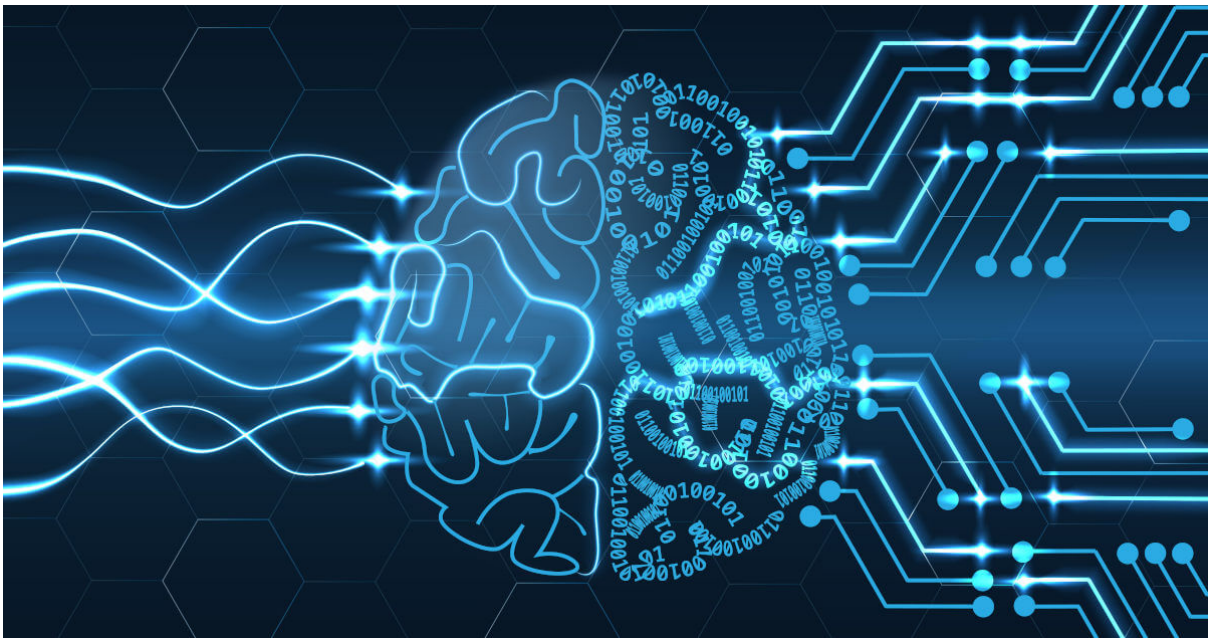
ESIEE
PARIS

DSIA-4301A : APPRENTISSAGE AUTOMATIQUE 2



Réalisé par:

Faissal Koutti

Maxime Merat

Benjamin Nahum

Encadré par Monsieur Jean-François Bercher

TABLE DES MATIÈRES:

PROBLÈME POSÉ

Un concours d'entrée à un groupe d'écoles d'ingénieurs comprend une note sur dossier. Le but est de prédire la note sur dossier en fonction de l'ensemble des données fournies. Au-delà de cela, il s'agit de comprendre quels sont les paramètres déterminants dans l'évaluation du dossier, voire d'approcher ou retrouver l'algorithme.

Les données ont été complètement anonymisées et les différentes notes décalées de quantités aléatoires.

Notre but est donc, à partir de l'ensemble des données disponibles, de prédire la note de dossier obtenue par un candidat.

Pour traiter le problème, nous pouvons utiliser l'ensemble des méthodes vues en cours et au-delà. Nous devons analyser les données, éventuellement imaginer et créer de nouvelles variables explicatives (feature engineering) par régression ou par analyse de texte text analytics, fusionner des tables, regrouper les données clustering, PCA, utiliser des approches non linéaires, etc… Il faut se montrer inventif et rigoureux, notamment dans le choix de vos paramètres et en évaluant systématiquement les performances de vos choix…

Données et features engineering

La première étape consiste à se familiariser avec les données mises à disposition, et prendre conscience de leur potentielle réelle implication et logique pour les futures prédictions.
Ainsi, en ayant d'ores et déjà repérer les variables catégorielles, les moyennes, et les colonnes qui nous sembleront inutiles par la suite, nous avons décidé de commencer de la manière la plus classique: se débarrasser des colonnes dont la quantité de données a été jugée trop faible.

