Report FIFA 2019 DataBase

Loïc Clerc

17 juin 2019

Introduction

In this project, we are analyzing the FIFA 2019 player Data set. This data set provides a lot of information - name, age, Nationality, potential, wage, skills, etc.- 89 columns overall - on about 18'000 players. Here is a first idea of what the database looks like for a select range of columns:

```
##                Name Age Nationality Potential     Value    Wage
## 1          L. Messi  31   Argentina        94 â,¬110.5M â,¬565K
## 2 Cristiano Ronaldo  33    Portugal        94    â,¬77M â,¬405K
## 3         Neymar Jr  26      Brazil        93 â,¬118.5M â,¬290K
## 4            De Gea  27       Spain        93    â,¬72M â,¬260K
## 5      K. De Bruyne  27     Belgium        92   â,¬102M â,¬355K
## 6         E. Hazard  27     Belgium        91    â,¬93M â,¬340K
##   International.Reputation Finishing HeadingAccuracy
## 1                        5        95              70
## 2                        5        94              89
## 3                        5        87              62
## 4                        4        13              21
## 5                        4        82              55
## 6                        4        84              61
```

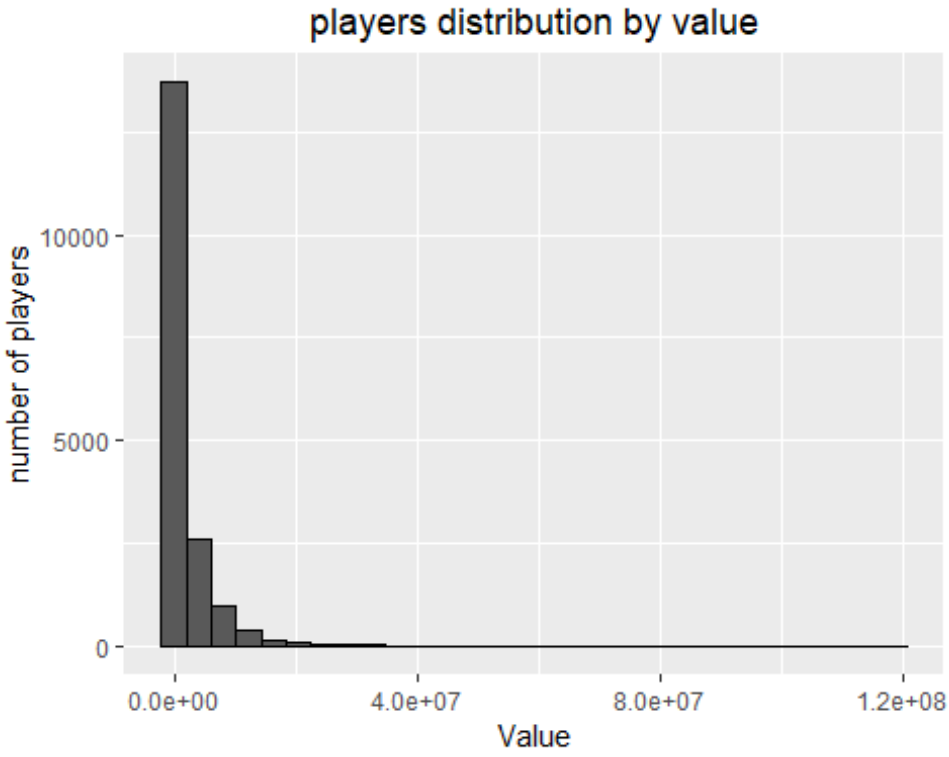
Note that the quite aweful character chains in column "Value" and "Wage" are the encoding for the "Euro" sign. This will obviously require some data wragling. The database may be found and downloaded from the Kaggle website (it is required to create an account) or from my Github directory "FIFA-2019-Project" (file "data.csv"):

