Kyle Arechiga

INFO 521 (Grad)

December 12, 2021

Predicting NBA Player Statistics Using Linear Regression

Introduction

Basketball analytics have boomed in the past decade, particularly in the National Basketball Association (NBA). Team executives, coaches, and players are always looking to gain an edge on the competition, while fans look for a way to better understand the game in front of them. The NBA introduces tracking data for all 30 teams during the 2013-14 season provided by SportVU. The SportVU cameras surrounded each team's arena, and provided state-of-the-art tracking data; some of which was made available to the public. In the 2015-16 season, the NBA partnered with Synergy Sports to capture even more data known as "play type" data. "Play type" data allowed fans and NBA personnel to observe the different types of plays that go on during the course of a game. Beginning in 2017-18, the NBA switched from SportVU to Second Spectrum, to provide the same tracking capabilities. Spatial tracking data and play type data allow us to look beyond the basic counting stats that have been around in basketball for decades. Instead of looking at a raw points per game value, we can observe *how* a player scores. For example, two-time NBA Most Valuable Player (MVP), Giannis Antetokounmpo, averaged 28.1 points per game in the 2020-21 season. Another two-time MVP, Stephen Curry, averaged 32 points per game that same season. Both are prolific scorers, but they score in vastly different manners. The "Transition" play type shows that Giannis led the league in transition points that season, scoring 8.2 transition points per game. Curry on the other hand, led the league in "Off-Screen" points, as he found many of his shooting opportunities coming off screens. These stats illustrate what makes these players so effective.

For this project, we want to use the many available tracking and play type stats to try and predict player stats (points, rebounds, assists) for the next season. With the help of Python's **nba_api** and **nba_stats_tracking** package, I was able to extract all the data needed for this task. From there, I used **sk-learn** package to perform Least Mean Squared (LMS) Linear Regression to fit my training data and used the test data to predict the next season's stats. Overall the results showed that we can use these play type and tracking stats to make good, generalized predictions of player's stats for the upcoming season.

Table 1: Model Features

Player Info	Play Type Stats		Tracking Stats		
AGE	**TRANS_FREQ**	**TRANS_PPP**	**DRIVES***	**PASSES_MADE***	**ELBOW_AST_PERC**
AGE_SQUARED	**ISO_FREQ**	**ISO_PPP**	**DRIVES_TS**	**PASSES_RECEIVED***	**ELBOW_TO_PERC**
YEARS_PRO	**PNR_BH_FREQ**	**PNR_BH_PPP**	**DRIVES_PTS_PERC**	**POTENTIAL_AST***	**POST_TOUCHES***
	PNR_R_FREQ	**PNR_R_PPP**	**DRIVES_AST_PERC**	**AST***	**POST_TS**
	POST_UP_FREQ	**POST_UP_PPP**	**DRIVES_TO_PERC**	**REB_CHANCES***	**POST_PTS_PERC**
	SPOT_UP_FREQ	**SPOT_UP_PPP**	**CATCH_SHOOT_FGA_2PT***	**CONTESTED_DREB***	**POST_AST_PERC**
	HANDOFF_FREQ	**HANDOFF_PPP**	**CATCH_SHOOT_FGA_3PT***	**CONTESTED_OREB***	**POST_TO_PERC**
	CUT_FREQ	**CUT_PPP**	**CATCH_SHOOT_EFG**	**TOUCHES***	**PAINT_TOUCHES***
	OFF_SCREEN_FREQ	**OFF_SCREEN_PPP**	**PULL_UP_FGA_2PT***	**ELBOW_TOUCHES***	**PAINT_TS**
	PUTBACK_FREQ	**PUTBACK_PPP**	**PULL_UP_FGA_3PT***	**ELBOW_TS**	**PAINT_PTS_PERC**
	MISC_FREQ	**MISC_PPP**	**PULL_UP_EFG**	**ELBOW_PTS_PERC**	**PAINT_AST_PERC**
					PAINT_TO_PERC

Data

All the data was obtained using nba_api[1] endpoints. Specifically, I used the *playercareerstats, synergyplaytypes, commonallplayers,* and *commonplayerinfo* endpoints. I used the features for the model are shown in Table 1.

For player information, I only used Age, Age Squared and Years Pro as features. The reason we want to include these is because players tend to start declining as they get older. I include age squared because I did not anticipate a completely linear relationship between age and points, rebound and assists. Young players are typically expected to improve early on until they hit their physical peak (usually in their late twenties) while older players tend to decline in their 30's.