Le nombre de valeurs manquantes dans chaque colonne est un élément très important. C'est pourquoi nous l'avons traité dès le départ. Nous avons fait le choix de supprimer toutes les colonnes avec plus de 90% de valeurs manquantes.


```
missing = df.isna().sum()/df.shape[0]
valeurs_manquante = df.columns[(missing > 0.9)]
valeurs_manquante
```

```
Index(['Aménagements Validés', 'Cohérence projet de formation',
       'Moyenne générale au bac (N-1)',
       'Moyenne candidat en Sciences Economiques et Sociales Trimestre 1 Terminale',
       'Moyenne classe en Sciences Economiques et Sociales Trimestre 1 Terminale',
       'Moyenne candidat en Français Trimestre 1 Terminale',
       'Moyenne classe en Français Trimestre 1 Terminale',
       'Moyenne candidat en Enseignement moral et civique Trimestre 1 Terminale',
       'Moyenne classe en Enseignement moral et civique Trimestre 1 Terminale',
       'Moyenne candidat en Sciences physiques et chimiques en laboratoire Trimestre 1 Terminale',
       ...
       'Moyenne candidat en Chimie, biochimie, sciences du vivant Trimestre 3 Première',
       'Moyenne classe en Chimie, biochimie, sciences du vivant Trimestre 3 Première',
       'Moyenne candidat en Biotechnologies Trimestre 3 Première',
       'Moyenne classe en Biotechnologies Trimestre 3 Première',
       'Moyenne candidat en Innovation technologique et eco-concept Trimestre 3 Première',
       'Moyenne classe en Innovation technologique et eco-concept Trimestre 3 Première',
       'Moyenne candidat en Sciences Sociales et politiques Trimestre 3 Première',
       'Moyenne classe en Sciences Sociales et politiques Trimestre 3 Première',
       'Moyenne candidat en Enseignements Technologiques Transversaux Trimestre 3 Première',
       'Moyenne classe en Enseignements Technologiques Transversaux Trimestre 3 Première'],
      dtype='object', length=161)
```

Ainsi, cette première étude nous a permis de nous débarrasser de 161 colonnes trop toxiques pour une élaboration d'un modèle de prédiction optimale.

Étant donné le nombre significatif de variables catégorielles présentes, nous avons ainsi décidé d'encoder ces variables afin de pouvoir les traiter, ce qui est,dans la majorité des cas, uniquement possible avec des valeurs numériques.
L'encodage que nous avons choisi d'utiliser est le one-hot encoding. Bien que pour la plupart de nos données soient des variables ordinales, nous avons poursuivi l'idée d'utiliser la fonction get_dummies() car cette méthode nous est plus familière.

```
debut = df.iloc[:, 0:6]
debut['Genre'] = debut['Genre'].replace(('H', 'F'),(0,1))
```

Ici, nous avons replacer les H(homme) et F(femme) par 0 et 1.

```
contexte = df.iloc[:, 6:13]
contexte["Engagement, Esprit d'initiative"] = contexte["Engagement, Esprit d'initiative"].replace(('Oui','Non'),(1,0
contexte = pd.get_dummies(contexte)
```

Ensuite, nous avons réalisé un get_dummies sur les varibles liés au contexte.

```
moyenne = df.iloc[:, 13:]
df = pd.concat([debut, contexte, moyenne], axis=1)
```

