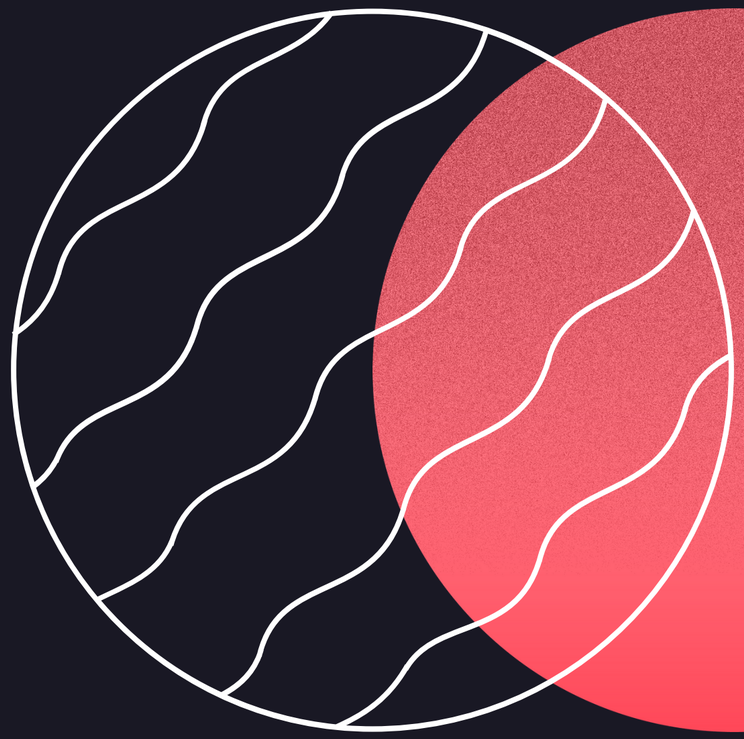


SkillCraft1 Dataset Analysis

Python for data analysis

Juliette Barthet

2020 - 2021

Contents

Introduction

About the game

Starcraft II: Wings of liberty is a multiplayer, strategy video game. It was released in 2010.

Online players are distributed into 7 leagues according to their level.

Introduction

About the dataset

Dataset Composition:

- 20 attributes
- 3 395 instances

Collection of Data:

- Telemetry from 3 340 players of 7 leagues contacted through social media and online gaming communities
- Replays of 35 professional players found on gaming websites

Problem Definition

The point of predicting leagues:

- By analyzing games, we will be able to assess gamers' level and allocate them to a concording league so that the enjoyment maximum: no boredom nor frustration
- Better train game's AI so that players can feel the same challenge even offline

DATA EXPLORATION-

FIRST GLANCE

5 first rows and data types.
As we can see, we only have numerical values, even the leagues (which are categories) are integers

	GameID	LeagueIndex	Age	HoursPerWeek	TotalHours	APM	SelectByHotkeys	AssignToHotkeys	UniqueHotkeys	MinimapAttacks	MinimapRightClicks
0	52	5	27.0	10.0	3000.0	143.7180	0.003515	0.000220	7	0.000110	0.000392
1	55	5	23.0	10.0	5000.0	129.2322	0.003304	0.000259	4	0.000294	0.000432
2	56	4	30.0	10.0	200.0	69.9612	0.001101	0.000336	4	0.000294	0.000461
3	57	3	19.0	20.0	400.0	107.6016	0.001034	0.000213	1	0.000053	0.000543
4	58	3	32.0	10.0	500.0	122.8908	0.001136	0.000327	2	0.000000	0.001329

NumberOfPACs	GapBetweenPACs	ActionLatency	ActionsInPAC	TotalMapExplored	WorkersMade	UniqueUnitsMade	ComplexUnitsMade	ComplexAbilitiesUsed
0.004849	32.6677	40.8673	4.7508	28	0.001397	6	0.0	0.000000
0.004307	32.9194	42.3454	4.8434	22	0.001194	5	0.0	0.000208
0.002926	44.6475	75.3548	4.0430	22	0.000745	6	0.0	0.000189
0.003783	29.2203	53.7352	4.9155	19	0.000426	7	0.0	0.000384
0.002368	22.6885	62.0813	9.3740	15	0.001174	4	0.0	0.000019

```
Data types:

GameID                 int64
LeagueIndex            int64
Age                  float64
HoursPerWeek         float64
TotalHours           float64
APM                  float64
SelectByHotkeys      float64
AssignToHotkeys      float64
UniqueHotkeys          int64
MinimapAttacks       float64
MinimapRightClicks   float64
NumberOfPACs         float64
GapBetweenPACs       float64
ActionLatency        float64
ActionsInPAC         float64
TotalMapExplored       int64
WorkersMade          float64
UniqueUnitsMade        int64
ComplexUnitsMade     float64
ComplexAbilitiesUsed float64
dtype: object
```

DATA EXPLORATION-

MISSING VALUES

Percentage of missing values:

```
GameID                  0.000000
LeagueIndex             0.000000
Age                     1.620029
HoursPerWeek            1.649485
TotalHours              1.678940
APM                     0.000000
SelectByHotkeys         0.000000
AssignToHotkeys         0.000000
UniqueHotkeys           0.000000
MinimapAttacks          0.000000
MinimapRightClicks      0.000000
NumberOfPACs            0.000000
GapBetweenPACs          0.000000
ActionLatency           0.000000
ActionsInPAC            0.000000
TotalMapExplored        0.000000
WorkersMade             0.000000
UniqueUnitsMade         0.000000
ComplexUnitsMade        0.000000
ComplexAbilitiesUsed    0.000000
dtype: float64
```

We have less than 2 percent of missing values in features non-related to the actions in the game.

```
df[df.isnull().any(axis=1)]
```

	GameID	LeagueIndex	Age	HoursPerWeek	TotalHours
358	1064	5	17.0	20.0	NaN
1841	5255	5	18.0	NaN	NaN
3340	10001	8	NaN	NaN	NaN
3341	10005	8	NaN	NaN	NaN
3342	10006	8	NaN	NaN	NaN
3343	10015	8	NaN	NaN	NaN
3344	10016	8	NaN	NaN	NaN
3345	10017	8	NaN	NaN	NaN
3346	10018	8	NaN	NaN	NaN
3347	10021	8	NaN	NaN	NaN
3348	10022	8	NaN	NaN	NaN
3349	10023	8	NaN	NaN	NaN
3350	10024	8	NaN	NaN	NaN

For non-professional players, the MVs are missing completely at random. However, after checking the columns of all the professional players, we never have their Age, HoursPerWeek and TotalHours. Indeed their data was gathered only thanks to replays so we do not have access to this information.

Dealing with missing values

For non-professional players, MVs were replaced by the mean of the column according to their leagues.

For professional players, no change was done since I could not find any information to replace these values.

For our predictions, we will keep in mind that we do not have these values. We may not need these columns.

DATA EXPLORATION-

DESCRIPTIVE STATISTICS

It seems like we have one (or more) outstanding value(s) for TotalHours. Indeed, the maximum is 1 million which corresponds to no more than 114 years. Knowing that Starcraft II was released ten years ago, this is impossible. Moreover, the maximum value reported for HoursPerWeek is 168 which corresponds to 7 days so this person would be playing 24/7 which again, is impossible. Therefore, we will tweak these values so that they can be legit. We will consider that these players indeed played all day long every day but with some sleep: 4 hours. This will result in 20 hours a day so 140 hours per week and 7300 hours per year. In order to get the total amount of hours, we will also consider that this was done since the game has been released so 10 years ago which results in 65 700 hours total. Any amount higher than these will be replaced by these values. Except for these values, nothing seems shocking. The standard deviation is high for APM, this can be explained by the fact that it surely requires high skills to do many actions per minute so it is not very common and most standard players can not achieve this.

```
df.describe().transpose()
```

