NBA Career Prediction

April 18, 2018

Abstract

Predicting the career success of NBA basketball players has great value to NBA franchises. The earlier these franchises know if a player will help drive their team to success, the earlier they will be able to make profitable decisions. Since these franchises want to make decisions as quickly as possible, a natural question arises. Is it possible to accurately predict career success of NBA players based on their performance during their rookie season (their first season in the NBA)? To answer this question, I obtain player data from stats.nba.com for a random sample of players. I then rigorously defined and justified a quantitative measure for the success of a player's career. I applied the K-Neighbors and Logistic Regression classifiers, then the Random Forest regressor and the Extremely Randomized Trees regressor to the dataset and analyzed the effectiveness of these results at predicting career success. I conclude that while it is possible to obtain better-than-random results, the results cannot be exceptionally good due to the inherit randomness in basketball.

1 Introduction

1.1 The Problem

Predicting the success of NBA athletes is a valuable asset to any NBA franchise. Like most sports teams, NBA teams want to maximize both wins and profit, and therefore, the most valuable players are those that help the team accomplish that goal. Knowing early on in a player's career if that player is likely to bring value to the franchise would allow teams to make smarter trade and contract decisions. The purpose of this project is to predict the success of NBA athletes over the course of their careers based on their rookie seasons.

1.2 Current Research

Statistics-based predictions are widely prevalent in sports, and basketball is no exception. Predictions are commonly made by sports analysts, broadcasting companies, and betting companies. These predictions, however, are mainly focused on the outcomes of specific games, the number of wins a team will have in a particular season, and other team-related questions[1]. When related to the players, the predictions tend to focus on short-term events such as which players will be MVP or how many points a player will score in a game [2][3]. Currently, predictions regarding how players will fare in the long-term range from uncommon to non-existant, and there are no publicly available methods for doing so. Of course, it must be pointed out that it is possible that teams have their own metrics for measuring the future success of a player, but they would likely want to keep that information secret to maintain whatever advantage that information may provide.

2 Data

2.1 Overview

The dataset I will be using is a collection of statistics from a random sample of NBA players for each season of their careers. This dataset was obtained from the statistics page of the official NBA website (stats.nba.com) and can be considered highly reliable because it simply lists recorded statistics for NBA players. For each player and each season, the dataset contains information on statistics such as minutes played per game, points per game, assists per game, field goal percentage, etc.

This dataset is relevant to the project objective because it provides comprehensive, quantitative data for NBA athletes for each season of their careers. This, of course, includes a wide range of statistics on a player's rookie season, and the remainder of the dataset can be used to compute a quantitative measurement for career success. This measurement will be defined in greater detail later. These data form the final dataframe, which contains the features described in Appendix 1 for each player in the randomly sampled set.

2.2 Collection and Cleaning

The collection process was done in a way to minimize the bias present in the data. What follows is a list of the main biases that could skew the dataset and an explanation of how these biases were avoided.

1. The 3-point line was introduced in 1980, and has since become an integral part of the game. Hence, player stats from before the 3-point line was introduced could look much different from player stats after that.

Since the goal of this project is to produce a prediction tool that is applicable today, I decided only to use data on players that played their careers after the 3-point line's impact took effect. So, I only obtained data for players who were active in the 1990 season and later.

2. Teams obviously have different tactics. Some are more defensive-minded and score fewer points, others may place higher value on getting offensive rebounds, etc. As a result, players will have slightly different statistics depending on which team they play for.

The second potential bias is trickier. A player's team clearly has an influence on his stats, and I considered a variety of ways to tackle this. The naive thing to do would be to control based on the average points, etc. that a player gets per game while on that team. However, how do we know that a team average is a result of their tactics and not a result of the poor quality of players on that team, which we would want to keep as part of the model? After considering these type of factors, I concluded that the best way to control for this bias without going way beyond the scope of this project was simply to have players from various teams in the dataset.

