Making an Expected Points Added Model for NBA Games

Jackson C. Hampton

Submitted under the supervision of Mike Weimerskirch to the University Honors Program at the University of Minnesota-Twin Cities in partial fulfillment of the requirements for the degree of Bachelor Science, summa cum laude in Mathematics.

Readers: Daniel Boley, Andy Hardt

May 5th, 2021

Abstract

In this paper we will introduce a new basketball statistic called Normalized Expected Points Added (Normalized EPA) per game that will help evaluate NBA players in a more accurate way than points and assists currently are able. This statistic follows the same ideas as Cervone et. al. [4] had with their Expected Possession Value statistic but has been formulated in a different way. Normalized EPA/Game uses an XGBoost framework (see Section 3) to get the possibility a team scores zero, one, two, or three points in a possession and uses that to get the Expected Points at any given moment. From there, EPA can be calculated by subtracting the current event from the next and it can be normalized by position to get Normalized EPA/Game. The goal of this statistic is to use player tracking data to better understand the offensive contributions of NBA players.

Section 1 - Introduction

An NBA box score has a lot of information in it. One can find statistics like how many shots a player took, their personal fouls, and their turnovers, among other things, but what do most people really look at? Because people move so quickly and only want the most pertinent information, many people just look at the “traditional” stats like points and assists to see how well a player played that game. While this is not a terrible way to look at things since good players will generally have high totals in these columns (think Kevin Durant or Chris Paul), it is not going to be the best way to evaluate these players. There are some plays that don’t fairly distribute the value added by each player by giving one an assist and the other two or three points.

For example, on December 1, 2015 the Cleveland Cavaliers played the Washington Wizards and with 36 seconds left in the first quarter, LeBron James threw a beautiful pass to Richard Jefferson that went between two defenders and set him up for an easy dunk. On that play, Jefferson gets the credit for two points and LeBron gets the assist but LeBron did most of the work. Conversely, there are many plays that are the opposite. One example is in the 2015 Eastern Conference Finals LeBron gets the assist on Kyrie Irving's made shot on the first possession. All LeBron did was toss the ball to Kyrie who was about five feet behind the three point line on the left side of the court. Kyrie then beat his defender with a crossover and then made a floating jump shot over a defender to get the two points. In that play, Kyrie should get most of the credit since he did most of the work to get a good shot and make the shot. LeBron did not really do much other than give the ball right back to Kyrie after it was passed to him.

That is where Expected Points Added (EPA) comes in. With our Expected Points model, you can get the predicted amount of points scored at any given point in the possession. This makes it so an observer can tell who really contributed the most to the points gained on the possession. If a player makes a great pass to help someone else get wide open for a shot, they get credit for that. If a player makes a tough shot with the defender in his face, they get credit for that. To get the EPA for a player on a possession, all you need to do is take the Expected Points when they got the ball and subtract that from the Expected Points when a different player received their pass. Or if that player shot the ball, you subtract it from the amount of points made on their shot. If you do that over the course of every possession of the game, you can get the total Expected Points Added for each player during the game.

Inspiration for this kind of model was taken from the idea of EPA in football that incorporates the field position, time left in the game, down, and distance to give the Expected Points for a team on that drive. Expected Points (and Expected Points Added) was first made popular in The Hidden Game of Football in the late 1980s and was brought back into the limelight by Brian Burke. ESPN then popularized it further by incorporating it in their Total QBR statistic which helped it become more mainstream (Pattani [8]). The final recent development was made by Ben Baldwin and Sebastian Carl [1] with their R package NFLFastR that included an open source EPA model. To be specific, Baldwin and Carl also used an XGBoost model (see Section 3) to get the Expected Points but some of their features also included roof type, era (year of game played), and timeouts remaining for each team. The only problem with bringing this idea to basketball is that football is discrete whereas basketball is not. To circumvent this problem, as stated above, each “play” was the time a specific player has the ball. This makes it so it is easy to subtract the Expected Points from “play” to “play” and get the EPA for each player over the course of the game. And with that, we were able to make an EPA model for the NBA.

Overall, this model is one that is fairly stable, even over fairly small measures of time with an r^2 value of 0.355 when comparing the first and second halves of the dataset. While it is stable, it does not really correlate with Player Efficiency Rating (PER) or Offensive Box Plus Minus (OBPM) other than a small amount with the point guard position group. This is understandable since, while both of those statistics are fairly advanced, neither uses player tracking technology. Despite the lack of correlation, many All-Stars were at or near the top of their positions in Normalized EPA/Game list with

players like Goran Dragic, LeBron James, Kristaps Porzingis, and DeAndre Jordan leading the way. This was especially evident with point guards since Damien Lillard, Derrick Rose, and Isaiah Thomas were in the top five.

With this statistic, it can be seen that guards tend to have higher values (and a larger variance) than forwards. This could show that guards may be more valuable than previously thought. Forwards and players that shoot frequently obviously have value, but it is possible that NBA evaluators have not placed enough emphasis on having at least one elite passer on the team as this statistic shows that is very important. It would be very interesting to see this studied further in another context.

Section 2 - Cleaning the data

To make the model, the SportVu dataset from the 2015-2016 NBA season was used. Every 0.04 seconds a measurement was taken that had the locations and timestamps for the ten players and ball. But there was still a need to create a column for the speed and angle at which each player (and the ball) is moving. In total, there are about nine million rows of data with the columns mentioned above plus the height for the ball and the player/team labels. When cleaning the data (i.e. getting the data ready for input into the model), we took inspiration from the 2019 NFL Big Data Bowl winner. In that competition, participants tried to predict how many yards a running back would get based on the locations and velocities of the 22 players on the field when the running back received the ball. The winning model by Philipp Singer and Dmitry Gordeev only used the relative location and speed values of the players with respect to the running back. Thus, the implementation was something very similar to this for our models. To do this, we just took the distance traveled between two events and divided it by the amount

of time between events. Due to the ball going out of bounds (and thus some events not being continuous), there was a need to filter out any speeds that were faster than 27.8 miles/hour which is the speed of Usain Bolt. Obviously no player is going to be going that fast so their new speed was manually set as 0 mph since it is most likely due to a discontinuous event from the ball going out of bounds. After that, the angle at which a player was running from the previous to current event was easy to obtain. The relative speed between the other nine players and the ball handler was not used at first in the model, so the actual locations/velocities of all the players was what was implemented in the model. In future implementations, this could potentially help improve the accuracy of the model. There was also a need to label the exact shot times so code from Seward [9] was used to do this.

