Analyse des tirs de joueurs NBA

07-07-2025

—

Robin EVANS

Hoang Cyril DINH-VU

Adam BELGAID

Tuteur : Antoine Fradin

Table des matières

Vue d'ensemble

Dans un contexte où la data science prend une place croissante dans le sport professionnel, la NBA est un terrain fertile pour l'analyse avancée. Grâce au tracking des joueurs et à la richesse des statistiques disponibles, les clubs s'appuient de plus en plus sur les données pour optimiser leurs performances.

Du point de vue **scientifique**, il s'agit d'un projet de modélisation probabiliste appliqué à la performance sportive.

Économiquement, les enjeux sont significatifs, les décisions issues de ces analyses pouvant influencer la stratégie d'acquisition ou de jeu des franchises NBA. Ces données peuvent également être pris en compte sur le prix des joueurs durant les drafts

Techniquement, nous disposons de jeux de données volumineux et variés (plusieurs centaines de milliers de tirs analysés).

Objectifs

1. Comparer l'efficacité des tirs par zone et situation de jeu
2. Focaliser sur 20 superstars encore actives (ex : Lebron,Curry, etc...)
3. Estimer la probabilité de réussite d'un tir

Données et périmètre d'étude.

1) NBA shot Locations

Données en libre accès sur KAGGLE. Nous avons 22 colonnes dont 12 de types numériques et 10 de type texte. 4,7 millions de tirs enregistrés en NBA, couvrant les saisons de 1997 à 2019.

Description du jeu de données:

- ☐ Identifiants de match et d'évènement (Game ID, Game Event ID)
- ☐ Informations sur le joueur (Player ID, Player Name)
- ☐ Équipe et adversaire (Team ID, Team Name, Home Team, Away Team)
- ☐ Contexte du tir (Period, Minutes Remaining, Seconds Remaining)
- ☐ Caractéristiques du tir (Action Type, Shot Type, Shot Zone Basic, Shot Zone Area, Shot Zone Range, Shot Distance, X Location, Y Location)
- ☐ Résultat du tir (Shot Made Flag)
- ☐ Date du match et type de saison (Game Date, Season Type)

Les données qui semblent pertinentes pour répondre à nos problématiques sont:

- ☐ **Shot Distance** : distance du tir par rapport au panier, élément fondamental pour estimer la difficulté d'un tir.
- ☐ **X Location / Y Location** : coordonnées spatiales du joueur lorsqu'il tire.
- ☐ **Shot Zone Basic / Area / Range** : classification du type de tir et sa localisation sur le terrain (par exemple : "Paint Area", "Mid-Range", "Left Corner").
- ☐ **Period, Minutes Remaining, Seconds Remaining :** contexte temporel du tir, qui peut influencer la pression ou le rythme du jeu.
- ☐ **Action Type / Shot Type** : nature du tir (jump shot...), un facteur technique essentiel.
- ☐ **Player Name :** identification du joueur permettant les comparaisons inter-joueurs.
- ☐ **Team Name, Home Team / Away Team** : domicile/extérieur qui peut influencer la performance.

Ce jeu de données nous offre une grande quantité de données avec beaucoup de variables liées aux positions sur le terrain à différents moments du match. Cependant, nous n'avons aucune information explicite sur la pression ou la proximité du défenseur, un facteur pourtant clé. et également pas d'indicateur de fatigue ou de forme physique du joueur à l'instant du tir.

2) NBA Players stats since 1950

Donnée en libre accès sur KAGGLE. Nous avons 3 jeux de données que nous devrons ensuite “merge”.

- ☐ Seasons_Stats.csv
  - ☐ Statistiques annuelles de joueurs NBA depuis 1950 : PTS, AST, PER, TS%, MP, FG%, etc.
  - ☐ Volume : 24 691 lignes × 53 colonnes
  - ☐ Utilité : enrichir le profil de performance global par saison et poste.

- ☐ Players.csv
  - ☐ Informations de carrière : nom, position, taille, poids, années de carrière, collège.
  - ☐ Volume : 4 550 lignes × 8 colonnes
  - ☐ Utilité : lier carrière et évolution du style de jeu, filtrer les joueurs actifs sur la période analysée.

- ☐ player_data.csv
  - ☐ Données biographiques : ville et état de naissance, année de naissance, taille, poids.
  - ☐ Volume : 3 922 lignes × 8 colonnes
  - ☐ Utilité : variables anthropométriques et d’environnement pour enrichir le modèle.

Former un jeu de données:

Pour cela , je merge les 3 jeux de données avec la colonne “Player”. Ensuite je filtre pour ne garder que les données après les années 2000.

Description du jeu de données:

- ☐ Player : nom complet du joueur
- ☐ Year, Age, Tm : saison, âge, equipe
- ☐ year_start, year_end : annees de carriere NBA
- ☐ - height_x / height_y : taille du joueur-
- ☐ weight_x / weight_y : poids du joueur

- ☐ birth_date : date de naissance
- ☐ birth_city, birth_state : lieu de naissance
- ☐ collage / college : université frequentée
- ☐ PTS, AST, REB, MP, G, GS… : Statistiques permettant l'analyse de performances d'un joueur

Unnamed: 0_x, Unnamed: 0_y , blanl et blank2 colonnes inutiles qui seront supprimées.

Les données qui semblent pertinentes pour répondre à nos problématiques sont:

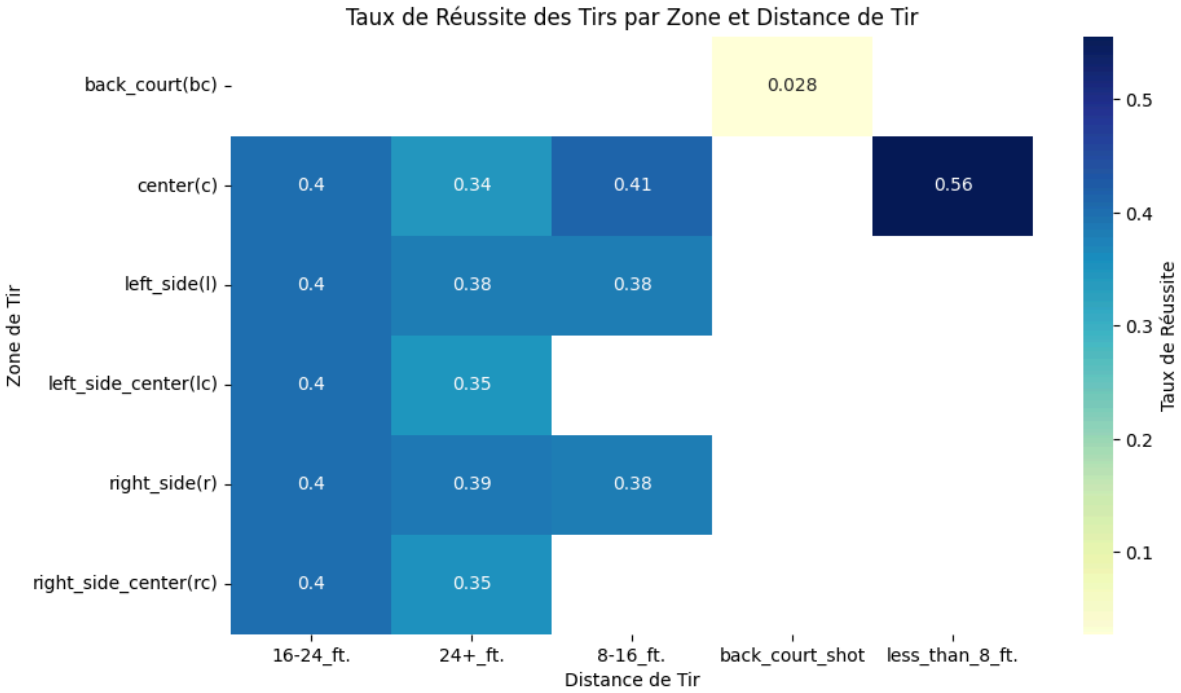
- ☐ Variables liées à la performance individuelle
  - PTS, AST, REB, MP, G : indicateurs du volume de jeu et de la régularité.
  - FG%, 3P%, FT%, TS% : efficacité au tir toutes zones confondues.
  - PER, VORP, BPM, WS : métriques avancées décrivant l'impact global d'un joueur.
- ☐ Variables liées au profil du joueur
  - Player, position : identification et rôle du joueur sur le terrain.
  - height, weight, Age : caractéristiques morphologiques et âge sportif.
  - year_start, year_end : expérience en NBA.
- ☐ **Variables contextuelles**
  - Tm, Year, G, MP : permettent d'analyser les performances par saison et équipe.

Nous n'avons pas de variables cibles sur ce jeu de données mais il serait intéressant d'incorporer les différentes statistiques sur le jeu de données vu précédemment.

Exploration de données

Dans ce premier graphique, nous allons observer que la probabilité de réussir ou non est proche. Nous n'aurons pas de rééquilibrer les données par les suite.

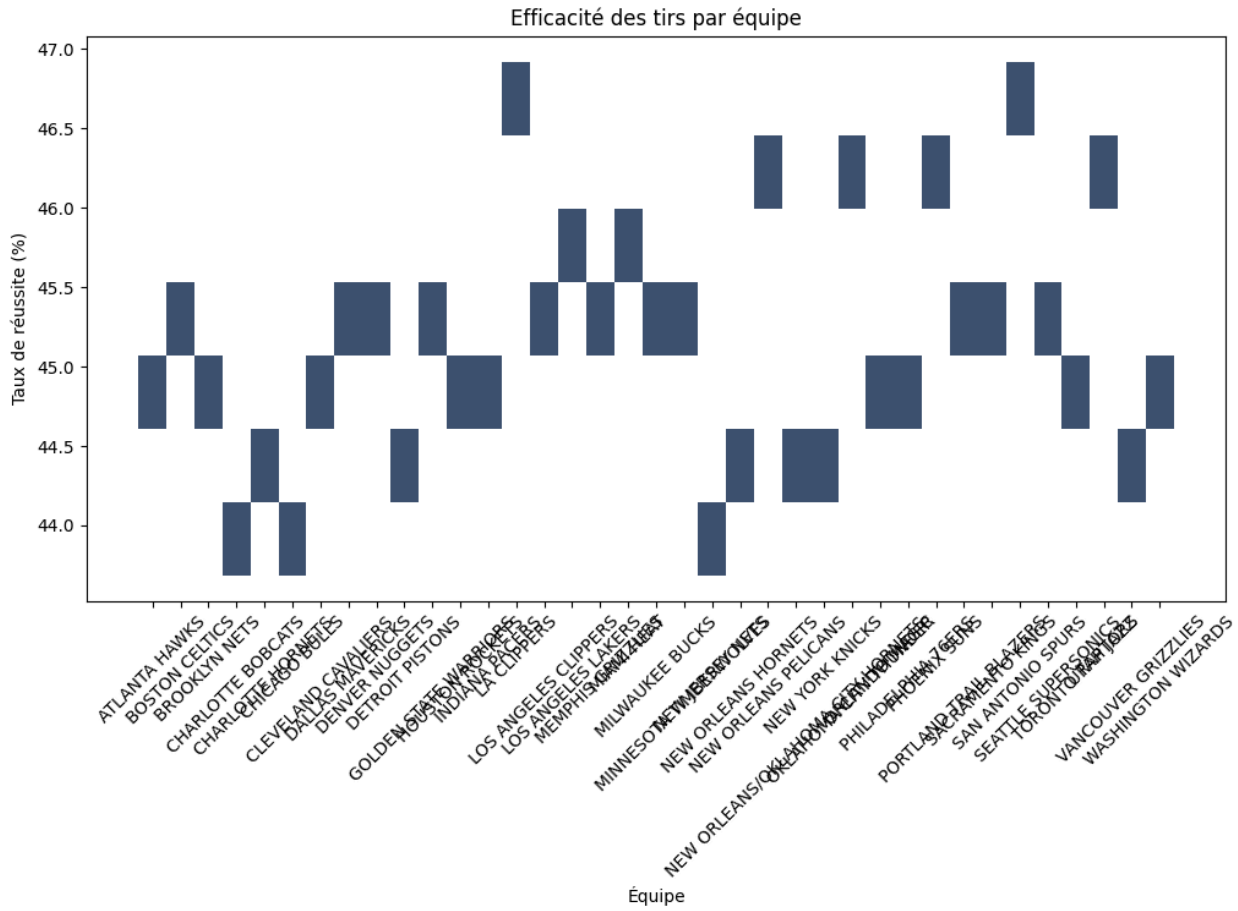
Nous pouvons remarquer que plus on s'éloigne du panier moins on a de chance de marquer. Entre 16-24ft , il n'y a aucune différence sur le nombre de paniers réussis selon la zone du terrain.



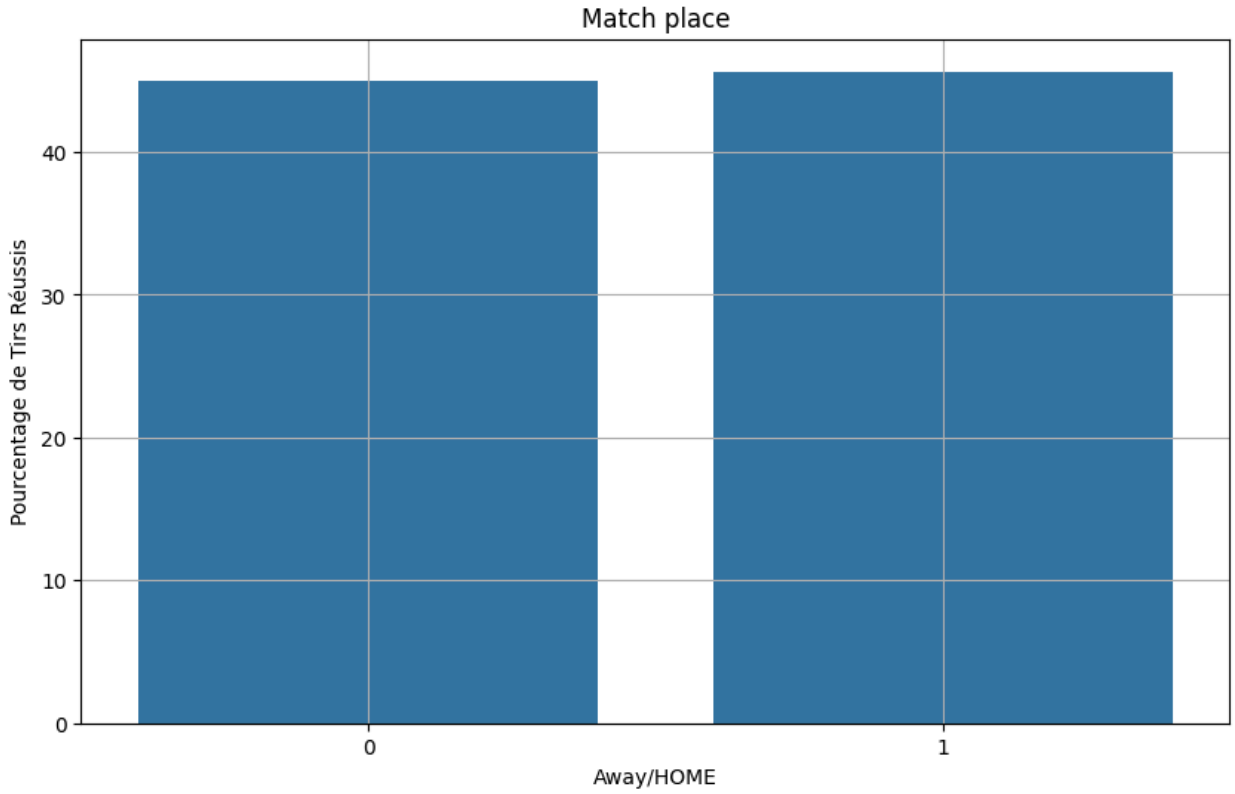
Le test de Pearson nous montre une corrélation négative significative entre la distance et le nombre de paniers marqués.

Nous avons effectué un test de khi2 également entre variables catégorielles comme “Shot Type” et “Shot Zone Area”. Il y’a une dépendance significative, on pourra les utiliser dans nos futures modélisations.