3. The average points scored by NBA teams in a game has steadily increased over the years, with the best team in the 2003-04 season scoring on average 104.9 PPG and the best team in the most recent season scoring on average 113.5 PPG[4]. So, a player that scored 30 points per game in 1995 made a relatively higher contribution to their team than a player that scored 30 points per game in 2017.

I handled the final potential bias similarly to how I handled the third one. Like the third model, there are bound to be many intricate dependent relations among the data, so controlling for everything would be difficult to accomplish reasonably. For example, perhaps points per game went up, but blocks per game went down over that same period. By taking players only from the last 25 years or so, rather than from the establishment of the NBA to now, much of this trend will already be eliminated.

This dataset, like most sports data, was already pretty clean and consistent. However, on occasion, a player would play for more than one team in a season. In these cases, the dataset had a row for stats for each team the player was a part of in that season, as well as a column for the player's total stats for that season. Only the total row was kept since the goal is to focus on player stats rather than team stats.

2.3 Defining Success

Before any predictions of career success can be made about a player, career success needs to be defined quantitatively and rigorously. Typically, a player is considered good if his stats are high. So, the question becomes how do we combine a players stats in a way that best represents his quality relative to other players? The NBA has a metric called efficiency that does something similar to this and can be easily computed from the dataset[5]. The formula is as follows:

$$\text{NBA efficiency} = \frac{\text{PTS} + \text{RB} + \text{AST} + \text{STL} + \text{BLK} + \text{FGM} + \text{FTM} - \text{FGA} - \text{FTA} - \text{TO}}{\text{GP}} \tag{1}$$

where the shorthand on the right side of the equation is defined in Appendix 1. There are two things to notice with this metric. First, this computation is done on a per-game basis. Therefore, a player that plays only 3 minutes in a game and scores 3 points will likely end up with a lower efficiency than a player that plays 40 minutes but only scores 10 points, even though the first one scores more points per-minute. Considering that rookies often play less minutes per game, I decided to compare the NBA's efficiency to a similar metric that's per-minute.

$$\text{Per minute efficiency} = \frac{\text{PTS} + \text{RB} + \text{AST} + \text{STL} + \text{BLK} + \text{FGM} + \text{FTM} - \text{FGA} - \text{FTA} - \text{TO}}{\text{MIN}} \tag{2}$$

The second thing to notice with the NBA's efficiency calculation is that it penalizes for field goals and free throws missed as well as turnovers. This could be bad because this would give players that are perhaps more aggressive with their shooting or passing a disadvantage. So, I created two other efficiency metrics (one per-game and one per-minute) that take this into account.

$$\text{No penalty efficiency} = \frac{\text{PTS} + \text{RB} + \text{AST} + \text{STL} + \text{BLK} + \text{FGM} + \text{FTM}}{\text{GP}} \tag{3}$$

$$\text{No penalty per minute efficiency} = \frac{\text{PTS} + \text{RB} + \text{AST} + \text{STL} + \text{BLK} + \text{FGM} + \text{FTM}}{\text{MIN}} \tag{4}$$

It is assumed that these efficiency metrics will map players that are generally considered good, like LeBron James and Kobe Bryant, to higher values, and players that are not as highly regarded to lower values. It also seems reasonable to suppose that the better the efficiency metric is, the more clear this distinction will be. We use this intuition to compare the four efficiency metrics to determine which is the best model to use.

Based on the graphs in Figure 1, we see that all of them manage to put better players like LeBron James and Shaquille O'Neal higher than lesser-known players like Carl Landry and Tiago Splitter. However, the per-minute efficiencies tend to jumble the data more than the per-game efficiencies. The graphs of the per-game efficiencies look fairly similar, and both seem to do a

