# Football : Qui va gagner ?

## challenge de prédiction de résultat de match

Guillaume **Pouponneau**
Nicolas **Agraffel**
**M2 Finance data**

# Sommaire

# Contexte et objectif

Dans le cadre du data challenge QRT de cette année, on nous propose un **challenge de prédiction de résultat de match**. Nous avons à disposition des données historiques réelles au niveau des équipes et des joueurs, et **nous devons prédire quelle équipe gagne ou s'il y a un match nul**.

Les données couvrent de nombreuses ligues dans le monde entier ainsi que différentes divisions.

**Notre but est de construire un modèle prédictif riche qui peut fonctionner pour n'importe quelle ligue de football, quel que soit le niveau de compétition ou la situation géographique**.

# Base de données

1. Données fournies

## Données de base

Les fichiers zip contiennent les données d'input, qui sont divisées en 4 fichiers csv. Les données sont **séparées en HOME et AWAY**, au grain des équipes et des joueurs. Toutes les métriques proviennent de **matchs historiques réels**, agrégés depuis le **début de la saison ainsi que sur les 5 derniers matchs précédant le match à prédire**.

Les ensembles de données d'input pour les équipes comprennent les trois colonnes d'identification suivantes : **ID, LEAGUE et TEAM_NAME**

Numéro du match qui relie les 4 tables d'entrainement avec Y train et Y train supp

## Tables d'entrainement

**4**
- Train away player
- Train away team
- Train home player
- Train home team

**4**
- Test away player
- Test away team
- Test home player
- Test home team

**2**
- Y train
- Y train supp

# Base de données

1. Données fournies

## Métriques fournies

- 'TEAM_ATTACKS'
- 'TEAM_BALL_POSSESSION'
- 'TEAM_BALL_SAFE'
- 'TEAM_CORNERS'
- 'TEAM_DANGEROUS_ATTACKS'
- 'TEAM_FOULS'
- 'TEAM_GAME_DRAW'
- 'TEAM_GAME_LOST'
- 'TEAM_GAME_WON'
- 'TEAM_GOALS'
- 'TEAM_INJURIES'
- 'TEAM_OFFSIDES'
- 'TEAM_PASSES'
- 'TEAM_PENALTIES'
- 'TEAM_REDCARDS'
- 'TEAM_SAVES'
- 'TEAM_SHOTS_INSIDEBOX'
- 'TEAM_SHOTS_OFF_TARGET'
- 'TEAM_SHOTS_ON_TARGET',
- 'TEAM_SHOTS_OUTSIDEBOX'
- 'TEAM_SHOTS_TOTAL'
- 'TEAM_SUBSTITUTIONS'
- 'TEAM_SUCCESSFUL_PASSES'
- 'TEAM_SUCCESSFUL_PASSES_PERCENTAGE'
- 'TEAM_YELLOWCARDS'

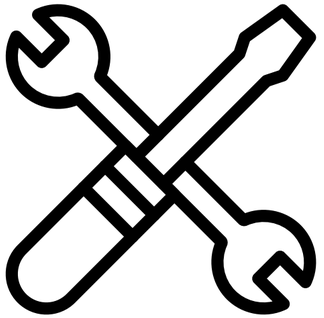
## Données de sorties à prédire

- ID : identifiant unique de match
- HOME_WINS,
- DRAW,
- AWAY_WINS,

Le score cible est l'accuracy de la prédiction pour les trois classes [HOME_WINS, DRAW, AWAY_WINS], Il existe donc pour un match trois outputs possibles, **[1,0,0]. [0,1,0] et [0,0,1].**

# Base de données

2. Méthodologie de construction



## Étape clés

1. Analyse exploratoire des données
2. Pré-traitement (nettoyage, gestion des valeurs manquantes, etc.)

## Objectif

Obtenir une base de donnée la plus propre possible afin d'appliquer des modèles de machine learning. C'est **l'étape la plus cruciale** car une base de donnée mal construite ne permets pas d'obtenir de bons résultats de prédiction. Il faut être **méticuleux** et **pertinent** dans le choix des variables et ne pas sous-estimer cette étape.

# Base de données

## 3. Analyse exploratoire des données

L’analyse des statistiques descriptive montrent que la base de données contient des NA et des valeurs “inf”. Nous reviendrons sur le traitement des NA plus tard, et les valeurs “’inf” ont été gérées comme dans l’exemple fourni par le site du challenge. Cette analyse nous a aussi montré que les données ont toutes une **échelle similaire** (distribution des valeurs entre 0 et 10). De ce fait, et sachant que des modèles de **machine learning complexe peuvent gerer sans problème des distributions non normales**, la modification des variables (en logarithme par exemple) ne semble **pas utile**.

