Classification
Ranking
Starcraft 2

JEAN-VICTOR
KRAUTH/BENJAMIN JOHNSON

Visualisation

	GameID	LeagueIndex	Age	HoursPerWeek	TotalHours	APM	SelectByHotkeys	AssignToHotkeys	UniqueHotkeys	MinimapAttacks	MinimapRightClic
0	52	5	27	10	3000	143.7180	0.003515	0.000220	7	0.000110	0.0003
1	55	5	23	10	5000	129.2322	0.003304	0.000259	4	0.000294	0.0004
2	56	4	30	10	200	69.9612	0.001101	0.000336	4	0.000294	0.0004
3	57	3	19	20	400	107.6016	0.001034	0.000213	1	0.000053	0.0005
4	58	3	32	10	500	122.8908	0.001136	0.000327	2	0.000000	0.0013
...	...	...	...	...	...	...	...	...	...	...	
3390	10089	8	?	?	?	259.6296	0.020425	0.000743	9	0.000621	0.0001
3391	10090	8	?	?	?	314.6700	0.028043	0.001157	10	0.000246	0.0010
3392	10092	8	?	?	?	299.4282	0.028341	0.000860	7	0.000338	0.0001
3393	10094	8	?	?	?	375.8664	0.036436	0.000594	5	0.000204	0.0007
3394	10095	8	?	?	?	348.3576	0.029855	0.000811	4	0.000224	0.0013

We may see that some values are missing, where and how many ?

Visualization

	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
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

We also want to see how the differents features are distributed to have a better understanding of the features.

Visualisation : missing values

```
: data.info() # des colonnes comme age ou nb d'heures qui devrai etre int sont des object, bizarre missing value ?

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3395 entries, 0 to 3394
Data columns (total 20 columns):
GameID                  3395 non-null int64
LeagueIndex             3395 non-null int64
Age                     3395 non-null object
HoursPerWeek            3395 non-null object
TotalHours              3395 non-null object
APM                     3395 non-null float64
SelectByHotkeys         3395 non-null float64
AssignToHotkeys         3395 non-null float64
UniqueHotkeys           3395 non-null int64
MinimapAttacks          3395 non-null float64
MinimapRightClicks      3395 non-null float64
NumberOfPACs            3395 non-null float64
GapBetweenPACs          3395 non-null float64
ActionLatency           3395 non-null float64
ActionsInPAC            3395 non-null float64
TotalMapExplored        3395 non-null int64
WorkersMade             3395 non-null float64
UniqueUnitsMade         3395 non-null int64
ComplexUnitsMade        3395 non-null float64
ComplexAbilitiesUsed    3395 non-null float64
dtypes: float64(12), int64(5), object(3)
memory usage: 530.6+ KB
```

There are int features, float features but also object features due to missing values. Thus, we assume that missing values are only in Age,HoursPerWeek, and TotalHours

Visualisation : missing values

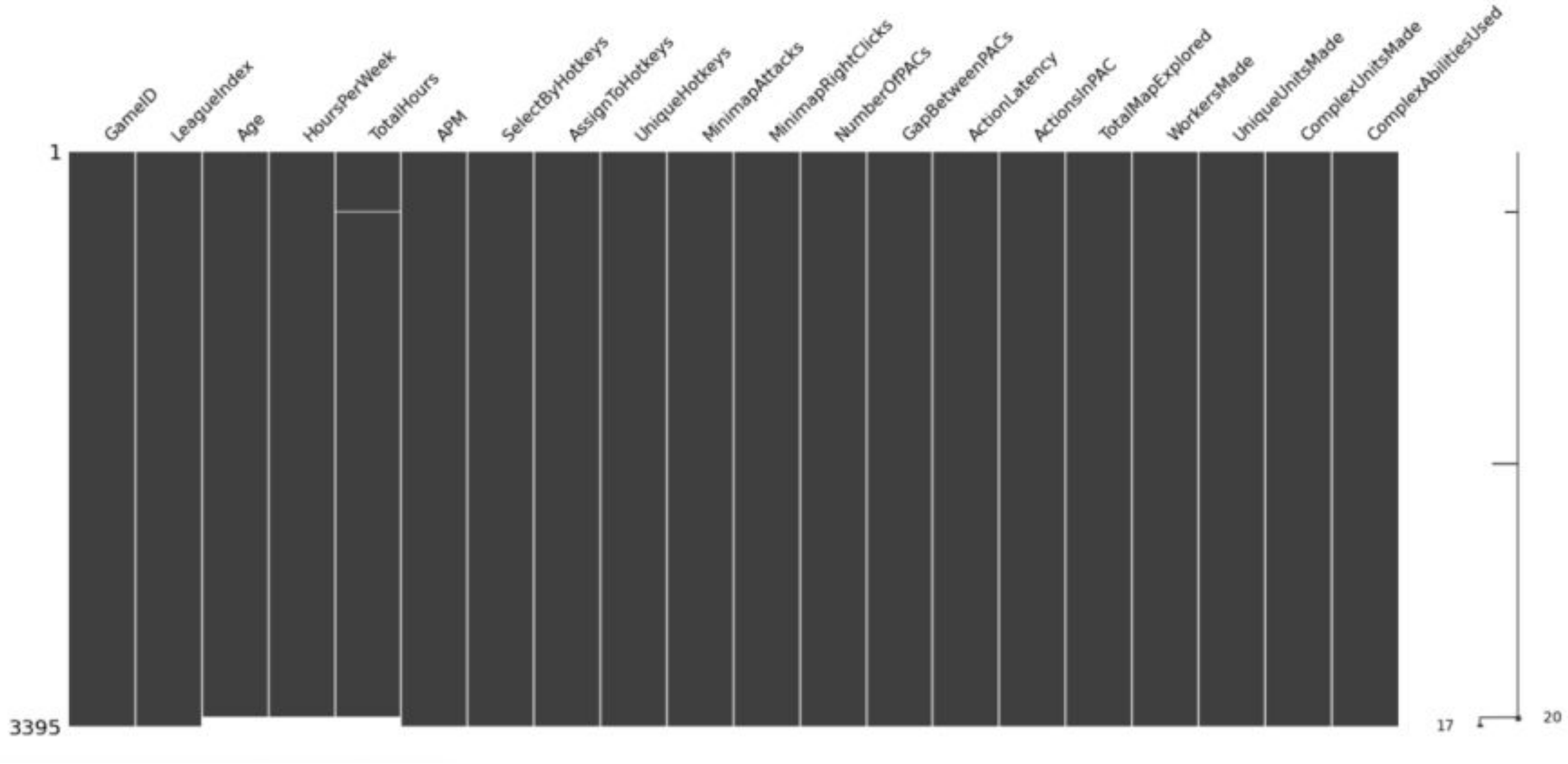
```
data.Age # okay donc on as des valeurs
# si on change vaux mieux mettre la m

0        27
1        23
2        30
3        19
4        32
         ..
3390      ?
3391      ?
3392      ?
3393      ?
3394      ?
Name: Age, Length: 3395, dtype: object
```

```
data.HoursPerWeek # on as aussi les ? pour repr

0        10
1        10
2        10
3        20
4        10
         ..
3390      ?
3391      ?
3392      ?
3393      ?
3394      ?
Name: HoursPerWeek, Length: 3395, dtype: object
```

```
data.TotalHours

0        3000
1        5000
2         200
3         400
4         500
         ...
3390        ?
3391        ?
3392        ?
3393        ?
3394        ?
Name: TotalHours, Length: 3395, dtype: object
```

There is indeed missing values in theses features and we remark that they are (maybe) on the same lines.