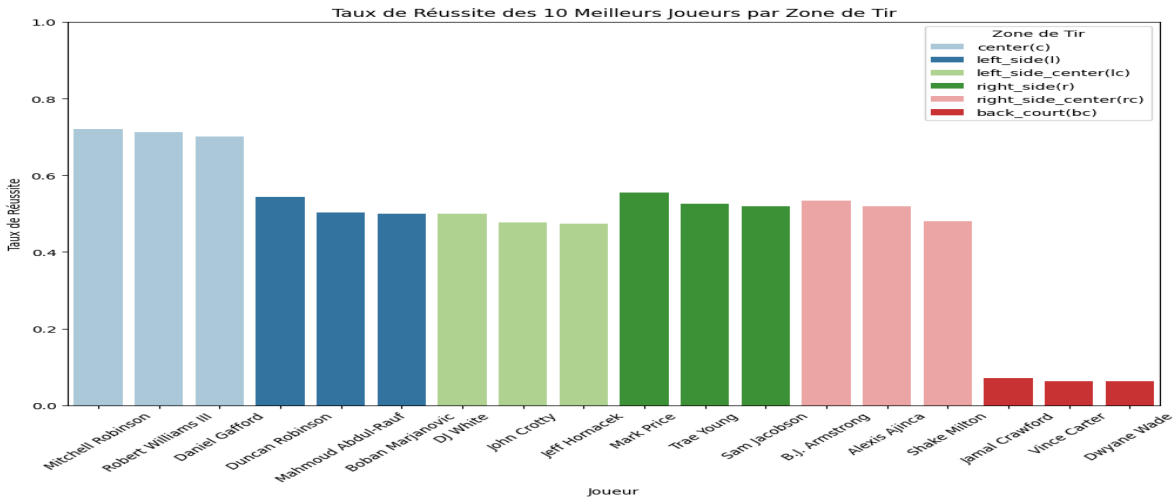
Les Golden State Warriors est l’équipe la plus efficace entre 1997 et 2019.



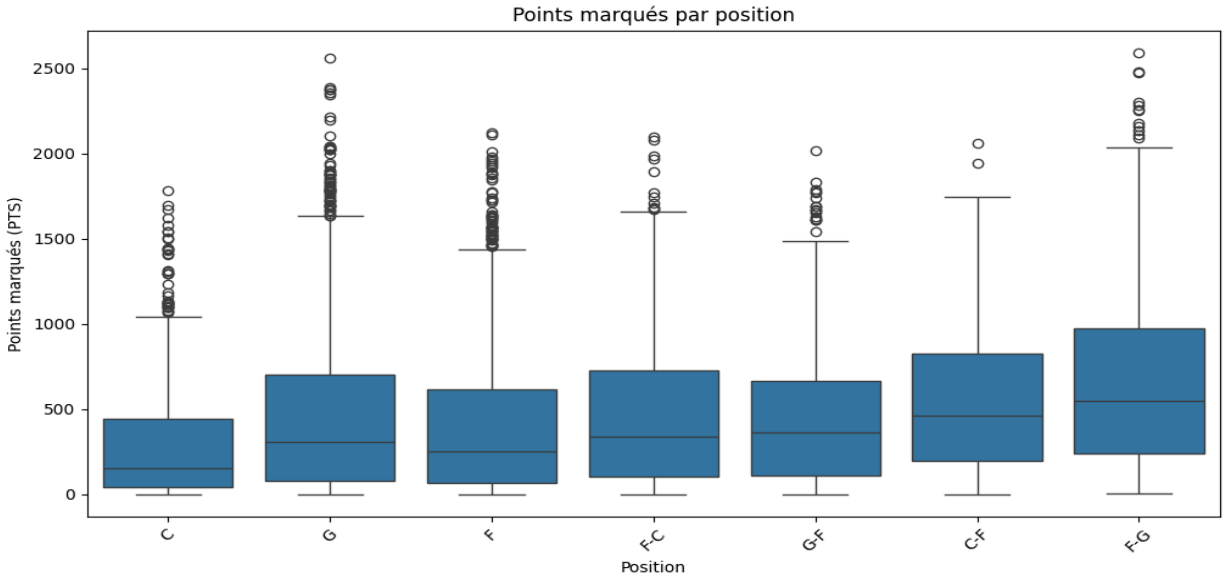
Globalement le fait de jouer à domicile ou à l’extérieur ne semble pas jouer dans le taux de réussite des paniers tentés.



Sur le prochain graphique , nous montrons les 3 meilleurs pour chaque zone de tirs. Nous avons mis un filtre avec un nombre minimum de 50 tirs tentée pour éviter de biaiser les résultats.

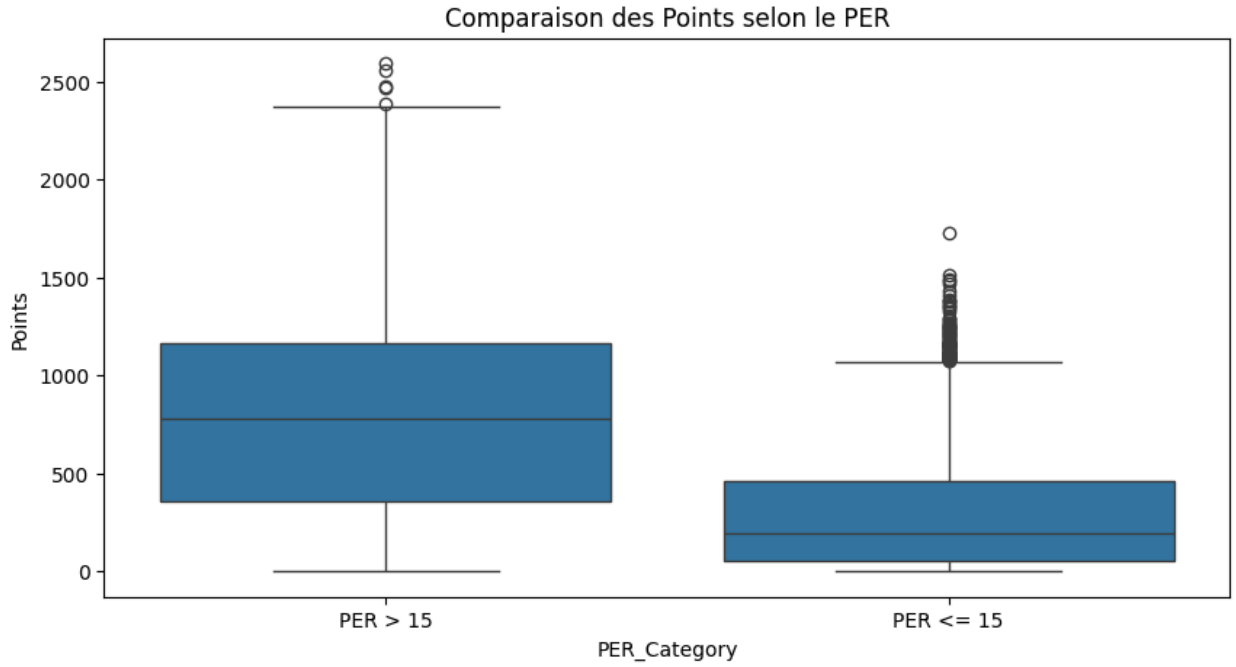


Les joueurs au poste F-G (c'est-à-dire qu'ils peuvent jouer ailier comme arrière) ont leurs médianes de points plus élevés que les autres postes. Les pivots de par leurs positionnements sur le terrain et leurs rôles dans le jeu ont moins l'occasion de tirer et donc de marquer.

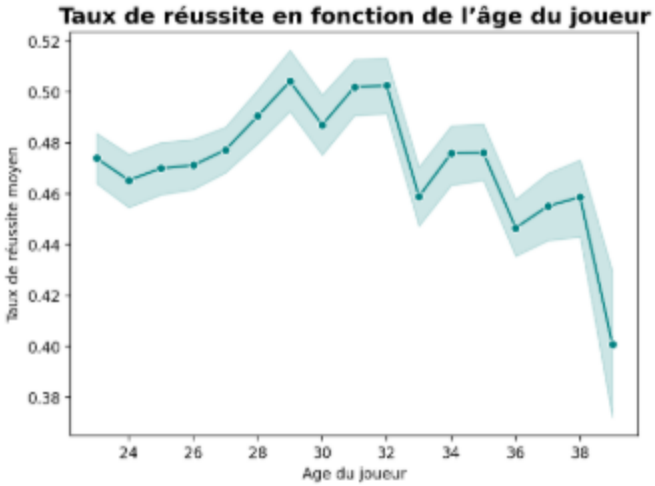


Les joueurs ayant un PER supérieur à 15 marquent significativement plus de points que ceux en dessous. Cela confirme que le PER est bien corrélé à la productivité offensive. (le choix de 15:

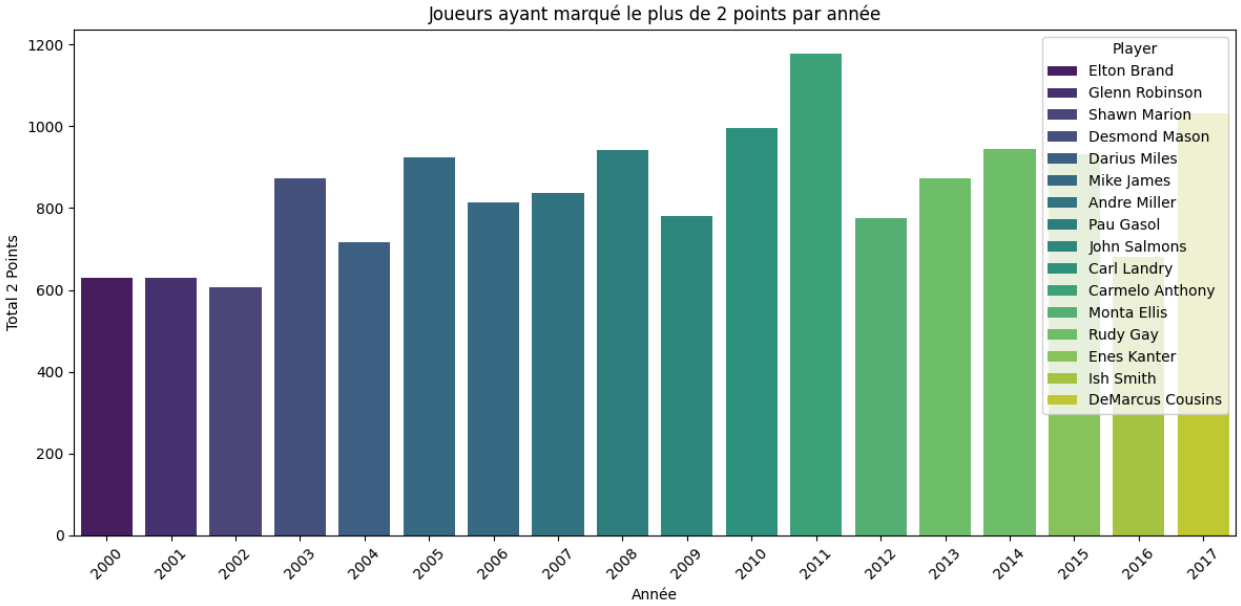
https://en-m-wikipedia-org.translate.goog/wiki/Player_efficiency_rating?_x_tr_sl=en&_x_tr_tl=fr&_x_tr_hl=fr&_x_tr_pto=rq)

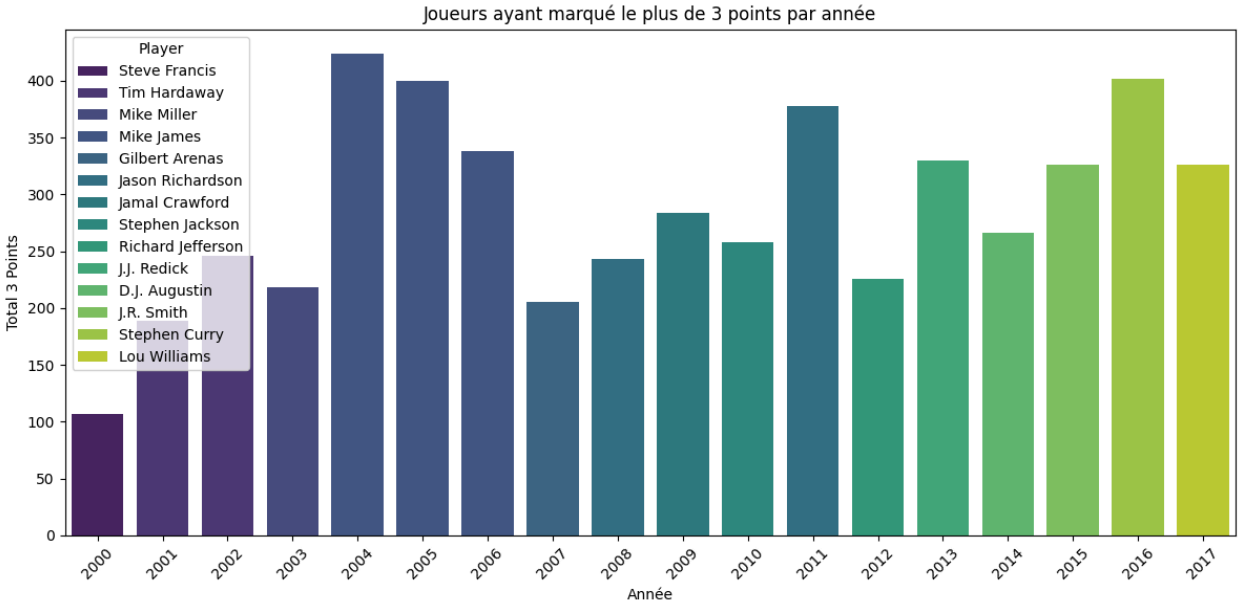


Dans le prochain graphique , nous pouvons observer que le pic de performances des joueurs est entre 24 et 28. Après 30ans , les joueurs commencent à marquer de moins en moins de points.

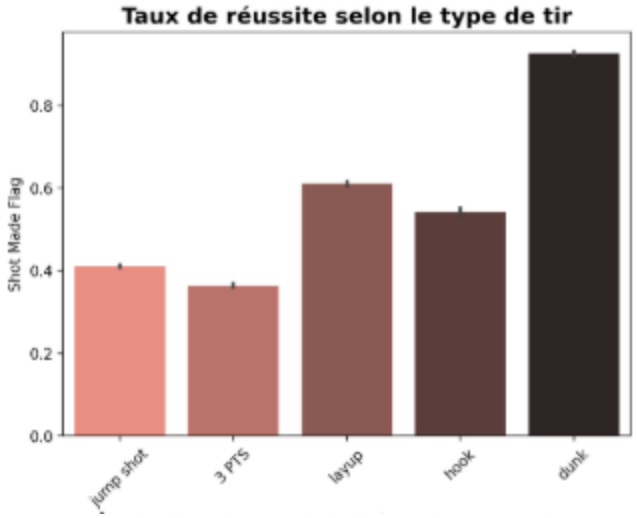


Par la suite, nous montrerons les joueurs qui ont marqué le plus de 2 pts et 3 pts par années.