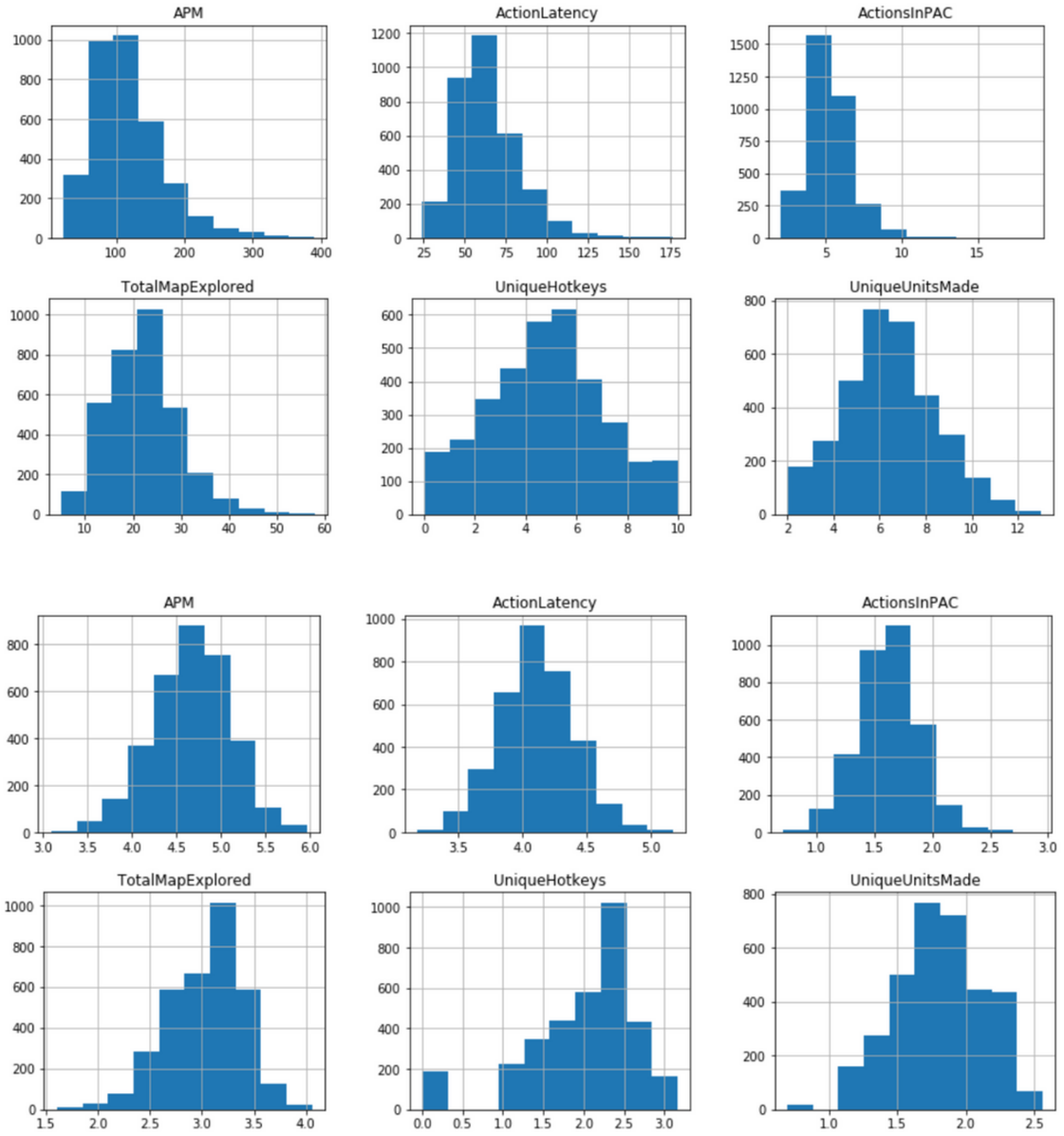
	count	mean	std	min	25%	50%	75%	max
GameID	3395.0	4805.012371	2719.944851	52.000000	2464.500000	4874.000000	7108.500000	10095.000000
LeagueIndex	3395.0	4.184094	1.517327	1.000000	3.000000	4.000000	5.000000	8.000000
Age	3340.0	21.647904	4.206341	16.000000	19.000000	21.000000	24.000000	44.000000
HoursPerWeek	3340.0	15.910833	11.961121	0.000000	8.000000	12.000000	20.000000	168.000000
TotalHours	3340.0	961.058978	17312.966097	3.000000	300.000000	500.000000	800.000000	1000000.000000
APM	3395.0	117.046947	51.945291	22.059600	79.900200	108.010200	142.790400	389.831400
SelectByHotkeys	3395.0	0.004299	0.005284	0.000000	0.001258	0.002500	0.005133	0.043088
AssignToHotkeys	3395.0	0.000374	0.000225	0.000000	0.000204	0.000353	0.000499	0.001752
UniqueHotkeys	3395.0	4.364654	2.360333	0.000000	3.000000	4.000000	6.000000	10.000000
MinimapAttacks	3395.0	0.000098	0.000166	0.000000	0.000000	0.000040	0.000119	0.003019
MinimapRightClicks	3395.0	0.000387	0.000377	0.000000	0.000140	0.000281	0.000514	0.004041
NumberOfPACs	3395.0	0.003463	0.000992	0.000679	0.002754	0.003395	0.004027	0.007971
GapBetweenPACs	3395.0	40.361562	17.153570	6.666700	28.957750	36.723500	48.290500	237.142900
ActionLatency	3395.0	63.739403	19.238869	24.093600	50.446600	60.931800	73.681300	176.372100
ActionsInPAC	3395.0	5.272988	1.494835	2.038900	4.272850	5.095500	6.033600	18.558100
TotalMapExplored	3395.0	22.131664	7.431719	5.000000	17.000000	22.000000	27.000000	58.000000
WorkersMade	3395.0	0.001032	0.000519	0.000077	0.000683	0.000905	0.001259	0.005149
UniqueUnitsMade	3395.0	6.534021	1.857697	2.000000	5.000000	6.000000	8.000000	13.000000
ComplexUnitsMade	3395.0	0.000059	0.000111	0.000000	0.000000	0.000000	0.000086	0.000902
ComplexAbilitiesUsed	3395.0	0.000142	0.000265	0.000000	0.000000	0.000020	0.000181	0.003084

DATA EXPLORATION-
VIZUALIZATION

Looking at the distribution of each variable, we can say that it is mostly gaussian and positively skewed. We do have some outliers that reduce the visibility but we could imagine that the value distributions match the league index distribution. Therefore, as the data is right-skewed our future model could have trouble predicting accurate leagues when using high values.

I checked mathematically and the data was indeed positively skewed. In order to avoid misclassification when using algorithms that assume our data is gaussian, I applied a log transform to my positive data and a square root transform to data close to 0.
It definitely reduced skewness and its distribution was closer to gaussian.

On the right you can see the data distribution before applying transforms (top figure) and after (bottom figure).

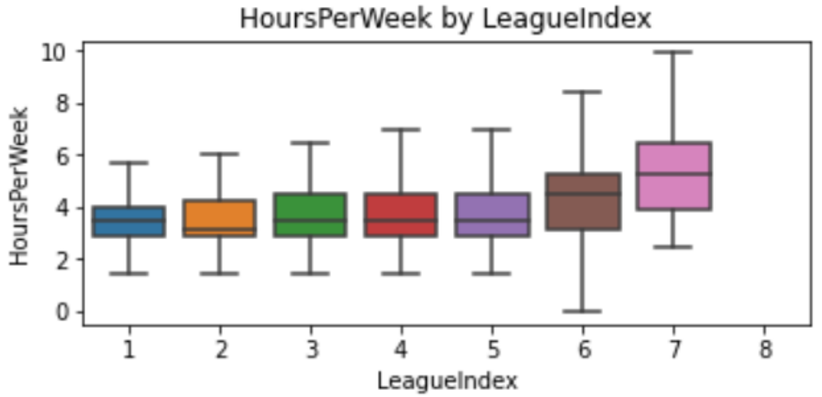


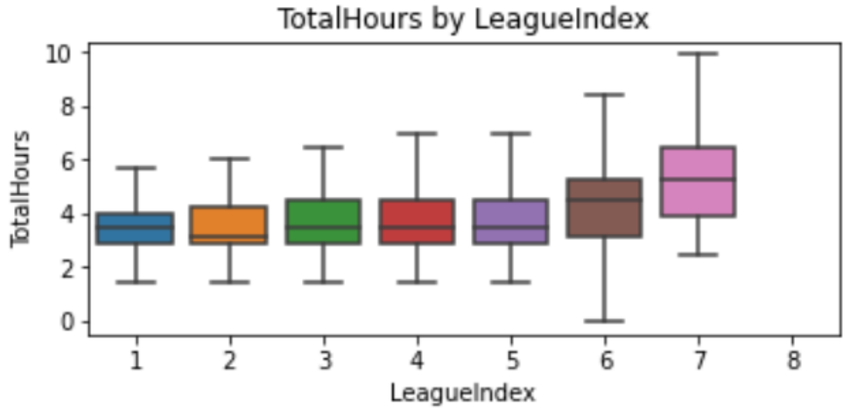
DATA EXPLORATION-

VIZUALIZATION

In this part, we will be looking at the distribution of each feature according to the different leagues by using boxplots.