Il convient cependant de noter que chaque variable est déclinée entre moyenne sur la saison, sur les 5 derniers match, en écart-type.. De ce fait, **il est très probable que plusieurs d’entre elles soient corrélées**. Il est nécessaire d’être attentif à ce point, en enlevant les variables les plus corrélées par exemple, ou en choisissant des hyperparamètres adaptés pour éviter le sur-apprentissage de notre modèle.

Nous ne mettrons pas de graphiques de distribution ainsi que d’exploration de données, puisqu’il y a trop de variables.

# Base de données

## 4. Stratégies adoptées

Pour réaliser ce projet, nous avons adopter **3 stratégies différentes:**

1 - Les données AWAY et HOME sont ajoutés

2 - Les données AWAY et HOME sont soustraites

3 - Construction de trois modèles différents

Ces stratégies de constructions de bases de données ont étés élaborées au fur et à mesure du challenge. Lors de l'élaboration d'une première base de donnée, nous nous sommes rendus compte de **certains points qui auraient pu être construits différemment**.

C'est pourquoi nous avons fait le choix de construire 3 bases de données différentes en testant pour chacune différente méthode de modélisation. Chaque base de donnée a ses **points forts** et ses **points faibles** que nous expliquerons dans les prochaines slides.

# Base de données

## 5. Traitement des valeurs manquantes

Nous avons traités de la même manière les valeurs manquantes de nos bases de données dans les trois cas.
Voici la méthode que nous avons utilisés:

Pour les trois méthodes, nous gérons les valeurs manquantes (NAs) en **remplaçant ces valeurs par la moyenne de la ligue** (par exemple, dans le cas de la ligue 1, nous remplaçons par la moyenne de la variable pour les clubs de ligue 1, et ainsi de suite)

# Base de données

## 6. Première stratégie

1. Comme mentionné dans le challenge, les variables **LEAGUE, TEAM_NAME et PLAYER_NAME ne sont pas inclus dans les données de test**. Nous commençons donc l'étape de pré traitement par enlever la colonne TEAM_NAME. Puis nous remplaçons les données manquantes (NA) par la moyenne de chaque ligne et de chaque database pour pouvoir les fusionner.

```
# Drop column: 'TEAM_NAME'
train_home_team = train_home_team.drop(columns=['TEAM_NAME'])
# Pour train_home_team
train_home_team = train_home_team.replace({np.inf: np.nan, -np.inf: np.nan})

# Pour train_away_team
train_away_team = train_away_team.replace({np.inf: np.nan, -np.inf: np.nan})

# Pour test_home_team
test_home_team = test_home_team.replace({np.inf: np.nan, -np.inf: np.nan})

# Pour test_away_team
test_away_team = test_away_team.replace({np.inf: np.nan, -np.inf: np.nan})
```

2. Par la suite, nous calculons la moyenne de chaque colonne par ligue et nous remplaçons les NA par les moyennes de la ligue correspondante.

```
#Voir les différentes league :
unique_leagues = train_home_team['LEAGUE'].unique()
print(unique_leagues)

#Calculer la moyenne de chaque colonne par ligue
means_per_league_home = train_home_team.groupby('LEAGUE').transform('mean')

# Remplacer les valeurs NA par les moyennes de la ligue correspondante
train_home_team = train_home_team.fillna(means_per_league_home)
```

3. Après quelques étapes de traitement (renommer les colonnes…), nous réunissons les deux ensembles (home et away) ensemble pour les fichiers train et test.

| ID | HOME_TEAM_SHOTS_TOTAL_season_sum | HOME_TEAM_SHOTS_INSIDEBOX_season_sum | HOME_TEAM_SHOTS_OFF_TARGET_season_sum | HOME_TEAM_SHOTS_ON_TARGET_season_sum |
|---|---|---|---|---|
| 0 | 3.0 | 2.0 | 5.0 | 2.0 |
| 1 | 6.0 | 8.0 | 3.0 | 6.0 |
| 2 | 4.0 | 2.0 | 5.0 | 2.0 |
| 3 | 7.0 | 5.0 | 5.0 | 6.0 |
| 4 | 3.0 | 3.0 | 2.0 | 3.0 |

# Base de données

## 6. Première stratégie

## Création de nouvelles variables

Nous avons fait le choix pour cette première stratégie de créer de nouvelles variables en plus de celles fournies de base. Ces nouvelles variables peuvent être la somme de certaines statistiques, des moyennes, ou encore des ratios.