The play type stats describe how frequently a player is involved in a type of play as well as how efficient they are. The frequency of each play is defined as the percentage of the player's total possessions that end up with that play type occurring. Points Per Possession (PPP) describes the number of points a player's team scores when they are involved in the specified play type. All available play types provided by Synergy were used for this model: Transition, Isolation, Pick & Roll Ball-Handler, Pick& Roll Roller, Post Up, Spot Up, Handoff, Cut, Off Screen, Putback, and Miscellaneous.

[1] "Nba_api/Nba_api/Stats/Endpoints at Master · Swar/Nba_api," GitHub, accessed December 12, 2021, https://github.com/swar/nba_api.

The tracking stats were collected using **nba_stats_tracking**[2] package in Python. I utilized most of the different tracking stats, and I normalized the volumetric measurements to per 36 minutes (* in Table 1 means per 36 minutes), to eliminate playing time as a factor. The tracking stats are all organized the same way as the play types are; some of them involve efficiency stats (EFG=Effective Field Goal Percentage, TS=True Shooting Percentage), some are ratios, and some are based on volume.

Two seasons were used to predict the 2018-19 season stats, and four seasons were used to predict the 2020-21 season stats. Players' feature vectors were removed from the training dataset if they didn't play at least 400 total minutes in either the current season or the next season (i.e. if a player played 1000 minutes in 2016-17 but only 300 in 2017-18, their features from the 2016-17 and 2017-18 season would not be included in the dataset. In total there were 597 player season feature vectors to predict the 2018-19 season and 1189 to predict the 2020-21 season.

Models

I used Ordinary Least Squares (OLS) Linear Regression and Regularized Least Squares (Ridge or RLS) Regression to fit the data and predict the outcomes. OLS is used to minimized the average squared error (loss) in your linear model. The loss function for OLS is defined in the following equation[3]:

$$\mathcal{L} = \frac{1}{N}(t - X\boldsymbol{w})^T(t - X\boldsymbol{w})$$

X is an N x M design matrix, where there are N datapoints and M features. t is an N x 1 matrix representing the output, which in this case will be the predicted points, predicted rebounds, and predicted assists for an NBA player. $\boldsymbol{w}$ is a M x 1 matrix which represents the coefficients that will be assigned to each feature. The goal is to minimize this loss on the training data. We can take the derivative of the loss function with respect to $\boldsymbol{w}$ and solve for it to obtain the coefficient weights that minimize this loss function. The equation for the optimal $\boldsymbol{w}$ vector to minimize the loss function is as follows:

$$\widehat{w} = (X^\top X)^{-1} X^\top t$$

One of the issues with ordinary least squares is that there is potential for over-fitting the training data. Over-fitting would mean that your loss function may be minimized on your training data, but it may diverge when introduced to the new test data. The way to solve this issue would be to use regularization. Regularized Least Squares introduces a coefficient α that is used to penalize the model for having $\boldsymbol{w}$ coefficients that are too large. It is shown in the equation below:

[2] Darryl Blackport, *Nba-Stats-Tracking: A Package to Work with NBA Player Tracking Stats Using the NBA Stats API*, version 0.0.10, OS Independent, Python, accessed December 13, 2021, https://github.com/dblackrun/nba-stats-tracking.

[3] Simon Rogers and Mark Girolami, *A First Course in Machine Learning* (Milton, UNITED STATES: CRC Press LLC, 2016), http://ebookcentral.proquest.com/lib/uaz/detail.action?docID=4718644.

$$\mathcal{L}' = \mathcal{L} + \alpha w^{\top} w$$

The larger the value is of α, the more the model will be penalized for overly complex models. In RLS the optimal value for w can be derived similarly to the OLS derivations; by taking the partial derivative with respect to w and solving for it.

$$\widehat{w} = (X^{\top}X + N\alpha I)^{-1}X^{\top}t$$

In the model used for this project, we look at the varying performance with $\alpha = 0$ (same as OLS model), $\alpha = 1$, $\alpha = 10$, or $\alpha = 30$.

Procedure

nba_stat_predictions.py is the main function that is ran, while all the functions it uses are referenced in packages or in the other file located in the same folder named **nba_stat_predictions_functions.py**.

The data was captured, as mentioned before, using nba_api and nba_stats_tracking packages in Python. When first acquiring the data, I have it save all the data structures that were formed in three separate .txt files named **nba_playtypes.txt**, **nba_tracking.txt**, and **nba_player_seasons.txt.** This is done so we don't have to reload the data from the NBA.com server every time we want to run the model. Loading the five seasons of data from NBA.com took about an hour and is not recommended to do that every time. The three Boolean variables at the beginning of **nba_stat_predictions.py** are used to determine whether we are loading new data from NBA.com or loading from the local .txt file.