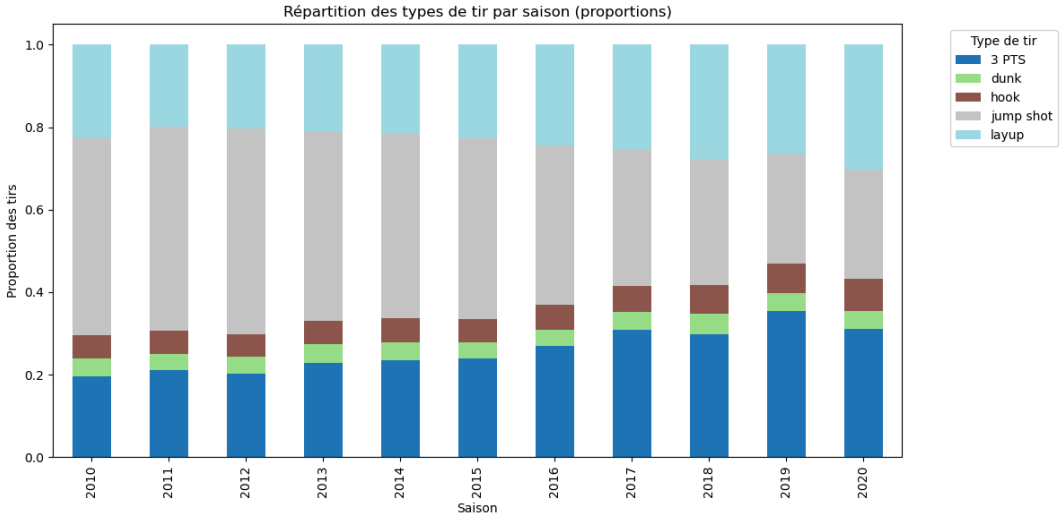




Les tirs proches du panier comme les layup , hook et surtout dunk ont de meilleurs taux de réussite. Les jump shot et 3 points sont des tirs plus difficiles (plus de distance, plus de contestation) , ce qui explique le taux de réussites plus bas 35-40%.



Le graphique ci-dessous montre l'évolution de jeu de la NBA qui historiquement était plus proche de la raquette, mais depuis quelques années , nous observons l'émergences de nouveaux joueurs comme Stephen Curry qui ont révolutionnés le style de jeu de la NBA avec une augmentation de paniers à 3 points .



Conclusions:

L'exploration des données de tirs NBA montre d'abord que la variable cible est relativement équilibrée : la proportion de tirs réussis et ratés est proche, ce qui nous évite de passer par des techniques de rééquilibrage et permet d'aborder la modélisation dans un cadre classique de classification binaire.

L'analyse de la distance et de la localisation confirme une intuition basket évidente : **plus on s'éloigne du panier, moins on a de chances de marquer**. Le test de corrélation de Pearson met en évidence une relation négative significative entre distance et probabilité de réussite, et l'on observe qu'entre 16 et 24 feet, les différences de réussite entre zones du terrain sont faibles, signe que la difficulté est avant tout liée à la distance. Les tests du khi^2 montrent par ailleurs une dépendance significative entre des variables catégorielles comme *Shot Type* et *Shot Zone Area*, ce qui justifie pleinement leur utilisation ultérieure comme variables explicatives.

Sur le plan collectif, les Golden State Warriors apparaissent comme l'équipe la plus efficace sur la période 1997–2019, tandis que l'effet du fait de jouer à domicile ou à l'extérieur reste limité sur le seul taux de réussite des tirs. À l'échelle individuelle, on observe des **profils de scoring différenciés** : les joueurs F–G (capables de jouer arrière et ailier) présentent des médianes de points plus élevées que les autres postes, alors que les pivots marquent moins, ce qui est cohérent avec leur rôle plus orienté vers l'intérieur et la protection du cercle.

Le lien entre PER et scoring est également confirmé : les joueurs avec un PER supérieur à 15 marquent significativement plus de points que les autres, ce qui valide l'utilisation de cet indicateur comme proxy de productivité offensive.

L'étude de l'âge met en évidence un **pic de performance entre 24 et 28 ans**, suivi d'un déclin après 30 ans, pattern classique mais important à objectiver dans les données. Enfin, l'analyse par type de tir montre que les tirs proches du cercle (layup, hook, dunk) affichent les meilleurs taux de réussite, alors que les jump shots et surtout les tirs à 3 points sont moins souvent convertis, bien qu'ils aient une valeur en points plus élevée. L'évolution temporelle souligne d'ailleurs le tournant pris par la NBA avec l'augmentation massive des tirs à 3 points ces dernières années, portée notamment par des joueurs comme Stephen Curry.

Transformations de données et Pre-processing

1. **Gestion des NA**

Données Players.csv

- ☐ Suppression des joueurs sans nom
- ☐ Nous avons retiré les doublons sur le nom de Joueur
- ☐ Conversion de la taille et poids en numérique → toute valeur invalide ou non convertible est transformée en NAN
- ☐ Filtrage des valeurs aberrantes: on conserve un intervalle réaliste et toute valeur sortant de cet intervalle sera supprimée.

Données Seasons_Stats.csv

- ☐ Suppression des lignes sans nom par joueur et des doublons sur le couple (Player, Year)
- ☐ Suppression des colonnes trop manquantes: les colonnes présentant plus de 40% de valeurs manquantes sont supprimées.
- ☐ Conversion des colonnes numériques: Les colonnes quantitatives sélectionnées (points, rebonds, pourcentages de tir, statistiques avancées, etc.) sont converties en numérique avec errors="coerce". Les valeurs non numériques ou incohérentes sont ainsi transformées en NaN. À ce stade, ces NaN ne sont pas supprimés : ils seront traités plus tard par l'imputation dans le pipeline.
- ☐ Normalisation des colonnes avec des pourcentages (ex: FG%, FT%..) si des valeurs dépassent 1 , elles seront divisées par 100.

Données NBA Shot Locations

- ☐ Suppression des lignes sans cible (Shot Made Flag) ni date (Game date)
- ☐ Conversion et filtrage des variables numériques: Ensuite, un filtrage par bornes plausibles est appliqué (par exemple, Seconds Remaining entre 0 et 59, Period entre 1 et 12, Shot Distance entre 0 et 45, etc.). * Si une valeur est en dehors de ces bornes, la ligne est supprimée. * Si la valeur est manquante (NaN), la ligne est conservée (s.between(lo, hi) | s.isna()), l'idée étant de laisser le pipeline imputer ces NA plutôt que de perdre de l'information.
- ☐ Variables Catégorielles avec valeurs par défaut.

Lors de la construction du jeu de données final, les informations de tirs sont enrichies avec les caractéristiques physiques et les statistiques de saison via des jointures sur Player et Year.

→ **Avant le pipeline**, la logique est de:

- ☐ supprimer ce qui est inutilisable (absence de clé, variable cible ou date, valeurs manifestement aberrantes)
- ☐ conserver les NA "raisonnables" pour qu'ils soient traités de manière systématique par le préprocesseur.

2. Création de variables

Géométrie de tir:

- ☐ Variable "dist": Calculée à partir de X Location et Y Location.
- ☐ Angle du tir "angle": Calculée à l'aide de arctan(X, Y) pour capturer l'orientation du tir. Cette variable permettra au modèle de différencier les zones du terrain

Contexte temporel et du match:

- ☐ Temps restant dans la période (time_left) : Combinaison de Minutes Remaining et Seconds Remaining en un nombre total de secondes. Cette variable représente la pression temporelle au moment du tir.
- ☐ Année de la saison (Year) : Extraite de Game Date. Elle permet de tenir compte de l'évolution du jeu NBA au fil du temps (augmentation des tirs à 3 points, changement de rythme, etc.).
- ☐ Avantage du terrain (is_home): permet de connaître si le fait de jouer à la maison donne un boost au joueur

3. Regroupement des types de tir

La colonne brute Action Type contient un grand nombre de libellés très détaillés (par exemple,Driving Floating Jump Shot, Running Finger Roll Layup Shot, etc.). Pour simplifier et rendre ces informations exploitables :

- ☐ Chaque libellé est associé à une catégorie plus générale via un mapping (par exemple jump shot, layup, dunk, hook, 3 PTS, etc.), stockée dans une nouvelle variable ACTION_CATEGORY.

- ☐ Les événements ne correspondant pas à un véritable tir sont repérés et retirés (no shot).
- ☐ La variable Action Type utilisée dans le modèle est remplacée par cette version catégorisée

4. Prétraitement global avant modélisation

Après avoir géré les NAs et créé des variables , nous allons construire un jeu de données unique. Nous allons enrichir le jeu de données Nba Shot Location afin d'obtenir , pour chaque tir, à la fois des informations sur le contexte du tir, la morphologie du joueur et son profil statistique sur la saison.

Avant jointure, nous avons harmonisé le nom du joueur dans la colonne Player. Du côté des statistiques de saison, le jeu Seasons_stat est nettoyé de manière à ne conserver qu'une seule ligne par couple (Player, Year). Les doublons éventuels sont supprimés, et les lignes sans nom de joueur sont écartées. Les colonnes inutiles ou trop incomplètes sont également retirées.

→ **Fusion des données des tirs avec les caractéristiques physiques des joueurs et statistiques**

Player_csv contient des informations biographiques et physiques sur les joueurs (notamment height et weight). Après nettoyage de ce fichier (suppression des joueurs sans nom, doublons sur Player, filtrage des tailles/poids aberrants), on obtient une base clean avec une ligne par joueur.

Une première fusion est alors réalisée entre les tirs et ce fichier de joueurs sur Player

Cette fusion permet d'enrichir chaque tir par les variables height et weight, qui sont ensuite utilisées comme variables explicatives dans le modèle.

Toujours via une jointure de type left join. On obtient une table qui associe, pour chaque joueur et saison présents dans les tirs, les principales statistiques de saison : points, passes décisives, rebonds, pourcentages de tir, minutes jouées, ainsi que des indicateurs avancés comme PER, VORP, WS.

Les tirs réalisés par des joueurs ou saisons absents de Seasobs_stats conservent des valeurs manquantes sur ces statistiques, qui seront gérées ultérieurement par l'imputation. L'important est que le grain de la table reste le même : une ligne par tir, mais avec désormais des colonnes supplémentaires décrivant la performance globale du joueur sur la saison.

→Pré-processing : séparation des variables et définition du préprocesseur

Une fois la base de données prête , l'objectif du pré processing est de construire un matrice d'apprentissage et de définir une pipeline permettant de transformer nos données en entrée du modèle.

Construction de la matrice d'apprentissage

Chaque ligne représente un tir avec:

- Des variables numériques: Shot Distance, X Location, Y Location, dist, angle, Minutes Remaining, Seconds Remaining, Period, ainsi que les statistiques de saison du joueur (Age, G, MP, PTS, AST, TRB, FG%, 3P%, FT%, WS, PER, VORP, etc.)
- Des variables catégorielles comme Season Type, is_home, Pos (poste), Tm (équipe).

La matrice d'entrée X est obtenue en sélectionnant l'ensemble des variables explicatives jugées pertinentes, tandis que y correspond à Shot Made Flag.

Découpage des données

Nous découpons notre jeu de données en trois sous ensembles

- un jeu d'entraînement (train), utilisé pour ajuster les paramètres des modèles ;
- un jeu de validation (valid), utilisé pour le choix des hyperparamètres et la comparaison de plusieurs modèles
- un jeu de test (test), conservé à part et utilisé uniquement pour l'évaluation finale.

Ce découpage se fait directement sur la matrice de tirs (une ligne = un tir), de sorte que les proportions de tirs réussis et ratés restent comparables entre les trois sous-ensembles. Le découpage est réalisé avant toute transformation (imputation, standardisation, encodage), pour éviter toute fuite d'information des ensembles de validation et de test vers l'ensemble d'entraînement.

Séparation numériques/ catégorielles

Nous séparons les variables explicatives en deux familles:

- Variables numériques
- Variables catégorielles

Préprocesseur: Column Transformer

```
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder

pre = ColumnTransformer(
    transformers=[
        ("num", Pipeline([
            ("impute", SimpleImputer(strategy="median")),
            ("scale", StandardScaler())
        ]), num_cols),
        ("cat", Pipeline([
            ("impute", SimpleImputer(strategy="most_frequent")),
            ("onehot", OneHotEncoder(
                handle_unknown="ignore",
                sparse_output=True,
                min_frequency=100
            )),
        ]), cat_cols),
    ],
    remainder="drop",
    verbose_feature_names_out=False
)
```

Traitement des variables numériques:

- ☐ Imputation par la médiane : SimpleImputer(strategy="median") remplace les valeurs manquantes dans chaque variable numérique par la médiane calculée sur le jeu d'entraînement. La médiane est robuste aux valeurs extrêmes et convient bien aux

distributions asymétriques, fréquentes en statistiques de jeu (minutes, points, rebonds, etc.).

- ☐ Standardisation : StandardScaler() recentre chaque variable (moyenne 0) et la met à l'échelle (variance 1). Cela place toutes les variables numériques sur des échelles comparables. Les paramètres de standardisation (moyenne et écart-type) sont appris uniquement sur le jeu d'entraînement, puis réutilisés pour transformer validation et test.

Traitement des variables catégorielles:

- ☐ Imputation par la modalité la plus fréquente: SimpleImputer (strategy="most_frequent") remplace les valeurs manquantes dans chaque variable catégorielle (par exemple Season Type, Pos, Tm) par la modalité la plus fréquente observée sur le jeu d'entraînement. Cette stratégie permet de conserver les observations même si certaines informations ne sont pas renseignées (poste inconnu, saison type manquante, etc.).

- ☐ Encodage One-Hot avec filtrage des catégories rare: OneHotEncoder(handle_unknown="ignore",sparse_output=True, min_frequency=100) transforme chaque variable catégorielle en un ensemble de colonnes binaires (0/1), une par modalité suffisamment fréquente. L'option handle_unknown="ignore" permet d'ignorer sans erreur les catégories inconnues apparaissant dans validation ou test mais absentes dans l'entraînement. Le paramètre min_frequency=100 fusionne implicitement les catégories trop rares, ce qui limite la dimension du jeu de données après encodage, évite que le modèle ne surapprenne sur des modalités très peu observées.

Nous avons maintenant un préprocesseur défini , nous pouvons l'intégrer dans un pipeline avec un modèle de classification choisi. Toutes les étapes ci-dessus seront toujours appliquées de manière identique à l'entraînement et à la prédiction.

Modélisation de la probabilité de réussite d'un tir

1. **Choix des modèles:**

L'objectif étant de prédire la probabilité de réussite d'un tir à partir de certains caractéristiques liées :

- ☐ au joueur comme la morphologie, le PER,..
- ☐ au tir lui-même comme la distance , le type de tir...
- ☐ au match comme le type de saison , le temps restant..

La variable cible est binaire → problème de classification

Il nous faut donc un modèle capable de;

- de produire une probabilité
- d'interpréter les variables qui influencent cette probabilité
- capturer mes mes relations non linéaires et interactions complexes entre variables

Pour cela nous avons décidé de choisir deux modèles:

- ☐ **Régression logistique**: modèle linéaire de référence
- ☐ **Modèle XGBoost** qui permet de capturer des relations complexes et rechercher une meilleur performance prédictive

2. Rappel fonctionnement des modèles

a. Classification par régression logistique

"La régression logistique est un modèle statistique permettant d'étudier les relations entre un ensemble de variables qualitatives Xi et une variable qualitative Y .

Il s'agit d'un modèle linéaire généralisé utilisant une fonction logistique comme fonction de lien.

Un modèle de régression logistique permet aussi de prédire la probabilité qu'un événement arrive (valeur de 1) ou non (valeur de 0) à partir de l'optimisation des coefficients de régression. Ce résultat varie toujours entre 0 et 1. Lorsque la valeur prédite est supérieure à un seuil, l'événement est susceptible de se produire, alors que lorsque cette valeur est inférieure au même seuil, il ne l'est pas.

$$y = \begin{cases} 1 & si \quad h_\theta(X) \geqslant seuil \\ 0 & si \quad h_\theta(X) < seuil \end{cases}$$

Pour éviter le problème de **sur-apprentissage** (*overfitting*), phénomène courant en apprentissage automatique (lorsque le modèle s'ajuste trop précisément aux données d'entraînement, mais peine à généraliser correctement), une *régularisation* est effectuée sur la fonction. La *régularisation* consiste à modifier la fonction de coût pour y intégrer une fonction dépendante uniquement des paramètres libres du modèle.

Trouver la fonction logistique optimale revient donc à optimiser la fonction de coût suivante selon le paramètre multi-dimensionnel θ:"

$$J(\theta) = -\frac{1}{m}.\left\{\sum_{i=0}^{m}\left\{y_i.log(h_\theta(x^{(i)})) + (1-y_i).log(1-h_\theta(x^{(i)}))\right\} + \lambda.\sum_{j=1}^{m}\theta_j^2\right\}$$

$$\text{où } \lambda = \frac{1}{C} : \text{le paramètre de régularisation}$$

Limites de ce modèle:

La régression logistique reste un modèle linéaire dans l'espace des variables après encodage. Cela signifie que :

- ☐ les interactions complexes entre variables ne sont pas capturées automatiquement (par exemple, l'effet de la distance pourrait dépendre du joueur, de l'angle ou du type de tir) ;

- ☐ les relations non linéaires (par exemple une probabilité qui baisse fortement après une certaine distance puis se stabilise) ne sont pas modélisées, sauf à créer explicitement des termes quadratiques ou des interactions, ce qui complexifie fortement le modèle.

