which is not data we currently have nor is it feasible to collect this data in the time allotted.

Beyond analyzing social and psychological aspects, evaluating biometric indicators shortly before an athletic performance may lead to better predictions during the sporting event. There is a large amount of literature that focuses on using mini tests prior to an athletic performance to predict what an athlete will do during an event. In the *European Journal of Sport Science*, Matthew I. Black et al. use experimental design to test whether data from a three-minute all-out sprint on a standing bike machine conducted prerace could predict a cyclist's performance in a 16.1 km timed trial (TT) race. They found that the critical power (CP) three-minute sprint test was significantly correlated with the performance on the 16.1 km TT race.

Similar studies on how an athlete's pre-performance biometric data can predict the agents actual performance have been conducted in the fields of swimming (Ribeiro et al.), cross-country skiing (Carlsson et al.), rowing (Ozgur), and fencing (Reinberg).

For the purposes of our paper, these articles implicate that data about the physical prowess of the batter immediately before his performance may be useful in predicting his performance for that day. Unfortunately, data regarding a batter's pregame warmup exercises is not publicly available. Also, teams do not publish information about tests they administer to players before games.

Literature Review as to Predicting Batter's DraftKings Fantasy Points

In this section of the literature review, we look at which variables are important for predicting batting fantasy points in DraftKings. We also discuss which analytical methods have previously been used to answer our question of interest.

In the articles "MLB All-Star – Lesson 01 Using Lineups" and "MLB All-Star – Lesson 03 Left/Right Splits," Jonathan Bales, author of *Fantasy Baseball for Smart People* says that the lineup is crucial to predicting the outcome of a game. A player's performance is heavily influenced not only by his position in the lineup but also by the players hitting before and after him. The lineup is also affected by other variables such as the game start time, the previous game's start time, the position of the players, the opposing pitcher and his handedness, the handedness of the batters, and also the weather. All of these variables need to at least be considered when developing models to predict player performance.

In another work by Jonathan Bales, "MLB All-Star – Lesson 04 Ballpark Factors," he discusses the importance of the ballpark. Some players perform better in their home park; others perform better based on the specific park's characteristics. For example, some parks are designed for left-handed batters, and some play better for pitchers. Other parks may have lighting issues that affect player performance. All of these factors need to be considered when building the model.

In his article “MLB All-Star – Lesson 05 Predicting Scores,” Jonathan Bales says that one factor that can be used to predict a player’s scores is Vegas lines and odds. Jonathan Bales says that analytics have become so accurate and payroll sizes so large that Vegas needs to put out good predictions; otherwise, they will lose money. Many of the professional DraftKings players that he has interviewed use Vegas lines in their models as it is usually an accurate predictor of runs scored and player performance. High Vegas lines for a game are correlated with higher player performance.

Josh Kay, author of *An MLB DFS Strategy Guide for the Beginner Player*, emphasizes the importance of including park factor information and handedness of the batter and opposing pitcher when predicting points for batters. He also mentions weighted on base average (wOBA), batting average on balls in play (BABIP), the opposing pitcher’s home run per fly ball rate, the opposing pitcher’s ground ball to fly ball ratio, and weighted runs created as important metrics to include when predicting a batter’s performance in baseball. Weighted on base average is a metric that measures a player’s overall offensive contributions per plate appearance. It values how the runner reached base, not just whether or not they reached base.

In their paper, Beating DraftKings at Daily Fantasy Sport, Barry et al. discovered the importance of the opposing team’s defensive statistics, the number of rest days since the player’s previous game, the recent trend of the player’s fantasy points per game, injury history, and home court advantage were important when it came to predicting an NBA basketball player’s performance in DraftKings. In order to account for the player’s trend of fantasy points per game, they used a weighted metric where more recent games were weighted more than prior games. The defense’s fantasy points allowed per game were also weighted. Barry et. al used multivariate linear regression to analyze the relationship between these metrics and fantasy points and found a statistically significant relationship. These variables and weighted metrics will be taken into consideration for our project.

In his article, Changing an Approach to Daily Fantasy Baseball Part 3: Offense, Matt Trollo discusses the impact of teammates on a player’s performance. Because the number of runs and RBIs a player scores is contributed to by batters that surround the player in the lineup, the on base percentage of players preceding the player and the hitting abilities of the batters following the player need to be considered. He also discusses the metric isolated power which is a metric that measures how many extra bases a player averages per at bat. As doubles are worth 2.5 times more points than singles, triples worth 1.3 times more points than doubles, and homeruns worth 1.25 times more than triples in terms of fantasy points, a power metric like ISO will be relevant for our analysis.

The journal entry “MLB DFS: Where Do Home Runs Come From?” by Dave Pott explores the fundamental metrics that are so often overlooked in elite baseball players, specifically the ones that indicate homerun potential. The bottom line is that homeruns have the most impact on a players overall fantasy points scored, so understanding this factor is critical. Potts takes us back

to fundamental mechanics of baseball saying, “Simply put, home runs come from fly balls, and specifically, fly balls that are hit far.” The way the game works out is that a high number of fly balls results in a lower batting average, a high strikeout rate, and typically worse than average AVG/OBP/SLG. What Potts reminds us about is that you have to have a fly ball in order to have a home run. This means that players who by the numbers have a higher potential for hitting a home run are less popular because they are playing a high stakes game and are more likely to get out. These are all or nothing players. Potts says that people typically look at HR/FB% (ratio of home runs to fly balls) to gauge home run potential, but this number could be artificially high or low depending on how risky the player is being that season. What you need to look for in a home run hitter is proven high power in terms of ISO (isolated power) and a high amount of fly balls.

In a second article called Why Hitting the Ball is Good, Dave Potts also discusses the importance of looking at strikeout rate and groundball to flyball ratio. Batters that strike out frequently do not score points. Also, if batters hit the ball in the air more frequently instead of on the ground, the defense has a greater opportunity of getting an out, which results in no points for the batter.

A study by Gerald T. Mangine, Jay R. Hoffman, Jose Vazquez, Napoleon Pichardo, Maren S. Fragala, and Jeffrey R. Stout called Predictors of Fielding Performance in Professional Baseball Players describes the relationship between preseason performance measures such as grip strength, vertical jump mean, 10-yard sprint, etc. and the same player’s performance over the course of a season in defensive fielding positions. Data for this study was collected on 22 MLB players from the Texas Rangers for five consecutive seasons (2007-2011). The study found that preseason measurements of VJMP (vertical jump) and agility time were statistically significant in predicting defensive performance over the following season. These variables accounted for 49.6% of the variance in UZR/150 (a measure of fielding performance).

A study by Bradbury, JC and Forman, SL called The Impact of Pitch Counts and Days of Rest on Performance AMong Major-League Baseball Pitchers tests the common assumption that pitchers perform worse and are more prone to injury when they are overused. This study examines the performance of MLB pitchers for the years 1988-2009 and concludes that rest days have little to no correlation with performance and that pitch count has a roughly linear relationship with pitcher effectiveness. Age plays a factor in the extent to which pitch load effects the performance of the pitcher. As one would expect, older pitchers are more sensitive to higher pitch counts but still unaffected by days off. The study also reveals that cumulative pitch load is a more significant factor in predicting performance than number of pitches in the previous game. Because both factors effects are small, these metrics would not be useful in predicting future performance.

Intended Contribution to Research Field

The research above has not shown any indication of variables that can accurately predict fantasy points for batters. While many claim to have researched this topic, all fail to share their work or produce any substantial results. Many sources talk about the factors that go into predicting how well a player will perform, but few of them reveal the relationship between those data points. Our project builds on this information we have gathered. Incorporating some of the variables they have mentioned into our analysis, we look to see how prediction tools such as nonparametric regression, neural networks, and random forests perform when using the numerous variables in our dataset to predict fantasy points. Finally, we combine the results of each of these methods into one model that should give us the best prediction.

There are two possible outcomes. One possible outcome is that even our best model is incapable of doing better than the naive approach of just predicting the average of the y-variable (in this case, fantasy points). If this is the case, then our project/results corroborate(s) the claim that none of the variables explored can predict fantasy points for batters. The other outcome is that we are able to predict better than just the naive approach, which would indicate that the claim is false.

Data Extraction and Management

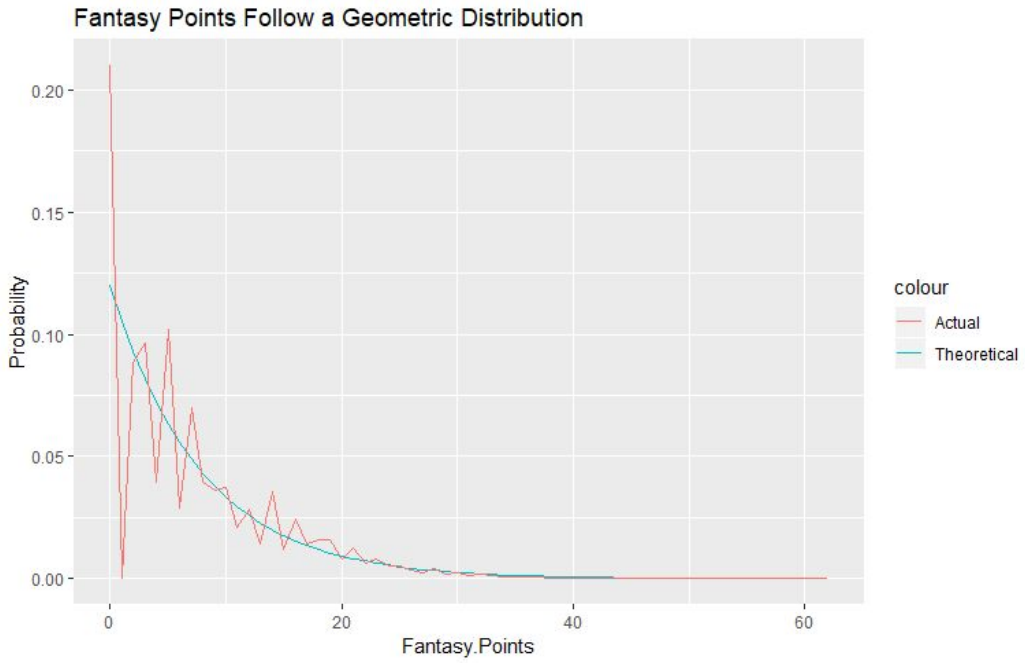
Using Python, we obtained our data by scraping over 6,000 web pages from Baseball-Reference.com. Each web page corresponded to a specific game or player in the 2017 and 2018 season. A text file was written to hold data for every game. From these files, a text file was written to hold data for every player and every team. We then used the player files to create average, weighted average, and last game metrics that corresponded to each batter/game observation. We also appended additional information about the game, team, and opposing team that could potentially influence fantasy points for the player during the game. The data scraping and cleaning process amounted to approximately 110 hours.

Our dataset has 136 variables and 40,557 observations. Each row corresponds to a batter for a specific game. The dataset was built using statistics from 2017 and 2018 Major League Baseball seasons; however, only games from 2018 are included. Observations for batters were not included if the batter did not have at least 30 games prior, if the batter's starting position was pitcher, or if the batter did not start the game. A description of each variable collected is included in **Appendix A**.

Initial Data Exploration and Analysis

Our dependent variable is count data, which means it would be inappropriate to model fantasy points using a simple linear regression. The following plot shows the relationship between

counts of fantasy points per game and their associated probabilities for the 2018 MLB season. The actual data is also compared to the theoretical geometric distribution with probability parameter p equal to 0.12.



The fit looks good except for when the count is equal to zero or one. The fit around one looks bad because it is impossible to score one DraftKings fantasy point in a game due to DraftKings' scoring system. Also, there are a disproportionate number of zeros in our dataset compared to the geometric distribution. This leads us to believe that an appropriate method for predicting fantasy points would be to first use a logit model to identify those who will score fantasy points in a game and then to fit a geometric or negative binomial model that predicts how many fantasy points these identified batters will score. The reason we do not use a Poisson model is because there is overdispersion in our data, or the variance is greater than the mean (we show this in the negative binomial and geometric models section of this paper).

Assuming the data is independently and identically distributed, the geometric distribution models the number of successes until failure (or failures until success). This distribution intuitively makes sense for fantasy points if we think of a batter earning another fantasy point during the course of the game with probability 1-*p* until he cannot any more, *p*. If the geometric distribution with probability equal to 0.12 is a good fit for our response variable, then the probability of a batter getting another fantasy point during the course of the game is approximately 88%.

In the following sections, we explore a mixture model that uses logistic regression to first determine the players who will score followed by a negative binomial regression model that predicts the fantasy points scored by these batters. After producing this model, we try nonparametric, decision tree, nonlinear SVR, and neural network approaches to see if another

model has greater predictive accuracy. Finally, we combine the predictions of these models in an ensemble model for our final prediction.

Predictive Modeling: Methods and Properties

For each of the following models below, we trained the models on 70% of the the data, reserving 20% of the data to validate our models, and using the remaining 10% to test the models selected through validation.

Mixture Model (Logistic and Negative Binomial Regressions)

Logistic Regression

We first created a logistic regression to determine the probability that a batter would score a fantasy point in the game. To decide which variables we would use in the model, we first looked at the correlation of all of the variables in our model, organized the independent variables by most correlated to least correlated, and then used forward selection (using log-likelihood ratio tests) to determine if a variable should be included the model. The following chart shows the correlation for all of the non-categorical variables of interest:

```
cor(numericaldata)[1,]
        FantasyPoints    WeightedFantasyPoints               WeightedS             Weighted2B
           1.00000000               0.14195421              0.05155747              0.08825078
           Weighted3B               WeightedHR               WeightedR            WeightedRBI
           0.02589678               0.10998331              0.13115734              0.10969420
           WeightedBB              WeightedHBP              WeightedSB               WeightedH
           0.08912141               0.01514366              0.05504101              0.10647190
           WeightedSO               WeightedPA              WeightedSH              WeightedSF
           0.03150884               0.10982735             -0.04305484              0.03100933
           WeightedCS               WeightedAB             WeightedWPA            WeightedRE24
           0.01986546               0.09800847              0.09886458              0.11187241
         WeightedaLI              WeightedPit             WeightedStr              WeightedBA
           0.02557130               0.11068786              0.10263551              0.08004051
          WeightedOBP              WeightedSLG             WeightedOPS         WeightedStr_Pit
           0.09721088               0.11941397              0.12165574             -0.05538540
    WeightedSBSuccess              Temperature PitcherWeightedFPAllowed              ParkFactor
           0.03628136               0.03607846              0.05411792              0.05418605
        WeightedwOBA             WeightedBABIP             WeightedISO
           0.11999606               0.03165183              0.10723177
```

After checking for multicollinearity, the model we decided to use (that minimized AIC) included the following independent variables: weighted fantasy points, weighted on base percentage, weighted pitcher fantasy points allowed, park factor, weighted strike outs, the opponent team, the day part, the home plate umpire, and starting lineup position. A printout of the model is included in **Appendix B**.

All of the non-factor variables were statistically significant at an alpha level of 0.05. The coefficients revealed that increasing a batter's weighted fantasy points per game by one would increase the odds that they scored in the game by 7.94%, increasing the weighted pitcher fantasy points allowed per game by one would increased the odds that the batter scored in the

game by 0.35%, increasing the batter's weighted strikeouts per game by one would decrease the odds that he scored in the game by 13.08%, and having the game at night versus during the day would increase the odds that the batter scored by 5.95%. Having a day game gives the batter less time to rest because they normally play their games at night. This may be the reason the odds that the batter scores go down for day games. It was also interesting to see that many of the teams, opposing teams, and umpires had a statistically significant impact on the batter's ability to score during a game. The signs of the coefficients for the batter's lineup position also followed intuition. Relative to the first lineup position, the odds that the player scored continued to decrease for each subsequent lineup position. This is because the further down the lineup the player is, the less opportunities the batter has to get fantasy points.

Although these variables in the model were significant at an alpha level of 0.05, the logistic model had an AIC of 28820 and only explained 3.86% of the deviance in whether or not a batter would score fantasy points. After trying other combinations of dependent variables and checking for nonlinear relationships, we determined that we could not improve the logistic regression model and that other factors outside of our dataset better explained the deviance in whether a batter scored in a game or not.

Cross-validation helped us determine that the best probability cutoff ratio to accurately predict whether or not a batter would score during a game was 0.53. This cutoff ratio resulted in the following confusion matrix for the test set:

	TrueLabels	
Predictions	0	1
0	2	1
1	2531	9634

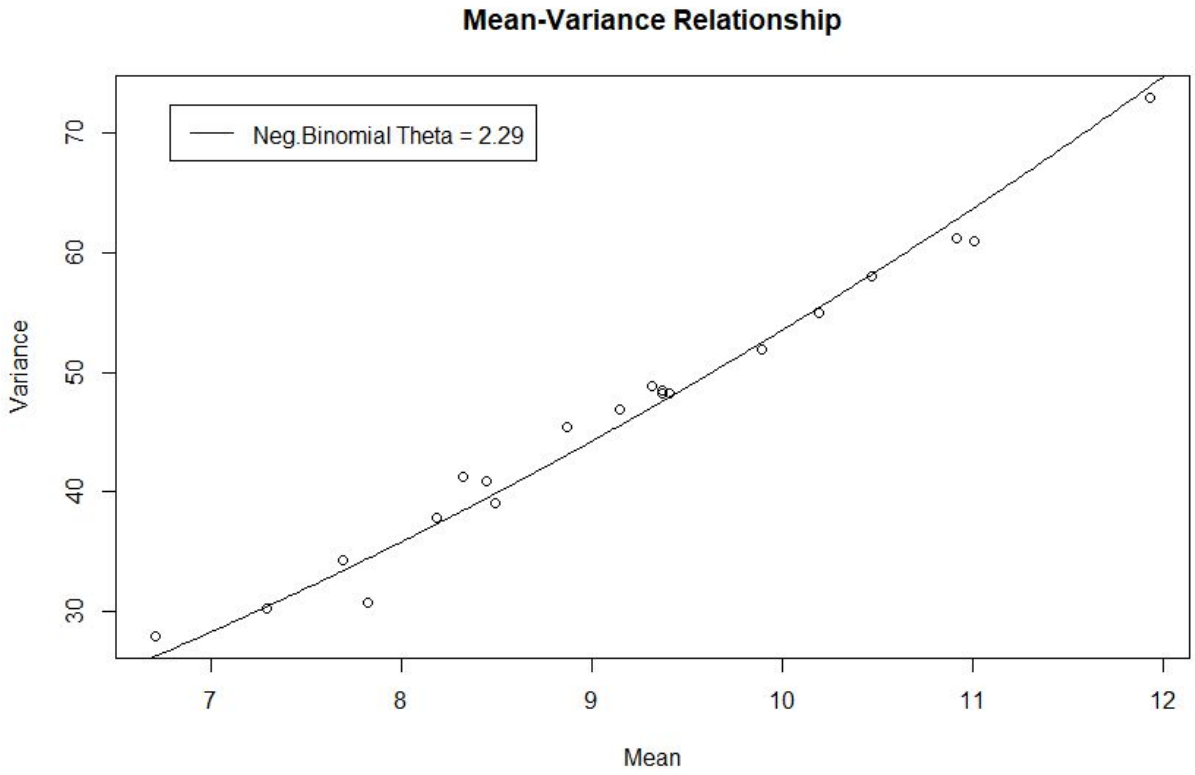
Although the logistic regression with a threshold level of 0.53 accurately predicted if a batter would score about 79% of the time, the confusion matrix shows that the model rarely predicts that an individual will not score. This means that logistic regression contributes little to the mixture model.

Negative Binomial Regression:

Even though our logistic model could not effectively predict which batters would not score during a game, a negative binomial regression model is an appropriate method for predicting the points of batters who do score. This is due to the dependent variable following a negative binomial distribution and to the variable being count data with overdispersion. Thus, we built a negative binomial model that predicts the fantasy points of batters who did score during a game.

To begin, we excluded batters who did not score from our dataset. Next, we used a GLM model and set the family equal to negative binomial. We trained the model on the training set. We used the canonical log link function as our response variable is strictly positive.

To select the appropriate dispersion parameter, we plotted the variance and mean of our dependent variable across different percentiles of the variable and chose the theta (dispersion parameter of the distribution) that best fit the mean-variance relationship. We found that the best dispersion parameter (theta) for our data is equal to 2.299 (See graph below).



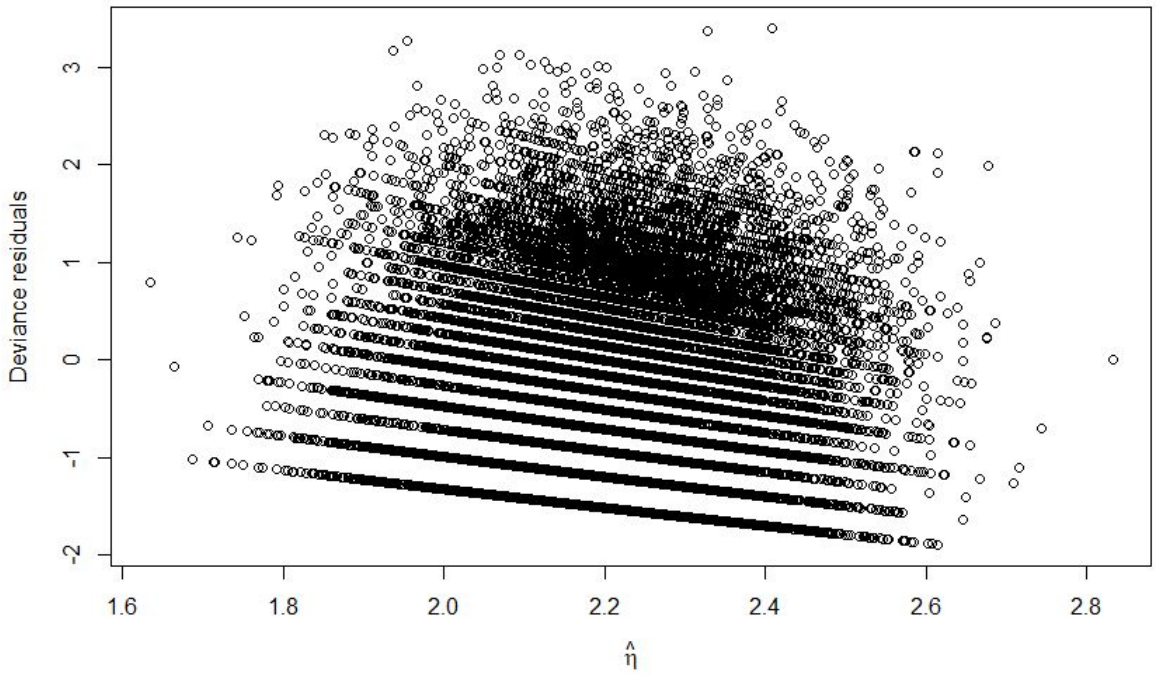
To select our variables, we used the same methodology described in the logit model section, namely look at the correlation of all of the variables in our model with fantasy points, organize the independent variables by most correlated to least correlated with fantasy points, and then use forward selection (using log-likelihood ratio tests) to determine if a variable should be included the model. In **Appendix B**, we show the correlation of the independent variables with fantasy points.

