NBA Player Salary Prediction

Vrisan Dubey
vdubey@ucsd.edu

ACM Reference Format:
Vrisan Dubey. 2024. NBA Player Salary Prediction. In . ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

The code and data used for this project can be found at this repository.

1 Introduction

Many NBA players make a lot of money. Even since the 1980's, NBA salaries are often found in the $1,000,000 range. However, obviously, some players are paid more than others, for the most part because they perform better in games than their peers. The focus of this project/paper is to use information about a player's performance (e.g. points, rebounds, assists, etc), attributes (e.g. height, weight, position, etc), and history to predict their salary.

2 Dataset

2.1 Identify Dataset

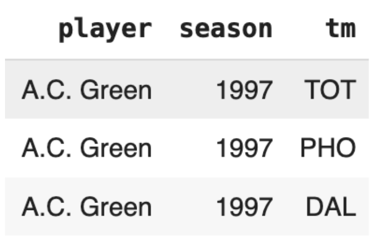
The data for this project is a collection of 4 different datasets.

1. The first dataset 'salaries.csv' is a dataset obtained from data.world, though the data was originally collected from basketball-reference.com. It contains the salary for NBA players from 1985 to 2018, with other columns that specify a player's id, the season the salary is from, and the team that the player played on that season. In total, it contains information about 14,163 salaries.
2. The second dataset 'players.csv' contains information about individual players. It is also taken from the same data.world directory as 'salaries.csv'. Found in this dataset are attributes like the name, height, weight, position, draft position, and more. It also contains a player_id that is a foreign key to the player_id in 'salaries.csv'. The dataset has 4685 rows - one per player.
3. The third dataset 'stats.csv' is a dataset from Kaggle.com, which was originally scraped from basketball-reference.com. It contains information about many player stats each season from 1945 to 2018. Since the salaries dataset starts only in 1985, we will not use the stats from the year prior. However, in total, it has the stats for 31814 players across all of these seasons.
4. The fourth dataset 'cpi.csv' is a dataset that has the Consumer Price Index (CPI) for each year since 1910. It uses 1983 as the base year. The price of market goods has not remained constant throughout the past decades, so we will have to adjust our salaries for inflation to keep predictions fair across time.

2.2 The Full Dataset

The process of creating the overall dataset is as follows:

1. First, we merged 'salaries.csv' and 'players.csv' on the foreign key player_id. This makes it so we have a player's salary and characteristics all in one dataframe. During this process, we also dropped columns we felt were irrelevant like birth year, birth place, high school, and hand dominance. This was just reasoned through domain knowledge. The important columns that were kept (as touched on before) were height, weight, position, name, and draft number. We also ended up dropping career stats like career points per game and career assists per game because we were predicting on a season-by-season basis.
2. Next, we had to clean up 'stats.csv'. This is because if a player was traded in the middle of the season, they would have multiple entries in the dataframe. An example of this is below:



player	season	tm
A.C. Green	1997	TOT
A.C. Green	1997	PHO
A.C. Green	1997	DAL

Figure 1: Subset of 'stats.csv' showing a player who played on multiple teams in one season

We wanted the dataframe to be indexable by (player, season) tuples. For example, there should only be one (Stephen Curry, 2017) entry, one (Lebron James, 2005) entry, and one (A.C. Green, 1997) entry. To make sure this is the case, we aggregated over (player, season)

Conference'17, July 2017, Washington, DC, USA
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

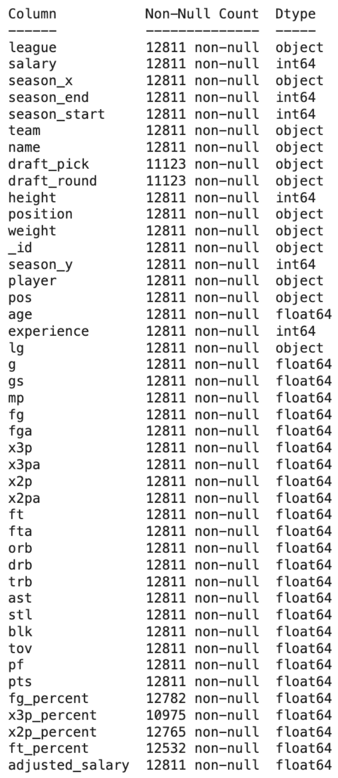
and summed, averaged or applied mode to all stats, depending on the stat.
After this is done, we merge this with the output from step 1.

3. Lastly, we factor in the CPI. This is done with the following calculation:

$$Salary_{real} = \frac{Salary_{nominal}}{CPI} * 100$$

This gives us a final column called adjusted salary, which we will predict with our model.