- Sommes des tirs de l'équipe à domicile
- Somme des tirs de l'équipe à l'extérieur
- Différentiel de tirs (cumulé, moyenne, sur les 5 derniers matchs)
- Possession  de balle relative (cumulé, moyenne, sur les 5 derniers matchs)
- Différentiels de tirs dangereux (cumulé et sur les 5 derniers matchs)
- Ratio tirs sur cible/tirs totaux (cumulé, moyenne, sur les 5 derniers matchs)
- Différence de réussite offensive
- Ratio attaque dangereuses / attaques totales
- Momentum des équipes

# Création de nouvelles variables

```
# Création de la colonne 'HOME_TEAM_SHOTS_TOTAL_season_sum' (somme des tirs de l'équipe à domicile)
train_data['HOME_TEAM_SHOTS_TOTAL_season_sum'] = train_data['HOME_TEAM_SHOTS_INSIDEBOX_season_sum'] + train_data['HOME_TEAM_SHOTS_OUTSIDEBOX_season_sum']

# Création de la colonne 'AWAY_TEAM_SHOTS_TOTAL_season_sum' (somme des tirs de l'équipe à l'extérieur)
train_data['AWAY_TEAM_SHOTS_TOTAL_season_sum'] = train_data['AWAY_TEAM_SHOTS_INSIDEBOX_season_sum'] + train_data['AWAY_TEAM_SHOTS_OUTSIDEBOX_season_sum']

# 1. Différentiel de tirs (cumulé, moyenne, sur les 5 derniers matchs)
train_data['DIFF_SHOTS_ON_TARGET'] = train_data['HOME_TEAM_SHOTS_ON_TARGET_season_sum'] - train_data['AWAY_TEAM_SHOTS_ON_TARGET_season_sum']
train_data['DIFF_SHOTS_ON_TARGET_5_LAST'] = train_data['HOME_TEAM_SHOTS_ON_TARGET_5_last_match_sum'] - train_data['AWAY_TEAM_SHOTS_ON_TARGET_5_last_match_sum

# 2. Possession de balle relative (cumulée, moyenne, sur les 5 derniers matchs)
train_data['DIFF_BALL_POSSESSION'] = train_data['HOME_TEAM_BALL_POSSESSION_season_average'] - train_data['AWAY_TEAM_BALL_POSSESSION_season_average']
train_data['DIFF_BALL_POSSESSION_5_LAST'] = train_data['HOME_TEAM_BALL_POSSESSION_5_last_match_average'] - train_data['AWAY_TEAM_BALL_POSSESSION_5_last_match

# 8. Différentiel de tirs dangereux (cumulé et sur les 5 derniers matchs)
train_data['DIFF_DANGEROUS_ATTACKS'] = train_data['HOME_TEAM_DANGEROUS_ATTACKS_season_sum'] - train_data['AWAY_TEAM_DANGEROUS_ATTACKS_season_sum']
train_data['DIFF_DANGEROUS_ATTACKS_5_LAST'] = train_data['HOME_TEAM_DANGEROUS_ATTACKS_5_last_match_sum'] - train_data['AWAY_TEAM_DANGEROUS_ATTACKS_5_last_mat

# 10. Ratio tirs sur cible/tirs totaux (cumulé, moyenne, sur les 5 derniers matchs)
train_data['HOME_RATIO_SHOTS_ON_TARGET'] = train_data['HOME_TEAM_SHOTS_ON_TARGET_season_sum'] / train_data['HOME_TEAM_SHOTS_TOTAL_season_sum']
train_data['AWAY_RATIO_SHOTS_ON_TARGET'] = train_data['AWAY_TEAM_SHOTS_ON_TARGET_season_sum'] / train_data['AWAY_TEAM_SHOTS_TOTAL_season_sum']
train_data['DIFF_RATIO_SHOTS_ON_TARGET'] = train_data['HOME_RATIO_SHOTS_ON_TARGET'] - train_data['AWAY_RATIO_SHOTS_ON_TARGET']
train_data['HOME_RATIO_SHOTS_ON_TARGET_5_LAST'] = train_data['HOME_TEAM_SHOTS_ON_TARGET_5_last_match_sum'] / (train_data['HOME_TEAM_SHOTS_INSIDEBOX_5_last_ma
train_data['AWAY_RATIO_SHOTS_ON_TARGET_5_LAST'] = train_data['AWAY_TEAM_SHOTS_ON_TARGET_5_last_match_sum'] / (train_data['AWAY_TEAM_SHOTS_INSIDEBOX_5_last_ma
train_data['DIFF_RATIO_SHOTS_ON_TARGET_5_LAST'] = train_data['HOME_RATIO_SHOTS_ON_TARGET_5_LAST'] - train_data['AWAY_RATIO_SHOTS_ON_TARGET_5_LAST']

# 13. Différence de réussite offensive (but par tir)
train_data['DIFF_GOALS_PER_SHOT'] = (train_data['HOME_TEAM_GOALS_season_sum'] / train_data['HOME_TEAM_SHOTS_TOTAL_season_sum']) - (train_data['AWAY_TEAM_GOAL

# 24. Ratio "Attaques dangereuses" / "Attaques totales"
train_data['HOME_RATIO_DANGEROUS_ATTACKS'] = train_data['HOME_TEAM_DANGEROUS_ATTACKS_season_sum'] / train_data['HOME_TEAM_ATTACKS_season_sum']
train_data['AWAY_RATIO_DANGEROUS_ATTACKS'] = train_data['AWAY_TEAM_DANGEROUS_ATTACKS_season_sum'] / train_data['AWAY_TEAM_ATTACKS_season_sum']
train_data['DIFF_RATIO_DANGEROUS_ATTACKS'] = train_data['HOME_RATIO_DANGEROUS_ATTACKS'] - train_data['AWAY_RATIO_DANGEROUS_ATTACKS']

# Momentum des équipes
train_data['MOMENTUM_TEAM_HOME'] = train_data['HOME_TEAM_GAME_WON_5_last_match_average'] - train_data['HOME_TEAM_GAME_WON_season_average']
train_data['MOMENTUM_TEAM_AWAY'] = train_data['AWAY_TEAM_GAME_WON_5_last_match_average'] - train_data['AWAY_TEAM_GAME_WON_season_average']
```

