# CS542 Final Project: Data Analysis of NBA Statistics

**Authors**
Yichen Mu ycmu@bu.edu
Fangxu Zhou fxzhou@bu.edu
Jingxuan Guo krysguo@bu.edu
Hao Fu haofu042@bu.edu

## Abstract

The goal of this paper was to analyze the NBA Statistics in some machine learning ways. By saying analyzing, we tried to predict NBA results of 2003 based on related statistics of players, coaches and results of season games of previous years. Also, we tried to figure out the outliers of players using the players' statistics. Besides, for specially exercising unsupervised learning technique, we clustered the players with statistics of their performance during season games. For the prediction part, we used techniques **KNN**, **SVM** and Decision Tree. We applied the **DBSCAN** clustering method to find the outliers. Then with the very common **Kmeans++** we cluster the players to explore this famous unsupervised learning technique.

## ACKNOWLEDGEMENTS

At the very beginning of this report, we would like to express our thanks to the professor **Peter Chin** and two teaching fellows **Peilun Dai** and **Andrew Wood** for giving us such an opportunity to try out the techniques we learned and improve ourselves.

## 1 Introduction

Basketball is all the time a fashionable sport all around the world. Machine learning is becoming more and more trendy nowadays. Therefore, that would be a nice try to combine these two elements to see if there are some findings fun and useful. As four novices to the domain of machine learning, we have to admit that we were somewhat overwhelmed by its complexity and interdisciplinarity. However, in spite of all the obstacles we've encountered, as well as the mistakes we may have made, we still found this science fascinating and were happy for having had chosen this **CS542**, which offered us this chance to have a brilliant embark on this venture of machine learning.

The goal of this paper was to exercise some ML techniques we learned from **CS542** as well as those we saw from the internet. We used techniques like **K-Nearest-Neighbors(KNN)**, **Support Vector Machine(SVM)** and Decision Tree for the prediction of outcomes of 2004 NBA games with data from previous years up to 1950s. For detecting the outliers of players and clustering the players, we tried **PCA** respectively with the unsupervised learning techniques **Kmeans++** and **DBSCAN**.

The data collected for this project was only use for the **CS542** project. All the methods defined below were used only for the **CS542** project. The code and the write-up for the **CS542** project are available at `https://github.com/FangxuZHOU/CS542_prj.git`

All the code for this paper is available at `https://github.com/FangxuZHOU/CS542_prj.git`

All the pictures or tables showed on this paper are available at `https://github.com/FangxuZHOU/CS542_prj.git`

## 2 Related Work

I guess that there must be loads of similar work that have been done before, but we managed to find only one, with surely totally different dataset. This one comes from Nadir N who wrote a paper entitled *Assessing NBA player similarity with Machine Learning (R)* published on *towards data science*. The goal of that paper was to use Unsupervised Machine Learning and **PCA** to determine which NBA superstars are the most alike. For the clustering technique, they used **Kmeans** together with **PCA**. Using data from the 2017–18 NBA Regular season combined three data-sets from the websites Basketball-reference and NBAmining.com as input, they try to cluster the players to see who are most alike. For the prediction part, pitifully, we haven't found any similar work done. Maybe it was because we were not able to do much research due to the limitation of time. However, I'm pretty sure the plenitude of such work similar to ours.

## 3 Dataset and Features

The data we used in this project is of *Project F: NBA statistics data* from the `http://www.cs.cmu.edu/~ggordon/10601/projects.html`, which is a publicly available website specially for purposes of exercising techniques learnt from machine learning courses. Basically, the data provided from the website contains 2004-2005 NBA and ABA stats for:

- Player regular season stats
- Player regular season career totals
- Player playoff stats
- Player playoff career totals
- Player all-star game stats
- Team regular season stats
- coaches_season.txt - nba coaching records by season
- coaches_career.txt - nba career coaching records

What we've used are the $1^{st}, 2^{nd}, 4^{th}, 6^{th}$ and the last two items

### 3.1 Prediction of the outcomes of season games

For this part, we used datasets *Player regular season stats*, *coaches_season.txt* and *Team regular season stats* which respectively contain the performance stats of every player, coach, team in their regular season matches starts from 1946. Since what we need to do is to use the data to fit learning models to predict the outcome, we have to combine and filter the above datasets to construct a comprehensive dataset. So after merging them, we observed the data and created some more representative feature columns. After that, this main dataset we use for this part has 72 feature columns, which mainly are some original features like WON, LOST, GP, MINUTES and some new features like T_o_Effic, T_d_3P_rate, Coach_Effic, Player_Effic and others we created based on the original features.

### 3.2 Outliers detection for NBA players

In this part, we used the datasets *Player regular season career totals* and *Player playoff career totals* which contain respectively the performance stats of each players' whole career of regular season games and of playoff games start from 1946. There were 21 input features: 1. The first four show the basic information of a player, like his id, his league, his first name and last name. 2. Other features can be used to evaluate performance of a player, like OREB (stands for offensive rebounds), DREB (stands for defensive rebounds), etc. (Those are the features we used for analyzing the outiers)

### 3.3 Finding outstanding Players

This part is only for trying out the **Kmeans++** technique, also, because of the lack of time, we decided to only use the dataset of *Player regular season career totals*. Features are like what are listed at the previous section.

# 4 Methods

## 4.1 Prediction of the outcomes of season games

### 4.1.1 Cluster the data to add labels

Since there are some congenital defects in our original datasets, including player, team, and coaches datasets. They all lack the label data. With that problem, after merging, filtering, calculating and get a comprehensive dataset, we still couldn't implement Supervised Learning algorithm on our dataset. So after preprocessing the data, in order to gain labels, we decide to use **Kmeans++** clustering algorithm(Unsupervised Learning) to cluster the data and get centroids, then we can set labels according to the sum of feature values of centroids. In this way, there will be label data in our training and testing dataset. After finishing these works, we can then use the supervised learning algorithms like **KNN** on the training data later.

**4.1.1.1 Data normalization**

After having tried other Normalization method like Standardscaler(), we found Minmaxscaler is optimal. Since there are more than 60 features and the scales of data are pretty different, and what we want to do is to scale the features between a specified maximum and minimum value, which in this case is between 0 and 1, that's why we use this Minmaxscaler method.

**4.1.1.2 Implementation procedure**

We defined two functions which are elbow function and create_label function. In elbow function, we used **Kmeans++** algorithm and matplotlib package to plot an image which is related to the optimal number for the clusters. As for the aim of create_label function, we use it to calculate the label for every row of data in both training and testing dataset. After implementing the elbow function, we can get images as following.

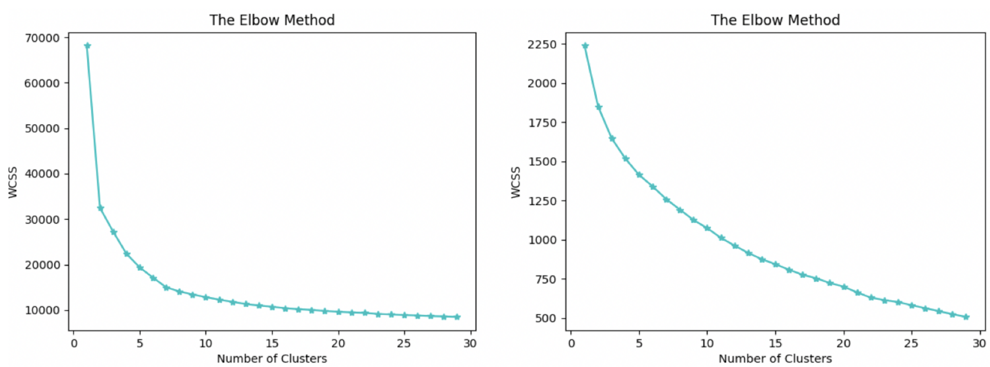



Figure 1: elbow

The left image is from the training data, and the right image is from the testing data. We can see that both two curves show an obvious inflection point when the Number of Clusters is equal to 5. So we set the 'n_cluster' parameter in **Kmeans++** as 5 and implemented the **Kmeans++** algorithm on both two datasets. Then we can get the 5 centroids and then inverse and transform them into their original data. Then we can sum every centroid data and sort them in descending order to get 5 benchmarks(0 4). After that, we can set the label to every row of data based on the distance between the sum of every row of data and these 5 benchmarks. Then, we got the training dataset with labels and testing dataset with labels. As for the reason why we need to set labels to the testing dataset as well, since later we need to extract the labels in testing dataset as the control data to compare with the outcomes we will predict using the trained learning model to get the accuracy of the algorithm. Results which are training and testing dataset with labels will be shown in result section.