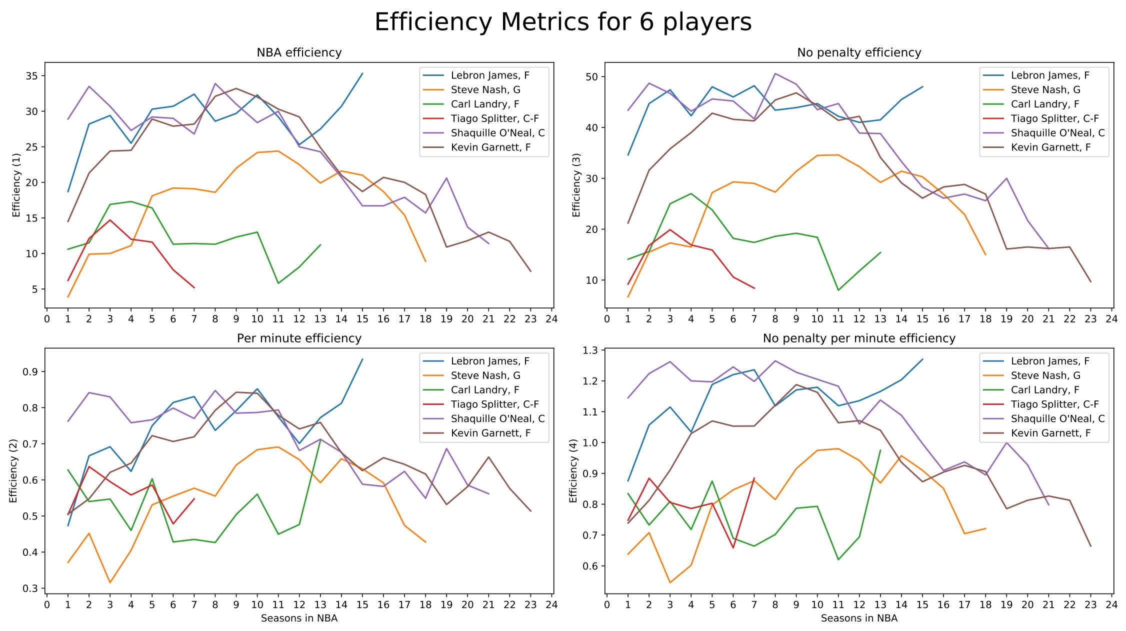

Figure 1: A comparison of the four efficiency metrics across the careers of six players.

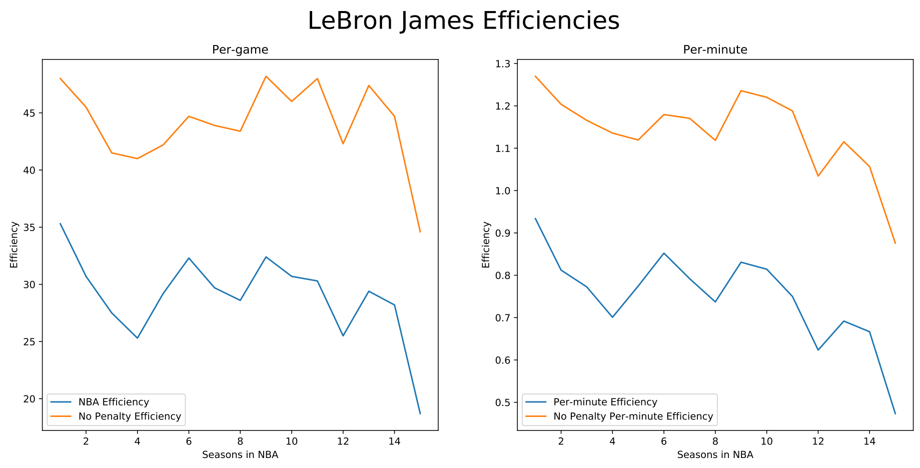

Figure 2: A comparison of the four efficiency metrics for LeBron James.

good job of separating players based on their quality. By comparing the four metrics for one player (see Figure 2), it is also clear that all four metrics are fairly similar in terms of consistency throughout a player's career. As a result, we choose the NBA's efficiency as our efficiency metric because it is the one used by the NBA and there does not appear to be any benefit to modifying it. So, we will define a player's quality (or success) for a given season as their efficiency for that season calculated according to (1) above.