The independent variables that minimized AIC and were not multicollinear included weighted fantasy points, weighted RE24, weighted pitcher fantasy points allowed, weighted stolen bases, weighted singles, park factor, temperature, team opponent team, home field, home plate umpire, and starting lineup position. All of these variables were statistically significant at an alpha level of 0.05. A printout of the model is included in **Appendix B.**

As our dependent variable is a count variable and the model is the log of the expected count as a function of the predictor variables, we interpreted the negative binomial regression coefficients in our model as follows: for a one unit change in the predictor variable, the difference in the logs of expected counts of the response variable is expected to change by the respective regression coefficient given the other predictor variables in the model are held constant. Although the interpretation of the coefficients is not easy, the signs of most of the coefficients are intuitive (e.g. the positive coefficient for weighted average fantasy points shows that if the batter's weighted average fantasy points increase by one unit, the difference in the logs of the expected counts for fantasy points will **increase** by 1.050; increasing by this difference in logs can be pretty significant in actual fantasy points depending on where the beginning point is).

This negative binomial regression model has an AIC of 139822 and only explains 5.83% of the deviance in fantasy points, which is very low. Using a Pearson Chi-Squared test on the model's deviance, we validated whether a negative binomial model was appropriate to explain the relationship between fantasy points and the predictors. With a *p*-value equal to one, we did not reject the null hypothesis that the model was correctly specified.

The residual diagnostic plots show that there is non-constant residual variance across the linear predictors; however, as the regression is negative binomial, we do not need the variance to be constant. There may be an important variable missing from the model or misspecification of a predictor in the model because the residual versus predicted value plot appears to have higher residuals for low predicted values and lower residuals for high predicted values. The residuals are not normally distributed either. Lastly, there are some observations that are outliers, but this is to be expected in a negative binomial distribution.



Predictive Accuracy of Mixture Model

Given the poor predictive accuracy of the logistic model in determining who will score and the low deviance explained by both models, using the mixture model produced worse predictions of batter's fantasy points per game than simply relying on the batter's average fantasy points per game. The RMSE of the mixture model on the test set was 7.216 versus 6.955 for the batter's average points per game. The MAE was 6.010 versus 5.364, respectively.

The negative binomial regression model did better at predicting a batter's fantasy points per game than relying on the batter's average when the data was limited to batters who did score. The RMSE is 6.674 versus 7.008, respectively (the MAE for the average is better, 5.047 versus 5.234). However, because the logistic model cannot accurately predict who will score in a game, it makes the negative binomial regression model pointless.

Given the low predictive power of the mixture model, we try other models to see if we can accurately predict fantasy points. We also do not include the predictions of the mixture model in the final ensemble model.

Nonparametric Methods

MARS

The Multivariate Adaptive Regression Splines (MARS) model uses basis functions [+-(X_i-x_t)]^q, where the basis function is positive. The algorithm begins by selecting an independent variable X by using a forward regression procedure. It then splits the independent variable into two sections and locally fits a polynomial on both sides of the split. The following function is used for step 1: β0 + β1,1 (X_i – x_t)+ + β1,2 (x_t – X_i)+, where the coefficients are estimated with a least-squares criterion. The reflective point (x_t) is also selected from one of the data observations using least-squares optimization.

After this step, the next variable is selected using forward regression procedure. A model similar to that in step 1 is estimated for this variable, but the terms for this model are interacted with all the terms in the model for step 1. This results in the following model for step 2:

β0 + β1,1 (X1 – x1,t1)+ + β1,2 (x1,t1 – X1)+ β0,2,1 (X2 – x2,t2)+ + β1,1,2,1 [(X1 – x1,t1)+ * (X2 – x2,t2)+] + β1,2,2,1 [(x1,t1 – X1)+ * (X2 – x2,t2)+]+ β0,2,2 (x2,t2 – X2)+ + β1,1,2,2 [(X1 – x1,t1)+ * (x2,t2 – X2)+] + β1,2,2,2 [(x1,t1 – X1)+ * (x2,t2 – X2)+]

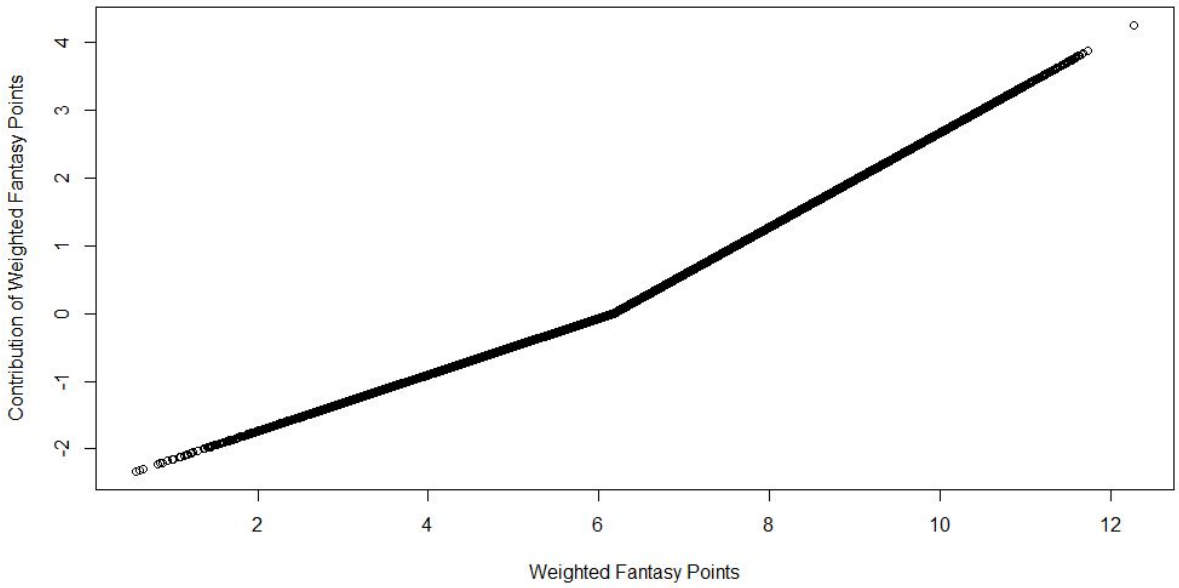
Again, the reflective point for X2 and the coefficients are estimated through least-squares optimization, and local polynomials are fitted to each section of the data. As more terms are added to the model, the above process is repeated. If one reflection point is included per independent variable, the total number of terms in the model will be 3^*p*, where *p* is the number of independent variables included. Because the model uses forward selection, we sorted our independent variables from most correlated with fantasy points to least correlated with fantasy points before applying the MARS algorithm.

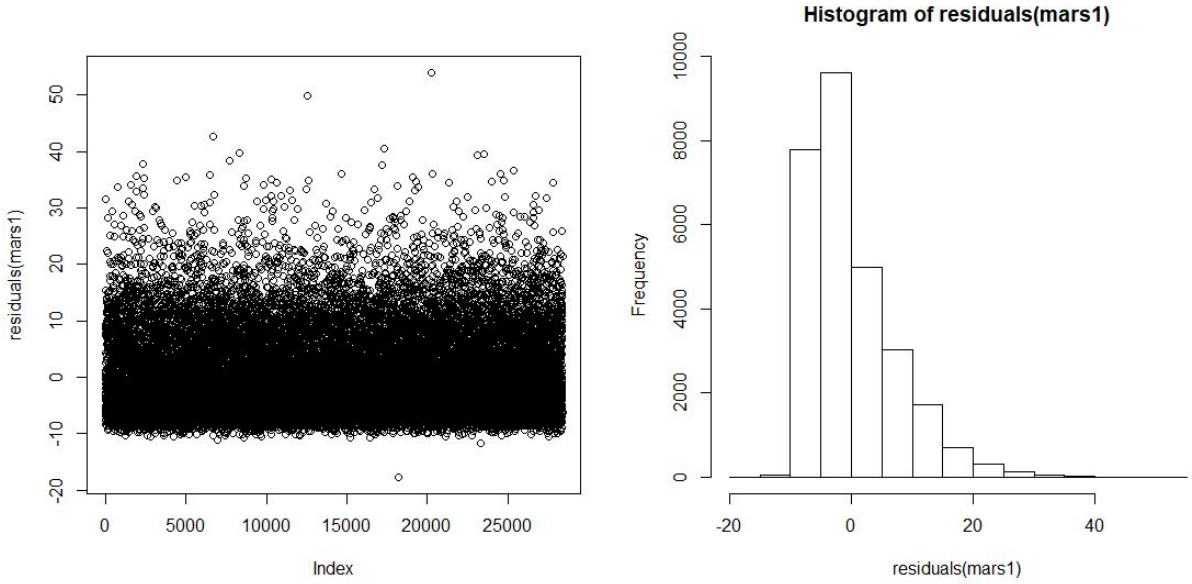
The benefit of using MARS is that it simultaneously considers the interactions between variables and fits splines locally in many dimensions. Because we do not know the functional form of the relationship between the independent variables and fantasy points or if there are significant interactions between the independent variables, the MARS algorithm is appropriate because it will help us discover these relationships.

The initial model included all numeric weighted variables in **Appendix A** in addition to temperature and park factor. We also included factor variables such as team, opponent team, starting lineup position, home plate umpire, batting hand, venue, day part, and home field. In addition, we specified the degree of the spline to be a second-order polynomial. A printout of the model is included in **Appendix B**. The R code for the model is included in **Appendix C.**

Using a partial F-test, the MARS algorithm eliminated all variables except for weighted fantasy points, weighted pitcher fantasy points allowed, and temperature. These variables were statistically significant at an alpha level of 0.05. The model had an R-Squared value of .026, which is quite low.

The batter's weighted average fantasy points per game and the pitcher's weighted fantasy points per game allowed were included in the final model. The interaction between temperature and pitcher weighted fantasy points allowed was also significant. The coefficient for this variable indicates that there is an additional impact to having temperatures greater than 76 degrees Fahrenheit and a pitcher whose weighted fantasy points allowed to a team is greater than 164 per game. This makes intuitive sense. Atmospheric pressure decreases with hot temperatures, allowing a baseball to travel further through the air. Combined with a bad pitcher, this would produce prime conditions for a batter to get extra base hits, which greatly increases fantasy points. A plot of the contributions of the weighted fantasy points in the MARS model and residual plots are included below.





The residual plots show that the errors of the model are not normally distributed and that the model tends to underestimate fantasy points. The RMSE and MAE of the MARS model on the validation set are 7.015 and 5.408, respectively. The prediction error was about the same as relying on the batter's average fantasy points per game for prediction: RMSE of 7.072 and MAE of 5.376. We will try to develop other models to see if we can increase predictive accuracy.

Local Polynomials

A nonparametric method we used to predict batter's fantasy points was a local polynomial model. Local polynomial models fit locally. The fit at each point is made by using the points in a neighborhood of x, weighted by their distance from x (the closer the point is the more weight it receives). The weight designated is (1 - (distance/max(distance))^3)^3. Before distance is measured, the variables are scaled. An *n* degree polynomial is then fitted in the chosen neighborhood. For our model, the degree chosen for the polynomial was 2.

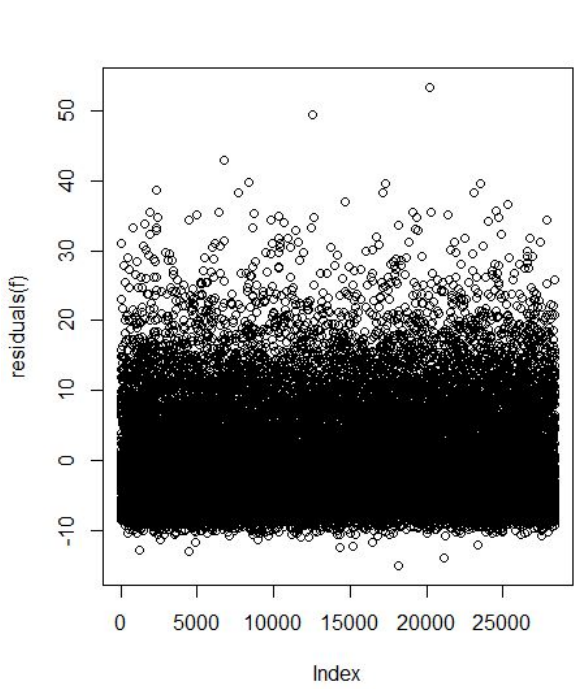
The size of the neighborhood is controlled by a span parameter. The user must specify the window size before the polynomial is fitted. To find the optimal span parameter we used a cross-validation technique to minimize the RMSE on a validation set. We found that the optimal span parameter that minimized the integrated squared error between the estimated and true function for the validation set was 0.75. This means that 75% of the data was used when estimating a specific response. Once the local polynomial fit the response locally, the span window was slid over, and the estimation process was repeated again.

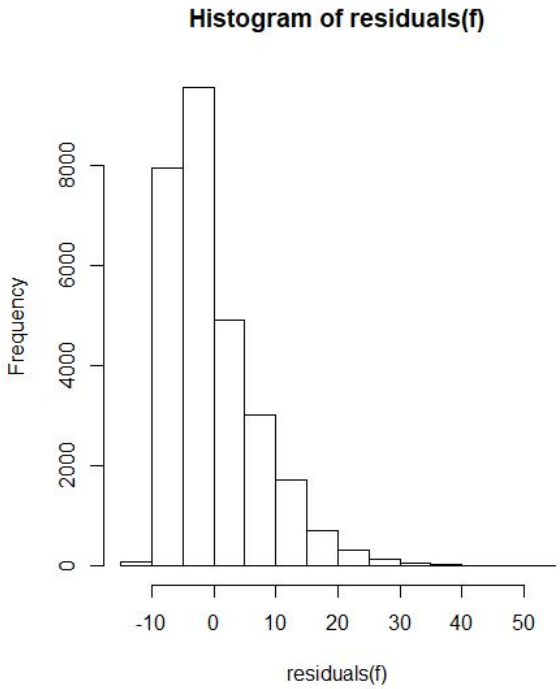
We wanted to use the loess model to estimate batter's fantasy points per game because, unlike kernel and spline methods that are vulnerable to outliers, the local polynomial model avoids giving weight to outliers (see weight function above). Because the response variable follows a

negative binomial distribution, there are a few observations in our data set where the batter scores a large number of points.

The loess package allows up to four variables to be included as predictors because the estimation is computationally intensive. We chose to include three variables that we found to be important predictors in predicting batter fantasy points in prior models. These three variables were weighted fantasy points, park factor, and pitcher weighted fantasy points allowed. These variables capture a lot of information about prior offensive performance, prior defensive performance, and the venue's friendliness to batters versus pitchers. A printout of the model is included in **Appendix B**, and R code for the model is included in **Appendix C**.

The local polynomial model had a RMSE of 7.020 and a MAE of 5.427 for the validation set. The prediction error was about the same as relying on the batter's average fantasy points per game to predict: RMSE of 7.072 and MAE of 5.376. The residual plots show that the local polynomial model tends to underpredict fantasy points for batters. This is because fantasy points per game have a negative binomial distribution with a long right tail. We will now explore an additive model that incorporates the smoothing of the local polynomial model while controlling for factor variables in a parametric fashion.





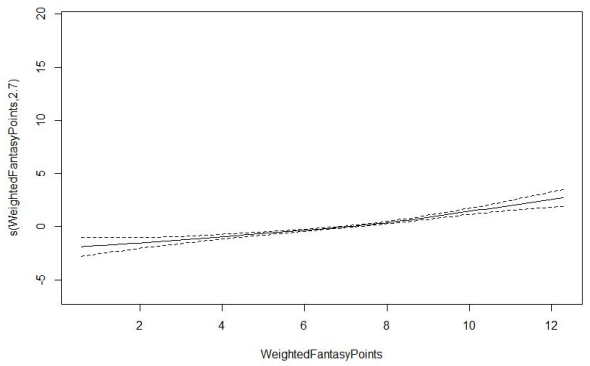
Additive Model

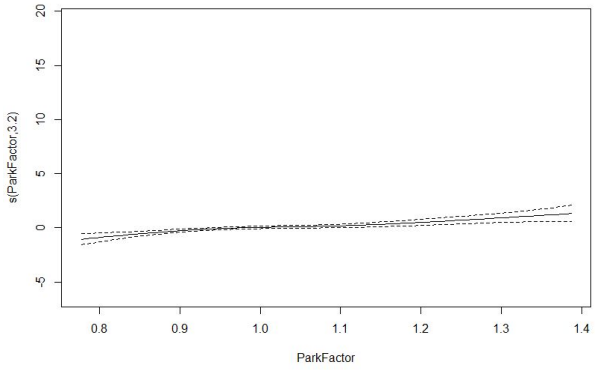
In the additive model, we combined nonparametric and parametric estimation methods. We chose this method to estimate because it was hard to determine the functional relationship between fantasy points and many of numerical independent variables. It also gave us the

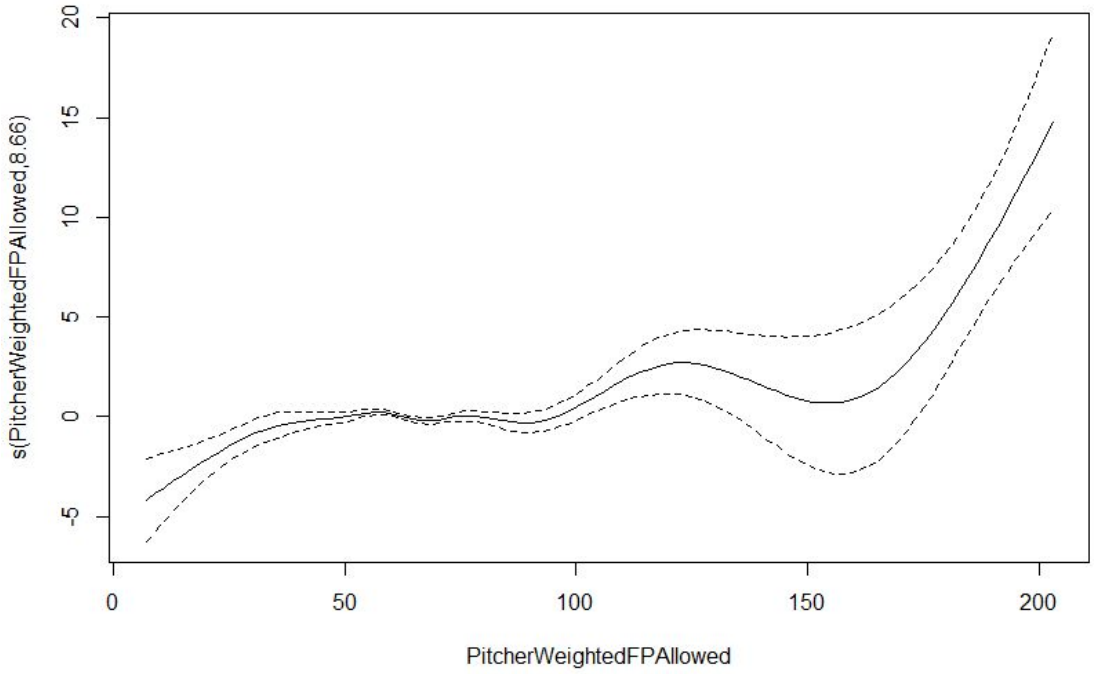
flexibility to estimate the impact of factor variables parametrically. Another benefit of the additive model is that it allowed us to plot the estimated functions of our independent variables with fantasy points. This helped us understand the marginal relationship between our numerical predictors and our dependent variable. To estimate the additive model, we used the gam function from the mgcv package in R.

The mgcv package uses regression splines to smooth nonparametric terms. It uses quadratic penalized likelihood to estimate the parameters. The number of nodes is automatically chosen through cross-validation. The lambda smoothing parameter is also chosen by default through cross-validation. The inputs of the gam function we chose include the variables to be estimated nonparametrically, the variables to be estimated parametrically, and the dependent variable. The initial nonparametric variables we included in the model were all of the weighted variables included in **Appendix A** as well as temperature and park factor. The parametric independent variables we included in the model included team, opposing team, home field, day part, batting hand, starting lineup position, opposing pitcher handedness, and home plate umpire. Using an F-test we eventually eliminated all variables from the model except for weighted fantasy points, park factor, and pitcher fantasy points allowed (all estimated nonparametrically) and team, opponent team, starting lineup position, opposing pitcher handedness, and home plate umpire (all estimated parametrically). A printout of the model is included in **Appendix B**. All of the variables included were statistically significant at an alpha level of 0.05. The deviance of the model explained was 4.14%, which is low but better than other models we used.

