Analyzing Player Statistics of Most Improved Players in the NBA using Clustering and Classification

Zoe Nahmiash, John McElroy, Yining Huang
zsnahmia@uwaterloo.ca, jwmcelro@uwaterloo.ca, yining.huang@uwaterloo.ca

University of Waterloo
200 University Avenue West
Waterloo, Ontario,
Canada

Presented to: Lukasz Golab
December 3, 2019
MSCI 446: Data Mining and Warehousing

ABSTRACT

The business problem addressed in this paper is determining the recipient of the NBA's Most Improved Player award, which acknowledges one of the league's best breakout stars each season. This award is difficult to predict because candidates are often not a household name similar to the likes of Lebron James or Stephen Curry. This is also a difficult award to determine due to the fact that these players have not yet hit their peak statistically speaking. It is easy to point out the best players in the league. However, attempting to determine who has the most improvement every year compared to 400 other players is a difficult task.

Clustering was leveraged as part of this analysis to get an understanding of players and their statistics. Following this, three different classification models were developed, which were trained and tested on the 13 NBA seasons between 2005 and 2018. Each of these were evaluated based on success metrics. The top two models and feature set pairs were then utilized to make predictions of the likely winners of the award for the 2018-2019 season. This information is extremely valuable to NBA front offices, as they can leverage this type of information to build stronger and more successful rosters.

1. INTRODUCTION

The purpose of this work is to identify certain characteristics and conditions that would make an NBA player likely to win the MIP award. Leveraging per game statistics and other advanced statistics, these characteristics and conditions were developed to be used as inputs for the classification model.

Our hypothesis was that the variables in feature set three (outlined is section four) would be good indicators of a player's likelihood to be voted as a potential MIP. Using the changes of per game statistics gives the model a true sense of statistical improvements. Players who improved from season to season will have a positive delta, and those who did not will have negative deltas. This is the best way of determining improvement and will truly separate those who improved the most from those who did not.

This project is interesting and non-trivial for many reasons. Firstly, it is quite difficult to predict an MIP because it is dependent on many factors not reflected in a given players' statistics.

For instance, player X's playing time could be significantly impacted by player Y's injury on the same team. This fact may lead to the potential for player X to have an unlikely rise to success as there is a new found opportunity to succeed. These conditions are quite difficult to capture in data without intensive data transformation measures, which contribute to the non-trivial nature of the problem. Additionally, the problem cannot be solved by merely looking at existing statistics, which could be the case for the MVP of the NBA. In predicting the MVP of a particular season, game statistics can be analyzed and the player with the highest ones across all categories would be a likely candidate for this award. However, with the MIP, the statistics must be looked at from a different angle. The existing player data must be manipulated in order to reflect their improvement. Additionally, not all statistics are necessarily relevant.

Our results confirm our initial hypothesis. By looking at the third feature set of a player and leveraging a logistic regression model, we were able to make more accurate predictions about the ten players who were likely to get the most votes for the MIP for the latest season in comparison to other feature sets and models.

2. RELATED WORK

A number of data mining scholarly work has been done on the topic of NBA games and players. Many existing solutions are dedicated in predicting the outcome of basketball games [9]. Naive Bayes, Random Forest, Logistic Regression, Support Vector Machines (SVM), and Neural Networks are all popular classification algorithms used to predict the outcome of matches [1]. Predicting the Most Valuable Player (MVP) is another popular topic that leverages data mining and machine learning. Being a classification problem, Random Forest is a popular choice to perform this task [6]. In addition, there are a number of more interesting research topics. For example, predicting whether a player can sustain for more than five years in the NBA using Naive Bayes, Support Vector Machine, Multi-layered Feedforward Network, and Random Forest [1]. Other interesting research is done on analyzing NBA player's mood leveraging Natural Language Processing [5].

3. DATA

3.1 Data Collection

The data collection process was very quick to complete. Given that the NBA is such a prolific entity in the world of sports, each data point that was needed was easily accessible and was in an ideal format. The data was collected entirely from basketball-reference.com. This is a source that provides NBA statistics dating back to 1946. To be able to effectively train and test the model, data was collected from 2005-2006 season up until the 2018-2019 season. Using 13 seasons worth of data gave the model enough data points to be able to efficiently learn. The NBA has been developing so rapidly in the past decade that any stats prior to this era could negatively impact the results. Therefore, we felt that limiting our dataset to seasons past 2005 was appropriate. The data included all major stat points for each player that played at least one minute in the NBA for each season. This included both per game statistics and all advanced statistics.

3.2 Data Transformation

3.2.1 Class Variable Labels

Once all the data was collected, Most Improved Player voting statistics needed to be added. Using the same source, the top 10 vote getters of each season between 2005-2019 for the Most Improved Player award was collected. This information was added to the data in the form of a boolean variable. In each season, players had a class variable value of 1 if they were in the top 10 vote getters and a 0 if they were not. This information was stored in a new additional column in the raw data.

3.2.2 Players that Changed Teams

In collecting this data, it was noticed that no players in the top 10 vote getters in the past 15 seasons had played for more than one team throughout the season. Often times players will play on two or more teams if they are traded or released and subsequently signed to a new contract through the season. Given this, each player who had played on more than one team throughout each season was removed from the dataset. This decision was made because the way these players were represented would skew the data. For example, Jimmy Butler (in the 2018-2019 season) was traded from Minnesota to Philadelphia after playing 10 games. In the data, this would be represented with one row for his statistics in Minnesota, one row for his statistics in Philadelphia and finally a third row with his total stats for the season. This meant that for every player that played on two or more teams each season (roughly forty players per season), they had three rows of data. While analyzing the dataset, it was discovered that no player in the past fifteen years had played for multiple teams and were any of the top ten candidates for Most Improved Player for that season. This is due to the fact that most of the players that play for multiple teams are either traded or released by a team and resigned by another. The majority of times a trade occurs for players that aren't performing well. This indicates that they would not be candidates for the award due to their performance. The other portion of trades are for star players, and these players are generally more likely to be considered for the Most Valuable Player award. With this in mind, each of these instances were removed from the dataset due the fact that they were not likely candidates for the award.

3.2.3 Missing Data

A common trend noticed throughout the dataset was that there were many empty cells. These empty cells always occurred in categories that described shooting percentage. The dataset included metrics for Field Goals, 3 Pointers and Free Throws. For each of these shooting categories, the data included how many shots the player took, how many shots the player made and the percentage of the shots they made. Missing data occurred in some instances where a player had 0 shots attempted and 0 shots made. In these cases the percentage of shots made would be 0%, however the data did not consistently reflect this. There were many instances where the percentage of shots made would be blank, each of these occurrences were replaced with 0's as this is what the data should reflect.

3.3 Feature Engineering

In order to effectively train and test a model, it was insufficient to rely only on statistical data in its current format within our dataset. Merely looking at these values would not yield satisfactory results. Therefore, new features were developed leveraging player statistics as a basis. These features were determined by researching related work and by leveraging our group's knowledge of basketball. Each new feature is described below:

3.3.1 Player Opportunity

$$Opportunity = \triangle USG\% / \triangle AST\% * (-1)$$

Rationale: Usage percentage (USG%) is the percentage of team possessions that ended with this player touching the ball last. This includes shots, free throws and turnovers. The higher the usage percentage, the higher the impact the player in question had on the game. Assist Percentage (AST%) looks at the amount of made shots that said player assisted their teammates on. Better players are able to create more opportunities by themselves without the help of their teammates. While players improve, their assist percentages go down, which is why this state is multiplied by negative one. The division of these two stats show the change in opportunity a player has experienced from the season previous to the current season

3.3.2 Defensive Ability

$$Defensive\ Ability = (STL + BLK)/PF$$

Rationale: These statistics are all earned on the defensive end of the floor. Steals (STL) and blocks (BLK) are positive statistics as they will always end up being beneficial to the team. Personal fouls (PF) are often seen as negative because they give the opposing team an advantage. This feature was created to look at the ratio of positive defensive impact versus negative defensive impact. MIPs will always show improvement on the defensive end of the court, which makes this feature important to track.

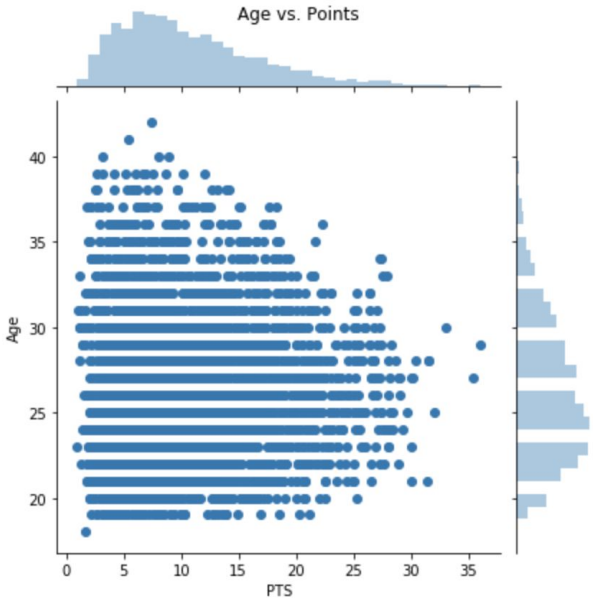
3.3.3 Total Minutes Played

$$Total\ Minutes\ Player\ =\ MPG * G$$

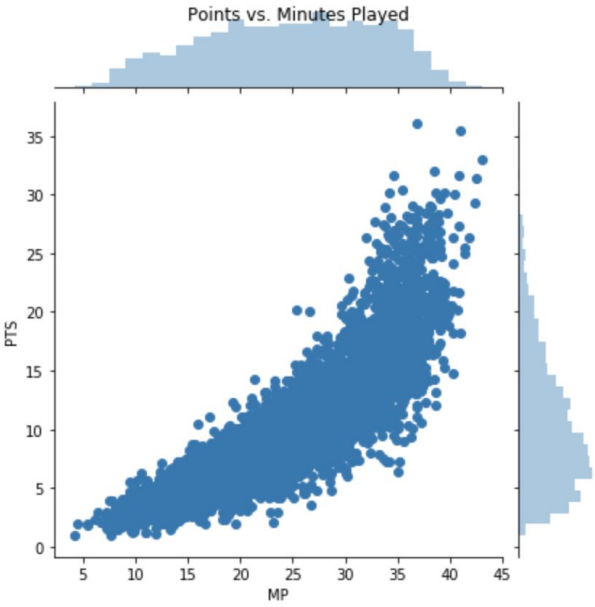
Rationale: To be able to determine player eligibility for the award, the total amount of time that they have played this season as well as last season needs to be taken into consideration. To be eligible a played needs to have played at least 250 minutes in the previous season and at least 1000 minutes in the current season. Without this feature, players can have a season where they did not play much due to injury and suddenly have great stats the next season because they are healthy, which would skew the data. This improvement is not due to them being an improved player, but rather to them being healthy again. This feature was developed to eliminate these players from being selected as a winner.