The last step in the data cleaning process was to get the label for the dataset. In other words, create a column that shows how many points were scored on the possession. This was fairly easy for all aspects except when it came to free throws. We ended up just making it the amount of points scored on the free throws, but another intriguing option is to make it the expected number of free throws made. In other words, the points scored on the possession would be assigned to be the amount of free throws attempted multiplied by the average free throw percentage of the NBA in 2015. One reason against doing this was that when doing the XGBoost model, it would have more outputs to give a probability of, reducing the simplicity of the current model that only outputs percent chances of zero, one, two, or three points. Another reason is that the average free throw percentage is different by how many free throws you shoot. In this dataset, the average FT% when shooting three (fouled behind the three point line while

shooting) was 84.4%, when shooting two was 75.7%, and when shooting one was 72.8%. This can most likely be explained by the fact that better shooters (more generally, guards) will predominantly be the ones that will get fouled when shooting a three so their FT% will be higher than the worse shooters that get fouled near the basket. Additionally, NBA players are very good at correcting themselves. NBA players make the second free throw approximately 5% more of the time than they make the first one. This is most likely explained by the fact that these players have shot so many free throws in their life so, if they miss the first shot, they know what adjustments to make when shooting the second. Conversely, if they make the first, they are able to approach the second shot in the same way.

A different way to approach this problem was to just multiply the amount of free throws by the free throw percentage of the specific shooter. This would have helped reward passers and could stabilize the metric (since players are more likely to play aggressive and foul bad free throw shooters like Shaquille O'Neal instead of great ones like Stephen Curry. The reason this was not implemented is because it is not very feasible with the XGBoost model. As stated before, an XGBoost model will output the probability for the selected potential outputs. If the implementation was the expected chance to make a free throw for every player, there would be hundreds of outputs with almost all of them barely happening in the training data. In other words, there would be many individual outputs that appear only once or twice in the data that would be used to train the XGBoost model. It is even possible that in the testing data (the data used to make sure the model is well calibrated and not overfit) will have few to none of these outputs. This would easily lead to overfitting due to the very small amount of data for

those outputs. So, we decided against it and used the original idea of just using the amount of points scored on the free throws. Once these labels were done, the data was ready for input.

One thing that is a common theme in the explanation of this model is the fact that there is a need to avoid overfitting. What overfitting is, to simplify things, is when you overtrain on your data and read too much into each individual data point instead of looking at the general trend. This can be done in many ways. One way is by going through too many iterations/rounds when training your model. This makes the model pick up on little nuances that should not matter as much. Another way is by making something as a super high order model that could be modeled as a linear (or even quadratic) model. This would cause your output to very closely follow each individual data point rather than the overall trend of the data. An example of this would be if you were making a model on when to select a quarterback in the NFL draft. A logical model would tell you that the earlier you select a quarterback, the higher the probability of success is and a linear model would most likely capture this. But say you wanted to model it with a 25th order equation, you might find that it is “very smart” to take a quarterback at pick 199. This is not because pick 199 is somehow the best spot to draft a quarterback, it is just that one of the greatest quarterbacks of all time, Tom Brady, was selected at that spot in the 2000 draft so an overfit model would weigh that datapoint much more than it realistically should.

Something else considered was just not including the possession that ended in free throws in the training data. Theoretically, this would make it so the model isn’t trying to “predict fouls” and could simplify the process. In addition, it would still reward the

good free throw shooters after the model is actually trained. In reality, even after tuning the parameters (details to come later) to combat overfitting, the model still ended up being just that. The model that included free throws in the training data ended up being more accurate and less overfit than this model. An example of this is in Figure 1 below from the first game where it is very obvious that the model is overfit from the extreme changes in EP. Additionally, as shown in Figure 3, the model used does not really predict free throws that often. In the first minute of the game, there was only one moment where the expected chance of one point was non-negligible. This is most likely when the ball gets near the basket and players are more likely to get fouled when shooting or dunking.

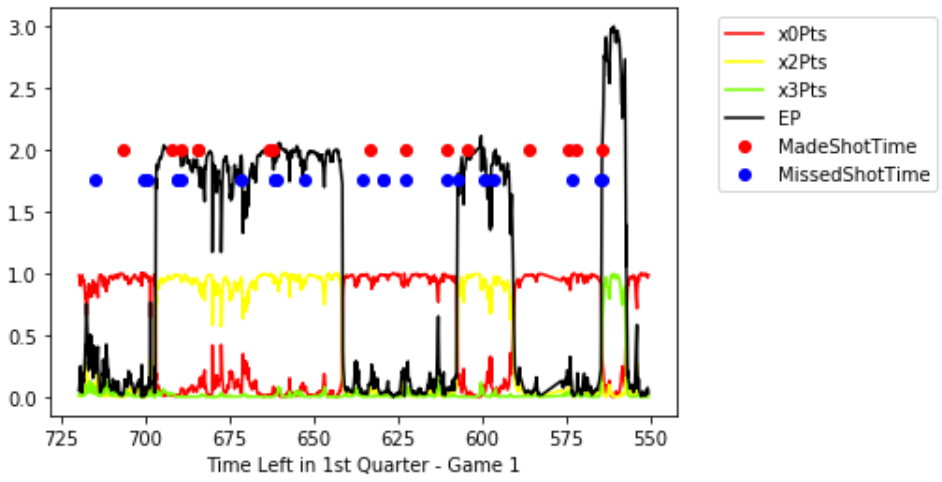

Figure 1: Overfitted model that included free throws in training data

At this point, the data was ready to be input into the XGBoost model. The inputs were the x and y locations, speed, and angle of movement of every player and the ball, plus the current height of the ball.

Section 3 - XGBoost Model

We decided to implement an XGBoost model as one of the model types to test out. This is because it is fairly easy to code up and efficient when training yet still provides very accurate results. XGBoost is an open-source software library that works with structured or tabular data and uses gradient boosted decision trees which work at a very fast rate. It works quickly because of the parallelization of the tree construction during training that uses all of your CPU cores (Brownlee [2]). Thus, unlike other models, it is able to use all of the processing power of your computer when making and tuning (also known as training) the model. With the sheer amount of data in use, quick and efficient training is extremely important so choosing XGBoost was an easy decision to help save time. Additionally, the parameters are very easy to tune when training/validating the data.