Our final dataset has these properties:



```
Column          Non-Null Count  Dtype
------          --------------  -----
league          12811 non-null  object
salary          12811 non-null  int64
season_x        12811 non-null  object
season_end      12811 non-null  int64
season_start    12811 non-null  int64
team            12811 non-null  object
name            12811 non-null  object
draft_pick      11123 non-null  object
draft_round     11123 non-null  object
height          12811 non-null  int64
position        12811 non-null  object
weight          12811 non-null  object
_id             12811 non-null  object
season_y        12811 non-null  int64
player          12811 non-null  object
pos             12811 non-null  object
age             12811 non-null  float64
experience      12811 non-null  int64
lg              12811 non-null  object
g               12811 non-null  float64
gs              12811 non-null  float64
mp              12811 non-null  float64
fg              12811 non-null  float64
fga             12811 non-null  float64
x3p             12811 non-null  float64
x3pa            12811 non-null  float64
x2p             12811 non-null  float64
x2pa            12811 non-null  float64
ft              12811 non-null  float64
fta             12811 non-null  float64
orb             12811 non-null  float64
drb             12811 non-null  float64
trb             12811 non-null  float64
ast             12811 non-null  float64
stl             12811 non-null  float64
blk             12811 non-null  float64
tov             12811 non-null  float64
pf              12811 non-null  float64
pts             12811 non-null  float64
fg_percent      12782 non-null  float64
x3p_percent     10975 non-null  float64
x2p_percent     12765 non-null  float64
ft_percent      12532 non-null  float64
adjusted_salary 12811 non-null  float64
```

Figure 2: Final Dataset Properties

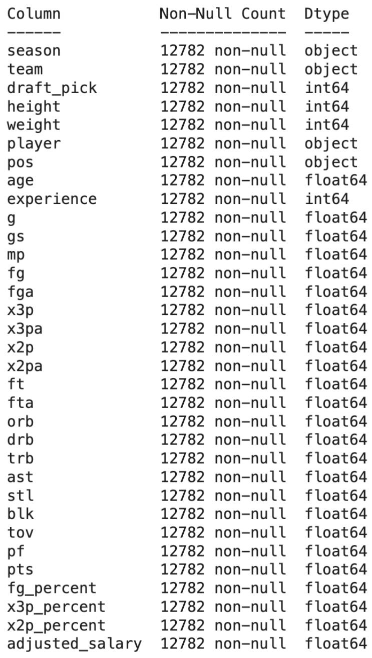
2.3 Data Cleaning

There were missing values in this dataset, which were primarily in columns that had to do with percentages. For example, Field Goal Percentage, 3 Point Percentage, 2 Point Percentage, etc all had missing values. This was due to the fact that some players went 0/0 in these categories so the percentage was undefined. To get rid of these missing values, we simply imputed 0 because if a player never shot any of these shots, he probably was not good enough to shoot them (or wasn't even able to be on the court enough to attempt them). Thus, we decided to give them the lowest value possible. However, if a player had a NaN in Field Goal Percentage, we just dropped that instance all together. This is because that just means that player did not take a shot all season. All of this was reasoned primarily through domain knowledge.

Other than that, columns like draft pick, which say the pick that the player was drafted at, were stored as strings like '1st overall' or '22nd overall'. We just extracted the number from these strings. We also noticed NaN's in this column, which occurred due to the fact that many NBA players go undrafted. We decided to give these players draft pick 61 since the last draft pick that a drafted player can have is 60.

Lastly, columns that contained minimal meaning or duplicate information were dropped. For example, 'season_end' and 'season_start' contain information which is already in the overall 'season' column. All of the information in column 'draft_round' is contained in 'draft_pick' since if draft pick is greater than 30, they were 2nd Round, else they were 1st round.

Our cleaned dataset has these properties:



```
Column          Non-Null Count  Dtype
------          --------------  -----
season          12782 non-null  object
team            12782 non-null  object
draft_pick      12782 non-null  int64
height          12782 non-null  int64
weight          12782 non-null  int64
player          12782 non-null  object
pos             12782 non-null  object
age             12782 non-null  float64
experience      12782 non-null  int64
g               12782 non-null  float64
gs              12782 non-null  float64
mp              12782 non-null  float64
fg              12782 non-null  float64
fga             12782 non-null  float64
x3p             12782 non-null  float64
x3pa            12782 non-null  float64
x2p             12782 non-null  float64
x2pa            12782 non-null  float64
ft              12782 non-null  float64
fta             12782 non-null  float64
orb             12782 non-null  float64
drb             12782 non-null  float64
trb             12782 non-null  float64
ast             12782 non-null  float64
stl             12782 non-null  float64
blk             12782 non-null  float64
tov             12782 non-null  float64
pf              12782 non-null  float64
pts             12782 non-null  float64
fg_percent      12782 non-null  float64
x3p_percent     12782 non-null  float64
x2p_percent     12782 non-null  float64
adjusted_salary 12782 non-null  float64
```

Figure 3: Cleaned Dataset Properties

2.4 EDA

adjusted salary is the predicted variable in this project (Note that 'salary' will always refer to 'adjusted salary' for the rest of this paper unless otherwise specified). Its distribution is shown below:

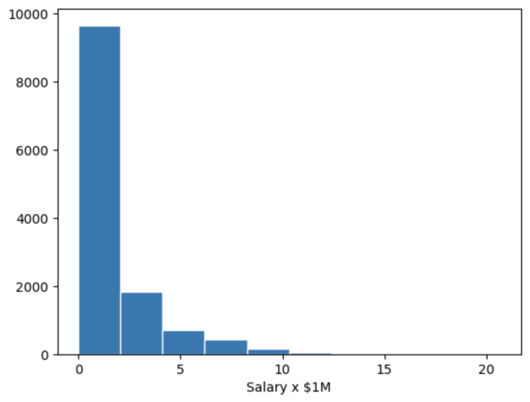


Figure 4: Distribution of Adjusted Salary

We can see a right-skewed distribution, largely because the amount of players who are much better than average throughout history is very small. There are a lot of players who make a small amount of money and then leave the NBA because their career did not work out (We just never hear about these players!)