If we set those Boolean variables to `true`, we will load from NBA.com all the play type stats from the five seasons, all the tracking stats from the five seasons, and all the eligible "player seasons" to be included in the dataset (i.e. LeBron James would have five "player seasons" in this dataset from 2015-16 to 2019-20 because he played at least 400 minutes in each of those seasons). The **seasons** variable can be modified to determine which seasons we will using as training data and which one will be used as the test data. The seasons must all be formatted as **YYYY-YY** (i.e. 2016-17), and the last season will be held out of the training data. If updating the seasons variable, you must set the three Boolean variables (**load_playtype, load_tracking,** and **load_player_seasons**) to `true`, so that the model can grab all the new seasons data from NBA.com.

If **load_playtype, load_tracking**, and **load_player_seasons** are set to `False`, the script will quickly load all the necessary data from the .txt files. The data is loaded using the `get_training_data` as well as the `get_players_seasons` functions that are defined in **nba_stat_predictions_functions.py.**

From this point, the script will organize the data into the training design matrix `X`, the test design matrix `X_test`, the training output data `t_pts, t_reb, t_ast`, and the test output data that will be compared to the predicted output `t_pts_test, t_reb_test`, and `t_ast_test.` To do this, the script iterates over the elements in the `seasons` variable (except for the last one which will be the test season) and then in a nested loop, iterates over the `player_seasons` variable. For example, one of the loops will be when the season is "2016-17" and we will go through each eligible player from that season and find their play type, tracking, and player info features. Their features will be added to a feature vector which will then be appended to the `X` design matrix. Likewise, the players' points, rebounds, and assists (per 36 minutes) for the following season will be stored in the output vectors, `t_pts,` `t_reb`, and `t_ast.`

We do the same thing to generate the test design matrix `X_test` and the test outputs `t_pts_test,` `t_reb_test`, and `t_ast_test`. To avoid feature leakage, the test outputs will not be used at all in training the data.

Once the design matrix and output vectors are complete, we can use the **sklearn**[4] package to implement the regression algorithms. Before fitting the data, some of the feature values needed to be imputed. Many players in the dataset had some **missing** data because some of the play types and tracking features were rarely performed by certain players. For example, Stephen Curry did not post up hardly at all during this time frame, so he doesn't not have any Post Up data for his play type features. In this case, his **POST_UP_FREQ** feature will be 0, and his **POST_UP_PPP** will be `np.nan`. NaN values will not work for the regression models, so the solution to this problem is to "impute" values instead of having NaN values. With **sklearn**, we are able to use its `SimpleImputer` to impute the missing values with the **mean** of the rest of the training dataset for that feature. Again, to avoid feature leakage from the test data, the test data will have the exact same imputations as the training data (i.e. the mean of **POST_UP_PPP** from the training data will be used for the test data's missing values for that feature). This is a crucial step for the regression to work. Code snippet for this step is shown below:

```
# Using the Simple imputer to replace all NaN values with the average of the rest
of the set
imp = SimpleImputer(missing_values=np.nan, strategy='mean')
imp.fit(X)
X_transform = imp.transform(X)
X_test_transform = imp.transform(X_test)
```

After transforming the data with the Simple Imputer, we then want to normalize the data so that the weight coefficients will be comparable to each other. **sklearn** has a `MinMaxScalar()` function which allows us to scale all of the data between 0 and 1. After this, we can finally use **sklearn** to fit to our different regression models with the training data and make predictions using the test data.

For the Ordinary Least Squares model, we use the `linear_model.LinearRegression()` function from **sklearn**. From there, we can fit the transformed training data into the models.

```
# Using sci-kit-learn to perform least squares linear regression
```

[4] "1.1. Generalized Linear Models — Scikit-Learn 0.15-Git Documentation," accessed December 13, 2021, https://scikit-learn.org/0.15/modules/linear_model.html.

```python
reg_pts = linear_model.LinearRegression()
reg_reb = linear_model.LinearRegression()
reg_ast = linear_model.LinearRegression()

# fit the regression model with the training data after processing
reg_pts.fit(X_train_minmax,t_pts)
reg_reb.fit(X_train_minmax,t_reb)
reg_ast.fit(X_train_minmax,t_ast)
```

After the model is fitted to our training data, we can predict the outcomes and observe which coefficients have the largest effect on predictions. We use the same methodology to perform Regularized least squares, except we use the `linear_model.Ridge()` function from **sklearn**. With this we can set our `alpha` value to 1, then 10, and then 30. In total, we end up with 12 different models to evaluate – OLS and the three RLS models that are used to predict points, rebounds, and assists.