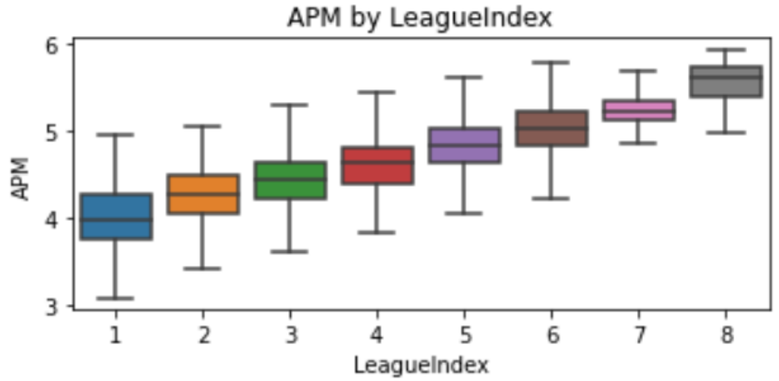
First, looking at the amount of time spent playing, of course, the more you play the more likely you are to get better. However, the differences between leagues are not so relevant. The amount of time played does not seem to have such an impact on a player's level. Also, we do not have any information about this for the 8th league.

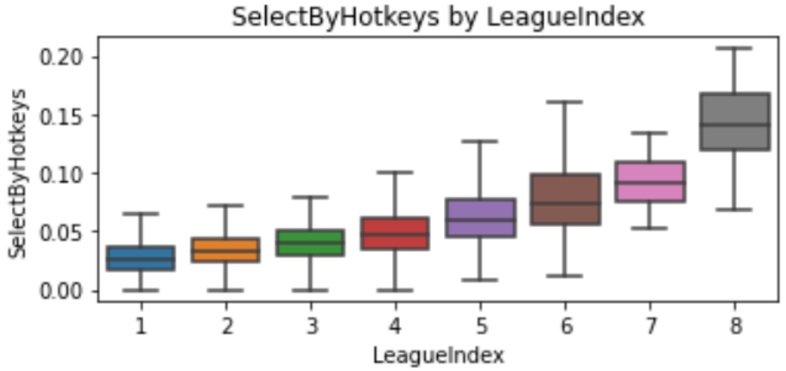


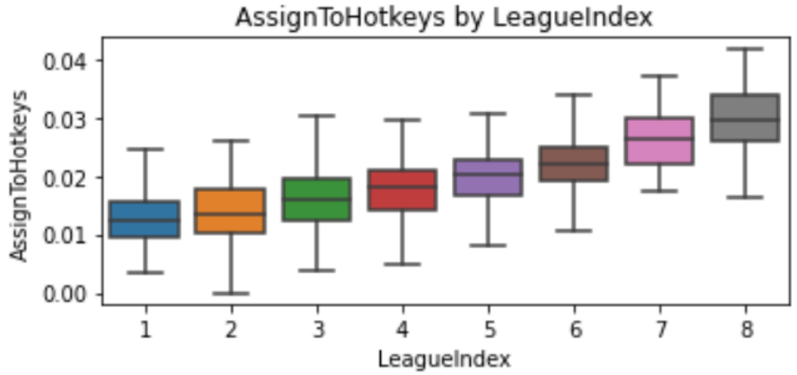


DATA EXPLORATION-
VIZUALIZATION

the higher the league index, the higher the APM, SelectByHotKeys, AssignToHotKeys.
Indeed, the number of Actions Per Minute is obviously higher in an advanced league since it requires habits and skill to master them and be able to do a series of actions. These APM can also be linked to the number of Hot Keys allocated and used since they are basically shortcuts that enable players to play faster and more efficiently.

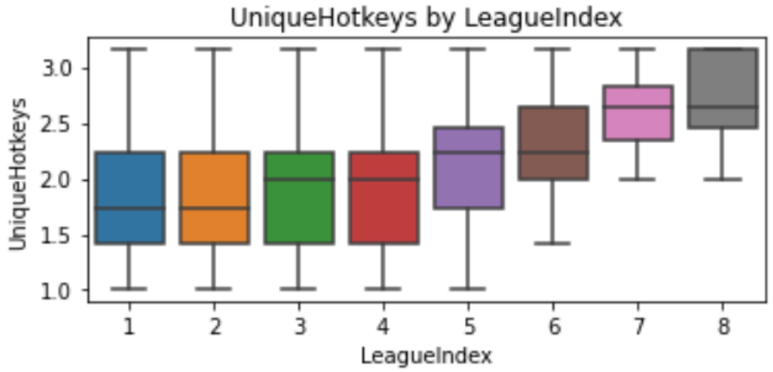


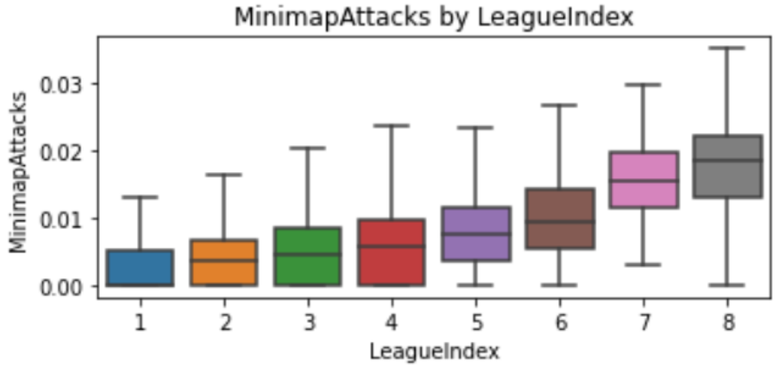


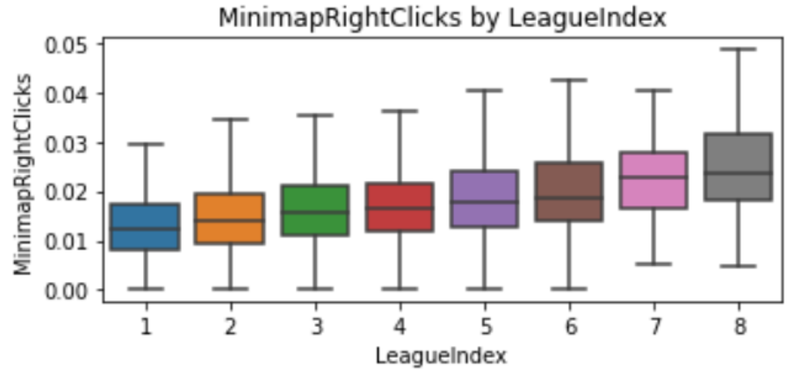


DATA EXPLORATION-
VIZUALIZATION

The higher the league index, the higher the UniqueHotKeys, MinimapAttacks, MinimapRightClicks.
Indeed, A high number of UniqueHotKeys can mean that a player diversifies his actions, he does not only use a single attack for example.
Considering the Minimap, it shows an overview of the game but can also be used to do actions like attacks or retreat. The more a player uses it, the less it has to move and lose time doing so. Also, right clicks on the minimap enable complex actions and trigger automatically casted abilities. All of these variables show how much a player is trained, prepared, and knows the game.

