Skill Craft Master Table Dataset Analysis

Rapport Skill Craft Master Table Dataset Analysis

Ryan Jebali

Christophe Haikal

Sommaire :

Exploration des données - Cleaning

Nous avons décidé pour ce projet d'analyser traitant d'informations de joueurs Star Craft. Notre objectif est dans un premier temps d'analyser et de bien comprendre les données mises à notre disposition.

Nous n'avons jamais joué à ce jeu donc certaines informations assez spécifiques n'étaient pas très familières. Après une étude un peu plus approfondie sur le sujet, nous obtenons les informations suivantes :

StarCraft est une franchise médiatique de science-fiction militaire créée par Chris Metzen et James Phinney et appartenant à Blizzard Entertainment. La série, qui se déroule au début du 26e siècle, est centrée sur une lutte galactique pour la domination de quatre espèces - les Terrans adaptables et mobiles, l'insectoïde Zerg en constante évolution, le puissant et énigmatique Protoss et la race créatrice Xel'Naga - dans une partie lointaine de la galaxie de la Voie lactée connue sous le nom de Secteur Koprulu. La série a débuté avec le jeu vidéo StarCraft en 1998. Elle s'est développée pour inclure un certain nombre d'autres jeux ainsi que huit romans, deux articles de Amazing Stories, un jeu de société et d'autres produits sous licence tels que des statues et des jouets de collection.

Les Ligues sont classées du plus bas au plus haut : Bronze, Argent, Or, Platine, Diamant, Maître et Grand Maître. La ligue Cuivre, qui était auparavant en dessous de la ligue Bronze, a été supprimée au profit de la ligue Diamant dans le patch bêta 13. La ligue des Maîtres a été ajoutée avec le patch 1.2, et la ligue des Grands Maîtres a été ajoutée en 1.3. Les joueurs sont placés dans une ligue après avoir terminé 5 matchs de placement. Après cela, un joueur peut être transféré dans une autre ligue, en fonction de ses performances. Bien que la durée et la fréquence de ces mouvements soient explicitement cachées. Toutefois, quelles que soient les performances d'un joueur, les matchs de classement ne placent pas actuellement les joueurs dans la ligue la plus élevée, celle des grands maîtres. Même avec un dossier de placement parfait, un joueur doit passer par la ou les divisions de placement initiales avant de pouvoir être attitré Grandmaster. L'objectif de ce projet est de prédire le classement d'un joueur de Starcraft II en utilisant uniquement les informations contenues dans une rediffusion. La nature en temps réel du jeu nous permet de différencier les joueurs par la rapidité et l'efficacité de leurs actions en jeu. Ceci dans le but de réduire le temps nécessaire pour arriver au bon placement.

Voici les données mises à notre disposition avec leurs explications :

1. GameID: Unique ID number for each game (integer)
2. LeagueIndex: Bronze, Silver, Gold, Platinum, Diamond, Master, GrandMaster, and Professional leagues coded 1-8 (Ordinal)
3. Age: Age of each player (integer)
4. HoursPerWeek: Reported hours spent playing per week (integer)
5. TotalHours: Reported total hours spent playing (integer)
6. APM: Action per minute (continuous)
7. SelectByHotkeys: Number of unit or building selections made using hotkeys per timestamp (continuous)
8. AssignToHotkeys: Number of units or buildings assigned to hotkeys per timestamp (continuous)
9. UniqueHotkeys: Number of unique hotkeys used per timestamp (continuous)
10. MinimapAttacks: Number of attack actions on minimap per timestamp (continuous)
11. MinimapRightClicks: number of right-clicks on minimap per timestamp (continuous)
12. NumberOfPACs: Number of PACs per timestamp (continuous)
13. GapBetweenPACs: Mean duration in milliseconds between PACs (continuous)
14. ActionLatency: Mean latency from the onset of a PACs to their first action in milliseconds (continuous)
15. ActionsInPAC: Mean number of actions within each PAC (continuous)
16. TotalMapExplored: The number of 24x24 game coordinate grids viewed by the player per timestamp (continuous)
17. WorkersMade: Number of SCVs, drones, and probes trained per timestamp (continuous)
18. UniqueUnitsMade: Unique unites made per timestamp (continuous)
19. ComplexUnitsMade: Number of ghosts, infestors, and high templars trained per timestamp (continuous)
20. ComplexAbilitiesUsed: Abilities requiring specific targeting instructions used per timestamp (continuous)

Après compréhensions et première visualisation de nos données, on remarque qu'un nettoyage s'impose.

Nous avons organisé notre nettoyage (Feature Engineering) en plusieurs étapes principales :

- Tout d'abord, nous avons supprimé toutes les lignes non complètes et donc non importantes (comportant des « ? ») car elles sont présentes en faible quantité. Les garder ne nous apporteraient donc pas beaucoup d'informations.
- Nous avons fusionné certaines ligues presque équivalentes (exemple Grand Master et Professionnel) car très minoritaires afin d'obtenir des proportions plus équilibrées (voir visualisations)
- Nous avons également réalisé par la suite du Fine-tuning grâce à la méthode de RFECV (Recursive Feature Elimination) dans le but d'obtenir la meilleure combinaison de Features possible.