# Base de données

## 6. Première stratégie

Nous avons calculé le **coefficient de corrélation** pour enlever certaines des variables qui étaient trop corrélées entre elles (80% ou plus).
**121 variables supprimées**

Cette étape marque la fin de la préparation des données pour cette première stratégie. Les données sont prêtes pour la phase de modélisation.

Pour résumer, dans cette première stratégie, nous avons construit une base de donnée en agrégeant les données HOME et AWAY, puis en créant des nouvelles variables pouvant apporter des meilleures performances pour nos modèles. Nous verrons cela dans la partie modélisation de cette présentation

**Stratégie 1 - HOME + AWAY + nouvelle variable**

# Base de données

## 7. Deuxième stratégie

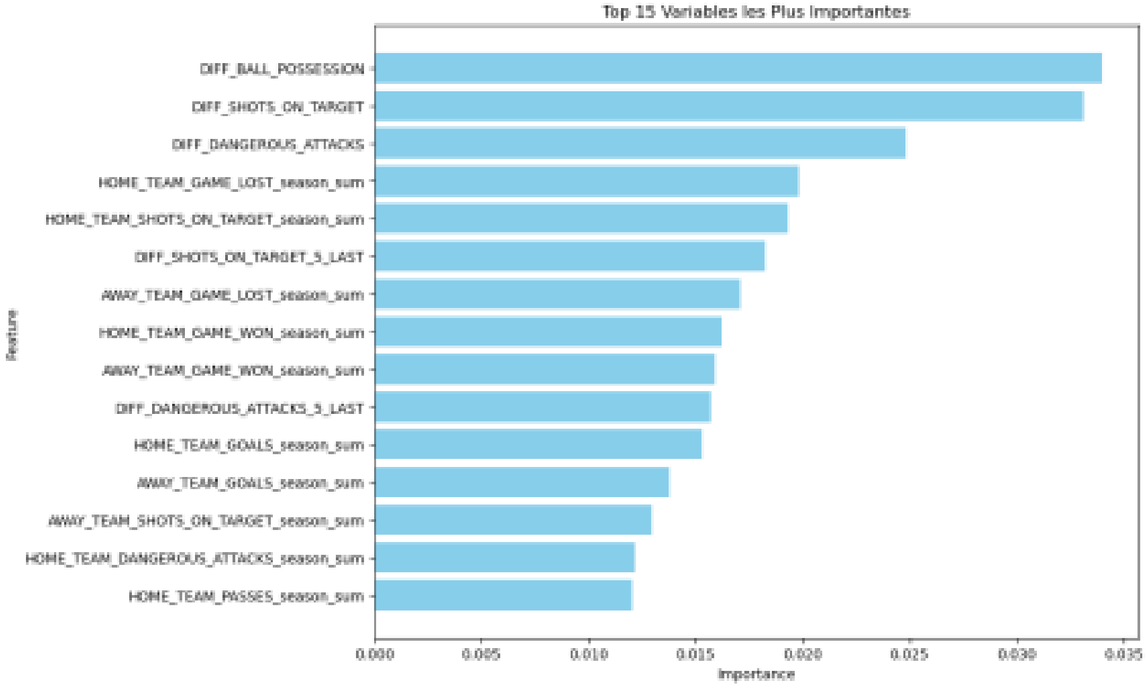
Dans cette deuxième stratégie, nous avons décidé de **soustraire les données AWAY et HOME afin de n'obtenir que des différences**. De ce fait, toutes les variables sont des "nouvelles variables". Nous le verrons dans la phase de modélisation, mais nous avons adopté cette technique puisque lors de la modélisation de la première stratégie, les variables **les plus importantes ont été celles construites comme des différences.**

**Concernant la fabrication de cette base de donnée,  nous avons traité de la même manière les NAs et la corrélation des variables**

**Stratégie 2 -  HOME - AWAY**

# Base de données

## 7. Deuxième stratégie




Comme le montre ce graphique, les variables les plus importantes de la méthode une semble être celles qui sont une difference entre 2 variables deja existante. De ce fait, il etait pertinent de tenter de créer un modèle en généralisant cette méthode à l'ensemble des variables. La base de données devient donc allégées d'un grand nombre de variable, et permet de mettre en lumière directement laquelle des équipes (Home ou away) à dominée le match

# Base de données

## 8. Troisième stratégie

Cette dernière stratégie consiste à créer 3 modèles : 1 qui prédit les victoires de l'équipe HOME, 1 qui prédit les victoires de l'équipe AWAY, et 1 qui prédit une égalité.