Pour dépasser ces limites et capturer des structures plus riches dans les données, un modèle non linéaire a été ajouté : XGBoost.

b. XGBoost

“Le Gradient Boosting est une généralisation du Boosting dans lequel on utilise la fonction de perte de manière similaire à une descente de gradient.

Par design, cette technique de boosting utilise de manière sous-jacente des arbres de décision.
L'idée principale est d'agréger plusieurs modèles créés itérativement, mais aussi d'accorder un poids différent à chacun d'entre eux.

L'approche suivante explique le raisonnement utilisé dans la conception d'un GBT :

- ☐ Prendre une pondération aléatoire (poids w_i) pour les classificateurs faibles (paramètres a_i) et former un classifieur final.
- ☐ Calculer l'erreur induite par ce classifieur final, et chercher le classifieur faible qui s'approche le plus de cette erreur.
- ☐ Retrancher ce classifieur faible du classifieur final tout en optimisant son poids par rapport à une fonction de perte.
- ☐ Répéter le procédé itérativement.

La procédure de *gradient boosting* consiste donc à trouver les poids qui optimisent la fonction de coût relative au problème de classification.
Il s'agit donc d'explorer un espace de fonctions simples par une descente de gradient.

XGBoost utilise une variation de la descente de gradient appelée descente de gradient fonctionnelle, puisqu’on travaille dans un espace fonctionnel lorsqu'on parle des classifieurs de l'algorithme du boosting. Le gradient de la fonction de perte sert au calcul des poids des individus lors de la construction de chaque nouveau modèle.

XGBoost propose d'enregistrer les matrices ou les modèles construits et de les recharger plus tard, pour éviter de relancer les mêmes calculs coûteux plusieurs fois.”

La régression logistique servira donc de baseline interprétable permettant de comprendre certaines tendances globales et d'avoir un modèle simple. Le XGBoost va aider à maximiser la performance prédictive au prix d'une interprétabilité plus limitée. On analysera l'importance des variables à l'aide de SHAP.

3. Premières modélisations sur l'ensemble du jeux de données

Le jeu de données initial couvre un très grand nombre d'observations. Pour limiter le temps de calcul et le coût sur la mémoire, on utilisera un échantillon de 300K tirs qui dispose déjà d'un grand nombre de combinaisons de contextes et d'un échantillon suffisamment large pour stabiliser les estimations de probabilité et limiter le risque de surajustement aux spécificités d'une petite portion du jeu de données.

Fin de pipeline:

La dernière étape du pipeline consiste à entraîner les modèles, calibrer leurs probabilités et évaluer leurs performances.

Les modèles sont encapsulés dans des pipelines scikit-learn de la forme :

- pour la régression logistique :
  Pipeline([("pre", pre), ("clf", LogisticRegression(...))])

- pour XGBoost :
  Pipeline([("pre", pre), ("clf", XGBClassifier(...))])

où pre désigne le ColumnTransformer appliquant le pré-processing décrit auparavant.

L'apprentissage se fait dans un premier temps sur le jeu d'entraînement, tandis qu'un jeu de validation distinct est utilisé pour la calibration des probabilités et, dans le cas de XGBoost, pour la recherche d'hyperparamètres.

Afin d'obtenir des probabilités interprétables, chaque modèle est ensuite calibré à l'aide d'un CalibratedClassifierCV avec une régression isotone (method="isotonic"). Concrètement, le modèle de base (régression logistique ou XGBoost) est d'abord ajusté sur le jeu d'entraînement, puis une fonction monotone est apprise sur le jeu de validation pour recalibrer les probabilités prédites. Le modèle final est donc un pipeline complet : pré-processing → modèle brut → calibration des probabilités.

Résultat des modèles :

Régression logistique:

En ajoutant une étape de calibration isotone sur le jeu de validation, les résultats sur test deviennent approximativement :

- AUC ≈ 0,645
- Brier score ≈ 0,225
- LogLoss ≈ 0,637
- ECE ≈ 0,014

La capacité de discrimination (AUC) reste quasiment inchangée, ce qui est attendu : la calibration ajuste les probabilités sans modifier l'ordre relatif des prédictions. En revanche, le Brier score, la log-loss et surtout l'ECE s'améliorent, ce qui signifie que les probabilités produites sont mieux alignées avec la réalité. Autrement dit, lorsqu'un tir est annoncé à 0,7 de probabilité, cette affirmation devient statistiquement fiable sur ce type de situations.

XGBoost:

Après une phase de recherche d'hyper paramètres (nombre d'arbres, profondeur, taux d'apprentissage, etc.) via RandomizedSearchCV et TimeSeriesSplit, puis une calibration, le modèle XGBoost calibré atteint, sur le jeu de test, des valeurs typiques de :

- AUC ≈ 0,67
- Brier score ≈ 0,218
- LogLoss ≈ 0,623
- ECE ≈ 0,015

Le modèle XGBoost surpasse la régression logistique sur l'ensemble des indicateurs de performance : meilleure séparation entre tirs réussis et ratés (AUC plus élevée) et meilleures probabilités (Brier et LogLoss plus faibles), tout en conservant une calibration comparable à celle de la logistique calibrée (ECE très faible). Cela confirme l'intérêt d'un

modèle non linéaire capable de capturer des interactions complexes entre les variables (distance × type de tir × joueur × contexte temporel, etc.)

En résumé, la régression logistique constitue un modèle de référence interprétable, tandis que XGBoost calibré représente le modèle le plus performant pour ce problème, lorsqu'on cherche à optimiser à la fois la discrimination et la qualité des probabilités prédites.

Par la suite, nous construirons nos modèles à partir des années 2010. Nous pouvons le voir sur le graphique sur les types de paniers marqués , une augmentation des tirs à 3 points. Ce qui montre un changement de jeu et de stratégies sur la NBA

De plus, pour répondre aux objectifs du projet cité précédemment, nous allons concentrer notre jeu de données sur les TOP players du 21ème siècle selon l'ESPN.

Analyse des meilleurs joueurs NBA du 21ème siècle

1. Contexte

D'après le classement établi par ESPN, les meilleurs joueurs du 21ème siècle sont les suivants :

1. Le Bron James	6. Kevin Garnett	11. Giannis Antetokounmpo	16. Kawhi Leonard	21. Tony Parker
2. Kobe Bryant	7. Nikola Jokic	12. Steve Nash	17. Manu Ginobili	22. Draymond Green
3. Stephen Curry	8. Dwyane Wade	13. James Harden	18. Allen Iverson	23. Russell Westbrook
4. Tim Duncan	9. Kevin Durant	14. Jason Kidd	19. Anthony Davis	24. Pau Gasol
5. Shaquille O'Neil	10. Dirk Nowitzki	15. Chris Paul	20. Ray Allen	25. Luka Doncic

Afin de comparer le tir de ces derniers, nous avons accès aux informations sur les tirs effectués entre 2000 et 2020.

Pour la modélisation du tir des meilleurs joueurs du 21ème siècle, nous n'allons pas exploiter les données entre 1950 et 2000. En effet, d'après l'analyse exploratoire, nous constatons que la proportion des tirs selon le type du tir a évolué au fil du temps, laissant suggérer que le style de jeu en NBA a évolué.

De plus, d'après nos recherches sur le contexte de la NBA, de nouvelles règles ont été mises en place à partir 2001 et ont eu un impact majeur sur le style de jeu de la NBA :

- ☐ Suppression de la règle de la "défense illégale" et autorisant les défenses élaborées comme les défenses en zone. Cette règle avait pour but de réduire les attaques isolées autour d'un seul scoreur.
  - ☐ Face aux nouvelles défenses en zone, les tirs tentés loin du panier ont progressivement augmenté
- ☐ Instauration de la règle des trois secondes en défense qui interdit à un défenseur de rester inactif dans la raquette plus de trois secondes
  - ☐ Les joueurs intérieurs sont devenus plus mobiles

Ainsi, pour modéliser le tir des meilleurs joueurs du 21ème siècle, nous avons considéré qu'utiliser les données des saisons antérieures à 2000 n'était pas pertinent, car le contexte et les règles de jeu étaient différents.

L'objectif étant de prédire le tir des joueurs encore en activité en 2020, nous avons fait le choix de nous focaliser sur les données disponibles des saisons 2010 à 2020, afin de rester au plus proche de la réalité. La taille des observations sur cette période nous semble significative :

- ☐ Nombre d'observations, tous joueurs confondus : 2 111 648 tirs
- ☐ Nombre d'observations sur les Top Joueurs : 187 564 tirs

Pour information, les joueurs encore actifs en 2020 sont les suivants :

1. Le Bron James	4. Kevin Durant	7. Chris Paul	10. Russell Westbrook
2. Stephen Curry	5. Giannis Antetokounmpo	8. Anthony Davis	11. Luka Doncic
3. Nikola Jokic	6. James Harden	9. Draymond Green	

2. Comparaison des tirs effectués entre 2010 et 2020

Tout d'abord, nous avons commencé par comparer les taux de réussite moyens des Top Joueurs par rapport à l'ensemble des joueurs :

	Tous joueurs confondus		Les Top Players (selon ESPN)	
	Nombre moyen de tirs tentés	Taux de réussite moyen	Nombre moyen de tirs tentés	Taux de réussite moyen
2010	91	48,28%	43	45,63%
2011	169	47,68%	69	45,85%
2012	244	47,78%	94	44,80%
2013	207	47,70%	93	45,40%
2014	206	48,42%	93	45,60%
2015	219	46,92%	93	44,69%
2016	230	47,45%	90	45,46%
2017	239	48,63%	88	46,07%
2018	221	48,14%	85	46,09%
2019	216	47,45%	85	45,86%
2020	89	49,48%	39	46,63%

En moyenne, nous constatons qu'un Top Joueur tente x2,4 plus de tirs qu'un joueur moyen et qu'il a un taux de réussite plus élevé (2,35 points de plus en moyenne).

Entre un Top Joueur et un joueur moyen, bien que l'écart sur le taux de réussite moyen ne soit pas considérable, nous constatons qu'un Top Joueur tente plus de tir, ce qui reflète l'importance de son rôle au sein d'une équipe.

Remarque : Au-delà du volume de tir et du taux de réussite, un autre facteur qui pourrait permettre de qualifier un joueur de "Top Player", c'est sa capacité à être décisif dans les moments importants (tir dans les derniers instants, dans les matchs importants, etc). Malheureusement dans notre dataset, nous n'avons pas tous les éléments contextuels des tirs.

3. Comparaison des tirs effectués parmi les Top Players

Ensuite, nous allons comparer les tirs effectués au sein des Top Joueurs :

	Entre 2010 et 2020	
	Nombre moyen de tirs tentés par saison	Taux de réussite moyen
Anthony Davis	207	51,75%
Chris Paul	213	47,13%
Dirk Nowitzki	179	46,63%
Draymond Green	125	43,90%
Dwyane Wade	214	47,64%
Giannis Antetokounmpo	195	52,43%
James Harden	274	44,14%
Jason Kidd	94	36,43%
Kawhi Leonard	175	49,39%
Kevin Durant	283	50,03%
Kevin Garnett	115	49,52%
Kobe Bryant	198	42,62%

	Entre 2010 et 2020	
	Nombre moyen de tirs tentés par saison	Taux de réussite moyen
LeBron James	304	52,41%
Luka Doncic	152	44,70%
Manu Ginobili	116	43,86%
Nikola Jokic	163	52,44%
Pau Gasol	154	48,70%
Ray Allen	130	45,91%
Russell Westbrook	294	43,79%
Shaquille O'Neal	25	66,50%
Stephen Curry	238	47,58%
Steve Nash	98	49,92%
Tim Duncan	162	49,92%
Tony Parker	204	48,78%

Globalement, nous constatons par saison, une moyenne de 180 tirs tentés pour un taux de réussite de 48,17%

Toutefois, au sein des Top Joueurs, nous constatons des statistiques différentes :

- Des joueurs tentent, en moyenne, 300 tirs par saison : LeBron James, Russell Westbrook, Kévin Durant ou James Harden.
- D'autres joueurs tentent moins de 100 tirs par saison :
  - Les meneurs "old school" : Jason Kidd et Steve Nash sont deux meneurs de joueurs qui ne sont pas connus pour leur volume de tirs, mais pour leur capacité à faire briller leur coéquipier. Remarque : Bien que le sujet du projet soit l'analyse des tirs, s'il fallait analyser un Top Joueur, nous aurions ajouté une variable "Nombre de passes décisives effectuées".
  - Les fins de carrières : Shaquille O'Neal et Kévin Garnett ont également un nombre de tirs tentés faible comparé aux autres Top Joueurs. Cela s'explique par le fait qu'ils sont en fin de carrière sur la période observée 2010-2020. Shaquille O'Neal entame sa dernière saison en 2011, et Kévin Garnett en 2016. Donc, ils ont un rôle moins important.

A présent, si nous nous focalisons sur le type de tirs en prenant des exemples avec Stephen Curry et Anthony Davis qui jouent à deux postes différents. Nous constatons les chiffres suivants :

	Répartition des tirs effectués	
Action Type	**Stephen Curry**	**Anthony Davis**
3 Points	48,3%	8,5%
Jump Shot	29,7%	47,5%
Layup	18,9%	22,6%
Hook	2,9%	6,9%
Dunk	0,2%	14,5%

Bien que Anthony Davis a un taux de réussite moyen, tous types de tir confondus, supérieur à Stephen Curry (51,75% versus 47,58%), nous ne pouvons pas directement comparer ces joueurs car ils ont un style de jeu très différents :

- ☐ Stephen Curry est un joueur spécialiste des tirs à longues distances. Quasiment la moitié de ses tirs sont effectués derrière la ligne des 3 points. Il ne va presque jamais au Dunk.
- ☐ Anthony Davis est un joueur Pivot/Ailier Fort qui va être plus proche du panier et il va tenter très peu de tirs à 3 points.

En résumé, en modélisant les tirs sur la base des données issues des Top Players, nous allons entraîner un modèle sur un ensemble de joueurs hétérogènes, avec des styles différents :

- ☐ des joueurs avec des volumes de tirs considérables,
- ☐ des joueurs de type Meneur/Passeur avec moins de tirs tentés,
- ☐ des joueurs qui tirent à longue distance,
- ☐ des joueurs qui tirent proche du panier.

Modélisation du tir sur la base des données 2010 à 2020

Nous allons effectuer nos modélisations sur la base des données 2010 à 2020. Les étapes de traitement restent identiques à celles détaillées aux paragraphes “Transformations de données et Pré-processing”.

La seule différence est que nous avons supprimé la limitation de 300 K observations → en effet, en terme de temps de calcul, le nombre d'observations entre 2010 et 2020 n’est pas aussi contraignant que le nombre d'observations entre 1950 et 2020.

La séparation des ensembles de données a été effectuée de la manière suivante :

- ☐ Ensemble d’entraînement : saisons 2010 à 2016
- ☐ Ensemble de validation : saisons 2016 à 2018
- ☐ Ensemble de test : saison 2018 à 2020

1. Modèle entraîné sur les données des Top Players entre 2010 et 2020

Pour la modélisation des tirs des meilleurs joueurs du 21ème siècle, nous avons décidé d’utiliser uniquement les données de ces derniers entre les périodes 2010 et 2020. En effet, il s’agit des meilleurs joueurs NBA, c’est-à-dire qu’ils se distinguent par leurs qualités exceptionnelles et leurs performances à travers le temps. Ils ont des spécificités uniques et ne peuvent pas être comparés à des joueurs standards. Ainsi, nous n’avons pas utilisé les données disponibles sur tous les joueurs, mais uniquement les données relatives aux meilleurs joueurs NBA du 21ème siècle.

