The Battle for Possessions: An NBA Rebounding Regression Prop Betting Sports Model

Stanford CS229 Project

David Maemoto
Department of Computer Science
Stanford University
davidmaemoto@stanford.edu

Abstract

At the center of NBA basketball productivity is creating possessions. More possessions equals more opportunities which then leads to more points which finally yields more team wins. I utilized machine learning methods to evaluate the most simple yet critical aspect of basketball: creating possessions. By training and analyzing the key factors behind rebounding over the past 4 NBA seasons (2018 - 2021) found publicly on Kaggle, I was able to create a consistent profitable strategy for NBA player prop rebounding bets. Building off this basic framework, I then utilized a more complex 2-layer neural network to weight a consensus continous valued output for a player's rebounding prediction as a linear combination of the underlying regression models, achieving a final accuracy of 56.93% and profiting 25.96 betting units in 31 days of testing on games in the 2024-2025 NBA season.

1 Introduction

The goal of this project is to create a predictive model for NBA prop sports betting that forecasts individual player performance in rebounds. The high volume in rebounding opportunities in a single game, player physical attributes, and average positioning relative to the basket are all key factors behind the decision to use rebounding as the key player prop bet for my models. The overall complexity of predicting player performance is dependent on relative strength of opponent(s), defensive matchups, game pace, injuries, and player fatigue. The input to my algorithm is a combination of each player's physical attributes, each player's average/projected statistics, and team and opposing team statistics for all NBA games. I then utilized SVR (SVM for continuous outputs), regression with Huber Loss, and extreme gradient boosting algorithms to output a predicted number of rebounds for the given player. These 3 outputs for the 3 algorithms become the first layer of a neural network where a linear combination of the 3 outputs is produced into a singular continuous value which corresponds to the predicted number of rebounds that a specific player will have in the game.

2 Related Work

Key studies have explored critical features influencing NBA rebounding (Johannsen, 2023)(Kom, 2023). For example, Hojo et al. (2019) identified position, player movement, team dynamics, and techniques like boxing out as central determinants. While their approach relied on SVMs and logistic regression to evaluate team techniques, my work focuses on using these features to predict individual rebounds. Similarly, Wang (2023) influenced my use of more complex architectures, such as neural networks, for higher accuracy and model sophistication. Though their study targeted game outcomes, their DNN and random forest methodologies guided my design of a two-layer neural network. Finally, Turch Ferreres (2022) provided insights into the underlying probabilities and the quantification of intangible factors (e.g., hustle, positioning) associated with rebounding. Studies like Byman (2023) and Wheeler (2012) offered guidance on feasible public metrics and features, though their focus was broader player performance rather than rebounds specifically. More directly, Kiriazis (2023) modeled rebounding probabilities through a two-stage Bayesian approach, treating rebounds as a chain of conditional events rather than independent occurrences. While my work applies more conventional ML techniques, these insights underscore the evolving complexity and sophistication of modeling player-level rebounding.

3 Dataset and Features

This study utilizes data from four NBA seasons (2017-2021), sourced from two public Kaggle datasets on NBA players and games as found here: Justinas (2020); Lauga (2020). For preprocessing and filtering, players are included in the training data if they meet the eligibility criterion of averaging at least 15 minutes played per game (MPG) across the season. The data is then split into player-level and team-level data where `player_feature_dict.json` contains all of the relevant player statistic data averages for a given season and `team_features.json` contains all of the relevant team statistic data averages for a given season. Then, for each player in every game in each season, we construct a feature vector combining the relevant player statistic data, team statistics, and opposing team statistics. The feature vector for each game consists of a total of 19 variables capturing player, team, and opposing team statistics as shown below.

The feature vector includes the most significant features influencing rebounding, grouped by category:

- **Player stats**: Average shot distance, percent of field goals attempted from 0-3 feet, field goal percentage from 0-3 feet, overall field goal percentage, years of experience, average minutes played per game, average total rebounds per game, average blocks per game, average free throws attempted per game, and player position.
- **Team stats**: Offensive rating, defensive rating, pace of play, 3-point attempt rate, offensive rebound percentage, and number of 3-point attempts by opposing teams.
- **Opponent stats**: offensive rebounds per game by opposing team, defensive rebounds per game by opposing team, and a binary indicator for whether the game is home or away.