https://www.kaggle.com/karangadiya/fifa19

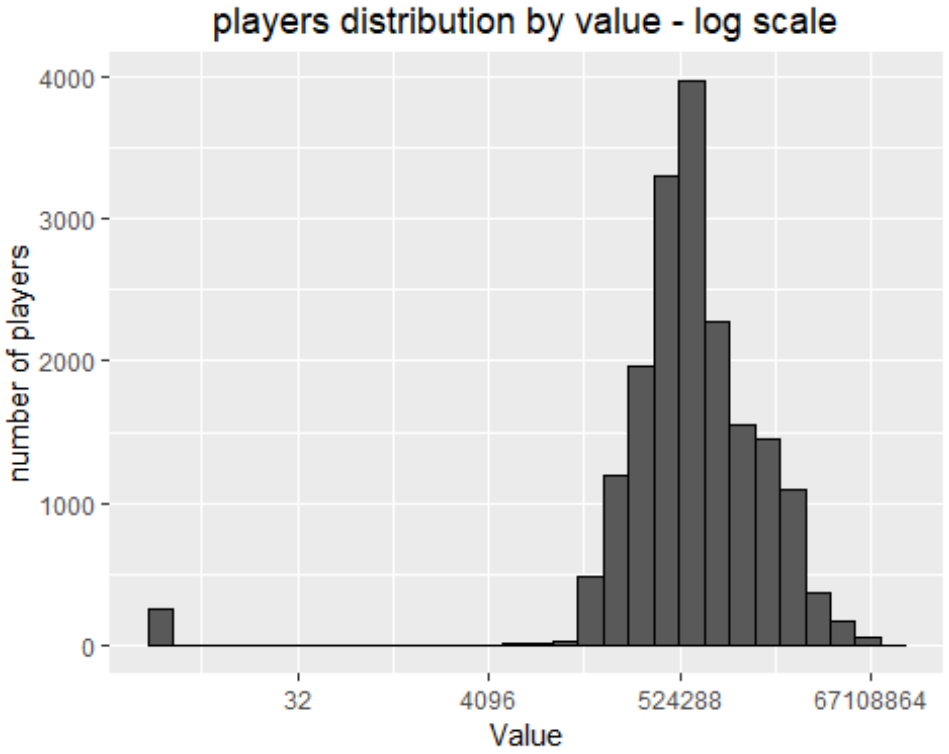
https://github.com/loicclerc/FIFA-2019-Project

After reflexion, it seemed quite clear that being able to evaluate the value of a player in the market (column value of above picture) is very useful. Indeed, taking the role of a professional team, it would be very practical to have a tool that would evaluate what a player's value is, either in order to set up his price during a transfer or to compare it to the value of a real offer that could be made by another team in order to see if the deal seems good or not.

Exploring a little bit the distribution of the values, it is clear that the task will be quite challenging as the range of different values is very large, with quite a lot of outlayers:



Indeed, we see that the values range from 0 to ~100M Euro with very few players when we go over 40M Euro. We get a better visualization when looking at the same graphic with a log-2 scale:



We see that some player have basically a value of 0 while most of them are worth between 100K Euro and 1M Euro. Exploring the bottom of the list, we see that a non neglectable number of players are considered having no or a very low value:

```
## # A tibble: 217 x 2
##    Value n_player
##    <dbl>    <int>
##  1     0      252
##  2 10000       15
##  3 20000       21
##  4 30000       23
##  5 40000       65
##  6 50000      127
##  7 60000      150
##  8 70000      139
##  9 80000      114
## 10 90000      136
## # ... with 207 more rows
```

At the other end of the list, three players are worth more than a million Euro:

```
## # A tibble: 217 x 2
##        Value n_player
##        <dbl>    <int>
##  1 118500000        1
##  2 110500000        1
##  3 102000000        1
##  4  93000000        1
##  5  89000000        1
##  6  83500000        1
##  7  81000000        1
##  8  80000000        1
##  9  78000000        1
## 10  77000000        2
## # ... with 207 more rows
```

In this project, attempts will be made to build a model to evaluate the values of the players from the other data available. Thus, the method followed will be regression.