### 4.1.2 Prediction of the outcomes of season games of 2003

Now we have preprocessed all the data and added labels to them, and we finally can implement the supervised learning algorithms on the data and see if we can predict the outcomes in 2003 correctly. As for this part, we used the make_pipeline method to construct a pipeline to deal with the data, the make_pipeline we used contains minmaxscaler, Principal Component Analysis and different learning algorithms. And as for the learning algorithms, we tried K-NearestNeighbor, Support vector machine, decision tree and neural network.

#### 4.1.2.1 Minmaxscaler

This has been mentioned in Data normalization in section **4.1.1.1**

#### 4.1.2.2 PCA

This has been mentioned in section 4.2.2

#### 4.1.2.3 KNN

K-Nearest Neighbors is a method that classifies every record in a data set. The main idea of it is in the feature space, if most of the K most adjacent samples of a sample belong to a certain category, then the sample also belongs to that certain category and has the characteristics of sample in that category. And in determining classification decision, the method determines the category of sample according to the category of the closest one or several samples. The distance formula we used here is **minkowski**, which is

$$dist(X,Y) = (\sum_{i=1}^{N} |x_i - y_i|^p)^(1/p)$$

in which p is a variable. And we use sklearn package to implement this algorithm. The parameter is

#### 4.1.2.4 SVM

Support Vector Machine is a kind of generalized linear classifier that classifies data in a supervised learning way. Its decision boundary is the maximum-margin hyperplane solved for the learning sample. And **SVM** can implement the kernel function to do the non-linear classification. And the kernel function we used here is **RBF kernel**, which is And we use sklearn package to implement this

$$\kappa\left(\boldsymbol{X}_1, \boldsymbol{X}_2\right) = \exp\left(-\frac{\left\|\boldsymbol{X}_1 - \boldsymbol{X}_2\right\|^2}{2\sigma^2}\right)$$

algorithm. The parameters are kernel='rbf',class_weight='balanced',C=3.0.

#### 4.1.2.5 decision tree

Decision tree algorithm is a method to approximate discrete function values. It uses the inductive algorithm to generate readable rules and decision trees, then these decisions are used to analyze the new data and classify the data through a series of rules. And the related parameters we used are $criterion =' entropy'$,$max\_depth = None$,$min\_samples\_split = 2$,$min\_samples\_leaf = 1$,$max\_features = None$,$max\_leaf\_nodes = None$,$min\_impurity\_decrease = 0$

#### 4.1.2.6 Neural Networks

Neural Networks is an algorithmic mathematical model that imitates the behavior characteristics of animal neural network and carries on distributed parallel information processing. It relies on the number of hidden layers in the network and the number of nodes on the hidden layers to adjust the complexity, and constructs an effective model for processing information by adjusting the interrelated relationship between a large number of nodes in the network. And in our project we implement the MLPClassifier in sklearn package. The main related parameters are solver='sgd',activation='relu',alpha=1e-4,hidden_layer_sizes=(300,300,300),random_state=1,max_iter=200,learning_rate_init=0.001, in which the 'sgd' means the sigmoid function which is

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

### 4.2 Outliers detection for players

This part, detecting the outliers of players, we removed first the players whose "minutes" column has value as 0, and take those players as outliers (We did so, because we thought it was quite reasonable to do so, a player cannot play 0 minute during his entire career and not have 0 as scores). We then divide each value of the features of the player by his total minutes. Normalizing by minutes gives us a fairer representation of each player's contributions. If we looked at game statistics instead, players who play more minutes would look better than players who play on elite teams and often don't play at the end of games that are blow-out wins. Then we normalized the features using **Z-Score**. We continued by utilizing the **PCA** technique to reduce the dimensions. Subsequently, with **Silhouette Score**, we tried different combinations of minimum number of clusters and the value of epsilon to find the most appropriate one for the **DBSCAN** clustering method. For the next step, we clustered the players with **DBSCAN** and finally generated a list of outliers.

#### 4.2.1 Z-Score (standard score)

With Z-Score, we normalized our dataset, which means that for each value, we are going to subtract the mean of the feature-range from that value and divide it by the standard deviation of the feature. This ensures that features with high-value ranges do not have a greater impact on our overall similarity comparison than features with low-value ranges.

$$z = \frac{x-\mu}{\sigma}$$

where: $\mu$ is the mean of the population. $\epsilon$ is the standard deviation of the population.

#### 4.2.2 Principal component analysis (PCA)

We decided to apply the **PCA** technique to do the dimension reduction. A sklearn library was used (Scikit-learn: Machine Learning in Python) to help us to deal with **PCA**. **PCA** reduce dimensions by Projecting each data point onto only the first few principal components to obtain lower-dimensional data while preserving as much of the data's variation as possible

#### 4.2.3 Silhouette Score

Silhouette Score is a metric used to calculate the goodness of a clustering technique. Its value ranges from -1 to 1. Here, we used it to find the most appropriate combination of minimum samples and epsilon value for **DBSCAN**. We used a method of the sklearn.metrics library to carry it out.

$$s(i) = \frac{b(i)-a(i)}{max\{a(i)-b(i)\}}$$

where: $a(i)$ can be interpreted as a measure of how well the point $i$ is assigned to its cluster (the smaller the value, the better the assignment). $b(i)$ is the smallest mean distance of $i$ to all points in any other cluster.

Its value ranges from -1 to 1:

- 1: Means clusters are well apart from each other and clearly distinguished.
- 0: Means clusters are indifferent, or we can say that the distance between clusters is not significant.
- -1: Means clusters are assigned in the wrong way.

#### 4.2.4 DBSCAN clustering

With a library of sklearn.cluster, we carried out this clustering technique. **DBSCAN**, whose full name is Density-based spatial clustering of applications with noise, given a set of points in some space, it groups together points that are closely packed together (points with many nearby neighbors), marking as outliers points that lie alone in low-density regions (whose nearest neighbors are too far away).

Unlike **Kmeans++** clustering, we don't have to pre-decide the number of clusters while using **DBSCAN**, the algorithm itself would take care of it.

But we do have to choose two parameters appropriately first, they are: $minPts$, the minimum number of points required to form a dense region, and $\epsilon$ who specifies how close points should be to each other to be considered a part of a cluster.