2.4 Career Success

Now that we have defined what success looks like for a given season, we must similarly define what success looks like for a player's career. There are a number of ways that I considered for how to do this. First, I considered simply taking the average of each player's efficiency over each of their seasons in the NBA and calling that their career success. However, as is manifest in Figure 1, often the most talented players play more seasons, and as they age their efficiencies drop off. Thus, their average efficiency for their career would end up lower than it should be to predict their success. The other possibility I considered was to define a player's career success to be the max of their efficiencies in each of their seasons. This also poses a problem, however, because this allows a player to have one high-efficiency outlier season and then be considered extremely successful despite that season not actually reflecting his true success. So, for the purposes of this project, career success is defined to be the average of the top three season efficiencies. This mitigates both of the potential issues with this measurement. Mathematically, we have

$$\text{Career Success} = \frac{\text{EFF}_1 + \text{EFF}_2 + \text{EFF}_3}{3} \tag{5}$$

where EFF_1, EFF_2, EFF_3 are the top three season efficiencies over a player's career.

Using this metric for career efficiency, the five players in the dataset with the highest scores are (in order) Michael Jordan, LeBron James, Hakeem Olajuwon, Charles Barkley, and Shaquille O'Neal. This seems reasonable, so we conclude that this is an effective metric for career efficiency. This metric is approximately normally distributed. The average efficiency is 16.85, with a standard deviation of about 6.3. The lowest efficiency is 3.93 and the highest is 35.47.

2.5 Summary

In summary, I cleaned the data I obtained from the NBA website by combining seasons where players had multiple teams into one row of data, then by removing the team feature entirely. I then engineered a new feature, career success, for each player according to the above definitions. So, taking the kept features for each player's rookie season, and adding the career success feature, I construct a new dataframe with the features defined in Appendix 1.

3 Methods

Finally, I apply a few machine learning methods to my model to predict a player's career success based on the player's rookie season statistics. The first two methods I chose are classifiers, so the career efficiency will have to be modified in a way that converts this to a classification problem. The remaining methods are regressors.

3.1 Logistic Regression

The first method I will use is a simple logistic regression model. This model classifies each data point into one of two categories by minimizing a logistic loss function to fit the model to the data. In the case of this problem, these two categories are "good" and "not good." I chose this method because it reflects a legitimate way this type of model could be applied in a real life situation. Due to the inherit randomness in sports, an NBA franchise may find it more valuable to be able to predict which players will end up in the top 30% or so for their careers rather than be able to predict the precise success level a player will have. By splitting up the data into complementary sets where one set contains players in the top 30% for career success and the other contains the remaining players, this can be converted to a logistic regression problem. The code for doing this, as well as splitting the data for a regression model, is found in Appendix 2.

For the logistic regression model, we use the LogisticRegression class in scikit-learn. An important hyperparameter for this model is the regularization parameter C. A grid search on this parameter is necessary in order to optimize results. After performing this grid search and using the optimal C, we see that the logistic regression model can determine whether a player will be in the top 30% of success for his career based on his rookie data with between 73-78% accuracy. Note that it is significant that this number is higher than 70%. If it were 70%, it would be possible that the model is just predicting no player will be in the top 30%, but since it is higher it must be predicting some are in that top 30%. After looking at the predictions, I verified that it was performing as desired.

3.2 K-Neighbors

The second model I will explore is a K-neighbors model. I chose this model for two reasons. First, it is likely that players that have more successful careers will have similar rookie stats. If a player scores a lot in his rookie season, he's likely to score a lot over his career. So, successful players are probably "close" to other successful players using a distance metric. The other reason I chose this model is that is is a generalization of the real life scenario from the logistic regression model. Perhaps, beyond knowing what players will be in the top 30% and which will not, a franchise wants to predict which will be in the bottom third, the middle third, and the top third. They may seek those who are expected to be in the top third of career success, but will be more likely to take a gamble on a player that is predicted to be in the middle than a player predicted to be at the bottom. Splitting up the target set into more than just two categories for the K-neighbors model allows us to explore this possibility.