3.4 Metadata

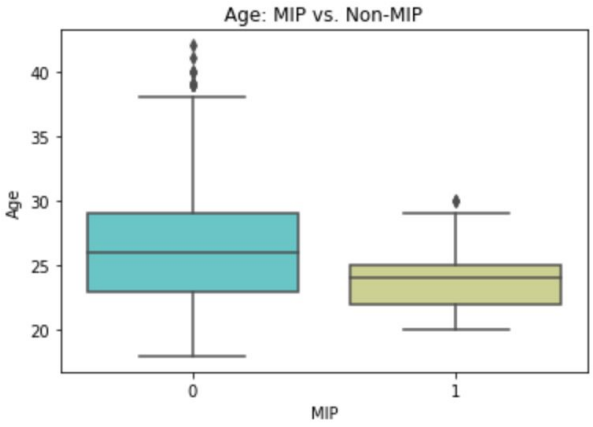
Total number of labeled observations (raw data)	5822
Total number of features available (raw data)	53
Number of eligible MIP candidates (cooked data)	4039
Number of total features available after data preparation (cooked data)	64
Number of observations labeled as MIP (cooked data)	139
Number of observations labeled as non-MIP	3900



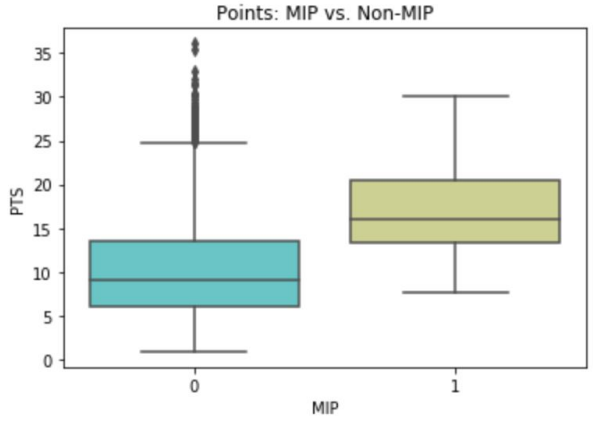
The above graph shows the relationship between points and age. It represents how players hit their scoring prime in their late 20s prior to decreasing for the final years of their careers.



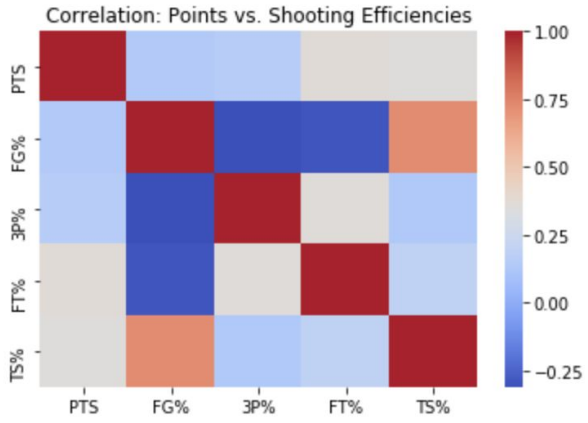
Comparing minutes played to points creates an exponentially increasing trend. Naturally, the more a player plays, the more points they score. However, it can be seen that this is not a linear relationship, which means that if a player were to double their playing minutes, they would score more than double the points.



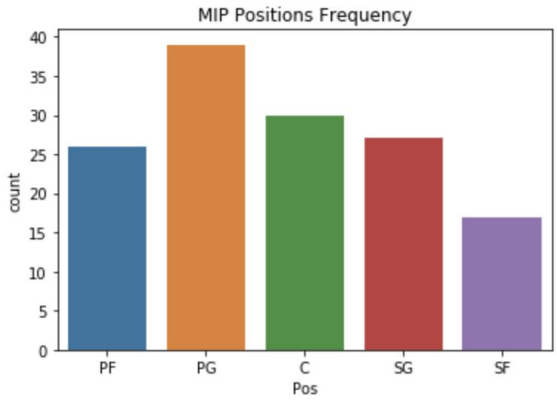
This box and whisker plot shows that MIPs are generally younger players, with the majority of them winning the award around prior to their 25th birthday. There are some outliers around 30 years old, but this is a very rare occurrence.



This plot shows that MIP's tend to score more than the average player. However, they do not score as much as the highest scoring players in the league. This proves the MIP candidates are in the upper echelon of players but have not yet achieved elite status.



In this correlation plot, it is seen that points are most strongly correlated to true shooting percentage and free throw percentage. This means that players that score more are very efficient at the free throw line. Another interesting observation is the strong correlation between 3 point percentage and free throw percentage. This proves that the best 3 point shooters are also very strong at shooting free throws. This is a logical relationship but not necessarily one that is commonly thought of.



The chart above shows the relationship between position and likelihood to be selected as a MIP candidate. It is obvious that point guards see the best results, which is likely due to the fact that they generally touch the ball the most and have the most impact on the game. Small forwards see the lowest likelihood of winning.

3.5 Final Feature Sets

Each feature set was used for each type of model. The included data points in each set are outlined below. Explanations of the meaning of each data point can be found in the appendix.

3.5.1 Feature Set 1

Included Data Points:
Age, Total Minutes Played, Points, Rebounds, Assists, Steals, Blocks, TS% (True Shooting %), Win Shares, Opportunity, Defensive Ability, BPM (box plus minus)

Rationale: The inspiration behind this feature set is to take a deeper look at the entirety of a player's statistics. This set covers all the per game statistics like points, rebounds, assists, blocks and steals as well as shooting percentage. However, it will also take into consideration how much a player impacts the game. Using statistics like Win Shares and Box Plus Minus, it can be determined where this player compares against the rest of the league from an overall contribution standpoint.

3.5.2 Feature Set 2

Included Data Points:
Age, Total Minutes Played, Defensive Ability, FG%, 3PT%, FT%, Win Shares, BPM

Rationale: This feature set was developed to specifically not include per game statistics. This way the set will only take into account statistics that are considered season long. It is an interesting comparison to do so since the average fan only takes into account per game statistics.

3.5.3 Feature Set 3

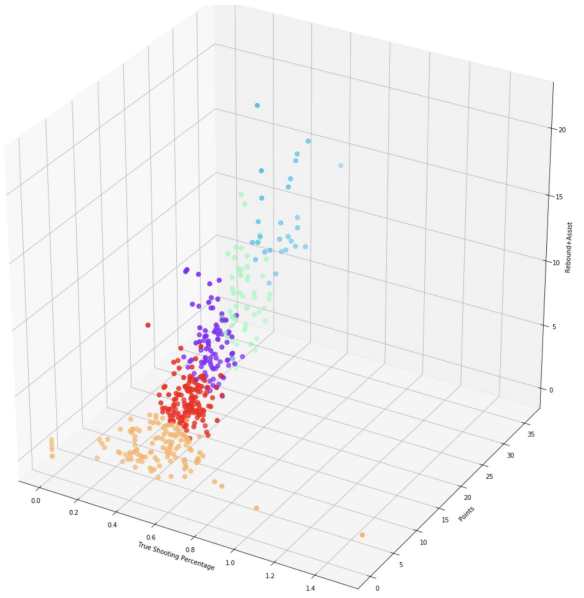
Included Data Points:
Age, Total Minutes Played, Delta Points, Delta Rebounds, Delta Assists, Delta Steals, Delta Blocks, BPM

Rationale: This feature set takes into consideration the comparison of statistics from this season and the previous season to truly look at statistical improvement. Statistical improvement is the focal point of the feature set and players with the best improvement will be singled out. BPM is also used so that each player is compared to the rest of the league.

4. RESULTS

4.1 Unsupervised Learning: Clustering

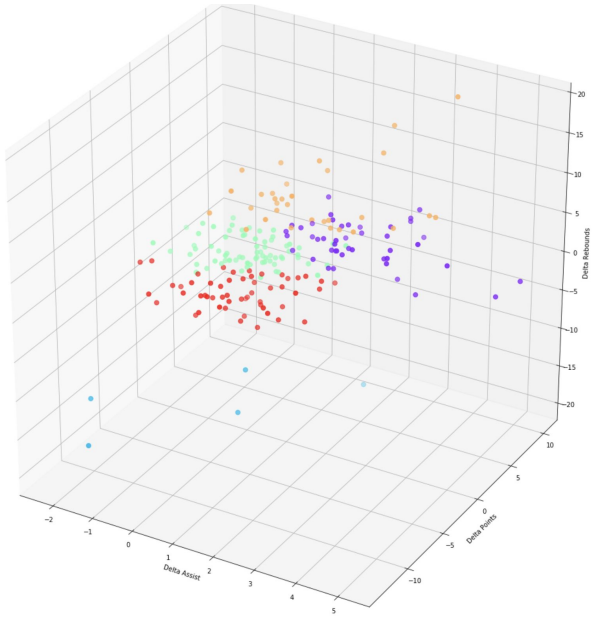
Graph 1: Subset 1
Points, True Shooting Percentage, Rebounds + Assists



Each data point represents a player in the 2018-2019 season. It can be seen that the majority of players have a similar True Shooting Percentage of around 0.4. The outliers that have a higher True Shooting percentage also tend to be lower on points and Rebounds + Assists. This indicates that these players do not play a lot as they have low stats. It can be inferred that they have not taken very many shots due to their limited time on the court and as a result have a higher shooting percentage because of this. Other than these few outliers, most players tend to be trending in a similar direction on the graph. The strongest players, statistically speaking, are

represented in the light blue colour. These are the players that have the highest points per game, the highest combined total of rebounds and assists and also tend to have a higher than average true shooting percentage. These would be the star calibre players that would be in the debate for Most Valuable Player. As players get nearer to the yellow cluster, their strength slowly gets worse. Towards the bottom of the graph are where players with low minutes and poor statistics will live. Average players would live in the purple subset of colours. These would be players that would be low level starters or the first off the bench for each team. Given this, there is a very obvious upwards trend when looking at these three statistics.