For the training and testing, a 75%/25% split was used and the data was shuffled to help reduce overfitting. The inputs used were: the x and y locations, speed, and angle of movement of every player and the ball, plus the current height of the ball to make the model give us an output of the percent chance the offensive team would score zero, one, two, or three points. There was also a label for each row of data used that stated how many points the offensive team scored on that possession. This label helps the model during training to continually improve its accuracy. When it comes to the parameters, Table 1 explains each one that was manually tuned during training. Tuning the parameters is a very important step because, after cleaning the data, choosing the

parameters for the model is all that is left before analyzing the results. Tuning them helps get the most accurate model possible.

Parameter	**Value Used**	**Reasoning**
Nrounds - number of rounds (or iterations) done during training	560	A value near 500 had worked well in past XGBoost models we had made and it worked well in the initial tests so it was kept. By saying it worked well, the output did not seem to be overfitted but it was still able to classify the various events in the game and give a varied amount of outputs. If the number of nrounds was too small, it is possible that the model would not be able to correctly differentiate between different game states so the Expected Point values would have a fairly high error. Conversely, if there is too high a number of nrounds, then the model is more likely to overfit. Through a couple tests on a smaller dataset, we found that 560 was a good compromise for the model.
Training Objective - what kind of output will the model have	multi:softprob	What this means is that for each row, the model will give a probability for each of the classes of the true outputs. Because there were four classes of true outputs (0, 1, 2, or 3 points on the even), each row would get four probabilities that would add up to 1. A different option would be to use multi:softmax for the training objective but that would just output the output class with the highest probability. This would make the Expected Points much more inaccurate since it's

		so discrete and a player's impact might not be fully shown. Using multi:softprob makes the Expected Points much smoother and is much more accurate especially when the dataset of games is not that large.
Booster - which booster to use (either gbtree or gblinear)	gbtree	We chose gbtree because gbtree would give a more accurate output since it uses tree based models whereas gblinear uses linear functions. Our model is not inherently linear structurally so gblinear is not super well suited for it. For this type of classification, gbtree will give better results ("XGBoost Parameters" [11]).
Eta - learning rate	0.2	Normally, when using XGBoost models, one likes to have eta be set to a smaller number like 0.01 but for this model, we had to use 0.2 since the sheer amount of data was so large and a number like 0.01 would increase the runtime significantly. The value of 0.2 still can help us get accurate results without taking a very long time.
Max-depth - maximum depth of a tree in the model	4	It was set to a smaller number than the default to simplify the model and make it less likely to overfit.
Min_child_weight - minimum sum of instance weight needed for a child tree to be formed	6	We set this higher than the default to again help the algorithm be more conservative.

Table 1: Parameters tuned for our model

Once the parameters were set, the model could be trained. As stated earlier, 75% of the data was used to train and 25% was saved to test it. After training the model and running data through it, the output is an array of four values for each row. That array was in the form of [expected percentage of 0 points,x1Points,x2Points,x3Points]. We could then take those values and get the Expected Points (EP) for the possession by using the equation.

$$EP = 0 * x0Points + 1 * x1Points + 2 * x2Points + 3 * x3Points$$

The actual mathematics behind how the XGBoost model works is as follows. It is an ensemble learning method (Sundaram [10]). This means that it is a single model that gives a solution from output aggregated from multiple different models. Aggregation is well known as a smart way to make a model due to its increased accuracy. It relies less on one specific model which could have a higher error rate in certain cases and uses the “wisdom of the group” to more accurately classify the results. XGBoost uses boosting to make multiple decision trees that will help make the final classification.

To simplify what decision trees are, they are models that give you an output based on every possible combination of input. Based on the input data, there are “splits” in the tree (called decision nodes) that classify the output for different inputs. For example, one could make a deterministic decision tree about whether or not a child is going to play basketball outside in the summer. In this, the first split may be on whether or not it is raining. If it is, then the child will not play outside. If it isn’t then the second split may be on if the temperature is above 50 degrees. If it is, then the child will play outside. If it isn’t, then they will not. Each decision node will lead to you taking one of the two branches until you hit the end of the tree and get the predicted output.

Unlike the deterministic decision tree which is based specifically on the decisions one makes, there are also probabilistic decision trees. These trees still have the splits on the decision nodes like the deterministic tree, but these trees will give the probability of a certain event happening at any given time. One example of this is in checkers. Each game begins as shown below in Figure 2. Black will move first and, before moving, will have approximately a 50% chance of winning. Now at this point, there would be a split on a decision node. Black can make seven moves to start this game: A3 to B4, C3 to B4, C3 to D4, E3 to D4, E3 to F4, G3 to F4, or G3 to H4. Each of these decisions would lead to a different probability that black will win the game. A common strategy in checkers is to try and make a “pyramid” structure with your pieces at the start of the game. Thus, C3 to D4 would be a good move and may increase your probability to win by 2%. Conversely, moving G3 to H4 would be considered a bad move and may decrease your probability of winning by 1%. Something like this can be done for every move in a checkers game until the end is reached. At that point you would have your probabilistic decision tree. This type is more similar to the EP model.

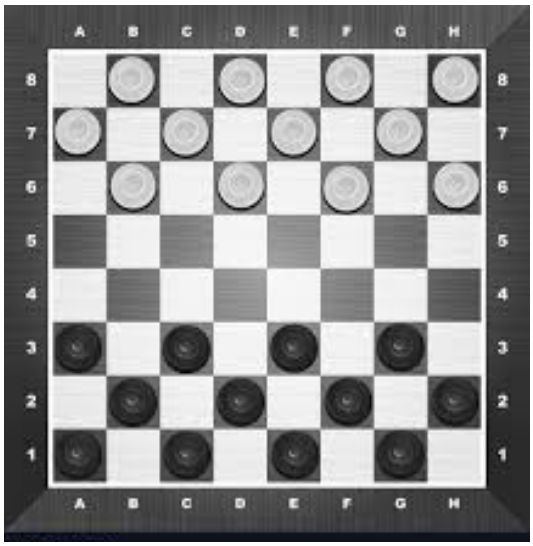

Figure 2: Beginning of checkers game

In the XGBoost model, the decision trees are built in a sequence so that each later tree can reduce the error of the earlier trees. The boosted trees learn from the previous ones and the residual error gets updated. Thus, each tree contributes a little more information to the final output.

Boosting uses trees that may have fewer splits than other decision tree models. These smaller trees are not very deep (in terms of the levels of decisions) but are easily interpretable. If there are many trees in an algorithm, it is very possible that the final model could be overfitted. For this reason, it is necessary to exercise caution when choosing the parameters to avoid this.