Nous avons construit cette base de donnée de la même manière que la stratégie 2 (HOME - AWAY).

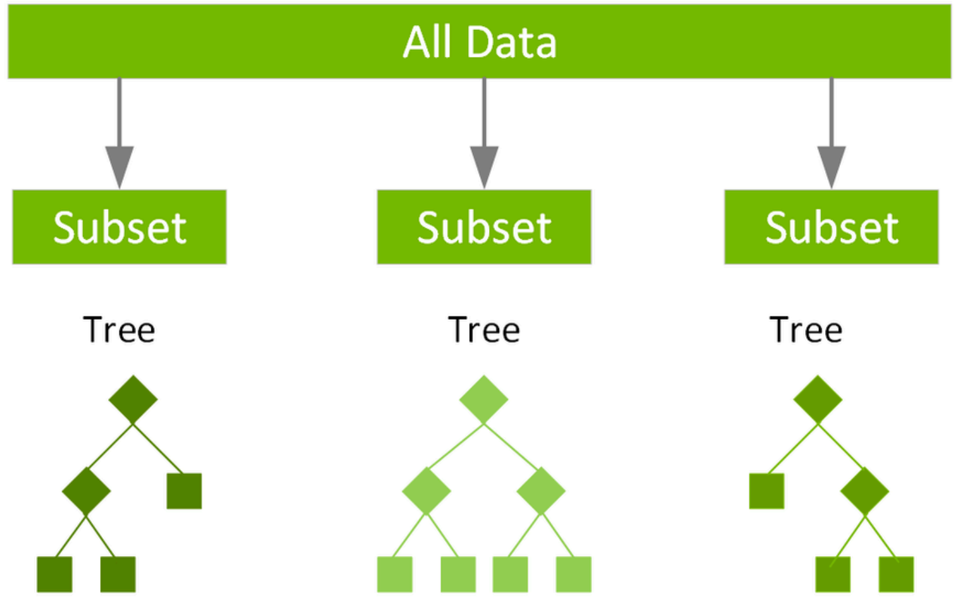
**Stratégie 3 - HOME - AWAY (diffère sur la modélisation)**

# Modélisation

1. Modèle XGBoost

## 1er modèle - XGBoost

XGBoost (Extreme Gradient Boosting) est un algorithme de machine learning basé sur des **arbres de décision,** optimisé pour la vitesse et la performance. Il utilise une technique appelée **boosting**, où des modèles faibles (arbres de décision) sont construits successivement, **chaque nouveau modèle corrigeant les erreurs des précédents**. À chaque itération, XGBoost tente de minimiser une fonction de perte en ajustant les prédictions de manière à mieux se rapprocher de la variable cible.



# Modélisation

## 1. Modèle XGBoost

Bien choisir les hyperparamètres est essentiel pour garantir un bon équilibre entre performance et généralisation du modèle. Un choix optimisé permet d'éviter l'overfitting (modèle trop complexe et peu généralisable) ou l'underfitting (modèle trop simple et peu performant). Cela maximise ainsi la capacité du modèle à extraire les relations pertinentes tout en s'adaptant correctement à des données inédites.

C’est pourquoi nous avons attaché un soin particulier au choix de nos hyperparamètres. Nous avons utilisés plusieurs méthodes pour cela: les grilles de recherche ou optuna.

# Modélisation

## 1. Modèle XGBoost

Pour optimiser notre modèle, nous avons utilisés une **grille de recherche** afin de trouver les **hyperparamètres les plus pertinents**.
Nous avons également utilliser le package optuna

### Exemple sur la première stratégie

- **subsample** : 0.8

Ce paramètre contrôle la fraction d'échantillons utilisés pour chaque arbre, permettant de réduire l'overfitting. Ici, 80 % des données sont échantillonnées pour chaque arbre, introduisant une certaine variance qui aide à améliorer la généralisation.

- **n_estimators** : 50

Il s'agit du nombre total d'arbres construits. Plus ce nombre est élevé, plus le modèle peut capturer des relations complexes, mais cela peut aussi accroître le risque d'overfitting.

- **min_child_weight** : 5

Ce paramètre contrôle la complexité de l'arbre en définissant le poids minimal nécessaire pour diviser un nœud. Des valeurs plus élevées conduisent à des arbres moins complexes. Ici, une valeur de 5 est utilisée pour éviter un sur-ajustement.