Visualisations

1 – League Index Proportion

Nous nous sommes intéressés aux différences entre les différents joueurs de Star Craft. Pour commencer nous avons étudié la proportion des différences leagues de Joueurs :

there is a very small proportions of Bronze, Grandmaster and Professional league players in the dataset. We can regroup theses league in 5 leagues such as: *Bronze-Silver

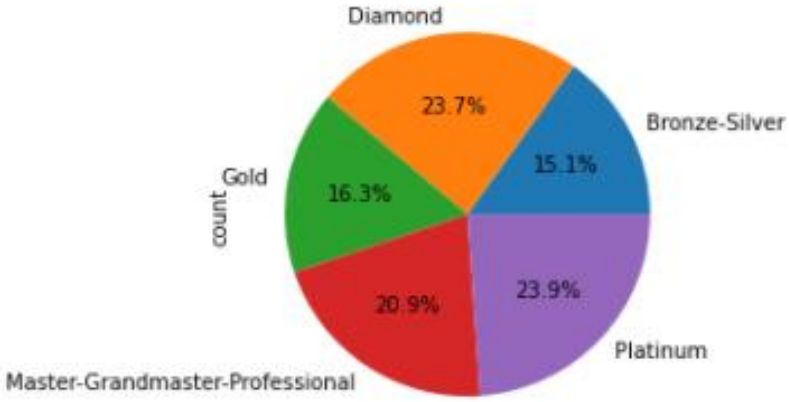
*Gold

*Platinum

*Diamond

*Master-Grandmaster-Professional

Après avoir remarqué que certaines catégories étaient de très faibles proportions, nous avons décidé d'en fusionner quelques-unes pour obtenir la proportion suivante :

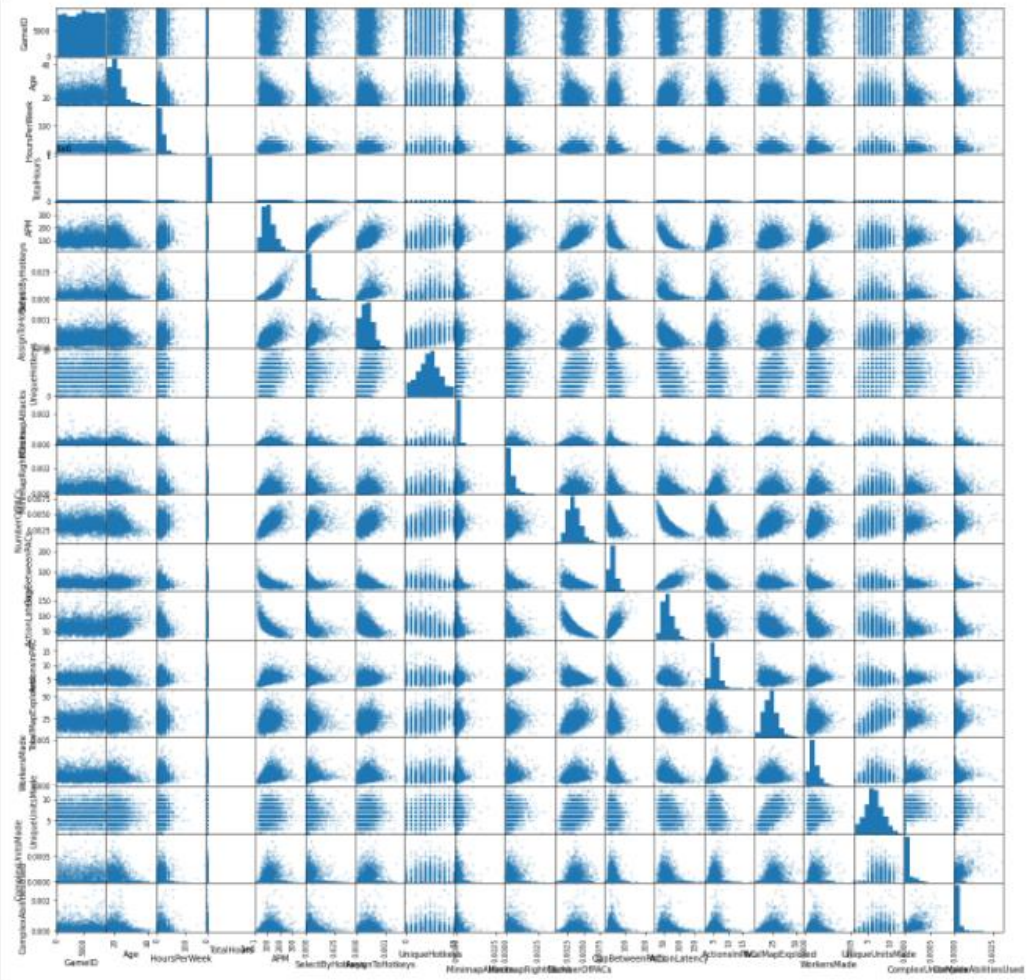


As a result we have those well balanced classes.