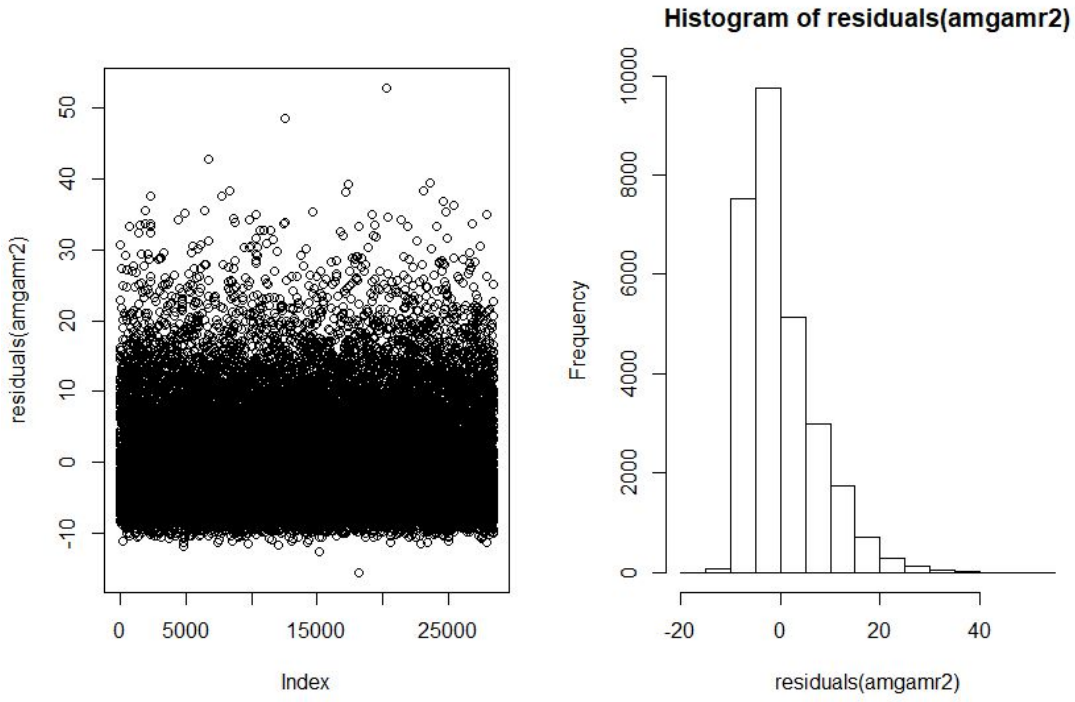
Plots of the nonparametric variables and their marginal contribution to fantasy points show a fairly linear relationship between weighted fantasy points and park factor. However, the marginal contribution of pitcher fantasy points allowed is nonlinear. The rate at which additional fantasy points are gained when the pitcher fantasy points allowed are high is much greater than when the pitcher fantasy points allowed are low.







The residuals of the additive model are similar to those of the local polynomial model (please see this section for commentary on the residuals).



The RMSE of the additive model on the validation set was 6.992, and the MAE was 5.376. This model was the best predictive model out of all models and had better prediction error than using the batter's average fantasy points per game to predict. We will still try additional models to see if we can further improve our predictive accuracy.

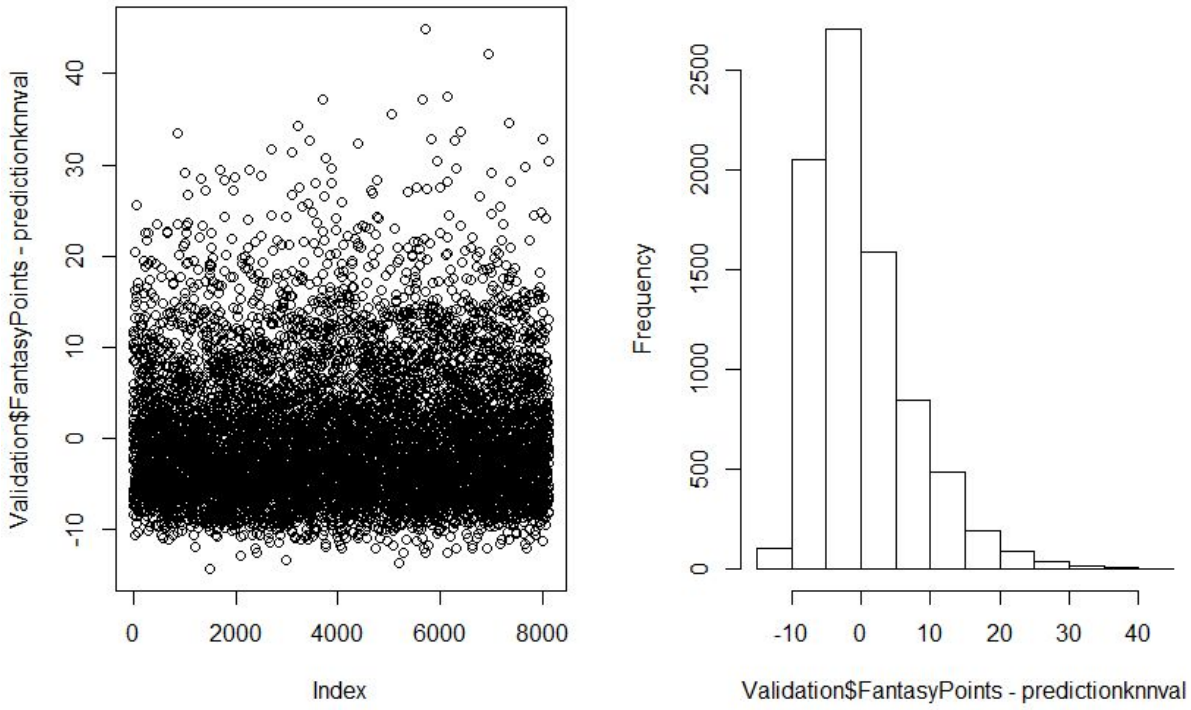
KNN

Another model we used to predict fantasy points for batters per game was K-nearest neighbors. K-nearest neighbors is an algorithm that measures, for point *i*, the euclidean distance in *p*-dimensions to *k* other points. It then uses an average, or weighted average based on distance, of each of the *k* points' dependent variables to predict a value for the dependent variables of point *i*.

For this method, we used the kknn package in R. For the R code, please see **Appendix C**. The *p*-dimensions we included were some of the most correlated variables with fantasy points but not perfectly correlated with each other. We chose $p = 8$ with the eight dimensions being weighted fantasy points, weighted on base plus slugging percentage, weighted plate appearances, weighted pitcher fantasy points allowed, park factor, weighted BABIP, weighted ISO, and weighted OBA. Before including these variables in the model, we scaled them so that the distances in each dimension would be comparable.

To choose the best value for the parameter *k* as well as the weight to assign to each *k*, we used a cross-validation technique where we iterated through different values of *k* (3 to 101 by increments of 2) and different types of kernels (unweighted and gaussian) to find the value of *k* and kernel that would give us the lowest RMSE and MAE on the validation set. We chose an odd number for the values of *k* for tiebreaking purposes. Once we found the best *k* and kernel, we then reran the algorithm on the test set to calculate the model's RMSE and MAE.

The number of neighbors we used was 77, and the optimal kernel was rectangular (unweighted). A plot of the residuals shows that KNN tends to underestimate on average.



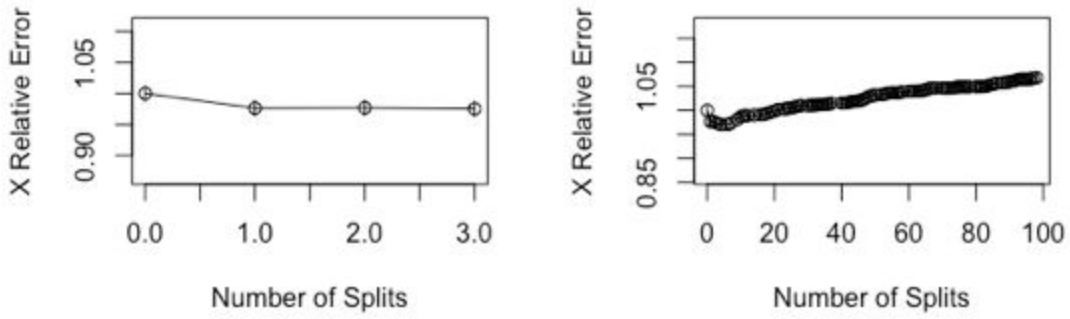
This algorithm resulted in a RMSE of 7.047 and a MAE of 5.423. This error was worse than the error calculated from using the batter's average fantasy points per game for prediction, RMSE of 7.072 and MAE of 5.376. We will continue to explore other models to see if they predict a batter's fantasy points per game better than KNN.

Decision Tree

We also used decision tree regression to predict fantasy points for batters per game. This method breaks down a dataset into smaller and smaller subsets while at the same time incrementally developing an associated decision tree. The final result is a tree with decision nodes and leaf nodes. A decision node has two or more branches, each representing values for the attribute tested. A leaf node represents a decision on the numerical target. The topmost decision node in a tree, which corresponds to the best predictor, is called the root node.

For this method, we used the rpart package in R. For the R code, please see **Appendix C**. Two parameters are included in the decision tree regression model. Minsplit (*n*) is the minimum number of observations that must be in a node before attempting a split. CP is used to control the size of the decision tree and to select the optimal tree size. If the cost of adding another variable to the decision tree from the current node is above the value of CP, then the tree building does not continue. We could also say that tree construction does not continue unless it would decrease the overall lack of fit by a factor of CP.

We tried several pairs of parameters. Our decision tree model did not select any variables in tree construction when CP = 0.1. Next, we set CP at 0.01 and 0.001. Two of the best parameters sets were Minsplit = 30, CP = 0.001 and Minsplit = 30, CP = 0.01.
Although the model with Minsplit = 30, CP = 0.01 had a slightly smaller MSE, we decided to choose the model with Minsplit = 30, CP = 0.001 because that model contained the variable "weighted fantasy point" in the tree construction. We thought that variable was valuable enough to be incorporated in the prediction.



Regression tree model:
rpart(formula = FantasyPoints ~ ., data = batters_train, minsplit = 30,
    cp = 0.001)

Variables actually used in tree construction:
 [1] DurationHours   EndMinutes
 [3] OpponentSPID   Position
 [5] StartLineupPos   Umpire1B
 [7] Umpire2B   Umpire3B
 [9] UmpireHP   WeightedBA
[11] WeightedFantasyPoints WeightedHR

MSE = 53.065
RMSE = 7.285
MAE = 5.517

Nonlinear SVR

Another model that was considered was nonlinear support vector regression (SVR). The software package that was used (the e1071 package in R) did not like working with data with missing values, so to rectify this, we imputed the missing values based on the median (for numerical x-variables) or the mode (for non-numerical values).

Usually a major concern regarding SVM or SVR is that the data needs to be scaled and shifted so that for each variable the mean is zero with unit variance. However, reading through the documentation for the software package that was used, we saw that the software package automatically did such scaling/shifting for us.

Since there are a fair amount of tuning parameters, there were multiple SVR models that we considered. In SVR, there is a tradeoff between maximizing the margin and minimizing the errors. In class, this was represented by the tuning parameter λ, and in the software package, this parameter is just called "cost."

Another important tuning parameter choice is the choice of kernel that is used. If one were to run SVR without a kernel, it would become linear SVR, which would be a poor choice since we know that a linear model would not predict our data well. In many software packages, including this one, the "radial basis" kernel is usually the default choice. Other choices are polynomial and sigmoid kernels. Each of these kernels is really a *family* of kernel functions, each with a set of tuning parameters. However, each family of kernel functions has one tuning parameter in common, namely γ. From a high-level perspective, the γ parameter controls how "detailed" the (nonlinear) margin can be. If γ is set too much in one direction, the boundary will be very detailed and will fit the training data well, but it will also become subject to overfitting. If γ is set too much in the other direction, we won't have the overfitting problem, but our boundary might not be detailed enough to accurately predict our target variable.

For each kernel type, we ran tune.svm, which is a function in the software package that uses cross-validation techniques and grid search in an attempt to find the best values of λ and γ. For the polynomial kernel, we also tried looking at polynomials of various degrees.

In the end, the radial kernel worked out to be the kernel that gave us the best model. However, the overall results were not too spectacular. On the test data, it received a MAE of 5.91 and a RMSE of 7.44, and on the validation data, it received a MAE of 5.90 and a RSME of 7.53.

Neural Network

Lastly, we explored the use of a neural network to predict fantasy points for batters per game. As neural networks tend to perform well without being given any specific rules about the nature of the input data, we were hopeful that a neural network would be able to discover the relationships that exist between our independent variables and dependent variable of fantasy points and, thus, provide more accurate predictions.

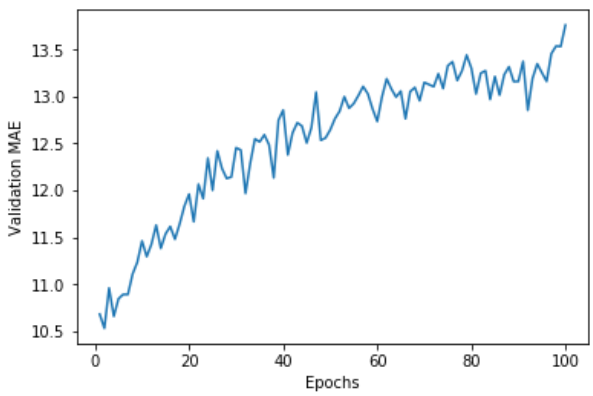
Overall, training a neural network consists of selecting layers, a loss function, and an optimizer. First, the selected layers compose a network that maps the input data to prediction results. The layers consist of an input layer, one or more hidden layers, and an output layer. They are

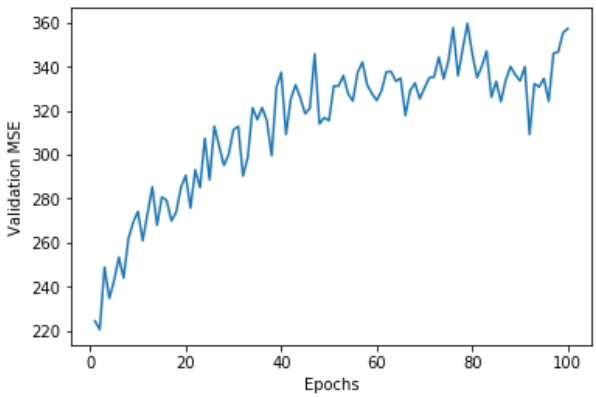
connected through linear combinations and nonlinear transformations of the data. Next, the loss function compares the difference between the predictions and the targets and computes a loss value. Finally, the optimizer updates the network based on the loss value and weights.

Prior to training the neural network, we performed additional preprocessing of the data. Observing the data, we noticed that the various independent variables had different scales. Inputting this data with different orders of magnitude into the neural network would cause it to perform poorly. As a result, we normalized the data for each independent variable by subtracting the mean and dividing by the standard deviation in order to obtain data that was zero-centered with a standard deviation of one. It should be emphasized that the mean and standard deviation used for normalizing all of the data were computed from the training set.

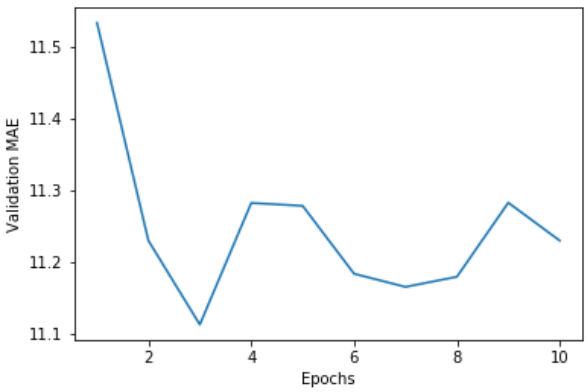
After normalizing the data, we started to build the neural network using Python's keras package. (Please see **Appendix C** for the full Python script.) Although there are several types of layers that can be used to model different data formats, we chose to use densely connected layers for our network as the batters' fantasy points dataset can be viewed as two-dimensional tensors (samples and features). Despite having a large amount of data, we selected two hidden layers with 64 neurons each due to training time constraints. The rectified linear unit (ReLU) was used as the activation function for both of these layers. The last layer of our network comprised of a single linear neuron. RMSprop was used as the optimizer and mean squared error (MSE) was chosen for the loss function. This is a typical setup for scalar regression. It is capable of predicting continuous values in any range.

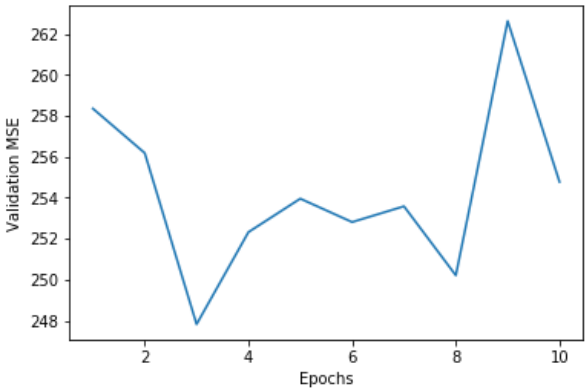
After defining the network layers, we performed four-fold cross-validation to train the neural network and tune the number of epochs. Throughout this process, we employed mean absolute error (MAE) and MSE as the model selection criteria. The relationship between the number of epochs and the MAE and MSE can be seen in the following figures.





As can be seen in the above figures, the validation MAE and MSE initially decreased but then generally increased as the number of epochs increased. As a result, we reran the training process to investigate the model's MAE and MSE for zero to ten epochs. As displayed in the following figures, the neural network with three epochs had the best performance.





After tuning the parameters of the model (tuning the number of hidden layers is omitted here due to training time considerations), we trained a final production model on all of the training data using the best parameters and then evaluated its performance on the validation data. The neural network resulted in a RMSE of 7.77 and a MAE of 5.83 on the validation dataset.

Composite Model

The results of the individual models may not have been spectacular, but that does not mean the project was a failure. The different models each attempted to predict the number of fantasy points that a player will earn in a different way. Therefore, the next step in this project is to look for a relationship between all of those models.

We chose to use simple linear regression to model this relationship. Each of the above models represented a predicting variable in the model. Each row represented a prediction on the validation set. Sample data for training this model can be seen below.

afp	add	dt	knn	lpv	mars	wfp	svr	nn	y
4.412	5.835	6.673	4.429	5.178	5.188	4.332	15.109	7.064	8
8.080	8.145	4.748	8.325	7.61	7.507	7.894	7.662	9.978	10

We checked the variables for multicollinearity, but the prediction results for the smaller model were almost identical to the full model. Thus for this project, we chose to leave all of the variables in the linear model.

The prediction results can be seen in the table below. To understand these results, we added a second model as a baseline. This new model predicted the fantasy points to be the average of the player's previous fantasy points. That is, if the player averages five fantasy points every game, we chose five as the prediction value.

	Within 1	Within 2	Within 5	Within 10	RMSE	MAE
Ensemble	11.59%	34.12%	56.24%	89.35%	6.83	5.40
Average FP	11.34%	33.60%	54.61%	88.76%	7.01	5.40

You can see from these results that our model barely outperforms the baseline case.

Conclusion

The results of this project are not good, barely outperforming the average of past fantasy points. This could be due to any number of reasons, such as the fact that we might not have all of the variables that we need. Another possible reason is that we do not fully understand the interaction between our predicting variables and chose to apply them in the wrong manner. We could also be focussing on the wrong goal. It may be better to break our analysis into a series of steps or to broaden or narrow our scope.

This project has taught us that predicting fantasy baseball points is not an easy task. If it were, the game would likely be a lot less interesting. This challenge is also what inspires us to continue to pursue this topic. When we finally find a good solution, we will have a considerable advantage over the competition.

References

Balasubramaniam, C.P. and Thigarasu, V. (2015) 'Agent-based modeling in supply chain management using improved C4-5', Research Journal of Applied Sciences, Engineering and Technology, 9(2), 91-97.

Bales, Jonathan. “MLB All-Star- Lesson 01- Using Lineups.” *Draft Kings Playbook.* March 31, 2015. https://www.draftkings.com/playbook/mlb/using-mlb-lineups-to-draft/. Accessed February 10, 2019.

Bales, Jonathan. “MLB All-Star- Lesson 03- Left/Right Splits.” *Draft Kings Playbook.* March 31, 2015. https://www.draftkings.com/playbook/mlb/the-importance-of-leftright-splits/. Accessed February 10, 2019.

Bales, Jonathan. “MLB All-Star- Lesson 04- Ballpark Factors.” *Draft Kings Playbook.* March 31, 2015. https://www.draftkings.com/playbook/mlb/ballpark-factors/. Accessed February 10, 2019.

Bales, Jonathan. “MLB All-Star- Lesson 05- Predicting Scores.” *Draft Kings Playbook.* March 31, 2015. https://www.draftkings.com/playbook/mlb/predicting-scores-in-daily-fantasy-baseball. Accessed February 10, 2019.

Bales, Jonathan. “MLB All-Star- Lesson 5- Predicting Scores.” *Draft Kings Playbook.* March 31, 2015. https://www.draftkings.com/playbook/mlb/predicting-scores-in-daily-fantasy-baseball. Accessed February 10, 2019.

Barry, C., Canova, N., and Capiz. K. Beating DraftKings at Daily Fantasy Sports. https://web.stanford.edu/ class/stats50/files/BarryCanovaCapiz-paper.pdf. Accessed February 10, 2019.

Black, M. I., Durant, J., Jones, A. M., & Vanhatalo, A. (2014). Critical power derived from a 3-min all-out test predicts 16.1-km road time-trial performance. *European journal of sport science*, *14*(3), 217-223.

Bradbury, J. C., & Forman, S. L. (2012). The Impact of Pitch Counts and Days of Rest on Performance Among Major-League Baseball Pitchers. Journal of Strength and Conditioning Research, 26(5), 1181-1187. doi:10.1519/jsc.0b013e31824e16fe

Carlsson, M., Carlsson, T., Knutsson, M., Malm, C., & Tonkonogi, M. (2014). Oxygen uptake at different intensities and sub-techniques predicts sprint performance in elite male cross-country skiers. *European journal of applied physiology*, *114*(12), 2587-2595.

Cheng, J., & Wang, J. (2019). An association-based evolutionary ensemble method of variable selection. *Expert Systems With Applications, 124,* 143-155. https://doi.org/10.1016/j.eswa.2019.01.039

Huang, He, et al. “Intelligent Retail Forecasting System for New Clothing Products Considering Stock-Out.” Fibres and Textiles in Eastern Europe, vol. 25, 2017, pp. 10–16., doi:10.5604/01.3001.0010.1704.

Jones, M. B. (2015). THE HOME ADVANTAGE IN MAJOR LEAGUE BASEBALL. Perceptual & Motor Skills: Motor Skills & Ergonomics, 121(3), 791-804. Retrieved February 20, 2019.