- **max_depth** : 7

La profondeur maximale des arbres. Une profondeur plus élevée permet de modéliser des relations plus complexes mais peut aussi causer de l'overfitting. La valeur 7 offre un compromis entre complexité et capacité de généralisation.

- **learning_rate** : 0.05

Ce taux d'apprentissage ajuste l'importance de chaque arbre dans le modèle final. Un taux plus faible rend le modèle plus robuste et moins sensible aux erreurs, mais nécessite plus d'arbres.

- **gamma** : 0.3

Ce paramètre contrôle la réduction de la perte requise pour faire une division. Une valeur de 0.3 impose une régularisation modérée, ce qui aide à éviter les divisions qui n'apportent pas de gains significatifs.

- **colsample_bytree** : 0.8

Il s'agit de la fraction des caractéristiques (colonnes) sélectionnées pour chaque arbre, ici 80 %. Cela aide à diversifier les arbres, améliorant ainsi la robustesse du modèle.

```
# Utiliser RandomizedSearchCV pour optimiser les hyperparamètres
random_search = RandomizedSearchCV(
    estimator=xgb_model,
    param_distributions=param_distributions,
    n_iter=20,  # Nombre d'itérations (pas trop haute pour le temps de calcul)
    cv=3,  # Validation croisée à 3 folds
    verbose=2,
    random_state=64,
    n_jobs=-1  # Utiliser tous les processeurs disponibles
)
```

# Modélisation

## 1. Modèle XGBoost

### Méthode avec Optuna

Optuna est une bibliothèque d'optimisation d'hyperparamètres qui utilise des techniques comme la **recherche bayésienne** pour explorer intelligemment l'espace des paramètres. Contrairement aux méthodes classiques, elle apprend progressivement des itérations précédentes pour orienter les tests vers les zones les plus prometteuses. En pratique, elle permet de maximiser ou minimiser une métrique définie par l'utilisateur (précision, F1-score, erreur, etc.).

### Avantage

**Efficacité** : Explore les hyperparamètres de manière intelligente, ce qui réduit le temps d'optimisation.
**Flexibilité** : Peut gérer des espaces de recherche complexes et des contraintes personnalisées.
**Simplicité** : Facile à intégrer dans des projets existants.

# Modélisation

## 1. Modèle XGBoost

Sur la première stratégie, nous avons trouvé les meilleures paramètres grâce à Optuna, et en soumettant notre fichier, nous avons obtenus un score public de **0,4850.**

**C’est notre 5ème meilleur score**

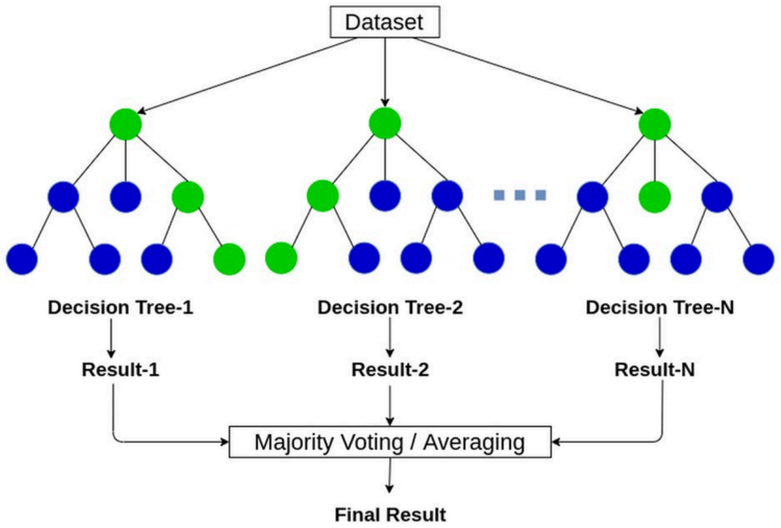
Nous étions plutôt satisfait mais étant donné la nature des variables les plus importantes pour le modèle XGBoost, nous nous sommes demandées si il n’était pas possible d’obtenir de meilleurs résultat avec un autre modèle, ou bien en construisant la base de donnée différemment.

Nous avons voulu dans un premier temps comparer les performances du modèle XGBoost (qui construit les arbres séquentiellement), avec un modèle random forest (qui construit les arbres indépendamment les uns des autres en parallèle).