La Data Frame regroupe à la fois des élèves issus de la filière scientifique et des élèves issus de la filière STI2D. Lorsqu'un élève en S fait de la SVT, il ne fait pas de la science de l'ingénieur et inversement pour les élèves en STI2D.
Pour régler ce problème, nous avons fait le choix de prendre la meilleure valeur entre les deux colonnes. Les "NaN" sont donc remplacés par de vraies valeurs.
Cependant, des valeurs nulles sont encore présentes dans le Data Frame. Afin d'éviter de perdre des données, nous avons remplacé toutes les valeurs par la valeur la plus présente dans la colonne.

```
df = df.fillna(df.mode().iloc[0])
```

Afin d'optimiser un maximum notre modèle, nous nous sommes rendus compte qu'il serait très utile de rajouter des colonnes à notre Data Frame. Nous avons ajouté des colonnes telles que la moyenne en terminale en première sur toutes les matières, la moyenne scientifique en première, en terminale, l'évolution de certaines moyennes en fonction des trimestres.

```
moyenne_terminale  = df.filter(regex='Moyenne candidat en').filter(regex = 'Terminale').mean(axis=1)
df['Moyenne en Terminale'] = moyenne_terminale

moyenne_terminale_trim1  = df.filter(regex='Moyenne candidat en').filter(regex = 'Trimestre 1 Terminale').mean(axis=
df['Moyenne en Terminale Trimèstre 1'] = moyenne_terminale_trim1

moyenne_terminale_trim2  = df.filter(regex='Moyenne candidat en').filter(regex = 'Trimestre 2 Terminale').mean(axis=
df['Moyenne en Terminale Trimèstre 2'] = moyenne_terminale_trim2
```

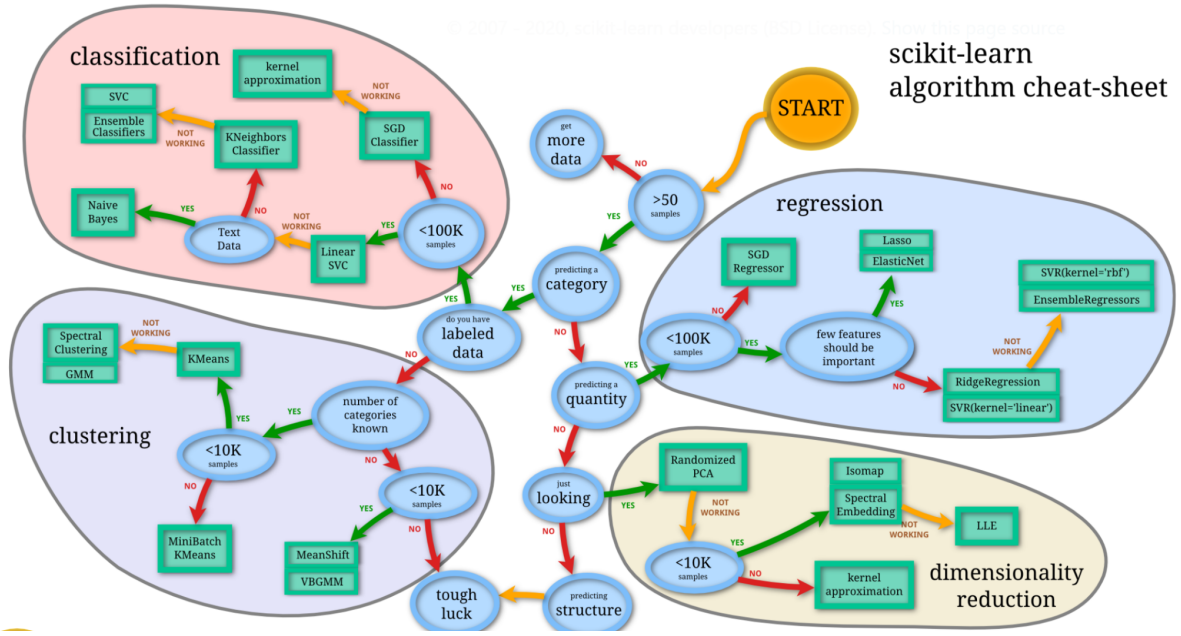
```
moyenne_premiere = df.filter(regex='Moyenne candidat en').filter(regex = 'Première').mean(axis=1)
df['Moyenne en Premiere'] = moyenne_premiere

moyenne_premiere_trim1 = df.filter(regex='Moyenne candidat en').filter(regex = 'Trimestre 1 Première').mean(axis=1)
df['Moyenne en Premiere Trimèstre 1'] = moyenne_premiere_trim1

moyenne_premiere_trim2 = df.filter(regex='Moyenne candidat en').filter(regex = 'Trimestre 2 Première').mean(axis=1)
df['Moyenne en Premiere Trimèstre 2'] = moyenne_premiere_trim2

moyenne_premiere_trim3 = df.filter(regex='Moyenne candidat en').filter(regex = 'Trimestre 3 Première').mean(axis=1)
df['Moyenne en Premiere Trimèstre 3'] = moyenne_premiere_trim3
```