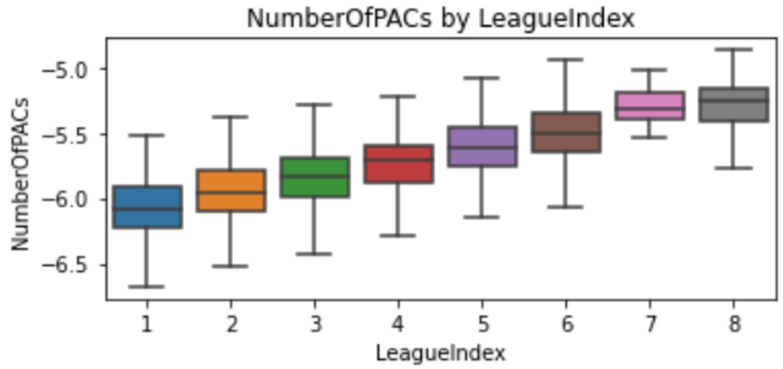


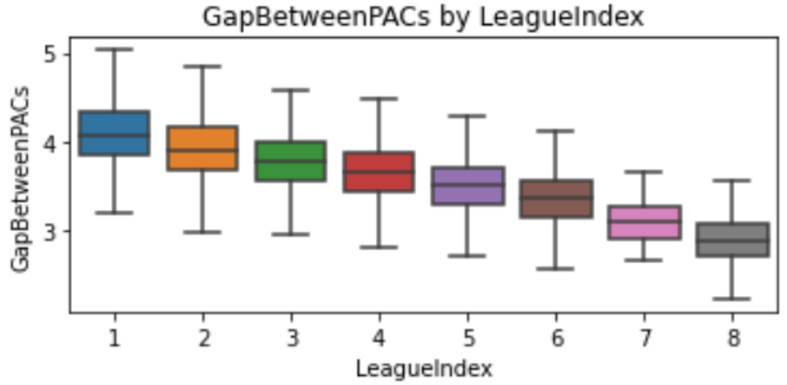


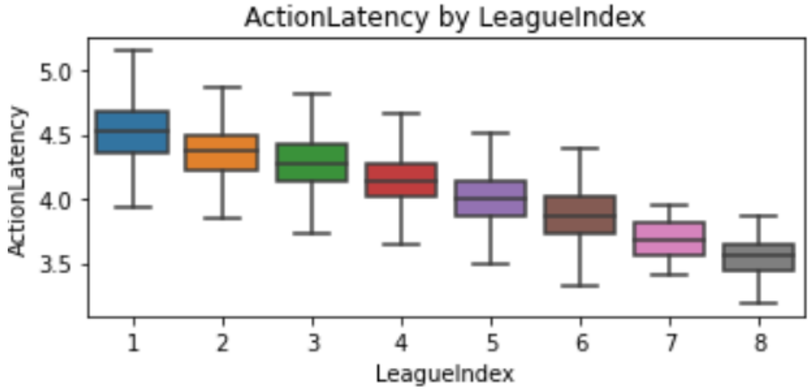


DATA EXPLORATION-
VIZUALIZATION

The higher the league index, the higher the NumberofPACs and the lower the GapBetweenPACs and ActionLatency.
The number of Perception-Action Cycles refers to the number of actions taken by the player after perceiving something, so the higher it is, the more aware he is. The lower the action latency and gaps between PCAs, the more responsive the player which is crucial in such a game.

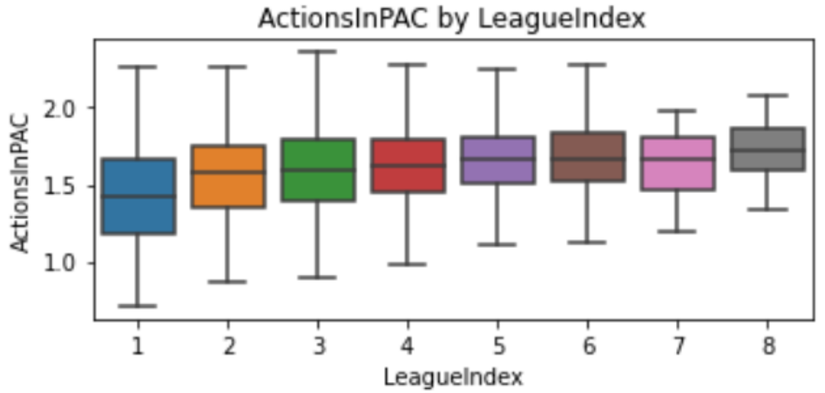


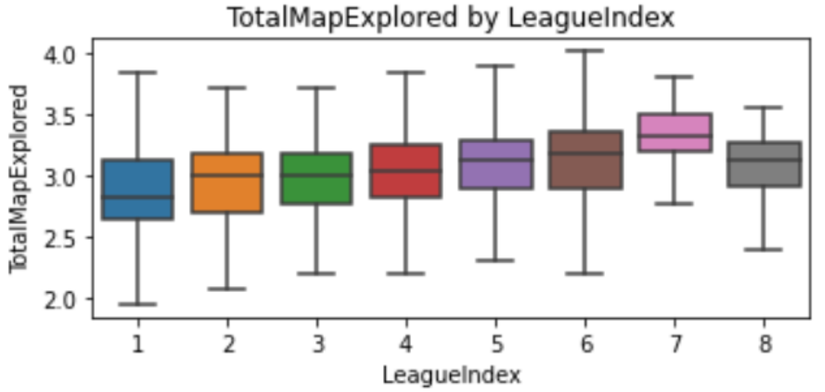




DATA EXPLORATION-
VIZUALIZATION

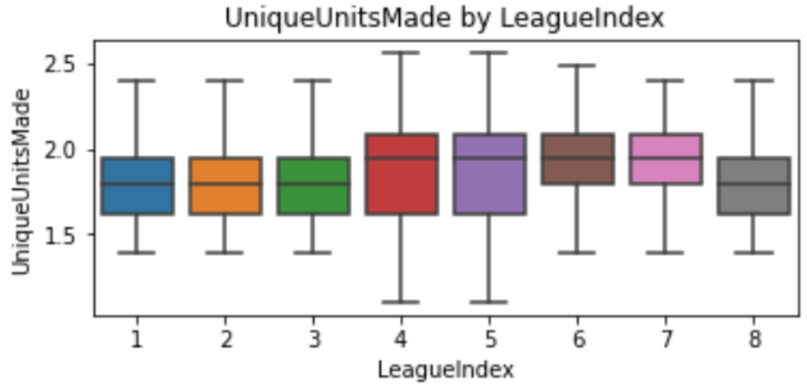
These graphs do not show a link between ActionsInPAC, TotalMapExplored, WorkersMade, and the league index.

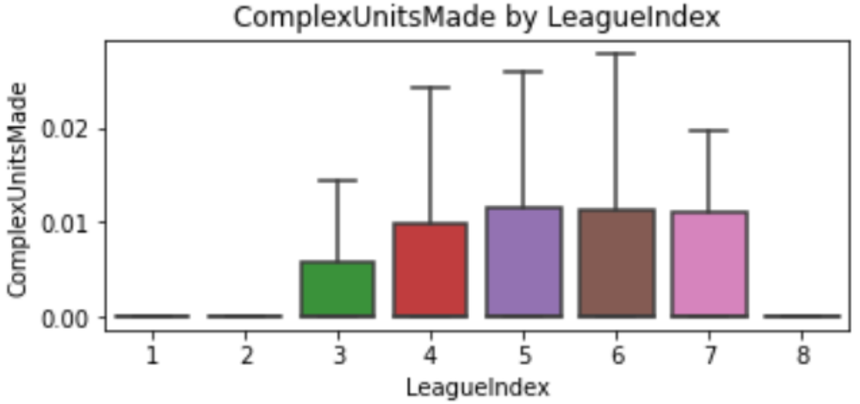


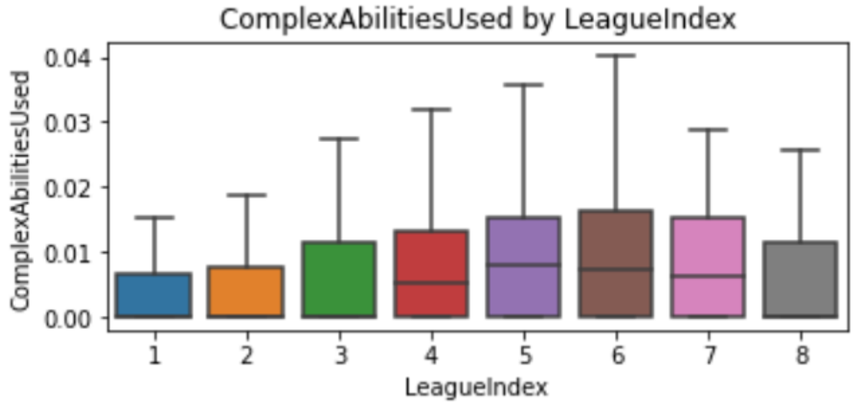


DATA EXPLORATION-
VIZUALIZATION

These graphs do not show a link between UniqueUnitsMade, ComplexUnitsMade, ComplexAbilitiesUsed, and the league index.