In the boosting ensemble technique, the model proceeds as follows. It creates an initial model which we will call M_0. This predicts the target variable y and will have a residual ($y - M_0$). Then, a new model R_0 is fit to the residuals of that previous model. Once that is completed, M_0 and R_0 are combined to give M_1 which is the boosted version of M_0. This new model will have a lower error (based on whatever error function is specified) and the process can be repeated. In equations, the process is below.

$$M_1(x) \Leftarrow M_0(x) + R_1(x)$$

$$M_2(x) \Leftarrow M_1(x) + R_2(x)$$

$$\vdots$$

$$M_n(x) \Leftarrow M_{n-1}(x) + R_n(x)$$

One can see that the model continuously creates new models until it gets to n iterations (which is the amount of nrounds specified in the parameters). Each iteration improves the residuals until it gets to the final step. This gives a well calibrated model that can predict the output fairly accurately. Additionally, because precautions were

taken along the way when tuning the parameters, it should mostly likely not be too overfit.

In this model, there are a lot of possibilities to where the decision tree decision points would be. One place that most likely is a decision point is how far away the closest defender is to the ball handler. If the closest defender is within three feet, then the EP is most likely going to be fairly low. If he is farther away, then the EP will most likely be higher. Another feature that will undoubtedly have importance is how far away from the basket a player is, with closer obviously correlating to higher EPs (except once a player is behind the three point line and there is a discontinuous jump in EP). This "thinking" in creating the decision trees is to always minimize the error. This way the final model will be as accurate as possible.

An example of how the Expected Points change throughout the game can be seen below. You will see in Figure 3 below that NBA players are generally good at shooting when their Expected Points is high.

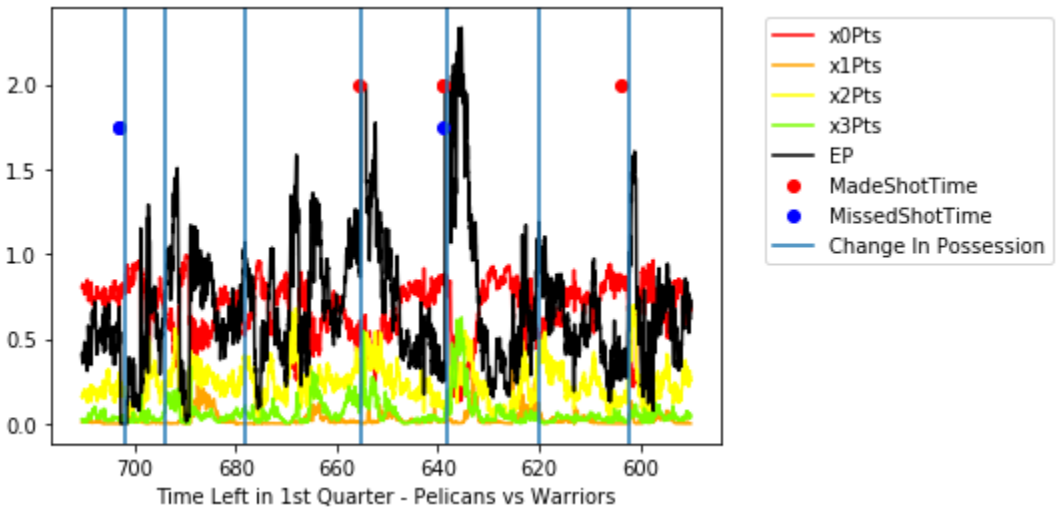

Figure 3: Expected Points during the first two minutes of a Pelicans-Warriors game

Section 4 - Obtaining Expected Points Added

After training the model and running the entire dataset on it, we were just about ready to get the Expected Points Added for the players. First, it was needed to find who was closest to the ball at any given point. This player would generally be the person who is labeled the ball handler. The only time this person would not be labeled the ball handler is if they were on defense, the ball was greater than four feet away from them, or the ball was higher than nine feet in the air. The reasoning behind this is because generally if the ball is greater than nine feet in the air, then it is most likely a shot or a skip pass that is intended to go over some of the people on the court. Additionally, if the ball is more than four feet away from someone then it would be out of their possession (since no one had a wingspan of eight feet in the NBA in that season). Thus, if no player is within four feet of the ball in the x-y plane and the ball is greater than nine feet in the air, the player that would be labeled the ball handler would be the person who last was the ball handler. This makes it so that person gets the full Expected Points Added from the throw of the pass to the catch of the ball from the second player. At this point, it was possible to calculate the EPA at each row and attribute it to a player by subtracting the EP of the next row from the current row.

Another stipulation that was included was that if the possession changed from one team to another, the EPA was 0. This statistic isn't made to measure defensive performance, solely offensive performance, so it would be counterintuitive to have a positive (or negative) EPA from a steal on defense. Lastly, whenever a player shot the ball, the EP was then "frozen" while the ball was in the air until it came down for the rebound or went in the hoop. The EPA for that player would then just end up being

whatever the EP was when he got the ball subtracted from whatever he ended up scoring on the shot (0 for miss, 2 or 3 for make).

To see some examples of the Expected Points at various moments in a game, one can look at the game between the New Orleans Pelicans and Golden State Warriors on October 27, 2015 from the dataset. With 11:17 left in the first quarter, the court looked like what you can see in Figure 4 (code to make figure from Linou [7]). At that moment, the Pelicans had an Expected Points of approximately 0.9 due to the fact that they were not really threatening anywhere on the court. Every offensive player is fairly well covered by the defense. Then approximately six seconds later at 11:11, there is an Expected Points of 1.4 which means the Pelicans increased their Expected Points by approximately 0.5 points in those 6 seconds. Logically, it makes sense that their Expected Points increased since Kendrick Perkins has the ball in the post near the basket with a very good chance to score (as seen in Figure 5).