2 – Corrélation et Importance

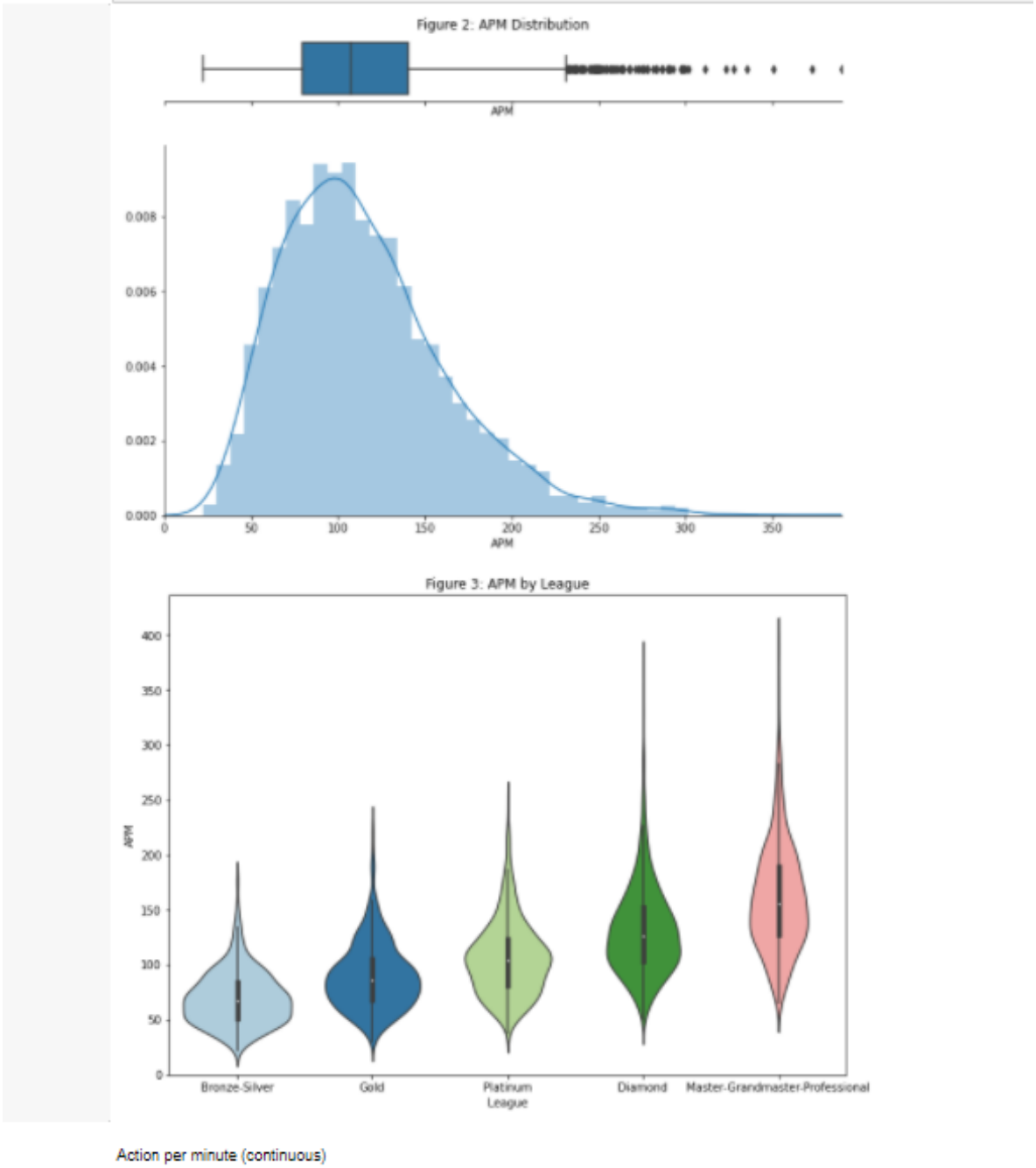
Afin de pouvoir visualiser les corrélations entre nos différentes colonnes nous avons tout d'abord réalisé une Scatter Matrix comme ci-dessous :

```
Entrée [11]: pd.plotting.scatter_matrix(df, alpha=0.2, figsize = (20,20))
             plt.show()
```



This scatter matrix plot outlines some relationships

On peut déjà voir certaines corrélations que nous avons essayé de mettre un peu plus en évidences comme avec les visualisations suivantes :



Action per minute (continuous)

L'objectif étant de visualiser l'importance de chaque colonne avec la ligue du joueur pour voir ce qui différencie les très bons joueurs des débutants.

On peut voir la proportion et la distribution qui évolue en fonction des ligues.

Autre exemple :