Visualisation : missing values



Its indeed the case except for 1 value in total hours

Visualisation : missing values

```
count=0
miss=data.TotalHours=='?'
for i in range(len(miss)):
  if miss[i] == True:
    count+=1
print(count)
# ici on as 57 missing values
```

57

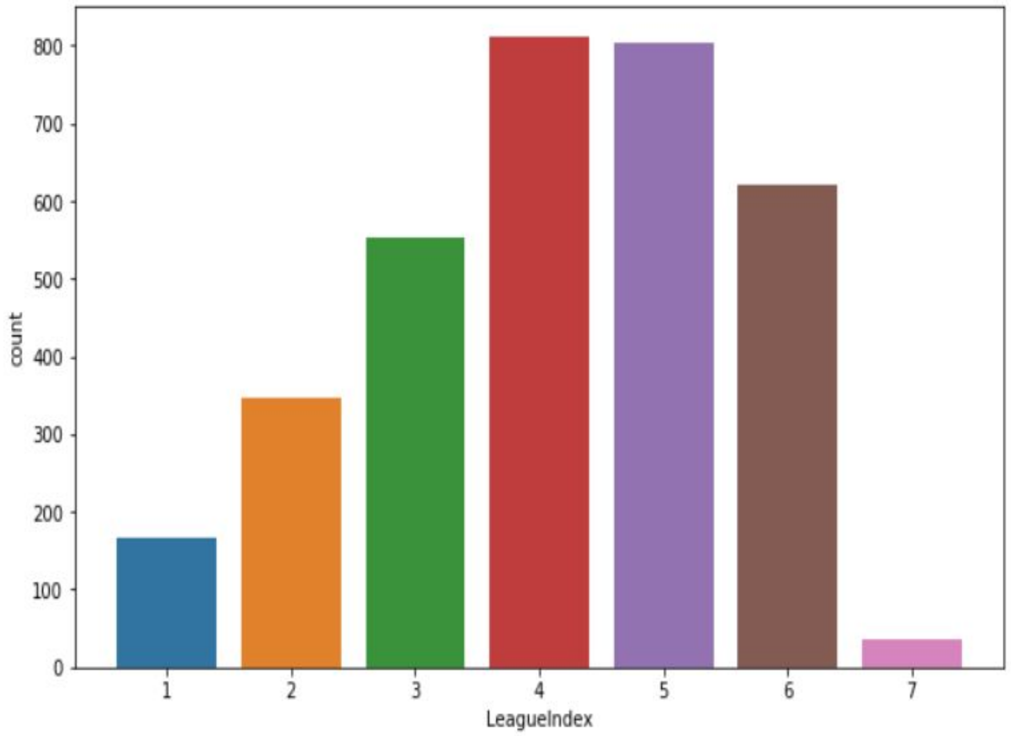
```
data.shape
```

(3395, 20)

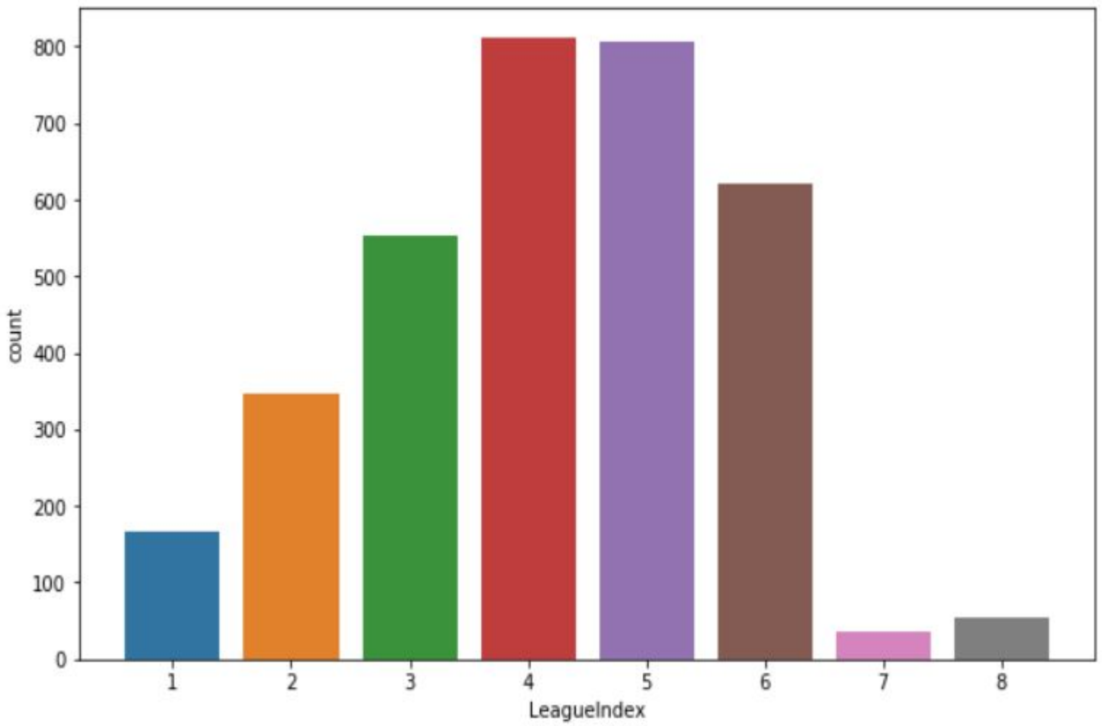
There are 57 missing values for 3395 lines in total Hours, and 56 in age and hours per week.

so we could say that we can delete them but we have remarked that all missing values corresponded to a rank (the pro rank) so deleted this latters will be non efficient for our model because we will delete an entire league index category