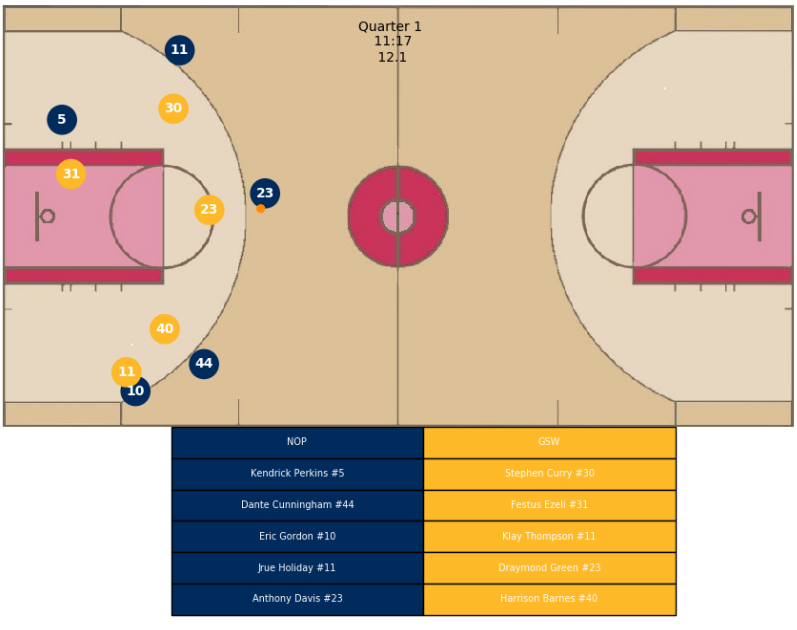

Figure 4: Pelicans vs. Warriors with 11:17 left in the 1st quarter

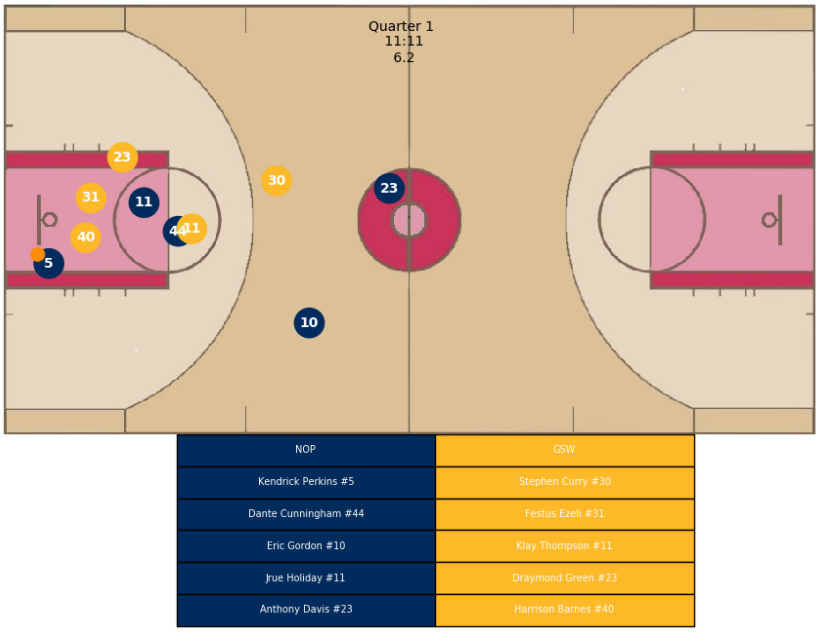

Figure 5: Pelicans vs. Warriors with 11:11 left in the 1st quarter

In this same game, Table 2 shows the final EPA results from all the individual players. One can notice that Steph Curry easily has the most EPA for this game with 17.28 and Anthony Davis led the Pelicans with 10.43 for the game. Intuitively this makes sense! Curry and Davis both made the NBA All-Star Game that season and were the bona fide stars of their teams. They also both led their teams in scoring that day while Curry also led his team in assists.

Name	EPA
Stephen Curry	17.28
Anthony Davis	10.43
Toney Douglas	9.82
Shaun Livingston	8.78
Ryan Anderson	2.07
Andre Iguodala	1.19
Alexis Ajinca	1.17
Kendrick Perkins	0.04
Dante Cunningham	-0.09
Luke Babbitt	-1.04
Alonzo Gee	-1.14
Leandro Barbosa	-1.38
Festus Ezeli	-2.07
Brandon Rush	-2.61
Klay Thompson	-2.77
Marreese Speights	-3.26
Eric Gordon	-4.74
Harrison Barnes	-6.09
Ish Smith	-8.96
Jrue Holiday	-9.06
Draymond Green	-12.39

Table 2: Individual player EPA for Pelicans vs. Warriors

Generally, from looking at the various examples of plays and model outputs for Expected Points, one will see that the higher Expected Points tend to result from having the ball near the basket or wide open behind the three point line (and to a lesser extent when open in the mid-range). This also makes sense! Any coach would tell you that the most valuable shots are ones near the basket and open threes. The NBA is even seeing a transition where these shots are “killing” the mid-range shot. Figure 6 from Kirk Goldsberry [5] below shows the most common shot locations as of March 24th in the

2020-2021 NBA season. It is very easy to see that the midrange is becoming increasingly unpopular compared to the more efficient three point shot and area near the basket.

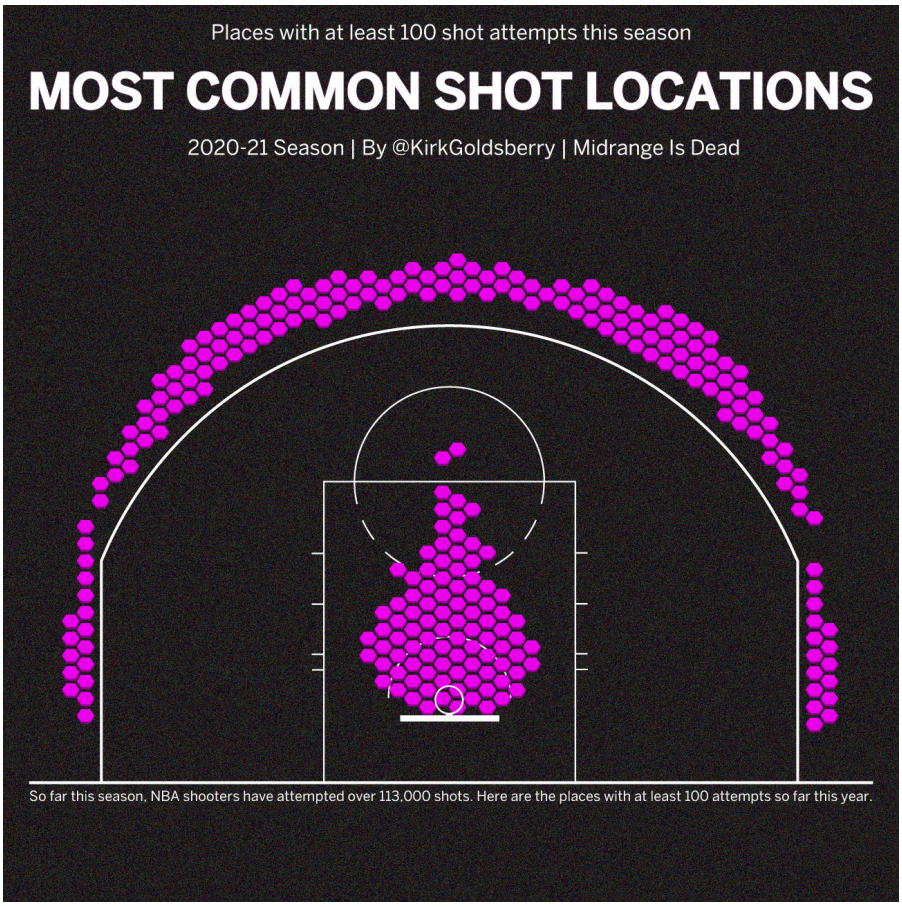

Figure 6: Most Common Shot Locations in the 2020-2021 NBA Season

Section 5 - Results

Once that whole process was complete, we were able to get the final EPA tally for every game and player. One thing that was noticed was that there seemed to have a bit of a bias towards the point guards in the sample. We are not 100% sure why exactly that bias happens but our intuition is that it could possibly happen as they bring the ball in from half court to start generating their offense. For that reason, when looking at the results it was normalized based on the position the player is listed at. All that was

