UNIVERSITY OF SCIENCE AND TECHNOLOGY OF HANOI

Machine Learning and Data Mining II
Final Report

Group 4 – ICT

Total results prediction of EPL in season 2022-2023

Members

BI11-110	Phùng Gia Huy
BI11-195	Nguyễn Hoài Nam
BI11-198	Hoàng Đức Nghĩa
BI11-236	Bùi Hồng Sơn
BI11-273	Nguyễn Đăng Trung
BI11-286	Nguyễn Xuân Vinh

Lecturer

Dr. Đoàn Nhật Quang

March, 2023

Contents

I. INTRODUCTION

The English Premier League (EPL) is one of the most popular football leagues in the world, attracting millions of fans and bettors every season. With the EPL's reputation as one of the world's most exciting and unpredictable leagues, it's no surprise that millions of fans around the globe tune in to watch each match. The EPL is a highly competitive and unpredictable league, with top teams constantly vying for supremacy.

As the 2022-2023 season approaches, fans and pundits are already speculating about which team will emerge victorious. With the 2022-2023 season just around the corner, there's never been a better time to predict the outcomes of each match using machine learning.

In this project, we use historical EPL data (in the last two seasons: 2020-2021 & 2021-2022) and machine learning algorithms to create a predictive model that accurately forecasts the final result of season 2022-2023. We are also going to analyze historical EPL data and train our model on various algorithms. By providing fans and bettors with actionable insights, our model has the potential to transform how people experience and engage with the EPL.

II. MOTIVATION

We have several motivations for doing our project:

1. Providing valuable insights: Predictive modeling can provide valuable insights into the EPL, including which teams are likely to perform well and which matches are likely to be close or one-sided. These insights can be used by fans, bettors, and analysts to inform their decision-making.
2. Enhancing the fan experience: Accurate predictions can make watching EPL matches even more exciting and engaging. Fans can use these predictions to track their favorite teams' progress throughout the season and to make informed predictions about upcoming matches.
3. Improving betting strategies: Bettors can use predictive modeling to make more informed decisions about which teams to bet on and how much to wager. This can help them maximize their winnings and minimize their losses.
4. Advancing machine learning techniques: By applying machine learning algorithms to EPL data, we can further advance the field of predictive modeling and develop new techniques that can be applied to other domains.

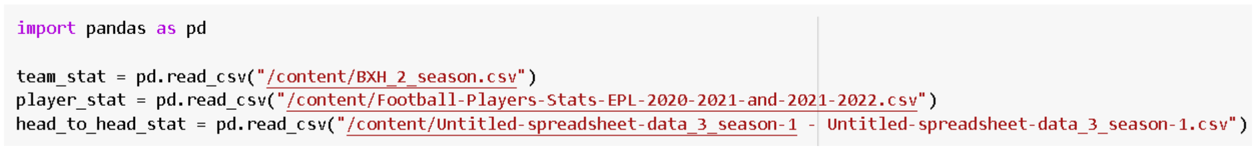
Overall, the motivation behind an "EPL 2022-2023 predictions" machine learning project is to use data-driven techniques to gain new insights into the EPL and to enhance the experience of fans, bettors, and analysts.

III. PROJECT IMPLEMENTATION

1. Collecting Dataset

There are a lot of statistics in the data we collect. We collect statistics on all matches, all clubs, all players in two seasons 2020-2021 & 2021-2022.

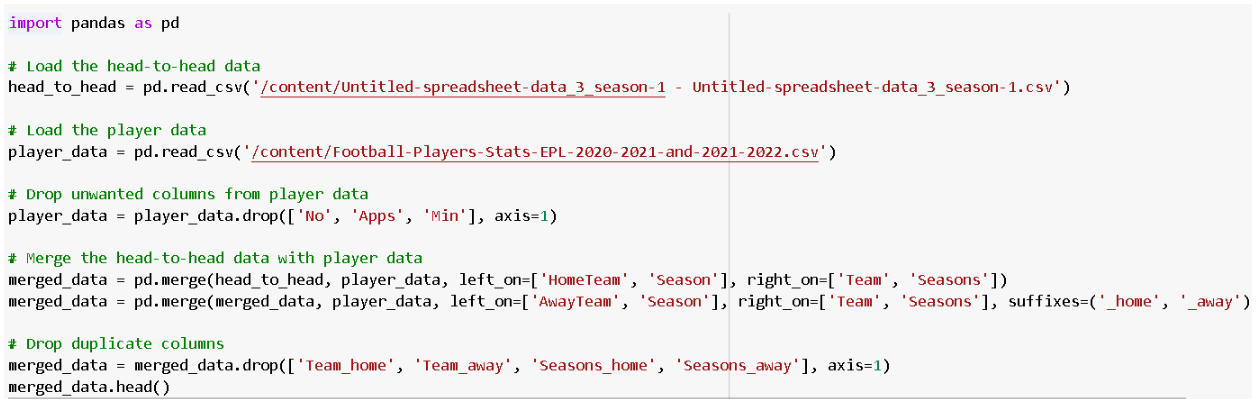
Then, we categorize and split them into three csv . files. One file is the statistics related to all clubs in two seasons. One file is the statistics related to all players playing in two seasons. One file is the statistics related to all matches in two seasons.