We use **Pandas DataFrames**[5] to organize the results and store them to an Excel file named **"2020-2021_predictions.xlsx"**.

Using Excel, I analyzed the data and created charts shown in the results. The all processed data analysis is located in the "2020-21_predictions_processed.xlsx" file.

Evaluation

To evaluate the models, I used the mean squared error calculations (MSE) and used Excel to partition the data as needed. The MSE will tell me how far off the model's predictions were from the actual result.

Some questions we want to ask ourselves about the different models:

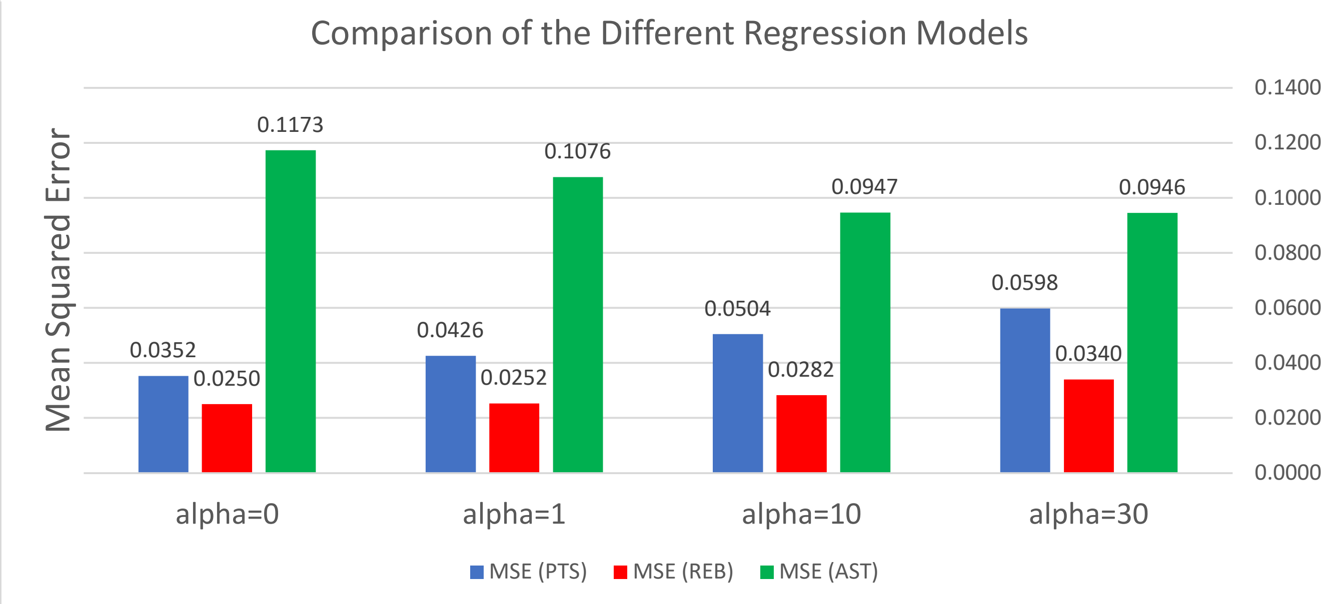
1. Which value of alpha does the model perform the best with?
2. How does the model perform with players of different ages?
3. How does the model perform with players who ended up having a breakout seasons or large declining seasons?
4. How does the model perform with players on new teams?
5. Which regression coefficients hold the most weight? Why?

Results

The models all perform fairly well with predicting points and rebounds, but not nearly as well with assists, as shown in Figure 1. The assists prediction actually shows to be the most accurate at alpha = 30 meaning there was probably some overfitting with some of the features. Points and

[5] "Pandas.DataFrame — Pandas 1.3.5 Documentation," accessed December 13, 2021, https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html.