Graph 2: Subset 2
Delta Points, Delta Assists, Delta Rebounds



This graph is much more difficult to interpret since there is not a whole lot of consistency between each cluster. The most obvious observation is that most players did not improve a lot when it comes to rebounding and assists. There is a lot of decline in those two categories, specifically for assists. It can be deducted that the league is shying more and more away from assist heavy basketball. This is expected with the amount of talented pure shooters that exist in the league. They are incredibly adept at creating their own scoring opportunities and often do not need an assist to make this happen. As for rebounding, this is a very difficult stat to improve year over year. A lot of players will have very similar averages for this statistic each season because of the fact that rebounds can be accurately predicted by looking at players heights. Characteristically speaking, taller players get more rebounds and this has always been the case. It is tough to improve drastically with this statistic. The largest amount of player improvement can be seen on the points axis. Basketball is evolving to be more offensively minded game, which means each year there will be more points scored and more players will score more points. This means that on average, most players will improve in scoring from year to year, even if slightly. This can obviously be seen on the graph as there are very few players that significantly declined in terms of their scoring points between seasons. The players who have the largest leap in points between seasons are traditionally the ones who do well in Most Improved Player voting.

4.2 Supervised Learning: Classification

Three classification models were considered for supervised learning. Firstly, logistic regression was considered. This model was appropriate for the business problem of determining the MIP because it applies when there is a binary class variable and numeric features variables. Secondly, we considered decision trees. While decision tree modelling tends to overfit the data, it does provide a comprehensive view of the classification rules that it follows, making it easily interpretable (Andrade, 2016). Lastly, we implemented a K Nearest Neighbors (KNN) model using two methods of parameter tuning.

Following this, we performed k-fold cross validation to evaluate each model and feature set pair. We looked at two metrics to evaluate the models. Firstly, we looked at accuracy. However,

we quickly realized that it was not an ideal metric to look at for our models. This is because there are many more rows of data that correspond to the class variable being zero in comparison to one: There are only ten rows of data in each season set which has a value of one for the class variable, in comparison to around 400 labelled as zero. Accuracy should be used as the performance metric when the cost of false positives and false negatives are similar. For our uneven binary classification, we used the F1 score of each model and feature set pair to evaluate their performances. F1 score is a balance of precision and recall, where precision measures the number of True Positives divided by the total number of true observations labeled in the dataset, and recall measures the number of True Positives divided by the total number of true observations predicted by the model (Brownlee, 2014). The results of each model and feature set pair are outlined below. Steps to generate these values can be found in the jupyter notebook files attached.

4.2.1 Logistic Regression

As was predicted, the highest mean F1 score was obtained from feature set three. However, the F1 scores are generally low regardless of feature set. In terms of accuracy, all three feature sets generated almost identical values.

1) **Feature Set 1**
   Mean Accuracy Score: 0.9524
   Mean F1 Score: 0.0737
2) **Feature Set 2**
   Mean Accuracy Score: 0.9618
   Mean F1 Score: 0.0383
3) **Feature Set 3:**
   Mean Accuracy Score: 0.9610
   Mean F1 Score: **0.4342**

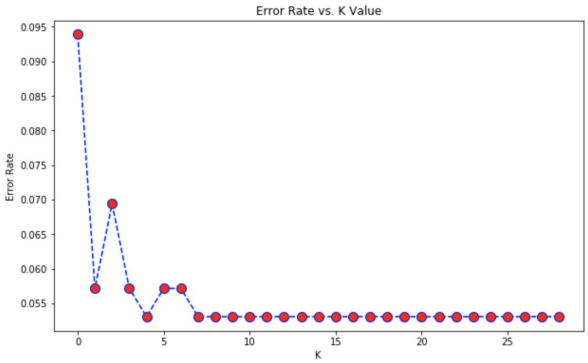
4.2.2 Decision Tree Model

Similarly to logistic regression, the third feature set generated the highest F1 value and the accuracy values were generally close for all feature sets.

1) **Feature Set 1**
   Mean Accuracy Score: 0.9512
   Mean F1 Score: 0.1098
2) **Feature Set 2**
   Mean Accuracy Score: 0.9618
   Mean F1 Score: 0.0645
3) **Feature Set 3:**
   Mean Accuracy Score: 0.9504
   Mean F1 Score: **0.3532**

4.2.3 K Nearest Neighbors (KNN)

The KNN model was implemented using k = 1, also called a One Nearest Neighbour model. In order to find the optimal number of neighbours for each feature set, two methods of parameter tuning were used. The first method is the intuitive method. For each value of k between 1 and 30, the average error rate was computed using K-Fold cross validation (10 folds). The error rate is defined as the number of occurrences where the predicted label is not equal to the actual label. The below graph is the error rate for feature set 3. We can see that using 5 neighbours yields the lowest error rate.



The second method parameter tuning method we used was Grid Search cross validation. When using F1 score as the scoring criteria and a default value of 5 folds, k = 1 is the optimal solution for all three features sets. Therefore, all three feature sets are implemented using the One Nearest Neighbours algorithm.

Out of the three models we evaluated, KNN performed the worst. This is because the algorithm does not construct a generalized model from the

training dataset, it simply memorizes the dataset instead, and uses it when testing is needed. (Brownlee, 2014). KNN generally performs better with a lower number of features, because as the number of features increases, the amount of training data needed in order to avoid overfitting also increases exponentially (Navlani, 2018). There are over 8 features in each of the feature sets, but only 130 observations that are labeled as MIP. Therefore KNN has the lowest performance compared to Decision Tree and Logistic Regression, despite using two methods of parameter tuning.

1) **Feature Set 1**
   Mean Accuracy Score: 0.9239
   Mean F1 Score: 0.1282
2) **Feature Set 2**
   Mean Accuracy Score: 0.9239
   Mean F1 Score: 0.1282
3) **Feature Set 3:**
   Mean Accuracy Score: 0.9317
   Mean F1 Score: **0.1831**

4.3 2018-2019 Season Predictions

4.3.1 Decision Tree Model Trained on Feature Set 3

The chart below depicts the results that the Decision Tree Model gave for potential Most Improved Player candidates using the third feature set:

	Player	DT Predictions
5481	Jae Crowder\crowdja01	1
5486	Anthony Davis\davisan02	1
5501	Damyean Dotson\dotsoda01	1
5505	Jared Dudley\dudleja01	1
5519	Yogi Ferrell\ferreyo01	1
5625	Zach LaVine\lavinza01	1
5638	Kevon Looney\looneke01	1
5646	Trey Lyles\lylestr01	1

The Outliers

The first thing that jumps out from this list of players is wide range of age of players that have been selected. The youngest being Kevon Looney who is only 23 compared to Jared Dudley who is 34. Dudley is certainly an outlier of the data. Not only is he the oldest of the group but he also has the lowest average points per game of the set. However, this is not necessarily an incorrect prediction. He did improve in every statistical category in 2018-2019 besides dropping from 1.6 assists per game to 1.4. Therefore, there is a valid argument that he did improve, but not enough so to be considered the most improved in the league. Another interesting outlier is Trey Lyles. He did not improve in any category that the model looked at, in fact he declined greatly from 2017-2018 to the 2018-2019 season. Another player that declined in 2018-2019 was Yogi Ferrell. He saw his playing minutes diminish significantly as he signed to a new team, the Sacramento Kings, in the off-season. This resulted in career-low statistics across the board besides his shooting percentages, which seemed to take a jump as they have each season he has played in the league. However, this is not taken into account in feature set 3. Each of these players have seen very minimal improvement in the 2018-2019 season and are nothing but outliers compared to the rest of the data.

The Contenders

Now with the outliers removed, a deep dive can be done on the legitimate contenders for winning the Most Improved Player Award. The four most likely to win the award are Damyean Dotson, Zach LaVine, Jae Crowder and Kevon Looney. Each of these players saw fairly strong improvements in the 2018-2019 season compared to the 2017-2018 season. From these four, there are two tiers of improvement. Jae Crowder and Kevon Looney are in the lower tier. They both had very similar statistical improvement which saw them both increase around 2 points per game, around 1 assist per game, close to 2 rebounds per game and then stayed consistent with their rebounds and steals. The point where they differ is

their BPM. Jae Crowder had a 0.4 BPM in that season which means he is a very average contributor. Kevon Looney had a 3.7 BPM. This simply comes down to the fact that Kevon Looney played for the Golden State Warriors which were championship contender. Whereas Jae Crowder played for Utah, which is a pretty average team. Due to the fact that Looney was on a better team, he had better players around him which improved his statistics. Unfortunately, Crowder did not have the same experience and suffered in this particular statistic.

The most likely candidates to win the award come down to Damyean Dotson and Zach LaVine. Both of them are young, explosive players. Dotson played in just his second NBA season and showed great improvement over his rookie campaign. He close to doubled his numbers from his rookie season. He improved over six points per game, over one assist per game and close to two rebounds per game. He was thrust into the starting lineup of a struggling Knicks team and made the most of it. After only playing forty games in his rookie season, he suddenly was playing seventy-three in his sophomore year and started in forty of them. Despite having very solid numbers, his BPM was much below zero. This was partially due to the fact that the Knicks were one of the worst teams in the league, but also Dotson's defensive efficiency did not match his offensive efficiency. Compared to Zach LaVine, a seasoned NBA player who entered his fifth season at just the age of 23. After declining in 2017-2018, LaVine had a fantastic bounceback year in 2018-2019. He improved his points per game by seven, brought up his assists by one and a half per game and improved his rebounding by nearly one per game as well. Increasing points from 16.7 to 23.7 is no small feat. He drastically improved his offensive ability and made it known on the court. Prior to this season, he was known from his flashy dunks and appearances in the dunk contest. After the 2018-2019 season, he showed that he was more than a dunker, he proved that he could contribute in multiple ways. This was partially due to the fact that his minutes increased by over seven a game and it was the first time since his sophomore season that he played more than sixty games. Bouncing back after a few injury riddled seasons is nothing short of impressive. LaVine is the obvious candidate of the bunch to win the award, and in fact he received a vote to win the award. This means that the decision tree model was able to accurately predict one player who was in the conversation to win the Most Improved Player award in the 2018-2019 season.