Visualisation : missing values



By dropping lines with missing values



Without dropping lines

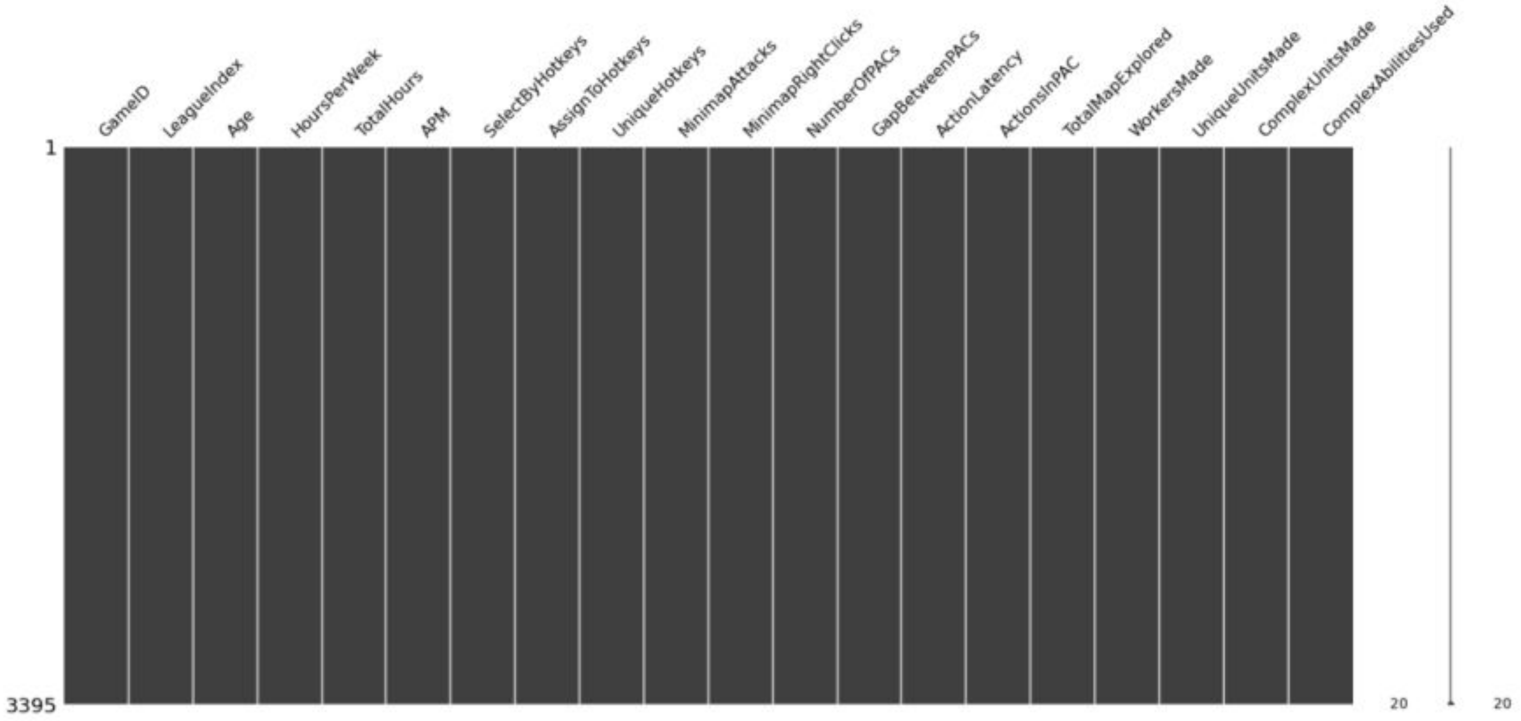
So dropping lines is not efficient

Visualisation : missing values

```
grandMaster = data[data['LeagueIndex'] == 7 ] # on prend que les grand master

ageGM=grandMaster.Age.value_counts().to_dict()     #on fait comme ça parce que
agecommun=0
effectifmax=0
for age,effectif in ageGM.items():
  if effectif>effectifmax:
    effectifmax=effectif
    agecommun=age
#agecommun

hoursGM=grandMaster.HoursPerWeek.value_counts().to_dict()
hourscommun=0
effectifmax=0
for hours,effectif in hoursGM.items():
  if effectif>effectifmax:
    effectifmax=effectif
    hourscommun=hours
#print(hourscommun)

totGM=grandMaster.TotalHours.value_counts().to_dict()
hourstotcommun=0
effectifmax=0
for hours,effectif in totGM.items():
  if effectif>effectifmax:
    effectifmax=effectif
    hourstotcommun=hours
#print(hourstotcommun)

data["Age"] = [pd.NaT if x=="?" else x for x in data["Age"]]
data["HoursPerWeek"] = [pd.NaT if x=="?" else x for x in data["HoursPerWeek"]]
data["TotalHours"] = [pd.NaT if x=="?" else x for x in data["TotalHours"]]

data["Age"] = data.Age.fillna(agecommun)
data["HoursPerWeek"] = data.HoursPerWeek.fillna(hourscommun)
data["TotalHours"] =  data.TotalHours.fillna(hourstotcommun)
```

- So we have decided to take the most often occurrence of age, hours per week and total hours of grand-master rank and attributes theses values for pro rank.
- Indeed in video games, pro players are often grandMaster players (there are many switching between them) so it's not a nonsense to do this.

Visualisation : missing values



We have no more missing values and we can continue to visualize our model

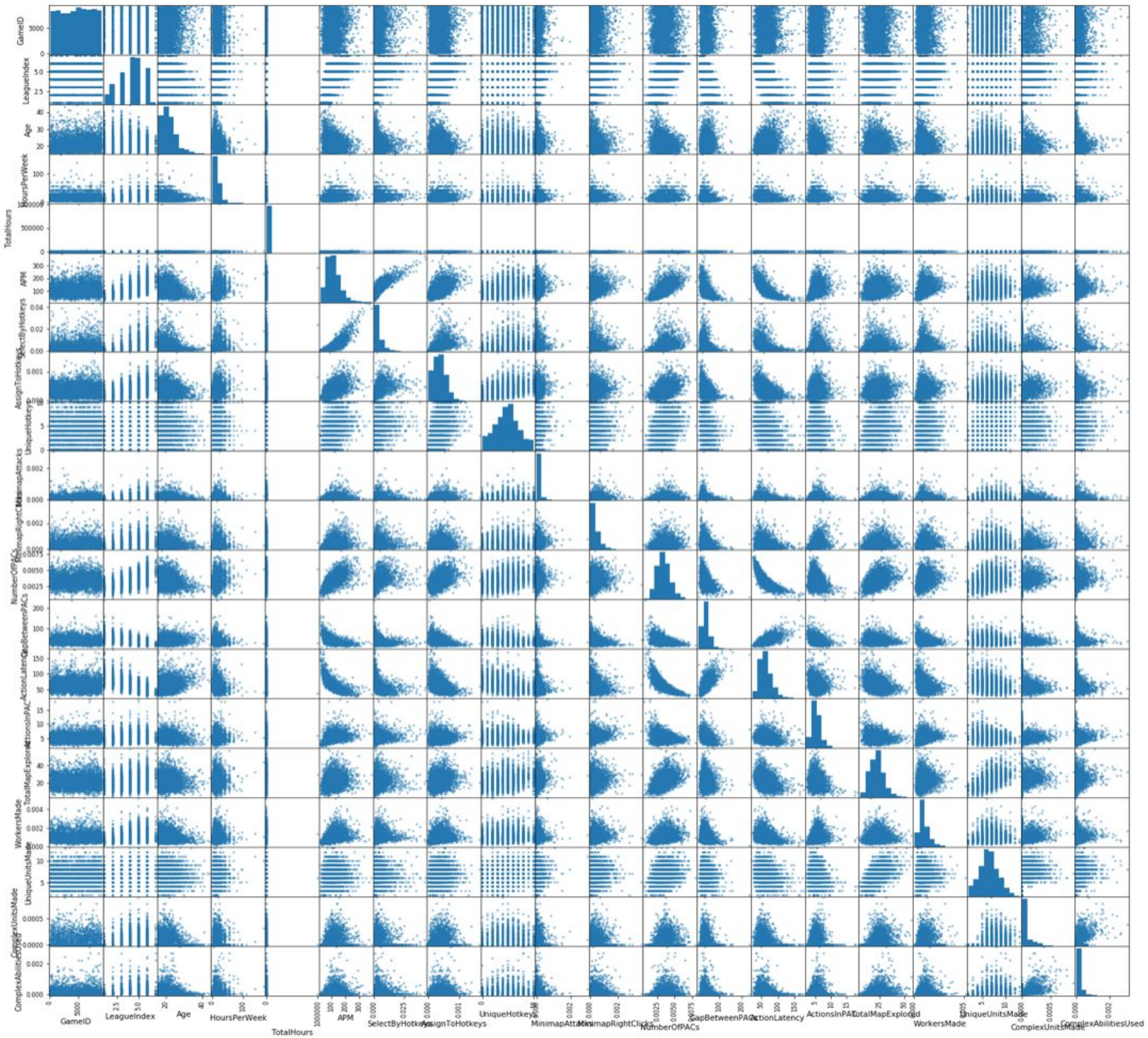
Visualisation : Scaling

	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
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