Kay, Josh. (2017) An MLB DFS Strategy Guide for the Beginner Player. https://rotogrinders.com/blog-posts/an-mlb-dfs-strategy-guide-for-the-beginner-player-262066. Accessed February 10, 2019.

Liu, S., Yu, J. (2010). Supply Chain Management Research Based on Data Mining, The Conference on Web Based Business Management. http://file.scirp.org/pdf/18-2.6.12.pdf. Accessed February 18, 2019.

Mangine, Gerald & Hoffman, Jay & Vazquez, Jose & Pichardo, Napoleon & Fragala, Maren & Stout, Jeffrey. (2013). Predictors of Fielding Performance in Professional Baseball Players. International journal of sports physiology and performance. 8. 510-6. 10.1123/ijspp.8.5.510.

M.G. Kendall Rank Correlation Methods, Hafner Publishing Co, New York, 1955.

Ozgur, T., Ozgur, B. O., Celik, Y., Guler, T., & Buyukdemirtas, T. (2011). The Relation Between Anaerobic Power and Rowing Ergometer Performance of Elite Rowers. International Archives of Medical Research, 1(1), 21-27.

Piedmont, R. L., Hill, D. C., & Blanco, S. (1999). Predicting athletic performance using the five-factormodel of personality. Personality and Individual Differences, 27(4), 769-777.

Potts, Dave. (2019) Why Hitting the Ball is Good. Rotogrinders. https://rotogrinders.com/articles/mlb-dfs-why-hitting-the-ball-is-good-680776. Accessed February 10, 2019.

Potts, Dave. (2017). MLB DFS: Where Do Home Runs Come From? Rotogrinders. https://rotogrinders.com/articles/mlb-dfs-where-do-home-runs-come-from-664563 Accessed February 20, 2019.

Quinlan, J.R. (1993). C4.5: Programs for machine learning. Morgan Kaufmann, San Francisco.

Reinberg, A., Proux, S., Bartal, J. P., Lévi, F., & Bicakova-rocher, A. (1985). Circadian rhythms in competitive sabre fencers: internal desynchronization and performance. *Chronobiology international*, *2*(3), 195-201.

Ribeiro, J. P., Cadavid, E., Baena, J., Monsalvete, E., Barna, A., & De Rose, E. H. (1990). Metabolic predictors of middle-distance swimming performance. *British Journal of sports medicine*, *24*(3), 196-200.

Trollo, Matt. (2013). Changing an Approach To Daily Fantasy Baseball Part 3: Offense. Rotogrinders. https://rotogrinders.com/blog-posts/changing-an-approach-to-daily-fantasy-baseball-part-3-offense-40846. Accessed February 10, 2019.

Wang, J., Marchant, D., Morris, T., & Gibbs, P. (2004). Self-consciousness and trait anxiety as predictors of choking in sport. Journal of Science and Medicine in Sport, 7(2), 174-185.

Warner, S., Bowers, M. T., & Dixon, M. A. (2012). Team dynamics: A social network perspective. *Journal of Sport Management*, *26*(1), 53-66.

Wheeler, Kevin. (2012). Predicting NBA Player Performance. http://cs229.stanford.edu/proj2012/Wheeler- PredictingNBAPlayerPerformance.pdf. Accessed February 10,2019.

Wu, Z., Lin, W., Zhang, Z., Wen, A., & Lin, L. (2017). An Ensemble Random Forest Algorithm for Insurance Big Data Analysis. *22017 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC)*. doi:10.1109/cse-euc.2017.99

Appendices

Appendix A: Data Definitions

Fantasy Points: Fantasy Points measures the number of Draft King fantasy points the batter earned in a given game. Fantasy Points are calculated as follows: Fantasy Points = 3*S + 5*2B + 8*3B + 10*HR + 2*R + 2*RBI + 2*BB + 2*HBP + 5*SB (see acronyms below).
Weighted Metrics: There are 30 game weighted metrics, where more recent games being weighted more than former games. For example, if the batter has played n games, the most recent game has a weight of n, the second most recent game has a weight of n-1, and so on until you reach the first game, which has a weight of 1. Each of the weights are divided by the sum of the weights. There are some metrics (such as batting average, on base percentage, slugging percentage, on base plus slugging percentage, strikes over pitches, and stolen base success) that are composites of other variables. These metrics were created by calculations using the weighted averages of their composite parts, not by taking a weighted average of the metric.
Average Metrics: These metrics are similar to the weighted metrics, but the average is a simple average, not a weighted average.
Last Game Metrics: These metrics record statistics in their prior game. Composite metrics (such as batting average, on base percentage, slugging percentage, on base plus slugging percentage, strikes over pitches, and stolen base success) correspond to the 2018 season for the batter, not just the prior game.
Singles (S): Hits on which the batter reaches first base safely without the contribution of a fielding error.
Doubles (2B): Hits on which the batter reaches second base safely without the contribution of a fielding error.
Triples (3B): Hits on which the batter reaches third base safely without the contribution of a fielding error.
Home Runs (HR): Hits on which the batter touches all four bases without the contribution of a fielding error.
Runs (R): The number of times a player crosses home plate.
Run Batted In (RBI): The number of runners who score due to a batters' action, except when a batter grounded into a double play or reached on an error.
Bases on Balls (BB): The number of times a hitter does not swing at four pitches called out of the strike zone and is awarded first base.
Hit by Pitch (HBP): The number of times a player is touched by a pitch and is awarded first base as a result.
Stolen Base (SB): The number of bases advanced by the runner while the ball is in the possession of the defense.
Hits (H): The number of times a batter reached base because of a batted fair ball, without error by the defense.

Strikeout (SO): The times the batter received strike three.
Plate Appearances (PA): The number of completed batting appearances (AB + BB + HBP + SH + SF + Catcher's Interference (CI)).
Sacrifice Fly (SF): Fly balls hit to the outfield which although caught for an out, allow a baserunner to advance.
Sacrifice Hit (SH): Bunts which get the runner out, but allow a baserunner to advance.
Caught Stealing (CS): The number of times a runner is tagged out while attempting to steal a base.
At Bat (AB): The number of times a batter faces a pitcher excluding BB, HBP, SH, SF, and CI.
Win Probability Added (WPA): Given average teams, the change in probability caused by the batter during the game. A change of +/- 1 would indicate one win added or lost.
Win Probability Added (WPA+): Sum of positive events for a batter.
Win Probability Subtracted (WPA-): Sum of negative events for a batter.
Base-Out Runs Added (RE24): Given the bases occupied/out situation, how many runs did the batter or baserunner add in the resulting play.
Average Leverage Index (aLI): The average pressure the batter saw in the game, with 1 being average pressure.
Pitches (Pit): The number of pitches the batter receives in a game.
Strikes (Str): The number of pitches in the strike zone and those swung at out of the strike zone.
Batting Average (BA): H/AB
On-Base Percentage (OBP): (H+BB+HBP)/(AB + BB + HBP + SF)
Slugging Percentage (SLG): (S + 2*2B +3*3B +4*HR)/AB
On-Base + Slugging Percentages (OPS): OBP+SLG
Strikes per Pitch (Str_Pit): Str/Pit
Stolen Base Percentage (SB Success): SB/(SB+CS)
Player ID: A unique player identification number.
Game ID: A unique game identification number.
Team: The team the batter plays for.
League: The league the batter plays in (American or National).
Division: The division the batter plays in (East, Central, or West).
Opponent Team, League, and Division: Same as above, but for the opposing team.
HomeAway: Designates whether the batter is the home or away team.
Day: The day of the week the game is played on.
Month: The month of the year the game is played on.
Year: In which year the game is played.
Start Hour: Eastern Standard Time hour for when the game starts.
Start Minute: Minute within the hour for which the game begins.
Time Zone: Start hour time zone (either ET or none given).
Attendance: Number of people who attended the game.
Venue: Where the game is being played.
Duration Hour: How many hours the game lasted
Duration Minutes: The additional minutes past the hour that the game endured.

End Hour: Eastern Standard Time hour for when the game ended.
End Minutes: Minutes past the End Hour for when the game ended.
Day Part: What part of the day the game was played (day or night).
Conditions: Whether the game was played on grass or on turf.
Substitute: The player either started the game or was a substitute (all players within this dataset were starters).
Position: The positions the batter played during the game. (a player must not have started the game as a pitcher to be included in the dataset).
Batting Hand: The batting hand of the batter (either left, right, or both).
Feet: How many feet tall the batter is.
Inches: How many inches past the foot the batter is.
Weight: How many pounds the batter is.
Umpire HP, Umpire 1B, Umpire 2B, Umpire 3B: The names of the home plate, first base, second base, and third plate umpires who are officiating the game.
Temperature: The temperature in Fahrenheit for the start of the game.
Wind Speed: The speed of the wind in miles per hour.
Wind Direction: What direction the wind is blowing.
Sky: Sky conditions at the start of the game (sunny, cloudy, in dome, etc.)
Precipitation: The precipitation conditions during the game (snow, rain, drizzle, no precipitation, etc.)
Start Lineup Position: which turn the player batted in the lineup (1^{st}, 2^{nd}, 3^{rd},…,9^{th})
Opponent Starting Pitcher (SP) Hand: The handedness of the opposing team's starting pitcher (left or right)
Opponent SP ID: a unique identifier of the starting pitcher.
Pitcher Fantasy Points Allowed: (Pitcher[*]FPAllowed): The number of fantasy points a team allows to the opposing team when this starting pitcher is pitching.
Park Factor: This metric takes into account the total number of runs scored by both teams when the team plays at home (RH) and the total number of runs scored by both teams when the team plays away (RA). The metric is then calculated using a quotient that also takes into account the total number of innings played by both teams at home and away. The calculation is as follows: (RH/Innings Home)/(RA/Innings Away).
Isolated Power (ISO): This metric takes a batter's slugging percentage and subtracts their batting average. It is intended to measure the slugging percentage that is attributable to extra base hits.
Batting Average for Balls Put in Play: ((H-HR)/(AB-SO-HR+SF))
Weighted on Base Average: ((.69*BB + .72*HBP + .89*S + 1.27*2B + 1.62*3B + 2.1*HR)/(AB+BB+SF+HBP))

Appendix B: Model Printouts

Logistic Model Printout:

```
Call:
glm(formula = Scored ~ WeightedFantasyPoints + WeightedOBP +
    PitcherWeightedFPAllowed + ParkFactor + WeightedSO + OpponentTeam +
    DayPart + UmpireHP + StartLineupPos, family = binomial, data = Train)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.3598   0.4988   0.6238   0.7238   1.1962

Coefficients:
                                  Estimate Std. Error z value Pr(>|z|)
(Intercept)                     -0.8997096  0.2863577  -3.142 0.001679 **
WeightedFantasyPoints            0.0764042  0.0140994   5.419 5.99e-08 ***
WeightedOBP                      1.4337238  0.5380081   2.665 0.007702 **
PitcherWeightedFPAllowed         0.0035297  0.0014986   2.355 0.018509 *
ParkFactor                       0.8218089  0.1464572   5.611 2.01e-08 ***
WeightedSO                      -0.1402188  0.0699792  -2.004 0.045100 *
OpponentTeamAtlanta Braves       0.1436552  0.1160143   1.238 0.215622
OpponentTeamBaltimore Orioles    0.4217528  0.1211779   3.480 0.000501 ***
OpponentTeamBoston Red Sox       0.0194119  0.1109177   0.175 0.861070
OpponentTeamChicago Cubs         0.0544218  0.1157682   0.470 0.638289
OpponentTeamChicago White Sox    0.4518730  0.1206628   3.745 0.000180 ***
OpponentTeamCincinnati Reds      0.3675309  0.1231135   2.985 0.002833 **
OpponentTeamCleveland Indians   -0.0336825  0.1112476  -0.303 0.762065
OpponentTeamColorado Rockies     0.0873889  0.1175247   0.744 0.457131
OpponentTeamDetroit Tigers       0.1891662  0.1163077   1.626 0.103859
OpponentTeamHouston Astros      -0.1217358  0.1106429  -1.100 0.271220
OpponentTeamKansas City Royals   0.2933757  0.1180588   2.485 0.012955 *
OpponentTeamLos Angeles Angels   0.1761216  0.1143021   1.541 0.123355
OpponentTeamLos Angeles Dodgers -0.0483071  0.1112471  -0.434 0.664120
OpponentTeamMiami Marlins        0.3693904  0.1231650   2.999 0.002707 **
OpponentTeamMilwaukee Brewers    0.0492356  0.1152373   0.427 0.669195
OpponentTeamMinnesota Twins      0.2877430  0.1172197   2.455 0.014099 *
OpponentTeamNew York Mets        0.2174269  0.1203753   1.806 0.070881 .
OpponentTeamNew York Yankees     0.0003756  0.1120242   0.003 0.997325
OpponentTeamOakland Athletics    0.0992206  0.1151896   0.861 0.389035
OpponentTeamPhiladelphia Phillies -0.0189063  0.1158961  -0.163 0.870415
OpponentTeamPittsburgh Pirates   0.3164701  0.1205812   2.625 0.008677 **
OpponentTeamSan Diego Padres     0.2691198  0.1199740   2.243 0.024887 *
OpponentTeamSan Francisco Giants 0.1518416  0.1180161   1.287 0.198227
OpponentTeamSeattle Mariners     0.0572197  0.1136473   0.503 0.614623
OpponentTeamSt. Louis Cardinals  0.3109402  0.1201784   2.587 0.009673 **
OpponentTeamTampa Bay Rays      -0.0844181  0.1129228  -0.748 0.454717
OpponentTeamTexas Rangers        0.2170594  0.1184439   1.833 0.066863 .
OpponentTeamToronto Blue Jays    0.2517545  0.1182149   2.130 0.033202 *
OpponentTeamWashington Nationals 0.0579062  0.1162178   0.498 0.618304
DayPartNight Game                0.0578329  0.0315928   1.831 0.067164 .
UmpireHPAdrian Johnson           0.4579076  0.1830695   2.501 0.012375 *
UmpireHPAlan Porter              0.3881639  0.1816035   2.137 0.032563 *
UmpireHPAlfonso Marquez          0.4526825  0.1822400   2.484 0.012992 *
UmpireHPAndy Fletcher            0.4620555  0.1864571   2.478 0.013209 *
UmpireHPAngel Hernandez          0.3239185  0.1814682   1.785 0.074263 .
UmpireHPBen May                  0.4840023  0.1916211   2.526 0.011543 *
UmpireHPBill Miller              0.2557728  0.1790617   1.428 0.153175
UmpireHPBill Welke               0.4081270  0.1816860   2.246 0.024683 *
UmpireHPBrian Gorman             0.3153822  0.1833128   1.720 0.085349 .
UmpireHPBrian Knight             0.5652960  0.1933219   2.924 0.003454 **
```

UmpireHPBrian O'Nora	0.1647330	0.1865964	0.883	0.377328	
UmpireHPBruce Dreckman	0.0425260	0.1757221	0.242	0.808774	
UmpireHPCarlos Torres	0.4530073	0.1833975	2.470	0.013508	*
UmpireHPCB Bucknor	0.3100630	0.1825928	1.698	0.089487	.
UmpireHPChad Fairchild	0.3257030	0.1789583	1.820	0.068760	.
UmpireHPChad Whitson	0.2371042	0.1710465	1.386	0.165687	
UmpireHPChris Conroy	0.5609171	0.1901565	2.950	0.003180	**
UmpireHPChris Guccione	0.3713194	0.1843794	2.014	0.044021	*
UmpireHPChris Segal	0.3050168	0.1766975	1.726	0.084310	.
UmpireHPCory Blaser	0.4497957	0.1888979	2.381	0.017258	*
UmpireHPDan Bellino	0.0244251	0.1756184	0.139	0.889387	
UmpireHPDan Iassogna	0.3182065	0.2963833	1.074	0.282988	
UmpireHPDavid Rackley	0.3441572	0.1893365	1.818	0.069110	.
UmpireHPDJ Reyburn	0.5844769	0.1872742	3.121	0.001803	**
UmpireHPDoug Eddings	0.2489673	0.1775003	1.403	0.160727	
UmpireHPEd Hickox	0.3382607	0.1805780	1.873	0.061039	.
UmpireHPEric Cooper	0.1491558	0.1802408	0.828	0.407933	
UmpireHPFieldin Culbreth	0.2311854	0.1865143	1.240	0.215158	
UmpireHPGabe Morales	0.3344192	0.1779712	1.879	0.060236	.
UmpireHPGary Cederstrom	0.3583296	0.1790695	2.001	0.045385	*
UmpireHPGerry Davis	0.2238038	0.1787335	1.252	0.210510	
UmpireHPGreg Gibson	0.3607777	0.1825429	1.976	0.048109	*
UmpireHPHunter Wendelstedt	0.2052201	0.1762599	1.164	0.244301	
UmpireHPJames Hoye	0.5110668	0.1851047	2.761	0.005763	**
UmpireHPJansen Visconti	0.3811056	0.1891046	2.015	0.043871	*
UmpireHPJeff Kellogg	0.1448723	0.1958751	0.740	0.459533	
UmpireHPJeff Nelson	0.1367672	0.1774153	0.771	0.440774	
UmpireHPJeremie Rehak	0.3603444	0.1978722	1.821	0.068592	.
UmpireHPJerry Layne	0.2895185	0.1821950	1.589	0.112047	
UmpireHPJerry Meals	0.2878380	0.1777586	1.619	0.105391	
UmpireHPJim Reynolds	0.0985083	0.1748661	0.563	0.573207	
UmpireHPJim Wolf	0.4250448	0.1748657	2.431	0.015070	*
UmpireHPJoe West	0.1956021	0.1739128	1.125	0.260711	
UmpireHPJohn Libka	0.4809511	0.2406648	1.998	0.045670	*
UmpireHPJohn Tumpane	0.2432357	0.1755988	1.385	0.165998	
UmpireHPJordan Baker	0.0010089	0.1776182	0.006	0.995468	
UmpireHPKerwin Danley	0.6101804	0.1877944	3.249	0.001157	**
UmpireHPLance Barksdale	0.4212131	0.1874971	2.247	0.024672	*
UmpireHPLance Barrett	0.4021896	0.1807784	2.225	0.026097	*
UmpireHPLarry Vanover	0.4913437	0.1886479	2.605	0.009199	**
UmpireHPLaz Diaz	0.0269890	0.1761925	0.153	0.878257	

```
UmpireHPManny Gonzalez            0.5464759  0.1944040   2.811 0.004938 **
UmpireHPMark Carlson              0.6592827  0.1919197   3.435 0.000592 ***
UmpireHPMark Ripperger            0.5350103  0.1930389   2.772 0.005580 **
UmpireHPMark Wegner               0.4217061  0.1811108   2.328 0.019889 *
UmpireHPMarty Foster              0.1871497  0.1896531   0.987 0.323741
UmpireHPMarvin Hudson             0.2890097  0.1796976   1.608 0.107767
UmpireHPMike DiMuro               0.2559343  0.2202336   1.162 0.245193
UmpireHPMike Estabrook           -0.0190857  0.1701850  -0.112 0.910707
UmpireHPMike Everitt              1.5907813  0.7519435   2.116 0.034382 *
UmpireHPMike Muchlinski           0.4784325  0.1888731   2.533 0.011306 *
UmpireHPMike Winters              0.5291005  0.1906864   2.775 0.005525 **
UmpireHPNic Lentz                 0.3697945  0.1695882   2.181 0.029217 *
UmpireHPNick Mahrley              0.3970774  0.2046031   1.941 0.052292 .
UmpireHPPat Hoberg                0.4272000  0.1806804   2.364 0.018059 *
UmpireHPPaul Nauert               0.0947938  0.1807538   0.524 0.599975
UmpireHPPhil Cuzzi                0.4337364  0.1881472   2.305 0.021150 *
UmpireHPQuinn Wolcott             0.4422304  0.1885513   2.345 0.019006 *
UmpireHPRamon De Jesus            0.3934441  0.2097001   1.876 0.060625 .
UmpireHPRob Drake                 0.3019318  0.2209636   1.366 0.171803
UmpireHPRoberto Ortiz             0.4103539  0.2298455   1.785 0.074205 .
UmpireHPRon Kulpa                 0.1649877  0.2849203   0.579 0.562545
UmpireHPRyan Additon              0.2062469  0.2073003   0.995 0.319776
UmpireHPRyan Blakney              0.0451154  0.1752197   0.257 0.796809
UmpireHPSam Holbrook              0.2538767  0.1768943   1.435 0.151233
UmpireHPScott Barry               0.1630523  0.1774702   0.919 0.358222
UmpireHPSean Barber               0.2045086  0.1855135   1.102 0.270291
UmpireHPShane Livensparger       -0.0614092  0.2303495  -0.267 0.789784
UmpireHPStu Scheurwater           0.4233067  0.1878310   2.254 0.024218 *
UmpireHPTed Barrett               0.5779049  0.1926299   3.000 0.002699 **
UmpireHPTim Timmons               0.2959229  0.1809877   1.635 0.102040
UmpireHPTodd Tichenor             0.3954675  0.1840085   2.149 0.031620 *
UmpireHPTom Hallion               0.4406499  0.1948770   2.261 0.023749 *
UmpireHPTom Woodring              0.3987650  0.2106402   1.893 0.058343 .
UmpireHPTony Randazzo             0.2222834  0.1801423   1.234 0.217228
UmpireHPTripp Gibson              0.3567241  0.1829383   1.950 0.051180 .
UmpireHPVic Carapazza             0.5407470  0.1883153   2.871 0.004085 **
UmpireHPWill Little               0.0455800  0.1773059   0.257 0.797125
StartLineupPos2                  -0.0582141  0.0652072  -0.893 0.371988
StartLineupPos3                   0.0235362  0.0666160   0.353 0.723855
StartLineupPos4                  -0.0869278  0.0644772  -1.348 0.177596
StartLineupPos5                  -0.2693586  0.0631194  -4.267 1.98e-05 ***
StartLineupPos6                  -0.2037495  0.0642918  -3.169 0.001529 **
StartLineupPos7                  -0.3561470  0.0647092  -5.504 3.72e-08 ***
StartLineupPos8                  -0.3997275  0.0677120  -5.903 3.56e-09 ***
StartLineupPos9                  -0.4533295  0.0773532  -5.861 4.61e-09 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 29269  on 28388  degrees of freedom
Residual deviance: 28556  on 28257  degrees of freedom
AIC: 28820

Number of Fisher Scoring iterations: 4
```

