Rapport de Projet Machine Learning

PROBLEMATIQUE :

Prédiction des résultats des matches de *"Premier League"*

REALISE PAR :	ENCADRE PAR:
Idrissi Mohammed Amine **Bader Rezzouki** **Oustad Mohammed** **Maahtala Mohammed**	**Mr Zakaria Hajja**

1 - Contexte et objectif :

1.1 - Introduction :

Le football, et en particulier la Premier League anglaise, est un sport suivi passionnément par des millions de personnes à travers le monde. La prédiction des résultats des matchs et des statistiques des clubs de football est une activité qui suscite un grand intérêt, que ce soit pour les fans, les analystes, les équipes sportives ou les parieurs. Avec l'avènement des technologies de l'information et la disponibilité croissante des données sportives, il est désormais possible d'appliquer des techniques d'apprentissage automatique pour tenter de prédire les résultats des matchs de football.

1.2 - Objectif :

Ce mini-projet vise à développer un modèle de machine Learning capable de prédire les de victoires pour chaque club pendant les prochaines saisons en Premier League. Le modèle sera entraîné sur un ensemble de données historiques de Premier League pendant des anciennes saisons et utilisera des techniques de machine Learning pour identifier les facteurs clés qui influencent les résultats.

2 - Collecte et exploration des données :

2.1 - Description des Données

Pour commencer, nous avons importé les bibliothèques nécessaires et chargé le dataset à partir d'un fichier CSV. Le dataset utilisé pour ce projet sera constitué de données historiques de matchs de Premier League. Les données incluront des informations (wins ,losses ,red_cerds , yellow_cards ,goals , saves ...) pour chaque équipe pour différentes saisons. Ces données peuvent être obtenues à partir de cette source :
Premier-league-dataset

```
import pandas as pd
```

```
data = pd.read_csv("C:/Users/Dell Latitude/Downloads/stats.csv")
#affiche dataset
data
```

2.2 - Analyse exploratoire des données du dataset

```
df = data
#visualise les premiere ligne de dataset
df.head()
```

```
#vusialise les dernires ligne de dataset
df.tail()
```

Cette instruction permet d'afficher les premiers et les dernières lignes du dataset. Cela permet de se faire une idée générale des données et de vérifier qu'il n'y a pas de valeurs aberrantes.

2. Visualisation du dataset regroupé par équipe

```
grouped = data.groupby('team')
sorted_data = []
for team, group in grouped:
    sorted_group = group.sort_values(by='season')
    sorted_data.append(sorted_group)
sorted_data = pd.concat(sorted_data)

sorted_data
```

Cette méthode regroupe les données par équipe et les trie par saison. Cela permet de visualiser les performances de chaque équipe au fil des saisons.

3. Voir les informations sur les types des colonnes

```
df.info()
```

Affiche des informations sur les types des colonnes du dataset. Cela permet de vérifier le format des données et de s'assurer qu'elles sont compatibles avec les analyses que vous souhaitez effectuer.

4. Visualisation des statistiques descriptives pour les colonnes numériques

```
df.describe()
```

Affichage des statistiques descriptives pour les colonnes numériques du dataset. Cela permet de connaître la moyenne, la médiane, l'écart-type, etc. des données.

5. Visualisation des statistiques descriptives pour les colonnes numériques par équipe

```
etat_equipe_pour_chaque_season = df.groupby('team').describe()
etat_equipe_pour_chaque_season
```

Affichage des statistiques descriptives pour les colonnes numériques du dataset, regroupées par équipe. Cela permet de comparer les performances des différentes équipes.

6. Nombre de saisons dans le dataset

```
nbr_saisons = df['season'].nunique()
print(f"Nombre de saisons distinctes dans le dataset: {nbr_saisons}")
print(df['season'].unique())
```

Cette instruction affiche le nombre de saisons distinctes dans le dataset. Cela permet de savoir sur combien de saisons les données portent.

7. Nombre d'équipes dans le dataset

```
nbr_equipe = df['team'].nunique()
print(nbr_equipe)
print(df['team'].value_counts())
```

Cette commande affiche le nombre d'équipes distinctes dans le dataset. Cela permet de savoir combien d'équipes sont présentes dans le dataset.

8. Visualisation de l'équipe gagnante de la Premier League pour chaque saison

```
gagnante = df.loc[df.groupby('season')['wins'].idxmax()]

print("equipe gagnante pour chaque saison:")
print(winners[['season', 'team', 'wins']])
```

Identifier l'équipe qui a remporté le plus de matchs pour chaque saison et l'affiche. Cela permet de savoir quelle équipe a été championne de Premier League chaque année.