Vous trouverez ci-dessous les performances de nos modèles après recherche des hyperparamètres :

Régression logistique:

- AUC ≈ 0,642
- Brier score ≈ 0,233
- LogLoss ≈ 0,658

XGBoost:

- AUC ≈ 0,663
- Brier score ≈ 0,226
- LogLoss ≈ 0,641

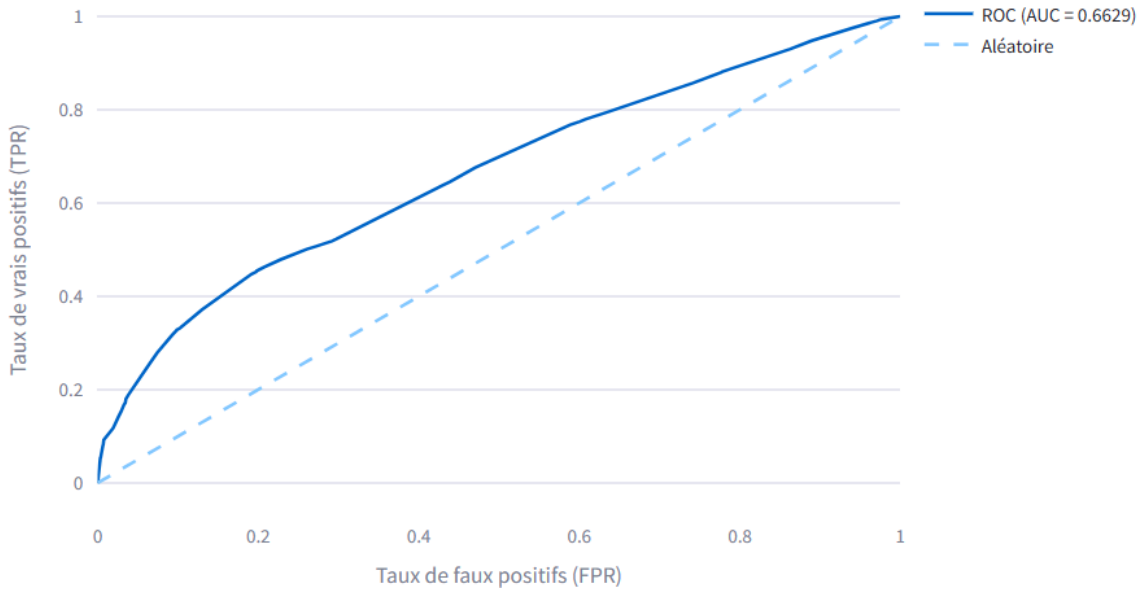
Lorsque nous observons la métrique AUC, nous constatons qu'il n'y a pas une grande différence de performance entre la Régression Logistique et XGBoost. Avec un AUC égal à 0.66, cela signifie que le modèle classe correctement les tirs dans 66% des cas. Le modèle fait mieux que le hasard, sans être très performant.

Pour rappel, l'AUC (Area Under the ROC Curve) est l'aire sous la courbe ROC ((Receiver Operating Characteristic) qui représente pour "tous les seuils possibles la relation entre qui donne le taux de vrais positifs, la sensibilité, en fonction du taux de faux positifs, l'antispécificité (= 1 - spécificité). Chaque valeur de seuil de classification fournira un point de la courbe ROC, qui ira de (0, 0) à (1, 1).

Plus la courbe se rapproche du point (0,1) (en haut à gauche), meilleures sont les prédictions. Un modèle avec une sensibilité et une spécificité égales à 1 est considéré comme parfait.

L'**aire sous la courbe** (AUC : Area Under the Curve) est très utile. En un seul nombre elle résume la capacité du modèle à distinguer la classe négative de la classe positive (Tir raté / Tir réussi).

Un score AUC de 0.5 signifie que le modèle n'est pas meilleur qu'une classification aléatoire, un score AUC de 1.0 signifie un modèle parfaitement prédictif, et un AUC de 0.0 est parfaitement anti-prédictif (très rare)."



Vous trouverez ci-dessous une comparaison entre les taux de réussite observés sur l'échantillon test et les probabilités de tirs prédites par les modèles, et par type de tir :

Action Type	y_test	proba_logit	proba_xgboost
3 Points	34,9%	38,3%	37,9%
Dunk	90,1%	93,9%	89,1%
Hook	48,6%	58,9%	53,5%
Jump Shot	43,1%	42,7%	41,5%
Layup	59,5%	63,5%	58,0%

Nous constatons que les prédictions faites par le modèle XGBoost sont plus proches de la réalité.

Vous trouverez ci-dessous quelques exemples par joueur :

	Action Type	y_test	proba_xgboost
Anthony Davis	3 Points	33,4%	37,8%
	Dunk	91,0%	89,2%
	Hook	39,2%	52,7%
	Jump Shot	40,0%	41,4%
	Layup	57,5%	57,8%
Lebron James	3 Points	34,6%	37,9%
	Dunk	95,7%	87,5%
	Hook	38,9%	53,2%
	Jump Shot	37,9%	41,3%
	Layup	63,6%	58,0%
Stephen Curry	3 Points	42,2%	37,9%
	Dunk	25,0%	91,9%
	Hook	53,8%	52,9%

	Jump Shot	43,1%	41,0%
	Layup	60,9%	54,7%

Nous constatons que le modèle n'arrive pas bien à distinguer les spécificités et l'hétérogénéité qu'il peut y avoir parmi les meilleurs joueurs.

En effet, pour LeBron James et Anthony Davis, nous sommes sur des profils de joueurs de type Ailier Fort, c'est-à-dire qui vont très souvent "driver" et attaquer le panier. Le modèle arrive a relativement bien prédire leur probabilité de réussite.

Par contre, un joueur atypique comme Stephen Curry qui tire majoritairement à 3 points et qui ne va quasiment jamais au panier, le modèle sous évalue les tirs à 3 points (37,9% au lieu de 42,2%) et surestime les dunks (91,9% au lieu de 25%)

Bien que nous ayons limité l'entraînement du modèle uniquement sur les données des meilleurs joueurs, nous pouvons considérer que même au sein des meilleurs joueurs, il y a des disparités.

Parmi les 25 meilleurs joueurs ESPN, nous constatons qu'il y a une majorité d'Ailiers Forts et de Pivot, et qu'il y a une minorité d'experts à 3 points.

Finalement, nous nous sommes demandés s'il ne serait pas préférable de faire un modèle par Top Joueur. En effet, chaque Top Joueur a un style de jeu et un style de tir unique. Le meilleur moyen de capter les performances d'un Top joueur serait finalement de faire un modèle par Top Joueur.

2. Modèle entraîné par Top Joueur : exemple Stephen Curry

Dans l'objectif de créer un modèle par Top Joueur, nous avons pris l'exemple de Stephen Curry qui a un profil et des caractéristiques différents de la plupart des meilleurs joueurs du 21ème siècle.

Nous constatons les probabilités de prédiction suivantes sur Stephen Curry :

Action Type	Y_test_Curry	Predict_Proba_XGB _Curry	Predict_Proba_XGB _TopPlayers
dunk	25,0%	68,2%	91,9%
hook	53,8%	61,9%	52,9%
layup	60,9%	57,5%	54,7%
3 PTS	**42,2%**	**44,7%**	**37,9%**
jump shot	43,1%	43,5%	41,0%

En utilisant un modèle entraîné uniquement sur les données de Stephen Curry, nous constatons que les tirs à 3 points sont mieux prédits que si le modèle était entraîné sur l'ensemble des meilleurs joueurs.

Finalement, nous pouvons conclure que si l'objectif est de modéliser le tir des meilleurs du 21ème siècle, le plus adapté serait de créer un modèle par Top Joueur afin de capter toutes les spécificités qu'un joueur peut avoir.

3. Problème de classification et limites

Jusqu'à présent, nous avons comparé les probabilités de prédiction des modèles et les taux de réussite réellement constatés sur l'échantillon test.

Si nous revenons à notre problème de classification, à savoir si un tir doit être classé en réussi (classe 1) ou en tir raté (classe 0), nous sommes confrontés au problème du seuil à déterminer.

Rappel des probabilités prédites par les modèles :

Action Type	y_test	proba_logit	proba_xgboost
3 Points	34,9%	38,3%	37,9%
Dunk	90,1%	93,9%	89,1%
Hook	48,6%	58,9%	53,5%
Jump Shot	43,1%	42,7%	41,5%
Layup	59,5%	63,5%	58,0%

Par défaut, la classification se fait avec un seuil de 0.5. Toutefois si nous conservons ce seuil, tous les tirs de type “3 points” et “Jump Shot” seront classés en raté (classe 0) car les probabilité de réussite est inférieure à 0.5. A titre d’information, si nous avions appliqué le seuil de 0.5, nous aurions obtenu la classification suivante par joueur :

	y_test	y_predict
Anthony Davis	51,50%	49,10%
Chris Paul	45,80%	13,30%
Dwyane Wade	43,40%	42,80%
Giannis Antetokounmpo	55,70%	62,70%
James Harden	43,70%	30,10%
Kawhi Leonard	48,40%	31,00%
Lebron James	50,40%	48,50%
Luka Doncic	44,70%	32,90%
Nikola Jokic	51,90%	48,10%
Russell Westbrook	44,50%	46,50%
Stephen Curry	46,50%	23,70%

Evidemment, nous pouvons affirmer que la prédiction des classes sur la base d’un seuil à 0.5 est fausse.

En effet, tous les tirs à 3 points ou tous les Jump Shot ne sont pas tous ratés. Les joueurs effectuant beaucoup de tirs à distance auraient une prédiction des tirs très basse (par exemple : Stephen Curry) , tandis que les joueurs effectuant beaucoup de tirs proches du panier auraient une prédiction très haute (par exemple : Giannis Antetokounmpo).

Comme nous pouvons le voir, rechercher un seuil optimal n'est pas simple car il va dépendre du type de tir. Nous ne pouvons pas choisir un seuil sans distinguer les différents types de tir. De même, le seuil d'un tir ne sera pas le même en fonction des joueurs.

Nous avons essayé de modifier le seuil, mais les prédictions sont très sensibles en fonction de la variation de cette dernière. L'affichage d'une matrice de confusion ne serait pas pertinente pour évaluer nos modèles.

En résumé, si l'objectif est la classification des tirs, il faudrait mettre en place un modèle par joueur et par type de tir.

Pour ce projet, nous avons choisi d'utiliser les probabilités prédites par les modèles afin de comparer avec les taux de réussite de l'échantillon de test. Quant à l'évaluation des performances des modèles, nous avons choisi d'utiliser les métriques telles que l'AUC, le Brier ou la LogLoss.

Interprétation des modèles

1. Interprétation du modèle Régression Logistique

Nous avons choisi la régression logistique comme premier modèle afin de comprendre les influences que peuvent avoir les variables explicatives sur le tir.

Ci-dessous, vous trouverez l'interprétation des coefficients issus du modèle de régression logistique, entraîné sur les meilleurs joueurs du 21ème siècle.

Variables à influence positive	Coefficient	Odds Ratio
Action Type Dunk	1.965	7.135
Shot Type 2 Points	0.372	1.450
Shot Zone Area Center	0.302	1.353
FG%	0.082	1.082
Age	0.043	1.044

Parmi les variables impactant positivement le tir, nous retrouvons :

- Le Dunk : faire un dunk augmente les chances de réussite de marquer par 7
- Tirer à 2 points augmente les chances de 45% par rapport à un tir à 3 points
- Tirer dans la zone centrale augmente les chances de 35%
- Augmenter le taux FG% de 1 point, augmente les chances de réussite de 8%
    - les performances historiques d'un joueur influencent le tir
- Augmenter l'âge de 1 point, augmente les chances de marquer de 4%
    - nous pouvons interpréter qu'avec l'âge, le joueur gagne en expérience, possède une qualité de tir perfectionné au fil des années d'entraînement et probablement une meilleure gestion du stress.

Variables à influence négative	Coefficient	Odds Ratio
Action Type Jump Shot	-1.011	0.363
Action Type 3 Points	-0.383	0.681
Shot Distance	-0.031	0.969
is_home_0	-0.028	0.972
Period	-0.028	0.972

Parmi les variables impactant négativement le tir, nous retrouvons :

- Le Jump Shot : un tir en extension est un tir difficile à exécuter, les chances de réussite diminue de 64%
- La Distance : augmenter la distance d'une unité diminue le taux de réussite de 3%
- Jouer à l'extérieur diminue les chances de réussite de 3%
- Period : au fil du match, quart-temps après quart-temps, le taux de réussite diminue progressivement de 3% (fatigue, pression, etc.)

Remarque : Lorsque nous avons un modèle basé sur l'ensemble des meilleurs joueurs ou bien un modèle par meilleur joueur, nous avons constaté que les variables explicatives n'avaient pas forcément les mêmes coefficients.

En effet, en prenant l'exemple du modèle Stephen Curry, nous avons par exemple constaté que la variable explicative Distance avait un impact positif sur le tir. Le modèle de régression linéaire basé sur Stephen Curry a probablement capté qu'il s'agissant d'un joueur expert des tirs à longue distance.

2. Interprétation du modèle XGBoost

XGBoost est un modèle ensembliste capable de capturer des relations non linéaires en faisant souvent un excellent candidat pour la classification.

Cette performance rend toutefois l'interprétation de décisions prises par le modèle peu voire pas lisibles. Il s'agit donc pour ce faire d'utiliser un outil spécialisé capable d'identifier quelles variables influencent le plus la probabilité d'un tir et dans quel sens.

La méthode SHAP permet d'expliquer globalement et localement le comportement d'un modèle complexe, à l'instar d'XGBoost en quantifiant la contribution de chaque variable à une prédiction.

SHAP se fonde sur la théorie des valeurs de Shapley (ou théorie des jeux coopératifs). Ainsi, la valeur SHAP d'une variable est sa contribution marginale moyenne à la prédiction. On a ainsi :

$$f(x) = E[f(X)] + \sum_j SHAP_j(x)$$

Par ailleurs l'intérêt d'utiliser une telle méthode pour interpréter notre modèle, notamment par rapport à une méthode "classique" de type model.feature_importances_ est double :

- SHAP est capable d'indiquer dans quel sens (localement, et par additivité, globalement) une variable influence une prédiction.
- SHAP est plus équitable, la méthode *gain* étant biaisée vers les variables à forte cardinalité

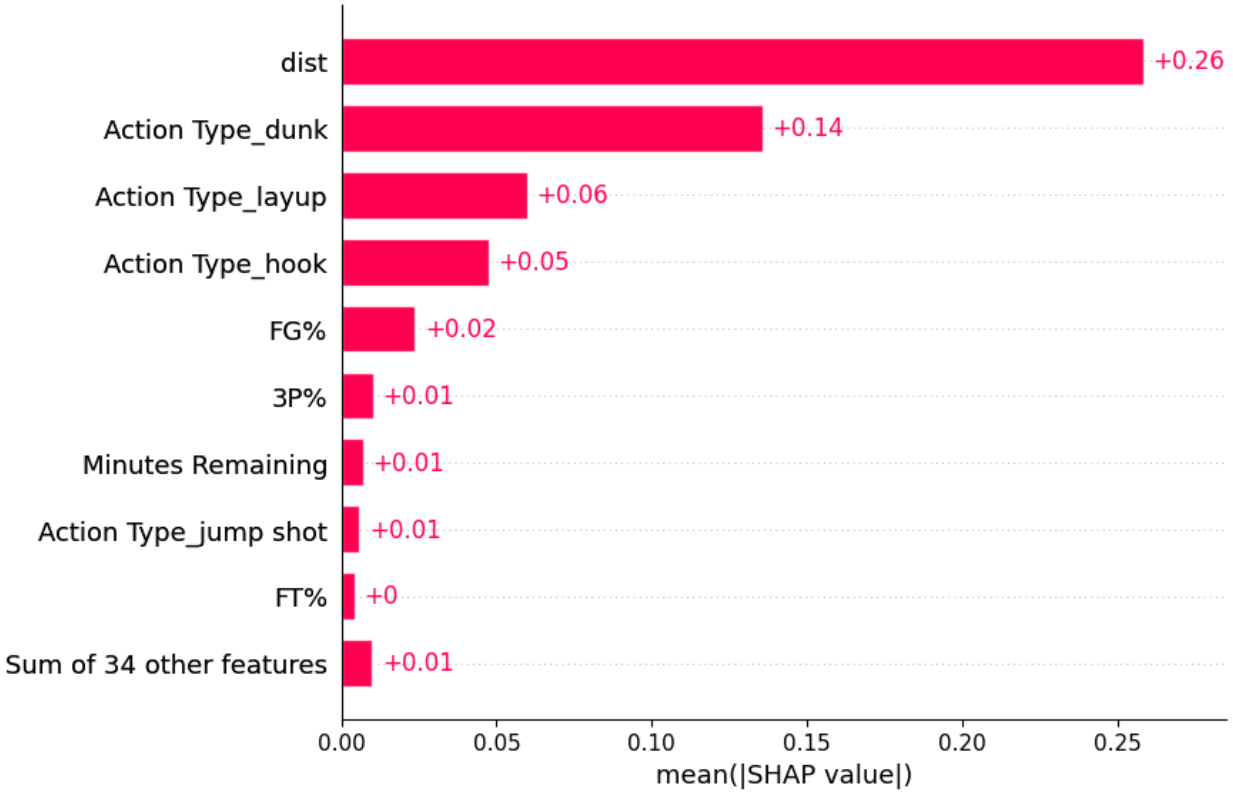
2. Interprétation globale du modèle

Une première approche consiste à considérer, pour chaque variable, la moyenne des valeurs absolues de ses contributions SHAP sur l'ensemble des prédictions. La variable de distance apparaît comme la plus structurante du modèle avec une contribution moyenne d'environ 0.26 sur l'échelle du logit. Ainsi, et sans surprise, pour la majorité des tirs, la distance au panier modifie significativement la probabilité prédite de réussite.