Negative Binomial Regression Model Printout:

```
Call:
glm(formula = FantasyPoints ~ WeightedFantasyPoints + WeightedRE24 +
    PitcherWeightedFPAllowed + WeightedSB + WeightedS + ParkFactor +
    Temperature + Team + OpponentTeam + HomeAway + UmpireHP +
    StartLineupPos, family = negative.binomial(2.298851), data = Gdata)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.8959  -0.9639  -0.3205   0.4778   3.4055

Coefficients:
                                 Estimate Std. Error t value Pr(>|t|)
(Intercept)                     1.428e+00  1.093e-01  13.071  < 2e-16 ***
WeightedFantasyPoints           4.907e-02  5.996e-03   8.184 2.89e-16 ***
WeightedRE24                   -1.886e-01  6.214e-02  -3.034 0.002413 **
PitcherWeightedFPAllowed        1.630e-03  4.870e-04   3.347 0.000817 ***
WeightedSB                      3.832e-01  9.845e-02   3.892 9.97e-05 ***
WeightedS                      -2.967e-01  4.193e-02  -7.077 1.51e-12 ***
ParkFactor                      2.980e-01  5.739e-02   5.192 2.09e-07 ***
Temperature                     1.908e-03  4.527e-04   4.215 2.51e-05 ***
TeamAtlanta Braves              1.970e-02  4.038e-02   0.488 0.625626
TeamBaltimore Orioles          -3.608e-02  4.288e-02  -0.841 0.400086
TeamBoston Red Sox             -2.802e-03  4.020e-02  -0.070 0.944420
TeamChicago Cubs               -3.085e-02  3.995e-02  -0.772 0.440040
TeamChicago White Sox          -1.124e-01  4.196e-02  -2.678 0.007402 **
TeamCincinnati Reds            -7.169e-03  4.061e-02  -0.177 0.859854
TeamCleveland Indians          -4.422e-02  4.058e-02  -1.090 0.275784
TeamColorado Rockies           -4.469e-03  4.035e-02  -0.111 0.911824
TeamDetroit Tigers             -1.342e-01  4.248e-02  -3.160 0.001581 **
TeamHouston Astros              5.581e-03  4.186e-02   0.133 0.893925
TeamKansas City Royals         -1.142e-01  4.210e-02  -2.712 0.006684 **
TeamLos Angeles Angels         -1.560e-02  4.193e-02  -0.372 0.709843
TeamLos Angeles Dodgers         2.130e-02  4.007e-02   0.532 0.595066
TeamMiami Marlins              -2.323e-02  4.293e-02  -0.541 0.588404
TeamMilwaukee Brewers          -3.422e-04  3.977e-02  -0.009 0.993134
TeamMinnesota Twins            -9.705e-02  4.171e-02  -2.326 0.020003 *
TeamNew York Mets               3.824e-02  4.137e-02   0.924 0.355361
TeamNew York Yankees           -2.875e-02  4.099e-02  -0.701 0.483041
TeamOakland Athletics          -2.104e-02  4.131e-02  -0.509 0.610463
TeamPhiladelphia Phillies      -6.461e-02  4.048e-02  -1.596 0.110434
TeamPittsburgh Pirates         -1.742e-02  4.144e-02  -0.420 0.674219
TeamSan Diego Padres           -5.096e-02  4.142e-02  -1.230 0.218649
TeamSan Francisco Giants       -6.966e-02  4.154e-02  -1.677 0.093579 .
TeamSeattle Mariners           -2.087e-02  4.231e-02  -0.493 0.621923
TeamSt. Louis Cardinals         9.364e-03  4.101e-02   0.228 0.819397
TeamTampa Bay Rays             -4.070e-02  4.277e-02  -0.952 0.341253
TeamTexas Rangers              -1.009e-01  4.096e-02  -2.463 0.013802 *
TeamToronto Blue Jays          -5.193e-02  4.223e-02  -1.230 0.218878
TeamWashington Nationals       -1.085e-02  4.056e-02  -0.267 0.789132
OpponentTeamAtlanta Braves      7.088e-02  4.061e-02   1.745 0.080946 .
OpponentTeamBaltimore Orioles   2.226e-01  4.156e-02   5.356 8.61e-08 ***
OpponentTeamBoston Red Sox      8.158e-02  4.121e-02   1.979 0.047778 *
OpponentTeamChicago Cubs        6.589e-02  4.097e-02   1.608 0.107797
OpponentTeamChicago White Sox   1.839e-01  4.169e-02   4.410 1.04e-05 ***
OpponentTeamCincinnati Reds     1.472e-01  4.085e-02   3.604 0.000314 ***
OpponentTeamCleveland Indians   1.288e-01  4.180e-02   3.082 0.002057 **
OpponentTeamColorado Rockies    6.479e-02  4.061e-02   1.595 0.110669
```

OpponentTeamDetroit Tigers	1.585e-01	4.162e-02	3.809	0.000140	***
OpponentTeamHouston Astros	2.215e-02	4.283e-02	0.517	0.605115	
OpponentTeamKansas City Royals	1.929e-01	4.207e-02	4.585	4.57e-06	***
OpponentTeamLos Angeles Angels	9.983e-02	4.151e-02	2.405	0.016189	*
OpponentTeamLos Angeles Dodgers	3.311e-02	4.066e-02	0.814	0.415443	
OpponentTeamMiami Marlins	1.370e-01	4.167e-02	3.289	0.001008	**
OpponentTeamMilwaukee Brewers	3.279e-02	4.080e-02	0.804	0.421604	
OpponentTeamMinnesota Twins	1.522e-01	4.153e-02	3.663	0.000249	***
OpponentTeamNew York Mets	1.239e-01	4.200e-02	2.950	0.003179	**
OpponentTeamNew York Yankees	7.338e-02	4.200e-02	1.747	0.080649	.
OpponentTeamOakland Athletics	7.540e-02	4.221e-02	1.786	0.074103	.
OpponentTeamPhiladelphia Phillies	6.443e-02	4.167e-02	1.546	0.122037	
OpponentTeamPittsburgh Pirates	9.335e-02	4.102e-02	2.276	0.022862	*
OpponentTeamSan Diego Padres	1.058e-01	4.101e-02	2.580	0.009888	**
OpponentTeamSan Francisco Giants	1.012e-01	4.140e-02	2.443	0.014564	*
OpponentTeamSeattle Mariners	1.506e-01	4.251e-02	3.543	0.000397	***
OpponentTeamSt. Louis Cardinals	4.489e-02	4.121e-02	1.089	0.276041	
OpponentTeamTampa Bay Rays	6.075e-02	4.342e-02	1.399	0.161791	
OpponentTeamTexas Rangers	1.273e-01	4.149e-02	3.069	0.002150	**
OpponentTeamToronto Blue Jays	1.771e-01	4.222e-02	4.194	2.75e-05	***
OpponentTeamWashington Nationals	9.257e-02	4.141e-02	2.235	0.025414	*
HomeAwayHome	-1.829e-02	9.969e-03	-1.835	0.066544	.
UmpireHPAdrian Johnson	1.472e-01	6.344e-02	2.321	0.020322	*
UmpireHPAlan Porter	6.573e-02	6.379e-02	1.030	0.302836	
UmpireHPAlfonso Marquez	6.709e-02	6.424e-02	1.044	0.296326	
UmpireHPAndy Fletcher	5.958e-02	6.500e-02	0.917	0.359344	
UmpireHPAngel Hernandez	4.094e-02	6.466e-02	0.633	0.526637	
UmpireHPBen May	7.022e-02	6.541e-02	1.074	0.283029	
UmpireHPBill Miller	-3.849e-02	6.487e-02	-0.593	0.552896	
UmpireHPBill Welke	-1.470e-03	6.410e-02	-0.023	0.981708	
UmpireHPBrian Gorman	1.160e-01	6.568e-02	1.767	0.077317	.
UmpireHPBrian Knight	1.674e-01	6.529e-02	2.563	0.010370	*
UmpireHPBrian O'Nora	1.287e-01	6.788e-02	1.896	0.057921	.
UmpireHPBruce Dreckman	-7.783e-02	6.567e-02	-1.185	0.235958	
UmpireHPCarlos Torres	7.691e-02	6.455e-02	1.192	0.233452	
UmpireHPCB Bucknor	-3.272e-03	6.506e-02	-0.050	0.959892	
UmpireHPChad Fairchild	-2.257e-02	6.437e-02	-0.351	0.725849	
UmpireHPChad Whitson	5.238e-02	6.219e-02	0.842	0.399610	
UmpireHPChris Conroy	-8.824e-03	6.470e-02	-0.136	0.891530	
UmpireHPChris Guccione	1.634e-01	6.448e-02	2.534	0.011284	*
UmpireHPChris Segal	9.001e-02	6.339e-02	1.420	0.155611	
UmpireHPCory Blaser	-1.299e-02	6.496e-02	-0.200	0.841546	
UmpireHPDan Bellino	-1.371e-02	6.686e-02	-0.205	0.837582	
UmpireHPDan Iassogna	2.381e-02	1.044e-01	0.228	0.819661	
UmpireHPDavid Rackley	6.691e-02	6.716e-02	0.996	0.319099	
UmpireHPDJ Reyburn	1.508e-01	6.306e-02	2.392	0.016762	*
UmpireHPDoug Eddings	-2.908e-02	6.467e-02	-0.450	0.652929	
UmpireHPEd Hickox	7.367e-03	6.483e-02	0.114	0.909531	
UmpireHPEric Cooper	1.098e-02	6.621e-02	0.166	0.868303	
UmpireHPFieldin Culbreth	5.646e-02	6.709e-02	0.842	0.399996	
UmpireHPGabe Morales	2.209e-02	6.374e-02	0.347	0.728949	
UmpireHPGary Cederstrom	1.839e-02	6.350e-02	0.290	0.772055	
UmpireHPGerry Davis	2.033e-02	6.558e-02	0.310	0.756602	
UmpireHPGreg Gibson	3.840e-02	6.465e-02	0.594	0.552618	
UmpireHPHunter Wendelstedt	-7.102e-03	6.466e-02	-0.110	0.912534	
UmpireHPJames Hoye	1.051e-01	6.381e-02	1.648	0.099429	.
UmpireHPJansen Visconti	1.795e-01	6.642e-02	2.702	0.006892	**
UmpireHPJeff Kellogg	1.224e-01	7.027e-02	1.743	0.081561	

```
UmpireHPManny Gonzalez             1.454e-01  6.576e-02   2.212 0.027000 *
UmpireHPMark Carlson               1.318e-01  6.335e-02   2.081 0.037419 *
UmpireHPMark Ripperger             7.762e-02  6.525e-02   1.190 0.234211
UmpireHPMark Wegner                8.386e-02  6.406e-02   1.309 0.190519
UmpireHPMarty Foster               1.491e-01  6.860e-02   2.173 0.029796 *
UmpireHPMarvin Hudson              2.052e-02  6.448e-02   0.318 0.750321
UmpireHPMike DiMuro               -1.438e-01  8.082e-02  -1.780 0.075155 .
UmpireHPMike Estabrook            -2.560e-02  6.521e-02  -0.393 0.694691
UmpireHPMike Everitt               1.520e-01  1.585e-01   0.959 0.337672
UmpireHPMike Muchlinski            3.159e-02  6.557e-02   0.482 0.630010
UmpireHPMike Winters               9.528e-02  6.504e-02   1.465 0.142943
UmpireHPNic Lentz                  6.981e-02  6.086e-02   1.147 0.251354
UmpireHPNick Mahrley               3.137e-02  7.079e-02   0.443 0.657636
UmpireHPPat Hoberg                 8.305e-02  6.342e-02   1.310 0.190349
UmpireHPPaul Nauert                5.220e-02  6.763e-02   0.772 0.440201
UmpireHPPhil Cuzzi                 1.891e-05  6.603e-02   0.000 0.999771
UmpireHPQuinn Wolcott              5.897e-02  6.544e-02   0.901 0.367493
UmpireHPRamon De Jesus             4.538e-02  7.357e-02   0.617 0.537368
UmpireHPRob Drake                 -6.086e-03  7.880e-02  -0.077 0.938440
UmpireHPRoberto Ortiz              5.341e-02  7.912e-02   0.675 0.499657
UmpireHPRon Kulpa                  1.725e-01  9.971e-02   1.730 0.083583 .
UmpireHPRyan Additon               1.069e-01  7.497e-02   1.425 0.154076
UmpireHPRyan Blakney               4.671e-02  6.623e-02   0.705 0.480643
UmpireHPSam Holbrook               6.387e-02  6.412e-02   0.996 0.319208
UmpireHPScott Barry                3.993e-02  6.560e-02   0.609 0.542669
UmpireHPSean Barber                8.590e-02  6.809e-02   1.261 0.207170
UmpireHPShane Livensparger         6.662e-02  8.912e-02   0.748 0.454755
UmpireHPStu Scheurwater           -4.546e-02  6.553e-02  -0.694 0.487873
UmpireHPTed Barrett                5.544e-02  6.453e-02   0.859 0.390296
UmpireHPTim Timmons                1.021e-02  6.541e-02   0.156 0.875979
UmpireHPTodd Tichenor              1.124e-01  6.491e-02   1.732 0.083315 .
UmpireHPTom Hallion                1.558e-01  6.759e-02   2.305 0.021190 *
UmpireHPTom Woodring               9.103e-02  7.253e-02   1.255 0.209457
UmpireHPTony Randazzo              8.499e-02  6.537e-02   1.300 0.193591
UmpireHPTripp Gibson               9.296e-02  6.486e-02   1.433 0.151845
UmpireHPVic Carapazza             -4.251e-02  6.430e-02  -0.661 0.508530
UmpireHPWill Little                8.264e-02  6.645e-02   1.244 0.213605
StartLineupPos2                   -2.920e-02  1.997e-02  -1.463 0.143617
StartLineupPos3                   -2.493e-02  2.053e-02  -1.214 0.224581
StartLineupPos4                   -3.604e-02  2.101e-02  -1.716 0.086264 .
StartLineupPos5                   -5.688e-02  2.114e-02  -2.690 0.007151 **
StartLineupPos6                   -1.003e-01  2.141e-02  -4.683 2.85e-06 ***
StartLineupPos7                   -1.104e-01  2.225e-02  -4.962 7.03e-07 ***
StartLineupPos8                   -1.528e-01  2.372e-02  -6.441 1.21e-10 ***
StartLineupPos9                   -1.897e-01  2.828e-02  -6.708 2.02e-11 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for Negative Binomial(2.2989) family taken to be 1.013307)

    Null deviance: 21554  on 22393  degrees of freedom
Residual deviance: 20732  on 22231  degrees of freedom
AIC: 139822

Number of Fisher Scoring iterations: 5
```

Negative Binomial Regression Correlation Printout:

FantasyPoints	WeightedFantasyPoints	WeightedS	Weighted2B
1.00000000	0.14195421	0.05155747	0.08825078
Weighted3B	WeightedHR	WeightedR	WeightedRBI
0.02589678	0.10998331	0.13115734	0.10969420
WeightedBB	WeightedHBP	WeightedSB	WeightedH
0.08912141	0.01514366	0.05504101	0.10647190
WeightedSO	WeightedPA	WeightedSH	WeightedSF
0.03150884	0.10982735	-0.04305484	0.03100933
WeightedCS	WeightedAB	WeightedWPA	WeightedRE24
0.01986546	0.09800847	0.09886458	0.11187241
WeightedaLI	WeightedPit	WeightedStr	WeightedBA
0.02557130	0.11068786	0.10263551	0.08004051
WeightedOBP	WeightedSLG	WeightedOPS	WeightedStr_Pit
0.09721088	0.11941397	0.12165574	-0.05538540
WeightedSBSuccess	Temperature	PitcherWeightedFPAllowed	ParkFactor
0.03628136	0.03607846	0.05411792	0.05418605
WeightedwOBA	WeightedBABIP	WeightedISO	
0.11999606	0.03165183	0.10723177	

MARS Model Printout:

```
> summary(mars1)
Call: earth(formula=FantasyPoints~WeightedFantasyPoints+WeightedR+Weig...), data=Train, degree=2)

                                                coefficients
(Intercept)                                        9.4027643
h(6.17373-WeightedFantasyPoints)                  -0.4159407
h(WeightedFantasyPoints-6.17373)                   0.6971059
h(164.667-PitcherWeightedFPAllowed)               -0.0292650
h(PitcherWeightedFPAllowed-164.667)                0.3188731
h(164.667-PitcherWeightedFPAllowed) * h(Temperature-76)    0.0006539

Selected 6 of 7 terms, and 3 of 224 predictors
Termination condition: RSq changed by less than 0.001 at 7 terms
Importance: WeightedFantasyPoints, PitcherWeightedFPAllowed, Temperature, WeightedR-unused, WeightedOPS-unused, WeightedSLG-unused, WeightedwOBA-unused,
Number of terms at each degree of interaction: 1 4 1
GCV 50.82666    RSS 1441546    GRSq 0.02529477    RSq 0.02615296
```

Local Polynomial Model Printout:

```
Call:
loess(formula = FantasyPoints ~ WeightedFantasyPoints + ParkFactor +
    PitcherWeightedFPAllowed, data = Train, span = 0.75)

Number of Observations: 28389
Equivalent Number of Parameters: 19.2
Residual Standard Error: 7.13
Trace of smoother matrix: 23.4  (exact)

Control settings:
  span     :  0.75
  degree   :  2
  family   :  gaussian
  surface  :  interpolate         cell = 0.2
  normalize:  TRUE
 parametric:  FALSE FALSE FALSE
drop.square:  FALSE FALSE FALSE
```

Additive Model Printout:

```
Family: gaussian
Link function: identity

Formula:
FantasyPoints ~ s(WeightedFantasyPoints) + s(ParkFactor) + s(PitcherWeightedFPAllowed) +
    Team + OpponentTeam + StartLineupPos + OpponentSPHand + UmpireHP

Parametric coefficients:
                              Estimate Std. Error t value Pr(>|t|)
(Intercept)                   5.806403   0.539413  10.764  < 2e-16 ***
TeamAtlanta Braves           -0.002612   0.344107  -0.008 0.993944
TeamBaltimore Orioles        -0.224659   0.353224  -0.636 0.524765
TeamBoston Red Sox            0.612526   0.342072   1.791 0.073362 .
TeamChicago Cubs              0.259091   0.339621   0.763 0.445540
TeamChicago White Sox        -0.209534   0.354317  -0.591 0.554274
TeamCincinnati Reds           0.247605   0.350618   0.706 0.480071
TeamCleveland Indians         0.215185   0.347238   0.620 0.535456
TeamColorado Rockies          0.484077   0.365140   1.326 0.184940
TeamDetroit Tigers           -0.769862   0.352642  -2.183 0.029035 *
TeamHouston Astros            0.446441   0.351846   1.269 0.204504
TeamKansas City Royals       -0.102998   0.354707  -0.290 0.771531
TeamLos Angeles Angels        0.029695   0.355176   0.084 0.933369
TeamLos Angeles Dodgers       0.711409   0.338210   2.103 0.035435 *
TeamMiami Marlins             0.079613   0.362773   0.219 0.826295
TeamMilwaukee Brewers         0.031204   0.340265   0.092 0.926932
TeamMinnesota Twins          -0.146750   0.351930  -0.417 0.676692
TeamNew York Mets             0.570805   0.355448   1.606 0.108313
TeamNew York Yankees          0.259047   0.346366   0.748 0.454527
TeamOakland Athletics         0.221745   0.348827   0.636 0.524985
TeamPhiladelphia Phillies    -0.141409   0.346743  -0.408 0.683409
TeamPittsburgh Pirates        0.298507   0.349018   0.855 0.392406
TeamSan Diego Padres          0.178385   0.345500   0.516 0.605642
TeamSan Francisco Giants     -0.257706   0.347586  -0.741 0.458447
TeamSeattle Mariners          0.113967   0.353780   0.322 0.747349
TeamSt. Louis Cardinals       0.422437   0.347132   1.217 0.223639
TeamTampa Bay Rays            0.630991   0.359715   1.754 0.079417 .
TeamTexas Rangers             0.098595   0.354254   0.278 0.780771
TeamToronto Blue Jays        -0.209346   0.356047  -0.588 0.556556
TeamWashington Nationals      0.397565   0.344288   1.155 0.248204
OpponentTeamAtlanta Braves    0.496429   0.351469   1.412 0.157832
OpponentTeamBaltimore Orioles 1.934643   0.366094   5.285 1.27e-07 ***
OpponentTeamBoston Red Sox    0.167007   0.345859   0.483 0.629187
OpponentTeamChicago Cubs      0.463019   0.345702   1.339 0.180465
OpponentTeamChicago White Sox 1.768130   0.362081   4.883 1.05e-06 ***
OpponentTeamCincinnati Reds   1.524324   0.360287   4.231 2.34e-05 ***
OpponentTeamCleveland Indians 0.459188   0.351631   1.306 0.191603
OpponentTeamColorado Rockies  0.802152   0.373056   2.150 0.031546 *
OpponentTeamDetroit Tigers    1.152261   0.355018   3.246 0.001173 **
OpponentTeamHouston Astros   -0.541102   0.354467  -1.527 0.126890
OpponentTeamKansas City Royals 1.702337  0.363196   4.687 2.78e-06 ***
OpponentTeamLos Angeles Angels 0.623430  0.353653   1.763 0.077940 .
OpponentTeamLos Angeles Dodgers -0.006416 0.339918 -0.019 0.984941
OpponentTeamMiami Marlins     1.765080   0.367144   4.808 1.54e-06 ***
OpponentTeamMilwaukee Brewers 0.141326   0.342629   0.412 0.679993
OpponentTeamMinnesota Twins   1.224972   0.360419   3.399 0.000678 ***
```