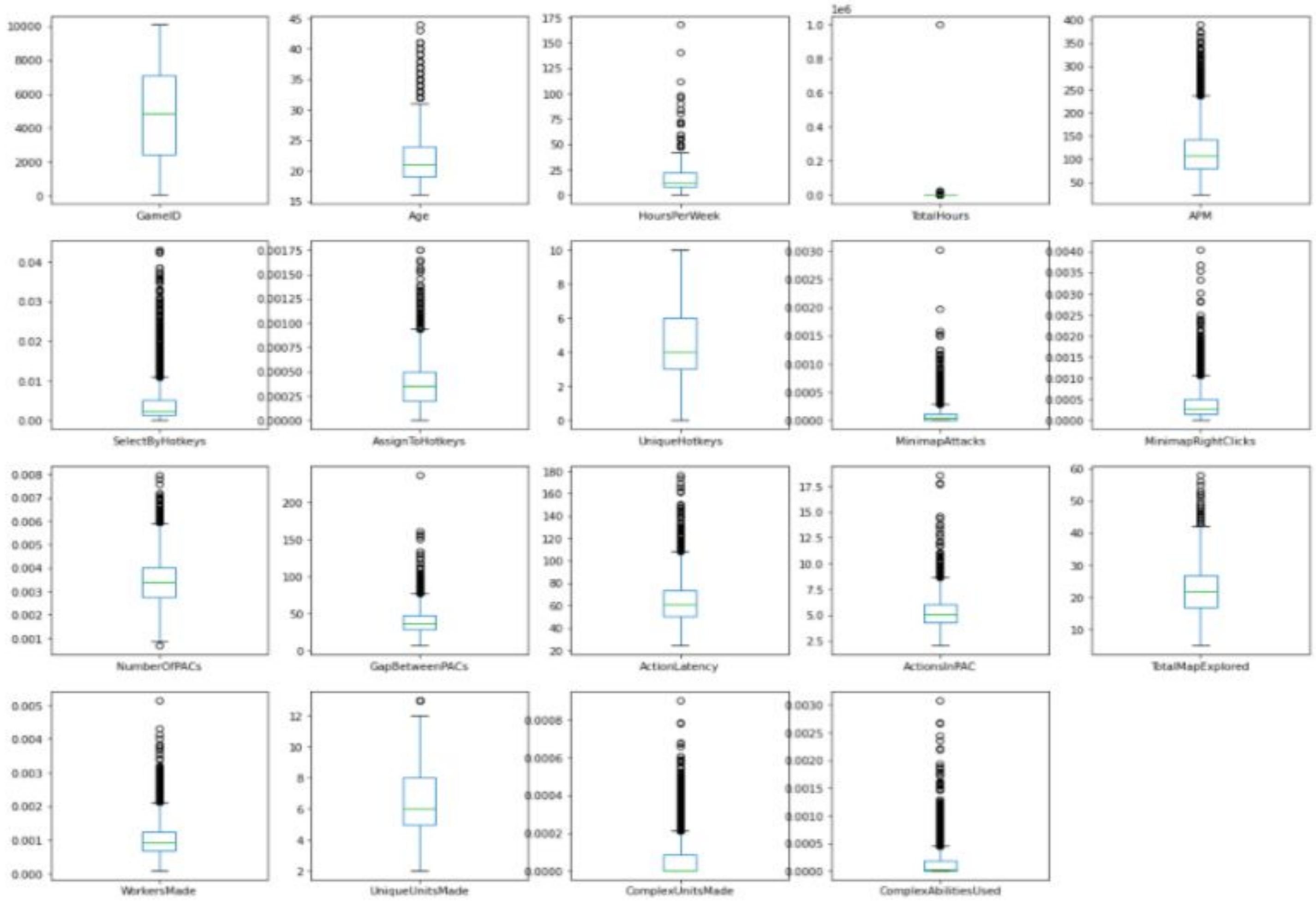
Some means are high in comparison with others means features so a scaling is intersting here

Visualisation : corrélation

- First we observe that the 5th column is flattened (total hours) so we will made boxplot to see if there is one or several outlayers.

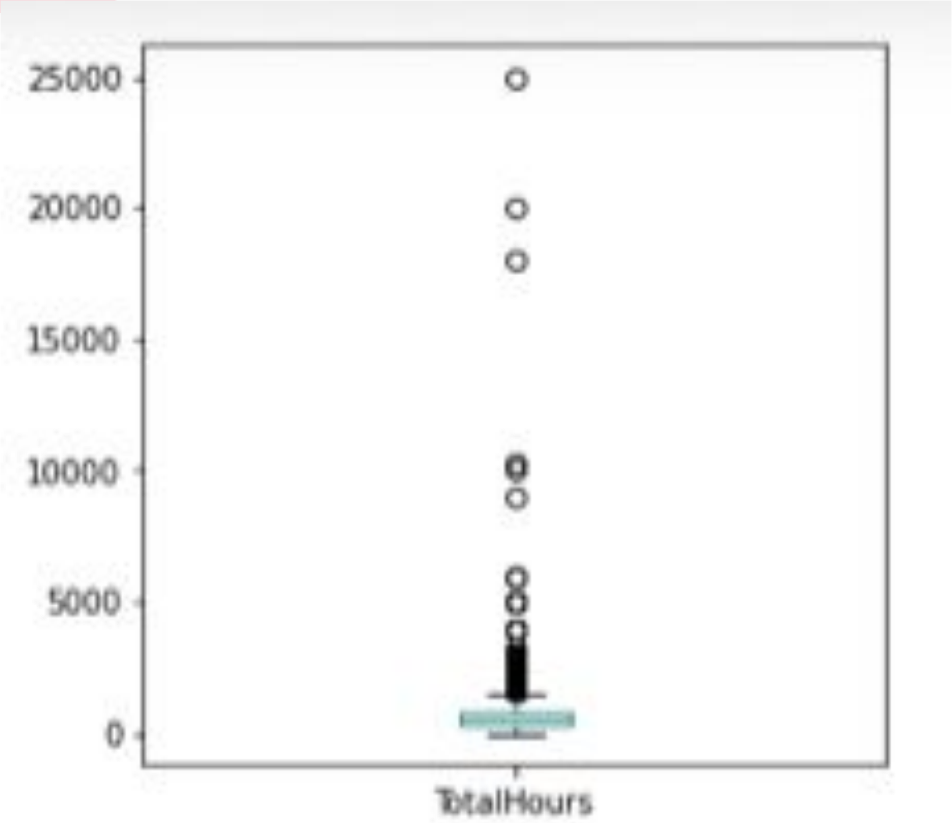


Visualisation : corrélation



- We can see one outlayers that we will delete in total hours
- we can also delete one in gap Between PACs
- and two in Minimap Attacks.