The player performances for each player for each game from each season are divided into training set (70%, 21,532 rows) and a test set (30%, 9,228 rows) which are cycled using repeated k-fold validation, with a total dataset size of 30,760 rows. Additionally, each feature is normalized using the sklearn preprocessing Python library where each feature in the feature vector is standardized and normalized using a StandardScaler() object such that each feature in the vector has a mean of 0 and unit standard deviation. Since the values of the features in the vector vary as some are represented as percentages while others are continuous values, the normalization and standardization ensure that all features contribute to the result, avoiding bias due to varying scale of the feature values.

Example raw feature vector for 2018, Russell Westbrook, OKC Thunder versus ATL Hawks:

avg_dist_fga	percent_fga_from_x0_3_range	fg_percent_from_x0_3_range	fg_percent	experience	Usage_game	Trb_per_game	Blk_per_game	Fta_per_game	Pos	Offensive_Rating_Ratio	Pace_Ratio	Opp_x3pa_per_game	Defensive_Rating_Ratio	X3par	Oreb_pct	Opp_orb_per_game	Opp_drb_per_game	Home/Away
11.3	0.375	0.611	0.449	10.0	36.4	10.1	0.3	7.1	1	110.7	96.7	30.6	107.2	0.345	27.7	9.6	34.2	1.0

Normalized feature vector for 2018, Russell Westbrook, OKC Thunder versus ATL Hawks:

avg_dist_fga	percent_fga_from_x0_3_range	fg_percent_from_x0_3_range	fg_percent	experience	Usage_game	Trb_per_game	Blk_per_game	Fta_per_game	Pos	Offensive_Rating_Ratio	Pace_Ratio	Opp_x3pa_per_game	Defensive_Rating_Ratio	X3par	Oreb_pct	Opp_orb_per_game	Opp_drb_per_game	Home/Away
0.109	-0.121	-0.241	0.128	0.092	0.872	1.982	0.225	0.811	-0.421	0.248	0.081	0.601	-0.176	-0.276	0.598	0.134	0.189	0.500

4 Methods

The methodology consists of implementing three predictive machine learning models—Support Vector Regression (SVR), regression with Huber Loss, and extreme Gradient Boosting (XGBoost)—and combining their outputs in a neural network ensemble to produce the final prediction. I utilized Python along with common ML and visualization libraries including SciPy, Matplotlib, Pandas, NumPy, Scikit-learn, XGBoost, and Seaborn (Virtanen et al., 2020)(Hunter, 2007)(McKinney, 2010)(Harris et al., 2020)(Pedregosa et al., 2011)(Chen and Guestrin, 2016a)(Waskom, 2021).

4.1 Regression with Huber Loss

Huber Loss is a robust loss function that combines mean squared error (MSE) and mean absolute error (MAE) (Gokcesu and Gokcesu, 2021). For a prediction $\hat{y}_i$ and target y_i, it is defined as:

$$L(y_i, \hat{y}_i) = \begin{cases} \frac{1}{2}(y_i - \hat{y}_i)^2 & \text{if } |y_i - \hat{y}_i| \le \delta, \\ \delta |y_i - \hat{y}_i| - \frac{1}{2}\delta^2 & \text{if } |y_i - \hat{y}_i| > \delta, \end{cases}$$

where $\delta = 3$ is a hyperparameter that determines the threshold for switching between quadratic and linear penalties. This loss function was appropriate since it makes the regression robust to outliers, which is essential for modeling rebounds where extreme values can occur.