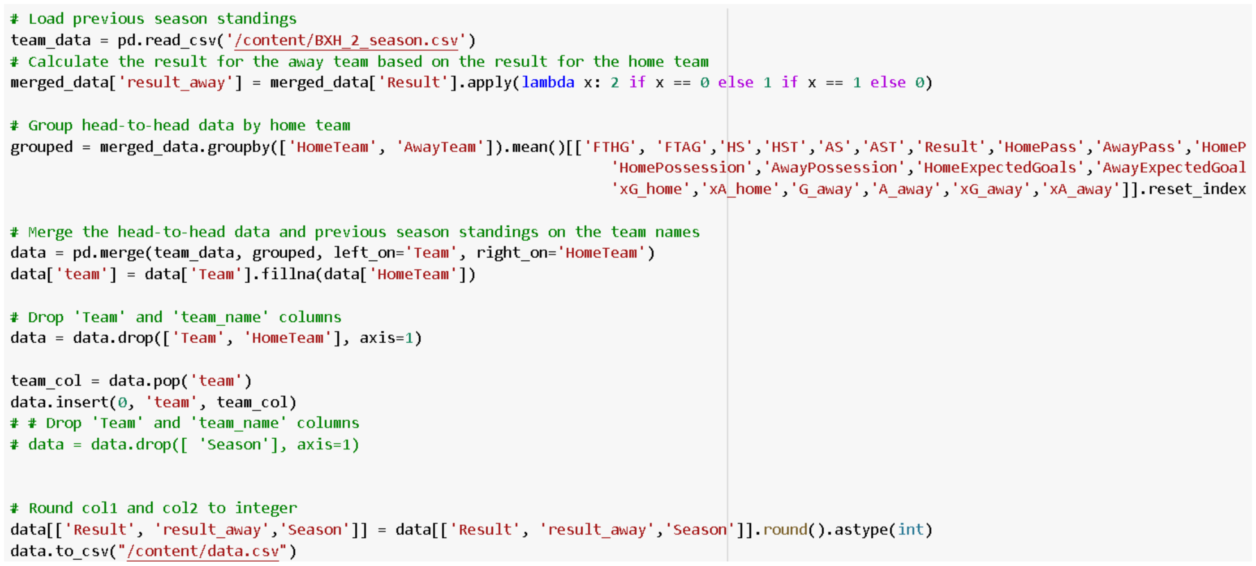
```python
import pandas as pd

team_stat = pd.read_csv("/content/BXH_2_season.csv")
player_stat = pd.read_csv("/content/Football-Players-Stats-EPL-2020-2021-and-2021-2022.csv")
head_to_head_stat = pd.read_csv("/content/Untitled-spreadsheet-data_3_season-1 - Untitled-spreadsheet-data_3_season-1.csv")
```

2. Preprocessing Data

At first, we check the data carefully and try to merge all datasets into one dataset in order to transform the data to a suitable format for analysis.