needed to do this was get the average EPA/minute for each position. Then, for each player and their position, that average was multiplied by the amount of minutes played and subtracted that from their previous total. This seems to help stabilize the results and give a better understanding of what players really were exceptional at moving the ball and getting it to good positions. When looking at the top players at each position by normalized EPA, we get the following interesting results in Table 3.

	Name	Pos	MPG	GamesPlayed	NormalizedEPA	NormalizedEPAperGame
0	Goran Dragic	SG	32.8	28	323.209	11.543
1	Ricky Rubio	PG	30.6	28	247.413	8.836
2	Kristaps Porzingis	PF	28.4	37	101.632	2.747
3	LeBron James	SF	35.6	29	85.153	2.936
4	DeAndre Jordan	C	33.7	28	68.048	2.430

Table 3: The top performers at each positions in Normalized EPA/Game

Something very pleasing about these results is that for every player other than Ricky Rubio, they made an All-Star game either this season (2016) or within the next two years. LeBron later this season won the Finals MVP. Another interesting thing about these results is that even after normalizing by position, the guards still have a ton of importance. Experts have always said a good point guard can help carry a team to victory but this model really shows that. The top five players at each position can be seen below in Tables 4-8.

	Name	Pos	MPG	GamesPlayed	NormalizedEPA	NormalizedEPAperGame
0	Ricky Rubio	PG	30.6	28	247.413	8.836
1	Damian Lillard	PG	35.7	34	205.282	6.038
2	Derrick Rose	PG	31.8	26	129.536	4.982
3	Isaiah Thomas	PG	32.2	34	123.833	3.642
4	Reggie Jackson	PG	30.7	34	123.024	3.618

	Name	Pos	MPG	GamesPlayed	NormalizedEPA	NormalizedEPAperGame
0	Goran Dragic	SG	32.8	28	323.209	11.543
1	Brandon Knight	SG	36.0	31	192.115	6.197
2	Jordan Clarkson	SG	32.3	33	167.906	5.088
3	James Harden	SG	38.1	33	156.785	4.751
4	Matthew Dellavedova	SG	24.6	29	155.019	5.345

	Name	Pos	MPG	GamesPlayed	NormalizedEPA	NormalizedEPAperGame
0	LeBron James	SF	35.6	29	85.153	2.936
1	Rudy Gay	SF	34.0	29	54.926	1.894
2	Robert Covington	SF	28.4	23	42.817	1.862
3	Marcus Morris	SF	35.7	33	33.979	1.030
4	Tony Snell	SF	20.3	29	33.382	1.151

	Name	Pos	MPG	GamesPlayed	NormalizedEPA	NormalizedEPAperGame
0	Kristaps Porzingis	PF	28.4	37	101.632	2.747
1	Nerlens Noel	PF	29.3	28	74.967	2.677
2	Derrick Favors	PF	32.0	20	73.481	3.674
3	Patrick Patterson	PF	25.6	33	50.308	1.524
4	Boris Diaw	PF	18.2	32	50.157	1.567

	Name	Pos	MPG	GamesPlayed	NormalizedEPA	NormalizedEPAperGame
0	DeAndre Jordan	C	33.7	28	68.048	2.430
1	Alex Len	C	23.3	28	61.999	2.214
2	John Henson	C	16.8	31	50.838	1.640
3	Channing Frye	C	17.1	27	50.814	1.882
4	Gorgui Dieng	C	27.1	34	50.459	1.484

Tables 4-8: Top players by Normalized EPA/Game at each position

Another encouraging result was that the difference in EPA between the two teams playing in each game was correlated with the final results. The team that had more Expected Points Added in the game generally won. This makes sense because

the EPA is supposed to be very correlated with the amount of points scored. The reason that it may not necessarily be 100% correlated with the final result is because it doesn't take into account the points added from offensive rebounds or steals. If you steal the ball you will have a high Expected Points right away so your EPA on that play may not be super high. Similarly, if you get an offensive rebound you most likely have a high Expected Points when you receive the ball and, thus, can't necessarily obtain a super high EPA on the play. Those two areas are the main places where Normalized EPA/Game could be improved.

Section 6 - Interpretation of Results

From these results, we can come to multiple conclusions. One is that even after normalizing by position, point guards and shooting guards are extremely important (and possibly more important than most people currently give them credit for). The median Normalized EPA/Game for the top five point guards and shooting guards was 4.98 and 5.34 respectively. For small forwards, power forwards, and centers, the median Normalized EPA/Game for top five players was 1.86, 2.68, and 1.88 respectively. This really shows just how important an elite guard is to a team. Logically, it does make sense that this statistic would think more highly of guards than traditional statistics would. Many guards not only score in similar amounts to the forwards on their teams but they are also handling the ball much more and setting up their teammates with passes. As stated earlier, one encouraging thing to see with this statistic is that there are some former all-stars in the top five leaderboards. For the point guards, one can see six time All-Star Damien Lillard, three time All-Star and one time MVP Derrick Rose, and two time All-Star Isaiah Thomas are in the top five.

Additionally, one can see that LeBron James is easily at the top of the small forwards. LeBron is known to be a “point forward” (a tall forward that can play the point guard position on offense) and is one of the greatest players of all time so it makes sense that he is so highly ranked in this stat. If anything, some may have expected him to be ranked a bit higher.