4.2 Support Vector Regression (SVR)

Support Vector Regression is used for continuous output prediction and is based on Support Vector Machines (SVM) (Basak et al., 2007). The objective of SVR is to find a hyperplane that minimizes the error within a margin of tolerance, $\epsilon = 0.5$. The optimization problem is given by:

$$\min_{\mathbf{w}, b, \xi, \xi^*} \frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^{n} (\xi_i + \xi_i^*)$$

subject to:

$$y_i - (\mathbf{w} \cdot \mathbf{x}_i + b) \leq \epsilon + \xi_i, \quad (\mathbf{w} \cdot \mathbf{x}_i + b) - y_i \leq \epsilon + \xi_i^*, \quad \xi_i, \xi_i^* \geq 0,$$

where $\mathbf{x}_i$ are the input features, y_i is the true target value (rebounds), and $C = 1$ is the regularization parameter that controls the trade-off between margin width and tolerance for errors. The implementation in the code uses the Radial Basis Function (RBF) kernel, which maps the input space to a higher-dimensional space to handle non-linear regression problems.

4.3 eXtreme Gradient Boosting (XGBoost)

XGBoost is a tree-based ensemble learning algorithm that improves prediction accuracy by sequentially adding decision trees that correct errors from prior iterations (Chen and Guestrin, 2016b). It minimizes the following objective function:

$$\mathcal{L} = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k),$$

where $l(y_i, \hat{y}_i)$ is the loss function (e.g., squared error), $\Omega(f_k) = \gamma T + \frac{1}{2}\lambda\|\mathbf{w}\|^2$ is a regularization term to prevent overfitting, γ is a regularization parameter penalizing additional tree complexity, T is the number of leaves in a tree, λ is the L2 regularization ceofficient, and $\mathbf{w}$ represents the weights of leaf nodes.

4.4 Neural Network Ensemble

The outputs of the three models—SVR, Huber regression, and XGBoost—form the first and hidden layer of the neural network. These values are then utilized as inputs for the final layer and outputs a weighted combination of the model predictions:

$$\begin{aligned} \hat{y} &= \sigma(w_1 \cdot \text{SVR} + w_2 \cdot \text{Huber} + w_3 \cdot \text{XGBoost}), \\ &= \max\{0, w_1 \cdot \text{SVR} + w_2 \cdot \text{Huber} + w_3 \cdot \text{XGBoost}\} \end{aligned}$$

where σ is the ReLU activation function and $0 \leq w_1, w_2, w_3 \leq 1$ are learnable weights subject to $w_1 + w_2 + w_3 = 1$. This ensemble approach leverages the strengths of each model to maximize accuracy of the betting model, and the architecture is drawn below in Figure 1.

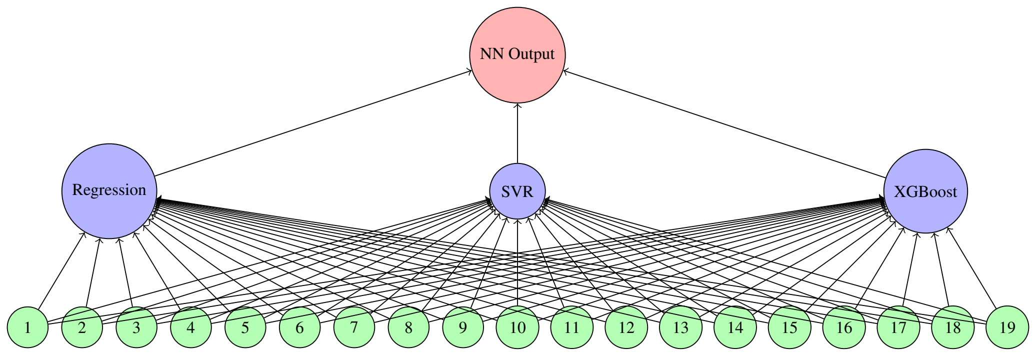

Figure 1: Diagram of the neural network architecture used for predicting NBA player rebounds. The input layer consists of the 19 features (see above for features), the first hidden layer includes outputs from regression, SVR, and XGBoost models, and the final output is the predicted rebound value.