2.3 - Visualisation des données :

1. Importation des librairies et du jeu de données

```
import matplotlib.pyplot as plt
import seaborn as sns
```

2.Nombre de victoires par équipe et par saison :

Ce barplot permet de visualiser facilement la répartition des victoires par équipe et par saison. On peut identifier les équipes qui se démarquent par leur nombre élevé de victoires au cours des différentes saisons.

```
df = data
plt.figure(figsize=(20, 20))
sns.barplot(x='season', y='wins', hue='team', data=df) # p
plt.title('Nombre de victoires par équipe et par saison')
plt.show()
```

3.Nombre de défaites par équipe et par saison

Ce barplot est similaire au précédent, mais il se concentre sur le nombre de défaites. Il permet de comparer les performances des équipes en termes de défaites et d'identifier celles qui en ont subi le plus.

```
df = data
plt.figure(figsize=(20, 20))
sns.barplot(x='season', y='losses', hue='team', data=df)
plt.title('Nombre de defaites par equipe et par saison')
plt.show()
```

4.Performances des équipes par saison (Victoires):

Ce graphique linéaire permet de suivre l'évolution des victoires de chaque équipe au fil des saisons. On peut observer si certaines équipes ont connu une période de domination ou si leurs performances ont fluctué.

```
df = data
fig1, ax1 = plt.subplots(figsize=(20, 10))
for team, df in df.groupby('team'):
    ax1.plot(df['season'], df['wins'], label=team, marker='o')

plt.title('Performances des equipes par saison')
plt.xlabel('Saison')
plt.ylabel('Nombre de victoires')
plt.legend()
plt.xticks(rotation=45)
plt.grid(True)

plt.tight_layout()
plt.show()
```

5.Performances des équipes par saison (Défaites)

Ce graphique linéaire est similaire au précédent, mais il suit l'évolution des défaites. Il permet de comparer la tendance des défaites subies par chaque équipe au cours des saisons.

```
df = data
fig2, ax2 = plt.subplots(figsize=(20, 10))

for team, df in df.groupby('team'):
    ax2.plot(df['season'], df['losses'], label=team, marker='o')

plt.title('Performances des equipes par saison')
plt.xlabel('Saison')
plt.ylabel('Nombre de losses')
plt.legend()
plt.xticks(rotation=45)
plt.grid(True)

plt.tight_layout()
plt.show()
```

3 – Pré-Processing des données :

3.1 - Nettoyage des Données

1. Identification des colonnes avec des valeurs manquantes :

Affiche du nombre de valeurs manquantes pour chaque colonne du dataset

```
print(data.isnull().sum())
```

2. Sélection des colonnes concernées:

Cette section sélectionne les colonnes contenant des valeurs manquantes. La variable colonnes contient la liste des noms de ces colonnes.

```
valeurs_manquantes = data.isnull().sum()
# filtrons les colonnes qui ont des valeurs manquantes
colonnes = valeurs_manquantes[valeurs_manquantes > 0].index.tolist()
#affichage des colonnes avec des valeurs manquantes
colonnes
```

3. Remplacement des valeurs manquantes par la moyenne par équipe / Vérification des valeurs manquantes restantes :

Cette boucle parcourt chaque colonne avec des valeurs manquantes (colonnes). Pour chaque colonne, elle utilise la fonction “transform” de “groupby” pour calculer la moyenne des valeurs non manquantes par équipe. Ensuite, elle remplace les valeurs manquantes de la colonne par la moyenne calculée pour l'équipe correspondante. Après affichage à nouveau le nombre de valeurs manquantes pour chaque colonne

après le remplacement. On devrait constater que toutes les valeurs manquantes ont été remplacées.

```
for colonne in colonnes:
    data[colonne] = data.groupby('team')[colonne].transform(lambda x: x.fillna(x.mean()))
print(data.isnull().sum())
data
```

4. Suppression des lignes restantes avec des valeurs manquantes (si nécessaire) :

Cette ligne supprime toutes les lignes du dataset qui contiennent encore des valeurs manquantes après le remplacement. Si vous souhaitez conserver ces lignes, vous pouvez commenter cette ligne.En fin afficher le dataset final après le nettoyage

```
data = data.dropna()
print(data.isnull().sum())
```

```
data
```

3.2 - Normalisation, Codage et Préparation des données pour le Machine Learning

1. Import des librairies

```
from sklearn.preprocessing import MinMaxScaler
```

2. Sélection des colonnes:

“colonnes_numerique” contient les noms des colonnes numériques (sauf 'wins').
“colonnes_object” contient les noms des colonnes catégorielles.

```
colonnes_numerique = data.select_dtypes(include=['float64']).columns.drop('wins')
colonnes_object = data.select_dtypes(include=['object']).columns
```

3. Normalisation des données numériques

Cette section utilise MinMaxScaler pour normaliser les valeurs des colonnes numériques dans la plage [0, 1]. Cela permet de mettre toutes les variables numériques sur une échelle commune, ce qui peut être important pour certains algorithmes de machine learning.

```
scaler = MinMaxScaler()
# Normaliser les colonnes numériques
data[colonnes_numerique] = scaler.fit_transform(data[colonnes_numerique])
```

4. Codage one-hot des données catégorielles

Cette ligne utilise la fonction get_dummies de pandas pour coder les données catégorielles en utilisant le codage one-hot. Cela crée de nouvelles colonnes pour

chaque catégorie distincte, avec des valeurs 1 ou 0 indiquant la présence ou l'absence de la catégorie.

```
data = pd.get_dummies(data, columns=colonnes_object, drop_first=True)
data
```