The key steps performed are the following:

1. Loading the data and data wrangling to convert the characters and to obtain workable numericals
2. Data Visualization and pre-processing of the data, especially scalling of the predictors
3. Model testing and optimization
4. Analysis of the results
5. Conclusion

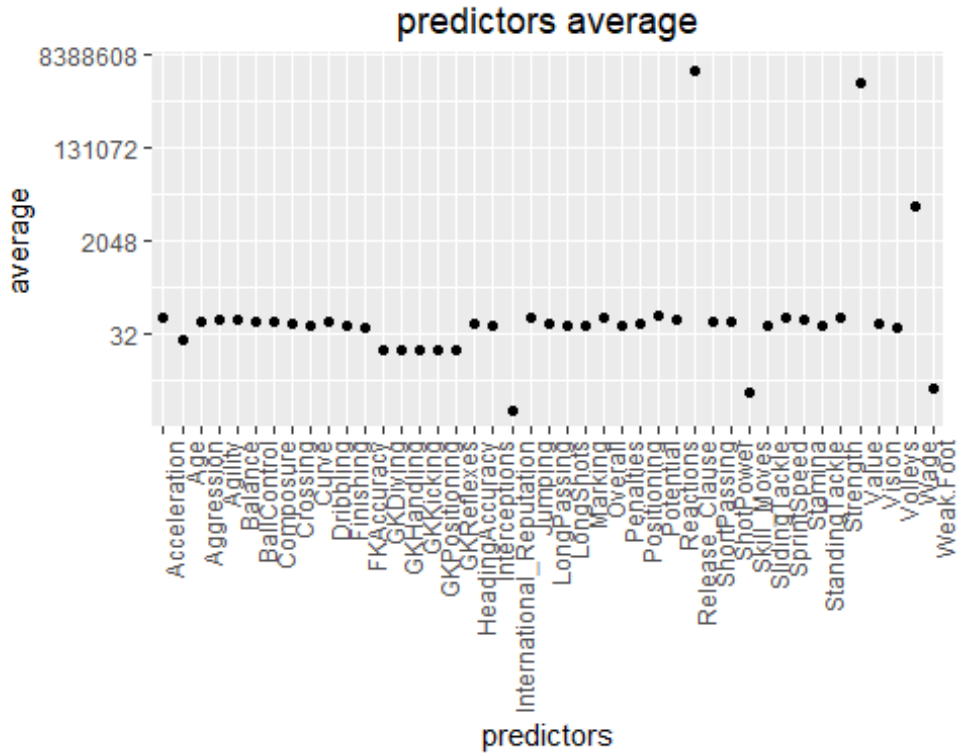
Method and modelling approach

Before anything else, the first step to perform is to wrangle the data in order to obtain workable numerical data for the columns that are considered as relevant for the model but that contain caracters that must be removed. They are mostly due to two things, the “Euro” sign and the suffix K and M. Difficulty here is that there are signs to remove before and after the actual numbers to extract and also that depending of the coefficents “K” or “M” (or if there is no coefficient) there would be a factor to apply to the results. For instance, for the column “Value”, the below table summarizes the number of occurences in each of the three groups:

```
## .
##     0     K     M
##   252 11108  6847
```

Besides this, a few non numerical values (na) had to be removed. Finally, the columns that were intuitively considered as meaningful for the modelling were extracted from the initial database and some column names were modified because contraining dots.