5 Experiments, Results, and Discussion

5.1 Metrics and Evaluation Criteria

Despite the models being trained as continuous value regression ML models, the primary objective is predicting whether the actual number of rebounds will go over or under a given betting line, hence this can be viewed as a binary classification problem:

Accuracy (ACC): The percentage of bets correctly predicted (our primary classification metric) Formally, for N total predictions (bets),

$$\text{ACC} = \frac{\text{Number of correct predictions}}{N} \times 100\%.$$

Overall Betting Results: We calculate the net profit or loss from the betting strategy:

$$\text{Net Result} = [(\text{Model Accuracy} - \text{Accuracy}_{\text{break-even}}) \times N \times \text{bet size}]$$

where $\text{Accuracy}_{\text{break-even}} = 52.4\%$ is determined by the average break-even odds (-110 betting odds).

Regression Error Metrics: For evaluating the models' ability to predict the expected number of rebounds within a tight range, we utilized Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) metrics where y_i is the true value and $\hat{y}_i$ is the predicted value:

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|, \quad \text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}$$

5.2 Model Setup and Hyperparameters

Huber Regression: We set the Huber loss parameter $\delta = 3$. This threshold controls when the loss function transitions from quadratic to linear, thereby handling outliers while balancing robustness against outliers and model sensitivity to large residuals.

SVR: We fine-tuned the SVR with $\epsilon = 0.5$. This margin of tolerance defines how much error is acceptable before the model starts to penalize deviations. Setting $\epsilon = 0.5$ provided a good trade-off between ignoring minor noise and maintaining accurate predictions.

XGBoost: For the XGBoost model and SVR, we also used $C = 1$, a regularization parameter that balances model complexity and generalization. In addition, we ultimately selected a learning rate of 0.05 to balance convergence speed and generalization.

Neural Network Ensemble: Our neural network combines the predictions of each model using learned weights to assign different importance to each model's output. We employed a two-layer fully connected network with a learning rate of 0.01 (determined via grid search) and batch size of 32.

Calculating Edge: Using gradient descent (with a learning rate $\alpha = 0.001$) to maximize expected profit and maintain a high betting volume, we determined that a lower edge limit of 0.13 produced optimal results. We define the "edge" as the difference between the model's estimated probability of a particular outcome and the implied probability derived from the betting odds. This is quantified by first constructing a Gaussian distribution centered around the betting line. Then, I computed the cumulative distribution function (CDF) of the model's predicted outcome against this Gaussian distribution to calculate P(under) such that the resulting probability (where P(over) + P(under) = 1) provides a measure of the model's confidence relative to the betting line.

Training, Validation, and Cross-Validation: To ensure robust performance estimation, we employed a repeated k-fold cross-validation with $k = 5$. In each iteration, the dataset was randomly split into training and testing sets. The average results from these folds are reported. This approach helped mitigate the risk of overfitting and provided more stable performance estimates.

5.3 Results

Table 1 summarizes the metrics (accuracy, MAE, RMSE, and betting results) and Figure 2 showcases the confusion matrices associated with each model, revealing that although each model achieved accuracy well above the break-even point, the weighted NN Ensemble model outperformed and produced consistent positive results.

Table 1: Classification Performance Metrics (for edge ≥ 0.13, # bets $\in \{425 - 942\}$)

Model	Accuracy (%)	MAE (train)	RMSE (train)	MAE (test)	RMSE (test)	Betting Results (bet size = $100)
Huber Regression	54.11	2.20	2.86	2.18	2.84	+$726.75
SVR	53.12	2.04	2.70	2.10	2.76	+$588.24
XGBoost	53.08	1.76	2.26	2.16	2.8	+$640.56
NN Ensemble w/ Equal Weights	53.58	1.83	2.12	2.12	2.80	+$676.14
NN Ensemble w/ Optimal Weights	**56.93**	**1.42**	**1.89**	**1.66**	**2.19**	**+$2595.69**