Ensuite, les actions associées au tirs constituent le second ensemble déterminant. Ici, les

dunks rendent compte d'un impact moyen mais notable (+0.14) sur les prédictions.
D'autres types d'action (notamment le layup et le hooj) exercent également une influence moyenne sur les prédictions, mais plus modérée. Ceci est cohérent avec la prépondérance de la variable de distance; chacun de ces types de tirs se conduisant à proximité du panier, et s'agissant du dunk se faisant généralement lorsque le pressing défensif est moindre et reste globalement plus difficile à défendre lorsqu'un joueur décide de le tenter.

Les variables liées au temps restant et aux performances globales du joueurs affichent quant à elles des contributions moyennes très faibles. Elles peuvent avoir un impact ponctuel mais à l'échelle de l'ensemble des prédictions, sont extrêmement marginales en termes d'impact.



En considérant le summary plot de la méthode SHAP, nous sommes également capables d'identifier la distribution directionnelles des contributions SHAP.

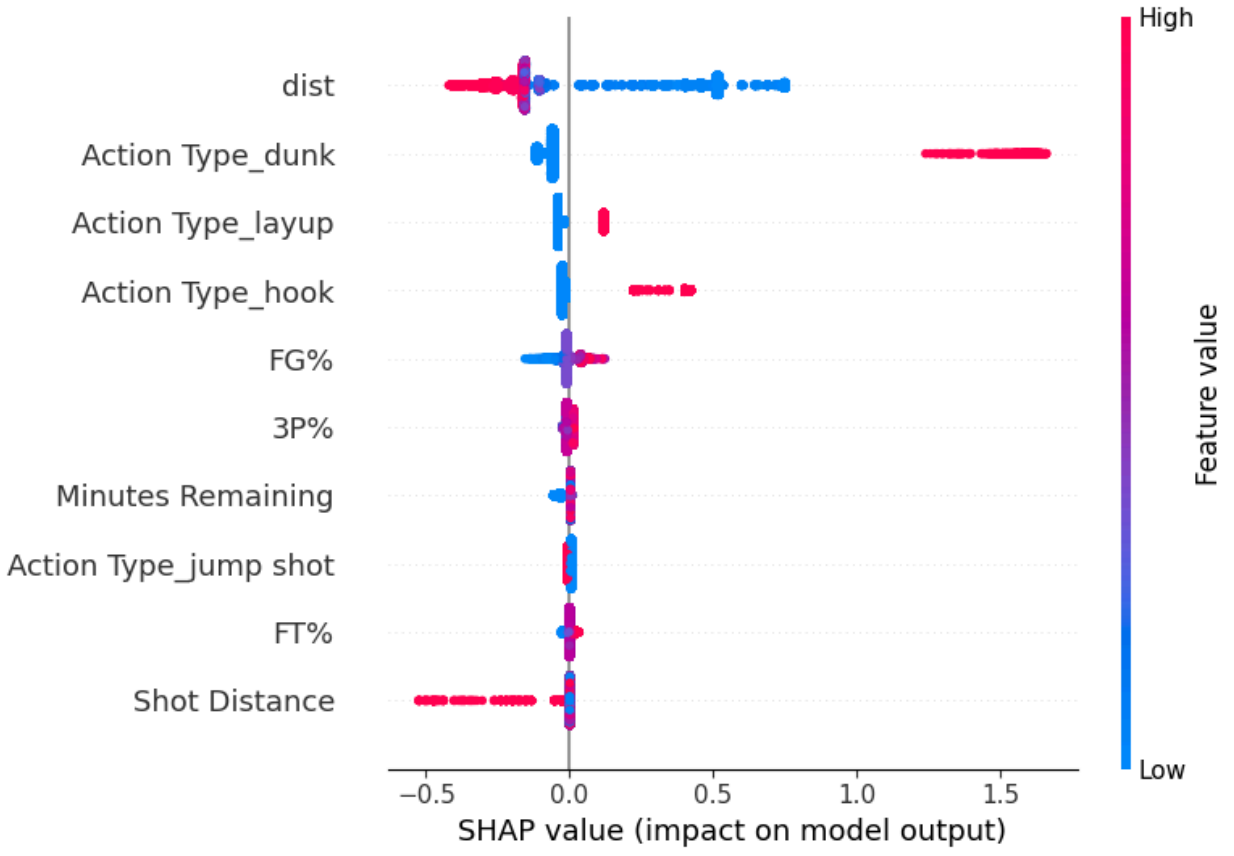
La variable de distance présente une structure tout à fait nette. Les valeurs faibles (en blues) sont associées à des contributions positives tandis que les valeurs élevées (en rouge) influencent négativement les prédictions.

La variable *Action Type_dunk* montre une concentration de points rouges avec des contributions exclusivement positives et souvent élevées, autour de +1.5 sur l'échelle du logit. Autrement dit, lorsqu'un tir correspond à un dunk, il a un impact fortement favorable

sur la prédiction. À l'inverse, lorsque l'action ne correspond pas à un dunk (points bleus), la contribution est proche de zéro, ce qui traduit un effet absent plutôt qu'un effet négatif.

Les autres types de tirs que nous avions mentionnés plus haut, présentent des effets directionnels similaires au dunk quoique beaucoup plus modérés. Ici aussi, un tir layup ou hook impactent positivement la prédiction mais de façon moins générale que le dunk (faible dispersion verticale); ces tirs étant généralement sujets à un pressing défensif plus intensifs et des appuis ou un positionnement au panier parfois précaires. Ici aussi en revanche, lorsque l'action ne correspond pas à un hook ou un layup, l'impact est neutre plutôt que négatif.

Enfin, les variables telles que *FG%*, *Minutes Remaining* ou *3P%* présentent des contributions distribuées autour de zéro, avec peu de dispersion et sans structure directionnelle nette. Elles n'ont qu'un impact marginal sur la plupart des prédictions, ce qui confirme leur faible importance moyenne. Cela suggère que le modèle se base surtout sur des caractéristiques géométriques et typologiques du tir, et moins sur les indicateurs de performance historique ou de contexte temporel.



3. Interprétation locale du modèle

L'interprétation locale doit nous permettre d'exploiter ce qui rend un tir très probable et très improbable selon le modèle en identifiant les combinaisons de variables qui maximisent ou inhibent fortement la probabilité de réussite.

Ainsi, nous ne nous intéressons plus tellement ici à l'effet moyen des variables dans la prédiction mais leur rôle lorsqu'il produit les résultats les plus tranchés.

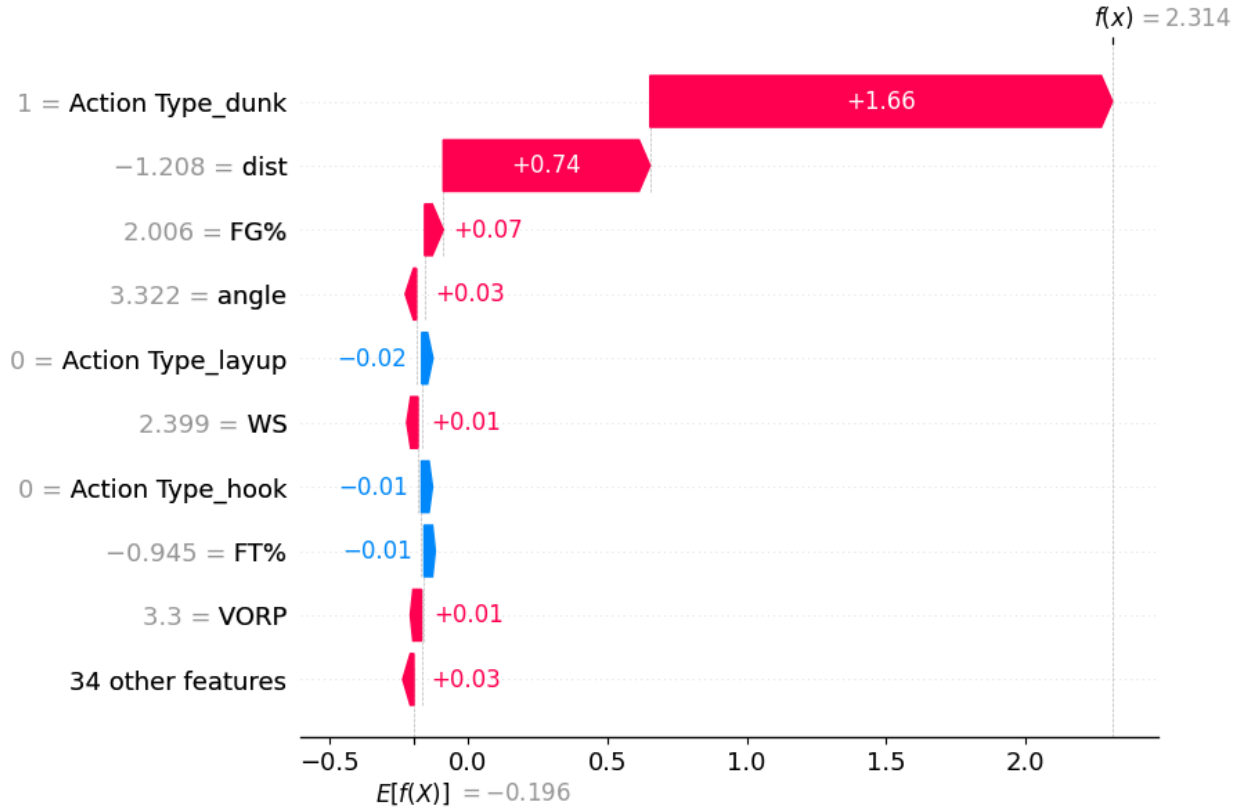
Afin d'illustrer ces mécanismes, nous avons décidé d'étudier les deux observations aux extrêmes des probabilités prédites par le modèle:

- la prédiction la plus élevée (p = 0.91)
- la prédiction la plus faible (p = 0.175)

Le premier graphique (waterfall plot) décompose la prédiction du tir avec la probabilité la plus élevée.

La valeur de référence située à gauche, E[f(X)] = −0,196, correspond au logit moyen sur l'ensemble des observations : c'est le niveau de prédiction en l'absence d'informations spécifiques. À partir de cette base, chaque variable contribue positivement ou négativement, et la somme de ces contributions permet d'atteindre la valeur finale f(x) = 2,314, soit un logit correspondant à une probabilité élevée de réussite.

La variable la plus déterminante ici est *Action Type_dunk* dans la modalité (True) ajoute 1,66 au score du modèle. Cela confirme donc ce qui était observé à l'échelle de l'analyse globale et directionnelle, le fait pour un tir d'être un dunk est un facteur de réussite particulier. La raison pour laquelle cette variable semblait moins importante que la distance dans l'analyse au globale s'explique par la distribution des tirs; seule une fraction d'entre eux sont des dunks. Cependant, la distance reste une variable clé à l'échelle de cette observation. La valeur observée (-1,208) correspond à un tir très proche du panier, et contribue positivement (+0,74) à la prédiction.
Cumulés, ces deux facteurs expliquent l'essentiel de l'écart avec la baseline. Les variables *FG%* et *angle* jouent également un rôle favorable, mais beaucoup plus marginal (+0,07 et +0,03).



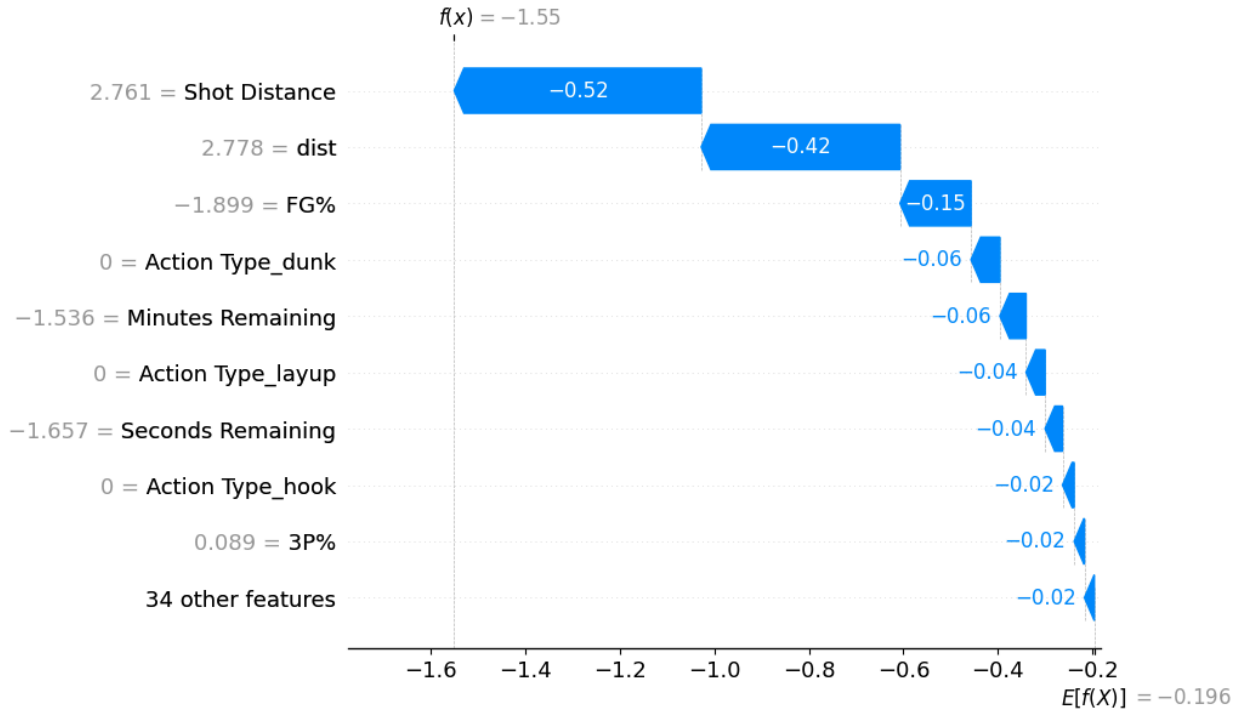
Ensuite, lorsqu'on considère l'observation pour laquelle le modèle a prédit la moins bonne probabilité de réussite, les principaux facteurs défavorables sont directement liés à la difficulté mécanique du tir.