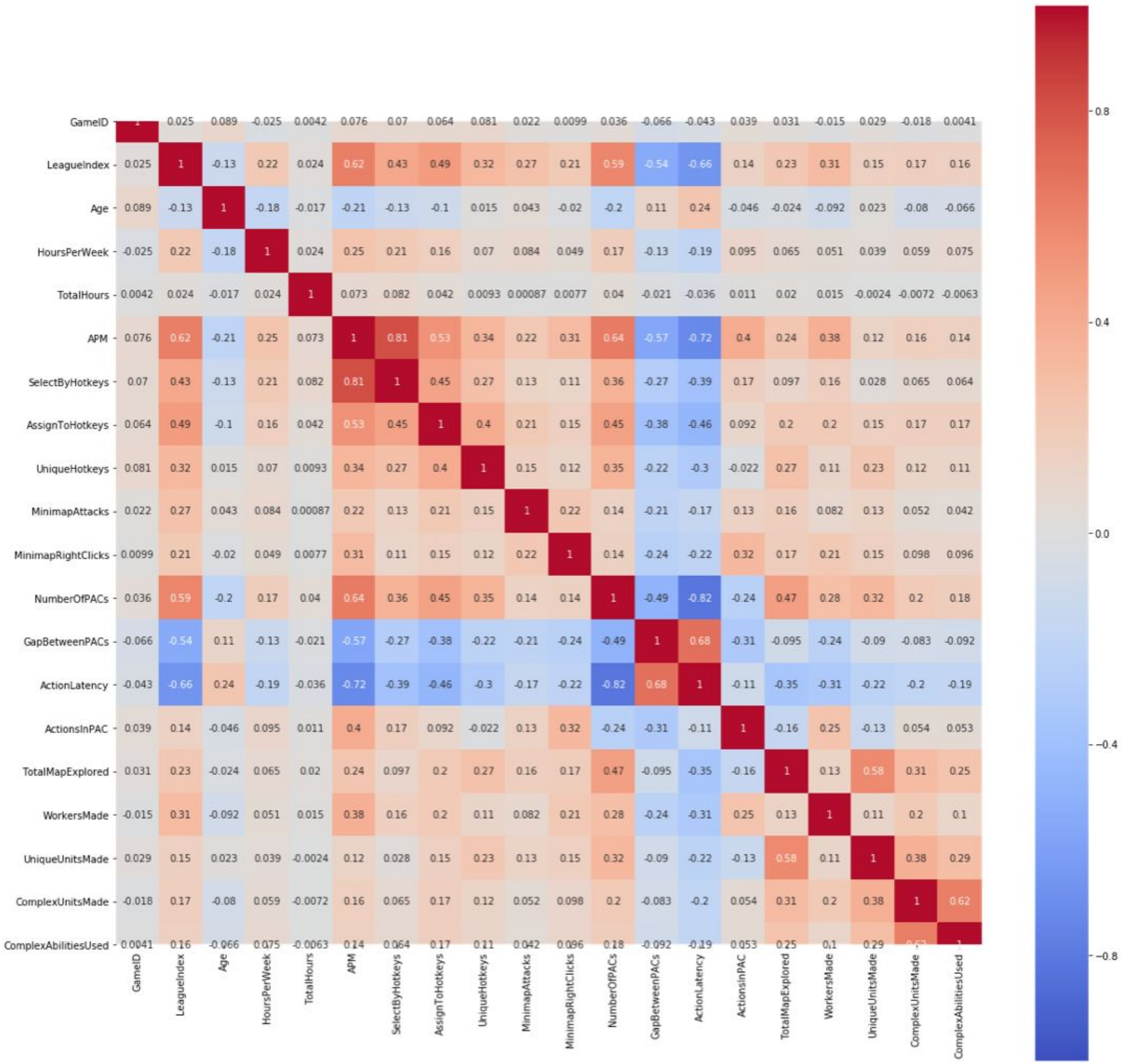
Visualisation:Corrélation



- By deleting one line in total hour we may see that the boxplot is better and this fact will improve our model (same thing with GapBetween pacs and Minimap attacks)

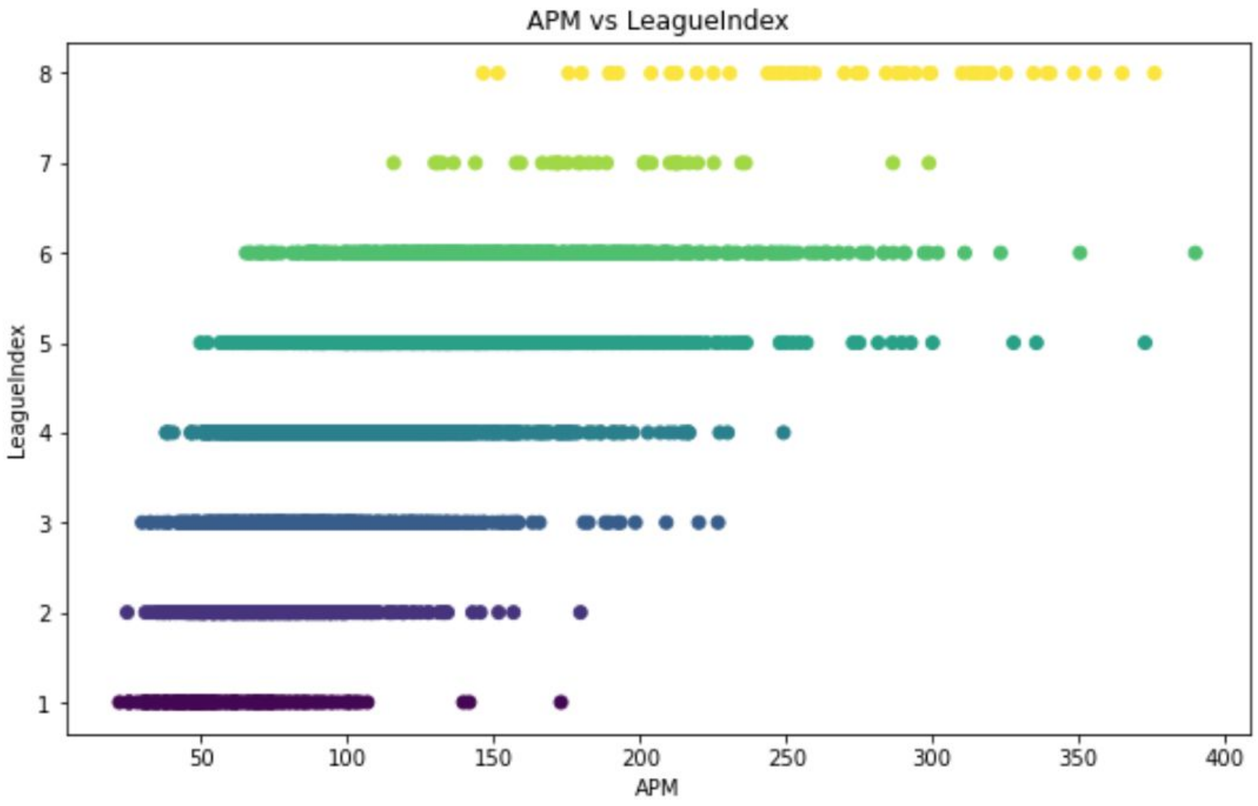
Visualisation: Corrélation

league index has a strong correlation with apm, NumberOfPACS, GapBetweenPACs, ActionLatency we can therefore assume that these features have a strong importance in the model (attention, numbers of pacs and Actionlattency are strongly correlated so we may assume that these variables define roughly the same chose)