### 4.3 Finding outstanding players

For this part, we used **Kmeans++**, since we had already tried **DBSCAN** at the previous part, we wanted to try a new one. And we are told by the internet that **Kmeans++** clustering technique is a very commonly used one. For the data pre-processing phase, it's almost the same as what we did for the "Outliers detection" part. Except that we removed the outliers we found at the outlier detection part. Then with the elbow method we tried to find the most appropriate number of clusters, which will be used later to cluster the points with **Kmeans++** technique. Introduction of these two methods, elbow method and **Kmeans++**, are given at the section **4.1**

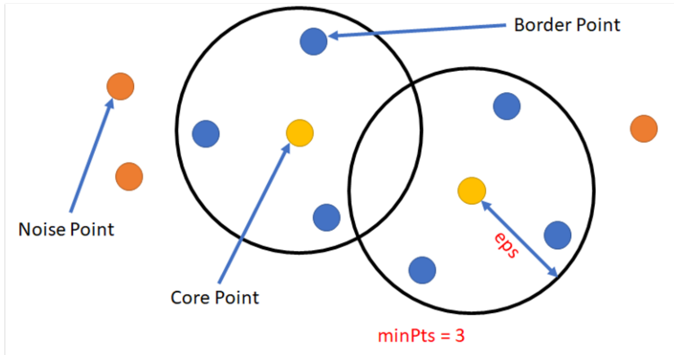



Figure 2: DBSCAN

# 5 Results

## 5.1 Prediction of the outcomes of season games

### 5.1.1 Results of clusters and labels

After we implemented the **Kmeans++** clustering algorithms in section **4.1.1**, we got five centroids and five benchmarks and we sorted them in descending orders. After that, we set label to every row of data based on the distance between the sum of every row of data and these 5 benchmarks. Then, we get the training dataset with labels and testing dataset with labels. Here due to the space limit, I just place the training data here as an example, as shown in *Figure 3*. And the testing data have the same structures. As for the meaning of labels, since there are 5 different labels which are

```
   year  o_fgm  o_fga  o_ftm  o_fta  o_oreb  o_dreb  o_reb  o_asts  o_pf  o_stl  o_to  o_blk  o_3pm  o_3pa  o_pts  d_fgm  d_fga
  d_ftm  d_fta  d_oreb  d_dreb  d_reb  d_asts  d_pf  d_stl  d_to  d_blk  d_3pm  d_3pa  d_pts      pace  won  lost  yr_order
 season_win  season_loss  gp  minutes   pts  oreb  dreb  reb  asts  stl  blk  turnover   pf   fga  fgm  fta  ftm  tpa  tpm
 T_o_Effic  T_o_FG_rate  T_o_FT_rate  T_o_3P_rate  T_d_Effic  T_d_FG_rate  T_d_FT_rate  T_d_3P_rate  T_WIN_LOSE_rate  Coach_Effic
  P_Season_Effic  P_Season_FG_rate  P_Season_FT_rate  P_Season_PTS_rate
0    1946  1397  5133  811  1375    0    0    0   470  1202    0    0    0    0    0  3605    0    0
     0    0    0    0    0    0    0    0    0    0    0    0  3900  0.000000  22  38    1
    22         38   6     0   13    0    0    0    1    0    0     0.0   15   22    5    4    3    0    0   -225
    0.272161    0.589818    0.000000     3900    0.000000    0.000000    0.000000       0.366667    0.733333
 -0.666667        0.227273        0.750000        0.000000
1    1946  1397  5133  811  1375    0    0    0   470  1202    0    0    0    0    0  3605    0    0
     0    0    0    0    0    0    0    0    0    0    0    0  3900  0.000000  22  38    1
    22         38  58     0  567    0    0    0   60    0    0     0.0  115  870  223  193  121    0    0   -225
    0.272161    0.589818    0.000000     3900    0.000000    0.000000    0.000000       0.366667    0.733333
 -1.586207        0.256322        0.626943        0.000000
```

```
0        4
1        4
2        4
3        4
4        4
        ..
18105    3
18106    4
18107    1
18108    4
18109    4
Name: label
```

Figure 3: training dataset with labels

from 0 to 4, and we sorted them in descending order, we can define the label 0 as having a very high probability of winning and we define the remaining labels as the winning rates which are in descending order.

### 5.1.2 Results of predictions of the outcomes of 2003

After trying different machine learning models to fit the data and then use them to predict the outcomes of 2003, finally we compare the result we predicted and the true results in the orginal testing dataset, we got the different accuracies as shown in *Figure 4*. So eventually we can tell that the Neural

When PCA (n_components=0.98, svd_solver='auto'),
KNN: RMSE on testing set = 1.3280510201777347
Decision Tree: RMSE on testing set = 1.3043080587787235
SVM: RMSE on testing set = 1.86429873038029

Figure 4: performances of different learning models

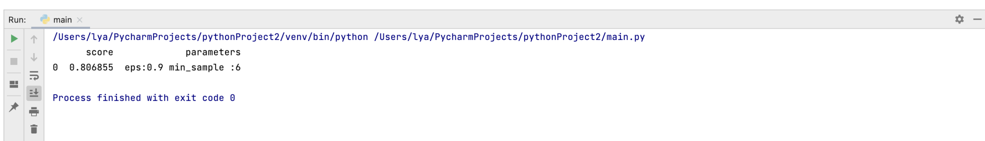
```
[0 3 3 4 3 4 3 3 3 3 4 4 3 3 3 3 4 3 4 4 4 3 4 4 1 3 4 4 3 4 3 4 3 3 4 4 3
 4 3 0 4 4 4 1 3 4 4 3 4 3 4 3 3 4 4 3 4 3 0 4 3 3 3 4 4 0 1 4 4 4 4 3 3 3
 3 3 3 4 3 3 4 3 3 3 4 4 0 4 4 4 4 4 3 3 3 3 3 3 4 3 3 4 3 4 3 4 4 0 4 4 4
 4 4 3 4 3 3 3 3 4 3 3 4 3 4 3 3 3 3 3 4 4 3 3 3 3 3 3 3 4 3 3 3 4 4 0 4 4
 4 0 4 4 4 4 1 4 4 4 1 4 1 4 0 4 4 1 4 4 4 3 4 4 1 4 4 4 4 4 4 4 4 4 4 3 3
 4 4 4 4 3 4 4 4 4 4 1 3 3 4 4 4 4 3 4 3 4 4 3 4 3 4 1 4 4 4 4 4 4 4 4 4 4
 3 4 4 4 4 4 4 4 3 3 4 3 3 4 3 1 4 3 4 4 4 4 4 4 4 4 3 4 4 3 3 4 1 3 1 4 3
 3 1 4 4 4 1 4 4 3 4 4 4 3 3 4 0 3 4 3 4 0 4 4 4 4 0 4 4 4 4 0 4 3 4 3 4 4
 3 3 4 4 1 4 3 4 4 4 4 1 1 3 4 3 4 3 4 3 4 4 4 4 4 1 4 4 4 4 4 3 3 4 4 4 3
 4 4 4 4 1 4 3 4 4 3 4 4 3 1 4 4 4 4 4 4 4 4 3 4 4 3 4 4 3 4 4 3 1 4 4 4 4
 4 4 4 4 3 4 4 3 4 4 3 3 3 3 4 4 4 4 3 3 3 4 3 4 3 4 4 3 4 3 4 4 3 3 3 3 4
 4 4 4 3 3 3 4 3 4 3 4 4 3 4 3 3 1 4 3 3 3 4 0 3 4 4 3 4 4 3 4 3 3 4 4 3 3
 1 4 3 3 3 4 0 3 4 4 3 4 4 3 4 3 3 4 4 3 4 4 3 4 4 4 4 4 4 3 3 3 4 4 4 4 3
 4 4 4 4 4 4 3 3 3 4 4 3 4 3 3 3 3 3 4 4 3 3 4 4 3 0 3 4 4 3 3 4 3 3 3 3 3
 4 4 3 3 4 4 3 0 3 4 4 3 4 3 4 4 4 3 4 4 4 4 3 3 4 4 3 4 4 4 4 4 4 3 4 4 4
 4 4 4 0 1 4 0 4 4 4 4 4 3 4 4 4 1 4 3 4 4 4 4 3 3 3 3 4 4 4 4 4 1 4 3 4 4
 0 4 4 4 4 3 4 4 4 4 3 4 4 4 4 4 4 4 3 4 4 4 4 4 4 4 4 4 4 3 3 4 4 3 4 4 4
 4 3 1 3 4 3 3 4 3 4 4 3 4 3 0 3 1 4 1 4 4 3 4 4 4 3 3]
RMSE on testing set =  1.0960016022776253

Process finished with exit code 0
```

Figure 5: accuracy of NNs and outcome it predicts

Networks has the highest accuracy and it will be the optimal learning model to predict the outcome of 2003. The accuracy of Neural Networks and outcome it predicted are shown in *Figure 5*.