For the K-neighbors model, we use the KNeighborsClassifer class in scikit-learn. An important hyperparameter for this model is the number of neighbors to use, k. After performing a grid search on k and using the optimal value, we find that the model can accurately classify players into the bottom, middle, or top third for their careers with about 45% accuracy. This means that though the K-neighbors model is not very well suited to predict career success for players based on their rookie season, it does do better than just guessing. However, does it do better than say a general manager of an NBA team using his intuition to predict the future success of a rookie? Probably not.

3.3 Random Forest Regressor

The next method I chose was a random forest regressor. I decided to use the regressor because the target data is real-valued so it makes sense to try to fit it to a real-valued function. The random

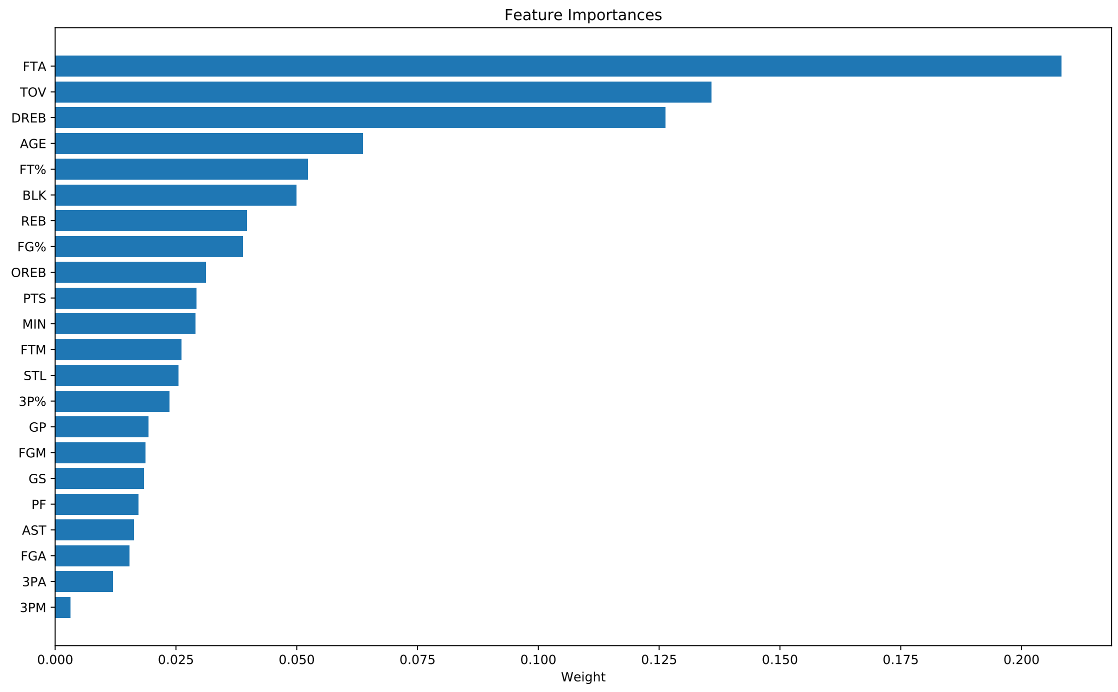

Figure 3: Feature importance for the random forest.

forest is also well suited for this problem because it splits the data on its various features. It's possible that some of the rookie year statistics have better predicting power than others, and this method will account for this. Finally, since it is an aggregate of many decision trees, it avoids overfitting better than a single tree would.

For the random forest regressor model, we use the RandomForestRegressor class in scikit-learn. Some important hyperparameters for this model are max depth and number of estimators. The max depth corresponds to how many levels of the tree will be used, and the number of estimators corresponds to the number of trees to use in the forest. We perform a grid search on both of these features to obtain the best model. This model, applied to the data, achieves an R^2 value of about .31. This is much lower than we would hope considering the ideal value is 1.