MVP-esque?

The most shocking result to come from this model was that it predicted Anthony Davis as one of the likely candidates. 2017-2018 was a fantastic season for Anthony Davis, he seemed unstoppable. He was putting up statistics that many compared to NBA legend Kareem Abdul-Jabbar, it was basically unheard of. Due to his unique athletic ability as a 6'10" center, he was turning heads every game he played. This resulted in him being an MVP finalist and he ended up finishing third in MVP voting in 2017-2018 as a 24 year old. He was paving his way to be a sure fire hall of famer before his 25th birthday. However, basketball is not always kind to its budding stars and as a result he was injured for much of the 2018-2019 season. Alongside of this, the New Orleans Pelicans were struggling mightily and it looked like Davis may miss the playoffs for the fifth time in his career. Due to his injury and the fact that the Pelicans were a lower tier team in the competitive Western Conference, Davis's stats took a hit. In fact, he averaged the lowest amount of points since the 2015-2016 season. Despite this, he averaged career highs in rebounds in assists. He upped his average of 11.1 rebounds per game to twelve, an extremely impressive number to achieve given his offensive prowess. He also averaged a career high 3.9 assists which was up from 2.3 the season prior. He had a great season and did improve, however he did not improve enough to be the Most Improved Player. However, if he had played a full season he would have certainly been a contender for the MVP award for a second season.

4.3.2 Logistic Regression Model Trained on Feature Set 3

	Player	LR Predictions
5427	Eric Bledsoe\bledser01	1
5481	Jae Crowder\crowdja01	1
5646	Trey Lyles\lylestr01	1

The Outliers

Once again Trey Lyles is seen in a prediction set and again he is an outlier to the data. As previously mentioned, he did not have a statistically strong year and regressed from year priors. This prompts an interesting question, which is what are the models seeing to be able to determine that Trey Lyles deserves to be considered for this award? In many ways he fits the mould properly. He is a dynamic young player who played on a great organization in the Denver Nuggets. He is a solid contributor to the team and seems to have a bright future. However, looking solely on statistics it is difficult to find bright spots where he would be a candidate for any NBA award. Another outlier in this model is Eric Bledsoe. Much like Trey Lyles, Bledsoe declined in a lot of areas in 2018-2019. He experienced his lowest points per game since 2012-2013 and only saw improvement in his assists per game where he improved by 0.5. Bledsoe had a decent year all round as he had a 3.5 BPM and was on a championship contender in the Milwaukee Bucks. However, did not improve enough to be considered for the Most Improved Player Award.

The Contenders

The only remaining option is Jae Crowder. This is the second model that has predicted Crowder to be a candidate. This solidifies the fact that he did in fact have a great season and took strides to improve as a player. After being on a new team in 2018-2019, Crowder seemed to improve a lot which propelled him to stronger stats. The 28 year-old has bounced around the league throughout his seven seasons but never fails to improve each team that he plays for.

4.3.3 Real World Comparison

Below are the real voting results from the 2018-2019 season:

Rank	Player	Age	Team
1	Pascal Siakam	24	TOR
2	D'Angelo Russell	22	BRK
3	De'Aaron Fox	21	SAC
4	Buddy Hield	26	SAC
5	Nikola Vučević	28	ORL
6	Domantas Sabonis	22	IND
7	Montrezl Harrell	25	LAC
8	Derrick Rose	30	MIN
9	John Collins	21	ATL
10	Giannis Antetokounmpo	24	MIL
11T	Malik Beasley	22	DEN
11T	Danilo Gallinari	30	LAC
11T	Paul George	28	OKC
14T	Bojan Bogdanović	29	IND
14T	Thomas Bryant	21	WAS
16T	Willie Cauley-Stein	25	SAC
16T	Spencer Dinwiddie	25	BRK
16T	Bryn Forbes	25	SAS
16T	Jerami Grant	24	OKC
16T	Zach LaVine	23	CHI
16T	Josh Richardson	25	MIA
16T	Derrick White	24	SAS

As seen in the table, from each of the models we were able to accurately predict 1 vote getter. The decision tree model did in fact decide that Zach LaVine was an acceptable candidate to win the Most Improved Player award. In reality, he received one vote which put him in contention for the award. He was nowhere close to achieving the 86 first place votes that Pascal Siakam received, but it was a vote nonetheless. The confusion matrix below depicts the accuracy of the decision tree model:

		Predicted Results	
		Yes	No
Actual Results	Yes	1	21
	No	7	415

As it can be seen, the model correctly predicted 410 true negatives, or 93% accurate. However, this model was only 4.5% accurate predicting true positives.

5. CONCLUSIONS

The main finding of this paper is how difficult it is to accurately predict the Most Improved Player award winner in the NBA. The award is very volatile and no two award winners are alike. Each have different kinds of improvements which are incredibly hard to track and replicate. This results in difficulty accurately predicting future winners. Between the results, we see commonalities with the likes of Trey Lyles and Jae Crowder appearing in both. We also see quite a good candidate in Zach LaVine and a very surprising candidate in Anthony Davis.

This type of information, if modelled correctly, can prove to be invaluable to NBA front offices. A lot of players who are eligible for this award tend to fly under the radar. Being able to accurately determine these players prior to their rise to stardom can result in teams having high value trade opportunities and increase the success of their organization for years to come.

All in all, the best model-feature pair that was for the Logistic Regression model. We expected this model to perform better than decision tree modelling because it doesn't overfit the data in the same way that decision trees do. This consideration was especially important for the type of data we were dealing with since there were a low number of rows of data with the class variable labelled as one. This makes it difficult for any model to make predictions for different datasets. Given more time, we would have done more research into which modelling techniques and scores would better suit our data and business problem. Additionally, we would have developed additional features from our existing dataset as well as other datasets.

6. SOURCES

1. Andrade, A. (2016). Decision tree. Retrieved from http://datascienceguide.github.io/decision-trees.
2. Brownlee, J. (2019, June 19). Classification Accuracy is Not Enough: More Performance Measures You Can Use. Retrieved from https://machinelearningmastery.com/classification-accuracy-is-not-enough-more-performance-measures-you-can-use/.
3. Basketball Statistics and History. (n.d.). Retrieved from https://www.basketball-reference.com/.
4. Bratulić, D. (2019, May 15). Predicting 2018–19 NBA's Most Valuable Player using Machine Learning. Retrieved from https://towardsdatascience.com/predicting-2018-19-nbas-most-valuable-player-using-machine-learning-512e577032e3.
5. Hore, S., & Bhattacharya, T. (2018, September 27). A Machine Learning Based Approach Towards Building a Sustainability Model for NBA Players. Retrieved from

https://ieeexplore.ieee.org/document/8473102.
6. Kalbrosky, B. (2019, January 28). What is the peak age in the NBA? Probably 27 years old. Retrieved from https://hoopshype.com/2018/12/31/nba-aging-curve-father-time-prime-lebron-james-decline/.
7. Krishni. (2018, December 21). K-Fold Cross Validation. Retrieved from https://medium.com/datadriveninvestor/k-fold-cross-validation-6b8518070833.
8. Maini, V. (2018, May 28). Machine Learning for Humans, Part 3: Unsupervised Learning. Retrieved from https://medium.com/machine-learning-for-humans/unsupervised-learning-f45587588294.
9. Miljković, D. (2010, November 29). The use of data mining for basketball matches outcomes prediction. Retrieved from https://ieeexplore.ieee.org/document/5647440.
10. Navlani, A. (2018, August 2). KNN Classification using Scikit-learn. Retrieved from https://www.datacamp.com/community/tutorials/k-nearest-neighbor-classification-scikit-learn.
11. randerson112358. (2019, August 4). NBA Data Analysis Using Python & Machine Learning. Retrieved from https://medium.com/@randerson112358/nba-data-analysis-exploration-9293f311e0e8.
12. Schuhmann, J. (2019, March 27). Numbers to Know - Most Improved Player award. Retrieved from https://www.nba.com/article/2019/03/27/numbers-know-most-improved.
13. Soliman, G., El-Nabawy, A., Misbah, A., & Edawlatly, S. (2018, March 26). Predicting all star player in the national basketball association using random forest. Retrieved from https://ieeexplore.ieee.org/document/8324371.
14. Xu, C., & Yu, Y. (2015, March 30). Measuring NBA Players' Mood by Mining Athlete-Generated Content. Retrieved from https://ieeexplore.ieee.org/document/7070015.