Figure 1: Comparison of the different Regression Models with varying alphas

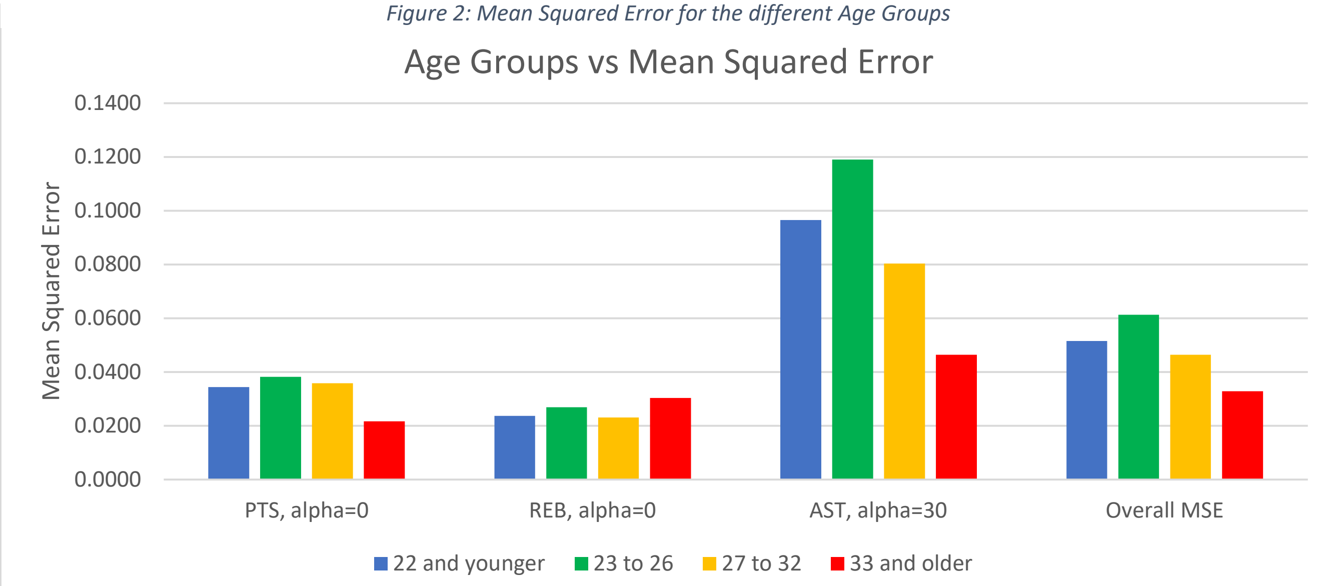


rebounds predictions both performed better at lower values of alpha, suggesting that the original OLS model was sufficient for predicting points and rebounds.

Using this information, I used alpha=0 for points and rebounds predictions for the rest of this analysis, and alpha=30 for predicting assists. Figure 2 shows the relationship that the different ages have on the models' performances. Players aged 33 years or older are shown to be a little easier to predict overall, and especially when predicting assists. Older players typically are who they are in terms of their skills by that age. Whereas ages under 22 are often improving and still learning what their niche is on a team in the NBA.

The assists are shown to be difficult to predict, because the model seems to be lacking features that are predictive of assists. Referring to the weight coefficients in Appendix A, the model

Figure 2: Mean Squared Error for the different Age Groups



seems to be highly dependent on the **AST** and **POTENTIAL_AST** features, and not much else. With predicting points and rebounds, those have a relationship with a lot more of the features.

Looking at the points coefficients closer, it has a very high dependency on Pull-Up 2-pointer attempts and a strong negative relationship with Passes Made. Coefficients that high may be a risk for overfitting, but it seemed to predict the points well without a need for regularization.

One of the most important parts of building this model was seeing how it could predict improve ments and declines. Table 2 show the 25 players who improved their Points per 36 minutes stat the most from the previous year to the 2020-21 season, and Table 3 shows who declined the most. The model does a decent job at predicting improvements, but typically underestimates the improvement of these breakout players, which is not necessarily a bad thing. The players highlighted in red in Table 2 were predicted to decline their scoring output, but actually increased their scoring. Sometimes major improvements and declines happen for reasons that are not illustrated in the previous season's features. One of the big factors is players who have changed teams over the offseason. A great example from Table 2 is Jerami Grant. Jerami Grant was known as a role player in his time with the Denver Nuggets, but when he signed with the

Breakout Players				
Name	Age	Years Pro	PTS improvement	Predicted Improvement
Kenrich Williams	25	1	125.7%	18.8%
Nicolas Batum	31	11	88.1%	50.4%
Luguentz Dort	20	0	58.8%	13.7%
Jordan Poole	20	0	56.9%	28.8%
Kyle Anderson	26	5	54.3%	10.6%
Jerami Grant	25	5	46.5%	-6.7%
Darius Bazley	19	0	45.2%	21.0%
Hamidou Diallo	21	1	38.8%	3.9%
Cory Joseph	28	8	37.3%	13.7%
Khem Birch	27	2	37.1%	17.6%
Edmond Sumner	24	2	36.3%	3.7%
OG Anunoby	22	2	34.1%	15.0%
Mo Bamba	21	1	33.9%	-4.9%
Wayne Ellington	32	10	33.2%	17.3%
Mike Muscala	28	6	33.0%	-19.9%
Daniel Gafford	21	0	33.0%	5.1%
Darius Garland	19	0	31.7%	6.2%
De'Andre Hunter	22	0	31.6%	1.8%
Joe Ingles	32	5	31.0%	6.7%
Dwayne Bacon	24	2	30.4%	16.5%
Al Horford	33	12	29.2%	-0.6%
Shai Gilgeous-Alexander	21	1	28.3%	2.2%
JaMychal Green	29	5	27.8%	-8.6%
Robin Lopez	31	11	27.2%	-16.3%
Mikal Bridges	23	1	26.5%	7.0%

Table 2: Predicted Improved Points vs Actual Points Improvement – Breakout Players

Detroit Pistons before the 2020-21 season, he took on a completely different role as a primary offensive scorer for the Pistons. Therefore the error on his prediction is very high, and there is not much that can be done with the current model to predict his role on a new team.