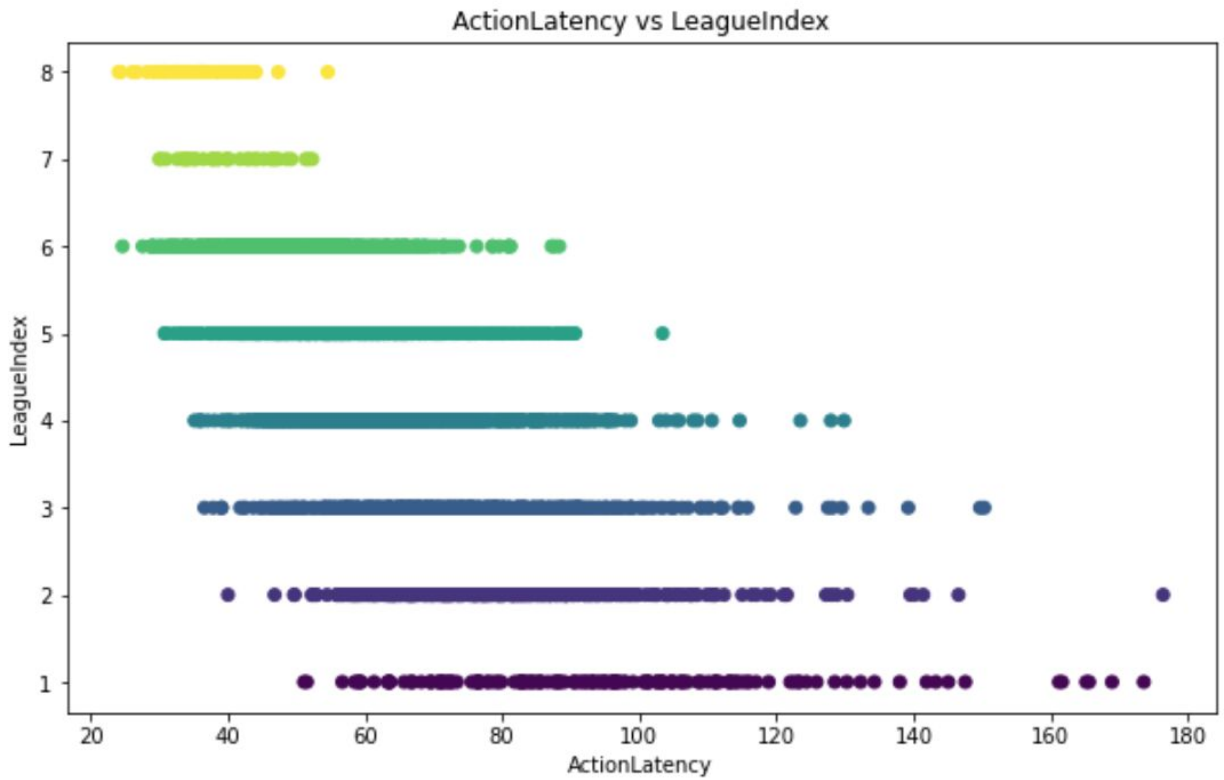
Visualisation: Corrélation

- as expected higher is the apm, more we have the capacity to have a high rank
- So this feature will be important in our model



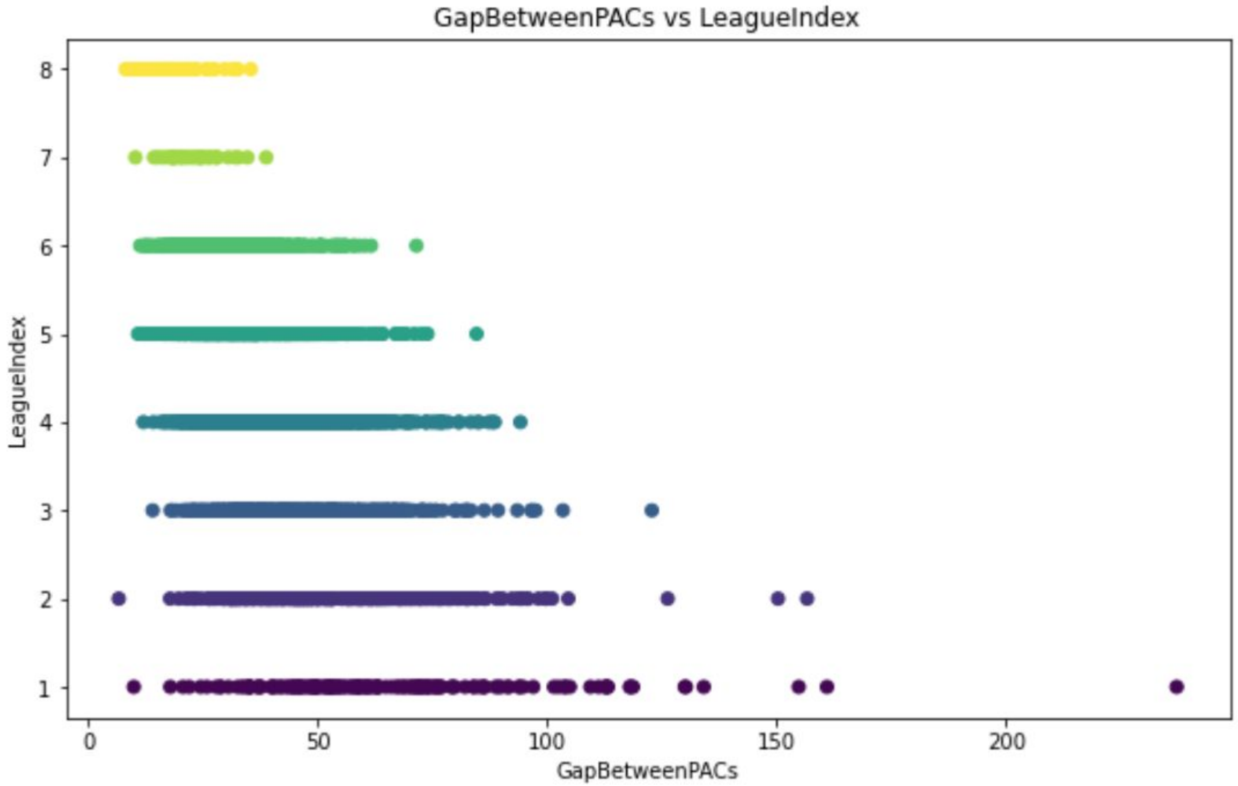
Visualisation: Corrélation

- the same thing vice versa with the latency action. The slower you are to do actions, the less likely you are to have a high ranking. Therefore someone with a very low latency action (30) cannot be at a low rank