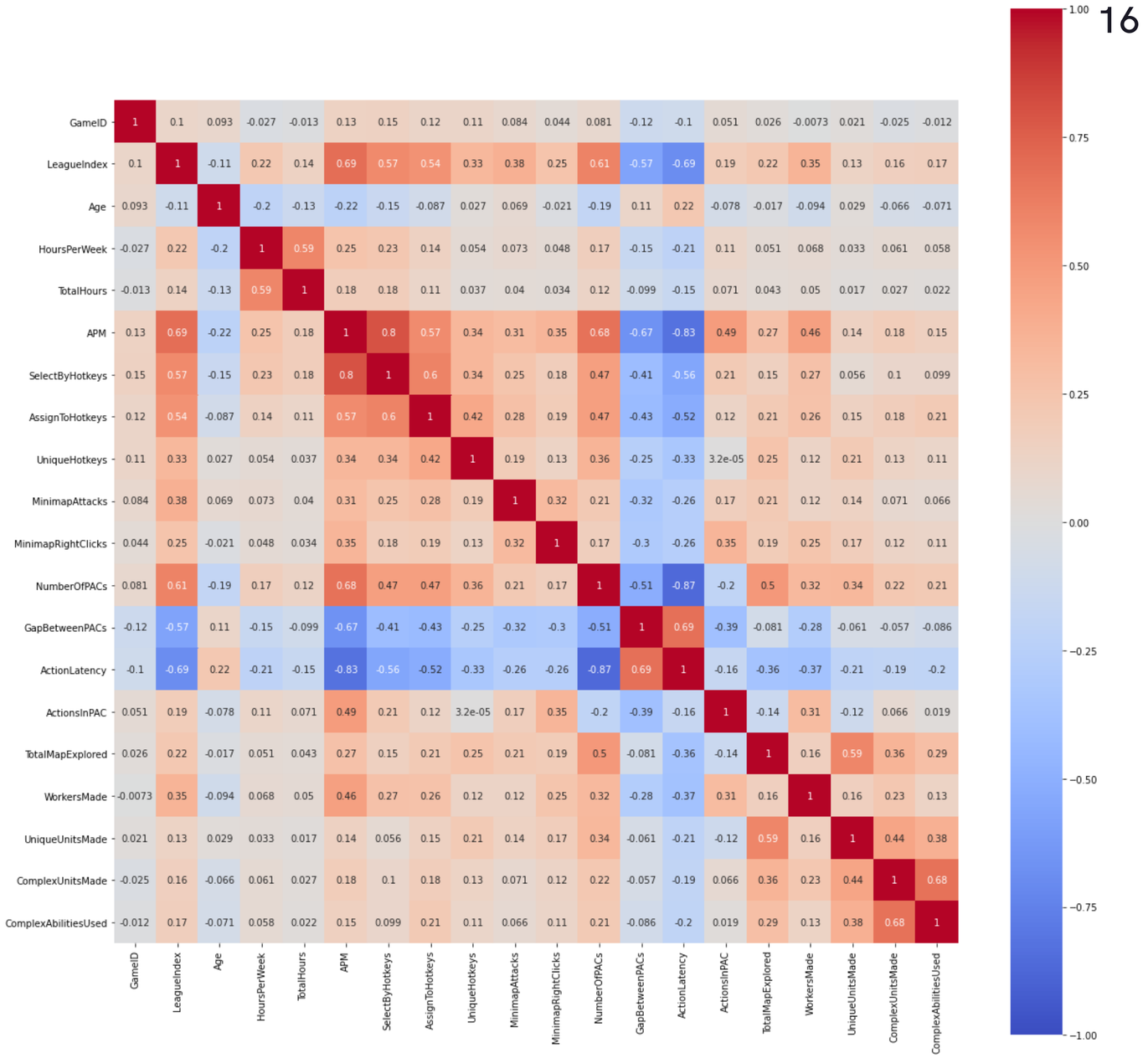
DATA EXPLORATION-

VIZUALIZATION

This correlation heatmap highlights the correlation between the League index and the variables identified above. But we need to keep in mind that these variables are also strongly correlated together.

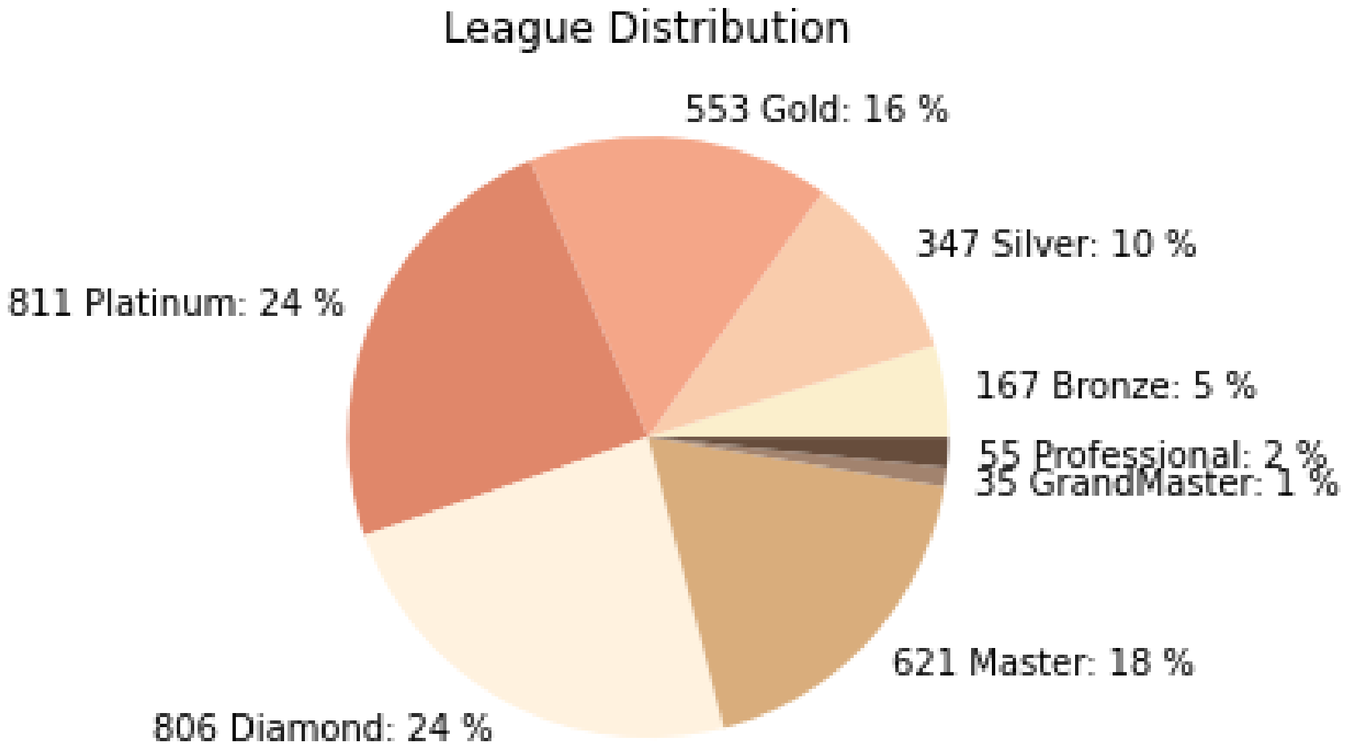
Also, it is important to bear in mind that our target variable is categorical and is here represented by a numerical value between 1 to 8 which can not translate the differences between leagues.

Indeed, looking at the leagues populations online, we see that although the distance is not so important between Silver to Diamond leagues, this gap is much more important with Master and GrandMaster leagues. The percentage of players in these last two leagues is divided by 5 when we compare Diamond to Master and by 10 between Diamond and GrandMaster! Also, we do not have the exact numbers of professional players but one can only imagine that it is even more selective. Thus, the chosen scale that represents our target is not so accurate, it could be better to apply a logarithmic scale.



DATA EXPLORATION-
VIZUALIZATION

In fact, looking at the distribution of our data, we can see that the only league well represented (corresponding to real-life data) is Diamond. The rest of them is either underrepresented (Bronze, Silver, Gold) or overrepresented (Platinum, Master, GrandMaster and Professional) which can lead to errors in predictions.
This is due to the way data was collected: through social media and gaming communities where players in higher leagues are usually more involved.