7. APPENDIX

7.1 Description of Player Statistics

1. Age: Current age of the player
2. Points: Average points scored per game played in a given season
3. Rebounds: Average total rebounds per game played in a given season (includes offensive and defensive)
4. Assists: Average assists per game played in a given season
5. Steals: Average steals per game played in a given season
6. Blocks: Average blocks per game played in a given season
7. TS% (True Shooting Percentage): A measure of shooting efficiency that takes into account field goals, 3 point shots and free throws
8. Win Shares: An estimate of the number of wins contributed by a player
9. BPM (Box Plus Minus): An estimate of the points per 100 possessions that the player contributed compared to an average player (combines offensive and defensive statistics). 0 is a completely average player, if you have a BPM of 10 you contribute 10 points more than the average player.
10. FG% (Field Goal Percentage): A field goal is any shot taken that is considered 2 points or 3 points. Field goal percentage is the number of shots made divided by number attempted
11. 3PT% (3 Point Percentage): The number of 3 point shots made divided by number attempted

12. FT% (Free Throw Percentage): The number of free throws made divided by number attempted
13. USG% (Usage Percentage): An estimate of the number of possessions where a player touched the ball last on their team (considers shots, free throws and turnovers).
14. AST% (Assist Percentage): An estimate of the percentage of field goals this player assisted on for his teammates

7.2 Features Code (format is .py)

```
#!/usr/bin/env python
# coding: utf-8

# In[176]:


import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns
from mpl_toolkits.mplot3d.axes3d import Axes3D
get_ipython().run_line_magic('matplotlib', 'inline')

from sklearn.linear_model import
LogisticRegression
from sklearn.model_selection import KFold
from sklearn import metrics
from sklearn import datasets
from sklearn.metrics import accuracy_score

import statsmodels.api as sm
from statsmodels.stats.outliers_influence import
variance_inflation_factor


# In[62]:


df19 =
pd.read_csv("player_stats_2018_2019.csv")


# In[63]:


df19["season"] = 19


# In[64]:


df18 =
pd.read_csv("player_stats_2017_2018.csv")


# In[65]:


df18["season"] = 18


# In[66]:


df17 =
pd.read_csv("player_stats_2016_2017.csv")


# In[67]:


df17["season"] = 17


# In[68]:


df16 =
pd.read_csv("player_stats_2015_2016.csv")
```

```
# In[69]:


df16["season"] = 16


# In[70]:


df15 =
pd.read_csv("player_stats_2014_2015.csv")


# In[71]:


df15["season"] = 15


# In[72]:


df14 =
pd.read_csv("player_stats_2013_2014.csv")


# In[73]:


df14["season"] = 14


# In[74]:


df13 =
pd.read_csv("player_stats_2012_2013.csv")


# In[75]:


df13["season"] = 13


# In[76]:


df12 =
pd.read_csv("player_stats_2011_2012.csv")


# In[77]:


df12["season"] = 12


# In[78]:


df11 =
pd.read_csv("player_stats_2010_2011.csv")


# In[79]:


df11["season"] = 11


# In[80]:


df10 =
pd.read_csv("player_stats_2009_2010.csv")


# In[81]:


df10["season"] = 10


# In[82]:
```

```
df9 = pd.read_csv("player_stats_2008_2009.csv")

# In[83]:

df9["season"] = 9

# In[84]:

df8 = pd.read_csv("player_stats_2007_2008.csv")

# In[85]:

df8["season"] = 8

# In[86]:

df7 = pd.read_csv("player_stats_2006_2007.csv")

# In[87]:

df7["season"] = 7

# In[88]:

df6 = pd.read_csv("player_stats_2005_2006.csv")

# In[89]:

df6["season"] = 6

# In[90]:

master = pd.DataFrame()

# In[91]:

master =
master.append([df6,df7,df8,df9,df10,df11,df12,df
13,df14,df15,df16,df17,df18,df19],
ignore_index=True, sort=False)

# In[92]:

master.shape

# In[93]:

master = master.sort_values(by=['season','Rk'],
ascending=[0,1])

# In[94]:

#CALCULATE ADDITIONAL DATA POINTS
FOR FEATURES
for index, row in master.iterrows():
    current_player = row["Player"]
    current_season = row["season"]
    prev_season = current_season - 1
    if(current_player in
master[master["season"]==prev_season]["Player"]
.values):
```

```
    #FEATURE 1: OPPORTUNITY - DATA
POINTS

    # DELTA USG
    prev_usg =
master[master["season"]==prev_season][master["
Player"]==current_player]["USG%"]
    current_usg = row["USG%"]
    delta_usg = float(current_usg) -
float(prev_usg)
    master.loc[index,"Delta USG%"] = delta_usg
    # DELTA AST
    prev_ast =
master[master["season"]==prev_season][master["
Player"]==current_player]["AST%"]
    current_ast = row["AST%"]
    delta_ast = float(current_ast) - float(prev_ast)
    master.loc[index,"Delta AST%"] = delta_ast

    #FEATURE 2: IMPORTANT DELTAS -
DATA POINTS

    #DELTA POINTS
    prev_pts =
master[master["season"]==prev_season][master["
Player"]==current_player]["PTS"]
    current_pts = row["PTS"]
    delta_pts = float(current_pts) -
float(prev_pts)
    master.loc[index,"Delta PTS"] = delta_pts
    #DELTA TOTAL REBOUNDS
    prev_trb =
master[master["season"]==prev_season][master["
Player"]==current_player]["TRB"]
    current_trb = row["TRB"]
    delta_trb = float(current_trb) - float(prev_trb)
    master.loc[index,"Delta TRB"] = delta_trb
    #DELTA ASSISTS
    prev_asts =
master[master["season"]==prev_season][master["
Player"]==current_player]["AST"]
    current_asts = row["AST"]
    delta_asts = float(current_asts) -
float(prev_asts)
    master.loc[index,"Delta AST"] = delta_asts
    #DELTA STL
    prev_stl =
master[master["season"]==prev_season][master["
Player"]==current_player]["STL"]
    current_stl = row["STL"]
    delta_stl = float(current_stl) - float(prev_stl)
    master.loc[index,"Delta STL"] = delta_stl
    #DELTA STL
    prev_blk =
master[master["season"]==prev_season][master["
Player"]==current_player]["BLK"]
    current_blk = row["BLK"]
    delta_blk = float(current_blk) -
float(prev_blk)
    master.loc[index,"Delta BLK"] = delta_blk



# In[95]:


# New Feature 1 : Opportunity
master["Opportunity"] = master["Delta USG%"] +
(master["Delta AST%"]*-1)


# In[96]:


# New Feature 2: Important Deltas
master["Impt Deltas"] = master["Delta PTS"] +
master["Delta TRB"] + master["Delta AST"]


# In[97]:


# New Feature 3: Defensive Ability
master["Defensive Ability"] = ( master["STL"] +
master["BLK"] ) / master["PF"]
```

```
# In[98]:


# New Feature 4: Total Mins
master["Total Mins"] = master["MP"] *
master["G"]


# In[99]:


# DATA CLEANING

# Remove any players with current season less
than 1000 mins and previous season less than 250
mins
dropped = 0
master = master[master["Total Mins"] > 250]
print(master.shape)
for index, row in master.iterrows():
    current_player = row["Player"]
    current_season = row["season"]
    prev_season = current_season - 1
    if(current_player in
master[master["season"]==prev_season]["Player"]
.values):
        if (row["Total Mins"] < 1000):
            master.drop(index, inplace=True)
            dropped += 1

print("dropped this many < 1000: " + str(dropped)
+ "rows")


# In[100]:


master.shape


# # Drop 2018-2019 season

# In[101]:


training_data = master[master["season"]!=19]


# In[102]:


# Define feature set 1
feature_set_1 = pd.DataFrame
feature_set_1 = training_data[['Player','Age','Total
Mins','Opportunity','Defensive
Ability','PTS','TRB','AST','STL','BLK','TS%','WS'
,'BPM','MIP']]


# In[103]:


# Define feature set 2
feature_set_2 = pd.DataFrame
feature_set_2 = training_data[['Player','Age','Total
Mins','Defensive
Ability','FG%','3P%','FT%','WS','BPM','MIP']]


# In[104]:


# Define feature set 3
feature_set_3 = pd.DataFrame
feature_set_3 = training_data[['Player','Age','Total
Mins','Delta PTS','Delta TRB','Delta AST','Delta
STL', 'Delta BLK','BPM','MIP']]


# # Use feature set 3 2018-2019 as testing data

# In[105]:
```

```
testing_data = master[master["season"]==19]


# In[106]:


last_season_data = pd.DataFrame
last_season_data =
testing_data[['Player','Age','Total Mins','Delta
PTS','Delta TRB','Delta AST','Delta STL', 'Delta
BLK','BPM','MIP']]


# In[107]:


get_ipython().run_line_magic('store', 'master')


# In[108]:


get_ipython().run_line_magic('store',
'feature_set_1')
get_ipython().run_line_magic('store',
'feature_set_2')
get_ipython().run_line_magic('store',
'feature_set_3')
get_ipython().run_line_magic('store',
'last_season_data')


# # Use master 2018-2019 for clustering

# In[109]:


clustering_data = master[master["season"]==19]


# In[110]:


get_ipython().run_line_magic('store',
'clustering_data')


# # Data Exploration Graphs

# In[137]:


master.columns


# In[146]:


corr_data =
pd.DataFrame(master[['PTS','FG%','3P%','FT%','T
S%']])


# In[160]:


sns.heatmap(corr_data.corr(),cmap='coolwarm')
plt.title('Correlation: Points vs. Shooting
Efficiencies')


# In[166]:


sns.jointplot(x='PTS',y='Age',data=master)
plt.suptitle("Age vs. Points")


# In[165]:


sns.jointplot(x='MP',y='PTS',data=master)
plt.suptitle('Points vs. Minutes Played')


# In[158]:
```

```
sns.boxplot(x='MIP',y='Age',data=master,palette='
rainbow')
plt.title('Age: MIP vs. Non-MIP')


# In[157]:


sns.boxplot(x='MIP',y='PTS',data=master,palette='
rainbow')
plt.title('Points: MIP vs. Non-MIP')


# In[167]:


mips = master[master["MIP"]==1]
sns.countplot(x="Pos", data=mips)
plt.title('MIP Positions Frequency')


# In[ ]:


test = pd.DataFrame(master["Delta AST"])
test = test.dropna()
fig = plt.figure(figsize=(14,6))

# `ax` is a 3D-aware axis instance because of the
projection='3d' keyword argument to add_subplot
ax = fig.add_subplot(1, 2, 1, projection='3d')

p = ax.plot_surface(master["Delta TRB"],
master["Delta PTS"], test, rstride=4, cstride=4,
linewidth=0)

# surface_plot with color grading and color bar
ax = fig.add_subplot(1, 2, 2, projection='3d')
p = ax.plot_surface(master["Delta TRB"],
master["Delta PTS"], test, rstride=1, cstride=1,
cmap='coolwarm', linewidth=0, antialiased=False)
cb = fig.colorbar(p, shrink=0.5)


# In[ ]:


x = master["Delta TRB"]
y = master["Delta PTS"]
z = pd.DataFrame(master["Delta AST"])
z = z.dropna()

ax.plot_surface(x, y, z, rstride=1, cstride=1,
            cmap='coolwarm', edgecolor='none')
ax.set_title('surface');


# In[ ]:
```