Des nouvelles colonnes sont ajoutées au DataFrame. Si l'évolution de la moyenne entre les différents trimestre et années et supérieur à 1, cela signifie que la moyenne de l'élève est en baisse, à l'inverse si cette évolution est inférieur à 1, alors cela signifie que l'élève a progressé et sa moyenne est en hausse. Nous avons utilisé une nouvelle fois la fonction get_dummies.

```
new_columns_evolution_maths = (df.filter(regex="Evolution de la moyenne du candidat en Maths en Terminale entre le T
maths = new_columns_evolution_maths > 1
df["Moyenne de l'évolution du candidat en Maths"] = np.where(maths, "Baisse", "Hausse")

new_columns_evolution_physiques = (df.filter(regex="Evolution de la moyenne du candidat en Physique/Chimie en Termin
physiques = new_columns_evolution_physiques > 1
df["Moyenne de l'évolution du candidat en Physique/Chimie"] = np.where(physiques, "Baisse", "Hausse")

new_columns_evolution_svt_si = (df.filter(regex="Evolution de la moyenne du candidat en SI ou SVT en Terminale entre
svt = new_columns_evolution_svt_si > 1
df["Moyenne de l'évolution du candidat en SI ou SVT"] = np.where(svt, "Baisse", "Hausse")


new_columns_evolution_francais = (df.filter(regex="Evolution de la moyenne du candidat en Français en Première entre
francais = new_columns_evolution_francais > 1
df["Moyenne de l'évolution du candidat en Francais"] = np.where(francais, "Baisse", "Hausse")


evolution_maths = pd.get_dummies(df["Moyenne de l'évolution du candidat en Maths"], prefix = "Moyenne de l'évolution
evolution_physiques = pd.get_dummies(df["Moyenne de l'évolution du candidat en Physique/Chimie"], prefix = "Moyenne
evolution_svt_si = pd.get_dummies(df["Moyenne de l'évolution du candidat en SI ou SVT"], prefix = "Moyenne de l'évol
evolution_francais = pd.get_dummies(df["Moyenne de l'évolution du candidat en Francais"], prefix = "Moyenne de l'évo

df = pd.concat([df, evolution_maths, evolution_physiques, evolution_svt_si, evolution_francais ], axis = 1)
```

Résultats

Dans un premier temps, il a fallu définir le type de modèle de machine learning qui correspond avec notre type de prédiction et notre type de données.



En nous aidant de cette fiche nous permettant de nous situer par rapport aux modèles que propose scikit-learn, nous avons pu les tester en fonction de leurs caractéristiques, et ce, dans la même logique que pour les modèles faisant partie d'autres bibliothèques.