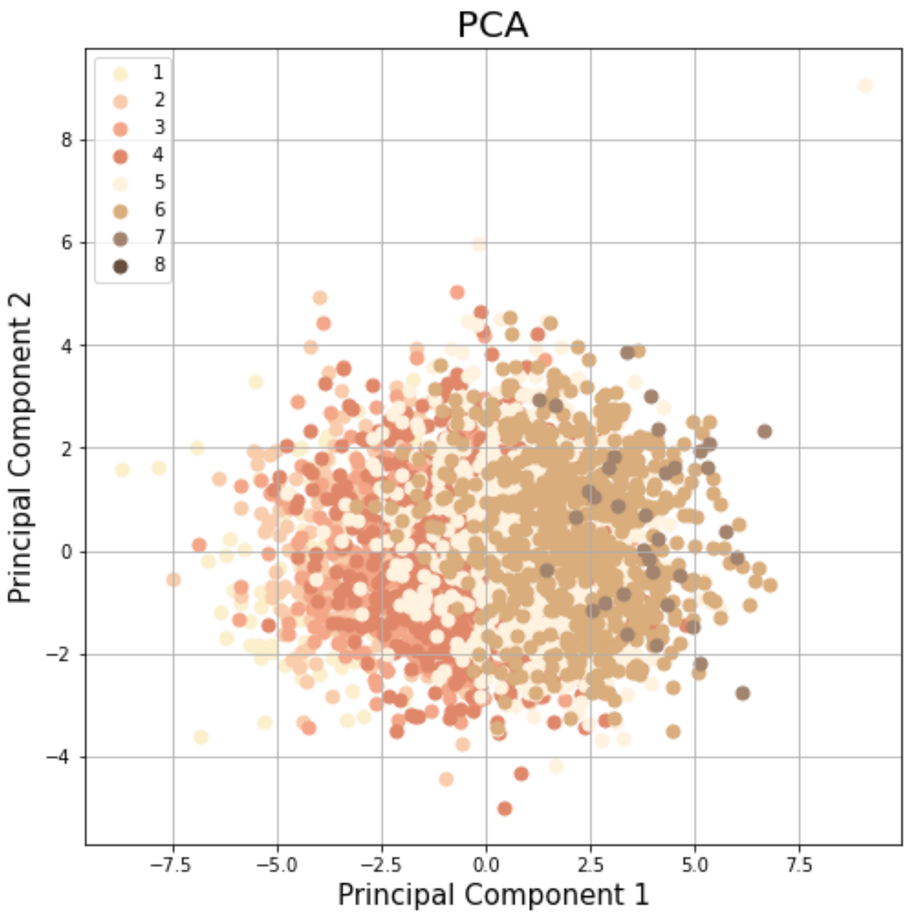
FEATURE SELECTION-

PCA / ALL FEATURES

As we can see from the graph, even when reducing dimensions, the data is overlapping, not well separated. But we can see that there seems to be an order, the leagues are ordered from 8 to 1 from the left to the right.

The explained variance ratio shows that the two first principal components (PCs) explain around 41 % of the variation of our model.

The main features for each component are related to actions taken during the game. However, PC 3 is really related to the amount of time spent playing.
But, what if we only had a replay of the game and did not have access to these last features?



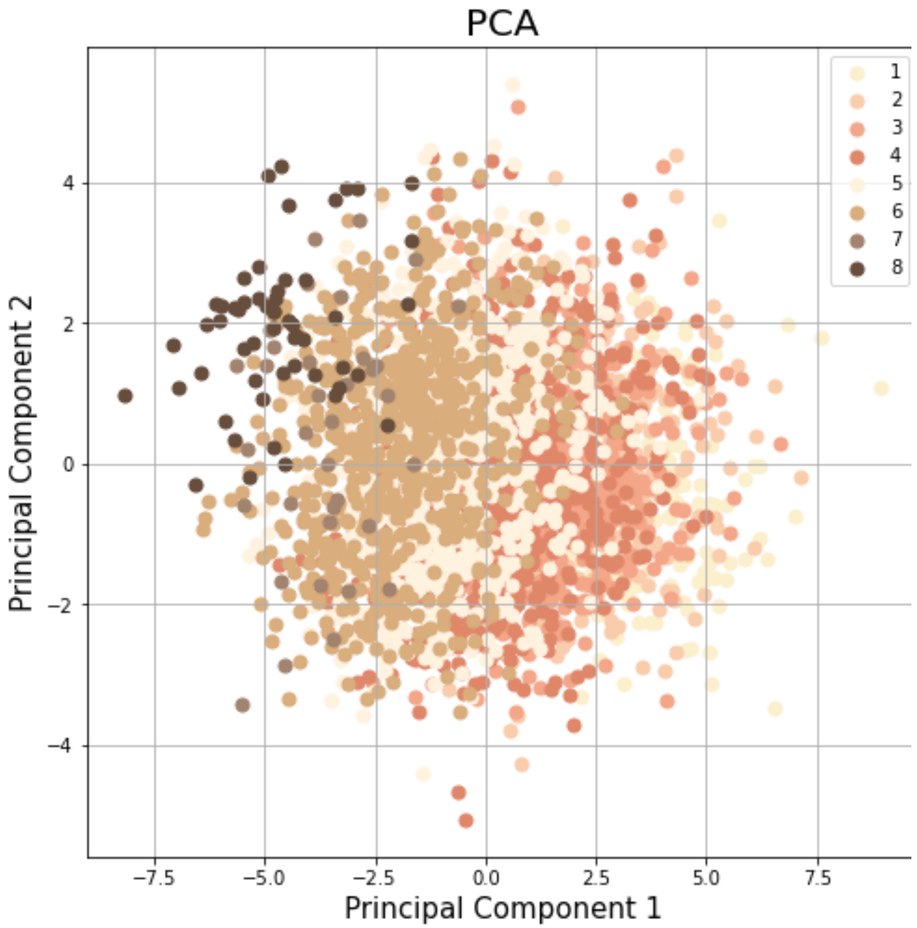
	Feature	Variance explained
PC0	APM	0.293736
PC1	UniqueUnitsMade	0.122485
PC2	TotalHours	0.088139
PC3	ActionsInPAC	0.081699
PC4	MinimapAttacks	0.064928
PC5	AssignToHotkeys	0.056070
PC6	WorkersMade	0.044904
PC7	Age	0.042592
PC8	UniqueHotkeys	0.037915
PC9	WorkersMade	0.033617
PC10	MinimapRightClicks	0.031800
PC11	AssignToHotkeys	0.025545
PC12	HoursPerWeek	0.022358
PC13	TotalMapExplored	0.020942
PC14	ComplexUnitsMade	0.016434
PC15	GapBetweenPACs	0.012692
PC16	ActionLatency	0.003839
PC17	APM	0.000306

FEATURE SELECTION-

PCA / WITHOUT TIME AND AGE

The order of the variables is slightly different but the data is still overlapping.

As we can see now, our two first PCs explain around 48 % of the variance of our data which is better than with all our variables.