## 5.2 Outliers detection for players

We tried a long list of combinations of $minPts$ and $\epsilon$, we computed respectively the silhouette score of **DBSCAN** clustering using each of the combinations. Then we chose the one with the largest score, which is shown in the *Figure 6*.



```
Run:  main
/Users/lya/PycharmProjects/pythonProject2/venv/bin/python /Users/lya/PycharmProjects/pythonProject2/main.py
      score           parameters
0  0.806855  eps:0.9 min_sample :6

Process finished with exit code 0
```

Figure 6: Silhouette score of the most appropriate $minPts$ and $\epsilon$

With the combination of $minPts$ and $\epsilon$ gained, we clustered with **DBSCAN**, and got a plot as shown in *Figure 7*.

Then, as presented in *Figure 8*, we came up a list of players who correspond to the black circles in the plot, and who therefore are considered to be outliers. We haven't yet analyzed why these players are set to be outliers since we ran out of time. But at a very first glance of the list, we noticed that those players, compared to others, they have very small values as their "minutes" features.

So we made a bold guess, maybe it can be one of the reasons: A player is taken as an outlier maybe it's partly because he played too less minutes during his entire career. Surely, there would be other reasons, but for now, we've figured only this one out.

## 5.3 Finding outstanding players

As what we have said before, this part is done with **Kmeans++**. With **Elbow** method, showed in *Figure 9*, we found the appropriate number of clusters is 10. Then we clustered and got a plot like what you can see in *Figure 10*. Just to let you know, there is a plot with names of players attached in the Annex section. Still, we don't have time to finish this part, we haven't done the analysis of the main characteristics of each cluster, so that we won't be able to figure out players of which cluster can be seen as outstanding.

But we made a guess, the cluster that is the farthest away from the others may be the most outstanding one, or the least. We can continue this part as future work later.