The same can be said about players who unexpectedly decline, such as Steven Adams who was traded from the Oklahoma City Thunder to the New Orleans Pelicans over the 2020 Offseason. The Pelicans seemed to be a worse fit for Adams, who is known for being a big man occupying the paint, grabbing rebounds and setting screens. The Pelicans team was built around Zion Williamson, a star player who also was more of an interior force. Adams' fit with Williamson and the rest of his new team seemed to be suboptimal, which could be one of the reasons behind why his production declined.

Overall, the model shows potential for predicting player stats for upcoming years. This model could help team personnel and fans predict the career trajectories for different players. Future improvements could include other features that relate to assists, as well as additional features to account for size, injury history, and team fit.

Declining Players				
Name	Age	Years Pro	PTS improvement	Predicted Improvement
Nerlens Noel	25	6	-47.4%	-13.1%
Wes Iwundu	25	2	-41.7%	0.9%
Nicolo Melli	28	0	-37.5%	-5.5%
Aron Baynes	33	7	-35.8%	-21.7%
Bismack Biyombo	27	8	-35.8%	-6.3%
Robert Covington	29	6	-35.1%	-7.2%
Gary Clark	25	1	-34.0%	-11.5%
Steven Adams	26	6	-32.9%	17.1%
Isaac Bonga	20	1	-31.6%	20.2%
Taj Gibson	34	10	-30.4%	-15.5%
Rodney Hood	27	5	-29.8%	5.8%
P.J. Tucker	34	13	-29.8%	-9.3%
Jordan McLaughlin	23	0	-29.7%	-11.2%
Solomon Hill	28	6	-29.5%	7.3%
James Harden	30	10	-28.4%	-3.5%
Mitchell Robinson	21	1	-28.2%	5.1%
Markieff Morris	30	8	-28.0%	-10.7%
Mike Scott	31	7	-26.0%	-2.4%
Maxi Kleber	27	2	-26.0%	-16.1%
Eric Bledsoe	30	9	-25.7%	-2.4%
Garrett Temple	33	10	-25.3%	-13.1%
Troy Brown Jr.	20	1	-23.4%	5.1%
Josh Okogie	21	1	-23.4%	5.4%
Brandon Goodwin	24	1	-23.2%	-3.7%
Ish Smith	31	9	-22.9%	-7.9%