7.3 Logistic Regression Code (format is .py)

```
#!/usr/bin/env python
# coding: utf-8

# # Feature Set 1

# In[38]:


import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns
get_ipython().run_line_magic('matplotlib', 'inline')

from sklearn.linear_model import
LogisticRegression
from sklearn.model_selection import KFold
from sklearn import metrics
from sklearn import datasets
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
```

```
import statistics

import statsmodels.api as sm
from statsmodels.stats.outliers_influence import
variance_inflation_factor


# In[39]:


get_ipython().run_line_magic('store', '-r master')
get_ipython().run_line_magic('store', '-r
feature_set_1')
get_ipython().run_line_magic('store', '-r
feature_set_2')
get_ipython().run_line_magic('store', '-r
feature_set_3')


# In[40]:


def k_fold_lr (X,Y):

    # Perform K-fold cross validation to make
training and testing datasets, k = 10
    kf = KFold(n_splits=10)
    i = 1 #counter for iterations
    acc_set = []
    F1 = []

    for training, testing in kf.split(X):

        #Intialize model
        model = LogisticRegression()

        print("------------------------- start k
=",i,"-------------------------")
        print("Train Index: ", training, "\n")
        print("Test Index: ", testing, "\n")

        X_training_data = X.iloc[training]
        X_testing_data = X.iloc[testing]
        Y_training_data = Y.iloc[training]
        Y_testing_data = Y.iloc[testing]

        #Fit model
        model.fit(X_training_data, Y_training_data)

        #Determine accuracy score
        class_predict =
model.predict(X_testing_data)
        acc_score = accuracy_score(Y_testing_data,
class_predict)
        print("Accuracy Score: ", acc_score, "\n")
        print("------------------------- end k
=",i,"-------------------------","\n")

        acc_set.append(acc_score)
        F1.append(f1_score(Y_testing_data,
class_predict))
        i += 1

    return acc_set, F1


# In[41]:


# Set X as features data
feature_set_1 = feature_set_1.dropna()
X = feature_set_1.drop('MIP',axis=1)
X = X.drop('Player',axis=1)


# In[42]:


#Set Y = target class
Y = feature_set_1.MIP


# In[43]:


#Run k-fold and get accuracy
```

```
acc_set1 = k_fold_lr(X,Y)[0]
F1_set1 = k_fold_lr(X,Y)[1]
print("Accuracy Score Set",acc_set1,"\n")
avg_acc1 = statistics.mean((acc_set1))
print("F1 Score Set",F1_set1,"\n")
avg_F1_1 = statistics.mean((F1_set1))
```

```
# In[44]:
```

```
vif = pd.DataFrame()
vif["VIF Factor"] =
[variance_inflation_factor(X.values, i) for i in
range(X.shape[1])]
vif["features"] = X.columns
vif.round(1)
```

```
# # Feature Set 2
```

```
# In[45]:
```

```
# Set X as features data
feature_set_2 = feature_set_2.dropna()
X2 = feature_set_2.drop('MIP',axis=1)
X2 = X2.drop('Player',axis=1)
```

```
# In[46]:
```

```
#Set Y = target class
Y2 = feature_set_2.MIP
```

```
# In[57]:
```

```
#Run k-fold and get accuracy
acc_set2 = k_fold_lr(X2,Y2)[0]
F1_set2 = k_fold_lr(X2,Y2)[1]
print("Accuracy Score Set",acc_set2,"\n")
avg_acc2 = statistics.mean((acc_set2))
print("F1 Score Set",F1_set2,"\n")
avg_F1_2 = statistics.mean((F1_set2))
```

```
# # Feature Set 3
```

```
# In[58]:
```

```
# Set X as features data (remove class variable
and player name)
feature_set_3 = feature_set_3.dropna()
X3 =
feature_set_3.drop('MIP',axis=1).drop('Player',
axis=1)
feature_names3 = list(X3.columns)
#Set Y as target class
Y3 = feature_set_3.MIP
```

```
# In[59]:
```

```
#Initialize the model
acc_set3 = k_fold_lr(X3,Y3)[0]
F1_set3 = k_fold_lr(X3,Y3)[1]
print("Accuracy Score Set",acc_set3,"\n")
avg_acc3 = statistics.mean((acc_set3))
print("F1 Score Set",F1_set3,"\n")
avg_F1_3 = statistics.mean((F1_set3))
```

```
# ## Accuracy Scores
```

```
# In[60]:
```

```
print("Mean Accuracy Score - Feature Set
1:",avg_acc1)
print("Mean Accuracy Score - Feature Set
2:",avg_acc2)
```

```
print("Mean Accuracy Score - Feature Set
3:",avg_acc3)


# ## F1 Scores

# In[61]:


print("Mean F1 Score - Feature Set 1:",avg_F1_1)
print("Mean F1 Score - Feature Set 2:",avg_F1_2)
print("Mean F1 Score - Feature Set 3:",avg_F1_3)


# In[ ]:
```

7.4 Decision Tree Code (format is .py)

```
#!/usr/bin/env python
# coding: utf-8

# # Decision Trees

# In[1]:


import pandas as pd
import numpy as np

from sklearn import tree
from sklearn.tree import DecisionTreeClassifier,
export_graphviz
from sklearn.model_selection import KFold
from sklearn.model_selection import
cross_val_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn import metrics
from sklearn import datasets
from matplotlib import pyplot as plt
import seaborn as sns
import statistics

# import graphviz
import pydotplus
import io
from scipy import misc

get_ipython().run_line_magic('matplotlib', 'inline')


# In[2]:


# Import Feature Set
get_ipython().run_line_magic('store', '-r master')
get_ipython().run_line_magic('store', '-r
feature_set_1')
get_ipython().run_line_magic('store', '-r
feature_set_2')
get_ipython().run_line_magic('store', '-r
feature_set_3')


# In[3]:


#Function that generates the decision tree
visualizations
def show_tree(tree, features, path):
    f = io.StringIO()
    export_graphviz(tree, out_file=f,
feature_names=features)

pydotplus.graph_from_dot_data(f.getvalue()).writ
e_png(path)
    img = plt.imread(path)
    plt.rcParams['figure.figsize'] = (20,20)
    plt.imshow(img)


# In[4]:
```

```
def k_fold_dt (X,Y):

   # Perform K-fold cross validation to make
training and testing datasets, k = 10
   kf = KFold(n_splits=10)
   i = 1 #counter for iterations
   acc_set = []
   F1 = []

   for training, testing in kf.split(X):

      #Initialize model
      model =
DecisionTreeClassifier(min_samples_split=100)

      print("-------------------------- start k
=",i,"--------------------------")
      print("Train Index: ", training, "\n")
      print("Test Index: ", testing, "\n")
      X_training_data = X.iloc[training]
      X_testing_data = X.iloc[testing]
      Y_training_data = Y.iloc[training]
      Y_testing_data = Y.iloc[testing]

      #Fit model
      model.fit(X_training_data, Y_training_data)

      #Determine accuracy score
      class_predict =
model.predict(X_testing_data)
      acc_score = accuracy_score(Y_testing_data,
class_predict)

      print("Accuracy Score: ", acc_score, "\n")
      print("-------------------------- end k
=",i,"--------------------------","\n")

      acc_set.append(acc_score)
      F1.append(f1_score(Y_testing_data,
class_predict))
      i += 1

   return acc_set, F1
```

```
# ## FEATURE SET 1

# In[5]:


# Set X as features data (remove class variable
and player name)
feature_set_1 = feature_set_1.dropna()
X =
feature_set_1.drop('MIP',axis=1).drop('Player',
axis=1)
feature_names1 = list(X.columns)


# In[6]:


#Set Y as target class
Y = feature_set_1.MIP


# In[7]:


#Initialize the model
acc_set1 = k_fold_dt(X,Y)[0]
F1_set1 = k_fold_dt(X,Y)[1]
print("Accuracy Score Set",acc_set1,"\n")
avg_acc1 = statistics.mean((acc_set1))
print("F1 Score Set",F1_set1,"\n")
avg_F1_1 = statistics.mean((F1_set1))


# ## FEATURE SET 2

# In[8]:


# Set X as features data (remove class variable
and player name)
feature_set_2 = feature_set_2.dropna()
```

```
X2 =
feature_set_2.drop('MIP',axis=1).drop('Player',
axis=1)
feature_names2 = list(X2.columns)
#Set Y as target class
Y2 = feature_set_2.MIP


# In[9]:


#Initialize the model
acc_set2 = k_fold_dt(X2,Y2)[0]
F1_set2 = k_fold_dt(X2,Y2)[1]
print("Accuracy Score Set",acc_set2,"\n")
avg_acc2 = statistics.mean((acc_set2))
avg_F1_2 = statistics.mean((F1_set2))


# ## FEATURE SET 3

# In[10]:


# Set X as features data (remove class variable
and player name)
feature_set_3 = feature_set_3.dropna()
X3 =
feature_set_3.drop('MIP',axis=1).drop('Player',
axis=1)
feature_names3 = list(X3.columns)
#Set Y as target class
Y3 = feature_set_3.MIP


# In[11]:


# K fold on Feature Set 3
acc_set3 = k_fold_dt(X3,Y3)[0]
F1_set3 = k_fold_dt(X3,Y3)[1]
print("Accuracy Score Set",acc_set3,"\n")
avg_acc3 = statistics.mean((acc_set3))
avg_F1_3 = statistics.mean((F1_set3))


# ## Accuracy Scores

# In[12]:


print("Mean Accuracy Score - Feature Set
1:",avg_acc1)
print("Mean Accuracy Score - Feature Set
2:",avg_acc2)
print("Mean Accuracy Score - Feature Set
3:",avg_acc3)


# ### The highest accuracy score is for Feature
Set 2

# ## F1 Scores

# In[13]:


print("Mean F1 Score - Feature Set 1:",avg_F1_1)
print("Mean F1 Score - Feature Set 2:",avg_F1_2)
print("Mean F1 Score - Feature Set 3:",avg_F1_3)


# ### The highest F1 score is for Feature Set 3

# In[ ]:
```