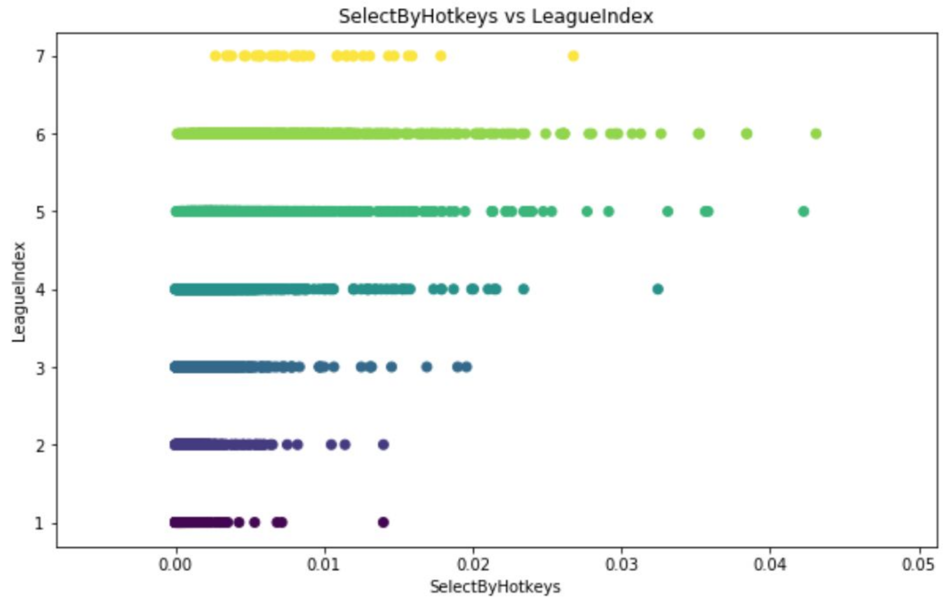
Visualisation: Corrélation

we observe a smallest difference



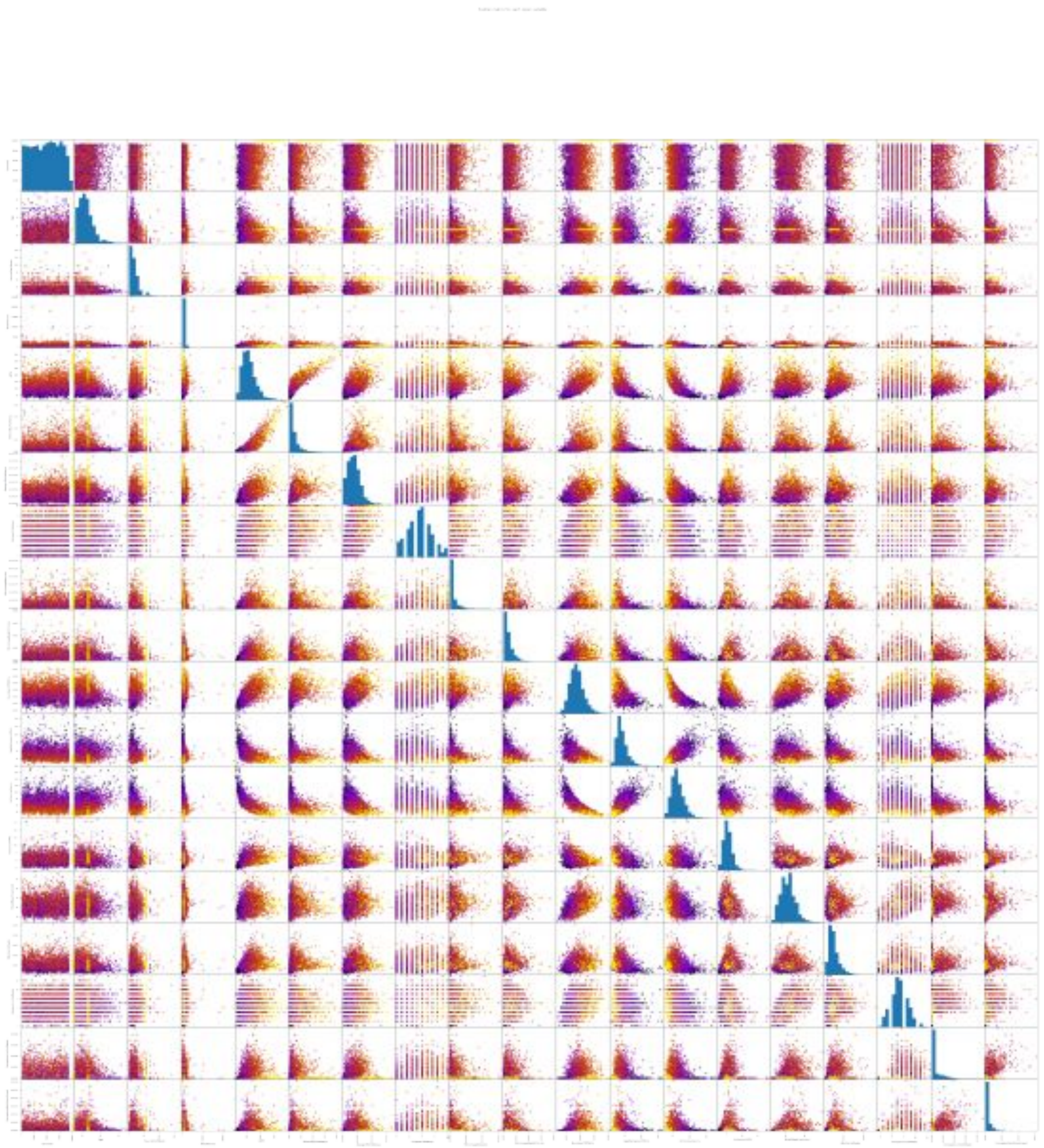
Visualisation: Corrélation

- In few case this features change nothing for the rank but we observe a tendance .
- Use a hotkeys favorize a liitle bit the rising in high rank



Visualisation: Corrélation

- We have finally this graph(colors are for the rank)



Visualisation:Corrélation

- It’s obvious that the game id is not related to players performances and its rank so we will can delete this feature.

Modelisation : few models

- We have try many classifications models : for exemple lda

```
[[ 20   7   2   7   0   0   0   0]
 [ 14  10  17  30   1   0   0   0]
 [ 11  20  22  81  16   4   0   0]
 [  3   7  23 122  43   9   0   0]
 [  0   1   6  71  81  48   1   1]
 [  0   0   1  22  40  79   4   3]
 [  0   0   0   0   1   4   2   0]
 [  0   0   0   0   0   1   1  13]]
```

	precision	recall	f1-score	support
1	0.42	0.56	0.48	36
2	0.22	0.14	0.17	72
3	0.31	0.14	0.20	154
4	0.37	0.59	0.45	207
5	0.45	0.39	0.41	209
6	0.54	0.53	0.54	149
7	0.25	0.29	0.27	7
8	0.76	0.87	0.81	15
accuracy			0.41	849
macro avg	0.41	0.44	0.42	849
weighted avg	0.40	0.41	0.39	849

Here is the confusion matrix and we have 0,41 of accuracy

We remark that its hard to guess the rank of a players between a grand-master(7) and a pro(8) because they have many similarities and because of missing values that we have previously replace

Modelisation : few models

Models	Logistic Regression	Decision Tree	KNN	LDA	Gaussian Naives Bayes	Support Vector Machine	Random Forest	Bagging
Accuracy	Test:0.42	Test 0.32	Test:0.33	Test:0.40	Test:0.34	Test:0.41	Test:0.43	Test:0.38

All models are around 0,40 accuracy because we are in a multiple classification problem so mistakes are more common.
In addition, if a model predicts a grand-master rank(7) whereas the player is a pro(8) we will have a mistake although there is only one rank between the prediction and the real rank.