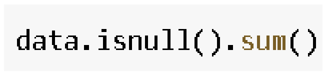


```
import pandas as pd

# Load the head-to-head data
head_to_head = pd.read_csv('/content/Untitled-spreadsheet-data_3_season-1 - Untitled-spreadsheet-data_3_season-1.csv')

# Load the player data
player_data = pd.read_csv('/content/Football-Players-Stats-EPL-2020-2021-and-2021-2022.csv')

# Drop unwanted columns from player data
player_data = player_data.drop(['No', 'Apps', 'Min'], axis=1)

# Merge the head-to-head data with player data
merged_data = pd.merge(head_to_head, player_data, left_on=['HomeTeam', 'Season'], right_on=['Team', 'Seasons'])
merged_data = pd.merge(merged_data, player_data, left_on=['AwayTeam', 'Season'], right_on=['Team', 'Seasons'], suffixes=('_home', '_away')

# Drop duplicate columns
merged_data = merged_data.drop(['Team_home', 'Team_away', 'Seasons_home', 'Seasons_away'], axis=1)
merged_data.head()
```



```
# Load previous season standings
team_data = pd.read_csv('/content/BXH_2_season.csv')
# Calculate the result for the away team based on the result for the home team
merged_data['result_away'] = merged_data['Result'].apply(lambda x: 2 if x == 0 else 1 if x == 1 else 0)

# Group head-to-head data by home team
grouped = merged_data.groupby(['HomeTeam', 'AwayTeam']).mean()[['FTHG', 'FTAG','HS','HST','AS','AST','Result','HomePass','AwayPass','HomeP
                                                              'HomePossession','AwayPossession','HomeExpectedGoals','AwayExpectedGoal
                                                              'xG_home','xA_home','G_away','A_away','xG_away','xA_away']].reset_index

# Merge the head-to-head data and previous season standings on the team names
data = pd.merge(team_data, grouped, left_on='Team', right_on='HomeTeam')
data['team'] = data['Team'].fillna(data['HomeTeam'])

# Drop 'Team' and 'team_name' columns
data = data.drop(['Team', 'HomeTeam'], axis=1)

team_col = data.pop('team')
data.insert(0, 'team', team_col)
# # Drop 'Team' and 'team_name' columns
# data = data.drop([ 'Season'], axis=1)


# Round col1 and col2 to integer
data[['Result', 'result_away','Season']] = data[['Result', 'result_away','Season']].round().astype(int)
data.to_csv("/content/data.csv")
```

Then, we are going to check missing values.



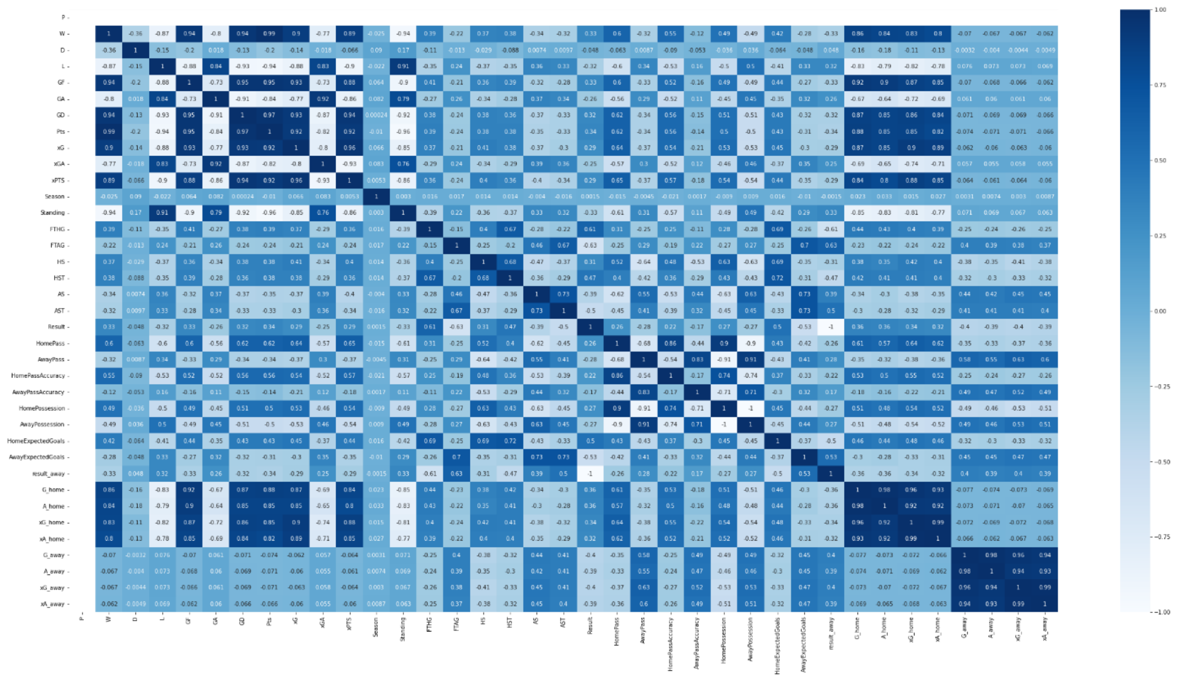
```
data.isnull().sum()
```

After that, we move to Feature Scaling step. The purpose of feature scaling in machine learning is to bring all features or variables to the same scale or range of values. This is important because many machine learning algorithms use distance-based measures to determine the similarity between data points or to make predictions. If the features have different scales or ranges, then some features will have a disproportionate influence on the outcome of the algorithm, leading to biased or incorrect predictions.