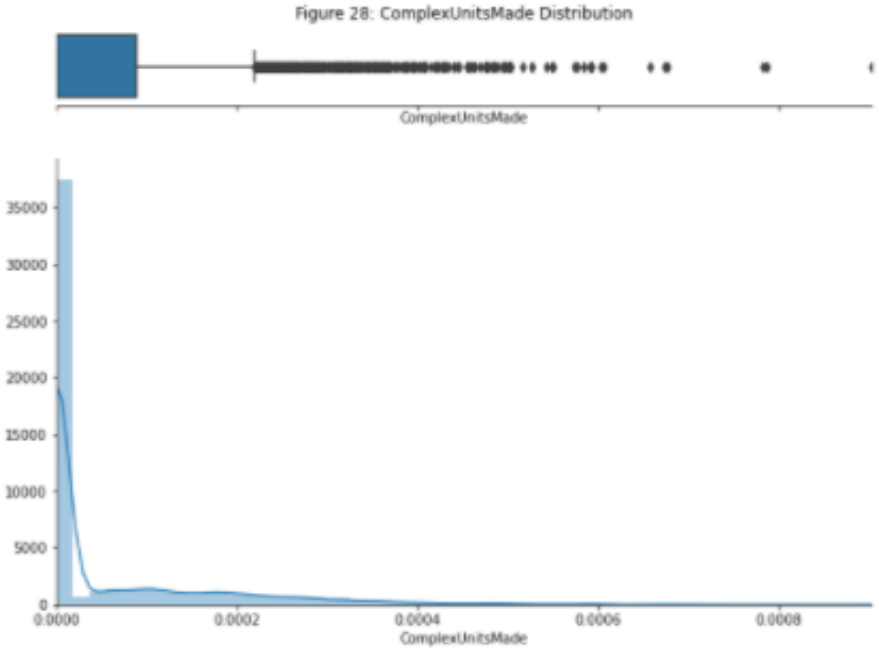


Figure 28: ComplexUnitsMade Distribution

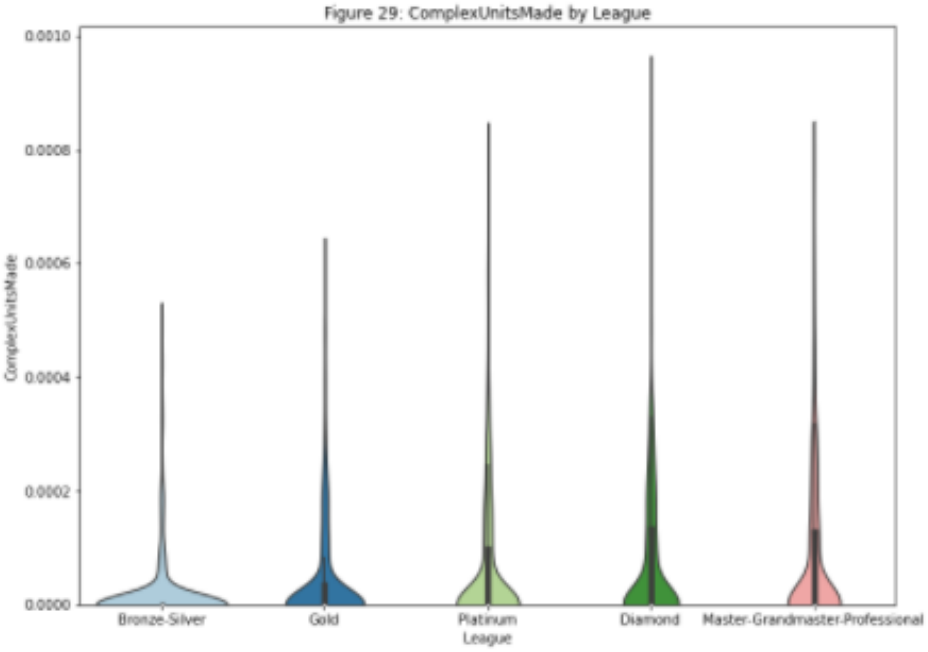
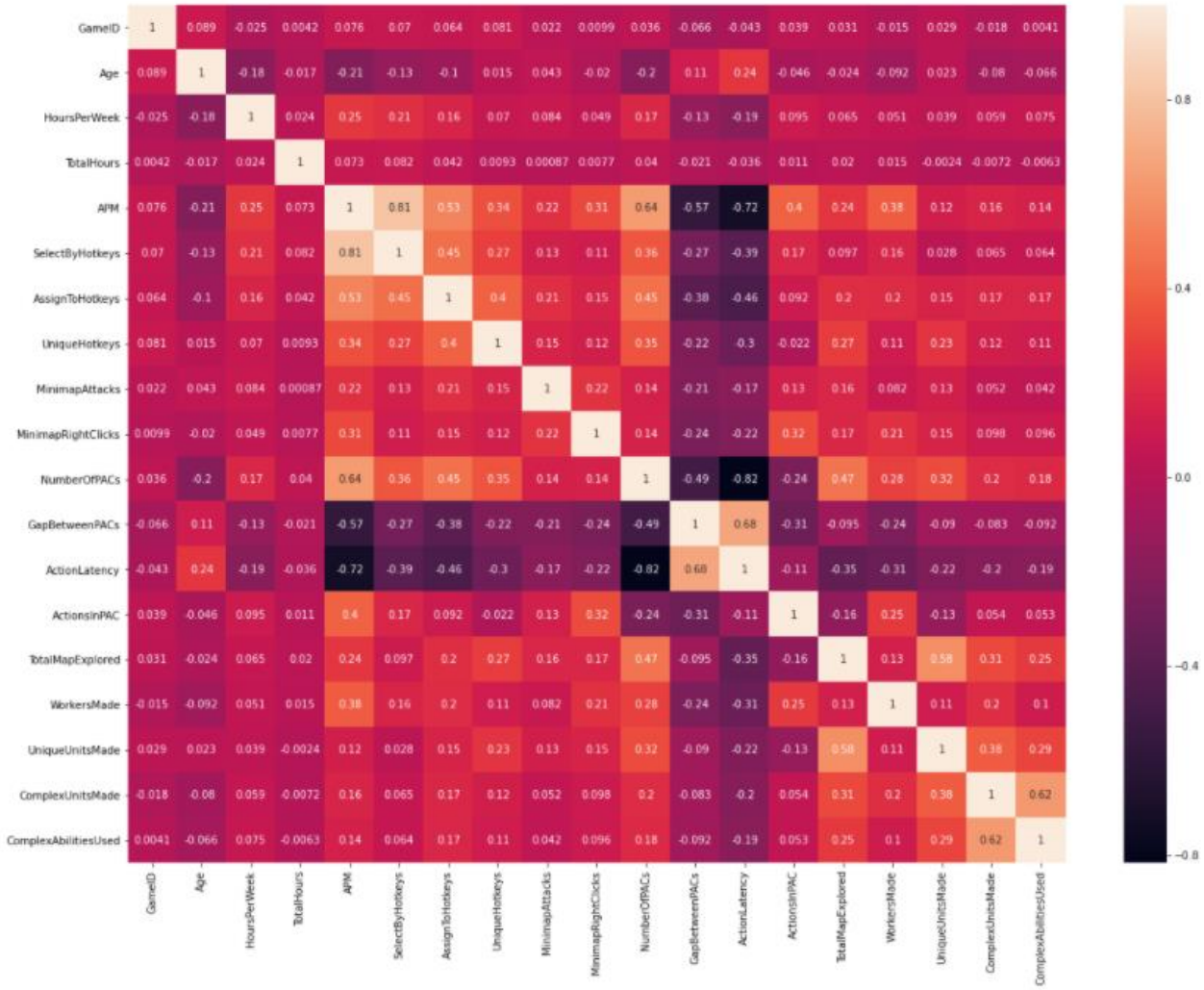


Figure 29: ComplexUnitsMade by League

Par exemple, on remarque ici que la création d'unités complexes est plus fréquente lorsque l'on monte de niveau et donc de ligues.