L'indice qui nous permet d'évaluer notre modèle est le RMSE, où Root mean square error. Il s'agit d'une des mesures les plus couramment utilisées pour évaluer la qualité des prédictions. Elle fournit une indication par rapport à la dispersion ou la variabilité de la qualité de la prédiction en utilisant la distance euclidienne.

C'est une des mesures les plus utilisées à cet effet. Il s'agit d'une règle de notation appropriée, dont la compréhension est intuitive.

Modèle	**RMSE**
LASSO	11.51
ELASTICNET	11.56
CATBOOST	8.96
ADABOOST	10.50
GBM	5.56
XGBOOST	**5.03**

Problèmes rencontrés

Bien que les difficultés ont été nombreuses et fréquentes, nous pouvons noter parmi elles :

La difficulté pour nous de créer de nouvelles colonnes significatives
En effet, après avoir nettoyé les données, nous nous sommes rendus compte que notre pré-processing n'a pas été poussé à son paroxysme. Ainsi, grâce à votre ouverture en groupe privé sur la création de nouvelles colonnes, nous avons décidé de créer des colonnes correspondant à la progression des élèves, en évaluant le rapport de leur notes du 1er au 3e trimestre. Ainsi, nous pouvons conjecturer l'évolution de l'élève sur différentes matières, ce qui a booster notre score RMSE de 14 à 8 dans un premier temps.

La compréhension des modèles
Après s'être familiarisé avec les différentes données présentes dans nos dataframes modifiées

ValueError: feature_names mismatch pour XGboost
En effectuant le model.predict() pour Xgboost, nous avons rencontré cette erreur, et uniquement sur ce modèle.
Nous avons alors convertis nos dataframes en numpy arrays grâce à as_matrix() avant d'effectuer model.fit() et model.predict(), en vain.

Nous nous sommes rendu compte que XGBoost exige que les colonnes des Data Frames d'entraînement et de test soient placées au même endroit. C'est pourquoi il est préférable de trier par ordre alphabétique les colonnes en fonction de la première lettre de leurs noms.
Ainsi, ce problème a pu être réglé grâce à la ligne

```
Entrée [83]:  test = test.reindex(sorted(test.columns), axis = 1)
```

Meilleur modèle : XGBoost

Pourquoi XGBoost Regressor ?

XGBoost est une implémentation efficace de gradient boosting qui peut être utilisée pour la modélisation prédictive par régression.

Les deux principales raisons d'utiliser XGBoost sont la vitesse d'exécution et les performances du modèle.

XGBoost domine les ensembles de données structurés sur les problèmes de modélisation prédictive de classification et de régression. La preuve en est qu'il s'agit de l'algorithme préféré des gagnants de la compétition sur la plateforme de science des données Kaggle. Il est rapide, précis et efficace, permettant une souplesse de manœuvre inédite sur le Gradient Boosting