Table 3: Predicted Improved Points vs. Actual Improved Points - Decliners

APPENDIX A: Regression Coefficients

Regression Coefficients	PTS (Alpha=0)	PTS (Alpha=1)	PTS (Alpha=10)	PTS (Alpha=30)	REB (Alpha=0)	REB (Alpha=1)	REB (Alpha=10)	REB (Alpha=30)	AST (Alpha=0)	AST (Alpha=1)	AST (Alpha=10)	AST (Alpha=30)
CONSTANT	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
AGE	-4.890	-3.297	-1.962	-1.358	0.375	-0.260	-0.229	-0.190	-2.821	-0.467	-0.042	0.031
AGE_SQUARED	1.267	-1.642	-1.681	-1.234	-1.152	-0.398	-0.247	-0.194	3.054	0.356	0.041	0.050
YEARS_PRO	0.874	1.725	0.744	0.280	0.515	0.477	0.274	0.135	-0.062	0.088	0.143	0.148
TRANS_FREQ	1.953	3.626	2.333	1.083	0.425	0.489	0.316	0.026	0.849	0.867	0.777	0.539
TRANS_PPP	2.014	1.926	1.553	1.094	0.533	0.478	0.250	0.168	-0.472	-0.466	-0.406	-0.321
ISO_FREQ	1.541	3.898	3.805	2.740	-0.700	-0.394	0.096	0.096	0.074	0.041	0.244	0.302
ISO_PPP	0.312	0.117	0.534	0.639	-0.059	-0.116	-0.150	-0.109	-0.067	-0.073	0.020	0.069
PNR_BH_FREQ	-0.729	-1.425	-0.273	0.286	-0.332	-0.567	-0.861	-0.879	0.269	0.291	0.583	0.764
PNR_BH_PPP	3.955	4.394	2.997	1.936	0.517	0.544	0.330	0.179	0.445	0.397	0.212	0.132
PNR_R_FREQ	0.552	0.558	0.467	0.245	-0.572	-0.408	0.139	0.534	0.346	0.240	-0.045	-0.146
PNR_R_PPP	2.305	1.691	0.787	0.463	1.203	1.004	0.492	0.269	0.292	0.235	0.157	0.097
POST_UP_FREQ	1.304	1.743	2.265	2.022	0.357	0.334	0.399	0.541	0.495	0.316	0.155	0.102
POST_UP_PPP	1.322	1.503	1.130	0.846	-0.319	-0.282	-0.233	-0.158	0.309	0.269	0.179	0.099
SPOT_UP_FREQ	-0.871	-1.707	-2.564	-2.380	0.103	0.014	-0.246	-0.408	-0.523	-0.495	-0.535	-0.501
SPOT_UP_PPP	3.828	3.465	2.519	1.748	-0.569	-0.458	-0.276	-0.257	-0.013	0.073	0.115	0.073
HANDOFF_FREQ	-1.350	-0.897	-0.048	0.192	0.213	0.076	-0.356	-0.512	0.117	0.107	0.004	-0.034
HANDOFF_PPP	0.485	0.596	0.756	0.701	0.000	-0.008	-0.058	-0.057	0.219	0.207	0.131	0.076
CUT_FREQ	0.025	-1.481	-1.846	-1.488	-0.672	-0.517	-0.056	0.328	0.131	-0.032	-0.111	-0.137
CUT_PPP	2.082	1.831	1.350	1.052	-0.308	-0.312	-0.183	-0.098	-0.105	-0.081	-0.008	0.024
OFF_SCREEN_FREQ	3.040	4.172	3.468	2.379	-0.551	-0.539	-0.521	-0.517	-0.251	-0.299	-0.301	-0.284
OFF_SCREEN_PPP	2.034	1.978	1.351	0.840	-0.047	-0.037	-0.033	-0.028	0.103	0.068	0.017	0.008
PUTBACK_FREQ	1.379	-1.562	-0.694	-0.452	0.695	0.568	1.046	1.087	-0.996	-0.608	-0.243	-0.190
PUTBACK_PPP	2.427	1.772	1.391	1.156	-0.086	-0.113	-0.031	0.024	0.167	0.164	0.126	0.073
MISC_FREQ	-1.688	-2.222	-2.175	-1.542	0.300	0.287	0.361	0.410	-0.037	0.058	0.284	0.264
MISC_PPP	1.028	1.737	2.405	2.313	-0.267	-0.200	-0.045	-0.034	-0.109	-0.126	-0.121	-0.067
DRIVES	-1.086	5.975	4.501	3.106	0.469	0.534	0.188	-0.083	-0.188	-0.006	0.576	0.780
DRIVES_TS	2.982	0.845	1.216	1.164	0.638	0.312	0.209	0.168	1.111	0.666	0.294	0.192
DRIVES_PTS_PERC	-2.451	1.670	2.050	1.762	-0.498	0.010	0.256	0.290	-1.016	-0.476	-0.189	-0.155
DRIVES_AST_PERC	-0.035	-0.956	-1.368	-1.221	-0.180	-0.074	-0.172	-0.311	-0.260	0.028	0.491	0.539
DRIVES_TO_PERC	-2.283	0.232	0.105	-0.006	-0.837	-0.281	0.037	0.052	-1.328	-0.647	-0.130	-0.059
CATCH_SHOOT_FGA_2PT	0.724	4.444	1.966	1.067	-0.051	-0.143	-0.231	-0.086	-0.425	-0.364	-0.209	-0.145
CATCH_SHOOT_FGA_3PT	-3.000	1.416	1.660	0.950	0.194	0.361	0.200	-0.098	-0.567	-0.575	-0.563	-0.527
CATCH_SHOOT_EFG	-0.254	0.108	0.742	0.648	0.027	-0.030	-0.210	-0.283	0.419	0.331	0.153	0.084
PASSES_MADE	-64.503	-10.997	-2.714	-1.195	-0.675	0.550	0.466	0.424	-0.022	0.223	0.576	0.739
PASSES_RECEIVED	-1.065	3.112	1.950	1.575	-3.853	-2.707	-1.018	-0.536	0.159	0.442	0.991	1.091
POTENTIAL_AST	-8.640	-3.760	-0.023	0.480	-1.786	-0.315	0.057	0.009	2.343	3.235	2.405	1.735
AST	6.129	2.716	1.036	0.870	2.918	1.343	0.411	0.170	6.522	4.866	2.775	1.890
PULL_UP_FGA_2PT	68.017	9.915	2.636	1.787	3.433	1.492	0.910	0.692	0.483	0.338	0.712	0.877
PULL_UP_FGA_3PT	-0.215	3.417	3.515	2.982	0.482	0.439	-0.014	-0.214	0.022	-0.019	0.006	0.143
PULL_UP_EFG	2.765	9.322	5.116	2.922	1.164	1.055	0.375	0.089	0.303	0.403	0.405	0.331
REB_CHANCES	0.149	-0.138	0.515	0.730	0.004	-0.009	-0.079	-0.154	0.069	0.087	0.109	0.092
CONTESTED_DREB	1.098	2.743	1.971	1.278	7.019	6.157	3.849	2.659	-0.639	-0.427	-0.080	-0.031
CONTESTED_OREB	1.472	2.419	2.419	1.724	4.208	4.452	3.694	2.677	0.445	0.316	0.052	-0.035
TOUCHES	-0.978	2.226	1.301	0.784	0.447	1.315	1.884	1.692	1.305	0.652	0.039	-0.064
ELBOW_TOUCHES	-1.234	0.338	1.141	0.964	-0.201	-0.096	0.237	0.514	-0.221	-0.157	0.026	0.037
ELBOW_TS	0.852	-0.324	-0.482	-0.141	0.006	0.001	0.093	0.116	0.341	0.351	0.271	0.183
ELBOW_PTS_PERC	-2.003	-0.241	1.028	1.268	0.085	0.167	0.173	0.090	-0.310	-0.288	-0.228	-0.193
ELBOW_AST_PERC	0.168	0.319	-0.142	-0.318	-0.068	-0.027	-0.031	-0.034	0.157	0.256	0.421	0.411
ELBOW_TO_PERC	-0.817	0.215	0.360	0.234	-0.027	0.029	0.073	0.053	1.051	0.972	0.523	0.268