These results suggest that passing is a bit more of a variable skill in the NBA than scoring and could potentially separate similarly skilled players. Obviously, it is very important to be able to score since that is the main objective of basketball, but these results seem to favor players that are great passers. This could be a very important development in team building so that teams choose to build more around the Steph Curry or Chris Paul types than the Klay Thompson or Carmelo Anthony types (though obviously no one will complain if prime Klay Thompson or Carmelo is on their team). It could also influence draft choices with teams possibly treating point guards and players that are skilled at passing similar to how the NFL treats quarterbacks where they are taken very early and often. Obviously, being able to score is a very important and necessary skill for players to have, but we may be seeing that passing well has a bit more importance.

These results also can show that there may be some players who are underrated offensively. In hockey, there are two players that get an assist when someone scores but in basketball there is only one that gets the assist. This statistic can help identify those “hockey assists” that may not get recorded in the actual play but are just as important as the actual assist to the eventual points scored on the play. Now, everyone can see which players who may not have super high assist totals are still positively

impacting the play. Thus, one can be better able to identify players that “play within an offense” and make good passes to set up teammates. By doing this, one can find players that may not be properly rated by traditional statistics like points and assists.

An important part of the process of creating a statistic is looking at its stability over time. Generally, one would want to look at it from one season to the next but since there was only half a season of data, it was just split that in half. This gave approximately 15 games for each player in each sample with a minimum of 10 games in each half. Filtering out the players that only played a couple games in one half or the other made it so we would not be looking at players that may have had one or two game outliers to skew the statistic. From that one can see the stability of the statistic. With an r^2 value of 0.355, Normalized EPA/Game is fairly stable. Again, this has a big caveat that it is a fairly small sample size of only about 10-20 games in each set per player so the r^2 value could very well be a bit bigger or smaller than what was found if the analysis were to be done again comparing two consecutive seasons. In Figure 7, it can be seen that there certainly is a relationship between the Normalized EPA/Game in the first half of the dataset and the second half. It just may not be as big as one might expect but a bigger sample could potentially remedy that.

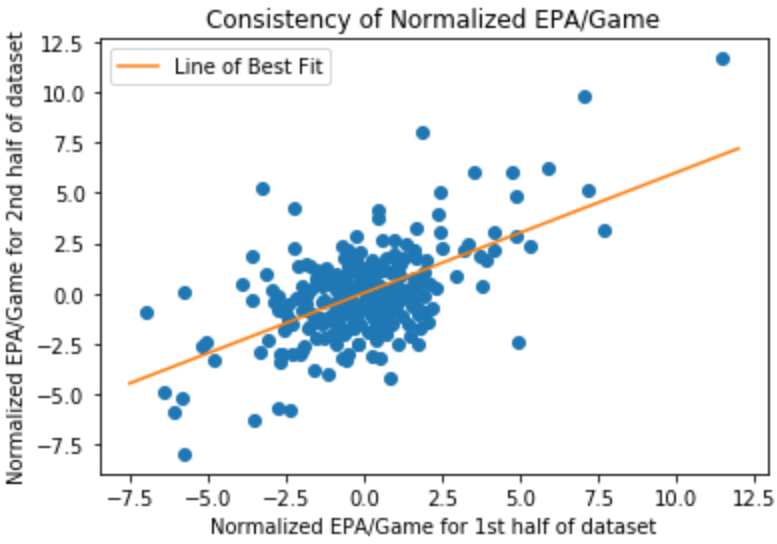

Figure 7: Consistency of Normalized EPA/Game with Dataset Split in Half

To put into perspective the r^2 value of 0.355, in Justin Jacob's [6] Adjusted Plus Minus model he only got an r^2 of 0.0245. A similar statistic that he looked at from Rosenbaum only had an r^2 of 0.15 so our statistic even in a small sample has a bit more stability than others. This shows that while the stability of Normalized EPA/Game may not be great, it is still sufficient enough to be a valid statistic.

One can also compare Normalized EPA/Game to other advanced NBA stats like Player Efficiency Rating (PER) and Offensive Box Plus Minus (OBPM). PER is a statistic (accessed at Basketball Reference [3]) created by John Hollinger that, according to Hollinger, "sums up all a player's positive accomplishments, subtracts the negative accomplishments, and returns a per-minute rating of a player's performance." For NBA players, 15 is average, below 10 is considered very poor, and above 25 is considered elite. Practically the entire box score goes into the formula that calculates PER including minutes played, points, assists, rebounds, field goal percentage, and much more. OBPM is an estimation of how many points a player would add to a team

per 100 possessions when compared to a league average player. It uses just about all the offensive statistics one could see in a box score like points, assists, offensive rebounds, turnovers, and more. Obviously, 0 is a league average score but scores above 5 are generally considered elite. Due to the fact that PER is more all inclusive whereas OBPM is just trying to incorporate the offensive side of the court, one might expect OBPM to correlate more with Normalized EPA/Game. That assumption would be correct with the caveat that neither correlates that well with Normalized EPA/Game.

When comparing these two statistics, it comes with the caveat that the PER and OBPM values were for the entire 2015-2016 season so it isn't a perfect match of games. PER has effectively no correlation with Normalized EPA/Game whereas OBPM has an r^2 of 0.01 when compared to Normalized EPA/Game. Going position by position, it is seen that some correlate more than others. When just looking at point guards, there is an r^2 of 0.06 between the two stats. With power forwards, there is an r^2 of 0.02 and every other position has effectively no correlation. Between PER and Normalized EPA/Game, there is a similar pattern with some positions correlating more than others. Point guards have an r^2 of 0.05 and centers have an r^2 of 0.06 but the other positions are largely uncorrelated. This can be seen in Figures 8 and 9 below. The trendlines of the data are in orange for each figure. The fact that there is very little correlation overall between Normalized EPA/Game and either stat can be seen. In reality, it seems like the only correlation seems to be with point guards for both stats. The r^2 is a higher number with point guards for both stats and the relationship can be seen in the figures below.