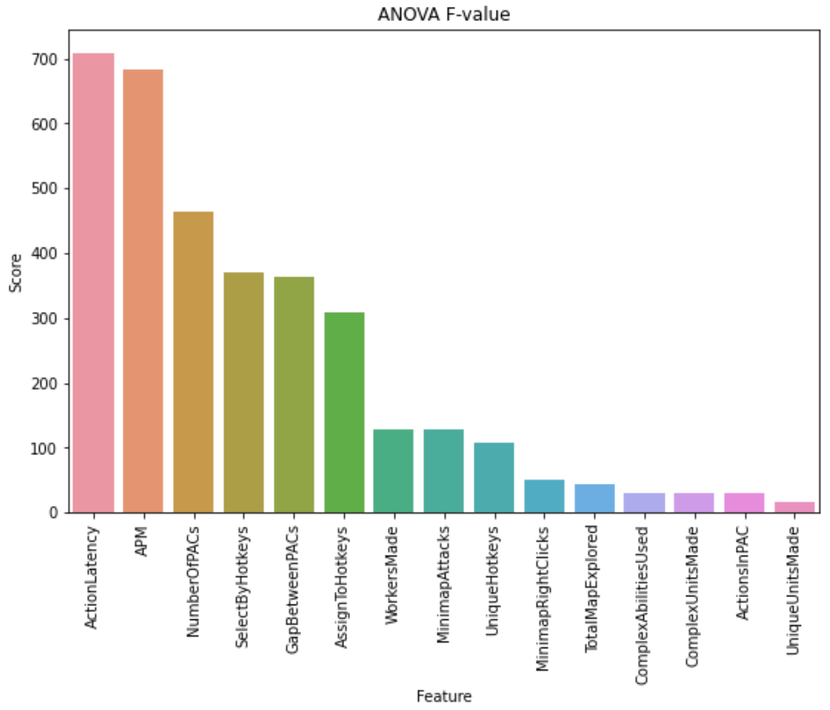


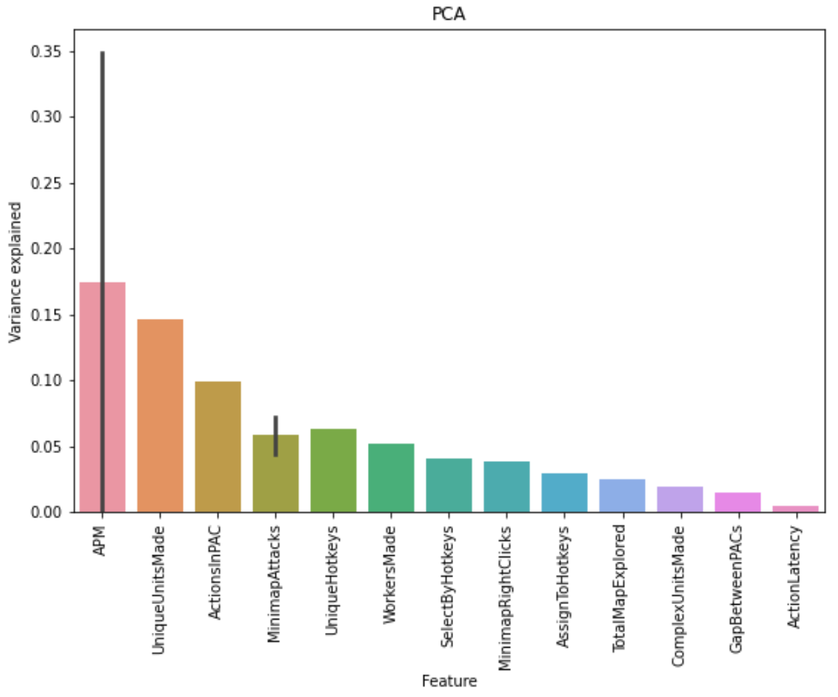
	Feature	Variance explained
PC0	APM	0.348971
PC1	UniqueUnitsMade	0.145926
PC2	ActionsInPAC	0.099642
PC3	MinimapAttacks	0.072598
PC4	UniqueHotkeys	0.063343
PC5	WorkersMade	0.052122
PC6	MinimapAttacks	0.044131
PC7	SelectByHotkeys	0.040228
PC8	MinimapRightClicks	0.038421
PC9	AssignToHotkeys	0.030014
PC10	TotalMapExplored	0.024966
PC11	ComplexUnitsMade	0.019979
PC12	GapBetweenPACs	0.014863
PC13	ActionLatency	0.004441
PC14	APM	0.000356

FEATURE SELECTION-

ANOVA VS. PCA

According to these plots, we can see 6 features standing out: ActionLatency, APM, GapBetweenPACs, NumberOfPACs, SelectByHotKeys and AssignToHotKeys.





DATA PREPARATION-

DROPPING COLUMNS & SAVING TARGET

As mentioned above, we only keep the data that we can gather from a game replay and get rid of the other ones.

```
to_drop = ['Age', 'GameID', 'HoursPerWeek', 'TotalHours']
df = df.drop(columns = to_drop)
leagues = df.LeagueIndex
df_data = df.drop(columns = 'LeagueIndex')
```

DATA PREPARATION-
BALANCING CLASSES

The target is imbalanced which can result in issues when trying to predict individuals of an under-populated category. Indeed, our model would not have enough data to learn how to predict if a player is a GrandMaster, a professional, or in the Bronze league. To avoid this issue, we will merge the classes that are under-represented with their closest league in terms of level.
To balance our classes, we merge Bronze with Silver (1 and 2) and Master with GrandMaster and Professional (6, 7 and 8).

```
Population in each League:

4    811
5    806
6    621
3    553
2    347
1    167
8     55
7     35

Population in each League after merging:

4    811
5    806
6    711
3    553
1    514
```

DATA PREPARATION-
STANDARDIZATION

Standardizing our data is important since our columns have different scales and units. Because of that, one feature could contribute more to our model than another one, creating a bias. The idea of standardization is to end up with a mean of 0 and standard deviation of 1.

	APM	SelectByHotkeys	AssignToHotkeys	UniqueHotkeys	MinimapAttacks	MinimapRightClicks
count	3.395000e+03	3.395000e+03	3.395000e+03	3.395000e+03	3.395000e+03	3.395000e+03
mean	-5.190415e-16	-4.185819e-17	-3.348655e-17	-2.678924e-16	-8.790219e-17	-1.255746e-16
std	1.000147e+00	1.000147e+00	1.000147e+00	1.000147e+00	1.000147e+00	1.000147e+00
min	-3.604682e+00	-1.778249e+00	-3.114451e+00	-2.807982e+00	-1.100218e+00	-2.055734e+00
25%	-6.596021e-01	-6.746729e-01	-6.959089e-01	-3.367777e-01	-1.100218e+00	-6.811108e-01
50%	3.019340e-02	-2.225673e-01	6.330022e-02	4.551889e-02	-1.526733e-01	-1.071583e-01
75%	6.689705e-01	4.510115e-01	6.654167e-01	6.868286e-01	5.345563e-01	5.776118e-01
max	2.967169e+00	4.680869e+00	3.969721e+00	1.703799e+00	7.139421e+00	5.327049e+00

DATA PREPARATION-
SPLITTING DATA

In order to train our model and test it, we split our data. We do so to avoid overfitting. I chose to use 70 % of my data to train the model and 30 % to test it.

```
Training dataset shape: (2376, 15)
Test dataset shape: (1019, 15)
Training target shape: (2376,)
Test target shape: (1019,)
```

CHOOSING A MODEL-

LOGISTIC REGRESSION

We will try different algorithm of classification, first with all the features but also, only with the 6 most important features identified earlier.

Here are the coefficients for our logistic regression.

Once we had a model, we tried different random splits for our data so that we could have the best split possible. Indeed, when splitting data we can set a random_state variable to make a split reproducible so we compared the accuracy of our logistic regression with different splits and kept the most accurate one: obtained with random_state = 4035.

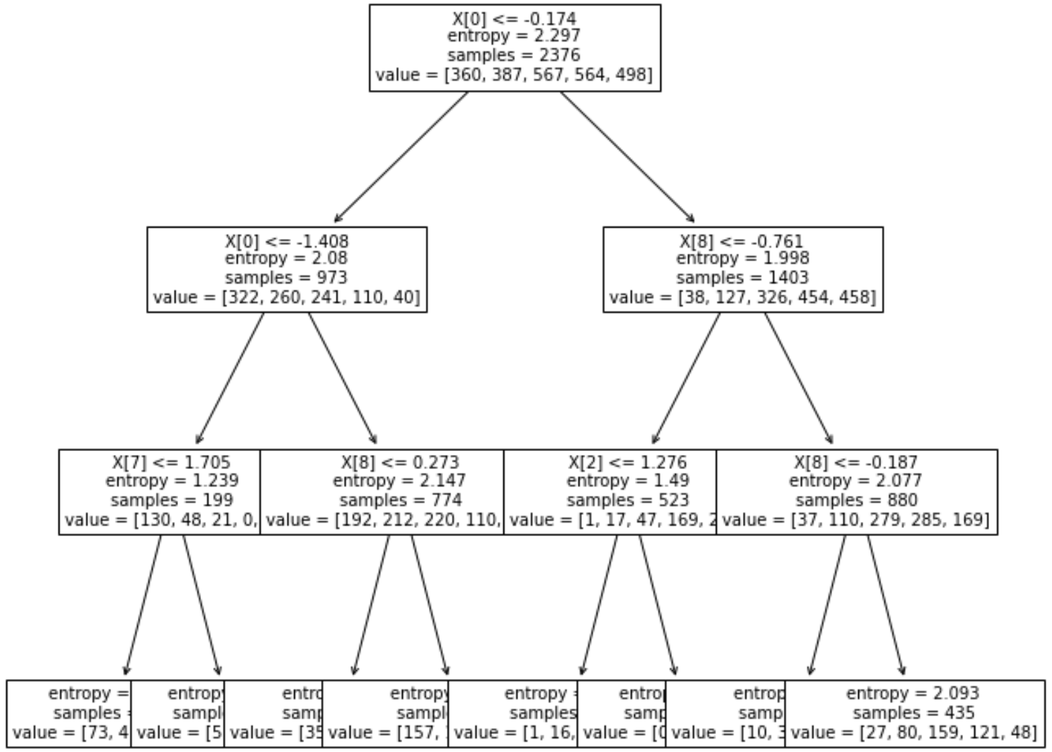
	0	1	2	3	4
intercept	-1.053812	0.010240	0.752425	0.597684	-0.306537
APM	-0.515376	0.107108	0.162207	0.353708	-0.107647
SelectByHotkeys	-0.296956	-0.310158	-0.103655	0.082674	0.628095
AssignToHotkeys	-0.499211	-0.109089	0.003367	0.152273	0.452661
NumberOfPACs	-0.213366	-0.015592	-0.031529	0.081721	0.178766
GapBetweenPACs	0.372748	0.231464	0.056312	-0.200320	-0.460204
ActionLatency	0.753135	0.708990	0.100305	-0.426722	-1.135708

CHOOSING A MODEL-

DECISION TREE

The decision tree algorithm breaks the datapoints into decision nodes resulting in a tree structure. The goal is to classify data points by answering if conditions.

To create a decision tree, we can either use the entropy or the GINI index, on the right you can see the decision tree obtained with the entropy.



CHOOSING A MODEL-

RANDOM FOREST

The random forest algorithm is basically a combination of multiple decision trees creating with different parameters and samples of data to find the most accurate model so it should perform better than the decision tree algorithm.