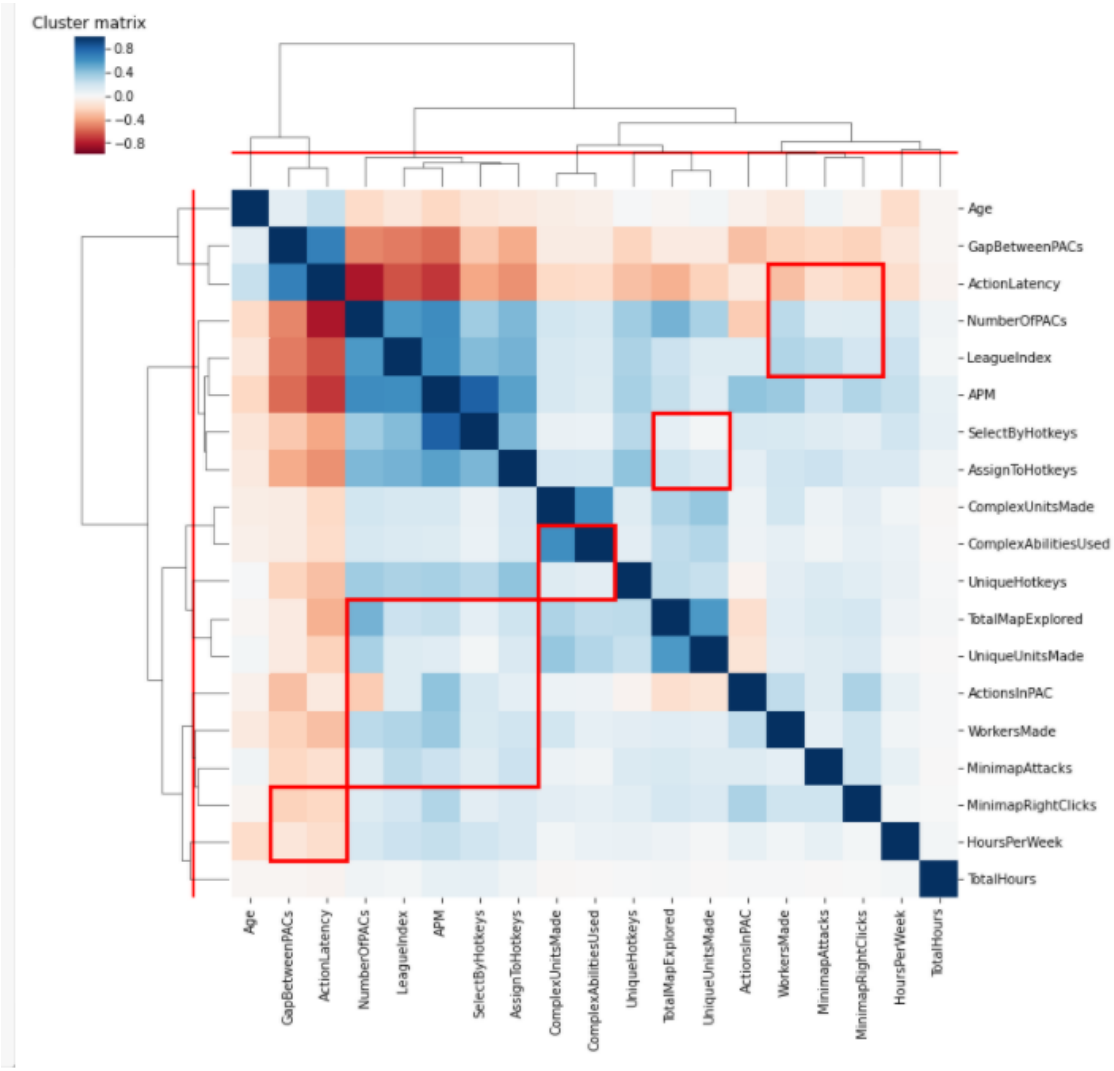
Pour mieux visualiser la corrélation entre nos Features, nous avons réalisé une HeatMap :



On peut tout de suite voir les fortes corrélations entre plusieurs matrices. Par exemple, on remarque que APM (Action par Minutes) et SelectByHotkeys (Utilisation des touches rapides) est fortement corrélé (ce qui prend du sens, plus le joueur va agir rapidement, plus il utilisera de raccourcis pour optimiser sa vitesse d'exécution). Autre exemple : ComplexUnitsMade (Nombre d'unité complexes créé) et ComplexAbilitiesUsed (Utilisation de compétence Complexes) sont fortement corrélé, ce qui signifie que plus un joueur va créé d'unité complexes, puis il utilisera de compétences complexes.

Puis, nous avons réalisé une Cluster matrix afin de regrouper nos Features en fonction de leur importance.

Cette Cluster matrix permet de regrouper les Features en « groupes de corrélations », c'est-à-dire en catégories de Features fortement corrélés.



...

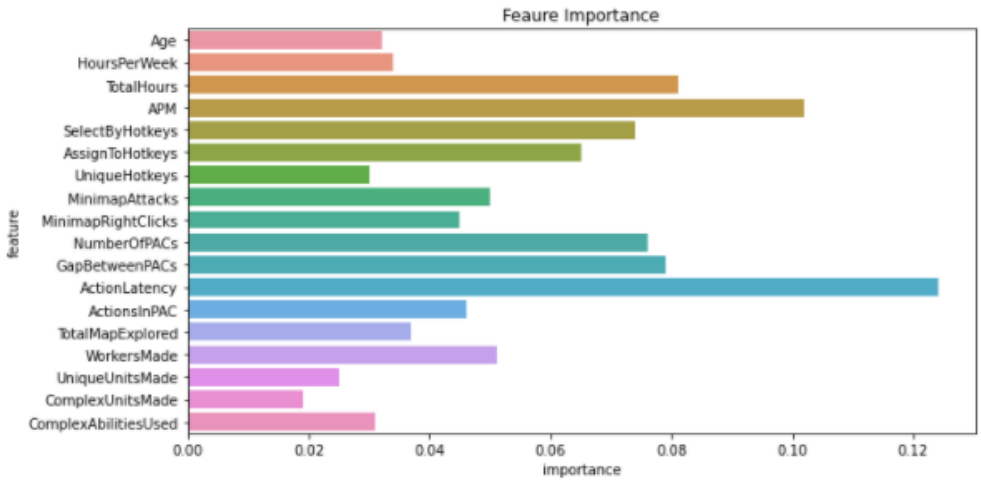
Modélisations

Avec comme objectif de comparer l'importance de chacune de nos Features par rapport au League Index des joueurs (et donc à son niveau de jeu), nous avons réalisés plusieurs algorithmes de classification.