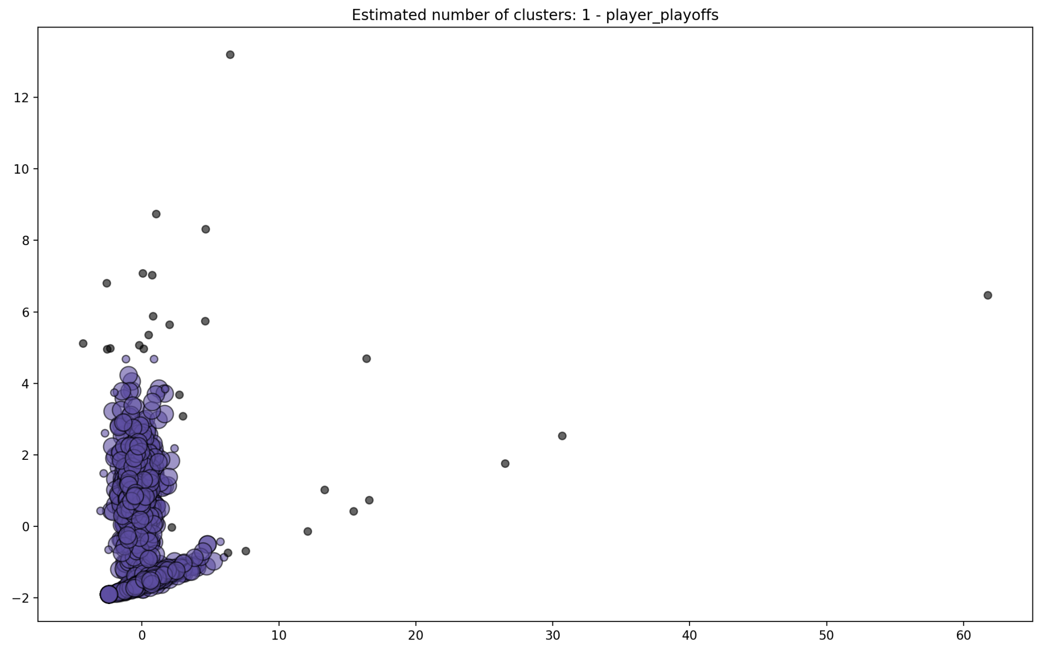



Figure 7: Plot of DBSCAN clustring

| 1 | ilkid | firstname | lastname | gp | minutes | pts | dreb | oreb | reb | asts | stl | blk | turnover | pf | fga | fgm | fta | ftm | tpa | tpm |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 77 | BREWEJA01 | Jamison | Brewer | 2 | 6 | -0.68038635 | 3.54665183 | 6.44700639 | 3.88675404 | 0.83867720 | -0.65438045 | -0.36398182 | 3.20388489 | 2.69381024 | -0.11531909 | -1.18969088 | 1.44455845 | 1 | -0.41238581 | -0.2403 |
| 78 | BRITTMI01 | Mike | Brittain | 1 | 2 | 1.76083703 | -0.81729117 | 9.99981290 | 2.52086339 | -0.82263293 | -0.65438045 | 14.5980314 | -0.77151527 | -1.02339191 | 1.91887527 | 2.69096736 | -0.81354132 | 0 | -0.41238581 | -0.2403 |
| 79 | BROOKKE01 | Kevin | Brooks | 2 | 5 | 1.76083703 | 1.80107462 | 3.60476117 | 1.70132899 | -0.82263293 | -0.65438045 | -0.36398182 | -0.77151527 | -1.02339191 | 3.13939189 | 1.91483571 | 1.89617841 | 1 | 2.83202264 | -0.2403 |
| 80 | CERVIAL01 | Al | Cervi | 27 | 116 | 4.28624054 | -0.81729117 | -0.65860664 | 1.46112063 | 8.28592951 | -0.65438045 | -0.36398182 | -0.77151527 | 5.51375671 | 3.04995748 | 2.15570415 | 7.01194240 | 116 | -0.41238581 | -0.2403 |
| 81 | CLOSSBI01 | Bill | Closs | 11 | 21 | 13.1997123 | -0.81729117 | -0.65860664 | 4.66726299 | 10.0945479 | -0.65438045 | -0.36398182 | -0.77151527 | 10.3052241 | 16.7394342 | 12.1154231 | 11.4447146 | 31 | -0.41238581 | -0.2403 |
| 82 | COOPEDU01 | Duane | Cooper | 2 | 4 | -1.16863103 | -0.81729117 | 9.99981290 | 2.52086339 | 1.66933226 | -0.65438045 | -0.36398182 | -0.77151527 | -1.02339191 | 3.44452105 | -1.18969088 | -0.81354132 | 0 | 11.7541459 | -0.2403 |
| 83 | EDMONKE01 | Keith | Edmonson | 1 | 2 | 1.76083703 | -0.81729117 | 9.99981290 | 2.52086339 | 4.16129746 | -0.65438045 | -0.36398182 | -0.77151527 | -1.02339191 | 0.39322949 | 2.69096736 | -0.81354132 | 0 | -0.41238581 | -0.2403 |
| 84 | HAYESST01 | Steve | Hayes | 1 | 1 | 4.69030510 | 12.2745378 | -0.65860664 | 6.61853535 | -0.82263293 | -0.65438045 | -0.36398182 | -0.77151527 | -1.02339191 | 1.91887527 | 6.57162561 | -0.81354132 | 0 | -0.41238581 | -0.2403 |
| 85 | HEALSH01 | Shane | Heal | 2 | 3 | 4.69030510 | -0.81729117 | -0.65860664 | -1.57680857 | 2.49998733 | -0.65438045 | -0.36398182 | -0.77151527 | -1.02339191 | 0.90177808 | 3.98452011 | -0.81354132 | 0 | 10.4023090 | 19.7999 |
| 86 | HENRYCA01 | Carl | Henry | 1 | 2 | 3.22557106 | -0.81729117 | -0.65860664 | -1.57680857 | -0.82263293 | -0.65438045 | -0.36398182 | -0.77151527 | -1.02339191 | 0.39322949 | 2.69096736 | -0.81354132 | 0 | 7.69863533 | 14.7898 |
| 87 | HOLLAJO01 | Joe | Holland | 21 | 37 | 6.66967542 | 0.24420847 | 0.49365492 | 2.18861971 | 7.25941635 | 4.25627905 | -0.36398182 | 1.80712267 | 8.42031086 | 7.77405636 | 8.03998278 | 3.39750962 | 11 | -0.41238581 | -0.2403 |
| 88 | HOLZMRE01 | Red | Holzman | 24 | 79 | 2.61372014 | -0.81729117 | -0.65860664 | 1.12039322 | 2.07940248 | -0.65438045 | -0.36398182 | -0.77151527 | 1.23516129 | 2.92309780 | 2.54360059 | 3.13098740 | 26 | -0.41238581 | -0.2403 |
| 89 | JOHNSAR01 | Arnie | Johnson | 22 | 166 | 2.64320693 | -0.81729117 | -0.65860664 | 6.17420948 | 3.32063450 | -0.65438045 | -0.36398182 | -0.77151527 | 3.72387831 | 1.55124978 | 1.70911407 | 4.16516060 | 92 | -0.41238581 | -0.2403 |
| 90 | JONESWA02 | Wallace | Jones | 6 | 8 | 26.2951321 | -0.81729117 | -0.65860664 | -1.57680857 | 26.5889842 | -0.65438045 | -0.36398182 | -0.77151527 | 24.9970232 | 27.8548534 | 21.1240940 | 27.9772309 | 29 | -0.41238581 | -0.2403 |
| 91 | JUDKIJE01 | Jeff | Judkins | 7 | 10 | 1.46789022 | 0.49189172 | 5.73644508 | 1.70132899 | -0.82263293 | 2.97950758 | -0.36398182 | -0.77151527 | -1.02339191 | 1.30861696 | 1.91483571 | -0.81354132 | 0 | 4.45422687 | 2.76571 |
| 92 | LAUDEPR01 | Priest | Lauderdale | 3 | 7 | -1.16863103 | 1.05297011 | 2.38665608 | 0.76471826 | -0.82263293 | -0.65438045 | -0.36398182 | 9.45094231 | 0.03866584 | 0.17528009 | -1.18969088 | -0.81354132 | 0 | -0.41238581 | -0.2403 |
| 93 | MCKINHO01 | Horace | Mckinney | 7 | 20 | 5.56914552 | -0.81729117 | -0.65860664 | 4.97946657 | 5.65647658 | -0.65438045 | -0.36398182 | -0.77151527 | 9.75649434 | 7.41120007 | 6.18355978 | 2.57360834 | 8 | -0.41238581 | -0.2403 |
| 94 | MEYERLO01 | Loren | Meyer | 3 | 14 | -1.16863103 | 2.92323140 | 2.38665608 | 1.93548168 | -0.11064287 | -0.65438045 | 3.91087912 | 4.33971351 | -1.02339191 | -0.26061869 | -1.18969088 | -0.81354132 | 0 | 0.74633149 | -0.2403 |
| 95 | MOSLEGL01 | Glenn | Mosley | 3 | 6 | 1.27259235 | -0.81729117 | -0.65860664 | -0.21091792 | 0.83867720 | -0.65438045 | 4.62335594 | -0.77151527 | -1.02339191 | 0.39322949 | 1.39741461 | 2.57360834 | 1 | -0.41238581 | -0.2403 |
| 96 | MURRAKE01 | Ken | Murray | 6 | 15 | 6.64328381 | -0.81729117 | -0.65860664 | 6.07217909 | 5.82260759 | -0.65438045 | -0.36398182 | -0.77151527 | 5.41975849 | 10.6659110 | 7.60646781 | 2.34779836 | 6 | -0.41238581 | -0.2403 |
| 97 | RANDAMA01 | Mark | Randall | 2 | 6 | -1.16863103 | 7.91059484 | 2.89419987 | 5.25264470 | -0.82263293 | -0.65438045 | 4.62335594 | 3.20388489 | 0.21567547 | -0.62386768 | -1.18969088 | -0.81354132 | 0 | -0.41238581 | -0.2403 |
| 98 | RATKOGE01 | George | Ratkovicz | 24 | 59 | 11.6912372 | -0.81729117 | -0.65860664 | 10.2300428 | 5.59734520 | -0.65438045 | -0.36398182 | -0.77151527 | 10.8212522 | 8.95236088 | 9.33412809 | 16.1796163 | 99 | -0.41238581 | -0.2403 |
| 99 | RAUTILE01 | Leo | Rautins | 3 | 5 | 0.58904980 | -0.81729117 | 7.86812899 | 1.70132899 | 1.17093922 | 6.61339561 | -0.36398182 | -0.77151527 | 1.95036981 | 0.69835865 | 0.36257241 | -0.81354132 | 0 | 6.07643110 | 5.77175 |
| 100 | SCHEFTO01 | Tom | Scheffler | 3 | 10 | 0.88199661 | 1.80107462 | 5.73644508 | 2.52086339 | -0.82263293 | 2.97950758 | -0.36398182 | 1.61372482 | -1.02339191 | -0.21702881 | 0.36257241 | 1.89617841 | 3 | -0.41238581 | -0.2403 |
| 101 | STROEJO01 | John | Stroeder | 1 | 1 | 7.61977317 | -0.81729117 | -0.65860664 | -1.57680857 | -0.82263293 | -0.65438045 | -0.36398182 | -0.77151527 | -1.02339191 | 1.91887527 | 6.57162561 | -0.81354132 | 0 | 15.8096564 | 29.8200 |
| 102 | VAUGHDA02 | David | Vaughn | 1 | 1 | -1.16863103 | -0.81729117 | -0.65860664 | -1.57680857 | -0.82263293 | -0.65438045 | 29.5600447 | -0.77151527 | -1.02339191 | -1.13241628 | -1.18969088 | -0.81354132 | 0 | -0.41238581 | -0.2403 |
| 103 | WARDHE01 | Henry | Ward | 1 | 1 | 10.5492412 | -0.81729117 | -0.65860664 | -1.57680857 | -0.82263293 | -0.65438045 | -0.36398182 | -0.77151527 | -1.02339191 | 8.02145838 | 14.3329421 | -0.81354132 | 0 | -0.41238581 | -0.2403 |
| 104 | WHEATDE01 | Dejuan | Wheat | 1 | 3 | 0.78434767 | 3.54665183 | -0.65860664 | 1.15497273 | -0.82263293 | 11.4585796 | -0.36398182 | -0.77151527 | -1.02339191 | 0.90177808 | 1.39741461 | -0.81354132 | 0 | -0.41238581 | -0.2403 |

Figure 8: List of outliers

# 6 Conclusion and Future Work

During this project, we tried to do the outliers detection, prediction of outcomes, with techniques like **DBSCAN**, **Kmeans++**, **KNN**, **SVM**, **Neural Networks**, etc. And eventually we got a pretty good result. We are really happy that we, every member of our team, never gave up and kept trying and learning new things though they are kind of hard to us. And meanwhile, during this project, we learned so many techniques and gained much project experience. For the future work, as what we have said, we can try to finish the analysis part of our unsupervised learning part. We can continue to try to figure out why the outliers are outliers, which cluster of players among all the clusters can be seen as outstanding or not that outstanding. Besides, in the future, we can try to collect more data like stats which are about the win-loss situation between every pair of two teams. With these data, we can set them as labels and I guess maybe we can get more valuable and concrete predictions. Although there are still some details to retouch, we've gained a lot during the process and we are proud of what we have done. We think we had a pretty good start in machine learning.