The other thing this model gives us is the relative importance of each of the features in predicting future success. By looking at the feature importances, we can determine which features were most useful in splitting the data. This is shown in Figure 3. Based on this, the most useful rookie statistics for predicting career success for this model were, interestingly, their number of free throws attempted and turnovers per game, while the number of three pointers they made and attempted had very little predictive power.

3.4 Extra Trees

After analyzing the results of the random forest regressor, it is clear that there could still be much improvement to this model. One similar model that may perform better is an extremely randomized trees regressor. This model is similar to the random forest regressor in that it selects features to split on. However, rather than finding the optimal split values, it selects a few split values randomly, then picks the split value to be the one of these that performs best. This commonly lowers variance, but increases bias[6]. This has the potential to improve upon the random forest regressor.

Player	Logistic Regression	K-Neighbors	Random Forest	Extra Trees
LeBron James	Top 30%	Top third	30.9	31.1
Kobe Bryant	Bottom 70%	Middle third	21.8	22.3
Adam Keefe	Bottom 70%	Top third	13.2	15.4
Eric Montross	Bottom 70%	Bottom third	13.6	16.8
Charles Barkley	Top 30%	Middle third	27.3	26.3

Table 1: Case study results.

Once again, for this model we use the ExtraTreesRegressor class from scikit-learn. The same hyperparameters should be considered for this model as for the random forest regression model. After performing a grid search on these parameters to obtain the best model, we obtain an R^2 value of about .34. This is marginally better than the random forest regressor, but the models perform effectively the same. In addition, the feature importances for this model are similar to those for the random forest regressor.

4 Case Study

After obtaining our best classifiers and regressors for each of the above methods, it is natural to ask how these methods could be used in a real life scenario. To simulate a real life situation, we take five players and pretend that we have only their data for their rookie season. We use each of the four models to predict their career success, then compare it to our expectations. The results of this process are summarized in Table 1. These results are definitely better than random, but are far from ideal. Common basketball knowledge would rank LeBron James, Kobe Bryant, and Charles Barkley much higher than Adam Keefe and Eric Montross. From the results in the table, we see that the logistic regression model, and the two regressors performed fairly well, though they seem to predict Kobe Bryant lower than he probably should be. The K-neighbors model, however, was far more random, and the results were inconsistent. Thus we confirm our previous conclusions that, while we can obtain better-than-random accuracy, it is difficult to obtain highly accurate results.

5 Conclusion

From the logistic regression and K-neighbors models, it is clear that the model can get better-than-random accuracy when attempting to gauge what percentile range a player will be in, but cannot get a whole lot better. The random forest regressor gives information about which variables are most useful in predicting future success, and the results are surprising. However, the random forest regressor does not perform exceptionally well. To remedy this, a new method, the extremely randomized trees regressor, was introduced. This performs slightly better than the random forest regressor, but not significantly. These results sense for the real world application. Basketball, like all sports, has a lot of inherit randomness. Players could get injured or traded in a way that positively or negatively affects their career statistics. So, it seems reasonable to suppose that while it may be possible to get some degree of predictive power using these methods, it can never be completely accurate.

6 References

[1] "NBA Basketball Lines." Odds Shark, www.oddsshark.com/nba.

[2] "NBA All-Star Game Odds Spread MVP Betting Picks 2018." Action Network, 18 Feb. 2018, www.actionnetwork.com/nba/article/all-star-game-bets-we-like-and-why.

[3] Campbell, Stephen. "Odds on Which NBA Player Will Score the Most Points Tonight." Odds Shark, OddsShark, 19 Apr. 2017, www.oddsshark.com/nba/odds-which-nba-player-will-score-most-points-april-19.

[4] "NBA Team Points per Game." NBA Stats - NBA Team Points per Game on TeamRankings.com, www.teamrankings.com/nba/stat/points-per-game.

[5] "Glossary." Hoopsstats.com - NBA Fantasy Basketball Stats - Glossary, www.hoopsstats.com/nbafolder/style/glossary.html.