Tout d'abord, à la suite d'un Random Forest, nous obtenons ces résultats :

Out[35]:

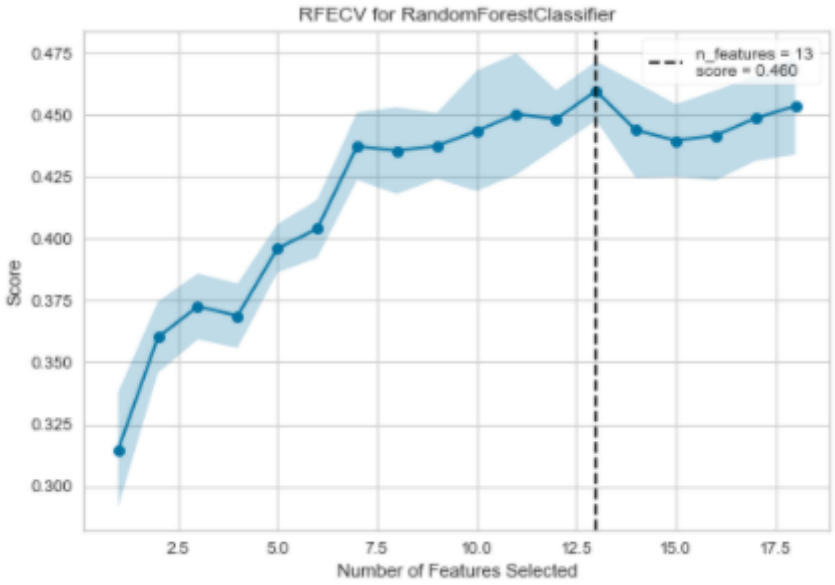
feature	importance
ActionLatency	0.124
APM	0.102
TotalHours	0.081
GapBetweenPACs	0.079
NumberOfPACs	0.076
SelectByHotkeys	0.074
AssignToHotkeys	0.065
WorkersMade	0.051
MinimapAttacks	0.050
ActionsInPAC	0.046
MinimapRightClicks	0.045
TotalMapExplored	0.037
HoursPerWeek	0.034
Age	0.032
ComplexAbilitiesUsed	0.031
UniqueHotkeys	0.030
UniqueUnitsMade	0.025
ComplexUnitsMade	0.019



Cela nous montre les Features importantes à la prédiction de notre modèle.

Nous avons par la suite utilisé un algorithme de Recursive Feature elimination. C'est un algorithme de Fine Tuning qui va tester toutes les combinaisons de Features une par une pour renvoyer les meilleures combinaisons afin d'optimiser nos résultats du modèle. Nous obtenons également le score de notre modèle avec les Features sélectionnés.

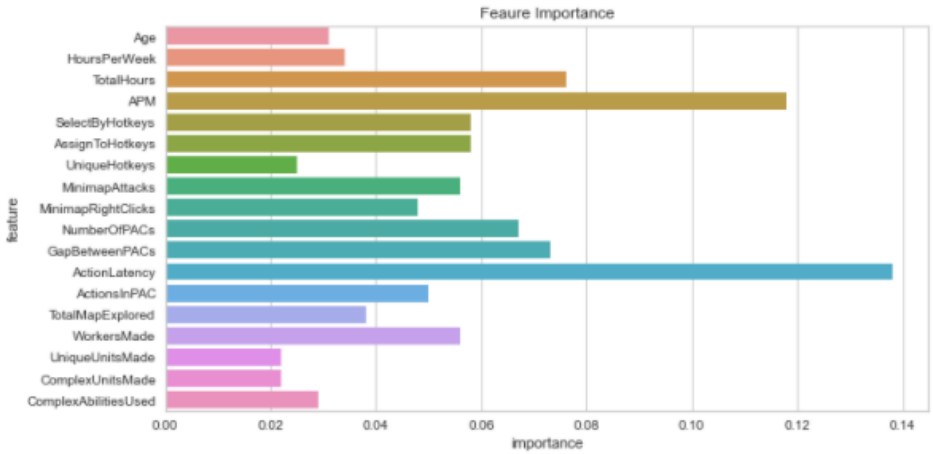
D'après notre Algorithme RFECV, les colonnes que nous devrions garder pour obtenir le score optimal de 0.460 sont :
['LeagueIndex','TotalHours','APM','SelectByHotkeys','AssignToHotkeys','MinimapAttacks', 'NumberOfPACs', 'GapBetweenPACs', 'ActionLatency', 'ActionsInPAC','WorkersMade']



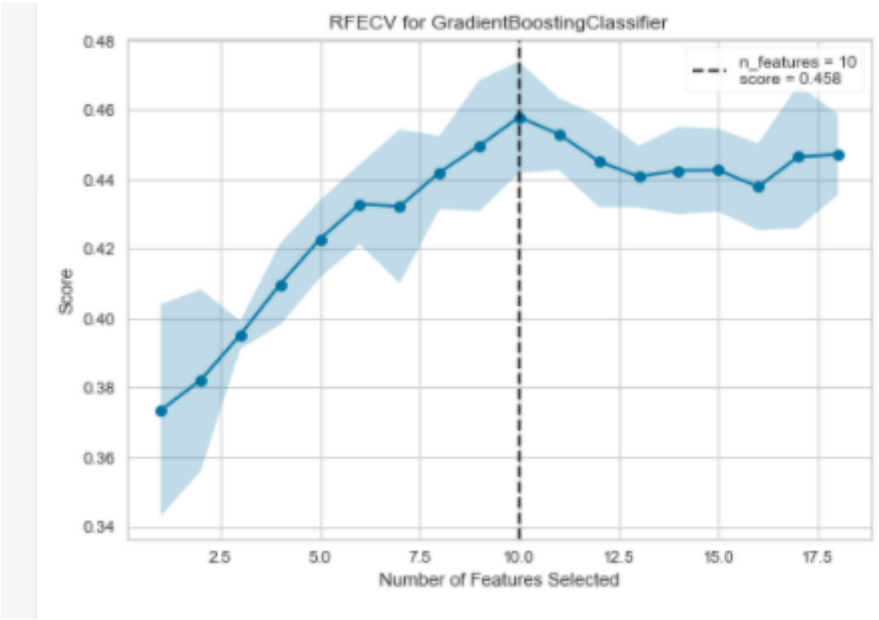
Puis, on obtient quasiment les mêmes résultats avec un Grandies Boosting Model, comme ci-dessous :

Out[48]:

feature	importance
ActionLatency	0.138
APM	0.118
TotalHours	0.076
GapBetweenPACs	0.073
NumberOfPACs	0.067
SelectByHotkeys	0.058
AssignToHotkeys	0.058
WorkersMade	0.056
MinimapAttacks	0.056
ActionsInPAC	0.050
MinimapRightClicks	0.048
TotalMapExplored	0.038
HoursPerWeek	0.034
Age	0.031
ComplexAbilitiesUsed	0.029
UniqueHotkeys	0.025
UniqueUnitsMade	0.022
ComplexUnitsMade	0.022