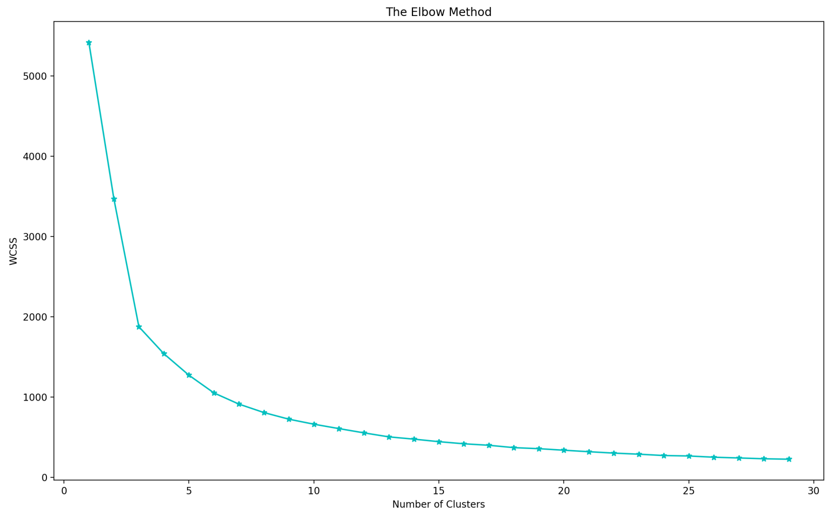



Figure 9: Elbow for finding the appropriate number of clusters

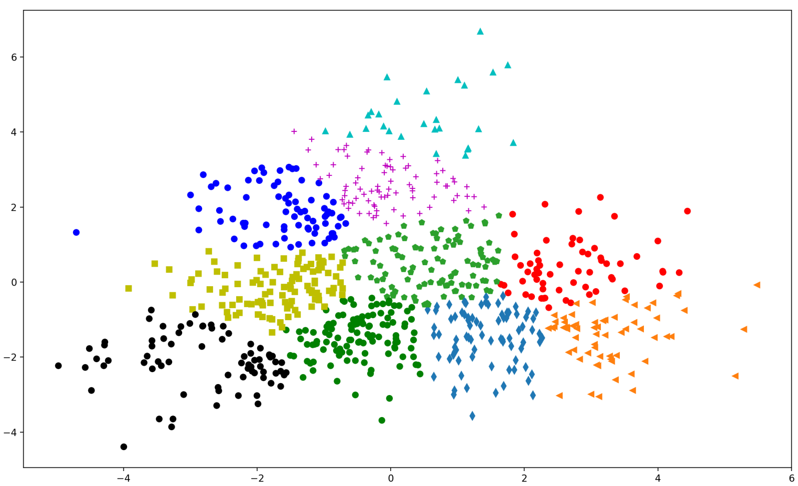


Figure 10: Clustering of players

# 7 Contributions

All of the work described above was done by our group members: Yichen Mu, Fangxu Zhou, Fu Hao, Jingxuan Guo.

# References

[1] Nadir, N. (2018) *Assessing NBA player similarity with Machine Learning (R)*

[2] Sebastian, R. (2018) *An overview of proxy-label approaches for semi-supervised learning*