So mistakes are not weighted and this, is a fact of a low accuracy

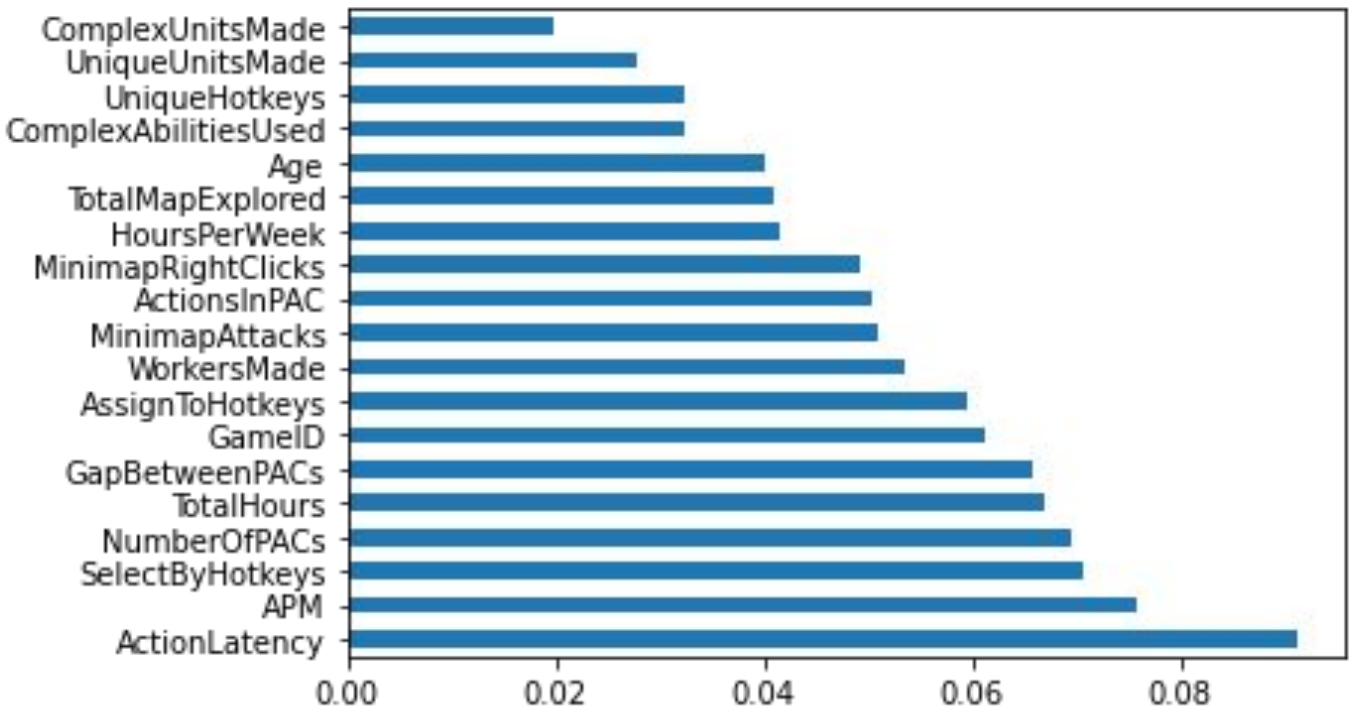
For the rest of our projects we will take the random forest model because it has the higher accuracy and it is more malleable. We will change hyper-parameters too.

Modelisation: features

- Statistical tests can be used to select those features that have the strongest relationship with the output variable.
- we will uses the chi-squared (chi^2) statistical test for non-negative features
- (more the score is high better is the feature)

	Specs	Score
3	TotalHours	1.451335e+06
0	GameID	3.254620e+05
4	APM	3.717381e+04
12	ActionLatency	9.041252e+03
11	GapBetweenPACs	7.770454e+03
2	HoursPerWeek	7.305613e+03
7	UniqueHotkeys	5.886106e+02
14	TotalMapExplored	4.588285e+02
1	Age	5.629270e+01
16	UniqueUnitsMade	4.328417e+01
13	ActionsInPAC	3.515011e+01
5	SelectByHotkeys	7.124112e+00
10	NumberOfPACs	3.695392e-01
6	AssignToHotkeys	1.455516e-01
8	MinimapAttacks	1.202775e-01
15	WorkersMade	8.730201e-02
9	MinimapRightClicks	7.614588e-02
18	ComplexAbilitiesUsed	4.264240e-02
17	ComplexUnitsMade	2.340584e-02

Modelisation: Random Forest features importance



- We have also features importance for our model
- We will delete features with the less importance to simplify and improve the model
- We can see that our suppositions at the begining of the project was true

Modelisation: Random Forest Backgroud Selection

```
#on vire total hours parce que on vois qu'elle est très peu corrélé avec la feature league index ou des features qui ont pas l'air importante
#data1 = data1.drop('TotalHours',axis=1)           #training set: 0.76  #test set: 0.40     ---> dégradation
data1 = data1.drop('GameID',axis=1)                #training set: 0.79  #test set: 0.44     ---> amélioration
data1 = data1.drop('ComplexUnitsMade',axis=1)      #training set: 0.79  #test set: 0.45     ---> amélioration
data1 = data1.drop('ComplexAbilitiesUsed',axis=1)  #training set: 0.79  #test set: 0.43     ---> Pas de changement
data1 = data1.drop('MinimapRightClicks',axis=1)    #training set: 0.79  #test set: 0.43     ---> Pas de changement
data1 = data1.drop('WorkersMade',axis=1)           #training set: 0.79  #test set: 0.43     ---> Pas de changement
data1 = data1.drop('MinimapAttacks',axis=1)        #training set: 0.77  #test set: 0.43     ---> Pas de changement
data1 = data1.drop('UniqueUnitsMade',axis=1)       #training set: 0.78  #test set: 0.44     ---> amélioration
data1 = data1.drop('ActionsInPAC',axis=1)          #training set: 0.79  #test set: 0.44     ---> amélioration
data1 = data1.drop('AssignToHotkeys',axis=1)       #training set: 0.78  #test set: 0.43     ---> Pas de changement
```

Modelisation: Random forest grid search of the hyperparameters

We will try adjusting the following set of hyperparameters:

- n_estimators = number of trees in the foreset
- max_features = max number of features considered for splitting a node
- max_depth = max number of levels in each decision tree
- min_samples_split = min number of data points placed in a node before the node is split
- min_samples_leaf = min number of data points allowed in a leaf node
- bootstrap = method for sampling data points (with or without replacement)

Modelisation: Random forest grid search of the hyperparameters

```
base_model = RandomForestRegressor(n_estimators = 10, random_state = 42)
base_model.fit(X_trainjv1, y_trainjv1)
base_accuracy = evaluate(base_model, X_testjv1, y_testjv1)
```

```
Model Performance
Average Error: 0.7727 degrees.
Accuracy = 74.83%.
```

By modify the random state (0 to 42) we have a huge amelioration of the accuracy

Modelisation: Random forest grid search of the hyperparameters

```
[ ]  rf_random.best_params_
```

```
{'bootstrap': True,
 'max_depth': 10,
 'max_features': 'sqrt',
 'min_samples_leaf': 1,
 'min_samples_split': 5,
 'n_estimators': 2000}
```

```
Model Performance
Average Error: 0.7727 degrees.
Accuracy = 74.83%.
```

We find the best parameters and apply these latters to the random forest model

We have then a little improvement (0.8%)

Modelisation: Random forest grid search of the hyperparameters

- With the cross validation we have another little improvement with the fit of hyperparameters . We have then our final model.

```
[ ]  best_grid = grid_search.best_estimator_ # best estimator, we choose that
     grid_accuracy = evaluate(best_grid, X_testjv1, y_testjv1)
```

```
Model Performance
Average Error: 0.7356 degrees.
Accuracy = 75.57%.
```

Improvement of 0,99%