3.2 - Sélection de Caractéristiques pour le Modèle de Prédiction de Premier League

1. Import des données et séparation des ensembles

Les données du dataset de Premier League ont été importées et préparées dans les étapes précédentes. Elles sont maintenant prêtes pour la sélection de caractéristiques.

```
data = pd.get_dummies(data, columns=colonnes_object, drop_first=True)
data
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

2. Affichage des tailles des ensembles

```
# affichage des tailles des ensembles
print("Taille de X_train :", X_train.shape)
print("Taille de X_test :", X_test.shape)
print("Taille de y_train :", y_train.shape)
print("Taille de y_test :", y_test.shape)
```

5 – Modélisation :

5.1 - Choix des Algoriyhmes

Import des librairies

```
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
```

```
lr = LinearRegression()
rf = RandomForestRegressor()
svr = SVR()
```

- **Régression linéaire (lr):** Cet algorithme établit une relation linéaire entre les variables d'entrée (caractéristiques) et la variable cible (nombre de victoires). Il est simple à interpréter mais peut ne pas capturer les relations non linéaires complexes dans les données.

- **Random Forest (rf):** Cet algorithme utilise une forêt d'arbres de décision pour prédire le nombre de victoires. Il est robuste au bruit et aux valeurs aberrantes et peut capturer des relations non linéaires.
- **Support Vector Machines (svr):** Cet algorithme recherche un hyperplan qui sépare au mieux les points de données de différentes classes (victoires et défaites). Il est efficace pour les problèmes de classification et de régression, mais peut être sensible aux valeurs aberrantes.

5.2 - Entrainement des modèles:

Vous avez maintenant entraîné les trois algorithmes de machine learning que vous avez sélectionnés précédemment :
lr: Régression linéaire
rf: Random Forest Regressor
svr: Support Vector Regression

```
lr.fit(X_train, y_train)
rf.fit(X_train, y_train)
svr.fit(X_train, y_train)
```

5.3 - Entrainement des modèles:

1. Import des bibliothèques

```
from sklearn.metrics import mean_squared_error, r2_score
```

2. Prédiction sur l'ensemble de test

```
y_pred_lr = lr.predict(X_test)
y_pred_rf = rf.predict(X_test)
y_pred_svr = svr.predict(X_test)
```

Vous avez utilisé la fonction predict de chaque modèle pour générer des prédictions du nombre de victoires sur l'ensemble de test (X_test). Ces prédictions sont stockées dans les variables y_pred_lr, y_pred_rf et y_pred_svr respectivement.

3. Évaluation des performances

RMSE (Root Mean Squared Error): Le RMSE est une mesure d'erreur qui calcule la racine carrée de l'erreur quadratique moyenne entre les valeurs prédites et les valeurs réelles. Un RMSE plus faible indique une meilleure performance du modèle.
R2 (Coefficient de détermination): Le R2 est une mesure de la proportion de la variance de la variable cible expliquée par le modèle. Un R2 plus proche de 1 indique une meilleure performance du modèle.

```
print("Linear Regression RMSE:", mean_squared_error(y_test, y_pred_lr, squared=False))
print("Random Forest RMSE:", mean_squared_error(y_test, y_pred_rf, squared=False))
print("SVR RMSE:", mean_squared_error(y_test, y_pred_svr, squared=False))

print("Linear Regression R2:", r2_score(y_test, y_pred_lr))
print("Random Forest R2:", r2_score(y_test, y_pred_rf))
print("SVR R2:", r2_score(y_test, y_pred_svr))
```

4. Résultats :

```
Linear Regression RMSE: 2.4275841256888793
Random Forest RMSE: 2.3764012663528447
SVR RMSE: 2.641311352839686
Linear Regression R2: 0.8072035663304067
Random Forest R2: 0.8152476511359261
SVR R2: 0.7717611269101724
```

5. Interprétation des résultats

En fonction des valeurs réelles obtenues pour le RMSE et le R2, vous pouvez interpréter les performances de chaque modèle :

> Le modèle avec le plus faible RMSE et le R2 le plus proche de 1 est susceptible d'être le plus performant pour prédire le nombre de victoires des équipes de Premier League.
> Il est important de comparer les résultats des différents modèles pour identifier celui qui convient le mieux à vos besoins.

Conclusion :

En résumé, cette étude a permis de développer un modèle de machine learning capable de prédire avec précision le nombre de victoires des équipes de Premier League pour une saison donnée.

Le modèle Random Forest s'est avéré être le plus performant parmi les algorithmes testés, obtenant un score RMSE de `2.42` et un R2 de `2.37`.

Ces résultats indiquent que le modèle Random Forest peut être un outil précieux pour les entraîneurs, les managers et les fans de sport qui souhaitent mieux comprendre les facteurs influençant le succès des équipes de Premier League.

Des recherches et des développements supplémentaires sont toutefois nécessaires pour améliorer encore la précision du modèle et explorer d'autres applications potentielles.