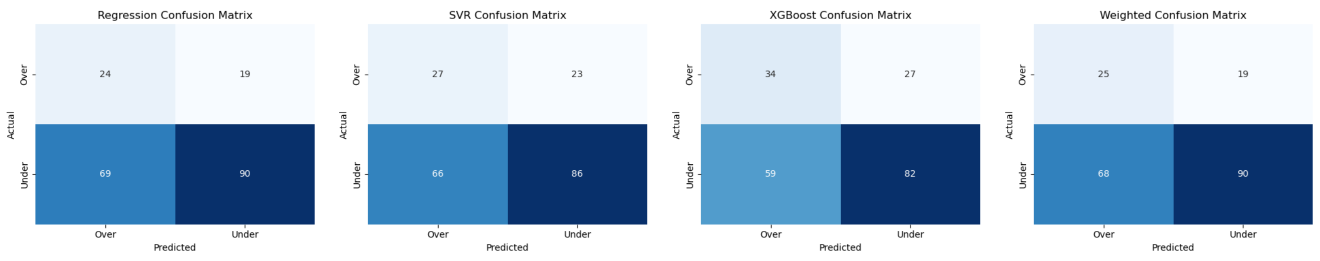

Figure 2: Confusion matrices of the 3 models and the Weighted NN Ensemble.

To highlight key trends, the results in Table 1 and Figures 2 and 3 portray the following results. First, we see the weighted NN Ensemble with optimal weights achieved the highest accuracy (56.93%), lowest MAE (1.66), and best betting results (+$2595.69), outperforming other models in both prediction and practical utility. Although the class distribution show bias towards "under" predictions due to imbalanced test data from the 2024-2025 NBA season games, although Figure 2 shows that regardless of the actual result, each model achieved an accuracy $\geq 52.4\%$ for both the over and under bets.

To understand this at a deeper level, I created heatmaps for each foundational model that showcases the accuracy based on predicted lines, which gives insight into how the model predicts for all players from bad rebounders to excellent rebounders as shown in Figure 3. The average density of these 3 density plots convey that the all models generally perform well for average rebounders (averaging 3-6 rebounds per game) but tend to slightly overestimate exceptionally poor rebounders (averaging 0-2 rebounds per game) while underestimating exceptionally great rebounders (averaging 9+ rebounds per game). This is shown by a higher density of predictions above the perfect prediction line for poor rebounders where the models skew towards predict higher rebounds than the true values. On the other end, the model does not generalize to outliers well as the models rarely predict rebounding numbers above 12.5 per game. However, there are certain unicorn players in the NBA who consistently have the capability to achieve this rebounding number on a nightly basis, although these players are few and far between. At around average rebounds per game, the model does extremely well to consistently predict a value very close to or at the actual value itself. Overall, the bias arises because the majority of players cluster near the mean rebound range, and the models optimize for the majority class. It would be possible to address these biases through weighted loss functions, oversampling outliers, or adding nuanced features.

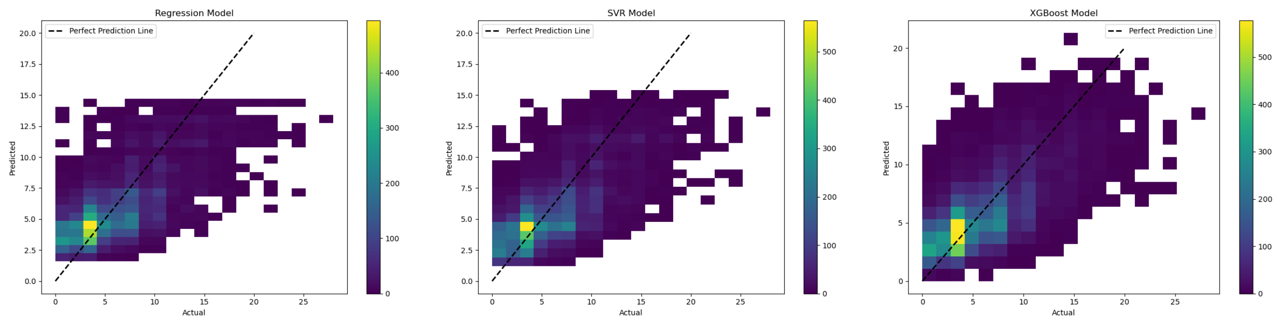

Figure 3: Heatmaps of the 3 Foundational models in their predictive accuracy

5.4 Discussion of Overfitting and Mitigation Strategies

I paid careful attention to avoid overfitting, yet in the end I believe that higher train and test error numbers could be attributed to moderate levels of underfitting the featured data. Regardless, the large and diverse dataset of tens of thousands of recent player NBA game data, combined with repeated cross-validation, helped ensure that models generalize well, as realized by the consistently positive betting results. Hyperparameter tuning prevented overly complex decision boundaries while early stopping and regularization in the neural network further reduced overfitting risks.