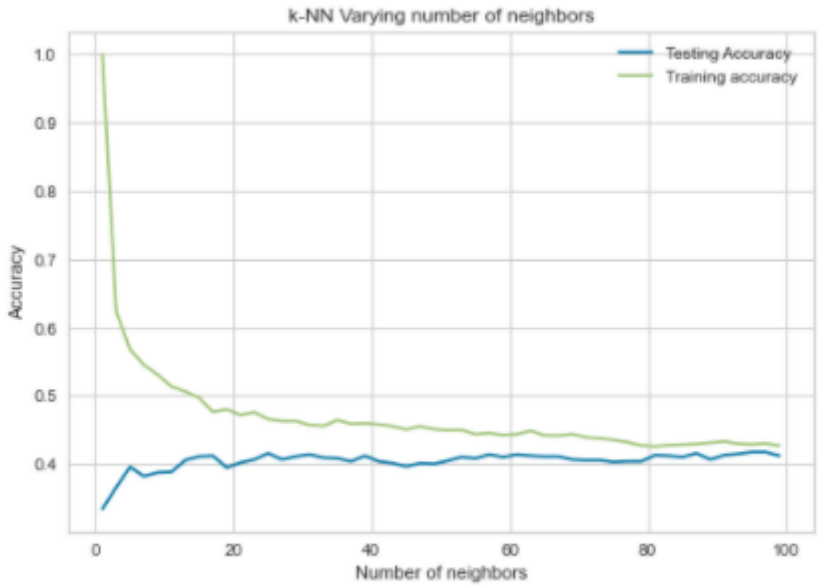


Cela nous montre les Features importantes à la prédiction de notre modèle.



On remarque qu'ici notre RFECV est moins performant que pour notre Random Forest. On garde donc notre modèle de Random Forest pour la suite.

Nous avons par la suite réalisé un KNN pour :




```
Out[39]: 95
```

Puis, à l'aide de la libraire all_estimators, nous avons exécuté un code permettant de comparer les classifier et obtenir le plus performant :

```
Out[40]: ExtraTreeClassifier()
```

Dans le même principe, on teste également par curiosité avec des algorithmes de régressions mais tout comme pour les classifiers, on obtient de plus faibles résultats que notre Random Forest à la suite de RFECV on obtient toujours de meilleurs résultats.

On test donc notre modèle afin de prédire l'accuracy des prédictions sur les différentes ligues :

Confusion Matrix

```
In [94]: y_pred_test = model_RFECV.predict(X_test_RFECV)
```

```
In [97]: from sklearn.metrics import confusion_matrix
         array = confusion_matrix(y_test, y_pred_test)
```

```
In [102]: df_cm = pd.DataFrame(array, index = [i for i in ['Bronze-Silver', 'Gold', 'Platinum', 'Diamond', 'Master-Grandmaster-Professiona
          l']],
                                columns = [i for i in ['Bronze-Silver', 'Gold', 'Platinum', 'Diamond', 'Master-Grandmaster-Professional']])
          plt.figure(figsize = (10,7))
          sn.heatmap(df_cm, annot=True)
```

```
Out[102]: <matplotlib.axes._subplots.AxesSubplot at 0x1adc2778548>
```

On voit que notre modèle est plus performant pour prédire le niveau des joueurs au niveau Platinium que pour les joueurs Gold.

Pour finir, nous avons exécuté un algorithme de réseau de neurones à l'aide d'AutoKeras pour faire de la classification (Auto-ML basé, test plein de modèles de RN et il trouve le meilleur résultat, acc plus élevés) et on obtient le meilleur modèle avec les paramètres utilisés :

On teste 50 modèles différents afin d'obtenir le plus performant :

```
Trial 28 Complete [00h 00m 11s]
val_accuracy: 0.4189189076423645

Best val_accuracy So Far: 0.4752252399921417
Total elapsed time: 00h 05m 27s
INFO:tensorflow:Oracle triggered exit
Epoch 1/100
70/70 [==============================] - 1s 2ms/step - loss: 1.5801 - accuracy: 0.2514
Epoch 2/100
70/70 [==============================] - 0s 2ms/step - loss: 1.4018 - accuracy: 0.3553
Epoch 3/100
70/70 [==============================] - 0s 2ms/step - loss: 1.3130 - accuracy: 0.3866
Epoch 4/100
70/70 [==============================] - 0s 2ms/step - loss: 1.2686 - accuracy: 0.4196
Epoch 5/100
70/70 [==============================] - 0s 2ms/step - loss: 1.2529 - accuracy: 0.4242
Epoch 6/100
70/70 [==============================] - 0s 2ms/step - loss: 1.2375 - accuracy: 0.4418
Epoch 7/100
70/70 [==============================] - 0s 2ms/step - loss: 1.2121 - accuracy: 0.4645
Epoch 8/100
70/70 [==============================] - 0s 2ms/step - loss: 1.2090 - accuracy: 0.4650
Epoch 9/100
70/70 [==============================] - 0s 2ms/step - loss: 1.2108 - accuracy: 0.4504
Epoch 10/100
70/70 [==============================] - 0s 2ms/step - loss: 1.2041 - accuracy: 0.4616
Epoch 11/100
70/70 [==============================] - 0s 1ms/step - loss: 1.1809 - accuracy: 0.4685
Epoch 12/100
70/70 [==============================] - 0s 2ms/step - loss: 1.1833 - accuracy: 0.4651
Epoch 13/100
70/70 [==============================] - 0s 2ms/step - loss: 1.1790 - accuracy: 0.4788
Epoch 14/100
70/70 [==============================] - 0s 2ms/step - loss: 1.1684 - accuracy: 0.4877
Epoch 15/100
70/70 [==============================] - 0s 2ms/step - loss: 1.1620 - accuracy: 0.4835
Epoch 16/100
70/70 [==============================] - 0s 2ms/step - loss: 1.1611 - accuracy: 0.4904
Epoch 17/100
70/70 [==============================] - 0s 2ms/step - loss: 1.1467 - accuracy: 0.4865
Epoch 18/100
70/70 [==============================] - 0s 2ms/step - loss: 1.1492 - accuracy: 0.5026
Epoch 19/100
70/70 [==============================] - 0s 2ms/step - loss: 1.1253 - accuracy: 0.5204
Epoch 20/100
70/70 [==============================] - 0s 1ms/step - loss: 1.1271 - accuracy: 0.5137
Epoch 21/100
70/70 [==============================] - 0s 1ms/step - loss: 1.1249 - accuracy: 0.5047
Epoch 22/100
70/70 [==============================] - 0s 1ms/step - loss: 1.1299 - accuracy: 0.5148
Epoch 23/100
```