OpponentTeamNew York Mets	0.928127	0.360333	2.576	0.010007	*
OpponentTeamNew York Yankees	0.128471	0.348536	0.369	0.712427	
OpponentTeamOakland Athletics	0.495699	0.355141	1.396	0.162792	
OpponentTeamPhiladelphia Phillies	0.727085	0.350127	2.077	0.037844	*
OpponentTeamPittsburgh Pirates	1.000709	0.352402	2.840	0.004519	**
OpponentTeamSan Diego Padres	1.034256	0.355991	2.905	0.003672	**
OpponentTeamSan Francisco Giants	0.683353	0.349989	1.952	0.050889	.
OpponentTeamSeattle Mariners	0.674772	0.355580	1.898	0.057750	.
OpponentTeamSt. Louis Cardinals	0.721160	0.353382	2.041	0.041286	*
OpponentTeamTampa Bay Rays	0.179702	0.359421	0.500	0.617096	
OpponentTeamTexas Rangers	1.442764	0.363265	3.972	7.16e-05	***
OpponentTeamToronto Blue Jays	1.697434	0.362092	4.688	2.77e-06	***
OpponentTeamWashington Nationals	0.432711	0.343715	1.259	0.208068	
StartLineupPos2	-0.238837	0.172679	-1.383	0.166638	
StartLineupPos3	-0.093195	0.173896	-0.536	0.592015	
StartLineupPos4	-0.313763	0.172762	-1.816	0.069358	.
StartLineupPos5	-0.543178	0.175336	-3.098	0.001951	**
StartLineupPos6	-0.866326	0.179333	-4.831	1.37e-06	***
StartLineupPos7	-1.198158	0.185162	-6.471	9.90e-11	***
StartLineupPos8	-1.358818	0.196950	-6.899	5.34e-12	***
StartLineupPos9	-1.512847	0.232875	-6.496	8.36e-11	***
OpponentSPHandRight	0.120620	0.102737	1.174	0.240378	
UmpireHPAdrian Johnson	2.029241	0.534697	3.795	0.000148	***
UmpireHPAlan Porter	0.887265	0.537075	1.652	0.098539	.
UmpireHPAlfonso Marquez	1.237852	0.532693	2.324	0.020145	*
UmpireHPAndy Fletcher	0.947368	0.538186	1.760	0.078368	.
UmpireHPAngel Hernandez	1.519471	0.531641	2.858	0.004265	**
UmpireHPBen May	1.537860	0.555114	2.770	0.005603	**
UmpireHPBill Miller	0.528342	0.536568	0.985	0.324795	
UmpireHPBill Welke	1.268836	0.537305	2.361	0.018209	*
UmpireHPBrian Gorman	1.466218	0.557267	2.631	0.008516	**
UmpireHPBrian Knight	1.938604	0.536207	3.615	0.000300	***
UmpireHPBrian O'Nora	1.442385	0.569089	2.535	0.011264	*
UmpireHPBruce Dreckman	0.262118	0.532051	0.493	0.622259	
UmpireHPCarlos Torres	1.245344	0.544735	2.286	0.022253	*
UmpireHPCB Bucknor	0.685266	0.543328	1.261	0.207234	
UmpireHPChad Fairchild	0.553445	0.531844	1.041	0.298063	
UmpireHPChad Whitson	0.663058	0.520282	1.274	0.202526	
UmpireHPChris Conroy	0.476621	0.538158	0.886	0.375812	
UmpireHPChris Guccione	1.953037	0.547474	3.567	0.000361	***
UmpireHPChris Segal	1.243036	0.536621	2.316	0.020543	*
UmpireHPCory Blaser	0.593075	0.538021	1.102	0.270329	
UmpireHPDan Bellino	0.517661	0.541214	0.956	0.338838	
UmpireHPDan Iassogna	0.632576	0.874467	0.723	0.469449	
UmpireHPDavid Rackley	1.373773	0.572257	2.401	0.016374	*
UmpireHPDJ Reyburn	1.597317	0.530730	3.010	0.002618	**
UmpireHPDoug Eddings	-0.056376	0.533740	-0.106	0.915881	
UmpireHPEd Hickox	0.733140	0.529425	1.385	0.166129	
UmpireHPEric Cooper	0.352957	0.536207	0.658	0.510385	
UmpireHPFieldin Culbreth	1.102100	0.566027	1.947	0.051535	.
UmpireHPGabe Morales	1.423088	0.534349	2.663	0.007744	**
UmpireHPGary Cederstrom	0.847188	0.531235	1.595	0.110779	
UmpireHPGerry Davis	0.530898	0.547072	0.970	0.331837	
UmpireHPGreg Gibson	0.698980	0.524536	1.333	0.182684	
UmpireHPHunter Wendelstedt	0.801182	0.526713	1.521	0.128247	
UmpireHPJames Hoye	1.599600	0.532586	3.003	0.002672	**

```
UmpireHPJames Hoye                1.599600   0.532586   3.003 0.002672 **
UmpireHPJansen Visconti           1.592063   0.554278   2.872 0.004078 **
UmpireHPJeff Kellogg              1.671865   0.597544   2.798 0.005147 **
UmpireHPJeff Nelson               0.682047   0.546867   1.247 0.212338
UmpireHPJeremie Rehak             0.798464   0.588173   1.358 0.174623
UmpireHPJerry Layne               1.357434   0.548509   2.475 0.013338 *
UmpireHPJerry Meals               0.906757   0.534951   1.695 0.090081 .
UmpireHPJim Reynolds              0.584295   0.530713   1.101 0.270922
UmpireHPJim Wolf                  1.312522   0.520173   2.523 0.011633 *
UmpireHPJoe West                  0.597717   0.523881   1.141 0.253904
UmpireHPJohn Libka                1.191536   0.670562   1.777 0.075592 .
UmpireHPJohn Tumpane              0.749225   0.527381   1.421 0.155428
UmpireHPJordan Baker              0.854479   0.545104   1.568 0.116997
UmpireHPKerwin Danley             1.813000   0.532793   3.403 0.000668 ***
UmpireHPLance Barksdale           1.523892   0.543409   2.804 0.005046 **
UmpireHPLance Barrett             0.935129   0.530384   1.763 0.077892 .
UmpireHPLarry Vanover             1.033642   0.545906   1.893 0.058309 .
UmpireHPLaz Diaz                  0.635477   0.543481   1.169 0.242304
UmpireHPManny Gonzalez            2.087767   0.549472   3.800 0.000145 ***
UmpireHPMark Carlson              2.215356   0.534927   4.141 3.46e-05 ***
UmpireHPMark Ripperger            2.049861   0.551783   3.715 0.000204 ***
UmpireHPMark Wegner               1.234211   0.533299   2.314 0.020659 *
UmpireHPMarty Foster              0.793706   0.587273   1.352 0.176543
UmpireHPMarvin Hudson             0.716497   0.535287   1.339 0.180735
UmpireHPMike DiMuro              -0.178696   0.643567  -0.278 0.781272
UmpireHPMike Estabrook            0.248868   0.532013   0.468 0.639941
UmpireHPMike Everitt              2.652930   1.450563   1.829 0.067426 .
UmpireHPMike Muchlinski           0.997072   0.552419   1.805 0.071098 .
UmpireHPMike Winters              2.071816   0.550007   3.767 0.000166 ***
UmpireHPNic Lentz                 1.029666   0.510221   2.018 0.043593 *
UmpireHPNick Mahrley              1.041958   0.600399   1.735 0.082673 .
UmpireHPPat Hoberg                1.263177   0.528426   2.390 0.016834 *
UmpireHPPaul Nauert               1.054828   0.552347   1.910 0.056179 .
UmpireHPPhil Cuzzi                1.066365   0.550634   1.937 0.052803 .
UmpireHPQuinn Wolcott             1.299670   0.542463   2.396 0.016588 *
UmpireHPRamon De Jesus            0.232715   0.628883   0.370 0.711351
UmpireHPRob Drake                 0.962588   0.643299   1.496 0.134579
UmpireHPRoberto Ortiz             1.907640   0.660689   2.887 0.003888 **
UmpireHPRon Kulpa                 0.702123   0.856986   0.819 0.412626
UmpireHPRyan Additon              0.605507   0.615147   0.984 0.324962
UmpireHPRyan Blakney              0.357524   0.541560   0.660 0.509147
UmpireHPSam Holbrook              0.394078   0.535842   0.735 0.462080
UmpireHPScott Barry               0.724765   0.536376   1.351 0.176635
UmpireHPSean Barber               0.892608   0.561997   1.588 0.112235
UmpireHPShane Livensparger        0.368013   0.712999   0.516 0.605755
UmpireHPStu Scheurwater           0.417222   0.545052   0.765 0.443997
UmpireHPTed Barrett               1.201541   0.535915   2.242 0.024967 *
UmpireHPTim Timmons               1.285594   0.538708   2.386 0.017019 *
UmpireHPTodd Tichenor             1.171997   0.537855   2.179 0.029338 *
UmpireHPTom Hallion               1.129307   0.566324   1.994 0.046151 *
UmpireHPTom Woodring              1.394834   0.605674   2.303 0.021289 *
UmpireHPTony Randazzo             0.717847   0.539593   1.330 0.183415
UmpireHPTripp Gibson              0.898240   0.538417   1.668 0.095268 .
UmpireHPVic Carapazza             0.699231   0.537961   1.300 0.193686
UmpireHPWill Little               0.825987   0.542358   1.523 0.127781
---
```

```
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Approximate significance of smooth terms:
                           edf Ref.df      F  p-value
s(WeightedFantasyPoints)  2.701  3.451 37.487  < 2e-16 ***
s(ParkFactor)             3.195  4.047  9.935 4.51e-08 ***
s(PitcherWeightedFPAllowed) 8.657  8.966  9.684 5.71e-15 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

R-sq.(adj) =  0.0356   Deviance explained = 4.14%
GCV = 50.591  Scale est. = 50.287    n = 28389
```

Appendix C: Computer Code

Logistic and Negative Binomial Regression R Code:

```
#Load in data
#Clean the environment
rm(list=ls())
cat("\014")
library(faraway)

setwd("C:/Users/camer/Desktop/Baseball/Dataset Building Scripts")
batters=read.csv("BatterDataset2.csv",header=T)

#Create additional variables
batters$Substitute<-NULL
batters$StartLineupPos<-as.factor(batters$StartLineupPos)
batters$Scored<-ifelse(batters$FantasyPoints <= 1, 0, 1)
batters$PositionNew<-batters$Position
batters$WeightedwOBA<-((.69*batters$WeightedBB+.72*batters$WeightedHBP+.89*batters$W
eightedS+
                1.27*batters$Weighted2B+1.62*batters$Weighted3B+
                2.1*batters$WeightedHR)/(batters$WeightedAB+batters$WeightedBB+
                                         batters$WeightedSF+batters$WeightedHBP)) #missing
-intentional walks in the denominator
batters$WeightedBABIP<-((batters$WeightedH-batters$WeightedHR)/(batters$WeightedAB-bat
ters$WeightedSO-batters$WeightedHR+batters$WeightedSF))
batters$WeightedISO<-batters$WeightedSLG-batters$WeightedBA
batters$AveragewOBA<-((.69*batters$AverageBB+.72*batters$AverageHBP+.89*batters$Avera
geS+
                1.27*batters$Average2B+1.62*batters$Average3B+
                2.1*batters$AverageHR)/(batters$AverageAB+batters$AverageBB+
```

```
                                    batters$AverageSF+batters$AverageHBP)) #missing -intentional
walks in the denominator
batters$AverageBABIP<-((batters$AverageH-batters$AverageHR)/(batters$AverageAB-batters
$AverageSO-batters$AverageHR+batters$AverageSF))
batters$AverageISO<-batters$AverageSLG-batters$AverageBA

fill<-c()
for (i in seq(1,nrow(batters),by=1)){
  fill<-c(fill,gsub("-.+","",batters$PositionNew[i]))
}

batters$PositionNew<-as.factor(fill)

#Set Seed
set.seed(123)

# Split the data into training (70%), validation (20%), and test (10%) sets
mask_train <- sample(nrow(batters), size = floor(nrow(batters) * 0.7))
Train <- batters[mask_train, ]

remaining <- batters[-mask_train, ]
mask_val <- sample(nrow(remaining), size = floor(nrow(remaining) * (2/3)))
Validation <- remaining[mask_val, ]

Test <- remaining[-mask_val, ]

#Summarize the training set
summary(Train)

#We will now run a logistic regression to predict who will score.

#Logistic Regression

numericaldata = dplyr::select(Train, Scored, FantasyPoints, WeightedFantasyPoints,
WeightedS, Weighted2B, Weighted3B, WeightedHR, WeightedR, WeightedRBI, WeightedBB,
                              WeightedHBP, WeightedSB, WeightedH, WeightedSO, WeightedPA,
WeightedSH, WeightedSF, WeightedCS,
                              WeightedAB, WeightedWPA, WeightedRE24, WeightedaLI, WeightedPit,
WeightedStr, WeightedBA,
                              WeightedOBP, WeightedSLG, WeightedOPS, WeightedStr_Pit,
WeightedSBSuccess, Temperature,
                              PitcherWeightedFPAllowed, ParkFactor, WeightedwOBA,
WeightedBABIP, WeightedISO)
```

```
cor(numericaldata)[1,]

require(dplyr)
require(lmtest)
lgMod2 = glm(Scored ~ WeightedFantasyPoints+WeightedOBP+PitcherWeightedFPAllowed+
        ParkFactor+WeightedSO+OpponentTeam+DayPart+UmpireHP+
        StartLineupPos,data=Train, family = binomial)

summary(lgMod2)

vif(lgMod2)

#variable section, did multiple iterations of drop1
drop1(lgMod2,test="Chi")

#deviance explained by the model
deviance(lgMod2)
(1-exp((lgMod2$dev-lgMod2$null)/nrow(Train)))/(1-exp(-lgMod2$null/nrow(Train)))


#Prediction
batters2 = dplyr::select(Train, WeightedFantasyPoints, WeightedOBP,
PitcherWeightedFPAllowed,
                ParkFactor, WeightedSO, OpponentTeam, DayPart, UmpireHP,
                StartLineupPos, Scored)

predprob = predict(lgMod2, batters2, type="response")

optimalprob<-0
largestcount<-100000
for (probCutOff in seq(0,1,by=.01)){
  Train$ScoredPred = ifelse(predprob > probCutOff,1,0)
  resultsTrain = table(Predictions = Train$ScoredPred, TrueLabels = Train$Scored)
  if (nrow(resultsTrain)==2){
        count <- (resultsTrain[2,1]+resultsTrain[1,2])
        if (count<largestcount){
        optimalprob<-probCutOff
        largestcount<-count
        }
  }
}
```

```
probCutOff<-optimalprob
Train$ScoredPred<-NULL

Train$ScoredPred = ifelse(predprob > probCutOff,1,0)
resultsTrain = table(Predictions = Train$ScoredPred, TrueLabels = Train$Scored)
# Train % correctly classified
resultsTrain
(resultsTrain[2,2]+resultsTrain[1,1])/(resultsTrain[2,1]+resultsTrain[2,2]+resultsTrain[1,2]+results
Train[1,1])

#Look at accuracy on test set

batters3 = dplyr::select(remaining, WeightedFantasyPoints, WeightedOBP,
PitcherWeightedFPAllowed,
                ParkFactor, WeightedSO, OpponentTeam, DayPart, UmpireHP,
                StartLineupPos, Scored)

predprob2 = predict(lgMod2, batters3, type="response")

remaining$ScoredPred = ifelse(predprob2 > probCutOff,1,0)
resultsTest = table(Predictions = remaining$ScoredPred, TrueLabels = remaining$Scored)

# Test % correctly classified
resultsTest
(resultsTest[2,2]+resultsTest[1,1])/(resultsTest[2,1]+resultsTest[2,2]+resultsTest[1,2]+resultsTes
t[1,1])

#Creating a dataset of only batters who we predict will score
predprob3 = predict(lgMod2, batters, type="response")

batters$WillScore = ifelse(predprob3 > probCutOff,1,0)

#New data frame including just the records of those expecting to score
battersWillScore = batters[which(batters$WillScore==1),]

nrow(battersWillScore)

#Geometric Regression

#install.packages('surveillance')
#install.packages('zoo')
library(surveillance)
library(MASS)
```

```
library(lmtest)
library(boot)

#Get the data for where the players scored
Test1<-Test[which(Test$Scored==1),]
Validation1<-Validation[which(Validation$Scored==1),]
Train1<-Train[which(Train$Scored==1),]
remaining1<-remaining[which(remaining$Scored==1),]

Gdata<-Train1

#Graph fantasy Points
hist(Gdata$FantasyPoints)

cor(numericaldata)[2,]

 #First geometric regression

gmodv1<-glm(FantasyPoints~WeightedFantasyPoints+WeightedRE24+
          PitcherWeightedFPAllowed+WeightedSB+WeightedS+ParkFactor+

Temperature+Team+OpponentTeam+HomeAway+UmpireHP+StartLineupPos,negative.binomia
l(1),Gdata)

summary(gmodv1)

#Check for multicollinearity
vif(gmodv1)

plot(Train$WeightedFantasyPoints,Train$FantasyPoints)
plot(Train$WeightedR,Train$FantasyPoints)
plot(Train$WeightedOPS,Train$FantasyPoints)
plot(Train$WeightedwOBA,Train$FantasyPoints)
plot(Train$WeightedSLG,Train$FantasyPoints)
plot(Train$WeightedPA,Train$FantasyPoints)
plot(Train$WeightedPit,Train$FantasyPoints)
plot(Train$WeightedRE24,Train$FantasyPoints)
plot(Train$WeightedRBI,Train$FantasyPoints)
plot(Train$WeightedH,Train$FantasyPoints)
plot(Train$WeightedHR,Train$FantasyPoints)
plot(Train$WeightedISO,Train$FantasyPoints)
plot(Train$WeightedStr,Train$FantasyPoints)
plot(Train$WeightedAB,Train$FantasyPoints)
```

```
plot(Train$WeightedOBP,Train$FantasyPoints)
plot(Train$WeightedWPA,Train$FantasyPoints)
plot(Train$Weighted2B,Train$FantasyPoints)
plot(Train$WeightedBB,Train$FantasyPoints)
plot(Train$WeightedBA,Train$FantasyPoints)
plot(Train$PitcherWeightedFPAllowed,Train$FantasyPoints)
plot(Train$WeightedSB,Train$FantasyPoints)
plot(Train$WeightedS,Train$FantasyPoints)
plot(Train$ParkFactor,Train$FantasyPoints)
plot(Train$WeightedSBSuccess,Train$FantasyPoints)
plot(Train$Temperature,Train$FantasyPoints)
plot(Train$WeightedSF,Train$FantasyPoints)
plot(Train$WeightedBABIP,Train$FantasyPoints)
plot(Train$Weighted3B,Train$FantasyPoints)
plot(Train$WeightedSO,Train$FantasyPoints)
plot(Train$WeightedaLI,Train$FantasyPoints)
plot(Train$WeightedCS,Train$FantasyPoints)
plot(Train$WeightedHBP,Train$FantasyPoints)
plot(Train$WeightedSH,Train$FantasyPoints)
plot(Train$WeightedStr_Pit,Train$FantasyPoints)

#Store dispersion parameter (theta)
dpv1 <-sum(residuals(gmodv1,type="pearson")^2)/gmodv1$df.res

#Deviance test (Pearson's Chi-Squared Test) to see if the model is correctly specified
pchisq(deviance(gmodv1),df.residual(gmodv1),lower=FALSE)
#The model is correctly specified

#Is the model a good fit (Deviance Explained)?
(1-exp((gmodv1$dev-gmodv1$null)/nrow(Gdata)))/(1-exp(-gmodv1$null/nrow(Gdata)))
#The model only explains about 4.5% of the deviance.

#See if the mean and variance follow a geometric distribution
x <- seq(1,12,0.02)
xb <- predict(gmodv1)
g <- cut(xb, breaks=quantile(xb,seq(0,100,5)/100))
m <- tapply(Gdata$FantasyPoints, g, mean)
v <- tapply(Gdata$FantasyPoints, g, var)
plot(m, v, xlab="Mean", ylab="Variance", main="Mean-Variance Relationship")
lines(x, x*(1+x/(1/0.4982022)))
legend("topleft", lty=c("solid"), legend=c("Geometric"), inset=0.05)
```

```
#The dispersion parameter of the geometric distibution is close to exactly matching our data, but
#is not perfect. We will examine negative binomial distributions.

nmodv1<-glm.nb(FantasyPoints~WeightedFantasyPoints+WeightedRE24+
        PitcherWeightedFPAllowed+WeightedSB+WeightedS+ParkFactor+
        Temperature+Team+OpponentTeam+HomeAway+UmpireHP+StartLineupPos,Gdata)

#Store dispersion parameter (theta)
nbdpv1 <-sum(residuals(nmodv1,type="pearson")^2)/nmodv1$df.res

summary(nmodv1)

#THe AIC of this model is better.

#Deviance test (Pearson's Chi-Squared Test) to see if the model is correctly specified
pchisq(deviance(nmodv1),df.residual(nmodv1),lower=FALSE)
#The model is NOT correctly specified.

#See if the mean and variance follow a negative binomial distribution
x <- seq(1,12,0.02)
xb <- predict(nmodv1)
g <- cut(xb, breaks=quantile(xb,seq(0,100,5)/100))
m <- tapply(Gdata$FantasyPoints, g, mean)
v <- tapply(Gdata$FantasyPoints, g, var)
plot(m, v, xlab="Mean", ylab="Variance", main="Mean-Variance Relationship")
lines(x, x*(1+x/(1/(1/2.6098))))
legend("topleft", lty=c("solid"), legend=c("Neg.Bin.w/min AIC"), inset=0.05)

#The negative binomial model explains more of the deviance, has better AIC, but
#fails to pass the deviance Pearson Chi-Squared test (which, means the model is
#not correctly specified). There may be another negative binomial model that we can
#use.

nmodv1<-glm(FantasyPoints~WeightedFantasyPoints+WeightedRE24+
        PitcherWeightedFPAllowed+WeightedSB+WeightedS+ParkFactor+

Temperature+Team+OpponentTeam+HomeAway+UmpireHP+StartLineupPos,negative.binomia
l(2.298851),Gdata)

#Store dispersion parameter (theta)
nbdpv1 <-sum(residuals(nmodv1,type="pearson")^2)/nmodv1$df.res

summary(nmodv1)
```

```
#THe AIC of this model is better than the geometric regression.

#Deviance test (Pearson's Chi-Squared Test) to see if the model is correctly specified
pchisq(deviance(nmodv1),df.residual(nmodv1),lower=FALSE)
#The model is correctly specified.

#Is the model a good fit (Deviance Explained)?
(1-exp((nmodv1$dev-nmodv1$null)/nrow(Gdata)))/(1-exp(-nmodv1$null/nrow(Gdata)))
#The model only explains about 5.7% of the deviance.

#See if the mean and variance follow a negative binomial distribution
x <- seq(6.5,12,0.02)
xb <- predict(nmodv1)
g <- cut(xb, breaks=quantile(xb,seq(0,100,5)/100))
m <- tapply(Gdata$FantasyPoints, g, mean)
v <- tapply(Gdata$FantasyPoints, g, var)
plot(m, v, xlab="Mean", ylab="Variance", main="Mean-Variance Relationship")
lines(x, x*(1+x/(1/(1/2.298851))))
legend("topleft", lty=c("solid"), legend=c("Neg.Binomial"), inset=0.05)

#We will keep both the geometric and negative binomial regressions to see which one has
#smaller prediction error through cross validation.

drop1(nmodv1,test="F")
drop1(gmodv1,test="F")

#Now we will run diagnostic plots to look at the residuals

#Diagnostics
plot(residuals(gmodv1) ~ predict(gmodv1,type="link"),
xlab=expression(hat(eta)),ylab="Deviance residuals")
plot(residuals(nmodv1) ~ predict(nmodv1,type="link"),
xlab=expression(hat(eta)),ylab="Deviance residuals")

#Looking at fit of model
glm.diag.plots(gmodv1)
glm.diag.plots(nmodv1)

#Plot Response Residuals to see if the variance is non constant
plot(residuals(gmodv1,type='response') ~
predict(gmodv1,type="link"),xlab=expression(hat(eta)),ylab="Deviance residuals")
```

```
plot(residuals(nmodv1,type='response') ~
predict(nmodv1,type="link"),xlab=expression(hat(eta)),ylab="Deviance residuals")

#The link function
#As this is positive count data, the log link function is appropriate.

#Cross Validation
set.seed(123)
(cv.err.gmodv110 <- cv.glm(Gdata, gmodv1, K = 10)$delta)
set.seed(123)
(cv.err.nmodv110 <- cv.glm(Gdata, nmodv1, K = 10)$delta)

#The Negative Binomial nmod2 formula wins! Now we will look at the
#interpretation of the model.

summary(nmodv1)
(1-exp((nmodv1$dev-nmodv1$null)/nrow(Gdata)))/(1-exp(-nmodv1$null/nrow(Gdata)))
#The negative binomial model only explains about 6% of the deviance.
sign<-c()
for (i in seq(1,length(coef(nmodv1)),by=1)){
  if(coef(nmodv1)[i]<0){
          sign<-c(sign,-1)
  } else {
          sign<-c(sign,1)
  }
}
nbincoef<-exp(coef(nmodv1))*sign
nbincoef

#Finally, we will see how well the model predicts on the test set compared to looking at a
#simple average.

rmse<-function(x,y) sqrt(mean((x-y)^2))
mae<-function(x,y) mean(abs(x-y))
me<- function(x,y) mean(x-y)
rmse(Test1$FantasyPoints,exp(predict(nmodv1,Test1)))
rmse(Test1$FantasyPoints,Test1$AverageFantasyPoints)
mae(Test1$FantasyPoints,exp(predict(nmodv1,Test1)))
mae(Test1$FantasyPoints,Test1$AverageFantasyPoints)
me(Test1$FantasyPoints,exp(predict(nmodv1,Test1)))
me(Test1$FantasyPoints,Test1$AverageFantasyPoints)
```