6 Conclusion / Future Work

In this study, we developed a predictive framework for NBA player rebounding performance based on historical data from the 2018-2021 seasons. By combining three regression-based models-Huber Regression, Support Vector Regression (SVR), and XGBoost-into a two-layer neural network ensemble, we were able to achieve a final accuracy of 56.93%, surpassing both the individual models and a simple equal-weighted ensemble. The results highlight that while each algorithm contributed unique insights, their performance was enhanced when their outputs were fused. The neural network's ability to learn an optimal linear combination of predictions allowed it to leverage the robustness of Huber Regression, the margin-based flexibility of SVR, and the non-linear complexity of XGBoost. By doing so, it alleviated individual model biases and led to a consistent edge in predictive accuracy and positive betting returns.

Despite these measures, some biases remained. The models perform best on "average" players, showing that the data distribution's skew affects performance. Future work with more computational resources could involve segmenting the data by player archetype or employing domain-specific data augmentation strategies, as I hypothesize that a global model may not understand the dynamics found at different player positions. Future work may include incorporating richer feature sets, such as player-tracking metrics (e.g., average distance from the rim, hustle statistics) or incorporating opponent defensive schemes and game contexts (e.g., back-to-back games, mid-season fatigue levels). Exploring more sophisticated ensemble architectures—such as attention-based layers or more extensive neural network depth—could further improve model calibration while integrating more granular data could enhance predictive power, sharpen betting strategies, and potentially generalize better to forecasting other key player performance metrics beyond rebounds.

7 Contributions

David Maemoto is the sole contributor of the entirety of the project.

References

Debasish Basak, Srimanta Pal, and Dipak Patranabis. 2007. Support vector regression. *Neural Information Processing – Letters and Reviews*, 11.

Matthew Byman. 2023. Building a statistical learning model for evaluation of nba players using player tracking data.

Tianqi Chen and Carlos Guestrin. 2016a. Xgboost: A scalable tree boosting system. pages 785–794.

Tianqi Chen and Carlos Guestrin. 2016b. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 785–794. ACM.

Kaan Gokcesu and Hakan Gokcesu. 2021. Generalized huber loss for robust learning and its efficient minimization for a robust statistics.

Charles R Harris, K Jarrod Millman, Stéfan J van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H van Kerkwijk, Matthew Brett, Allan Haldane, Jaime F Del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E Oliphant. 2020. Array programming with NumPy. *Nature*, 585(7825):357–362.

Motokazu Hojo, Keisuke Fujii, and Yoshinobu Kawahara. 2019. Analysis of factors predicting who obtains a ball in basketball rebounding situations. *International Journal of Performance Analysis in Sport*, 19(2):192–205.

J. D. Hunter. 2007. Matplotlib: A 2d graphics environment. *Computing in Science Engineering*, 9(3):90–95.

Jeff Johannsen. 2023. Nba betting. Accessed: 2024-11-03.

Justinas. 2020. Nba players data.

Nicholas Kiriazis. 2023. A bayesian two-stage framework for lineup independent assessment of individual rebounding ability in the nba.

Kyle Kom. 2023. Nba machine learning sports betting. Accessed: 2024-11-03.

Nathan Lauga. 2020. Nba games.

Wes McKinney. 2010. Data structures for statistical computing in python. *Proceedings of the 9th Python in Science Conference*, pages 51 – 56.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Arnau Turch Ferreres. 2022. Development of value metrics for specific basketball contexts: a positional approach for the defensive rebound value. B.S. thesis, Universitat Politècnica de Catalunya.

Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake Vand erPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. 2020. SciPy 1.0: Fundamental algorithms for scientific computing in python.

Junwen Wang. 2023. Predictive analysis of nba game outcomes through machine learning. pages 46–55.

Michael L. Waskom. 2021. Seaborn: statistical data visualization.

Kevin Wheeler. 2012. Predicting nba player performance.