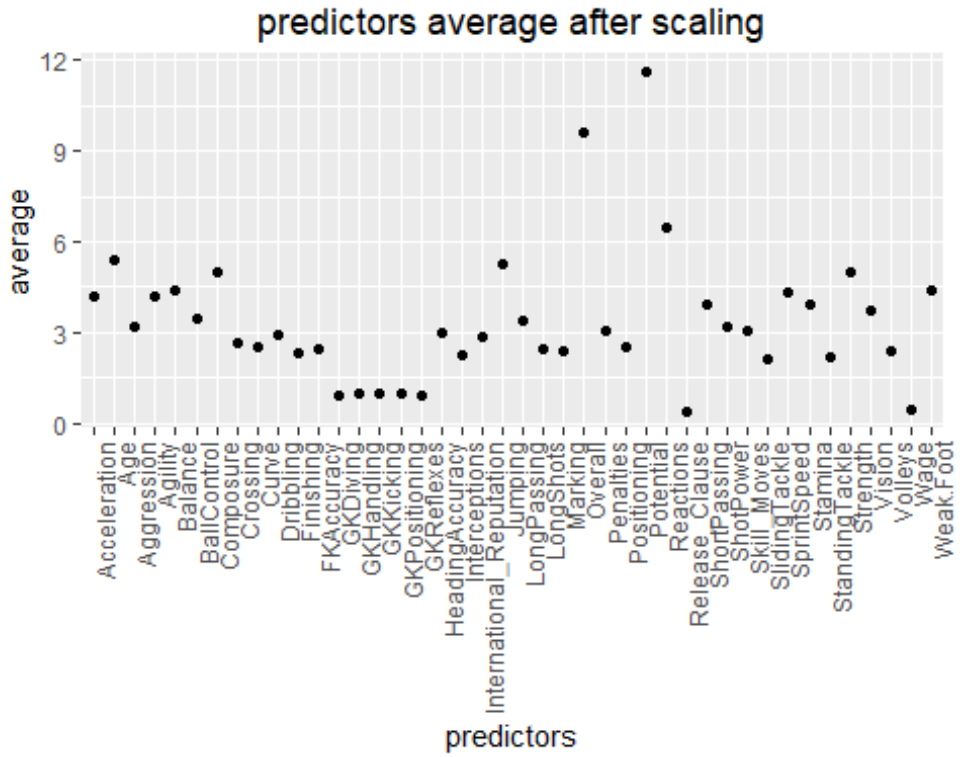
At this stage, a quick data visualization shows that the average values are considerably different from one column to the other (x-scale in log-2):



Especially, the order of magnitude of “Value”, “Wage”" and “Release_Clause” is much higher than for the other variables. “Value” is our output and should therefore not be modified. However, the other columns - the predictors - must be scaled. Note that to make it properly,

it is necessary to first split the training and the test set as it is not allowed to use the test set in the scalling operation.

The scaled data is, as expected, much more balanced:



The data is now ready for the modelling. To test the error of our model, two metrics will be used, given by the following formulas:

```
    #function RMSE
    RMSE <- function(true_values, predicted_values){
      sqrt(mean((true_values - predicted_values)^2))
    }
  #function WMAPE
    WMAPE <- function(true_values, predicted_values){
      sum(abs(true_values - predicted_values))/sum(true_values)
    }
```

In addition to the RMSE metric, it is decided to use another one, the WMAPE. Indeed, as the Value that we are trying to fit ranges from 0 to 100M, the RMSE that will result from the modelling are quite huge and therefore not so easy to interpretate. As the WMAPE metric is a ratio over the sum of the true values, it is much easier to interpretate.

We start with the naive approach of predicting the global average for all players.

```
  RMSE_Global_Average
```

```
## [1] 6329785
```

```
  WMAPE_Global_Average
```

```
## [1] 1.132273
```

We see that, as expected, the RMSE value is quite huge. The next obvious model to test is the linear model.

```
  RMSE_lm
```

```
## [1] 1337315
```

```
  WMAPE_lm
```

```
## [1] 0.2090762
```

This surely shows a great improvement over the obtained result with the global average. However, it seems clear that much better can be done. Running a logistic regression model leads to very similar results than the linear model:

```
  RMSE_glm
```

```
## [1] 1337097
```

```
  WMAPE_glm
```

```
## [1] 0.209069
```