7.5 KNN Code (format is .py)

```
#!/usr/bin/env python
# coding: utf-8

# # Feature set 1

# In[125]:
```

```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns
get_ipython().run_line_magic('matplotlib', 'inline')

from sklearn.neighbors import
KNeighborsClassifier
from sklearn.model_selection import KFold
from sklearn.model_selection import
GridSearchCV
from sklearn.model_selection import
train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn import datasets
from sklearn import metrics

from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score,
precision_score, recall_score, f1_score

from statistics import mean
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import
variance_inflation_factor


# In[126]:


import warnings
warnings.filterwarnings('always')  # "error",
"ignore", "always", "default", "module" or "once"


# In[127]:


get_ipython().run_line_magic('store', '-r master')
get_ipython().run_line_magic('store', '-r
feature_set_1')


# In[128]:


# Set X as features data
feature_set_1 = feature_set_1.dropna()
X1 = feature_set_1.drop('MIP',axis=1)
X1 = X1.drop('Player',axis=1)

# Set Y = target class
Y1 = feature_set_1.MIP
X = X1
Y = Y1


# ## Scalarize the features

# In[129]:


sc = StandardScaler()
sc.fit(X)
sc_features = sc.transform(X)


# ## Get best k value approach 1: Intuitive method

# In[130]:


def get_error_rate(X, Y):
    error_rate = []
    for i in range(1,30):
        # Implement K-fold with 10 splits
        kf = KFold(n_splits=10)
        for training, testing in kf.split(X):
            knn =
KNeighborsClassifier(n_neighbors=i)
            print("-------------------------iter
starts----------------------")
```

```
        print("Train Index: ", training, "\n")
        print("Test Index: ", testing)
        X_training_data = X.iloc[training]
        X_testing_data = X.iloc[testing]
        Y_training_data = Y.iloc[training]
        Y_testing_data = Y.iloc[testing]
        # Fit model
        knn.fit(X_training_data, Y_training_data)
        knn.score(X_training_data,
Y_training_data)
        class_predict =
knn.predict(X_testing_data)
        acc_score =
accuracy_score(Y_testing_data, class_predict)
        print("Class Predict: ", class_predict)
        print("Accuracy Score: ", acc_score)
        predicted = knn.predict(X_testing_data)
        error = predicted != Y_testing_data
        print("-------------------------iteration
ends----------------------")
      error_rate.append(mean(error))
   return error_rate
```

```
# In[131]:
```

```
error_rate = get_error_rate(X,Y)
```

```
# In[132]:
```

```
plt.figure(figsize=(10,6))
plt.plot(error_rate, color='blue', linestyle='dashed',
marker='o',
       markerfacecolor='red', markersize=10)
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
```

```
# In[133]:
```

```
plt.plot(error_rate[0:20])
plt.show
```

```
# In[134]:
```

```
print("Error rate is minimized when k is " +
str(error_rate.index(min(error_rate))+1))
```

```
# ## We can tell that the error rate is minimized
when k = 5
```

```
# In[135]:
```

```
knn = KNeighborsClassifier(n_neighbors=5)
```

```
# ## Get best k value approach 2: GridSearchCV
Optimization
```

```
# In[136]:
```

```
parameter = {'n_neighbors':np.arange(1,40)}
knn_grid_cv= GridSearchCV(knn, parameter,
cv=10, scoring='accuracy')
knn_grid_cv.fit(X,Y)
print("optimal accuracy: " +
str(knn_grid_cv.best_score_))
print("optimal k value: " +
str(knn_grid_cv.best_params_))
```

```
# ## k = 4 gives the most optimal accuracy score
```

```
# In[137]:
```

```
parameter = {'n_neighbors':np.arange(1,40)}
knn_grid_cv= GridSearchCV(knn, parameter,
cv=10, scoring='precision')
knn_grid_cv.fit(X,Y)
print("optimal precision: " +
str(knn_grid_cv.best_score_))
print("optimal k value: " +
str(knn_grid_cv.best_params_))
```

```
# ## k = 1 gives the most optimal precision score

# In[138]:
```

```
parameter = {'n_neighbors':np.arange(1,40)}
knn_grid_cv= GridSearchCV(knn, parameter,
cv=10, scoring='recall')
knn_grid_cv.fit(X,Y)
print("optimal recall: " +
str(knn_grid_cv.best_score_))
print("optimal k value: " +
str(knn_grid_cv.best_params_))
```

```
# ## k = 1 gives the most optimal recall score

# In[139]:
```

```
parameter = {'n_neighbors':np.arange(1,40)}
knn_grid_cv= GridSearchCV(knn, parameter,
cv=10, scoring='f1')
knn_grid_cv.fit(X,Y)
print("optimal f1: " +
str(knn_grid_cv.best_score_))
print("optimal k value: " +
str(knn_grid_cv.best_params_))
```

```
# ## k = 1 gives the best F1 score, therefore we
use k = 1

# ## Implement knn using k = 1

# In[140]:


def k_fold_knn(k, X, Y):
    # Perform K-fold cross validation to make
training and testing datasets, k = 10
    kf = KFold(n_splits=10)
    i = 1 #counter for iterations
    accuracy = []
    precision = []
    recall = []
    F1 = []
    print("k is " + str(k))

    for training, testing in kf.split(X):
        print("-------------------------- start k
=",i,"--------------------------")
        # Define testing and training row indices
(indicates which rows fall under which dataset)
        print("Train Index: ", training, "\n")
        print("Test Index: ", testing, "\n")
        knn = KNeighborsClassifier(n_neighbors=k)
        X_training_data = X.iloc[training]
        X_testing_data = X.iloc[testing]
        Y_training_data = Y.iloc[training]
        Y_testing_data = Y.iloc[testing]

        #Fit model
        knn.fit(X_training_data, Y_training_data)

        #Determine performance metrics
        class_predict = knn.predict(X_testing_data)


accuracy.append(accuracy_score(Y_testing_data,
class_predict))
```

```
precision.append(accuracy_score(Y_testing_data,
class_predict))

recall.append(precision_score(Y_testing_data,
class_predict))
      F1.append(f1_score(Y_testing_data,
class_predict))

      i += 1
      print("-------------------------- end k
=",i,"--------------------------","\n")
   print("accuracy score is ", mean(accuracy))
   print("precision score is ", mean(precision))
   print("recall score is ", mean(recall))
   print("F1 score is ", mean(F1))
   return mean(accuracy), mean(F1)


# In[141]:


feature_set_1_result = k_fold_knn(1,X,Y)
accuracy_1 = feature_set_1_result[0]
F1_1 = feature_set_1_result[1]


#
#
#
#
# # Feature set 2

# In[142]:


get_ipython().run_line_magic('store', '-r
feature_set_2')


# In[143]:


# Set X as features data
feature_set_2 = feature_set_2.dropna()
X2 = feature_set_2.drop('MIP',axis=1)
X2 = X2.drop('Player',axis=1)

# Set Y = target class
Y2 = feature_set_2.MIP

X = X1
Y = Y1


# ## Scalarize the features

# In[144]:


sc = StandardScaler()
sc.fit(X)
sc_features = sc.transform(X)


# ## Get best k value approach 1: Intuitive method

# In[145]:


error_rate = get_error_rate(X,Y)


# In[146]:


plt.figure(figsize=(10,6))
plt.plot(error_rate, color='blue', linestyle='dashed',
marker='o',
      markerfacecolor='red', markersize=10)
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')


# In[147]:
```

```
plt.plot(error_rate[0:50])
plt.show


# In[148]:


print("Error rate is minimized when k is " +
str(error_rate.index(min(error_rate))+1))


# ## We can tell that the error rate is minimized
when k = 5

# In[149]:


knn = KNeighborsClassifier(n_neighbors=5)


# ## Get best k value approach 2: GridSearchCV
Optimization

# In[150]:


parameter = {'n_neighbors':np.arange(1,40)}
knn_grid_cv= GridSearchCV(knn, parameter,
cv=10, scoring='accuracy')
knn_grid_cv.fit(X,Y)
print("optimal accuracy: " +
str(knn_grid_cv.best_score_))
print("optimal k value: " +
str(knn_grid_cv.best_params_))


# ## k = 10 gives the most optimal accuracy score

# In[151]:


parameter = {'n_neighbors':np.arange(1,40)}
knn_grid_cv= GridSearchCV(knn, parameter,
cv=10, scoring='precision')
knn_grid_cv.fit(X,Y)
print("optimal precision: " +
str(knn_grid_cv.best_score_))
print("optimal k value: " +
str(knn_grid_cv.best_params_))


# ## k = 1 gives the most optimal precision score

# In[152]:


parameter = {'n_neighbors':np.arange(1,40)}
knn_grid_cv= GridSearchCV(knn, parameter,
cv=10, scoring='recall')
knn_grid_cv.fit(X,Y)
print("optimal recall: " +
str(knn_grid_cv.best_score_))
print("optimal k value: " +
str(knn_grid_cv.best_params_))


# ## k = 1 gives the most optimal recall score

# In[153]:


parameter = {'n_neighbors':np.arange(1,40)}
knn_grid_cv= GridSearchCV(knn, parameter,
cv=10, scoring='f1')
knn_grid_cv.fit(X,Y)
print("optimal f1: " +
str(knn_grid_cv.best_score_))
print("optimal k value: " +
str(knn_grid_cv.best_params_))


# ## k = 1 gives the best F1 score, therefore we
use k = 1
```

```
# ## Implement knn using k = 1

# In[154]:


feature_set_2_result = k_fold_knn(1,X,Y)
accuracy_2 = feature_set_2_result[0]
F1_2 = feature_set_2_result[1]


# # Feature set 3

# In[155]:


get_ipython().run_line_magic('store', '-r
feature_set_3')


# In[156]:


# Set X as features data
feature_set_3 = feature_set_3.dropna()
X3 = feature_set_3.drop('MIP',axis=1)
X3 = X3.drop('Player',axis=1)

# Set Y = target class
Y3 = feature_set_3.MIP

X = X3
Y = Y3


# ## Scalarize the features

# In[157]:


sc = StandardScaler()
sc.fit(X)
sc_features = sc.transform(X)
```

```
# ## Get best k value approach 1: Intuitive method

# In[ ]:


error_rate = get_error_rate(X,Y)


# In[ ]:


plt.figure(figsize=(10,6))
plt.plot(error_rate, color='blue', linestyle='dashed',
marker='o',
        markerfacecolor='red', markersize=10)
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')


# In[ ]:


plt.plot(error_rate[0:50])
plt.show


# In[ ]:


print("Error rate is minimized when k is " +
str(error_rate.index(min(error_rate))+1))


# ## We can tell that the error rate is minimized
when k = 5

# ## Get best k value approach 2: GridSearchCV
Optimization

# In[ ]:
```

```
knn = KNeighborsClassifier(n_neighbors=5)
parameter = {'n_neighbors':np.arange(1,40)}
knn_grid_cv= GridSearchCV(knn, parameter,
cv=10, scoring='accuracy')
knn_grid_cv.fit(X,Y)
print("optimal accuracy: " +
str(knn_grid_cv.best_score_))
print("optimal k value: " +
str(knn_grid_cv.best_params_))


# ## k = 10 gives the most optimal accuracy score

# In[ ]:


parameter = {'n_neighbors':np.arange(1,40)}
knn_grid_cv= GridSearchCV(knn, parameter,
cv=10, scoring='precision')
knn_grid_cv.fit(X,Y)
print("optimal precision: " +
str(knn_grid_cv.best_score_))
print("optimal k value: " +
str(knn_grid_cv.best_params_))


# ## k = 1 gives the most optimal precision score

# In[ ]:


parameter = {'n_neighbors':np.arange(1,40)}
knn_grid_cv= GridSearchCV(knn, parameter,
cv=10, scoring='recall')
knn_grid_cv.fit(X,Y)
print("optimal recall: " +
str(knn_grid_cv.best_score_))
print("optimal k value: " +
str(knn_grid_cv.best_params_))


# ## k = 1 gives the most optimal recall score

# In[ ]:


parameter = {'n_neighbors':np.arange(1,40)}
knn_grid_cv= GridSearchCV(knn, parameter,
cv=10, scoring='f1')
knn_grid_cv.fit(X,Y)
print("optimal f1: " +
str(knn_grid_cv.best_score_))
print("optimal k value: " +
str(knn_grid_cv.best_params_))


# ## k = 1 gives the best f1 score, therefore we use
k = 1

# ## Implement knn using k = 1

# In[ ]:


feature_set_3_result = k_fold_knn(1,X,Y)
accuracy_3 = feature_set_3_result[0]
F1_3 = feature_set_3_result[1]


# ## Feature set 3 has the best F1 score!

# # Results:

# In[ ]:


print("Feature set 1 accuracy score: ", accuracy_1)
print("Feature set 2 accuracy score: ", accuracy_2)
print("Feature set 3 accuracy score: ", accuracy_3)


# In[ ]:


print("Feature set 1 F1 score: ", F1_1)
```

```
print("Feature set 2 F1 score: ", F1_2)
print("Feature set 3 F1 score: ", F1_3)
```