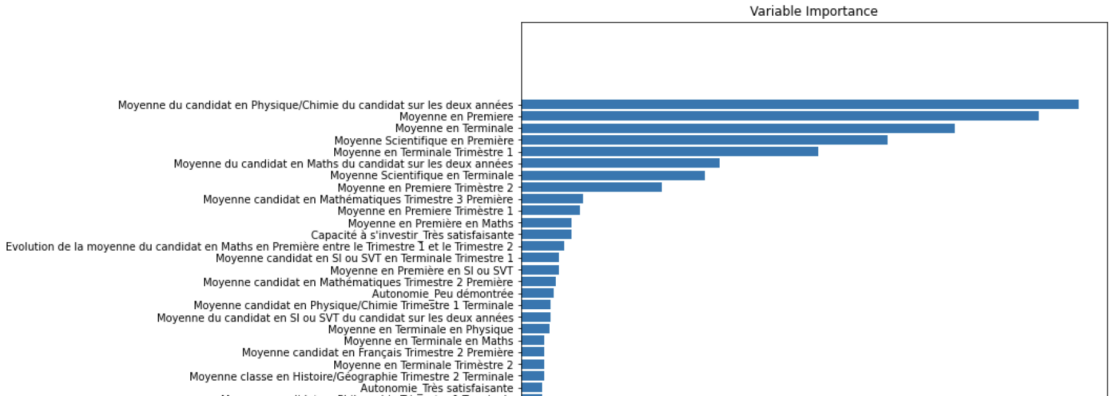
Après plusieurs essais sur d'autres modèles de régressions, nous nous sommes rendu compte que XGBoost était le plus efficace pour résoudre notre problème.

```
model = XGBRegressor(learning_rate =0.047,
 n_estimators=400,
 max_depth=4,
 min_child_weight=1,
 gamma=8,
 subsample=0.4,
 colsample_bytree=0.5,
 nthread=3,
 seed=26,
 random_state = 885,

)



model.fit(X_train, y_train)
print('score :' , model.score(X_test, y_test))
y_pred = model.predict(X_test)
print("Valeur moyenne de la prédiction :", y_pred.mean())

accuracies1 = cross_val_score(estimator = model, X = X_train, y = y_train, scoring = rmse)
print('RMSE Train =', accuracies1.mean())

accuracies2 = cross_val_score(estimator = model, X = X_test, y = y_test, scoring = rmse)
print('RMSE Test =', accuracies2.mean())
```

```
score : 0.9559099736442314
Valeur moyenne de la prédiction : 96.02925
RMSE Train = 4.7604344161079934
RMSE Test = 2.720669569253075
```

Nous avons fait le choix d'observer les variables les plus importantes avec le modèle XGBoost. D'après ce graphique, nous pouvons remarquer que les colonnes que nous avons ajouté lors du pré-processing sont celles qui ont la plus grande importance sur le modèle.

```
feature_importance = model.feature_importances_

feature_importance = 100.0 * (feature_importance / feature_importance.max())
sorted_idx = np.argsort(feature_importance)
pos = np.arange(sorted_idx.shape[0]) + .5

plt.figure(figsize=(10, 30))
plt.barh(pos, feature_importance[sorted_idx], align='center')
plt.yticks(pos, df.keys()[sorted_idx])
plt.xlabel('Relative Importance')
plt.title('Variable Importance')
plt.show()
```



Nous avons aussi réalisé une étude sur les paramètres de ce modèles pour déterminer les meilleurs.
C'est pourquoi nous utilisons ensuite RandomizedSearchCV.

```
param_dist = {
 'random_state': np.arange(1,1000),
 'n_estimators': np.arange(1, 500),
 'learning_rate' : [0.01,0.05,0.1,0.3,1],
    "max_depth" : np.arange(1,10),
    "subsample" : np.linspace(0.1,1,num=10),
 "colsample_bytree" :np.linspace(0.1,1,num=10),
 "nthread" :np.arange(1,10),


}
pre_gs_inst = RandomizedSearchCV(XGBRegressor(),
param_dist,
 cv=3,
 n_iter = 10,
 n_jobs=-1)

pre_gs_inst.fit(X_train, y_train)
```

```
print(pre_gs_inst.best_params_)
print(pre_gs_inst.score)
```

```
{'subsample': 0.9, 'random_state': 303, 'nthread': 9, 'n_estimators': 310, 'max_depth': 5, 'learning_rate': 0.05, '
colsample_bytree': 0.30000000000000004}
```

Nous avons donc utilisé ces nouveaux paramètres sur notre modèle.

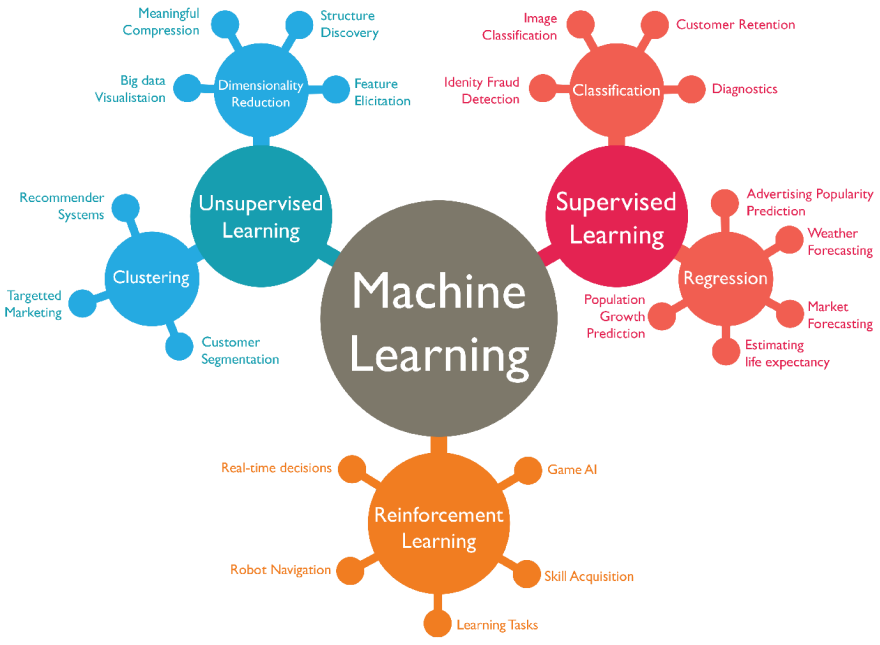
```
model = XGBRegressor(learning_rate =0.05,
 n_estimators=310,
 max_depth=5,
 min_child_weight=1,
 gamma=8,
 subsample=0.9,
 colsample_bytree=0.3,
 nthread=9,
 seed=26,
 random_state = 303,

)



model.fit(X_train, y_train)
print('score :' , model.score(X_test, y_test))
y_pred = model.predict(X_test)
print(y_pred.mean())

accuracies1 = cross_val_score(estimator = model, X = X_train, y = y_train, scoring = rmse)
print('RMSE Train =', accuracies1.mean())

accuracies2 = cross_val_score(estimator = model, X = X_test, y = y_test, scoring = rmse)
print('RMSE Test =', accuracies2.mean())
```

```
score : 0.9540121040468088
96.03206
RMSE Train = 4.786362904829203
RMSE Test = 2.599763728004204
```

Nous obtenons un meilleur score en RMSE, ce qui prouve que le changement de paramètres a été bénéfique.

Conclusion

Ce projet, dans sa globalité, nous a permis d'en savoir beaucoup plus sur l'importance d'une bonne étude de ses données, et ce que l'on cherche réellement à prédire.
En effet, d'après les différentes soumissions que nous avons effectuées, les plus grandes progressions ne résultent pas forcément sur l'appui de meilleurs modèles de machine learning et d'hyper paramètres mais plutôt sur un feature engineering plus efficace. La compréhension de l'importance de certaines features nous promulgue vers beaucoup de créations de data supplémentaires, qui n'ont rien d'anodin.
Par la suite, en plus de tourner autour d'une implémentation de dataFrame plus convaincante en vue d'une prédiction via le feature engineering, nous avons réalisé tous les trois l'importance de l'utilisation du modèle le plus adapté en fonction du type de données, son nombre, le type de prédiction et bien plus encore.
Même si les modèles de régression se sont tout de suite avérés les plus efficaces selon notre cas, comparer les résultats entre les différents modèles de scikit-learn et autres, étudier leurs paramètres ou encore les nombreuses recherches internet sur le sujet nous ont vraiment fait prendre conscience de l'immensité d'alternatives possibles et des domaines d'utilisation de ce que l'on appelle le machine learning.



Finalement, nous avons obtenu un score de 4.38 et une 6ème place dans le leaderboard public et un score de 5.36 sur le leaderboard privé qui nous classe 9ème.