The next model that is tried is knn. In spite of much longer running time, the obtained result is worse than that of lm and glm.

```
  RMSE_knn
```

```
## [1] 1452063
```

```
  WMAPE_knn
```

```
## [1] 0.2195284
```

Attempts have been made to optimized k with cross-validation but this did not lead to any significant improvement, that's why this part is not shown is the report. The next model to be tried is rpart, fitting a single regression tree:

```
  RMSE_rpart
```

```
## [1] 1731746
```

```
  WMAPE_rpart
```

```
## [1] 0.2964264
```

The result is even worse than that of lm, glm and knn. Optimzing cp did not help much which is why it is not presented here. However, as explained in the lecture "Machine Learning", averaging over a lot of trees shows always better results than having a single tree. This is why randomForest is the next model that is tried out:

```
  RMSE_rf
```

```
## [1] 871768.3
```

```
  WMAPE_rf
```

```
## [1] 0.07010057
```

This time, the improvement is very significant. So far, this is the result to beat. The next model that is tested is xgbTree. We see that this model does about as well as randomForest:

```
    RMSE_xgbTree
```

```
## [1] 900097.3
```

```
    WMAPE_xgbTree
```

```
## [1] 0.08059646
```

Two other models are tested: Gamboost & gamLoess. Many other algorithm were tested but they would not converge in a reasonabe time. Adding them to the table gives us the following results:

```
  #gamboost model
    RMSE_gamboost
```

```
## [1] 1130319
```

```
    WMAPE_gamboost
```

```
## [1] 0.1467369
```

```
  #gamLoess model
    RMSE_gamLoess
```

```
## [1] 985049.7
```

```
    WMAPE_gamLoess
```

```
## [1] 0.1487534
```

Results Analysis and further optimization

The below table summarizes the so far obtained results with the different method explained above:

method	RMSE	WMAPE
Global Average	6329784.7	1.1322728
Lm model	1337314.6	0.2090762
glm model	1337097.0	0.2090690
knn model	1452063.4	0.2195284

Rpart model	1731746.3	0.2964264
RF model	871768.3	0.0701006
xgbTree model	900097.3	0.0805965
gamboost model	1130318.5	0.1467369
gamLoess model	985049.7	0.1487534

Using the tunning parameter mtry - rf could be furhter optimized.

```
  #Random Forest model - Best tunning parameters
    RMSE_rf_mtry_20

## [1] 840821.6

    WMAPE_rf_mtry_20

## [1] 0.06452538
```