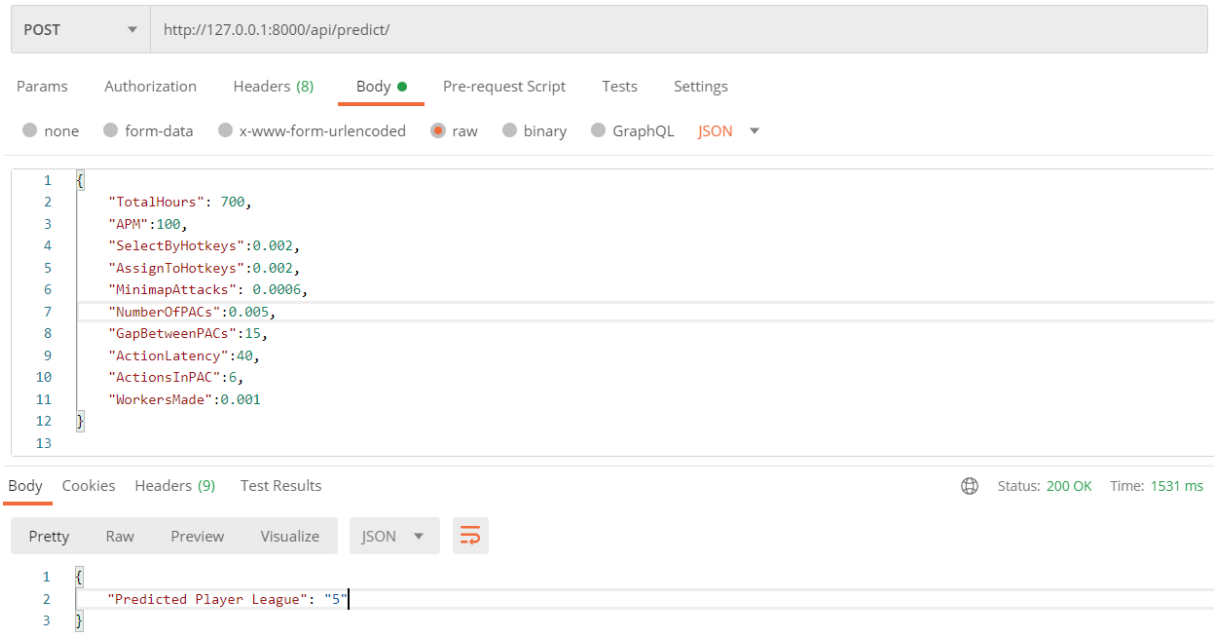
On obtient notre meilleur modèle avec les paramètres utilisés :

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 18)]              0
_________________________________________________________________
multi_category_encoding (Mul (None, 18)                0
_________________________________________________________________
normalization (Normalization (None, 18)                37
_________________________________________________________________
dense (Dense)                (None, 32)                608
_________________________________________________________________
re_lu (ReLU)                 (None, 32)                0
_________________________________________________________________
dense_1 (Dense)              (None, 32)                1056
_________________________________________________________________
re_lu_1 (ReLU)               (None, 32)                0
_________________________________________________________________
dense_2 (Dense)              (None, 32)                1056
_________________________________________________________________
re_lu_2 (ReLU)               (None, 32)                0
_________________________________________________________________
dropout (Dropout)            (None, 32)                0
_________________________________________________________________
dense_3 (Dense)              (None, 5)                 165
_________________________________________________________________
classification_head_1 (Softm (None, 5)                 0
=================================================================
Total params: 2,922
Trainable params: 2,885
Non-trainable params: 37
_________________________________________________________________
INFO:tensorflow:Assets written to: model_clf.tf\assets
```

Cependant, on trouve que c'est un peu une boite noire car on a du mal à expliquer de tels résultats donc on a gardé notre modèle précédent (best accuracy à 0.475). Donc nous avons décidé de garder notre modèle de Random Forest après RFECV.

API

Nous avons réalisé une API sur Django pour illustrer notre modèle. Nous avons rentré les 10 colonnes sélectionnées au préalable à la suite de notre Fine-Tuning. On y entre des valeurs arbitraires pour pouvoir prédire la Ligue du joueur que nous avons créé comme ci-dessous :



Nous avons mis des paramètres ressemblant à un joueur professionnel afin de voir si la prédiction allait être correcte. La League prédit pour le joueur créé est de 5 (Master / Grand Master / Professional League). La prédiction a donc bien été effectué.

Conclusion

Dans la pratique, pour des applications réelles il faudrait effectuer ces tests sur beaucoup plus de données, et non pas sur quelques lignes comme nous l'avons fait. On récupère ensuite les paramètres du modèle avec l'erreur la plus faible.

Pour conclure, ce projet était challengeant et nous as permis de développer nos compétences en python. Nous avons totalement découvert ce domaine du Jeux vidéo et ce défi très intéressant a vraiment mis l'accent sur l'importance du Fine Tuning dans l'analyse de données, ce qui est peu présent dans les enseignements qu'on a pu avoir précédemment. Après Fine Tuning Notre modèle avait de meilleurs résultats que les meilleures estimations réalisées par la suite. Les algorithmes de réseau de neurones sont parfois bien plus performants mais souvent très lourd à comprendre et à entrainer. On peut également risquer l'overfitting. Cependant cela nous stimule et nous motive à essayer de développer au maximum nos recherches et nos analyses futur dans ce domaine qui est pour tout un tournant majeur de l'innovation futur.