Season is also an important variable to consider. It tells us what season (year) the salary is from. Below is the distribution of adjusted salaries by decade.

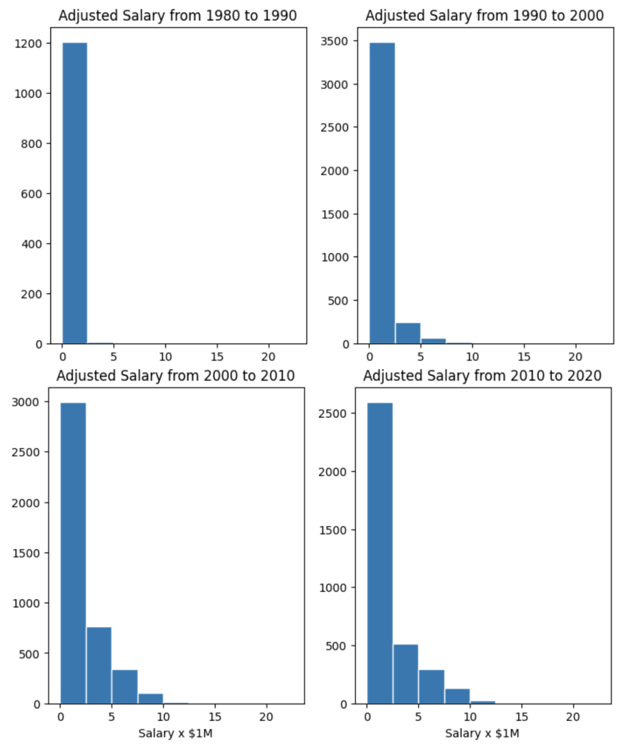


Figure 5: Distribution of Salaries by Decade

And the median salary of players by season:

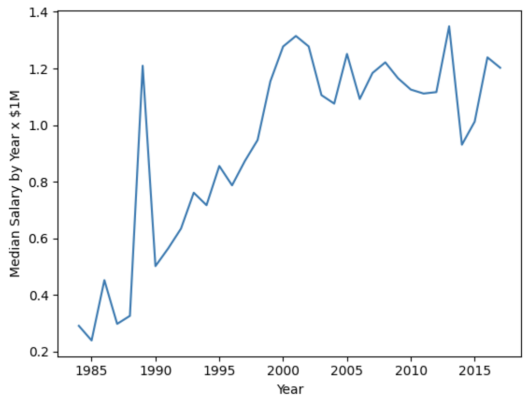


Figure 6: Median Salary by Season

We can see from both figures that even though we already adjusted for inflation, as time moves forward, players become more and more expensive. This makes sense, as almost every offseason, a player becomes the highest paid player in NBA history. In the 1980's, a player never even had a salary of more than $2.5M. Last offseason, Boston Celtics' Jaylen Brown signed a contract worth $60.8M, making him the highest paid player of all time. The NBA and its teams simply have more of an audience and more money.

There are many concrete stats and characteristics in this dataset, each with their own relationship to salary. For the purposes of this paper, we will show plots for **points**, **rebounds**, and **assists**, which for the most part are the 3 stats most commonly looked at in the NBA.

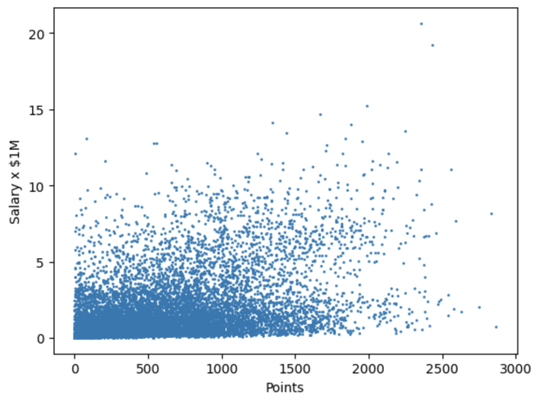


Figure 7: Total Points vs Salary

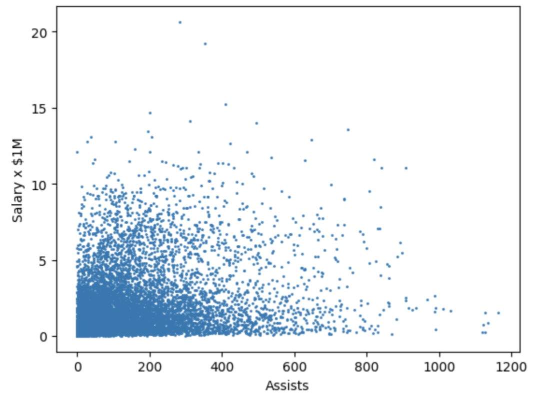


Figure 8: Total Assists vs Salary

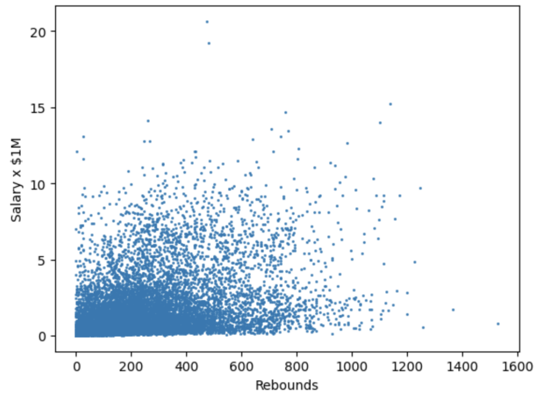


Figure 9: Total Rebounds vs Salary

For points and rebounds, we can see a more clear increasing relationship with salary. For assists, it is not as clear.