# Modélisation

## 2. Random Forest

Le modèle Random Forest est un **algorithme d'apprentissage supervisé** qui combine plusieurs arbres de décision indépendants pour améliorer la précision et réduire le risque de surapprentissage (overfitting). Chaque arbre est construit à partir d'un échantillon aléatoire des données et utilise une **sélection aléatoire de variables à chaque division**, ce qui introduit de la diversité. Les prédictions finales sont obtenues par vote **majoritaire (classification) ou moyenne (régression) des prédictions des arbres.**



# Modélisation

## 2. Random Forest

### Les hyperparamètres

Comme pour le modèle XGBoost, nous avons utilisé une grille de recherche afin de trouver les meilleures hyperparamètres de notre modèle.

```
param_distributions = {
    'n_estimators': [50, 100, 200],
    'max_features': ['sqrt', 'log2'],
    'max_depth': [5, 8, 10],
    'min_samples_split': [10, 15, 20],
    'min_samples_leaf': [4, 6, 8],
    'bootstrap': [True]
}

# Utiliser RandomizedSearchCV pour trouver les meilleurs hyperparamètres
rf_random = RandomizedSearchCV(estimator=rf, param_distributions=param_distributions,
                               n_iter=20, cv=3, verbose=2, random_state=42, n_jobs=-1)
```

### Résultat

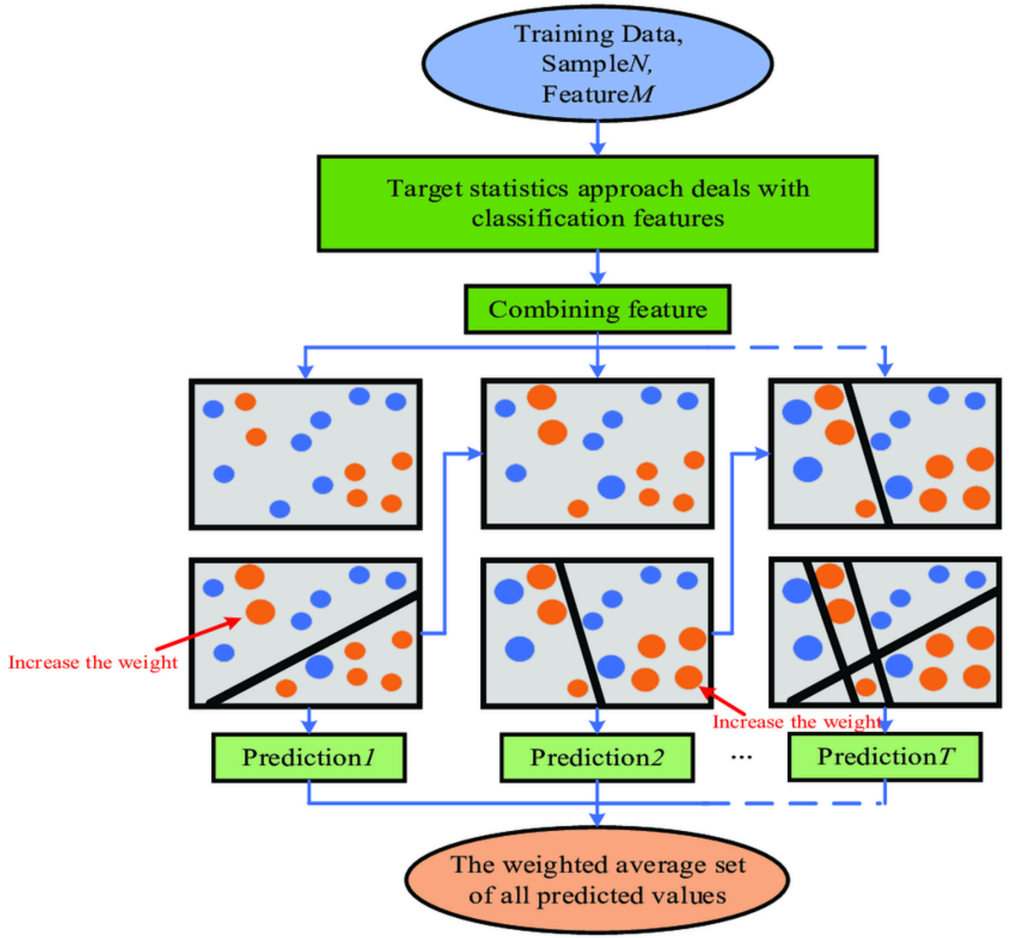
**Stratégie 1-** score public de 4852, 6ème meilleur score.

**Stratégie 2-** score public de 4876, 4ème meilleur score.

**Le score ne s'étant pas amélioré significativement, nous avons décidé de tester un nouveau type de modèle.**

# Modélisation

## 3. Catboost




- CatBoost et XGBoost sont deux bibliothèques puissantes pour le boosting d'arbres décisionnels, mais elles se **distinguent par leurs approches et leurs domaines d'excellence**.