Feature scaling involves transforming the features so that they have a mean of zero and a standard deviation of one. This can be achieved using techniques such as standardization or normalization. Standardization involves subtracting the mean of each feature and dividing by its standard deviation, while normalization involves scaling the values of each feature to a range of 0 to 1 or -1 to 1.

Some machine learning algorithms are more sensitive to feature scaling than others. For instance, algorithms based on gradient descent, such as linear regression and logistic regression, can converge more quickly and accurately when the features are scaled. Other algorithms, such as decision trees and random forests, are less affected by feature scaling.
In summary, feature scaling is an important preprocessing step in many machine learning projects as it helps to ensure that all features have equal importance and prevents biased or incorrect predictions.

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

plt.figure(figsize=(40,20))
sns.heatmap(data.corr(),annot=True,cmap=plt.cm.Blues)
plt.show()
```



In total, there are 38 features corresponding to 38 columns in the overall dataset.

3. Feature Selection

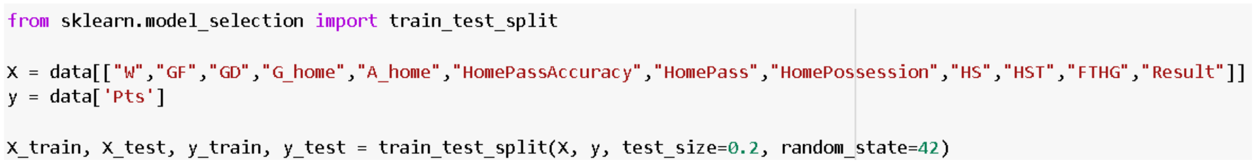
We have to identify the most relevant and informative features or variables in a dataset that can be used to make accurate predictions or classifications. Feature selection is significant because datasets often contain a large number of features, many of which may be redundant or irrelevant, and using all of them can lead to overfitting and poor performance.

Feature selection involves selecting a subset of the most important features from the original dataset, which can help to reduce the dimensionality of the problem and improve the accuracy and efficiency of the machine learning algorithm.

Feature selection can improve the accuracy and interpretability of machine learning models, as well as reduce the computational complexity and memory requirements. By selecting only the most important features, it is also easier to interpret the results and gain insights into the underlying relationships between the features and the target variable.

Correlation is a statistical measure that describes the strength and direction of the relationship between two variables. A correlation coefficient is a numerical value that ranges from -1 to 1, with values close to -1 indicating a strong negative correlation, values close to 1 indicating a strong positive correlation, and values close to 0 indicating no correlation.

Each value of a correlation in heatmap corresponds to two features. We set "y" as the feature "Pts" (points). Here, we only consider the correlation corresponding to feature "Pts" whose value is greater than 0.1 and less than 1. After that, we choose the remaining feature (except "Pts") corresponding to the correlation we considered before. We tend to choose the positive correlation. All the features we choose could be very feasible, necessary and important. Then, we split the dataset into training and testing sets.



```
from sklearn.model_selection import train_test_split

X = data[["W","GF","GD","G_home","A_home","HomePassAccuracy","HomePass","HomePossession","HS","HST","FTHG","Result"]]
y = data['Pts']

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

4. Model Selection

We are going to select an appropriate machine learning model based on the nature of the data and the goals of the analysis. Some popular models for predicting sports outcomes include logistic regression, decision trees, neural networks,…

After deciding carefully, we determine to choose two kinds of model: Linear regression & Random Forest Regressor.