However, for a stat like **total fouls**, there may not be any relationship at all.

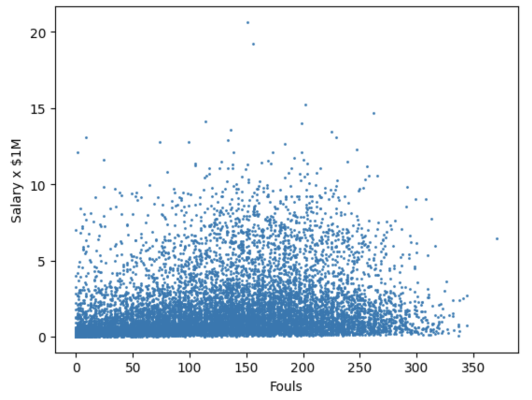


Figure 9: Total Fouls vs Salary

The relationship above for fouls could be explained by that if a player is on the court a lot, they are prone to more chances to foul (which would indicate a positive relationship). But by nature, fouling a lot is a bad thing in basketball (which would indicate a negative relationship).

Lastly, we can examine the variable **position**. In the NBA, there are 5 typical positions: Point Guard (PG), Shooting Guard (SG), Small Forward (SF), Power Forward (PF), and Center (C). These are listed from shortest to tallest, so typically the point guard is shortest on the court and the center is the tallest.

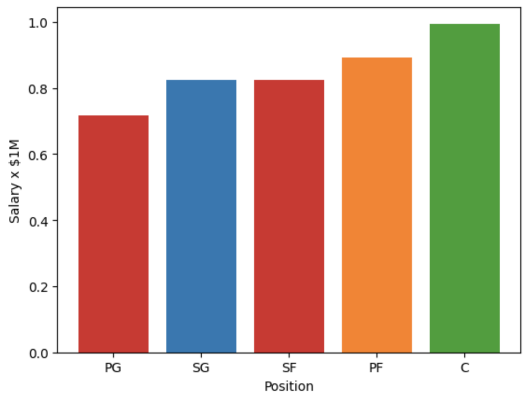


Figure 10: Median Salary by Position

We can see that as a player gets taller their salary, on average, will increase. However, it is worth noting something interesting that even though we see a positive correlation between points and salary, centers (who make the most on average) do not score the most. They actually score the least. See below:

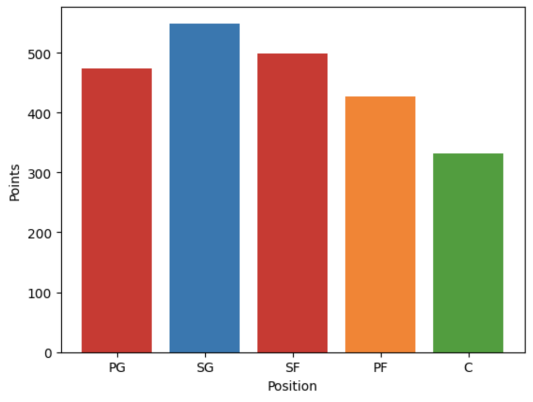


Figure 10: Median Points by Position

The spike at SG's can probably be explained that many of the notorious, best scorers in the NBA of all time play SG (Kobe Bryant and Michael Jordan, for example).

Even if we add up points, rebounds, and assists for each player, we will still find that centers perform the worst, statwise.

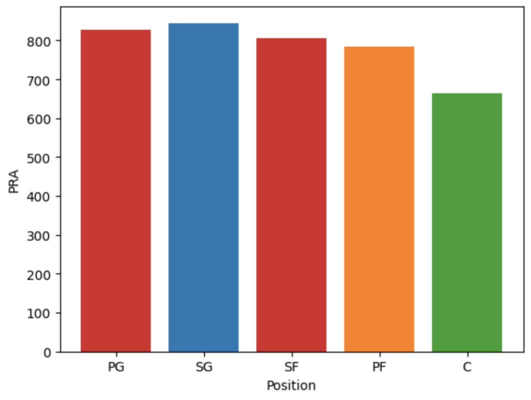
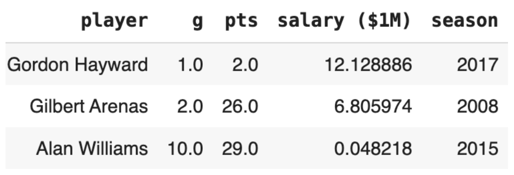


Figure 11: Median PRA by Position

This immediately drove the belief that there were unobvious trends in the data. The first and most direct conclusion was that salary is based off much more than these simple stats.