La *Shot Distance* est élevée (2,761), ce qui réduit la prédiction de 0.52, tandis que la variable *dist* (2,778), ici redondante avec la distance réelle, contribue également négativement (−0.42). Ces deux variables constituent les déterminants majeurs de la faible probabilité, confirmant que la distance au panier est un facteur essentiel aussi bien pour les prédictions très favorables que défavorables.

Le fait que l'action ne soit pas un *dunk* (*Action Type_dunk = 0*) réduit encore la probabilité (−0.15). Ceci tempère l'observation qui issue de l'analyse globale selon laquelle si les actions proches du panier (*dunk, layup, hook*) avaient un effet positif sur la prédiction, leur absence semblait être neutre. Ici, on constate, que même s'il est marginal, ces variables influencent négativement la prédiction.

D'autres composantes liées au contexte temporel contribuent légèrement de manière négative (*Minutes Remaining* et *Seconds Remaining*). Enfin, certaines statistiques individuelles du joueur (par exemple un faible *FG%* ou *3P%* dans ce cas) apportent également des contributions négatives modestes.

La somme de ces effets, tous de faible amplitude mais orientés dans le même sens, entraîne une prédiction globalement pessimiste. Contrairement au cas précédent, aucun facteur positif majeur ne vient compenser la distance ou la nature du tir. Ce cas met donc en évidence une configuration mécanique défavorable (tir lointain, absence d'action favorable) sans contexte compensatoire.



L'analyse via SHAP montre que le modèle repose principalement sur des déterminants mécaniques du tir (distance et type d'action), ce qui lui permet de prédire correctement les situations très favorables ou défavorables mais limite sa capacité à discriminer les cas intermédiaires. En l'absence de variables contextuelles plus riches, le modèle capte une logique partielle de la réussite des tirs sans saisir la complexité stratégique du jeu.

Difficultées rencontrées

La première difficulté a été la préparation et la fusion des jeux de données. Le projet reposant sur plusieurs sources hétérogènes. Il a fallu harmoniser les identifiants , nettoyer les doublons , gérer des valeurs manquantes et des colonnes incomplètes, tout en conservant un maximum d'informations pour la modélisation. La constitution d'un jeu des tirs enrichis par les stats de saison a nécessité plusieurs étapes de jointures successives et de filtrages.

Une autre difficulté étant la taille du jeu de données et les contraintes de temps de calculs.on dispose de plus de deux millions de tirs pour l'ensemble des joueurs, et près de 200 000 tirs pour les "Top Players" seuls. Travailler sur l'intégralité de ces données aurait rendu la phase d'entraînement, de recherche d'hyperparamètres et de calibration très lourde, en particulier pour XGBoost. Il a donc fallu trouver un compromis en échantillonnant un sous-ensemble (environ 300 000 tirs) , sur les premières modélisations suffisamment riches pour entraîner des modèles robustes, mais assez léger pour permettre des itérations rapides sur le pré-processing et la modélisation. Par la suite nous avons resserré l'échantillonnage en prenant les top players des années 2010 à 2020.

Enfin, la complexité intrinsèque du phénomène à modéliser a constitué un obstacle important. La réussite d'un tir dépend de nombreux facteurs non observés dans les données disponibles : pression défensive, schémas tactiques, fatigue, contexte psychologique, etc. Le jeu de données ne contient essentiellement que des informations sur le tireur, la localisation et quelques variables de contexte (période, temps restant, domicile/extérieur). Malgré la création de variables dérivées (distance, angle, temps restant, agrégation des types de tir), il reste une part de variabilité très importante, que les modèles ont du mal à capturer.

Conclusion Générale

L'objectif de ce travail était de modéliser la probabilité de réussite d'un tir en NBA pour les top joueurs , à partir d'informations sur le tir (distance, angle, type d'action, contexte temporel), sur le joueur (morphologie, poste, statistiques de saison) et sur le match (domicile/extérieur, type de saison, période). Nous avions plusieurs jeux de données à disposition qui ont été nettoyés, fusionnés et enrichis, puis intégrés dans un pipeline de prétraitement standardisé avant d'entraîner différents modèles de classification.

La mise en place d'un pipeline complet (gestion des valeurs manquantes, création de variables dérivées, standardisation, encodage, calibration des probabilités) a permis de rendre la démarche reproductible, de limiter les fuites d'information entre apprentissage et test, et de comparer de manière équitable plusieurs familles de modèles. Deux modèles ont été retenus : une régression logistique, servant de modèle de référence interprétable, et un modèle de gradient boosting d'arbres (XGBoost), plus flexible et capable de capturer des relations non linéaires.

Les résultats obtenus montrent que les modèles parviennent à faire nettement mieux qu'un classifieur aléatoire, avec des AUC de l'ordre de 0,65 pour la régression logistique et autour de 0,67 pour XGBoost. Ces performances restent loin d'une prédiction parfaite : la réussite d'un tir demeure un phénomène fortement aléatoire, influencé par de nombreux facteurs non observés dans les données (pression défensive, stratégie offensive, fatigue, contexte psychologique, etc.).

Finalement, ce projet a mis en évidence les limites de nos données. Il est possible de construire des modèles raisonnablement fiables pour estimer la probabilité de réussite d'un tir, mais que la marge de progression reste importante. Le projet a permis de poser une base méthodologique solide (pipeline, gestion des NA, choix de modèles, calibration) sur laquelle des travaux plus poussés pourront s'appuyer.

Plusieurs pistes d'amélioration se dégagent naturellement de ce travail.

La première concerne l'enrichissement des données. Les variables utilisées restent centrées sur le joueur et son tir sans prendre en compte le le contexte défensif et tactique : distance du défenseur, nombre de défenseurs impliqués, type de système offensif, issue de l'action ou encore si la balle a été touchée par le tir.

Une deuxième piste consiste à aller vers des modèles plus spécifiques et moins "globaux". Les résultats suggèrent que les modèles entraînés sur l'ensemble des joueurs capturent mal les profils très atypiques (spécialistes du tir à 3 points lointains, intérieurs très mobiles, etc.). Il serait intéressant d'explorer des approches par sous-groupes (par poste, par type de rôle) ou même des modèles personnalisés par joueur comme nous l'avons fait pour Stephen Curry et les résultats s'avéraient très proches de la réalité.

Sur le plan méthodologique, des approches séquentielles ou temporelles pourraient également être explorées. Dans ce projet, chaque tir est traité comme une observation indépendante. Or, les tirs s'inscrivent dans une dynamique de match : séquences d'actions, séries de réussite ou d'échec, gestion du score, fatigue progressive. L'utilisation des modèles de séries temporelles par joueur/match) comme les modèles ARFIMA qui sont des processus de mémoires longues et qui captent les phénomènes de persistances.

En résumé, ce travail constitue une première étape vers une modélisation probabiliste des tirs en NBA. Il montre à la fois ce qu'il est déjà possible de faire avec des données tabulaires enrichies et des modèles supervisés classiques, et met en lumière les limites liées à la nature des données disponibles. Les perspectives ouvertes, tant en termes de données que de méthodes, laissent entrevoir de nombreuses possibilités pour des recherches futures, à l'interface entre data science et analyse du jeu.

ANNEXES

Données utilisées:

dataset des tirs NBA entre 1997 et 2019 :

https://www.kaggle.com/jonathangmwl/nba-shot-locations

dataset des actions de chaque match entre 2000 et 2020 :

https://sports-statistics.com/sports-data/nba-basketball-datasets-csv-files/

dataset des bilans d'équipe entre 2014 et 2018 :

https://www.kaggle.com/nathanlauga/nba-games?select=ranking.csv

dataset des joueurs de NBA depuis 1950 :

https://www.kaggle.com/drgilermo/nba-players-stats?select=Players.csv

Premières modélisations:

1. **Régression logistique**

```
pd.options.mode.copy_on_write = True

SEED = 42

SAMPLE_TOTAL = 300_000



needed = [

    "Player", "Player Name", "Game Date", "Shot Made Flag",

    "Shot Distance", "X Location", "Y Location",

    "Minutes Remaining", "Seconds Remaining", "Period",

    "Shot Type", "Action Type", "Season Type", "is_home",

    "dist", "angle"

]

keep = [c for c in needed if c in data.columns]

df = data.loc[:, keep]


if "Game Date" in df.columns and not np.issubdtype(df["Game Date"].dtype,
np.datetime64):

    df["Game Date"] = pd.to_datetime(df["Game Date"].astype(str),
format="%Y%m%d", errors="coerce")


for c in ["Shot Distance","X Location","Y Location","Minutes
Remaining","Seconds Remaining","Period","Shot Made Flag"]:

    if c in df.columns:

        df[c] = pd.to_numeric(df[c], errors="coerce")
```

```
if {"X Location","Y Location"}.issubset(df.columns):
    if "dist" not in df.columns:
        df["dist"] = np.hypot(df["X Location"], df["Y Location"])
    if "angle" not in df.columns:

        df["angle"] = np.arctan2(df["X Location"], df["Y Location"])

if "Shot Type" not in df.columns:
    sdist = pd.to_numeric(df.get("Shot Distance", np.nan),
errors="coerce")
    xabs  = pd.to_numeric(df.get("X Location", np.nan),
errors="coerce").abs()
    three = ((sdist >= 22) & (xabs > 220)) | (sdist >= 23.75)
    df["Shot Type"] = np.where(three, "3PT Field Goal", "2PT Field Goal")

if "Season Type" not in df.columns:
    df["Season Type"] = "Regular Season"
if "is_home" not in df.columns:
    df["is_home"] = 0

df = df.dropna(subset=["Shot Made Flag","Game Date"]).copy()
bounds = {
    "Minutes Remaining": (0, 11),
    "Seconds Remaining": (0, 59),
    "Period": (1, 12),
    "X Location": (-260, 260),
    "Y Location": (-60, 480),
```

```
    "Shot Distance": (0, 45),
}
for c, (lo, hi) in bounds.items():
    if c in df.columns:
        s = pd.to_numeric(df[c], errors="coerce")
        df = df[s.between(lo, hi) | s.isna()]



df["Year"] = df["Game Date"].dt.year.astype("Int64")



##########################################################################
######
########## REGROUPEMENT DE ACTION TYPE
##########################################
##########################################################################
#####

action_type_mapping = {
    # Jump Shots
    'Jump Shot': 'jump shot',
    'Running Jump Shot': 'jump shot',
    'Turnaround Jump Shot': 'jump shot',
    'Pullup Jump shot': 'jump shot',
    'Step Back Jump shot': 'jump shot',
    'Floating Jump shot': 'jump shot',
    'Fadeaway Jump Shot': 'jump shot',
    'Driving Floating Jump Shot': 'jump shot',
```

```
    'Driving Jump shot': 'jump shot',


    # Hook Shots
    'Hook Shot': 'hook',
    'Turnaround Hook Shot': 'hook',
    'Jump Hook Shot': 'hook',
    'Running Hook Shot': 'hook',
    'Turnaround Finger Roll Shot': 'hook',
    'Turnaround Fadeaway shot': 'hook',
    'Jump Bank Shot': 'hook',
    'Hook Bank Shot': 'hook',
    'Turnaround Bank shot': 'hook',
    'Turnaround Bank Hook Shot': 'hook',
    'Jump Bank Hook Shot': 'hook',
    'Fadeaway Bank shot': 'hook',
    'Pullup Bank shot': 'hook',
    'Running Bank shot': 'hook',
    'Running Bank Hook Shot': 'hook',
    'Driving Bank shot': 'hook',
    'Driving Bank Hook Shot': 'hook',
    'Driving Floating Bank Jump Shot': 'hook',
    'Step Back Bank Jump Shot': 'hook',
    'Turnaround Fadeaway Bank Jump Shot': 'hook',
    'Driving Hook Shot': 'hook',


    # Layups
    'Layup Shot': 'layup',
```

```
    'Driving Layup Shot': 'layup',
    'Reverse Layup Shot': 'layup',
    'Running Layup Shot': 'layup',
    'Alley Oop Layup shot': 'layup',
    'Finger Roll Shot': 'layup',
    'Driving Finger Roll Shot': 'layup',
    'Running Finger Roll Shot': 'layup',
    'Driving Finger Roll Layup Shot': 'layup',
    'Running Finger Roll Layup Shot': 'layup',
    'Putback Layup Shot': 'layup',
    'Finger Roll Layup Shot': 'layup',
    'Driving Reverse Layup Shot': 'layup',
    'Running Reverse Layup Shot': 'layup',
    'Tip Layup Shot': 'layup',
    'Cutting Layup Shot': 'layup',
    'Cutting Finger Roll Layup Shot': 'layup',
    'Running Alley Oop Layup Shot': 'layup',


    # Dunks
    'Slam Dunk Shot': 'dunk',
    'Dunk Shot': 'dunk',
    'Driving Dunk Shot': 'dunk',
    'Alley Oop Dunk Shot': 'dunk',
    'Reverse Dunk Shot': 'dunk',
    'Follow Up Dunk Shot': 'dunk',
    'Driving Slam Dunk Shot': 'dunk',
    'Putback Dunk Shot': 'dunk',
```

```
    'Reverse Slam Dunk Shot': 'dunk',
    'Putback Slam Dunk Shot': 'dunk',
    'Cutting Dunk Shot': 'dunk',
    'Running Slam Dunk Shot': 'dunk',
    'Driving Reverse Dunk Shot': 'dunk',
    'Running Alley Oop Dunk Shot': 'dunk',
    'Running Reverse Dunk Shot': 'dunk',
    'Tip Dunk Shot': 'dunk',
    'Running Tip Shot': 'dunk',

    # No Shot
    'No Shot': 'no shot',
}

df['ACTION_CATEGORY'] = df["Action Type"].map(action_type_mapping)

#on ajoute la notion de 3 points qui est aussi une catégorie
df['ACTION_CATEGORY'] = df.apply(
    lambda row: '3 PTS' if row['Shot Type'] == '3PT Field Goal' else
row['ACTION_CATEGORY'],
    axis=1
)

# Garder uniquement les lignes qui ne sont pas "no shot"
df = df.loc[df['ACTION_CATEGORY'] != 'no shot'].copy()

df['Action Type'] = df['ACTION_CATEGORY']
df = df.drop('ACTION_CATEGORY', axis = 1)
```

```
###########################################################################
###################

############################FIN DU MAPPING SUR ACTION
TYPE################################

###########################################################################
######################



val_start  = pd.Timestamp("2016-07-01")

test_start = pd.Timestamp("2018-07-01")


mask_tr = df["Game Date"] < val_start

mask_va = (df["Game Date"] >= val_start) & (df["Game Date"] < test_start)

mask_te = df["Game Date"] >= test_start


y_all = df["Shot Made Flag"].astype("int8")

X_all = df.drop(columns=["Shot Made Flag"])


def stratified_sample(X, y, n, seed=SEED):

    if n >= len(X): return X, y

    tmp = X.assign(__y__=y.values)

    counts = tmp["__y__"].value_counts()

    if len(counts) < 2:

        idx = tmp.sample(n=n, random_state=seed).index

        return X.loc[idx], y.loc[idx]

    frac = n / len(tmp)
```

```
    n_per = (counts * frac).astype(int)
    diff = n - int(n_per.sum())
    if diff:
        for cls in counts.sort_values(ascending=False).index:
            if diff == 0: break
            n_per[cls] += 1; diff -= 1
    parts = []
    for cls, g in tmp.groupby("__y__", sort=False):
        k = min(int(n_per.get(cls, 0)), len(g))
        if k > 0: parts.append(g.sample(n=k, random_state=seed))
    out = pd.concat(parts, axis=0)
    y_s = out.pop("__y__")
    return out, y_s


sizes = np.array([mask_tr.sum(), mask_va.sum(), mask_te.sum()],
dtype=float)
ratios = sizes / max(sizes.sum(), 1)
take = np.floor(SAMPLE_TOTAL * ratios).astype(int)
take[0] += SAMPLE_TOTAL - take.sum()


X_train, y_train = stratified_sample(X_all[mask_tr], y_all[mask_tr],
int(take[0]))
X_valid, y_valid = stratified_sample(X_all[mask_va], y_all[mask_va],
int(take[1]))
X_test,  y_test  = stratified_sample(X_all[mask_te], y_all[mask_te],
int(take[2]))


print(f"Échantillon → train:{len(X_train)} | valid:{len(X_valid)} |
test:{len(X_test)}")
```

```
seasons_small = seasons.copy()
if "Year" in seasons_small.columns:
    seasons_small["Year"] = pd.to_numeric(seasons_small["Year"],
errors="coerce").astype("Int64")

for pct in ["FG%","3P%","FT%"]:
    if pct in seasons_small.columns:
        m = seasons_small[pct] > 1
        seasons_small.loc[m, pct] = seasons_small.loc[m, pct] / 100.0

if {"Player","Year","Tm","G"}.issubset(seasons_small.columns):
    seasons_small = (
        seasons_small.assign(_prefTOT=(seasons_small["Tm"].astype(str) ==
"TOT").astype(int))
                     .sort_values(["Player","Year","_prefTOT","G"],
ascending=[True, True, False, False])
                     .drop_duplicates(["Player","Year"], keep="first")
                     .drop(columns="_prefTOT")
    )
players_years = pd.concat([
    X_train[["Player","Year"]],
    X_valid[["Player","Year"]],
    X_test[["Player","Year"]]
], axis=0).dropna().drop_duplicates()
```

```
seasons_small = players_years.merge(seasons_small, on=["Player","Year"],
how="left")


def merge_split(Xs):
    return Xs.merge(seasons_small, on=["Player","Year"], how="left",
suffixes=("", "_season"))


X_train = merge_split(X_train)
X_valid = merge_split(X_valid)
X_test  = merge_split(X_test)



num_cols = [c for c in [
    "dist","angle","Shot Distance","X Location","Y Location",
    "Minutes Remaining","Seconds Remaining","Period",
    "Age","G","MP","PTS","FG%","3P%","FT%","WS","PER","VORP"
] if c in X_train.columns]


cat_cols = [c for c in [
    "Shot Type","Action Type","Season Type","is_home","Pos"
] if c in X_train.columns]


pre = ColumnTransformer(
    transformers=[
        ("num", Pipeline([
            ("impute", SimpleImputer(strategy="median")),
            ("scale", StandardScaler())
        ]), num_cols),
```

```
        ("cat", Pipeline([
            ("impute", SimpleImputer(strategy="most_frequent")),
            ("onehot", OneHotEncoder(handle_unknown="ignore",
sparse_output=True, min_frequency=100)),
        ]), cat_cols),
    ],
    remainder="drop",
    verbose_feature_names_out=False
)

clf = LogisticRegression(max_iter=300, solver="lbfgs")
pipe = Pipeline([("pre", pre), ("clf", clf)])

drop_id = ["Player","Player Name"]
Xtr_in = X_train.drop(columns=[c for c in drop_id if c in
X_train.columns])
Xva_in = X_valid.drop(columns=[c for c in drop_id if c in
X_valid.columns])
Xte_in = X_test.drop(columns=[c for c in drop_id if c in X_test.columns])

pipe.fit(Xtr_in, y_train)

def eval_pipe(pipe, Xs, ys, name):
    proba = pipe.predict_proba(Xs)[:, 1]
    return {"split": name,
            "AUC": roc_auc_score(ys, proba),
            "Brier": brier_score_loss(ys, proba),
            "LogLoss": log_loss(ys, proba, labels=[0,1])}
```

```
results = []
results.append(eval_pipe(pipe, Xtr_in, y_train, "train"))
if len(X_valid): results.append(eval_pipe(pipe, Xva_in, y_valid, "valid"))
if len(X_test):  results.append(eval_pipe(pipe, Xte_in, y_test,  "test"))

print(pd.DataFrame(results).round(3))
```