[6] "1.11. Ensemble Methods." 1.11. Ensemble Methods - Scikit-Learn 0.19.1 Documentation, scikit-learn.org/stable/modules/ensemble.html.

7 Appendix 1 - Feature Definitions

1. AGE --- The player's age his rookie season
2. GP --- The number of games a player played in his rookie season
3. GS --- The number of games a player started in his rookie season
4. MIN --- The average number of minutes a player played per game for his rookie season
5. PTS --- The average number of points a player scored per game for his rookie season
6. FGM --- The average number of field goals a player made per game for his rookie season
7. FGA --- The average number of field goals a player attempted per game for his rookie season
8. FG% --- The average field goal percentage a player had for his rookie season
9. 3PM --- The average number of 3-pointers a player made per game for his rookie season
10. 3PA --- The average number of 3-pointers a player attempted per game for his rookie season
11. 3P% --- The average 3-point percentage a player had for his rookie season
12. FTM --- The average number of free throws a player made per game for his rookie season
13. FTA --- The average number of free throws a player attempted per game for his rookie season
14. FT% --- The average free throw percentage a player had for his rookie season
15. OREB --- The average number of offensive rebounds a player had per game for his rookie season
16. DREB --- The average number of defensive rebounds a player had per game for his rookie season
17. REB --- The average number of rebounds a player had per game for his rookie season
18. AST --- The average number of assists a player made per game for his rookie season
19. STL --- The average number of steals a player made per game for his rookie season
20. BLK --- The average number of blocks a player made per game for his rookie season
21. TOV --- The average number of turnovers a player made per game for his rookie season
22. PF --- The average number of personal fouls a player committed per game for his rookie season
23. EFF --- The efficiency rating a player had for his career (described in more detail in the section on defining success)

8 Appendix 2 - Code to Acquire Data

```
def get_train_test_data(players, test_size=.3, classifier=False, percentile=↩
    None):
    """
    Get the train and test data for either a classifier or
    a regressor.
    Parameters:
        players (DataFrame): The dataframe to split.
        test_size (float): The test size to use for the split.
        classifier (bool): Returns y as array of floats if False,
            as an array of ints (for the categories) if True.
        percentile (float or list): The percentile division. Only necessary
            if classifier=True. If a float, splits y into 0's and 1's
            so that the 0's represent the bottom percentile of that data.
            If a list of floats, splits the data into groups based on the list↩
                .
            For example, if percentile = [.25, .75], then y will be split into↩
                 0's,
            1's and 2's, where 0 is the bottom 25%, 1 is between 25% and 75%, ↩
                and
            2 is above 75%.
    Returns:
        X_train (ndarray): The training data. All but the EFF column of the ↩
            dataframe
            are included.
        X_test (ndarray): The test data. Same format as above.
        y_train (ndarray): The target for the training set.
            Could be an array of floats or ints.
        y_test (ndarray): The target for the test set. Same format as above.
    """
    X = players.drop('EFF', axis=1)
    y = players['EFF']

    # If a regressor is wanted
    if classifier == False:
        return train_test_split(X,y, test_size=test_size)

    # If splitting into two categories
    if type(percentile) == np.float:
        cutoff = sorted(y)[int((percentile)*len(y))]
        # Change y to zeros if below cutoff, ones if above
        y = (y > cutoff).astype(int)
        return train_test_split(X,y, test_size=test_size)

    # If splitting into multiple categories
    percentile = [0] + percentile + [1]
    ycopy = y.copy()
    for i in range(len(percentile) - 1):
        cutoff1 = sorted(ycopy)[int((percentile[i])*len(ycopy))]
        try:
            cutoff2 = sorted(ycopy)[int((percentile[i+1])*len(ycopy))]
        except IndexError:
            cutoff2 = sorted(ycopy)[-1] + 1
        y[(ycopy < cutoff2)*(ycopy >= cutoff1)] = i
    return train_test_split(X,y, test_size=test_size)
```