Non Parametric (MARS, Local Polynomial, Additive, KNN) R Code:

```
#Load in data
#Clean the environment
rm(list=ls())
cat("\014")
library(faraway)

setwd("C:/Users/camer/Desktop/Baseball/Dataset Building Scripts")
batters=read.csv("BatterDataset2.csv",header=T)

#Create additional variables
batters$Substitute<-NULL
batters$StartLineupPos<-as.factor(batters$StartLineupPos)
batters$Scored<-ifelse(batters$FantasyPoints <= 1, 0, 1)
batters$PositionNew<-batters$Position
batters$WeightedwOBA<-((.69*batters$WeightedBB+.72*batters$WeightedHBP+.89*batters$W
eightedS+
                  1.27*batters$Weighted2B+1.62*batters$Weighted3B+
                  2.1*batters$WeightedHR)/(batters$WeightedAB+batters$WeightedBB+
                                             batters$WeightedSF+batters$WeightedHBP)) #missing
-intentional walks in the denominator
batters$WeightedBABIP<-((batters$WeightedH-batters$WeightedHR)/(batters$WeightedAB-bat
ters$WeightedSO-batters$WeightedHR+batters$WeightedSF))
batters$WeightedISO<-batters$WeightedSLG-batters$WeightedBA
batters$AveragewOBA<-((.69*batters$AverageBB+.72*batters$AverageHBP+.89*batters$Avera
geS+
                  1.27*batters$Average2B+1.62*batters$Average3B+
                  2.1*batters$AverageHR)/(batters$AverageAB+batters$AverageBB+
                                            batters$AverageSF+batters$AverageHBP)) #missing -intentional
walks in the denominator
batters$AverageBABIP<-((batters$AverageH-batters$AverageHR)/(batters$AverageAB-batters
$AverageSO-batters$AverageHR+batters$AverageSF))
batters$AverageISO<-batters$AverageSLG-batters$AverageBA

fill<-c()
for (i in seq(1,nrow(batters),by=1)){
  fill<-c(fill,gsub("-.+","",batters$PositionNew[i]))
}

batters$PositionNew<-as.factor(fill)
```

```
#Set Seed
set.seed(1)

# Split the data into training (70%), validation (20%), and test (10%) sets
mask_train <- sample(nrow(batters), size = floor(nrow(batters) * 0.7))
Train <- batters[mask_train, ]

remaining <- batters[-mask_train, ]
mask_val <- sample(nrow(remaining), size = floor(nrow(remaining) * (2/3)))
Validation <- remaining[mask_val, ]

Test <- remaining[-mask_val, ]

# Bin the training set
train_bin0 <- Train[Train$FantasyPoints == 0, ]
train_bin2_4 <- Train[Train$FantasyPoints >= 2 & Train$FantasyPoints <= 4, ]
train_bin5_8 <- Train[Train$FantasyPoints >= 5 & Train$FantasyPoints <= 8, ]
train_bin9_17 <- Train[Train$FantasyPoints >= 9 & Train$FantasyPoints <= 17, ]
train_bin18_39 <- Train[Train$FantasyPoints >= 18 & Train$FantasyPoints <= 39, ]
train_bin40_62 <- Train[Train$FantasyPoints >= 40 & Train$FantasyPoints <= 62, ]

# Oversample the last two bins of the training set
mask_train_bin18_39 <- sample(nrow(train_bin18_39), size = nrow(train_bin0), replace =
TRUE)
train_bin18_39_bootstrap <- train_bin18_39[mask_train_bin18_39, ]

mask_train_bin40_62 <- sample(nrow(train_bin40_62), size = nrow(train_bin0), replace =
TRUE)
train_bin40_62_bootstrap <- train_bin40_62[mask_train_bin40_62, ]

# Finalize the training set
TrainBoosted <- rbind(train_bin0, train_bin2_4, train_bin5_8, train_bin9_17,
train_bin18_39_bootstrap, train_bin40_62_bootstrap)

#Plot Geomoetric Points
library(ggplot2)
setwd("C:/Users/camer/Desktop/Baseball/Dataset Building Scripts")
geo<-read.csv("GeoData.csv",header=T)
ggplot(geo, aes(Fantasy.Points)) +
  geom_line(aes(y = Probability, colour = "Theoretical")) +
  geom_line(aes(y = Actual.Data, colour = "Actual")) +
  ggtitle("Fantasy Points Follow a Geometric Distribution")
```

```
#The proportion of zeros in our data are more than what a theoretical distribution that aligns with
our
#empiracle distribution would allow for.To account for this we will first use a logistic regression
#model to predict which batters will most likely score. After choosing a threshold, we will then
predict
#fantasy points using a negative binomial distribution for those we have identified as
#being most likely to score.

#MARS
#install.packages('mda')
#library(mda)
#datamars = dplyr::select(Train, FantasyPoints, WeightedFantasyPoints,
PitcherWeightedFPAllowed,
#                   ParkFactor)

#a<-mars(datamars[,-1],datamars[,1])
#summary(lm(datamars[,1]~a$x-1))

#Consider parameters nk and degree

#Indicator Matrix
#a$factor[a$selected.terms,]

#plot(datamars[,4],a$x[,7]*a$coef[7]+a$x[,6]*a$coef[6],xlab="Park Factor",ylab="Contribution of
Park Factor")
#plot(datamars[,3],a$x[,5]*a$coef[5]+a$x[,4]*a$coef[4],xlab="PitcherWeightedFPAllowed",ylab="
Contribution of PitcherWeightedFPAllowed")
#plot(datamars[,2],a$x[,3]*a$coef[3]+a$x[,2]*a$coef[2],xlab="WeightedFPPoints",ylab="Contribu
tion of WeightedFantasyPoints")

#qqnorm (a$res, main="")
#plot (a$fitted.values, a$res, xlab="Fitted Values", ylab="Residuals")

#MARS Part2
#install.packages('earth')
#install.packages('caret')
#install.packages('vip')
#install.packages('pdp')
library(earth)
library(caret)
library(vip)
```

```
library(pdp)

dataearth = dplyr::select(Train, FantasyPoints, WeightedFantasyPoints, WeightedS,
Weighted2B,
                  Weighted3B, WeightedHR, WeightedHR, WeightedR, WeightedRBI,
WeightedBB, WeightedHBP,
                  WeightedSB, WeightedH, WeightedSO, WeightedPA, WeightedSH, WeightedSF,
WeightedCS,
                  WeightedAB, WeightedWPA, WeightedRE24, WeightedaLI, WeightedPit,
WeightedStr, WeightedBA,
                  WeightedOBP, WeightedSLG, WeightedOPS, WeightedStr_Pit,
WeightedSBSuccess,PitcherWeightedFPAllowed,
                  ParkFactor, Temperature, WeightedISO, WeightedBABIP, WeightedwOBA)

#summary(dataearth)
cor(dataearth)[1,]

mars1<-
earth(FantasyPoints~WeightedFantasyPoints+WeightedR+WeightedOPS+WeightedSLG+

WeightedwOBA+WeightedRBI+WeightedPA+WeightedHR+WeightedRE24+WeightedPit+Weig
htedH+WeightedISO+WeightedStr+

WeightedAB+WeightedWPA+WeightedOBP+Weighted2B+WeightedBB+WeightedBA+Weighte
dSB+

PitcherWeightedFPAllowed+WeightedS+ParkFactor+WeightedSBSuccess+WeightedSF+Weigh
tedBABIP+

WeightedSO+Temperature+WeightedaLI+Weighted3B+WeightedCS+WeightedHBP+Weighted
SH+

WeightedStr_Pit+Team+OpponentTeam+StartLineupPos+UmpireHP+BattingHand+Venue+
                  DayPart+HomeAway,data=Train, degree = 2)
print(mars1)
summary(mars1)
plot(mars1,which=1)
predictionmarsval<-predict(mars1,Validation)
predictionmarstest<-predict(mars1,Test)
plot(residuals(mars1))
hist(residuals(mars1))
qqnorm(residuals(mars1))
```

```
#Local Polynomial
# Are my X variables bound within a range?

newdata = dplyr::select(Validation, FantasyPoints, WeightedFantasyPoints, ParkFactor,
                        PitcherWeightedFPAllowed)
newdata2 = dplyr::select(Test, FantasyPoints, WeightedFantasyPoints, ParkFactor,
                        PitcherWeightedFPAllowed)
rmse<-function(x,y) sqrt(mean((x-y)^2))
mae<-function(x,y) mean(abs(x-y))
me<- function(x,y) mean(x-y)

#best_s_root<-2
#best_s_mae<-2
#best_s_me<-2
#root<-100000
#meanabserror<-100000
#meanerror<-100000

# for (s in seq(.7,.95,by=.05)){
#
f<-loess(FantasyPoints~WeightedFantasyPoints+ParkFactor+PitcherWeightedFPAllowed,Train,
span = s)
#   print(s)
#   prediction<-as.vector(predict(f,newdata))
#   c<-c()
#   for (i in seq(1,length(prediction),by=1)){
#         if(is.na(prediction[i])){
#         c<-c(c,i)
#         }
#   }
#   valcopy<-Validation[-c,]
#   predcopy<-prediction[-c]
#   rootmod<-c(root,rmse(valcopy$FantasyPoints,predcopy))
#   meanabserrormod<-c(meanabserror,mae(valcopy$FantasyPoints,predcopy))
#   meanerrormod<-c(meanerror,me(valcopy$FantasyPoints,predcopy))
#   if (rmse(valcopy$FantasyPoints,predcopy)<root){
#         best_s_root<-s
#         root<-rmse(valcopy$FantasyPoints,predcopy)
#   }
#   if (mae(valcopy$FantasyPoints,predcopy)<meanabserror){
#         best_s_mae<-s
#         meanabserror<-mae(valcopy$FantasyPoints,predcopy)
#   }
```

```
#   if (me(valcopy$FantasyPoints,predcopy)<meanerror){
#         best_s_me<-s
#         meanerror<-me(valcopy$FantasyPoints,predcopy)
#   }
# }

f<-loess(FantasyPoints~WeightedFantasyPoints+ParkFactor+PitcherWeightedFPAllowed,Train,
span = .75)
predictionloessval<-as.vector(predict(f,newdata))
predictionloesstest<-as.vector(predict(f,newdata2))
predictionloesstest[is.na(predictionloesstest)] <- 7.154832
# c<-c()
# for (i in seq(1,length(prediction),by=1)){
#   if(is.na(prediction[i])){
#         c<-c(c,i)
#   }
# }
# valcopy<-Validation[-c,]
# predcopy<-prediction[-c,]

summary(f)
plot(residuals(f))
hist(residuals(f))
qqnorm(residuals(f))

#Additive Model
#install.packages('gam')
#library(gam)
library(mgcv)

#amgamr<- gam(FantasyPoints ~
lo(WeightedFantasyPoints,span=.75,degree=2)+lo(ParkFactor,span=.75,degree=2)+
#
lo(PitcherWeightedFPAllowed,span=.75,degree=2)+Team+OpponentTeam+HomeAway+DayPa
rt+
#                   BattingHand+StartLineupPos+OpponentSPHand+UmpireHP,data = Train)

# amgamr<-gam(FantasyPoints ~
s(WeightedFantasyPoints)+s(WeightedR)+s(WeightedOPS)+s(WeightedSLG)+
#
s(WeightedwOBA)+s(WeightedRBI)+s(WeightedPA)+s(WeightedHR)+s(WeightedRE24)+s(Wei
ghtedPit)+
```

```
#
s(WeightedH)+s(WeightedISO)+s(WeightedStr)+s(WeightedAB)+s(WeightedWPA)+s(Weighted
OBP)+
#
s(Weighted2B)+s(WeightedBB)+s(WeightedBA)+s(WeightedSB)+s(PitcherWeightedFPAllowed)
+
#
s(WeightedS)+s(ParkFactor)+s(WeightedSBSuccess)+s(WeightedSF)+s(WeightedBABIP)+
#
s(WeightedSO)+s(Temperature)+s(WeightedaLI)+s(Weighted3B)+s(WeightedCS)+s(Weighted
HBP)+
#
s(WeightedSH)+s(WeightedStr_Pit)+Team+OpponentTeam+HomeAway+DayPart+
#                   BattingHand+StartLineupPos+OpponentSPHand+UmpireHP,data=Train)

amgamr<- gam(FantasyPoints ~ s(WeightedFantasyPoints)+s(ParkFactor)+
        s(PitcherWeightedFPAllowed)+Team+OpponentTeam+HomeAway+DayPart+
        BattingHand+StartLineupPos+OpponentSPHand+UmpireHP,data = Train)

summary(amgamr)

#amgamr2<- gam(FantasyPoints ~
lo(WeightedFantasyPoints,span=.75,degree=2)+lo(ParkFactor,span=.75,degree=2)+
#                   lo(PitcherWeightedFPAllowed,span=.75,degree=2)+Team+OpponentTeam+
#                   StartLineupPos+OpponentSPHand+UmpireHP,data = Train)

# amgamr2<-gam(FantasyPoints ~ s(WeightedFantasyPoints)+
#                   +s(WeightedRE24)+
#                   s(WeightedISO)+s(WeightedSB)+s(PitcherWeightedFPAllowed)+
#                   s(ParkFactor)+s(Temperature)+
#                   s(WeightedSH)+Team+OpponentTeam+HomeAway+DayPart+
#                   BattingHand+StartLineupPos+OpponentSPHand+UmpireHP,data=Train)

amgamr2<- gam(FantasyPoints ~ s(WeightedFantasyPoints)+s(ParkFactor)+
        s(PitcherWeightedFPAllowed)+Team+OpponentTeam+
        StartLineupPos+OpponentSPHand+UmpireHP,data = Train)

summary(amgamr2)
anova(amgamr,amgamr2,test='F')
plot(amgamr2)
plot(residuals(amgamr2))
hist(residuals(amgamr2))
qqnorm(residuals(amgamr2))
```

```
# newdata3 = dplyr::select(Validation, FantasyPoints, WeightedFantasyPoints, WeightedRE24,
WeightedISO,
#                         WeightedSB, ParkFactor,
#                         PitcherWeightedFPAllowed, Temperature, WeightedSH, Team,
OpponentTeam, HomeAway, DayPart,
#                         BattingHand,
#                         StartLineupPos,OpponentSPHand,UmpireHP)
#
# newdata4 = dplyr::select(Test, FantasyPoints, WeightedFantasyPoints, WeightedRE24,
WeightedISO,
#                         WeightedSB, ParkFactor,
#                         PitcherWeightedFPAllowed, Temperature, WeightedSH, Team,
OpponentTeam, HomeAway, DayPart,
#                         BattingHand,
#                         StartLineupPos,OpponentSPHand,UmpireHP)

newdata3 = dplyr::select(Validation, FantasyPoints, WeightedFantasyPoints, ParkFactor,
                  PitcherWeightedFPAllowed, Team, OpponentTeam,
                  StartLineupPos,OpponentSPHand,UmpireHP)

newdata4 = dplyr::select(Test, FantasyPoints, WeightedFantasyPoints, ParkFactor,
                  PitcherWeightedFPAllowed, Team, OpponentTeam,
                  StartLineupPos,OpponentSPHand,UmpireHP)

predictionadditiveval<-as.vector(predict(amgamr2,newdata3))
predictionadditivetest<-as.vector(predict(amgamr2,newdata4))

# g<-c()
# for (i in seq(1,length(prediction),by=1)){
#   if(is.na(prediction[i])){
#         g<-c(g,i)
#   }
# }
#
# valcopy<-Validation[-g,]
# predcopy<-prediction[-g,]

#Alternating Conditional Expectations
#library(acepack)
#data = dplyr::select(Train, WeightedFantasyPoints, ParkFactor,
#                  PitcherWeightedFPAllowed)
#acefit<-ace(data,Train$FantasyPoints)
```

```
#summary(lm(acefit$ty ~ acefit$tx))
#plot (Train$FantasyPoints,acefit$ty,xlab="FantasyPoints",
ylab=expression(theta(Train$FantasyPoints)))
#plot(data[,1],acefit$tx[,1],xlab="WeightedFantasyPoints",ylab="f(WeightedFantasyPoints)")
#plot(data[,2],acefit$tx[,2],xlab="ParkFactor",ylab="f(ParkFactor)")
#plot(data[,3],acefit$tx[,3],xlab="PitcherWeightedFPAllowed",ylab="f(PitcherWeightedFPAllowed
)")

#predictace<-predict(acefit,Validation)

#Nearest Neighbors
library('kknn')

# bestnn_RMSE<-NULL
# bestnn_MAE<-NULL
# best_RMSE<-10000000
# best_MAE<-10000000

# for (j in seq(19,81,by=2)){
#   fittedv<-c()
#   actual<-c()
#   for (k in seq(1,400,by=1)){
#         n<-sample(1:nrow(Train),1)
#         Traincopy<-Train[c(-n),]
#         x<-kknn(FantasyPoints~WeightedFantasyPoints+WeightedOPS+WeightedPA+
#                   PitcherWeightedFPAllowed+ParkFactor+WeightedBABIP+WeightedISO+
#                   WeightedwOBA,Traincopy,Train[c(n),],k=j,kernel='rectangular',scale=TRUE)
#         fittedv<-c(fittedv,x$fitted.values)
#         actual<-c(actual,Train[c(n),1])
#   }
#   if(rmse(actual,fittedv)<best_RMSE){
#         bestnn_RMSE<-j
#         best_RMSE<-rmse(actual,fittedv)
#   }
#   if(mae(actual,fittedv)<best_MAE){
#         bestnn_MAE<-j
#         best_MAE<-mae(actual,fittedv)
#   }
# }

bestnn_MAE<-77 #75,37,71,73,77
bestnn_RMSE<-77 #75,37,71,73,77
```

```
#75 RMSE 7.047056 MAE 5.421854 -Best RMSE Best MAE
#37 RMSE 7.087281 MAE 5.451796
#71 RMSE 7.087281 MAE 5.451796
#73 RMSE 7.047249 MAE 5.424046
#77 RMSE 7.047393 MAE 5.42275

predictionknnval<-c()
actualval<-c()
for (k in seq(1,nrow(Validation),by=1)){
  Valcopy<-Validation[c(-k),]
  x<-kknn(FantasyPoints~WeightedFantasyPoints+WeightedOPS+WeightedPA+
          PitcherWeightedFPAllowed+ParkFactor+WeightedBABIP+WeightedISO+

WeightedwOBA,Valcopy,Validation[c(k),],k=bestnn_MAE,kernel='rectangular',scale=TRUE)
  predictionknnval<-c(predictionknnval,x$fitted.values)
  actual<-c(actualval,Validation[c(k),1])
}

predictionknntest<-c()
actualtest<-c()
for (k in seq(1,nrow(Test),by=1)){
  Testcopy<-Test[c(-k),]
  x<-kknn(FantasyPoints~WeightedFantasyPoints+WeightedOPS+WeightedPA+
          PitcherWeightedFPAllowed+ParkFactor+WeightedBABIP+WeightedISO+
          WeightedwOBA,Testcopy,Test[c(k),],k=bestnn_RMSE,kernel='rectangular',scale=TRUE)
  predictionknntest<-c(predictionknntest,x$fitted.values)
  actualtest<-c(actualtest,Test[c(k),1])
}

#RMSE Validation
rmse(Validation$FantasyPoints,predictionmarsval)
rmse(Validation$FantasyPoints,predictionloessval)
rmse(Validation$FantasyPoints,predictionadditiveval)
rmse(Validation$FantasyPoints,predictionknnval)
rmse(Validation$FantasyPoints,Validation$WeightedFantasyPoints)
rmse(Validation$FantasyPoints,Validation$AverageFantasyPoints)

#MAE Validation
mae(Validation$FantasyPoints,predictionmarsval)
mae(Validation$FantasyPoints,predictionloessval)
mae(Validation$FantasyPoints,predictionadditiveval)
mae(Validation$FantasyPoints,predictionknnval)
mae(Validation$FantasyPoints,Validation$WeightedFantasyPoints)
```

```
mae(Validation$FantasyPoints,Validation$AverageFantasyPoints)

#RMSE Test
rmse(Test$FantasyPoints,predictionmarstest)
rmse(Test$FantasyPoints,predictionloesstest)
rmse(Test$FantasyPoints,predictionadditivetest)
rmse(Test$FantasyPoints,predictionknntest)
rmse(Test$FantasyPoints,Test$WeightedFantasyPoints)
rmse(Test$FantasyPoints,Test$AverageFantasyPoints)

#MAE Test
mae(Test$FantasyPoints,predictionmarstest)
mae(Test$FantasyPoints,predictionloesstest)
mae(Test$FantasyPoints,predictionadditivetest)
mae(Test$FantasyPoints,predictionknntest)
mae(Test$FantasyPoints,Test$WeightedFantasyPoints)
mae(Test$FantasyPoints,Test$AverageFantasyPoints)

Mars_Validation_Prediction<-write.csv(predictionmarsval, file =
"Mars_Validation_Prediction.csv")
LocalPolynomial_Validation_Prediction<-write.csv(predictionloessval, file =
"LocalPolynomial_Validation_Prediction.csv" )
AdditiveModel_Validation_Prediction<-write.csv(predictionadditiveval, file =
"AdditiveModel_Validation_Prediction.csv")
KNN_Validation_Prediction<-write.csv(predictionknnval, file = "KNN_Validation_Prediction.csv")
WeightedFantasyPoints_Validation<-write.csv(Validation$WeightedFantasyPoints,file =
"WeightedFantasyPoints_Validation.csv")
AverageFantasyPoints_Validation<-write.csv(Validation$AverageFantasyPoints,file
="AverageFantasyPoints_Validation.csv")
Mars_Test_Prediction<-write.csv(predictionmarstest,file = "Mars_Test_Prediction.csv")
LocalPolynomial_Test_Prediction<-write.csv(predictionloesstest,"LocalPolynomial_Test_Predicti
on.csv")
AdditiveModel_Test_Prediction<-write.csv(predictionadditivetest,"AdditiveModel_Test_Predictio
n.csv")
KNN_Test_Prediction<-write.csv(predictionknntest,"KNN_Test_Prediction.csv")
WeightedFantasyPoints_Test<-write.csv(Test$WeightedFantasyPoints,"WeightedFantasyPoints
_Test.csv")
AverageFantasyPoints_Test<-write.csv(Test$AverageFantasyPoints,"AverageFantasyPoints_T
est.csv")
```