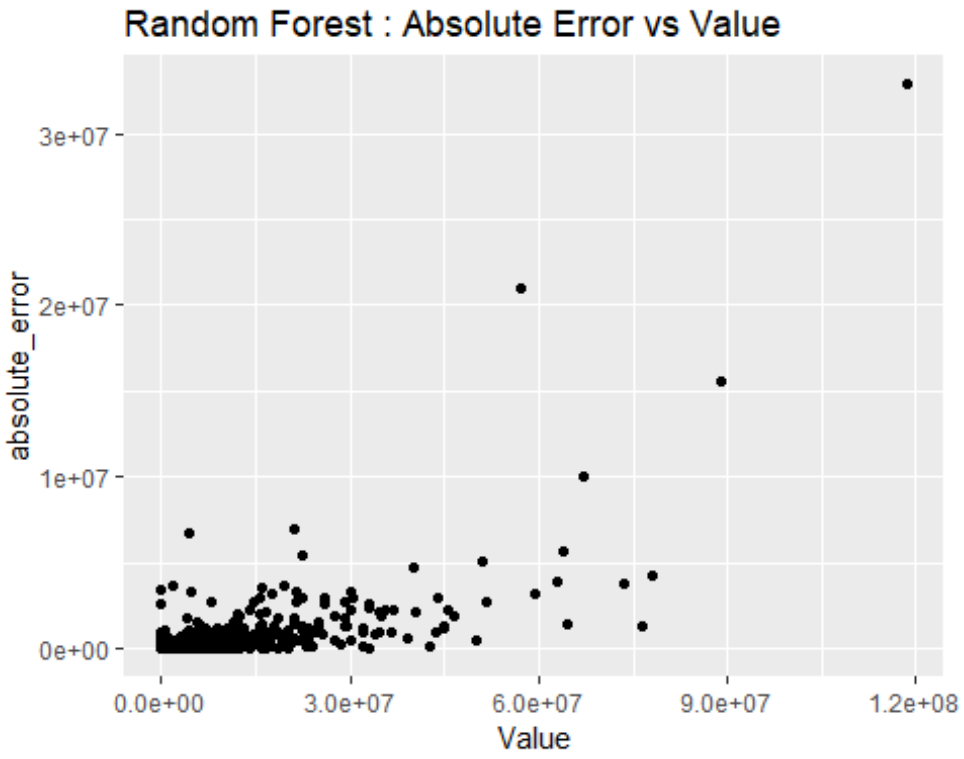
Inspecting the training object of XgbTree, we see that the model is tested for 108 combinations of 7 tunning parameters. It is possible to optimize the parameters and a better result could be obtained:

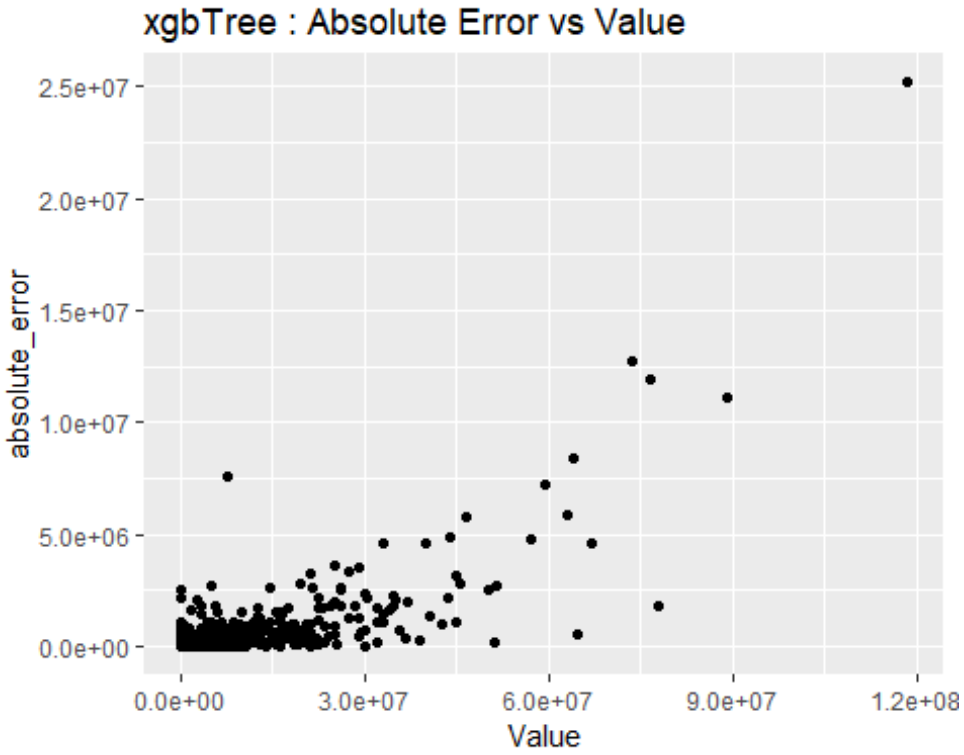
```
  #xgbTree model - best tunning parameters
    RMSE_xgbTree_Best

## [1] 985049.7

    WMAPE_xgbTree_Best

## [1] 0.1487534
```