# Annexe

| ilkid | firstname | lastname | gp | minutes | pts | dreb | oreb | reb | asts | stl | blk | turnover | pf | fga | fgm | fta | ftm | tpa | tpm |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BREWEJA01 | Jamison | Brewer | 2 | 6 | -0.68038635 | 3.546651833 | 6.447006390 | 3.886754046 | 0.838677201 | -0.65438045 | -0.36398182 | 3.203884897 | 2.693810245 | -0.11531909 | -1.18969088 | 1.444558455 | 1 | -0.41238581 | -0.24032266119363793 |
| BRITTMI01 | Mike | Brittain | 1 | 2 | 1.760837033 | -0.81729117 | 9.999812908 | 2.520863390 | -0.82263293 | -0.65438045 | 14.59803148 | -0.77151527 | -1.02339191 | 1.918875272 | 2.690967363 | -0.81354132 | 0 | -0.41238581 | -0.24032266119363793 |
| BROOKKE01 | Kevin | Brooks | 2 | 5 | 1.760837033 | 1.801074629 | 3.604761176 | 1.701328996 | -0.82263293 | -0.65438045 | -0.36398182 | -0.77151527 | -1.02339191 | 3.139391894 | 1.914835714 | 1.896178412 | 1 | 2.832022646 | -0.24032266119363793 |
| CERVIAL01 | Al | Cervi | 27 | 116 | 4.286240542 | -0.81729117 | -0.65860664 | 1.461120639 | 8.285929516 | -0.65438045 | -0.36398182 | -0.77151527 | 5.513756711 | 3.049957487 | 2.155704157 | 7.011942405 | 116 | -0.41238581 | -0.24032266119363793 |
| CLOSSBI01 | Bill | Closs | 11 | 21 | 13.19971235 | -0.81729117 | -0.65860664 | 4.667262992 | 10.09454793 | -0.65438045 | -0.36398182 | -0.77151527 | 10.30522419 | 16.73943425 | 12.11542311 | 11.44471464 | 31 | -0.41238581 | -0.24032266119363793 |
| COOPEDU01 | Duane | Cooper | 2 | 4 | -1.16863103 | -0.81729117 | 9.999812908 | 2.520863390 | 1.669332267 | -0.65438045 | -0.36398182 | -0.77151527 | -1.02339191 | 3.444521050 | -1.18969088 | -0.81354132 | 0 | 11.75414591 | -0.24032266119363793 |
| EDMONKE01 | Keith | Edmonson | 1 | 2 | 1.760837033 | -0.81729117 | 9.999812908 | 2.520863390 | 4.161297464 | -0.65438045 | -0.36398182 | -0.77151527 | -1.02339191 | 0.393229495 | 2.690967363 | -0.81354132 | 0 | -0.41238581 | -0.24032266119363793 |
| HAYESST01 | Steve | Hayes | 1 | 1 | 4.690305104 | 12.27453784 | -0.65860664 | 6.618535358 | -0.82263293 | -0.65438045 | -0.36398182 | -0.77151527 | -1.02339191 | 1.918875272 | 6.571625612 | -0.81354132 | 0 | -0.41238581 | -0.24032266119363793 |
| HEALSH01 | Shane | Heal | 2 | 3 | 4.690305104 | -0.81729117 | -0.65860664 | -1.57680857 | 2.499987332 | -0.65438045 | -0.36398182 | -0.77151527 | -1.02339191 | 0.901778088 | 3.984520113 | -0.81354132 | 0 | 10.40230905 | 19.79992807959356 |
| HENRYCA01 | Carl | Henry | 1 | 2 | 3.225571069 | -0.81729117 | -0.65860664 | -1.57680857 | -0.82263293 | -0.65438045 | -0.36398182 | -0.77151527 | -1.02339191 | 0.393229495 | 2.690967363 | -0.81354132 | 0 | 7.698635338 | 14.78986539439676 |
| HOLLAJO01 | Joe | Holland | 21 | 37 | 6.669675422 | 0.244208475 | 0.493654928 | 2.188619716 | 7.259416359 | 4.256279055 | -0.36398182 | 1.807122672 | 8.420310869 | 7.774056363 | 8.039982787 | 3.397509620 | 11 | -0.41238581 | -0.24032266119363793 |
| HOLZMRE01 | Red | Holzman | 24 | 79 | 2.613720143 | -0.81729117 | -0.65860664 | 1.120393223 | 2.079402489 | -0.65438045 | -0.36398182 | -0.77151527 | 1.235161296 | 2.923097809 | 2.543600595 | 3.130987408 | 26 | -0.41238581 | -0.24032266119363793 |
| JOHNSAR01 | Arnie | Johnson | 22 | 166 | 2.643206934 | -0.81729117 | -0.65860664 | 6.174209482 | 3.320634506 | -0.65438045 | -0.36398182 | -0.77151527 | 3.723878313 | 1.551249784 | 1.709114072 | 4.165160604 | 92 | -0.41238581 | -0.24032266119363793 |
| JONESWA02 | Wallace | Jones | 6 | 8 | 26.29513212 | -0.81729117 | -0.65860664 | -1.57680857 | 26.58898424 | -0.65438045 | -0.36398182 | -0.77151527 | 24.99702320 | 27.85485348 | 21.12409404 | 27.97723091 | 29 | -0.41238581 | -0.24032266119363793 |
| JUDKIJE01 | Jeff | Judkins | 7 | 10 | 1.467890226 | 0.491891727 | 5.736445087 | 1.701328996 | -0.82263293 | 2.979507583 | -0.36398182 | -0.77151527 | -1.02339191 | 1.308616962 | 1.914835714 | -0.81354132 | 0 | 4.454226877 | 2.765714949924442 |
| LAUDEPR01 | Priest | Lauderdale | 3 | 7 | -1.16863103 | 1.052970114 | 2.386656084 | 0.764718260 | -0.82263293 | -0.65438045 | -0.36398182 | 9.450942317 | 0.038665845 | 0.175280099 | -1.18969088 | -0.81354132 | 0 | -0.41238581 | -0.24032266119363793 |
| MCKINHO01 | Horace | Mckinney | 7 | 20 | 5.569145525 | -0.81729117 | -0.65860664 | 4.979466571 | 5.656476583 | -0.65438045 | -0.36398182 | -0.77151527 | 9.756494348 | 7.411200070 | 6.183559787 | 2.573608347 | 8 | -0.41238581 | -0.24032266119363793 |
| MEYERLO01 | Loren | Meyer | 3 | 14 | -1.16863103 | 2.923231403 | 2.386656084 | 1.935481680 | -0.11064287 | -0.65438045 | 3.910879124 | 4.339713518 | -1.02339191 | -0.26061869 | -1.18969088 | -0.81354132 | 0 | 0.746331492 | -0.24032266119363793 |
| MOSLEGL01 | Glenn | Mosley | 3 | 6 | 1.272592355 | -0.81729117 | -0.65860664 | -0.21091792 | 0.838677201 | -0.65438045 | 4.623355949 | -0.77151527 | -1.02339191 | 0.393229495 | 1.397414614 | 2.573608347 | 1 | -0.41238581 | -0.24032266119363793 |
| MURRAKE01 | Ken | Murray | 6 | 15 | 6.643283818 | -0.81729117 | -0.65860664 | 6.072179096 | 5.822607596 | -0.65438045 | -0.36398182 | -0.77151527 | 5.419758495 | 10.66591106 | 7.606467812 | 2.347798369 | 6 | -0.41238581 | -0.24032266119363793 |
| RANDAMA01 | Mark | Randall | 2 | 6 | -1.16863103 | 7.910594840 | 2.894199873 | 5.252644702 | -0.82263293 | -0.65438045 | 4.623355949 | 3.203884897 | 0.215675472 | -0.62386768 | -1.18969088 | -0.81354132 | 0 | -0.41238581 | -0.24032266119363793 |
| RATKOGE01 | George | Ratkovicz | 24 | 59 | 11.69123727 | -0.81729117 | -0.65860664 | 10.23004285 | 5.597345205 | -0.65438045 | -0.36398182 | -0.77151527 | 10.82125225 | 8.952360889 | 9.334128094 | 16.17961636 | 99 | -0.41238581 | -0.24032266119363793 |
| RAUTILE01 | Leo | Rautins | 3 | 5 | 0.589049805 | -0.81729117 | 7.868128997 | 1.701328996 | 1.170939227 | 6.613395619 | -0.36398182 | -0.77151527 | 1.950369813 | 0.698358651 | 0.362572414 | -0.81354132 | 0 | 6.076431108 | 5.771752561042522 |
| SCHEFTO01 | Tom | Scheffler | 3 | 10 | 0.881996612 | 1.801074629 | 5.736445087 | 2.520863390 | -0.82263293 | 2.979507583 | -0.36398182 | 1.613724826 | -1.02339191 | -0.21702881 | 0.362572414 | 1.896178412 | 3 | -0.41238581 | -0.24032266119363793 |
| STROEJO01 | John | Stroeder | 1 | 1 | 7.619773174 | -0.81729117 | -0.65860664 | -1.57680857 | -0.82263293 | -0.65438045 | -0.36398182 | -0.77151527 | -1.02339191 | 1.918875272 | 6.571625612 | -0.81354132 | 0 | 15.80965649 | 29.820053449987157 |
| VAUGHDA02 | David | Vaughn | 1 | 1 | -1.16863103 | -0.81729117 | -0.65860664 | -1.57680857 | -0.82263293 | -0.65438045 | 29.56004479 | -0.77151527 | -1.02339191 | -1.13241628 | -1.18969088 | -0.81354132 | 0 | -0.41238581 | -0.24032266119363793 |
| WARDHE01 | Henry | Ward | 1 | 1 | 10.54924124 | -0.81729117 | -0.65860664 | -1.57680857 | -0.82263293 | -0.65438045 | -0.36398182 | -0.77151527 | -1.02339191 | 8.021458381 | 14.33294211 | -0.81354132 | 0 | -0.41238581 | -0.24032266119363793 |
| WHEATDE01 | Dejuan | Wheat | 1 | 3 | 0.784347677 | 3.546651833 | -0.65860664 | 1.154972733 | -0.82263293 | 11.45857966 | -0.36398182 | -0.77151527 | -1.02339191 | 0.901778088 | 1.397414614 | -0.81354132 | 0 | -0.41238581 | -0.24032266119363793 |

Table 1: Full list of outliers ("Outliers detection" part)