- CatBoost est particulièrement adapté aux jeux de données contenant des variables catégoriques grâce à sa **gestion native de ces données**, évitant des étapes d'encodage manuel. En revanche, XGBoost nécessite un prétraitement, mais excelle sur les données numériques bien préparées. CatBoost est également **plus intuitif à utiliser "out of the box"**, avec des hyperparamètres par défaut efficaces, tandis que XGBoost offre plus de flexibilité, souvent au prix de plus de configuration. Enfin, CatBoost, optimisé pour les entraînements sur GPU, est **rapide à configurer** pour de grandes bases de données, tandis que XGBoost, est préféré pour ses performances sur des jeux de données complexes et entièrement numériques.

# Modélisation

## 3. Catboost

## Les hyperparamètres

Nous avons utilisés optuna pour la recherche des meilleurs hyperparamètres.

## Résultat

| | | |
|---|---|---|
| **Stratégie 1-** score public de 4861, 5ème meilleur score. | {'learning_rate': 0.01, 'l2_leaf_reg': 7, 'iterations': 300, 'depth': 6, 'border_count': 32} | 0,4861242510249133 |
| **Stratégie 2-** score public de 4888, 1er score. | 'learning_rate': 0.05, 'l2_leaf_reg': 15, 'iterations': 50, 'depth': 6, 'border_count': 128, 'loss_function': 'MultiClass', 'random_seed': 64, 'verbose': 0 | 0,48880479344055505 |
| **Stratégie 2-** score public de 4878, 3ème score. | {'iterations': 300, 'depth': 8, 'learning_rate': 0.02582812306743591, 'l2_leaf_reg': 5, 'border_count': 128} | 0,4878587196467991 |

# Axe d’amélioration

Dans le cadre de ce projet de machine learning, **plusieurs axes d'amélioration** peuvent être envisagés pour optimiser les performances et la robustesse du modèle.

Tout d’abord, il serait pertinent d’étudier plus en détail le **contexte footballistique** afin d’avoir une meilleure compréhension des variables qui apporteraient de meilleures performances à nos modèles.

Enfin, il serait intéressant de réaliser une **analyse approfondie des erreurs du modèle**, notamment en identifiant les sous-populations mal prédites, afin de mieux ajuster les données d’entraînement ou les algorithmes. L’ajout de nouvelles données ou l’enrichissement des jeux de données avec des sources externes pourrait également renforcer la qualité des prédictions.

# Retours d’éxpérience

## Ce que nous avons appris sur la préparation des données

- L’importance de la qualité des données : Nous avons appris que passer du temps sur le nettoyage et l’analyse des données est souvent plus important que de construire un modèle complexe.
- L’impact des features bien conçues : Les nouvelles variables dérivées ont souvent amélioré les performances plus que le changement de modèle.
- Nous avons également fait appel à des professionnels de statistiques de football afin d’obtenir de l’aide sur les variables qui peuvent le plus impactés la prédiction d’un match. Cela montre l’importance de collaborer avec des personnes plus qualifiés que nous sur certains points que nous ne maitrisons pas très bien. C’est un point qui est également applicable en entreprise.

## Ce que nous avons appris sur la modélisation

- Nous retenons qu’il ne faut pas se lancer au hasard sur le test d’un modèle: il est important de se fier aux tests que l’on a effectué au préalable lors de la préparation des données afin de cibler un type de modèle qui permettra d’obtenir de meilleures performances de prédiction.

# Conclusion

Au cours de ce projet, nous avons construit trois bases de données distinctes afin d'explorer différentes perspectives sur notre problématique et de maximiser les performances des modèles.

Sur chacune de ces bases, nous avons appliqué plusieurs algorithmes de machine learning, notamment XGBoost, Random Forest, et d'autres modèles complémentaires, pour évaluer leurs performances et leur capacité à généraliser.

Cette approche multi-dimensionnelle nous a permis de comparer les résultats en fonction des spécificités de chaque base, **d'identifier les forces et les limites des modèles testés**, et de mieux comprendre **les facteurs clés** qui influencent les prédictions. Cette méthodologie nous a offert une vue complète et robuste sur le problème, tout en permettant d'optimiser nos choix techniques pour obtenir les meilleurs résultats possibles.

Nous sommes satisfait de notre performance, mais nous avons beaucoup appris de ce challenge. Dans un premier temps, **la gestion de la base de donnée nous a paru impossible** (étant donné la grande quantité des données). Mais en appliquant vos conseils et en prenant le temps de bien cerner les données avec lesquelles on travaille, nous avons réussi à élaborer plusieurs bases de données en autonomie, chose que nous sommes fier. Nous avons également **beaucoup appris sur les modèles de machine learning**: au départ nous étions familier avec certains modèles mais cela restait très en surface. Puis au fur et à mesure, nous avons réussis à bien différencier les modèles que nous utilisions et à prendre en compte leurs points forts et leurs points faibles afin de **gagner de la précision sur nos prédictions.**

**Nous repartons de ce challenge avec la ferme envie d'améliorer notre score dans le futur, nous reviendrons sur nos bases de données lorsque l'on aura le temps.**