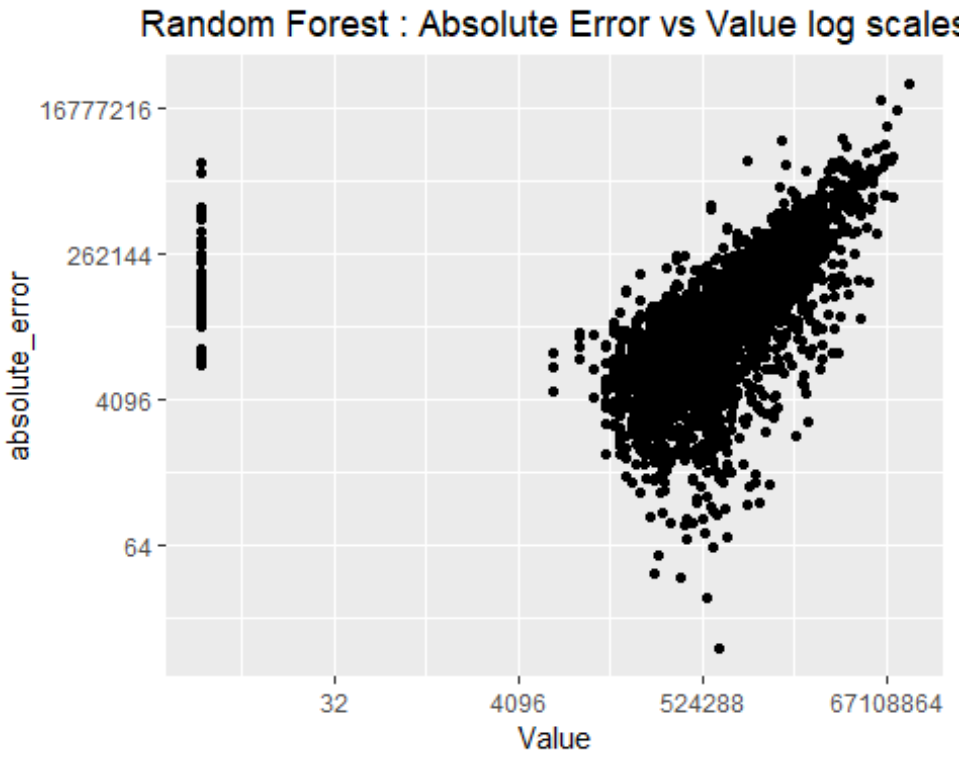
Here is the final table result:

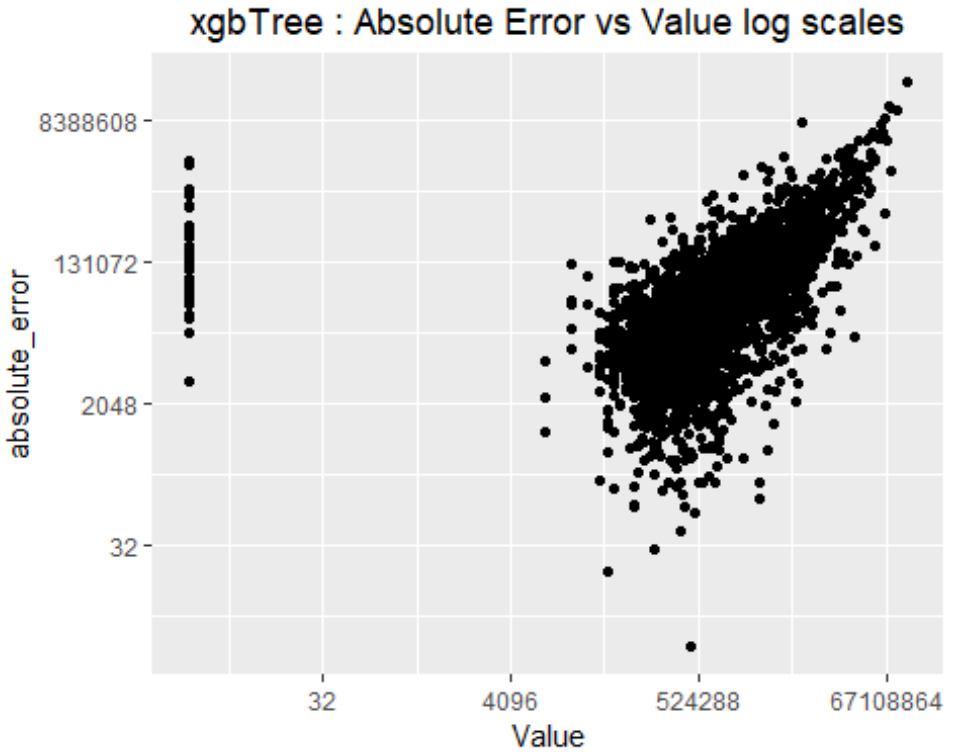
method	RMSE	WMAPE
Global Average	6329784.7	1.1322728
Lm model	1337314.6	0.2090762
glm model	1337097.0	0.2090690
knn model	1452063.4	0.2195284
Rpart model	1731746.3	0.2964264
RF model	871768.3	0.0701006
xgbTree model	900097.3	0.0805965
gamboost model	1130318.5	0.1467369
gamLoess model	985049.7	0.1487534
RF best tune	840821.6	0.0645254

At this stage, it is relevant to look at the error of both models in order to see if one is better in a certain value “Area”.


Random Forest : Absolute Error vs Value
absolute_error
3e+07
2e+07
1e+07
0e+00
0.0e+00
3.0e+07
6.0e+07
9.0e+07
1.2e+08
Value


xgbTree : Absolute Error vs Value
absolute_error
2.5e+07
2.0e+07
1.5e+07
1.0e+07
5.0e+06
0.0e+00
0.0e+00
3.0e+07
6.0e+07
9.0e+07
1.2e+08
Value





Looking at the above plots, we see that both models are doing quite well for low to average value, but they are both rather bad on extreme values, wether 0's or very large values. This

is actually very normal as there are less players on the extreme and therefore less error to minimize. Looking at the top_10 errors of both models.

Top 10 error of RandomForest:

```
##        Value absolute_error Value_Rank
## 1  118500000       32895367          1
## 2   57000000       20964903         11
## 3   89000000       15523917          2
## 4   67000000        9992067          6
## 5   21000000        6967907         73
## 6    4200000        6733567        517
## 7   64000000        5675350          8
## 8   22500000        5379873         64
## 9   51000000        5066133         13
## 10  40000000        4714067         23
```

Top 10 error of XgbTree:

```
##        Value absolute_error Value_Rank
## 1  118500000       25213072          1
## 2   73500000       12760216          5
## 3   76500000       11899664          4
## 4   89000000       11150320          2
## 5   64000000        8449388          8
## 6    7500000        7620542        271
## 7   59500000        7205796         10
## 8   63000000        5881548          9
## 9   46500000        5745432         15
## 10  44000000        4859624         19
```

Here again, it is difficult to draw any conclusion on extreme errors. In both cases, we see most of the players with highest values appearing in the top 10.

Conclusion

To conlude this project, it can be said that two of the models are showing fair results given the huge range of values of the output - the value of the players. As both models are doing better on one metric and none of them is really bad on one of the metric, we conclude that both are suitable for this database.

In order to improve to results further, several approaches are possible. Among other, more pre-processing could be done. Especially, some of the variables that have been excluded from the beginning could have been tested. (after being converted to dummy variables, for somme of them or after being wrangled) This is mostly a reason of time why this is not tried in this project.

In the end, it is very likely that deep learning would have improved the results further as it usually shows better results than machine learning. This would be a next learning step as this is not covered at all in the Lecture.