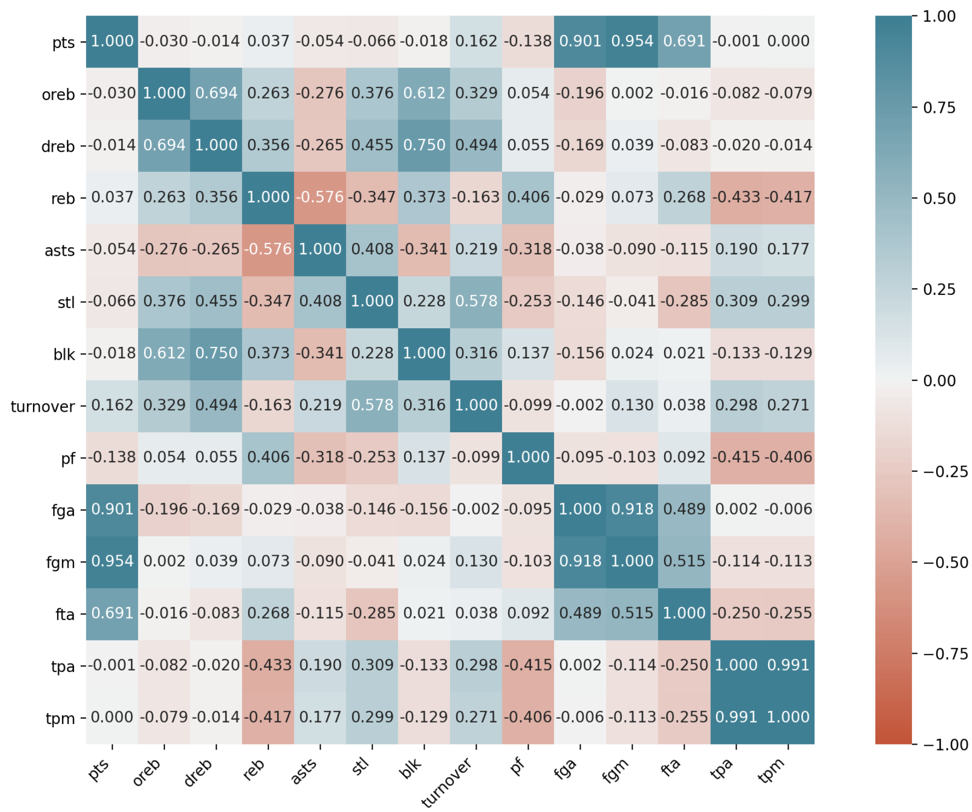



Figure 11: Correlation between normalized features("Outliers detection" part)

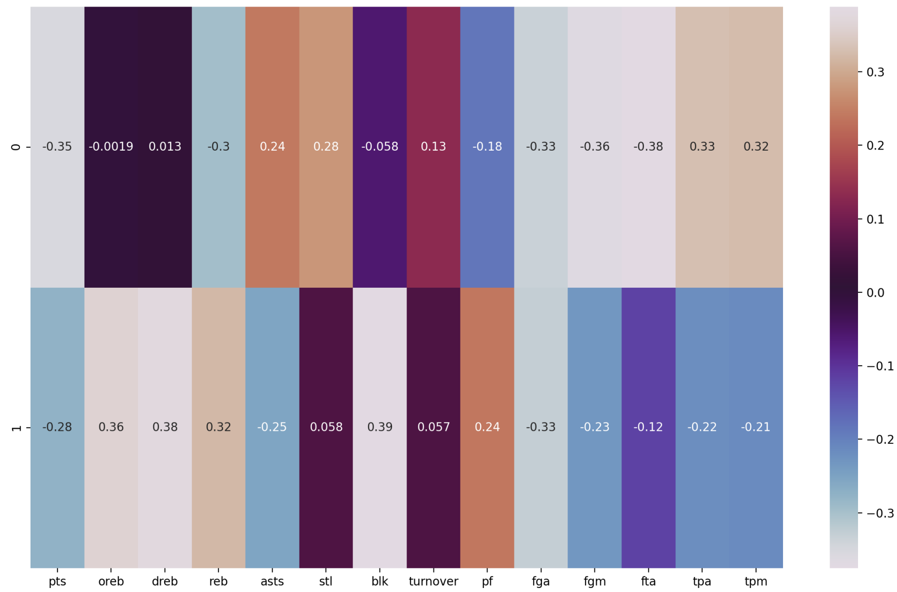



Figure 12: Correlation between features and principal components ("Outliers detection" part)

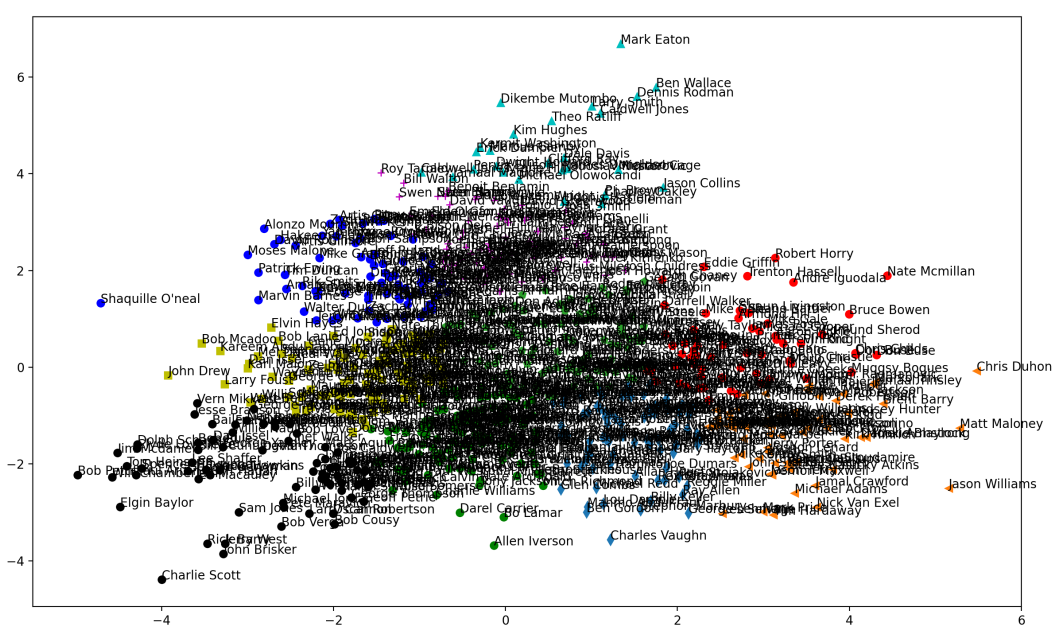



Figure 13: Plot of clustering with names attached ("Finding outstanding players" part)

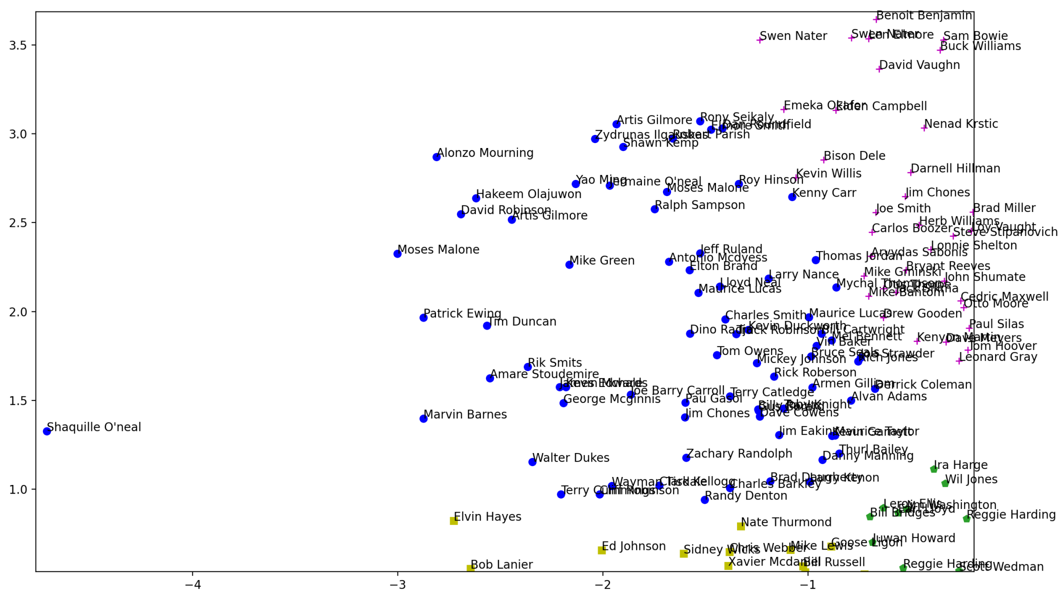



Figure 14: Plot of clustering with names attached - zoomed in ("Finding outstanding players" part)