2. XGBoost

Code similaire , juste le choix du modèle à modifier:

```
pre = ColumnTransformer(
    transformers=[
        ("num", Pipeline([
            ("impute", SimpleImputer(strategy="median")),
            ("scale", StandardScaler())
        ]), num_cols),
        ("cat", Pipeline([
            ("impute", SimpleImputer(strategy="most_frequent")),
            ("onehot", OneHotEncoder(handle_unknown="ignore",
sparse_output=True, min_frequency=100)),
        ]), cat_cols),
    ],
    remainder="drop",
    verbose_feature_names_out=False
)


xgb = XGBClassifier(
    n_estimators=250,
```

```
    max_depth=5,
    learning_rate=0.05,
    subsample=0.9,
    colsample_bytree=0.9,
    reg_lambda=1.0,
    n_jobs=-1,
    tree_method="hist",
    objective="binary:logistic",
    eval_metric="logloss",
    random_state=42
)

pipe = Pipeline([("pre", pre), ("clf", xgb)])

drop_cols = [c for c in ["Player","Player Name","Game Date","Year"] if c
in X_train.columns]
Xtr_in = X_train.drop(columns=drop_cols)
Xva_in = X_valid.drop(columns=drop_cols)
Xte_in = X_test.drop(columns=drop_cols)



pipe.fit(Xtr_in, y_train)



def evaluate(pipe, Xs, ys, name):
    proba = pipe.predict_proba(Xs)[:, 1]
    return {"split": name,
            "AUC": roc_auc_score(ys, proba),
```

```
            "Brier": brier_score_loss(ys, proba),
            "LogLoss": log_loss(ys, proba, labels=[0,1])}

results = []
results.append(evaluate(pipe, Xtr_in, y_train, "train"))
if len(X_valid): results.append(evaluate(pipe, Xva_in, y_valid, "valid"))
if len(X_test):  results.append(evaluate(pipe, Xte_in, y_test,  "test"))

print(pd.DataFrame(results).round(3))
```

Modélisation années 2010 à 2020:

1. **Régression Logistique**

```
start = pd.Timestamp("2010-07-01")
val_start  = pd.Timestamp("2016-07-01")
test_start = pd.Timestamp("2018-07-01")

df = df.copy()

if not np.issubdtype(df["Game Date"].dtype, np.datetime64):
        df["Game Date"] = pd.to_datetime(df["Game Date"], errors="coerce",
format='%Y%m%d')

df = df[(df["Game Date"] >= start)]



train = df[df["Game Date"] < val_start].copy()
```

```
valid = df[(df["Game Date"] >= val_start) & (df["Game Date"] <
test_start)].copy()
test  = df[df["Game Date"] >= test_start].copy()



y_train = train["Shot Made Flag"].astype("int8")
X_train = train.drop("Shot Made Flag", axis = 1)
y_valid = valid["Shot Made Flag"].astype("int8")
X_valid = valid.drop("Shot Made Flag", axis = 1)
y_test = test["Shot Made Flag"].astype("int8")
X_test = test.drop("Shot Made Flag", axis = 1)




seasons_small = seasons.copy()
if "Year" in seasons_small.columns:
    seasons_small["Year"] = pd.to_numeric(seasons_small["Year"],
errors="coerce").astype("Int64")

for pct in ["FG%","3P%","FT%"]:
    if pct in seasons_small.columns:
        m = seasons_small[pct] > 1
        seasons_small.loc[m, pct] = seasons_small.loc[m, pct] / 100.0

if {"Player","Year","Tm","G"}.issubset(seasons_small.columns):
    seasons_small = (
        seasons_small.assign(_prefTOT=(seasons_small["Tm"].astype(str) ==
"TOT").astype(int))
```

```
                    .sort_values(["Player","Year","_prefTOT","G"],
ascending=[True, True, False, False])
                    .drop_duplicates(["Player","Year"], keep="first")
                    .drop(columns="_prefTOT")
    )
players_years = pd.concat([
    X_train[["Player","Year"]],
    X_valid[["Player","Year"]],
    X_test[["Player","Year"]]
], axis=0).dropna().drop_duplicates()

seasons_small = players_years.merge(seasons_small, on=["Player","Year"],
how="left")

def merge_split(Xs):
    return Xs.merge(seasons_small, on=["Player","Year"], how="left",
suffixes=("", "_season"))

X_train = merge_split(X_train)
X_valid = merge_split(X_valid)
X_test  = merge_split(X_test)

num_cols = [c for c in [
    "dist","angle","Shot Distance","X Location","Y Location",
    "TIME_LEFT","Period",
    "Age","G","MP","PTS","FG%","3P%","FT%","WS","PER","VORP"
] if c in X_train.columns]
```

```
cat_cols = [c for c in [
    "Shot Type","Action Type","Season Type","is_home","Pos", "Shot Zone
Area"
] if c in X_train.columns]

pre = ColumnTransformer(
    transformers=[
        ("num", Pipeline([
            ("impute", SimpleImputer(strategy="median")),
            ("scale", StandardScaler())
        ]), num_cols),
        ("cat", Pipeline([
            ("impute", SimpleImputer(strategy="most_frequent")),
            ("onehot", OneHotEncoder(handle_unknown="ignore",
sparse_output=True, min_frequency=100)),
        ]), cat_cols),
    ],
    remainder="drop",
    verbose_feature_names_out=False
)

clf = LogisticRegression(max_iter=300, solver="lbfgs")
pipe_logit = Pipeline([("pre", pre), ("clf", clf)])

drop_id = ["Player","Player Name"]
Xtr_in = X_train.drop(columns=[c for c in drop_id if c in
X_train.columns])
```

```
Xva_in = X_valid.drop(columns=[c for c in drop_id if c in
X_valid.columns])
Xte_in = X_test.drop(columns=[c for c in drop_id if c in X_test.columns])


pipe_logit.fit(Xtr_in, y_train)


def eval_pipe(pipe_logit, Xs, ys, name):
    proba = pipe_logit.predict_proba(Xs)[:, 1]
    return {"split": name,
            "AUC": roc_auc_score(ys, proba),
            "Brier": brier_score_loss(ys, proba),
            "LogLoss": log_loss(ys, proba, labels=[0,1])}


results = []
results.append(eval_pipe(pipe_logit, Xtr_in, y_train, "train"))
if len(X_valid): results.append(eval_pipe(pipe_logit, Xva_in, y_valid,
"valid"))
if len(X_test):  results.append(eval_pipe(pipe_logit, Xte_in, y_test,
"test"))


print(pd.DataFrame(results).round(3))
############################################################
######### EVALUATION DU MODELE DE REGRESSION LOGISTIQUE###########
############################################################


# AFFICHAGE DU RAPPORT A TITRE D'INFORMATION MAIS PAS PERTINENT POUR NOTRE
SUJET
```

```
from sklearn.metrics import classification_report, roc_auc_score,
log_loss, brier_score_loss

# Prédictions
y_pred_logit = pipe_logit.predict(Xte_in)
y_proba_logit = pipe_logit.predict_proba(Xte_in)[:, 1]

print("=== Classification Report (Régression Logistique) ===")
print(classification_report(y_test, y_pred_logit, digits=3))

print(f"AUC : {roc_auc_score(y_test, y_proba_logit):.3f}")
print(f"Brier Score : {brier_score_loss(y_test, y_proba_logit):.3f}")
print(f"LogLoss : {log_loss(y_test, y_proba_logit):.3f}")




#########INTERPRETATION DES COEFFICIENTS DE LA REG LOG
############################
print("\n \n === Influence des variables explicatives (Régression
Logistique) ===")

# Récupérer les coefficients
coefs = pipe_logit.named_steps["clf"].coef_[0]

# Noms des colonnes après transformation
features = pipe_logit.named_steps["pre"].get_feature_names_out()
```

```
# Construire le DataFrame des coefficients
df_coef = pd.DataFrame({
    "feature": features,
    "coef": coefs,
    "odds_ratio": np.exp(coefs)
})

# Trier par importance décroissante
print("\n --- Les 10 variables les plus influentes positivement ---")
df_coef = df_coef.sort_values(by="odds_ratio", ascending=False)
print(df_coef.head(10))

#interprétation :
# les variables impactant positivement :
    #si le type de tir est un dunk, les chances de réussite par rapport à un tir de référence est de x7
    #le fait de faire un tir à 2 points augmente les chances de réussite de x1,5 qu'un tir à 3 points
    #Gagner une unité sur %FG, augmente les chances de réussite de de +7%
    #les joueurs évoluant au poste de PG (meneur de jeu) ont des chances de réussite +7% que les autres postes
    #jouer en saison régulière augmente les chances de réussites +7% par rapport au playoff



print("\n --- Les 10 variables les plus influentes négativement ---")
```

```
# Filtrer les variables avec coef négatif
df_negatives = df_coef[df_coef["coef"] < 0]


# Trier par coef croissant (plus négatif en haut)
df_negatives = df_negatives.sort_values("coef")
df_negatives.head(10)


#interprétation des variables impactant négativement
#### le fait de tirer de loin (à 3 PTS) réduit le taux de réussite du tir : -30% de chance réussite
#### le fait d'effectuer un jump shot (tir en extension = technique tir difficile à effectuer) : -62% de chance de réussite
#### Period : lorsqu'on avance dans le match, le taux de réussite diminue de 3% (stress, fatigue, enjeu, etc)
### La distance : augmenté la distance d'une unité fait baisse le taux de réussite de 3%
```

2. Modèle XGBoost

```
xgb = XGBClassifier(
    n_estimators=250,
    max_depth=5,
    learning_rate=0.05,
    subsample=0.9,
    colsample_bytree=0.9,
    reg_lambda=1.0,
    n_jobs=-1,
    tree_method="hist",
    objective="binary:logistic",
```

```
    eval_metric="logloss",
    random_state=42
)

pipe = Pipeline([("pre", pre), ("clf", xgb)])




pipe.fit(Xtr_in, y_train)



def evaluate(pipe, Xs, ys, name):
    proba = pipe.predict_proba(Xs)[:, 1]
    return {"split": name,
            "AUC": roc_auc_score(ys, proba),
            "Brier": brier_score_loss(ys, proba),
            "LogLoss": log_loss(ys, proba, labels=[0,1])}

results = []
results.append(evaluate(pipe, Xtr_in, y_train, "train"))
if len(X_valid): results.append(evaluate(pipe, Xva_in, y_valid, "valid"))
if len(X_test):  results.append(evaluate(pipe, Xte_in, y_test,  "test"))

print(pd.DataFrame(results).round(3))
############################################################
######### EVALUATION DU MODELE XGBOOST ###########
```

```
#####################################################################

# AFFICHAGE DU RAPPORT A TITRE D'INFORMATION MAIS PAS PERTINENT POUR NOTRE
SUJET



from sklearn.metrics import classification_report, roc_auc_score,
log_loss, brier_score_loss

# Prédictions
y_pred_xgb = pipe.predict(Xte_in)
y_proba_xgb = pipe.predict_proba(Xte_in)[:, 1]

print("=== Métriques (XGBOOST) ===")

print(f"AUC : {roc_auc_score(y_test, y_proba_xgb):.3f}")
print(f"Brier Score : {brier_score_loss(y_test, y_proba_xgb):.3f}")
print(f"LogLoss : {log_loss(y_test, y_proba_xgb):.3f}")




#########INTERPRETATION DE XGBOOST ############################
# Récupérer le modèle XGBoost entraîné
xgb_model = pipe.named_steps["clf"]

# Noms des variables (après preprocessing)
features = pipe.named_steps["pre"].get_feature_names_out()
```

```
# Importances des features selon le modèle
importances = xgb_model.feature_importances_

# DataFrame d'analyse
df_importances = pd.DataFrame({
    "feature": features,
    "importance": importances
}).sort_values(by="importance", ascending=False)

print(df_importances.head(20))

#Le dunk est la variable la plus utilisée dans la prédiction du modèle : 38% de l'importance
# Ensuite, cest le tir à 3 PTS : 20% du poids décisionnel etc
```

########## Evaluation du MODELE XGBOOST avec SHAP ##########

```
pre  = pipe_xgb.named_steps["pre"]
clf  = pipe_xgb.named_steps["clf"]

Xva_tr = pre.transform(Xva_in)

if hasattr(Xva_tr, "toarray"):
   Xva_tr = Xva_tr.toarray()

feat_names = pre.get_feature_names_out()
```

```
explainer   = shap.TreeExplainer(clf)
shap_values = explainer(Xva_tr)

shap.summary_plot(shap_values, Xva_tr, feature_names=feat_names,
max_display=30)
```