Decision Tree Regression Code in R:

```
# Clear the environment
rm(list = ls())

# Set the random number generator seed so results are reproducible
set.seed(1)

# Read in the data
batters <- read.csv("BatterDataset2.csv",header=T)

# Check to ensure the data is read in correctly
head(batters)

# Add the new variables
batters$Substitute <- NULL
batters$StartLineupPos <- as.factor(batters$StartLineupPos)
batters$WeightedwOBA <-
((.69*batters$WeightedBB+.72*batters$WeightedHBP+.89*batters$WeightedS+
                    1.27*batters$Weighted2B+1.62*batters$Weighted3B+
                    2.1*batters$WeightedHR)/(batters$WeightedAB+batters$WeightedBB+
                                    batters$WeightedSF+batters$WeightedHBP)) #missing
-intentional walks in the denominator
batters$WeightedBABIP <-
((batters$WeightedH-batters$WeightedHR)/(batters$WeightedAB-batters$WeightedSO-batters$
WeightedHR+batters$WeightedSF))
batters$WeightedISO <- batters$WeightedSLG-batters$WeightedBA
batters$AveragewOBA <-
((.69*batters$AverageBB+.72*batters$AverageHBP+.89*batters$AverageS+
                    1.27*batters$Average2B+1.62*batters$Average3B+
                    2.1*batters$AverageHR)/(batters$AverageAB+batters$AverageBB+
                                    batters$AverageSF+batters$AverageHBP)) #missing
-intentional walks in the denominator
batters$AverageBABIP <-
((batters$AverageH-batters$AverageHR)/(batters$AverageAB-batters$AverageSO-batters$Aver
ageHR+batters$AverageSF))
batters$AverageISO <- batters$AverageSLG-batters$AverageBA

# Split the data into training (70%), validation (20%), and test (10%) sets
mask_train <- sample(nrow(batters), size = floor(nrow(batters) * 0.7))
batters_train <- batters[mask_train, ]

remaining <- batters[-mask_train, ]
```

```
mask_val <- sample(nrow(remaining), size = floor(nrow(remaining) * (2/3)))
batters_val <- remaining[mask_val, ]

batters_test <- remaining[-mask_val, ]

# Bin the training set
train_bin0 <- batters_train[batters_train$FantasyPoints == 0, ]
train_bin2_4 <- batters_train[batters_train$FantasyPoints >= 2 & batters_train$FantasyPoints
<= 4, ]
train_bin5_8 <- batters_train[batters_train$FantasyPoints >= 5 & batters_train$FantasyPoints
<= 8, ]
train_bin9_17 <- batters_train[batters_train$FantasyPoints >= 9 & batters_train$FantasyPoints
<= 17, ]
train_bin18_39 <- batters_train[batters_train$FantasyPoints >= 18 &
batters_train$FantasyPoints <= 39, ]
train_bin40_62 <- batters_train[batters_train$FantasyPoints >= 40 &
batters_train$FantasyPoints <= 62, ]

# Oversample the last two bins of the training set
mask_train_bin18_39 <- sample(nrow(train_bin18_39), size = nrow(train_bin0), replace =
TRUE)
train_bin18_39_bootstrap <- train_bin18_39[mask_train_bin18_39, ]

mask_train_bin40_62 <- sample(nrow(train_bin40_62), size = nrow(train_bin0), replace =
TRUE)
train_bin40_62_bootstrap <- train_bin40_62[mask_train_bin40_62, ]

# Finalize the training set
batters_train2 <- rbind(train_bin0, train_bin2_4, train_bin5_8, train_bin9_17,
train_bin18_39_bootstrap, train_bin40_62_bootstrap)

# Bin the validation set
val_bin0 <- batters_val[batters_val$FantasyPoints == 0, ]
val_bin2_4 <- batters_val[batters_val$FantasyPoints >= 2 & batters_val$FantasyPoints <= 4, ]
val_bin5_8 <- batters_val[batters_val$FantasyPoints >= 5 & batters_val$FantasyPoints <= 8, ]
val_bin9_17 <- batters_val[batters_val$FantasyPoints >= 9 & batters_val$FantasyPoints <= 17,
]
val_bin18_39 <- batters_val[batters_val$FantasyPoints >= 18 & batters_val$FantasyPoints <=
39, ]
val_bin40_62 <- batters_val[batters_val$FantasyPoints >= 40 & batters_val$FantasyPoints <=
62, ]

# Oversample the validation set
```

```
mask_val_bin18_39 <- sample(nrow(val_bin18_39), size = nrow(val_bin0), replace = TRUE)
val_bin18_39_bootstrap <- val_bin18_39[mask_val_bin18_39, ]

mask_val_bin40_62 <- sample(nrow(val_bin40_62), size = nrow(val_bin0), replace = TRUE)
val_bin40_62_bootstrap <- val_bin40_62[mask_val_bin40_62, ]

# Finalize the validation set
batters_val2 <- rbind(val_bin0, val_bin2_4, val_bin5_8, val_bin9_17, val_bin18_39_bootstrap,
val_bin40_62_bootstrap)

#### decision tree regression
library(C50)
library(rpart)
library(randomForest)
batters_train2[batters_train2==""]  <- 0
batters_train2[is.na(batters_train2)] <- 0
print(batters_train2$WindDirection)

#train_y <- as.factor(batters_train2[1])
#train_y <- str(train_y)
#train_x <- batters_train2[2:141]
#levels(train_x$WindDirection)[1] = 'missing'
#levels(train_x$Precipitation)[1] = 'missing'
#levels(train_x$Sky)[1] = 'missing'
#decision_model <- C5.0(x = train_x, y = train_y)
#summary(decision_model)
#fit <- rpart(FantasyPoints ~., data = batters_train2)
fit <-rpart(FantasyPoints ~ ., data = batters_train2, minsplit=3000, cp=0.001)
#par(mfrow=c(1,2))
rsq.rpart(fit)
pred_y <- predict(fit, batters_val2)
mse <- mean((batters_val2$FantasyPoints - pred_y)^2)
print(mse)
plot(fit, uniform=TRUE,main="Decision Tree for FantasyPoints")
text(fit, use.n=TRUE, all=TRUE, cex=.5)
pfit<- prune(fit, cp = fit$cptable[which.min(fit$cptable[,"xerror"]),"CP"])
plot(pfit, uniform=TRUE, main="Decision Tree for FantasyPoints")
text(pfit, use.n=TRUE, all=TRUE, cex=.5
```

Nonlinear SVR code in R:

```
setwd("C:/Users/Benguin/Documents/Grad School/Spring 2019/Data Mining and Stat Learning")
# Clear the environment
rm(list = ls())

# Set the random number generator seed so results are reproducible
set.seed(1)

# Read in the data
batters <- read.csv("BatterDataset2.csv",header=T)

# Check to ensure the data is read in correctly
head(batters)

# Add the new variables
batters$Substitute <- NULL
batters$StartLineupPos <- as.factor(batters$StartLineupPos)
batters$WeightedwOBA <-
((.69*batters$WeightedBB+.72*batters$WeightedHBP+.89*batters$WeightedS+
                    1.27*batters$Weighted2B+1.62*batters$Weighted3B+
                    2.1*batters$WeightedHR)/(batters$WeightedAB+batters$WeightedBB+
                                               batters$WeightedSF+batters$WeightedHBP)) #missing
-intentional walks in the denominator
batters$WeightedBABIP <-
((batters$WeightedH-batters$WeightedHR)/(batters$WeightedAB-batters$WeightedSO-batters$
WeightedHR+batters$WeightedSF))
batters$WeightedISO <- batters$WeightedSLG-batters$WeightedBA
batters$AveragewOBA <-
((.69*batters$AverageBB+.72*batters$AverageHBP+.89*batters$AverageS+
                    1.27*batters$Average2B+1.62*batters$Average3B+
                    2.1*batters$AverageHR)/(batters$AverageAB+batters$AverageBB+
                                              batters$AverageSF+batters$AverageHBP)) #missing -intentional
walks in the denominator
batters$AverageBABIP <-
((batters$AverageH-batters$AverageHR)/(batters$AverageAB-batters$AverageSO-batters$Aver
ageHR+batters$AverageSF))
batters$AverageISO <- batters$AverageSLG-batters$AverageBA

fill<-c()
for (i in seq(1,nrow(batters),by=1)){
 fill<-c(fill,gsub("-.+","",batters$PositionNew[i]))
}
```

```
batters$PositionNew<-as.factor(fill)

# Split the data into training (70%), validation (20%), and test (10%) sets
mask_train <- sample(nrow(batters), size = floor(nrow(batters) * 0.7))
batters_train <- batters[mask_train, ]

remaining <- batters[-mask_train, ]
mask_val <- sample(nrow(remaining), size = floor(nrow(remaining) * (2/3)))
batters_val <- remaining[mask_val, ]

batters_test <- remaining[-mask_val, ]

# Bin the training set
train_bin0 <- batters_train[batters_train$FantasyPoints == 0, ]
train_bin2_4 <- batters_train[batters_train$FantasyPoints >= 2 & batters_train$FantasyPoints
<= 4, ]
train_bin5_8 <- batters_train[batters_train$FantasyPoints >= 5 & batters_train$FantasyPoints
<= 8, ]
train_bin9_17 <- batters_train[batters_train$FantasyPoints >= 9 & batters_train$FantasyPoints
<= 17, ]
train_bin18_39 <- batters_train[batters_train$FantasyPoints >= 18 &
batters_train$FantasyPoints <= 39, ]
train_bin40_62 <- batters_train[batters_train$FantasyPoints >= 40 &
batters_train$FantasyPoints <= 62, ]

# Oversample the last two bins of the training set
mask_train_bin18_39 <- sample(nrow(train_bin18_39), size = nrow(train_bin0), replace =
TRUE)
train_bin18_39_bootstrap <- train_bin18_39[mask_train_bin18_39, ]

mask_train_bin40_62 <- sample(nrow(train_bin40_62), size = nrow(train_bin0), replace =
TRUE)
train_bin40_62_bootstrap <- train_bin40_62[mask_train_bin40_62, ]

# Finalize the training set
batters_train2 <- rbind(train_bin0, train_bin2_4, train_bin5_8, train_bin9_17,
train_bin18_39_bootstrap, train_bin40_62_bootstrap)

library(e1071)

batters_train2 <- impute(obj = batters_train2, target =character(0),classes = list(integer =
imputeMedian(), numeric = imputeMedian(), factor = imputeMode()))$data
```

```
batters_test <- impute(obj = batters_test, target =character(0),classes = list(integer =
imputeMedian(), numeric = imputeMedian(), factor = imputeMode()))$data
batters_val <- impute(obj = batters_val, target =character(0),classes = list(integer =
imputeMedian(), numeric = imputeMedian(), factor = imputeMode()))$data

D <- batters_train2[sample(nrow(batters_train2), 5000, replace = FALSE, prob = NULL),]

train_x <- subset(batters_train2, select = -FantasyPoints)
test_x <- subset(batters_test, select = -FantasyPoints)
val_x <- subset(batters_val, select = -FantasyPoints)

train_y <- as.numeric(batters_train2$FantasyPoints)
test_y <- as.numeric(batters_test$FantasyPoints)
val_y <- as.numeric(batters_val$FantasyPoints)

model <- svm(FantasyPoints ~ ., data = D)

test_pred <- predict(model, test_x)
val_pred <- predict(model, val_x)

print("Test MAE:")
mean(abs(test_y - test_pred))
print("Validation MAE:")
mean(abs(val_y - val_pred))
print("Test RMSE:")
sqrt(mean((test_y - test_pred)^2))
print("Validation RSME:")
sqrt(mean((val_y - val_pred)^2))

write.csv(test_pred, file = "SVR_Test_Prediction.csv")
write.csv(val_pred, file = "SVR_Validation_Prediction.csv")
```

Neural Network Code in Python:

```
import pandas as pd
import random as rd

trainset=pd.read_csv('C:/Users/Changeme/Desktop/Howard/FinalProject/BattersTrainr1.csv')
testset=pd.read_csv('C:/Users/Changeme/Desktop/Howard/FinalProject/BattersTestr1.csv')
trainset.describe()
testset.describe()
```

```
train_data = trainset.iloc[:, 2:].values
train_targets = trainset.iloc[:, 1].values
test_data = testset.iloc[:, 2:].values
test_targets = testset.iloc[:, 1].values

# Normalize the data
mean = train_data.mean(axis=0)
train_data -= mean
std = train_data.std(axis=0)
train_data /= std
test_data -= mean
test_data /= std

print(train_targets)

from keras import models
from keras import layers

def build_model():
    model = models.Sequential()
    model.add(layers.Dense(64, activation='relu', input_shape=(train_data.shape[1],)))
    model.add(layers.Dense(64, activation='relu'))
    model.add(layers.Dense(1))
    model.compile(optimizer='rmsprop', loss='mse', metrics=['mae','mse'])
    return model

import numpy as np

k=4
num_val_samples = len(train_data) // k
num_epochs = 10
all_mae_histories = []
all_mse_histories = []

for i in range(k):
    print('processing fold #', i)
    val_data = train_data[i * num_val_samples: (i + 1) * num_val_samples]
    val_targets = train_targets[i * num_val_samples: (i + 1) * num_val_samples]

    partial_train_data = np.concatenate( [train_data[:i * num_val_samples], train_data[(i + 1) *
num_val_samples:]], axis=0)
    partial_train_targets = np.concatenate( [train_targets[:i * num_val_samples], train_targets[(i +
1) * num_val_samples:]], axis=0)
```

```
    model = build_model()
    history = model.fit(partial_train_data, partial_train_targets, validation_data=(val_data,
val_targets), epochs=num_epochs, batch_size=1, verbose=0)
    mae_history = history.history['val_mean_absolute_error']
    mse_history = history.history['val_mean_squared_error']
    all_mae_histories.append(mae_history)
    all_mse_histories.append(mse_history)

average_mae_history = [ np.mean([x[i] for x in all_mae_histories]) for i in range(num_epochs)]
average_mse_history = [ np.mean([x[i] for x in all_mse_histories]) for i in range(num_epochs)]
import matplotlib.pyplot as plt
plt.plot(range(1, len(average_mae_history) + 1), average_mae_history)
plt.xlabel('Epochs')
plt.ylabel('Validation MAE')
plt.show()

plt.plot(range(1, len(average_mse_history) + 1), average_mse_history)
plt.xlabel('Epochs')
plt.ylabel('Validation MSE')
plt.show()

# Build real model
model = build_model()
history = model.fit(train_data, train_targets,
          epochs=3, batch_size=1, verbose=0)

test_predicts=model.predict(test_data)

# Output test MAE
test_mae_score = model.evaluate(test_data, test_targets)
print(test_mae_score)

# Output test RMSE
from sklearn.metrics import mean_squared_error
from math import sqrt
rms = sqrt(mean_squared_error(test_targets, test_predicts))
print(rms)

# Export Prediction on test data
import numpy as np
np.savetxt("test_predicts.csv", test_predicts, delimiter=",")
```

```
# Import validation data and Test on validation set
valset=pd.read_csv('/Users/howard/Documents/GT 2017 Fall/ISYE7406-Data Mining and Stat
Learning/FinalProject/batters_val.csv')
val_data = valset.iloc[:, 2:].values
val_targets = valset.iloc[:, 1].values
val_data -= mean
val_data /= std
val_predicts=model.predict(val_data)
print(val_predicts)
import numpy as np
np.savetxt("NeuralNet_Val_Prediction.csv", val_predicts, delimiter=",")
rms = sqrt(mean_squared_error(val_targets, val_predicts))
print(rms)
val_mae_score = model.evaluate(val_data, val_targets)
print(val_mae_score)
```

Ensemble Model:

```
# ------------------------------------------------------------
# Load the datasets
amt = read.csv("Ensemble/AddTest.csv",header=T)
amv = read.csv("Ensemble/AddVal.csv",header=T)

aft = read.csv("Ensemble/AFPTest.csv",header=T)
afv = read.csv("Ensemble/AFPVal.csv",header=T)

dtt = read.csv("Ensemble/DTTest.csv",header=T)
dtv = read.csv("Ensemble/DTVal.csv",header=T)

knt = read.csv("Ensemble/KNNTest.csv",header=T)
knv = read.csv("Ensemble/KNNVal.csv",header=T)

lpt = read.csv("Ensemble/LPVTest.csv",header=T)
lpv = read.csv("Ensemble/LPVVal.csv",header=T)

mst = read.csv("Ensemble/MARSTest.csv",header=T)
msv = read.csv("Ensemble/MARSVal.csv",header=T)

nnt = read.csv("Ensemble/NNTest.csv",header=F)
nnv = read.csv("Ensemble/NNVal.csv",header=F)

svt = read.csv("Ensemble/SVRTest.csv",header=T)
```

```
svv = read.csv("Ensemble/SVRVal.csv",header=T)

wft = read.csv("Ensemble/WFPTest.csv",header=T)
wfv = read.csv("Ensemble/WFPVal.csv",header=T)

v = read.csv("Val.csv",header=T)
t = read.csv("T.csv",header=T)

# ------------------------------------------------------------
# Assemble The Data
val = as.data.frame(amv)
names(val) = c("afp","add")
val$afp = afv$x
val$dt = dtv$x
val$knn = knv$x
val$lpv = lpv$x
val$mars = msv$FantasyPoints
val$wfp = wfv$x
val$svr = svv$x
val$nn = nnv$V1
val$Y = v$FantasyPoints

test = as.data.frame(amt)
names(test) = c("afp","add")
test$afp = aft$x
test$dt = dtt$x
test$knn = knt$x
test$lpv = lpt$x
test$mars = mst$FantasyPoints
test$wfp = wft$x
test$svr = svt$x
test$nn = nnt$V1
test$Y = t$FantasyPoints

# ------------------------------------------------------------
# Preview the new Datasets
head(val)
head(test)

# ------------------------------------------------------------
# Linear Regression
lin1 = lm(Y~.,data=val)
summary(lin1)
```

```
sqrt(c(crossprod(lin1$residuals))/length(lin1$residuals))
# RSE 6.896 AR2 6.212%

# check for multicollinearity. Removing did not improve the model however
vif(lin1)
cor(val)

# ------------------------------------------------------------
# Predict on Val and Test to see how well the model performed
pred1t = predict(lin1,test)
pred1v = predict(lin1,val)


# Store these values
val$PredictedV1 = pred1v
test$PredictedT1 = pred1t

# ------------------------------------------------------------
# Export to csv for error checks
write.csv(val,"valResults.csv")
write.csv(test,"testResults.csv")
```