Random forests also have a reproducibility parameter so we tried different ones and found that 34 with the best one in terms of accuracy.

CHOOSING A MODEL-

COMPARING RESULTS

As we can see here, our very first model is the one that performs the best with almost 49 % of accuracy. The Random Forest is also close to the logistic regression.
What we can also say is that even with only 6 features our models are performant.
Next, we will use grid searching to find the best parameters for our most promising models: Logistic Regression and Random Forest.

	Accuracy
Logreg	0.487733
Logreg6	0.480864
Decision tree Gini	0.442591
Decision tree Entropy	0.424926
Decision tree Gini6	0.442591
Decision tree Entropy6	0.424926
Random Forest	0.477920
Random Forest6	0.443572

CHOOSING A MODEL-

GRIDSEARCHING / RANDOM FOREST

Here are the different parameters that we tried.

It turned out that the best ones were:
n_estimators = 100
max_features = 'auto'
max_depth = 4
criterion = 'gini'

```
param_grid = {
    'n_estimators': [200, 500, 100],
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth' : [4,5,6,7,8],
    'criterion' :['gini', 'entropy']
}
```

CHOOSING A MODEL-

GRIDSEARCHING / LOGISTIC REGRESSION

Here are the different parameters that we tried.

It turned out that the best ones were:
penalty = 'l2'
C = 0.1

With this last model, we can see that our MSE and our R^2 are close which means that our model did not overfit.

```
param_grid_logreg = {
    'penalty': ['l1','l2'],
    'C': [0.001,0.01,0.1,1,10,100,1000]
}
```

```
MSE Train: 1.68
R2 Train: 0.35
MSE Test: 1.51
R2 Test: 0.42
```

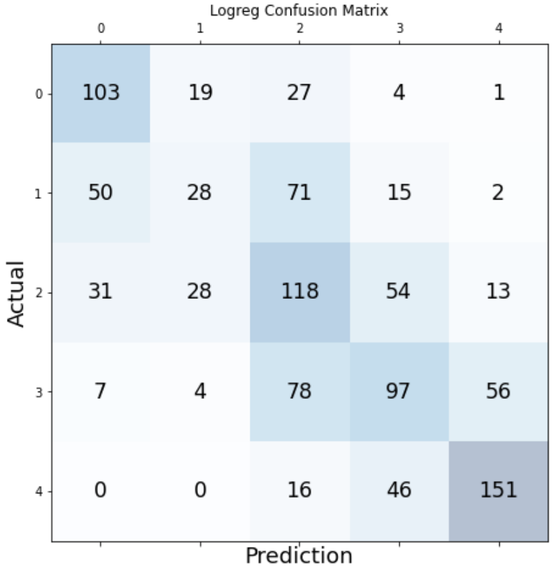
CHOOSING A MODEL-
COMPARISON

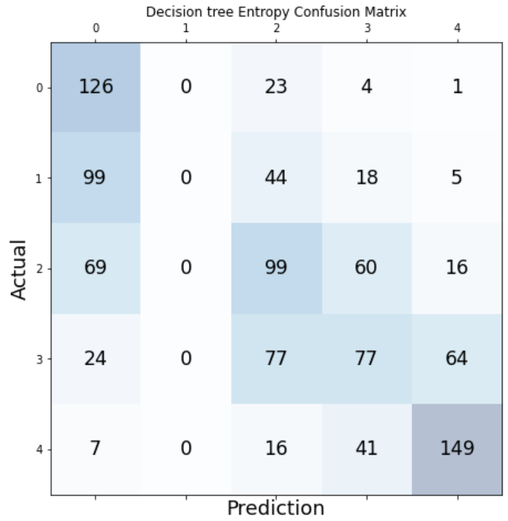
When we compare our accuracy, we can see that gridsearching definitely improved our models and that the most accurate one is the logistic regression with 48.9 % of accuracy.

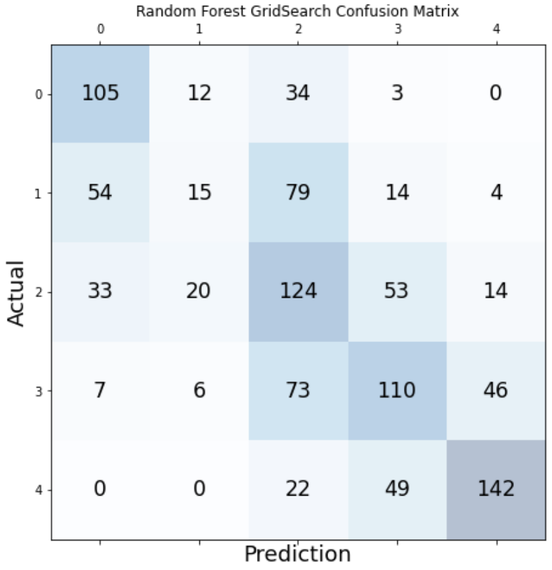
	Accuracy
Logreg	0.487733
Logreg6	0.480864
Decision tree Gini	0.442591
Decision tree Entropy	0.424926
Decision tree Gini6	0.442591
Decision tree Entropy6	0.424926
Random Forest	0.477920
Random Forest6	0.443572
Random Forest GridSearch	0.486752
Logistic Regression GridSearch	0.489696

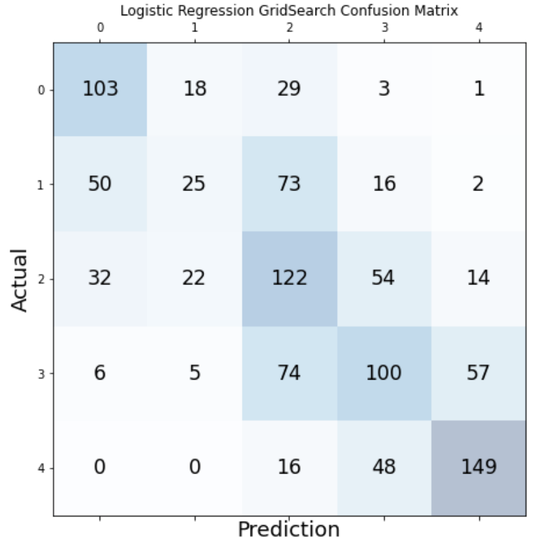
CHOOSING A MODEL-
COMPARISON

The confusion matrices are useful to see where our model does misclassification and to calculate other indicators such as the sensitivity of the model.



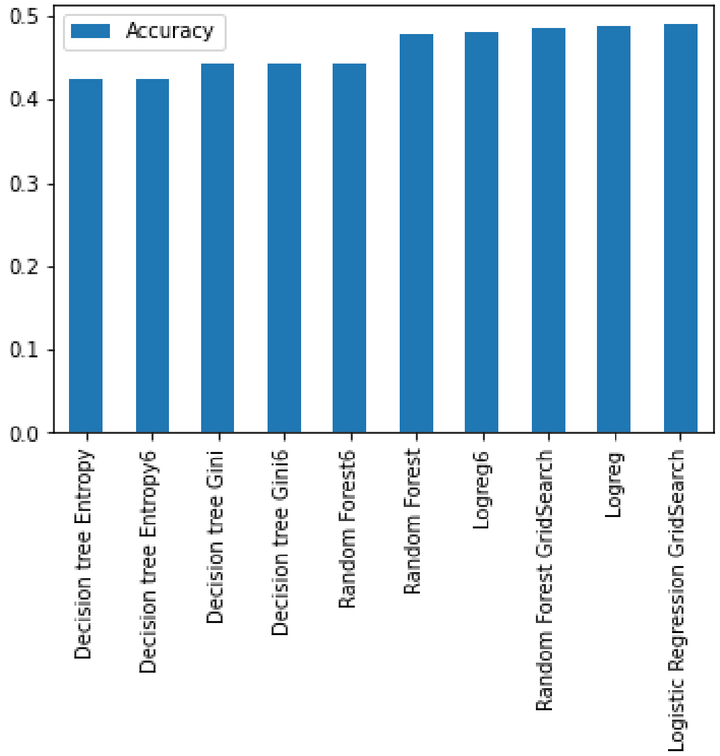






CHOOSING A MODEL-
COMPARISON

Here is a plot of all of our models ordered by accuracy (ascending order).



Conclusion

Our study of the SkillCraft1 dataset led to the implementation of a logistic regression classifier. We tuned its parameters thanks to grid searching and got an accuracy of 49 % when a simple decision tree only had a 42 % accuracy.

Also, by using principal components analysis and ANOVA testing, we were able to identify the most influential features when it comes to predicting a league. These results are useful to modify the game's AI to better simulate a human player. They also provide information to gamers to evaluate their level, identify their strength and weaknesses so that they can improve.