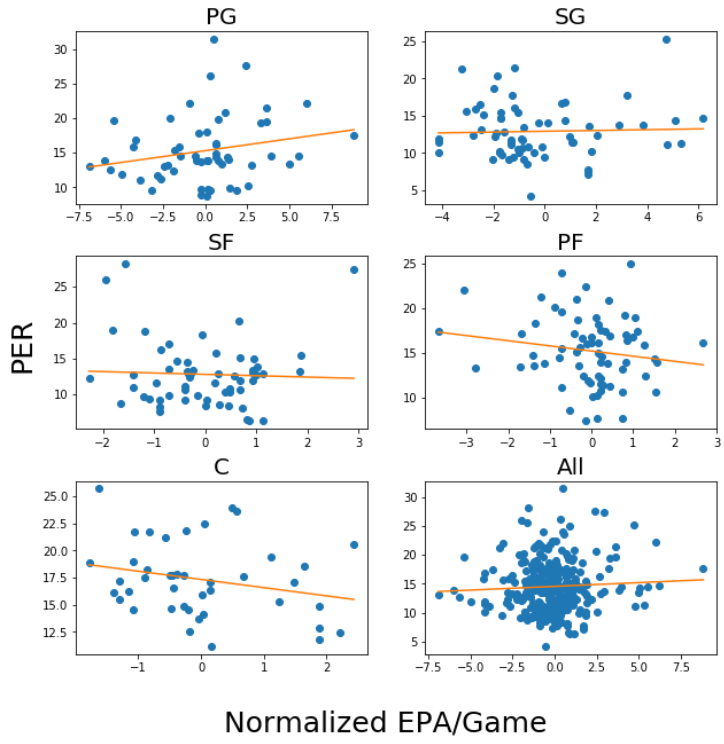

Figure 8: Correlation by position between Normalized EPA/Game and PER

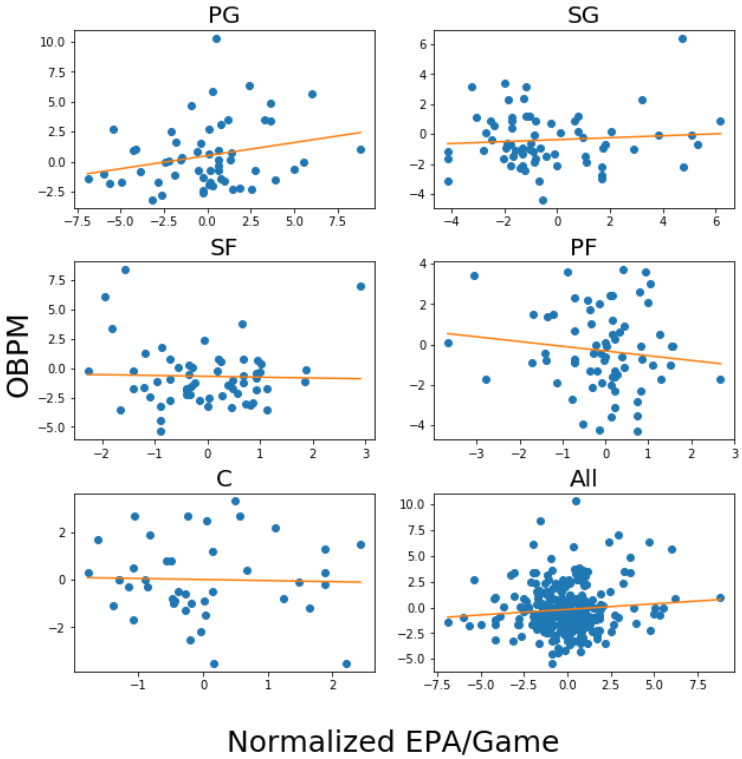

Figure 9: Correlation by position between Normalized EPA/Game and OBPM

Section 7 - Applications of Normalized EPA/Game

The importance of Normalized EPA/Game is pretty clear; it can help NBA executives more clearly evaluate players offensively. While most personnel evaluators would use statistics like points and assists to evaluate the scoring/playmaking ability of a player, Normalized EPA/Game is a much more inclusive statistic that takes more context into account. This would potentially help evaluators find players that are underrated because they do not have as good traditional metrics but may have a high Normalized EPA/Game. Additionally, it shows teams that they may be underrating the importance of the guards on the team. Obviously having players like LeBron or Kristaps Porzingis helps a lot to a team's success, but having distributors like Goran Dragic or Ricky Rubio could help the team a similar amount on the offensive side of the court. At the very least, it seems like Normalized EPA/Game correlates some with more "traditional" advanced statistics like PER and OBPM with point guards. Thus, personnel evaluators could use it to help choose who is better offensively when deciding between players that play the same position. Lastly, coaches could potentially alter their offensive schemes to take advantage of "high EP areas." Coaches already do this to an extent with plays that get the ball to the low post or corner three, but now they could find even more areas of the court that may have a high EP and attack those areas.

Works Cited

[1] Baldwin, Ben and Sebastian Carl. “NflfastR EP, WP, CP XYAC, and XPass Models.” Open Source Football, 5 Feb. 2021, www.opensourcefootball.com/posts/2020-09-28-nflfastr-ep-wp-and-cp-models/.

[2] Brownlee, Jason. “A Gentle Introduction to XGBoost for Applied Machine Learning.” Machine Learning Mastery, 16 Feb. 2021, machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/#:~:text=XGBoost%20is%20an%20algorithm%20that,designed%20for%20speed%20and%20performance.

[3] “Calculating PER.” *Basketball Reference*, www.basketball-reference.com/about/per.html.

[4] Cervone, Daniel, et al. “A Multiresolution Stochastic Process Model for Predicting Basketball Possession Outcomes.” Journal of the American Statistical Association, vol. 111, no. 514, 2016, pp. 585–599., doi:10.1080/01621459.2016.1141685.

[5] Goldsberry, Kirk. “Most Common Shot Locations In The NBA This Season. Pic.twitter.com/Kfk2MnKmBW.” Twitter, Twitter, 24 Mar. 2021, twitter.com/kirkgoldsberry/status/1374807806876868608.

[6] Jacobs, Justin. “Deep Dive on Regularized Adjusted Plus Minus II: Basic Application to 2017 NBA Data with R.” Squared Statistics: Understanding Basketball Analytics, 18 Sept. 2017, squared2020.com/2017/09/18/deep-dive-on-regularized-adjusted-plus-minus-ii-b

asic-application-to-2017-nba-data-with-r/.

[7] Linou, Kostya. “NBAPlayerMovement.py.” Github, 19 Sept. 2016.

[8] Pattani, Alok. “Expected Points and EPA Explained.” ESPN, ESPN Internet Ventures, 15 Sept. 2012, www.espn.com/nfl/story/_/id/8379024/nfl-explaining-expected-points-metric.

[9] Seward, Neil. “fix_shot_time.Py.” Github, 7 Feb. 2018.

[10] Sundaram, Ramya Bhaskar. “XGBoost Algorithm: XGBoost In Machine Learning.” Analytics Vidhya, 23 Dec. 2020, www.analyticsvidhya.com/blog/2018/09/an-end-to-end-guide-to-understand-the-math-behind-xgboost/.

[11] “XGBoost Parameters.” XGBoost Parameters - Xgboost 1.4.0-SNAPSHOT Documentation, XGBoost, xgboost.readthedocs.io/en/latest/parameter.html.