7.6 Final Results Code (format is .py)

```
#!/usr/bin/env python
# coding: utf-8

# # Final Results

# Based on the F1 scores, Feature Set 3 did better than any other feature set.
# We calculated the accuracy scores as well. However for this type of dataset it is a not a good score
# to look at to evaluate our model.

# We will now test our best modesl (DT - Feature Set 3) and (LR - Feature Set 3) with the 2018-2019 statistics and analyze the results. Note that this data was not included to train and test the model.

# In[27]:


import pandas as pd
import numpy as np

from sklearn import tree
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn import metrics
from sklearn import datasets
from matplotlib import pyplot as plt
import seaborn as sns
import statistics

# import graphviz
import pydotplus
import io
from scipy import misc

get_ipython().run_line_magic('matplotlib', 'inline')


# In[28]:


get_ipython().run_line_magic('store', '-r master')
get_ipython().run_line_magic('store', '-r feature_set_1')
get_ipython().run_line_magic('store', '-r feature_set_2')
get_ipython().run_line_magic('store', '-r feature_set_3')
get_ipython().run_line_magic('store', '-r last_season_data')


# In[29]:


#Function that generates the decision tree visualizations
def show_tree(tree, features, path):
    f = io.StringIO()
    export_graphviz(tree, out_file=f, feature_names=features)

pydotplus.graph_from_dot_data(f.getvalue()).write_png(path)
    img = plt.imread(path)
    plt.rcParams['figure.figsize'] = (20,20)
    plt.imshow(img)


# In[30]:
```

```
# Initialize the models
dt_model =
DecisionTreeClassifier(min_samples_split=100)
lr_model = LogisticRegression()
```

```
# In[31]:
```

```
# Set X as features data (remove class variable
and player name)
feature_set_3 = feature_set_3.dropna()
X_train =
feature_set_3.drop('MIP',axis=1).drop('Player',
axis=1)
feature_names1 = list(X_train.columns)
#Set Y as target class
Y_train = feature_set_3.MIP
```

```
# In[32]:
```

```
# Set X_test and Y_test as the data from
2018-2019
X_test_names = last_season_data.Player
rows = X_test_names.size
X_test =
last_season_data.drop('MIP',axis=1).drop('Player',
axis=1)
X_test = X_test.dropna()
Y_test = last_season_data.MIP
rows
```

```
# In[33]:
```

```
# Fit DT to training dataset
dt = dt_model.fit(X_train, Y_train)
# Fit LR to training dataset
lr = lr_model.fit(X_train, Y_train)
```

```
# In[34]:
```

```
show_tree(dt,feature_names1,'Decision_Tree_1')
```

```
# In[35]:
```

```
# Calculate predictions
dt_predictions = dt.predict(X_test)
lr_predictions = lr.predict(X_test)
```

```
# In[36]:
```

```
#empty dataframe to store the class predictions
predictions = pd.DataFrame()
predictions['Player'] = last_season_data['Player']
predictions['DT Predictions'] = "Nothing"
predictions['LR Predictions'] = "Nothing"
```

```
# In[37]:
```

```
j = 1
print("Rows:", rows)
for j in range(0,206):
    predictions.iloc[j, 1] = dt_predictions[j]
    predictions.iloc[j, 2] = lr_predictions[j]
```

```
# ## DT Results: 2018-2019 Season
```

```
# In[38]:
```

```
DT_MIPs = predictions[predictions['DT
Predictions']==1].drop('LR Predictions', axis=1)
DT_MIPs


# ## LR Results: 2018-2019 Season

# In[39]:


LR_MIPs = predictions[predictions['LR
Predictions']==1].drop('DT Predictions', axis=1)
LR_MIPs


# In[ ]:




# In[ ]:
```

7.7 Clustering Code (format is .py)

```
#!/usr/bin/env python
# coding: utf-8

# In[26]:


import seaborn as sns
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
from pylab import rcParams
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import plotly.express as px
get_ipython().run_line_magic('matplotlib', 'inline')

# In[27]:


s1819 =
pd.read_csv("player_stats_2018_2019.csv")

# In[28]:


s1718 =
pd.read_csv("player_stats_2017_2018.csv")

# In[29]:


get_ipython().run_line_magic('store', '-r
clustering_data')

# In[30]:


clustering_data.head()

# # Subset 1: True Shooting%, Points,
Rebound+Assist

# In[31]:


TS_Pts_RA =
pd.DataFrame(data=[s1819["TS%"],s1819["PTS"]
,s1819["TRB"]+s1819["AST"]],
index=['TS','PTS',"RA"])

# In[32]:
```

```
TS_Pts_RA = TS_Pts_RA.transpose()

# In[33]:

TS_Pts_RA = TS_Pts_RA.dropna()

# In[34]:

kmeans = KMeans(n_clusters=5)

# In[35]:

kmeans.fit(TS_Pts_RA)

# In[36]:

kmeans.cluster_centers_

# In[37]:

kmeans.labels_

# ## Static 3D graph

# In[38]:

rcParams['figure.figsize'] = 20,20
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
x = TS_Pts_RA.TS
y = TS_Pts_RA.PTS
z = TS_Pts_RA.RA

ax.scatter(x, y, z, c=kmeans.labels_,
cmap='rainbow', s=50, marker='o')

ax.set_xlabel('True Shooting Percentage')
ax.set_ylabel('Points')
ax.set_zlabel('Rebound+Assist')

plt.show()

# ## Interactive 3D graph

# In[39]:

fig = px.scatter_3d(TS_Pts_RA, x='TS', y='PTS',
z='RA',
            color=kmeans.labels_, width=1000,
height=1000, color_continuous_scale='Portland')
fig.update_traces(marker=dict(size=5))

fig.show()

# # Subset 2: BPM,

# In[40]:

clustering_data.head()

# In[41]:

subset_2 =
pd.DataFrame(data=[clustering_data["Delta
```

```
AST%"],clustering_data["Delta
PTS"],clustering_data["Delta TRB"]],
index=['Delta AST','Delta PTS',"Delta TRB"])


# In[42]:


subset_2 = subset_2.transpose()


# In[43]:


subset_2 = subset_2.dropna()


# In[44]:


kmeans = KMeans(n_clusters=5)


# In[45]:


kmeans.fit(subset_2)


# In[46]:


kmeans.cluster_centers_


# In[47]:


kmeans.labels_


# In[48]:


subset_2.head()


# ## Static 3D graph

# In[49]:


rcParams['figure.figsize'] = 20,20
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

x = subset_2["Delta TRB"]
y = subset_2["Delta PTS"]
z = subset_2["Delta AST"]


ax.scatter(x, y, z, c=kmeans.labels_,
cmap='rainbow', s=50, marker='o')

ax.set_xlabel('Delta Assist')
ax.set_ylabel('Delta Points')
ax.set_zlabel('Delta Rebounds')

plt.show()


# ## Interactive 3D graph

# In[50]:


fig = px.scatter_3d(subset_2, x=subset_2["Delta
AST"], y=subset_2["Delta PTS"],
z=subset_2["Delta TRB"],
            color=kmeans.labels_, width=1000,
height=1000, color_continuous_scale='Portland')
fig.update_traces(marker=dict(size=5))

fig.show()
```

In[ ]: