

CS6220 Final Report
NBA Salary Projection
Samuel, Lincoln, Randy, Simin
May 5, 2022

1. Introduction

Stephen Curry, LeBron James, and Kevin Love are just some of the biggest names in the NBA world. Raking millions of dollars from salaries as well as endorsement deals, they have become some of the richest athletes in the world. They are the inspiration for many young kids, and who young athletes aspire to be. Nevertheless, not every star began their journey making millions of dollars. Many started on the lower end of the salary scale because they simply do not know what they are worth. That is why we chose to create the NBA Salary projection website for our final project in Data Mining class.

The purpose of this website is to help basketball players who are currently acting as free agents estimate how much they are worth. The yearly salary is estimated based on the player's statistics and performance in the field. Sometimes, younger players have no idea of how much they are worth, so by having this website, we hope that they can estimate what their potential salary is going to be and they are better-informed regarding which of their stats could realistically be improved so they can better negotiate their salary.

This is an interesting application of data science because we are trying to make a projection of how much a player can potentially make. While an athlete's success can be determined by factors beyond their stats and there are situations that we can not predict, we hope to use their measurable metrics and statistics to make a forecast of how much an NBA could potentially make, or to determine if they are being overpaid or underpaid relative to their peers. This can also be a useful tool for players to determine what skills they should focus on improving during training (i.e. shoot better from 3, play better defense, score more points, etc.) to meet their salary goals for their career.

2. Dataset

We found two datasets (player_stats.csv and player_salaries.csv) from the NBA reference website.

1). player_stats.csv (Figure 1)

			player_stats																													
	Rk	Player	Pos	Age	Tm	G	GS	MP	FG	FGA	FG%	3P	3PA	3P%	2P	2PA	2P%	eFG%	FT	FTA	FT%	ORB	DRB	TRB	AST	STL	BLK	TOV	PF	PTS	Year	
0	1	Álex Abrines	SG	23	OKC	68	6	15.5	2.0	5.0	0.393	1.4	3.6	0.381	0.6	1.4	0.426	0.531	0.6	0.7	0.898	0.3	1.0	1.3	0.6	0.5	0.1	0.5	1.7	6.0	2017	
1	2	Quincy Acy	PF	26	TOT	38	1	14.7	1.8	4.5	0.412	1.0	2.4	0.411	0.9	2.1	0.413	0.521	1.2	1.6	0.750	0.5	2.5	3.0	0.5	0.4	0.4	0.6	1.8	5.8	2017	
2	2	Quincy Acy	PF	26	DAL	6	0	8.0	0.8	2.8	0.294	0.2	1.2	0.143	0.7	1.7	0.400	0.324	0.3	0.5	0.667	0.3	1.0	1.3	0.0	0.0	0.0	0.3	1.5	2.2	2017	
3	2	Quincy Acy	PF	26	BRK	32	1	15.9	2.0	4.8	0.425	1.1	2.6	0.434	0.9	2.2	0.414	0.542	1.3	1.8	0.754	0.6	2.8	3.3	0.6	0.4	0.5	0.6	1.8	6.5	2017	
4	3	Steven Adams	C	23	OKC	80	80	29.9	4.7	8.2	0.571	0.0	0.0	0.000	4.7	8.2	0.572	0.571	2.0	3.2	0.611	3.5	4.2	7.7	1.1	1.1	1.0	1.8	2.4	11.3	2017	
5	4	Aron Afflalo	SG	31	SAC	61	45	25.9	3.0	6.9	0.440	1.0	2.5	0.411	2.0	4.4	0.457	0.514	1.4	1.5	0.892	0.1	1.9	2.0	1.3	0.3	0.1	0.7	1.7	8.4	2017	
6	5	Alexis Ajinça	C	28	NOP	39	15	15.0	2.3	4.6	0.500	0.0	0.1	0.000	2.3	4.5	0.511	0.500	0.7	1.0	0.725	1.2	3.4	4.5	0.3	0.5	0.6	0.8	2.0	5.3	2017	
7	6	Cole Aldrich	C	28	MIN	62	0	8.6	0.7	1.4	0.523	0.0	0.0	0.7	1.4	0.523	0.523	0.2	0.4	0.682	0.8	1.7	2.5	0.4	0.4	0.4	0.3	1.4	1.7	2017		
8	7	LaMarcus Aldridge	PF	31	SAS	72	72	32.4	6.9	14.6	0.477	0.3	0.8	0.411	6.6	13.8	0.480	0.488	3.1	3.8	0.812	2.4	4.9	7.3	1.9	0.6	1.2	1.4	2.2	17.3	2017	
9	8	Lavoy Allen	PF	27	IND	61	5	14.3	1.3	2.8	0.458	0.0	0.0	0.000	1.3	2.7	0.461	0.458	0.4	0.5	0.697	1.7	1.9	3.6	0.9	0.3	0.4	0.5	1.3	2.9	2017	
10	9	Tony Allen	SG	35	MEM	71	66	27.0	3.9	8.4	0.461	0.2	0.8	0.278	3.6	7.6	0.479	0.473	1.1	1.8	0.615	2.3	3.2	5.5	1.4	1.6	0.4	1.4	2.5	9.1	2017	
11	10	Al-Farouq Aminu	PF	26	POR	61	25	29.1	3.0	7.6	0.393	1.1	3.5	0.330	1.9	4.2	0.445	0.468	1.6	2.2	0.706	1.3	6.1	7.4	1.6	1.0	0.7	1.5	1.7	8.7	2017	
12	11	Chris Andersen	C	38	CLE	12	0	9.5	0.8	1.8	0.409	0.0	0.3	0.000	0.8	1.6	0.474	0.409	0.8	1.2	0.714	0.8	1.8	2.6	0.4	0.4	0.6	0.4	1.7	2.3	2017	
13	12	Alan Anderson	SF	34	LAC	30	0	10.3	1.0	2.7	0.375	0.5	1.5	0.318	0.5	1.2	0.444	0.463	0.4	0.5	0.750	0.1	0.7	0.8	0.4	0.1	0.0	0.2	1.2	2.9	2017	
14	13	Justin Anderson	SF	23	TOT	75	10	16.4	2.5	5.9	0.424	0.8	2.6	0.299	1.7	3.3	0.522	0.490	1.4	1.7	0.791	0.9	2.3	3.3	0.9	0.5	0.3	0.9	1.5	7.1	2017	
15	13	Justin Anderson	SF	23	DAL	51	2	13.9	2.2	5.4	0.401	0.7	2.4	0.303	1.5	3.0	0.477	0.468	1.4	1.7	0.795	0.8	2.2	2.9	0.6	0.5	0.3	0.8	1.3	6.5	2017	
16	13	Justin Anderson	SF	23	PHI	24	8	21.6	3.1	6.8	0.463	0.9	3.0	0.292	2.3	3.8	0.600	0.528	1.3	1.7	0.780	1.3	2.8	4.0	1.4	0.5	0.3	1.1	1.9	8.5	2017	
17	14	Kyle Anderson	SF	23	SAS	72	14	14.2	1.3	2.9	0.445	0.2	0.6	0.375	1.1	2.3	0.462	0.481	0.6	0.8	0.789	0.5	2.4	2.9	1.3	0.7	0.4	0.5	0.9	3.4	2017	
18	15	Ryan Anderson	PF	28	HOU	72	72	29.4	4.5	10.7	0.418	2.8	7.0	0.403	1.7	3.7	0.446	0.550	1.8	2.1	0.860	1.6	3.0	4.6	0.9	0.4	0.2	0.8	2.0	13.6	2017	
19	16	Giannis Antetokounmpo	SF	22	MIL	80	80	35.6	8.2	15.7	0.521	0.6	2.3	0.272	7.6	13.5	0.563	0.541	5.9	7.7	0.770	1.8	7.0	8.8	5.4	1.6	1.9	2.9	3.1	22.9	2017	
20	17	Carmelo Anthony	SF	32	NYK	74	74	34.3	8.1	18.8	0.433	2.0	5.7	0.359	6.1	13.1	0.466	0.488	4.1	4.9	0.833	0.8	5.1	5.9	2.9	0.8	0.5	2.1	2.7	22.4	2017	
21	18	Joel Anthony	C	34	SAS	19	0	6.4	0.5	0.8	0.625	0.0	0.0	0.5	0.8	0.625	0.625	0.3	0.4	0.625	0.4	1.2	1.6	0.2	0.1	0.3	0.2	0.6	1.3	1.3	2017	

Figure 1

player_stats.csv contains 32 columns including 31 features. Each feature describes the player's basic information and performance such as the player's name, position, age, team, points, etc.

2). player_salaries.csv (Figure 2)

player_salaries				
	0	1	2	3
1	1	LeBron James, SF	Cleveland Cavaliers	\$30,963,450.00
2	2	Mike Conley, PG	Memphis Grizzlies	\$26,540,100.00
3	3	James Harden, SG	Houston Rockets	\$26,540,100.00
4	4	Kevin Durant, PF	Golden State Warriors	\$26,540,100.00
5	5	Russell Westbrook, PG	Oklahoma City Thunder	\$26,540,100.00
6	6	DeMar DeRozan, SF	Toronto Raptors	\$26,540,100.00
7	7	Al Horford, C	Boston Celtics	\$26,540,100.00
8	8	Dirk Nowitzki, F	Dallas Mavericks	\$25,000,000.00
9	9	Carmelo Anthony, PF	New York Knicks	\$24,559,380.00
10	10	Damian Lillard, PG	Portland Trail Blazers	\$24,328,425.00

Figure 2

Player_salaries.csv contains 4 columns including 3 features - name, team, and salary.

3) final_merged.csv(Figure 3)

final_merged																																		
	new_name	salary	year	Unnamed: 0	Rk	Pos	Age	Tm	G	GS	MP	FG	FGA	FG%	3P	3PA	3P%	2P	2PA	2P%	eFG%	FT	FTA	FT%	ORB	DRB	TRB	AST	STL	BLK	TOV	PF	PTS	
0	LeBron James	\$30,963,450.00	2017		280	221	SF	32	CLE	74	74	37.8	9.9	18.2	0.548	1.7	4.6	0.363	8.3	13.5	0.611	0.594	4.8	7.2	0.674	1.3	7.3	8.6	8.7	1.2	0.6	4.1	1.8	26.4
1	Mike Conley	\$26,540,100.00	2017		108	87	PG	29	MEM	69	68	33.2	6.7	14.6	0.460	2.5	6.1	0.408	4.2	8.6	0.497	0.545	4.6	5.3	0.859	0.4	3.0	3.5	6.3	1.3	0.3	2.3	1.8	20.5
2	James Harden	\$26,540,100.00	2017		218	173	PG	27	HOU	81	81	36.4	8.3	18.9	0.440	3.2	9.3	0.347	5.1	9.6	0.530	0.525	9.2	10.9	0.847	1.2	7.0	8.1	11.2	1.5	0.5	5.7	2.7	29.1
3	Kevin Durant	\$26,540,100.00	2017		147	119	PF	28	GSW	62	62	33.4	8.9	16.5	0.537	1.9	5.0	0.375	7.0	11.5	0.608	0.594	5.4	6.2	0.875	0.6	7.6	8.3	4.8	1.1	1.6	2.2	1.9	25.1
4	Russell Westbrook	\$26,540,100.00	2017		579	458	PG	28	OKC	81	81	34.6	10.2	24.0	0.425	2.5	7.2	0.343	7.7	16.8	0.459	0.476	8.8	10.4	0.845	1.7	9.0	10.7	10.4	1.6	0.4	5.4	2.3	31.6
5	DeMar DeRozan	\$26,540,100.00	2017		134	108	SG	27	TOR	74	74	35.4	9.7	20.9	0.467	0.4	1.7	0.266	9.3	19.2	0.484	0.477	7.4	8.7	0.842	0.9	4.3	5.2	3.9	1.1	0.2	2.4	1.8	27.3
6	Al Horford	\$26,540,100.00	2017		257	204	C	30	BOS	68	68	32.3	5.6	11.8	0.473	1.3	3.6	0.355	4.3	8.2	0.524	0.527	1.6	2.0	0.800	1.4	5.4	6.8	5.0	0.8	1.3	1.7	2.0	14.0
7	Dirk Nowitzki	\$25,000,000.00	2017		418	332	PF	38	DAL	54	54	26.4	5.5	12.6	0.437	1.5	3.9	0.378	4.0	8.7	0.463	0.495	1.8	2.1	0.875	0.4	6.1	6.5	1.5	0.6	0.7	0.9	2.1	14.2
8	Carmelo Anthony	\$24,559,380.00	2017		20	17	SF	32	NYK	74	74	34.3	8.1	18.8	0.433	2.0	5.7	0.359	6.1	13.1	0.466	0.488	4.1	4.9	0.833	0.8	5.1	5.9	2.9	0.8	0.5	2.1	2.7	22.4
9	Damian Lillard	\$24,328,425.00	2017		337	266	PG	26	POR	75	75	35.9	8.8	19.8	0.444	2.9	7.7	0.370	6.0	12.1	0.492	0.516	6.5	7.3	0.895	0.6	4.3	4.9	5.9	0.9	0.3	2.6	2.0	27.0
10	Dwyane Wade	\$23,200,000.00	2017		569	450	SG	35	CHI	60	59	29.9	6.9	15.9	0.434	0.8	2.4	0.310	6.2	13.5	0.456	0.457	3.7	4.7	0.794	1.1	3.5	4.5	3.8	1.4	0.7	2.3	1.9	18.3

Figure 3

We merged player_stats.csv and player_salaries.csv as one data file, which represents the player's information, performance, and yearly salary.

3. Data Preparation

3.1 Data Cleaning

The first step we did was to remove unnecessary features. We removed features that we were unsure what their meaning was, like 'Unnamed: 0'. Then, we chose to remove features that are the ratio, the sum, the difference, or any other form of a combination of two other features in the dataset. Doing this made us remove features like '2P%', which is the ratio of '2P' and '2PA', and 'FT%', which is the ratio of 'FT' and 'FTA'. We also opted to remove ORB and DRB, choosing to keep TRB, the combination of both features. The last feature that we removed was the players' names since we expected there would be no contribution to the model prediction.

After the initial feature removal, we started cleaning the data. Initially, the salary column was in string format, so we had to change it to integer after removing the \$ symbol. Then we used linear interpolation to fill the missing data. Luckily, there were only 7 missing values in the whole dataset.

3.2 Label Encoding on Categorical Data

Since we were aiming to do a regression model, we had to make sure that all of our data are in numeric format and no categorical data. In our dataset, we had 2 categorical features: Team(Tm) and Position(Pos). The Tm feature has 30 unique values, while the position has 14. Due to the number of unique values, we believe that doing label encoding for these two features would be the more realistic approach compared to one-hot encoding, as doing the latter would create too many features.

4. Feature selection

After exploring the data, it was time for us to do feature selection to find out which features to include in our final model. The goal of feature selection is to reduce the number of input variables to only the features believed to be most useful in predicting the target variable, which is the salary in our project. Including too many features, especially ones that are not very relevant to the target variable, can degrade the performance of models and can slow the model's development/training as it increases computational cost (requires a large amount of system memory). There are a lot of different feature selection methods and for our project, we chose to try out three different methods: Pearson correlation, variance inflation factor (VIF), and backward elimination.

4.1 Pearson Correlation

Our initial assumption about the dataset was that the features would have a positive correlation with the target, which in this case the salary. We believed that having better stats would lead to higher pay and vice versa. For that reason, the most obvious solution for us was to do a Pearson Correlation analysis between the features and the target. We wanted to analyze which of the features has a high positive correlation with salary

The Pearson correlation is a number between -1 and 1 that showcases the extent to which two variables are linearly related. A Pearson correlation close to 0 implies there is a weak correlation while closer to 1 implies there is a strong positive correlation (as x goes up, y goes up as well) and closer to -1 implies there is a strong negative correlation (as x goes up, y goes down). For our dataset, to visualize the Pearson correlation, we created a heatmap. The heatmap is a correlation matrix that shows the Pearson correlation between every feature in the dataset and the strength of those relationships. Looking at the heatmap, we chose the top 12 features with the highest correlation, which we found to be between 0.53 and 0.63. Those final 12 features we chose were PTS, FG, FGA, FTA, 2PA, MP, 2P, AST, STL, BLK, TOV, and PF.

The result of the Pearson correlation was not as good as we had hoped, resulting in only 0.63 in R2 score with the 12 features on Random Forest Regression with an ever-decreasing R2 score as we decrease the number of features.

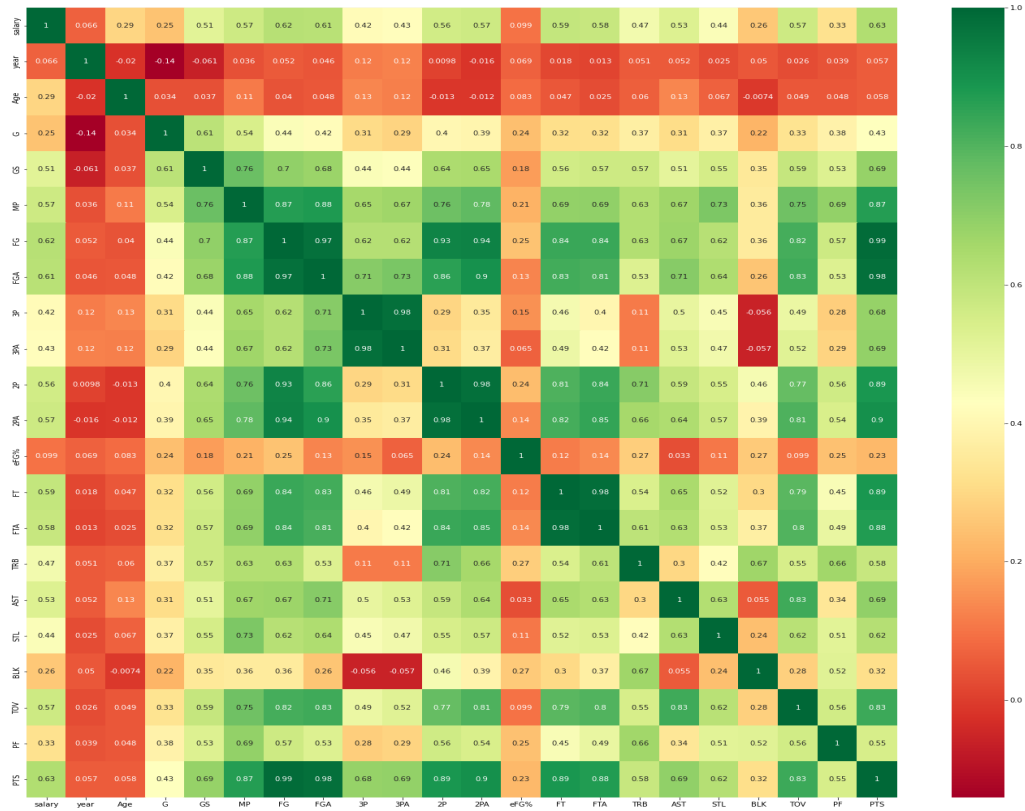


Figure 4(correlation between features)

4.2 Variance Inflation Factor(VIF)

The variance Inflation Factor is a statistical method to detect multicollinearity in regression analysis. In our model, we made some initial assumptions that the numerical features of the dataset (independent variables) would have a positive correlation with the salary (dependent variable). In our case, multicollinearity exists when there exists a linear relationship, or correlation, between the independent variables. Multicollinearity creates a problem in regression analysis because it indicates a high intercorrelation between the independent variables, and since they are supposed to be independent of each other, having high multicollinearity can lead to skewed or misleading results in our analysis, and obviously, we want to avoid that.

Based on the VIF analysis that we did, we concluded that we were going to eliminate any 3-digit number. While some argue that is too high, we feel that due to the nature of the result, that threshold makes sense, giving us 14 features as a result of this analysis: year, Pos, Age, Tm, G, GS, MP, eFG%, TRB, AST, STL, BLK, TOV, and PF.

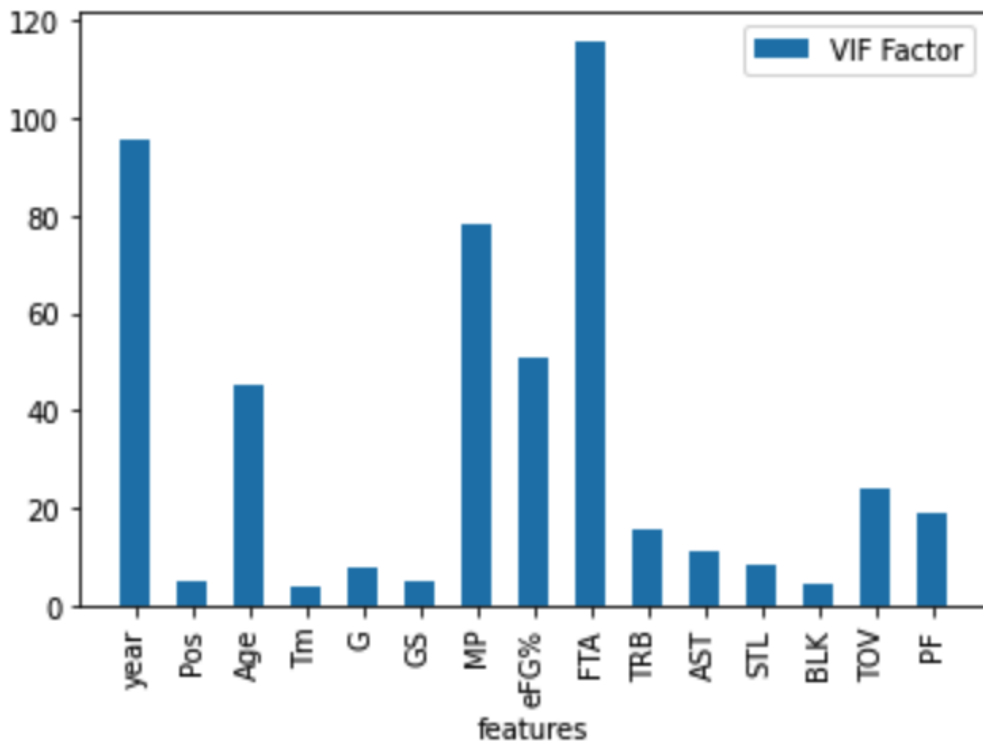


Figure 5 (features with VIF value and the VIF value of features less than 500)

	VIF Factor	features
0	95.5	year
1	4.9	Pos
2	45.3	Age
3	4.0	Tm
4	7.8	G
5	5.1	GS
6	78.2	MP
7	15697.6	FG
8	34024.9	FGA
9	1246.0	3P
10	5321.3	3PA
11	4420.5	2P
12	15054.3	2PA
13	50.8	eFG%
14	781.5	FT
15	115.8	FTA
16	15.8	TRB
17	11.0	AST
18	8.5	STL
19	4.3	BLK
20	23.7	TOV
21	18.9	PF
22	21427.9	PTS

Looking at the result between VIF and Pearson Correlation, we can see that 7 features appear in both analyses. The VIF however, did not show a significant improvement in Random Forest Regression, resulting in 0.64 in R2 score.

4.3 Backward Elimination

In the Backward Elimination method, the first step is to select a significance level that determines whether or not a feature can stay in the model. For our project, the significance level we selected was 0.05 because that is simply the standard level used in machine learning tasks. The next step is then to fit the complete model with all possible predictors and independent variables. Once the fitting is done, you choose the feature which has the highest p-value. If the p-value is greater than 0.05, then we remove that feature from the model and rebuild and refit the model with the remaining variables and repeat the previous steps until you reach the point where the p-value is less than 0.05, which means that you are finished (can't remove any more features) and the model is ready. So, in short, in backward elimination, you start with all the features and remove the least significant feature at each iteration that improves the performance of the model until there is no improvement due to feature removal. After applying this method to our project, the features we were left with were Age, Games played, Games started, Minutes Played, Field Goal Attempts, 3 point makes, 2 point attempts, effective field goal percentage, total rebounds, assists, steals, blocks, personal fouls, and points.

Out of the three different feature selection methods we tried, Backward elimination shows the highest R² score using Random Forest Regression at 0.66. We ultimately opted to use backward elimination as our feature selection method.

5. Method and Training

In this section, we start to try the different models first and then select the best models based on their R² score and Mean squared error (MSE). We chose to use Regression analysis because we believe that the independent and the dependent variable have a linear relationship. We hypothesized that having higher and better stats would lead to a higher salary. To prove our hypothesis, we used 5 different regression models with various degrees of success. We split the whole data into 70% and 30% for the training and testing datasets, respectively.

1) Multiple Linear Regression

We first started with the LinearRegression model to predict the value of a variable based on the value of other variables. We started with this because Multiple Linear Regression seems to be the most straightforward regression there is.

2) PolynomialFeatures

We also used linearRegression with PolynomialFeatures in order to generate polynomial features. After not having the highest result with Linear regression, we decided to try using polynomial regression, thinking that the outliers in the data could mean the dataset is polynomial and not linear.

With the PolynomialFeatures, we set the degree of the PolynomialFeature as 2 because a large value of a degree can cause the polynomial curve to become overly flexible and the shapes can be very strange. For the parameter of include_bias, we set it as False to avoid biased data.

3) RidgeCV

We tried ridge regression because we were worried that there would be overfitting in the model of linear regression. When we're working on the RidgeCV, we also choose to apply the cross validator - Repeated K-Fold to reduce bias. We also hoped that the bias-variance tradeoff of the regression model could help with the R2 number as well as MSE.

4) RandomForestRegressor

We chose random forest regressor because of the quality of the model. While random forest is computationally intensive compared to other regression models, since it is robust to outliers and is known to be more effective than a single decision tree, we chose to use the random forest.

5) Lasso

Based on the VIF values above, we feel that it made sense to try using Lasso regression. While we know our ridge regression didn't bear the best result, we wanted to try Lasso regression because of its well-suitedness to models with high multicollinearity.

Name of Regression	R2 Score	MSE
LinearRegression	0.57	5,535,863.40
Polynomial Regression	0.63	5,138,159.85
Ridge	0.58	5,446,570.84
RandomForest	0.66	4,863,334.22
Lasso	0.46	5,723,154.83

Figure 6 (comparison with different regression models)

The table above (figure 6) is a summary of the different models that we used in this project. Comparing these five different models, RandomForestRegression is the one with the highest R2 Score and the lowest MSE so we chose that as our final model.

6. Stats Improvement

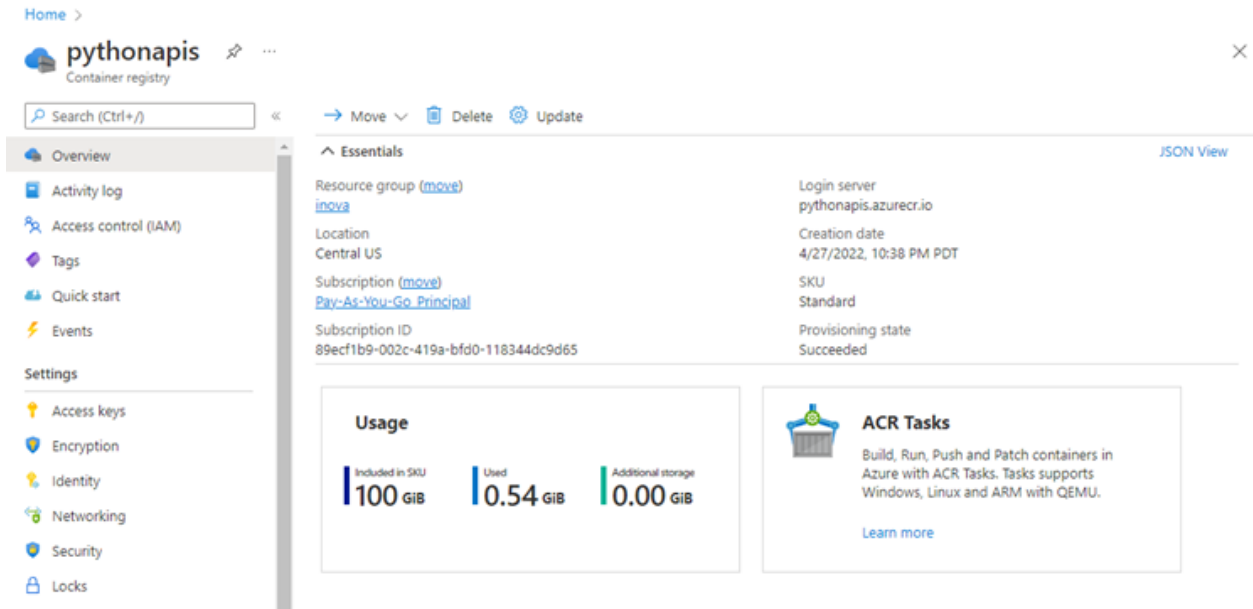
Another feature that we have on our website is the stats improvement feature. The purpose of this feature is to provide each player with realistic, improvable, and personalized suggestions for the stats they are lacking so they can increase their salary. At first glance, it is easy to just tell the players that improve all their stats, they will make more money. However, such a suggestion is not only unproductive but also unrealistic.

To provide a realistic and personalized suggestion, we researched how many percent can a basketball player realistically improve in a season. Our research showed that 20% seems to be the highest number. This leads to us creating a benchmark by dividing all the features used in the model into 3 different quartiles at 25%, 50%, and 75%. If a player's stat in any feature is at a max 20% lower than any of the quartile, then we inform the player that if he can increase his stat to said quartile, then his overall salary can increase by X dollar.

For example, if Johnny's PTS stat is 4.8 and the 25% quartile value of PTS feature is 5.6, we tell Johnny to increase his PTS to 5.6 and then we recalculate his new stats in the model. Another example is if Rob's BLK is 5 and the 25% quartile of BLK is 6.5, we do not advise Rob with any increase because that is below the 20% threshold.

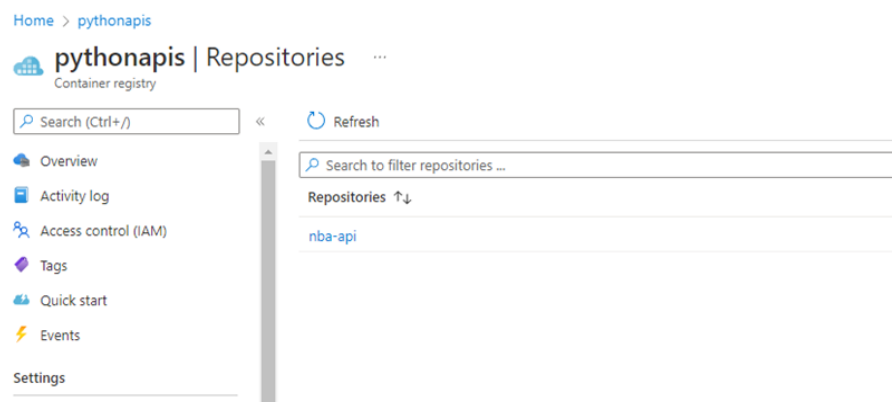
7. Flask API

Azure has a service Container Registry for this project. We chose the standard version one which provides 100GB of space and can be used to host as many Docker images as possible within this space, our image consumed 540 MB of this space.



After creating the Azure Container Register a couple of steps were necessary to deploy our Docker image to this server:

- o Add Azure extension to the VS code
- o Create locally the image “docker build -t pythonapis.azurecr.io/nba-api:latest .”
- o Login to azure from Vs Code “docker login pythonapis.azurecr.io”
- o And upload our image “docker push pythonapis.azurecr.io/nba-api:latest”



As you can see in the image below the action was logged on the Container Register side.

The screenshot displays the Container Registry interface for the 'nba-api:latest' image. The breadcrumb navigation shows 'Home > pythonapis > nba-api >'. The image name 'nba-api:latest' is followed by its SHA256 digest: 'sha256:7094854a6320d5938d7dabf7fce66cd718463b6445617b88434e38f57ee869'. Below this, the 'Essentials' section provides metadata: Repository (nba-api), Tag (latest), Tag creation date (4/27/2022, 10:52 PM PDT), Tag last updated date (4/27/2022, 10:52 PM PDT), Digest (sha256:7094854a6320d5938d7dabf7fce66cd718463b644...), Manifest creation date (4/27/2022, 10:52 PM PDT), and Platform (linux / amd64). A 'JSON View' link is available. The 'Docker pull command' is shown as 'docker pull pythonapis.azurecr.io/nba-api:latest'. The 'Manifest' section displays the JSON structure of the image manifest, including schema version, media types, config, and layers.

```
{
  "schemaVersion": 2,
  "mediaType": "application/vnd.docker.distribution.manifest.v2+json",
  "config": {
    "mediaType": "application/vnd.docker.container.image.v1+json",
    "size": 8446,
    "digest": "sha256:537dccee47d9b079d895dc8cbb6a96247f3f0be6634cfeff2f72fe99d27853f8"
  },
  "layers": [
    {
      "mediaType": "application/vnd.docker.image.rootfs.diff.tar.gzip",
      "size": 54917063,
      "digest": "sha256:e4d61adff20770048c6372d73c41b0b06f525a41f5530ef0509a876683055"
    },
    {
      "mediaType": "application/vnd.docker.image.rootfs.diff.tar.gzip",
      "size": 5153034,
      "digest": "sha256:4ff1945c672b08a1791df62afaf8aff14d3047155365f9c3646902937f7ffe6"
    },
    {
      "mediaType": "application/vnd.docker.image.rootfs.diff.tar.gzip",
      "size": 10071885,
      "digest": "sha256:ff5010aec998344606441aec43a335ab6326f32aee331a0270a16a0b04ec2be"
    },
    {
      "mediaType": "application/vnd.docker.image.rootfs.diff.tar.gzip",
      "size": 54075903,
      "digest": "sha256:12debc754e45686ace9e25d11bee372b070ee0505a020aa3b4f0b0c936496002"
    }
  ]
}
```

We created an APP Services for our Python Flask API. This service was chosen because it offers as an option a docker image from Container Register as a base.

After selecting the right Docker Image set “Yes” for continuous deployment which means every time we push our Image to the Container Register it will deploy again to our APP services, as we can see in the image below.

Home > App Services > inovapythonapis

App Services
Default Directory

+ Create Manage view ...

Filter for any field...

Name ↑

- inovafuntions
- inovapythonapis

Deployment Center

Settings

- Configuration
- Authentication
- Application Insights
- Identity
- Backups
- Custom domains
- TLS/SSL settings
- TLS/SSL settings (preview)
- Networking
- Scale up (App Service plan)
- Scale out (App Service plan)
- Webjobs
- Push
- MySQL in App
- Service Connector (Preview)
- Properties

Search (Ctrl+F)

Save Discard Browse Manage publish profile Leave Feedback

Settings Logs FTPS credentials

These are the logs from Docker Engine emitted during the provisioning phase of your container image.

Refresh

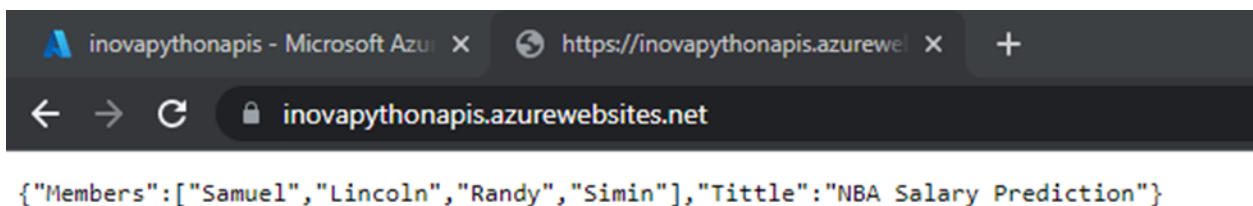
```

2022-05-02T01:56:06.118Z INFO - 33e859012cc2 Extracting 20MB / 20MB
2022-05-02T01:56:07.237Z INFO - 33e859012cc2 Extracting 20MB / 20MB
2022-05-02T01:56:07.436Z INFO - 33e859012cc2 Extracting 20MB / 20MB
2022-05-02T01:56:08.263Z INFO - 33e859012cc2 Extracting 33MB / 20MB
2022-05-02T01:56:08.433Z INFO - 33e859012cc2 Extracting 34MB / 20MB
2022-05-02T01:56:09.033Z INFO - 33e859012cc2 Extracting 37MB / 20MB
2022-05-02T01:56:09.157Z INFO - 33e859012cc2 Extracting 38MB / 20MB
2022-05-02T01:56:09.699Z INFO - 33e859012cc2 Extracting 40MB / 20MB
2022-05-02T01:56:09.930Z INFO - 33e859012cc2 Extracting 41MB / 20MB
2022-05-02T01:56:10.940Z INFO - 33e859012cc2 Extracting 45MB / 20MB
2022-05-02T01:56:11.110Z INFO - 33e859012cc2 Extracting 45MB / 20MB
2022-05-02T01:56:11.220Z INFO - 33e859012cc2 Extracting 46MB / 20MB
2022-05-02T01:56:11.658Z INFO - 33e859012cc2 Extracting 49MB / 20MB
2022-05-02T01:56:12.000Z INFO - 33e859012cc2 Extracting 57MB / 20MB
2022-05-02T01:56:12.279Z INFO - 33e859012cc2 Extracting 62MB / 20MB
2022-05-02T01:56:12.620Z INFO - 33e859012cc2 Extracting 70MB / 20MB
2022-05-02T01:56:12.768Z INFO - 33e859012cc2 Extracting 73MB / 20MB
2022-05-02T01:56:13.112Z INFO - 33e859012cc2 Extracting 78MB / 20MB
2022-05-02T01:56:13.523Z INFO - 33e859012cc2 Extracting 82MB / 20MB
2022-05-02T01:56:13.734Z INFO - 33e859012cc2 Extracting 87MB / 20MB
2022-05-02T01:56:13.843Z INFO - 33e859012cc2 Extracting 90MB / 20MB
2022-05-02T01:56:14.262Z INFO - 33e859012cc2 Extracting 94MB / 20MB
2022-05-02T01:56:14.402Z INFO - 33e859012cc2 Extracting 95MB / 20MB
2022-05-02T01:56:14.503Z INFO - 33e859012cc2 Extracting 95MB / 20MB
2022-05-02T01:56:15.262Z INFO - 33e859012cc2 Extracting 98MB / 20MB
2022-05-02T01:56:15.723Z INFO - 33e859012cc2 Extracting 99MB / 20MB
2022-05-02T01:56:15.946Z INFO - 33e859012cc2 Extracting 99MB / 20MB
2022-05-02T01:56:16.737Z INFO - 33e859012cc2 Extracting 102MB / 20MB
2022-05-02T01:56:16.868Z INFO - 33e859012cc2 Extracting 103MB / 20MB
2022-05-02T01:56:17.038Z INFO - 33e859012cc2 Extracting 104MB / 20MB
2022-05-02T01:56:18.860Z INFO - 33e859012cc2 Extracting 106MB / 20MB
2022-05-02T01:56:19.138Z INFO - 33e859012cc2 Extracting 107MB / 20MB
2022-05-02T01:56:20.106Z INFO - 33e859012cc2 Extracting 110MB / 20MB
2022-05-02T01:56:22.138Z INFO - 33e859012cc2 Extracting 113MB / 20MB
2022-05-02T01:56:22.126Z INFO - 33e859012cc2 Extracting 114MB / 20MB
2022-05-02T01:56:22.127Z INFO - 33e859012cc2 Extracting 115MB / 20MB
2022-05-02T01:56:22.128Z INFO - 33e859012cc2 Extracting 119MB / 20MB
2022-05-02T01:56:22.158Z INFO - 33e859012cc2 Extracting 120MB / 20MB
2022-05-02T01:56:22.166Z INFO - 33e859012cc2 Extracting 120MB / 20MB
2022-05-02T01:56:22.167Z INFO - 33e859012cc2 Extracting 122MB / 20MB
2022-05-02T01:56:22.168Z INFO - 33e859012cc2 Extracting 123MB / 20MB

```

By now we can access the root URL of our API and to a get using this Address:

<https://inovapythonapis.azurewebsites.net/>



Right now, our API is online and waiting for requests.

For our UI we choose to use the .net Core Asp Net MVC framework because of previous experience with the technology.

The UI can be found at this URL : <https://nba.inovagenetica.com/>


NBA

Player's age on February 1 of the season	Games
<input type="text" value="39"/>	<input type="text" value="59"/>
Games Started	Minutes Played Per Game
<input type="text" value="40"/>	<input type="text" value="16.2"/>
Field Goal Attempts Per Game	3-Point Field Goals Per Game
<input type="text" value="14.3"/>	<input type="text" value="7.1"/>
2-Point Field Goal Attempts Per Game	Effective Field Goal Percentage
<input type="text" value="2.5"/>	<input type="text" value="69%"/>
Total Rebounds Per Game	Assists Per Game
<input type="text" value="15.0"/>	<input type="text" value="9.6"/>
Steals Per Game	Blocks Per Game
<input type="text" value="1.9"/>	<input type="text" value="9.6"/>
Personal Fouls Per Game	Points Per Game
<input type="text" value="2.7"/>	<input type="text" value="32.6"/>

A button “Random” is available to easily generate all the parameters necessary for the prediction. The random numbers generated by the button are between the min-max numbers for each feature as it appears on the data source. After generating all the parameters the next step is “Submit” these parameters to run the prediction, what happens in the back and is, this parameter arrives at our “Controller” in the backend of our UI, it converts this feature to a JSON format and it creates a POST to our Flask API, after the prediction it returns a JSON response to our UI.

NBA

Player's age on February 1 of the season	Games
<input type="text" value="31"/>	<input type="text" value="77"/>
Games Started	Minutes Played Per Game
<input type="text" value="20"/>	<input type="text" value="16.2"/>
Field Goal Attempts Per Game	3-Point Field Goals Per Game
<input type="text" value="11.3"/>	<input type="text" value="7.1"/>
2-Point Field Goal Attempts Per Game	Effective Field Goal Percentage
<input type="text" value="18.1"/>	<input type="text" value="69%"/>
Total Rebounds Per Game	Assists Per Game
<input type="text" value="14.2"/>	<input type="text" value="9.6"/>
Steals Per Game	Blocks Per Game
<input type="text" value="3.3"/>	<input type="text" value="9.6"/>
Personal Fouls Per Game	Points Per Game
<input type="text" value="2.1"/>	<input type="text" value="32.6"/>



NBA Salary Recommendation

Based on the information you provide, the salary should be \$20,737,097.45

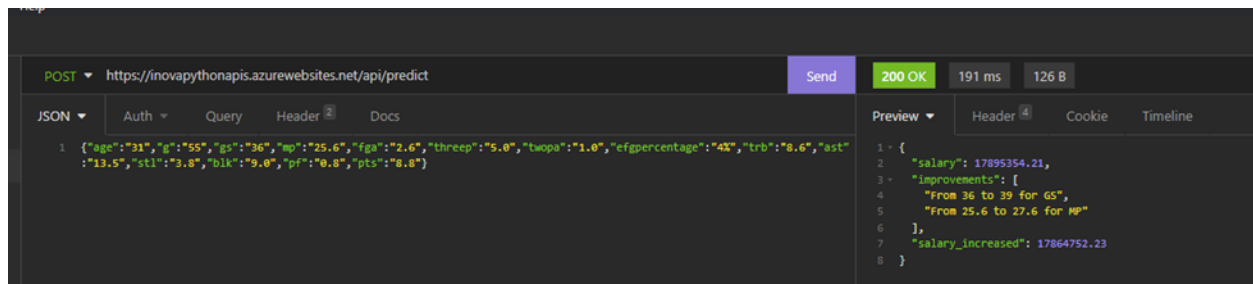
If you improve these stats:

From 2.1 to 2.3 for PF

Your salary can increase from \$20,737,097.45 to \$20,903,454.55

When the Flask API response hits the UI service it pops up the response in a window showing the salary found based on the parameters provided when it is possible it also recommends how to improve the salary with a small change in the initial parameters.

Post request using Insomnia directly to the Python API.



8. Conclusion

From this project, we learned that predicting salaries for NBA players is not as simple as it seems. After doing feature selection (using backward elimination), we found that there were 14 features (age, games played, among others) that were most important in determining salary. Improving upon stats such as points per game or effective field goal percentage among others was associated with significant increases in salary. However, the highest r-squared value we found out of all the models (random forest regressor) was only 0.66, which is a solid but not elite performance for the model. The mean squared error was also around 4.9 million dollars which indicates that there was quite a big difference between the predicted and actual values of salary.

Overall, the performance of our model taught us that there are probably a lot of other factors that influence salary that we didn't take into account. One possible factor could be the player's level of fame. More famous players, such as Stephen Curry and LeBron James, are likely to continue making very high salaries even if their stats were to decline simply due to the revenue that their name brand generates (increased ticket sales, jersey sales, advertisements, etc.). However, it is pretty difficult to perfectly quantify fame.

One way we could have done it was to include data about a combination of factors that reflect fame levels such as number of Twitter followers, number of Instagram followers, number of all star games made, number of all-NBA teams made, endorsement salary and draft pedigree (player drafted number one overall is more likely

to have high name recognition due to previous hype from high school/college). We couldn't find a dataset that included this information. To get this information, we would probably need to do some more web scraping or calls to the APIs of various websites which would have been rather difficult as we didn't have much experience with doing that and we were on a deadline with this project.

Another way we could have improved this project is by including more data of players from previous years. Our dataset only included players from the past five seasons because the salary cap (amount of money teams could spend on players) exploded in that time frame due to new television deals. Thus, it wouldn't make much sense to predict a new player's salary by comparing their stats and corresponding salary level to players who played in a previous era where the salary was naturally lower regardless of the stats. To get around this, we could have tried to find the corresponding salary cap information for each year and put each player's salary as a relative percentage of salary cap and tried to predict the percentage of salary cap that a player would receive and convert that percentage to a yearly salary. Incorporating this information would have been pretty time-consuming and may not have even led to more accurate results so we didn't do that. Furthermore, we could have included advanced stats in addition to the traditional counting stats that were in the dataset. Adding these features to the dataset may have improved our r-squared score but the performance of the model would still most likely be imperfect as there are intangible qualities that can't be measured that also affect salary such as a player's reputation as a teammate/leader, fit within team's system and willingness to take a pay cut (for example, may value championships more than money).

Lastly, to expand on this work in the future, instead of just predicting a yearly salary, I would also try to predict the length of the contract the free agent would sign, as NBA players tend to be on several multi-year contracts over the length of their career and don't just have one yearly salary that stays constant throughout their career. This addition would make our model even more realistic and thus, more helpful, as it would be more in tune with the players' needs.