2.4.1 Outlier Analysis. The main type of outliers in this dataset occurs when a player, who normally performs well, has a dip in performance due to something outside their basketball capabilities. Below are three different rows of the dataset.



player	g	pts	salary ($1M)	season
Gordon Hayward	1.0	2.0	12.128886	2017
Gilbert Arenas	2.0	26.0	6.805974	2008
Alan Williams	10.0	29.0	0.048218	2015

Figure 12: Outlier Example

In the figure above, we see three players, the first two who make much more money than the third. However, from column 'g', we can see that Gordon Hayward, in 2017, played only 1 game. Gilbert Areanas in 2008 played only 2. In 2017, in the first quarter of the first game, Hayward suffered a catastrophic ankle injury that sidelined him for the rest of the year. Arenas suffered a knee injury in 2008 that also ended his season. However, Alan Williams, compared to the other two players, is just a lower-caliber player. He doesn't play much regardless, so he's paid less. Based on our data, it is not super easy to tell the difference between a Gordon Hayward season vs an Alan Williams season off the stats alone. Without the knowledge of Hayward and Arenas, we could very easily think they are Alan Williams-esque players, which is not the case. There is not a column saying why a player performed the way that they did, so it becomes difficult to pick and choose which rows are valid without going through 14,000 instances and evaluating each. We end up have to keep all these instances, as a result.

3 Predictive Task

3.1 Explicit Task

As alluded to throughout this paper, the predictive task is as follows: Given data about an NBA player's characteristics and performance over one season, predict their salary (adjusted for inflation). Since salary is a continuous variable, this is a regression problem.

3.2 Evaluation

All the models used will be evaluated with the root mean squared error (RMSE) defined as follows, $\sqrt{\frac{1}{n}\sum_i (y_{pred}^{(i)} - y^{(i)})}$ where $y^{(i)}$ refers to the salary of the i-th player, $y_{pred}^{(i)}$ is the model's prediction of that player, and n is the number of datapoints. Since salaries are normally in the millions of dollars, the RMSE's will be in that range also. In order to grasp the RMSE values visually, the errors will be displayed divided by 1,000,000. An RMSE of 3.2 can be interpreted that on average, for each datapoint, the model is predicts a salary that is off by $3.2M dollars.

For each model, three metrics will be presented. Two will be the RMSE on the train set and a holdout test set that was separated from the data prior to any modeling. The other will be the average RMSE from the results of a 5-fold cross validation on the training set.

3.3 Baseline Models

Two baseline models were used in this project 1. LinearRegression 2. DecisionTreeRegressor.

1. LinearRegression (sklearn): A linear regression model assumes that the output is a linear combination of its features. It tries to learn a weight vector $\vec{w}$ such that $y_{pred}^{(i)} = \vec{w} \cdot \vec{x}^{(i)}$ will minimize the RMSE for the dataset. The main limitation with this model is that there may not be a linear relationship between each feature and the output. In that case, a simple linear model (assuming no kernel function) would underfit the data. Also, without feature engineering, LinearRegression cannot handle categorical variables.
2. DecisionTreeRegressor (sklearn): This model can fit to any degree of complexity present in the data. However, because of this, decision tree can tend to overfit to the training data (provided minimal hyperparameter tuning is done). This can result in arbitrarily big models that do not generalize well. Also, as with the LinearRegression model, without feature engineering, the DecisionTreeRegressor cannot handle categorical variables

Since these are baseline models, no feature engineering was done (i.e. no one-hot encoding of categorical variables). The models were limited to all numerical features and so all variables of type *object* in **Figure 3** were dropped for the baseline.

3.4 Baseline Results

	LinearRegression	DecisionTreeRegressor
Train RMSE	1.451	0.203
Test RMSE	1.439	1.769
CV RMSE	1.455	1.801

	DecisionTreeRegressor(*max_depth*=5)
Train RMSE	1.437
Test RMSE	1.529
CV RMSE	1.493

Note that the RMSE's, as mentioned before, are divided by 1,000,000. So, for example, the RMSE of LinearRegression for the training set was $1,451,000.

3.5 Baseline Interpretation

The LinearRegression model generalizes well as the three errors are roughly equal, which is expected. However, many features from the original dataset were left out in its training, so the idea is that it can be improved.

For the DecisionTreeRegressor, the overfitting issue is very apparent when the *max_depth* parameter is not set as the training error is very low and the test and cross-validation error are significantly higher. When the *max_depth* is set, the regressor achieves similar performance to LinearRegression. Boosting and bagging are techniques that can be used to reduce the error and variance of tree based models, respectively. Incorporating either or both of the techniques become a clear path for improvement.

4 Model

The model that we ended up using as our final predictor was a LightGBM Regressor. As mentioned in **Section 3.5**, a clear path for improvement for the tree-based models was to add bagging and/or boosting. LightGBM is a gradient-boosting framework that also employs bagging, so it made sense theoretically that this choice could improve the baseline.

The other potential route for improvement involved using a kernel linear regression technique since ordinary linear regression cannot capture non-linear relationships. The thought was to use an RBF kernel to map the features into an infinite dimension space and calculate a linear relationship there. This was attempted through sklearn's KernelRidge Regressor. However, as will be mentioned in the results, this actually performed worse than the baseline.

4.1 Feature Engineering

There were two main subsets of features engineered for the model: Ranks and Per-game stats. Note that LightGBM has support for categorical features so we did not have to one-hot encode such features.

4.1.1 Ranks.

The idea for 'Ranks' is to order the players based on some stats during the respective season. An example is shown using the fathom data below.

	Season	Pos	Pts
James	2022	SF	20
Curry	2022	PG	30
Durant	2022	SF	10
Jokic	2017	C	5

So if we rank Pts by Season, for 2022, Curry would be 1, James would be 2, and Durant would be 3. Since the row for Jokic is from 2017 and no other row is from 2017, Jokic would be 1 for that year.

We can also group even further. Instead of grouping by Season, we can group by both Season AND Pos. This would tell that, for example, for all PFs in 2022, who scored the most. For the above example, Curry would be 1; James would be 1 (only PG in 2022), Durant would be 2 (since they share the same season and position and James's Pts > Durant's Pts); and Jokic would be 1.

All in all, we generated ranks for points, minutes, age, experience, games, games started, rebounds, assists, rebounds, field goals made, field goals attempted, and minutes played, each grouped by 1. season 2. season AND position 3. season AND team. This is 12 stats * 3 separate group criteria = 36 new features.

4.1.2 Per-Game Stats.

This class of features is much simpler. All of the stats in our data were total stats. So the variable Pts refers to the total amount of points a player scored during the entirety of that season. Dividing the total stat by the number of games that player player gives a per-game interpretation of that stat. For example, for Pts, we get points-per-game (ppg) for each player, which is probably the most looked-at stat in basketball. For games started, we get the fraction of games started for that player. Typically, better players start games in the NBA. The reason total stats can be misleading is if a good player gets injured during the season, their total stats will be low, but their per-game stats will not falter.

We did the per-game calculations for points, assists, rebounds, minutes played, field goals attempted, field goals, and games started. We found that blindly adding all the per-game stats lowered the model's performance, so we only kept those that the model found meaning in.

4.2 Model Optimization

We found that many of the default parameters for the LightGBM Regressor worked best for our tasks. However, we ran a Grid Search on the *max_depth* and *num_leaves* parameters. These are two parameters, specified in the LightGBM documentation, that help control overfitting. We ended with *num_leaves*=50 and *max_depth*=7. We also added feature subsets one by one to examine the impact that different types of features had on the model. The results of these optimizations will be discussed in the **Results** section.

We also ended up dropping any column related to percentage, for example 3pt percentage, free throw percentage, field goal percentage, etc. This is because a player can just take 1 3-pointer all season, make it, and then boast a 100% 3pt percentage. Using these feature did not make sense intuitively as we felt any information given by these features was better captured in per-game and total stats.

5 Literature

NBA Player Salary Analysis based on Multivariate Regression Analysis by Xuanhao Feng et al

Feng et al's carried out a project similar to what is laid out in this paper. They used NBA player stats to predict salaries. They collected their data from basketball-reference.com. However, the main differences boil down to the nature of their data and their methods used.

The data used in their model was 500 players' stats from the 2020-21 and 2021-22 seasons. After removing instances with missing values, they were left with around 330 instances. Their data also included many more statistics than ours. Whereas we only had only basic statistics like points, rebounds, assists, etc, they also had stats that are considered "advanced statistics".

Their methods were much simpler than what we used. They started with regular linear regression (OLS) and then added regularization in the form of Ridge Regression, Lasso Regression, and Elastic-Net. With these methods they were able to achieve an RMSE of around $0.5M.

Though both of projects are similar task-wise, the difference in data makes the results and methods tough to replicate. Our data comes from multiple decades while their's comes from two years. How basketball monetarily and structurally evolves over time is something that our models will be forced to capture. Also our dataset contains instances for many more players, which introduces outliers and more inexplicable instances. However, understanding that simple models like regularized linear regression have been proven to perform well for this task gave a good indication that we had a solid baseline.

Classification of NBA Salaries through Player Statistics by Li et al

This paper, written by UC Berkeley Sports Analytics Group, describes a classification model used to cluster NBA players into salary groups. They also get their data from basketball-reference.com - data from 2017-18 to present. They, similar to the previous paper, use advanced stats in their model rather than ordinary stats. They actually use only four features in their model, which were chosen by examining the correlation between these features and salary. Their main goal was not necessarily to perfectly classify each player but give a theoretical projection of a players salary class and compare that to what actually happened in real life. For example, if a player was in the highest salary group in real life but the model predicted that player to be in the lowest group, they say that player is drastically overpaid. They used a K-Nearest-Neighbors technique for their classification.

This project's feature selection was very unique in that they dropped many features that did not have a super strong correlation with salary. We learned from this that understanding most of the variables and their relationship with our output variable was extremely paramount in our overall feature engineering process. Since our projects do not align much from an objective perspective, much of the model development insights were not extremely useful, but we were definitely able to learn from their EDA and use of features.

6 Results

6.1 Comparison

The final LightGBM Regressor with the hyperparameter tuning and feature engineering done outperformed both baseline models. It gave a test RMSE of $1.037M and a validation (CV) RMSE of $1.014M. This means that on average, it was able to predict the salary of an NBA player within around $1M. We thought the improvement from the baseline was vastly significant because we saw a 30.9% decrease in CV RMSE and a 27.9% decrease in test RMSE (compared to the linear regression baseline since it performed better than the decision tree).

Below is a table that shows the effect of adding different features and the result of the hyperparameter tuning. (Note that feature selection below refers to inclusion of categorical variables and season, exclusion of percentages)

	CV RMSE
Linear Regression + baseline feats	1.455
DTR + baseline feats	1.801
DTR + baseline feats + max_depth=5	1.495
LGB + baseline feats	1.043
LGB + feat selection + ranks	1.052
LGB + feat selection + per-game stats	1.027
LGB + feat selection + per-game stats + ranks	1.021
LGB + max_depth=7, num_leaves=50	1.014

We can see that some of the feature subsets do not work well by themselves but work better when complemented with other features. For example, the rank features that we engineered, when added by themselves, hurt the model so badly that it would have been better not to even have them. However, when coupled with the per-game stats, the ranks improve the model. Overall, the addition of new features were not as effective when compared to changing the model itself, but they did help lower the RMSE nonetheless.

We touched on using kernel-based regression earlier in the paper. We tried to fit a Kernel Ridge Regression model; however With it, we could not get under $2M RMSE for any combinations of kernel types, regularization factors, and other hyperparameters. This was even worse than the baseline. Because of this, we decided that it probably just was not suited well for our data and moved forward with the LightGBM since it showed sufficient improvement.

6.2 Feature Importance

Below is the plot of the 25 most important features from the final LightGBM model.

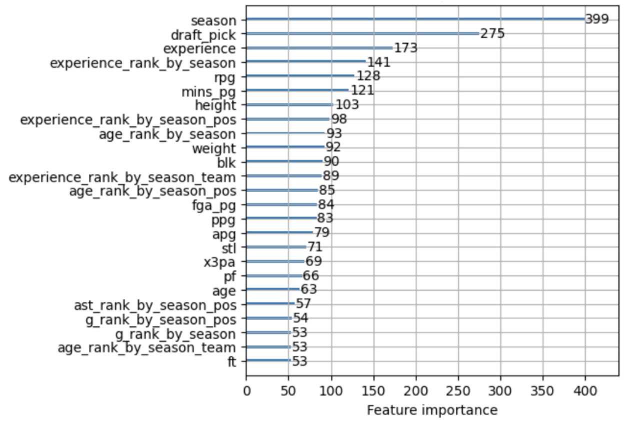


Figure 14: Final Model 25 Most Important Features

The model places a heavy emphasis on the season. This is expected since in **EDA**, we saw an increase in salary as years moved forward. We can also see some of the features that we engineering like 'experience_rank_by_season' (the rank of each player by their experience grouped by season), 'mins_pg' (minutes per-game), 'age_rank_by_season_pos' (the rank of each player by their age grouped by season AND position), and 'ppg', to name some. This plot shows that many of the engineered features helped and were vital in the prediction process. However, there were a lot of engineered features that did not help as much. For example, the least important feature of all was 'pts_rank_by_season_pos' (rank of each player by total points grouped by season AND position). We thought this feature would hold more value intuitively, but the model clearly did not agree.

6.3 Discussion

6.3.1 Assessing the RMSE. As mentioned before, the RMSE for the test set was $1.037M. On a surface level, this seems pretty high given that the median salary in the training set was <$1M. Blindly interpreting this number can lead us to believe that for 50% of the data, we are off, on average, by more than double a player's salary. However, if we break the test set down into players with low, medium, and high salary, we get a better picture of what is going on.

	Test RMSE
salary < $1M	0.572
$1M < salary < $3M	0.881
salary > $3M	2.095

What we can see in the above table is that from an RMSE perspective, the model is much more accurate on lower salary players than it is on higher salary players. Actually around 62% of the squared error comes from the 15% of the data that makes up the 'salary > $3M' group.

If we deviate away from RMSE a little, we can take another perspective on the results. Define Absolute Relative Error (ARE) as $\frac{|y_{pred}^{(i)} - y^{(i)}|}{y^{(i)}}$ and the Mean ARE as the average relative error over the whole dataset. This is motivated by the notion that if a player makes $30M, predicting $31M is much less alarming than if a player made $1M and we predicted double their salary at $2M (a $1M difference in both cases). This is inherent in the relative error formula as in the first case, we get a relative error of $\frac{1}{30}$ whereas in the second case, we get $\frac{1}{2}$. Basically, the error penalizes wrong predictions much more when the actual salary is low. The below table shows the Mean ARE and the Median ARE for the whole dataset and the salary cutoffs used above.

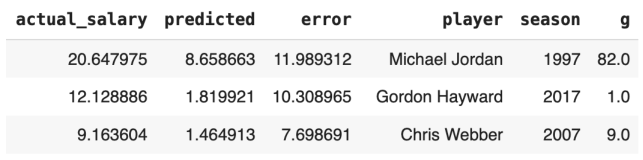
	Mean ARE	Median ARE
entire test set	1.050	0.305
salary < $1M	1.632	0.426
$1M < salary < $3M	0.342	0.237
salary > $3M	0.282	0.230

Looking at the Mean ARE column, this time we see somewhat an opposite story as the RMSE. This time, most of the ARE comes from the low salary group. But this is expected due to the nature of ARE. On the other hand, both the mean and median ARE show that our model is not as bad on the higher salary players as the RMSE shows. The model is normally within around 20% of the actual salary for these players, which is around the same as its performance on the medium earning players and better than the lower earning players.

To wrap up this discussion, the model's predictions on high salary players are, in absolute sense, the most wrong (as shown by the RMSE). However, this is balanced out by the fact that the salaries themselves are high so the large errors are relatively not very significant (from the ARE). For the lower salary predictions, the RMSE tells us the predictions are the most accurate. In an absolute sense, this is correct, but since the salaries are already small, the errors become more prominent from a relative perspective. The model's performance on medium salary players is, when compared to the other groups, better and more consistent.

A future iteration of this project would look into ways to minimizing the discrepancies among salary groups in both the RMSE and ARE cases.

6.3.2 Prediction Analysis. Taking a look at the holdout test set, we can see where the model performed the worst. The three predictions that were the most off are below:



actual_salary	predicted	error	player	season	g
20.647975	8.658663	11.989312	Michael Jordan	1997	82.0
12.128886	1.819921	10.308965	Gordon Hayward	2017	1.0
9.163604	1.464913	7.698691	Chris Webber	2007	9.0

Figure 15: Most Incorrect Predictions

It is extremely interesting that the most wrong prediction is for Michael Jordan, a player many consider as the best of all time. The reason for this is that in 1997, the salary at the 75th percentile was $1.86M. That means that Jordan was making almost 10x the money as the player at the 75th percentile. Even moreso, the player at the 98th percentile was making only $6.54M, which is around 3 times less than Jordan's. With the model's great emphasis on the season, the prediction explains itself.

We already discussed the outlier nature of Gordon Hayward's 2017 season. The model has no way of knowing that he injured himself in the first game of the season and never played during that season again. For Chris Webber, we can see the low amount of games also, but his situation that year is not as explainable.

If we remove, from the test set, that Michael Jordan instance and other instances where a player did not play more than 5 games in a specific season, the test RMSE drops from 1.037 to 0.99.

On another note, see the plot below of error by season.

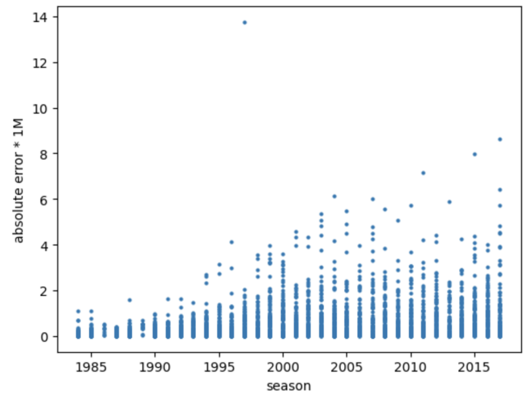


Figure 16: Prediction Error by Season

Here, the plot shows that predictions for more recent seasons tend to be worse than for predictions for seasons way in the past. The variance for predictions in the 1980s is much less than the variance in the 2010s. We see a lot more instances where the model was way off. (Also we can see the Michael Jordan 1997 prediction way up on the top of the plot). An idea for improvement is to train a model separately on separate decades. The NBA game has evolved over time and maybe separate models will be able to find different weightage to different features as the game has changed. For example, in today's game, 3pt shooting is much more prevalent than it was in past decades. In previous decades, centers used to dominate the sports; nowadays, though there still are many dominant centers, many teams are guard-oriented. Shorter players have much more profound roles in today's game (think Steph Curry). Furthermore, as mentioned in the **EDA** portion, players are just paid in much higher magnitudes than before. By looking at a single time period only, a model may be able to better learn these nuanced changes that occur over time.

6.4 Overall Takeaways

1. The year of the season of the salary is extremely important even after the salaries have been adjusted for inflation. 'Season' as a feature runs away as the most important feature.
2. Our model pretty significantly outperforms the baseline. Its performance on the test/validation sets is up to interpretation. The accuracy of predictions vary depending on the magnitude of the actual salary.
3. The change in model type from the baseline seems to be the cause of the greatest increase in accuracy. The feature selection, feature engineering, and hyperparameter tuning processes, though providing a slight boost, did not increase performance as drastically.
4. Future editions of this project will focus on how to make results more consistent across players of different types. This could involve training many more models - each which are specialized on different subcategories of the data. This comes to mind because other projects (as mentioned in the **Literature** section) were able to achieve pretty solid RMSE when working on small amounts of data from select seasons.

7 References

(1) Zhang, J. 2024. National Basketball Association Salary Prediction: A Data-Driven Linear Regression Analysis. Highlights in Business, Economics and Management. 24, (Jan. 2024), 1059–1064. DOI:https://doi.org/10.54097/c966f539.

(2) Li, R., Wu, W., Feng, K., Sengupta, K., & Cheng, A. (2018). Classification of NBA salaries through player statistics. Sports Analytics Group at Berkeley, https://sportsanalytics.berkeley.edu/projects/nba-salaries-stats.pdf [19.09. 2020]

(3) Wen, Riguang (2017). NBA data. figshare. Dataset. https://doi.org/10.6084/m9.figshare.5414170.v1

(4) Salaries Dataset: https://data.world/datadavis/nba-salaries

(5) CPI Data: https://www.minneapolisfed.org/about-us/monetary-policy/inflation-calculator/consumer-price-index-1913-