POST_TOUCHES	-0.862	3.447	3.053	2.453	0.366	0.396	0.573	0.650	-0.385	-0.132	0.130	0.124
POST_TS	0.783	0.234	0.386	0.564	-0.388	-0.368	-0.203	-0.110	-0.323	-0.277	-0.074	-0.010
POST_PTS_PERC	0.022	1.169	1.391	1.222	0.678	0.613	0.295	0.154	0.582	0.538	0.183	0.046
POST_AST_PERC	-0.214	-0.525	-0.603	-0.452	0.156	0.165	0.108	0.059	-0.237	-0.145	0.052	0.092
POST_TO_PERC	-0.228	-0.227	-0.330	-0.249	0.317	0.276	0.179	0.134	-0.088	-0.077	-0.053	-0.036
PAINT_TOUCHES	-0.724	1.913	1.324	0.831	2.055	1.782	1.578	1.472	-0.701	-0.347	-0.105	-0.098
PAINT_TS	-0.966	-1.847	-0.338	0.158	0.128	0.012	-0.070	-0.037	0.616	0.470	0.212	0.125
PAINT_PTS_PERC	0.619	3.419	2.664	2.062	0.324	0.488	0.436	0.373	-0.682	-0.513	-0.226	-0.191
PAINT_AST_PERC	-0.705	-1.412	-1.486	-1.196	0.099	0.105	-0.046	-0.133	-0.382	-0.190	0.211	0.323
PAINT_TO_PERC	-0.970	-0.535	-0.571	-0.445	-0.214	-0.178	-0.159	-0.114	-0.198	-0.123	0.079	0.104

References

"1.1. Generalized Linear Models — Scikit-Learn 0.15-Git Documentation." Accessed December 13, 2021. https://scikit-learn.org/0.15/modules/linear_model.html.

Blackport, Darryl. *Nba-Stats-Tracking: A Package to Work with NBA Player Tracking Stats Using the NBA Stats API* (version 0.0.10). OS Independent, Python. Accessed December 13, 2021. https://github.com/dblackrun/nba-stats-tracking.

GitHub. "Nba_api/Nba_api/Stats/Endpoints at Master · Swar/Nba_api." Accessed December 12, 2021. https://github.com/swar/nba_api.

"Pandas.DataFrame — Pandas 1.3.5 Documentation." Accessed December 13, 2021. https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html.

Rogers, Simon, and Mark Girolami. *A First Course in Machine Learning*. Milton, UNITED STATES: CRC Press LLC, 2016. http://ebookcentral.proquest.com/lib/uaz/detail.action?docID=4718644.