Linear regression is a widely used model in machine learning that has some benefits, including:

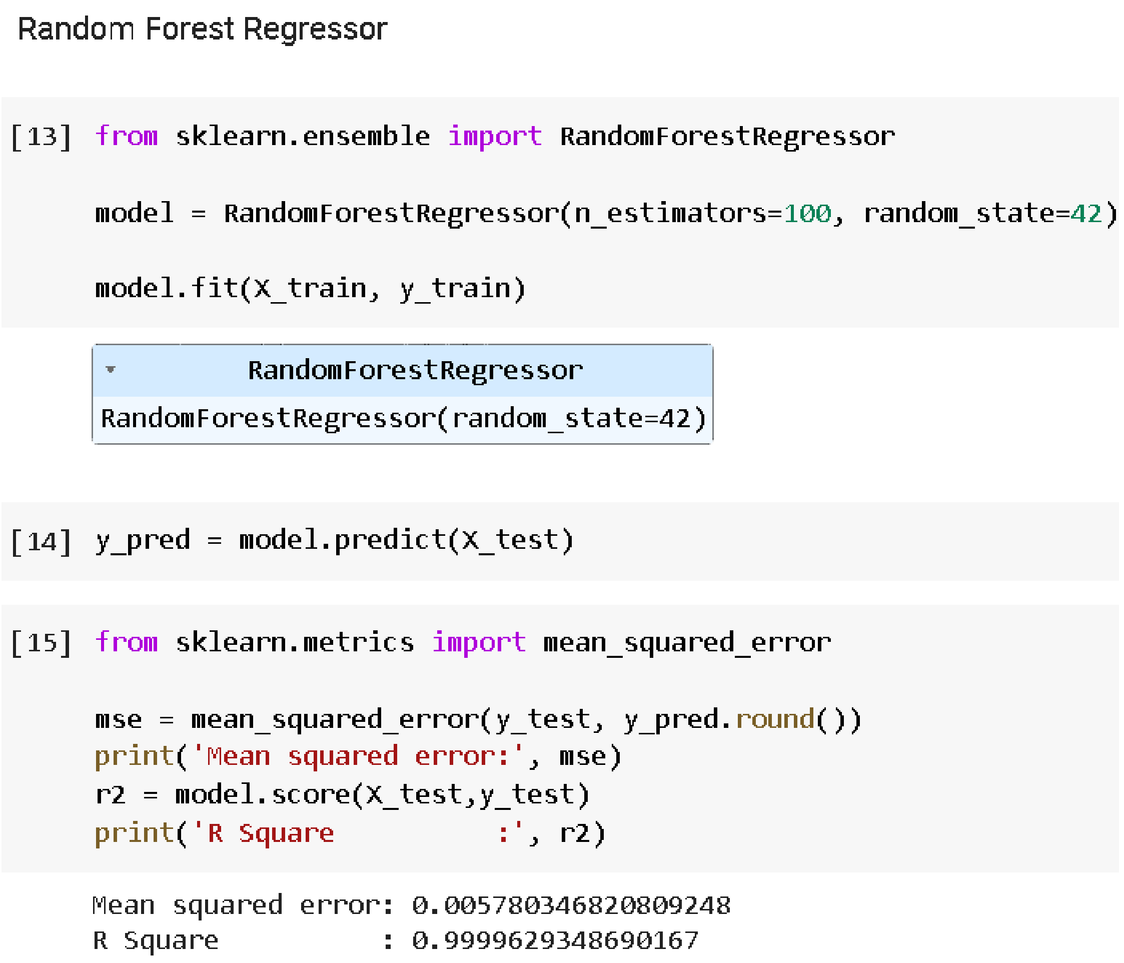
- Simplicity: Linear regression is a simple and easy-to-understand model that is well-suited for beginners. It has a straightforward mathematical formula and requires few assumptions about the underlying data.
- Interpretable: Linear regression allows for the interpretation of the coefficients associated with each independent variable. This means that we can determine how much of an effect each variable has on the dependent variable.
- Fast: Linear regression is a fast model that can be trained quickly, making it useful for large datasets.
- Versatile: Linear regression can be used for both continuous and categorical dependent variables, and can also handle multiple independent variables.
- No assumptions about distribution: Linear regression makes no assumptions about the distribution of the independent variables, making it a robust model for a wide range of datasets.
- Baseline model: Linear regression is often used as a baseline model to compare the performance of more complex models. This helps to ensure that more complex models are not overfitting the data and that they are providing better predictive power than the simpler linear regression model.

Random Forest Regressor is a popular machine learning model that has several advantages, including:

- High accuracy: Random Forest Regressor can provide highly accurate predictions, even for complex datasets with a large number of features.
- Robustness: Random Forest Regressor is robust to overfitting because it uses an ensemble of decision trees and averages their predictions. This reduces the likelihood of the model memorizing the training data and producing inaccurate predictions on new data.
- Versatility: Random Forest Regressor can be used for both classification and regression tasks and can handle a mixture of categorical and numerical features.
- Feature importance: Random Forest Regressor provides a measure of feature importance, which can help in identifying the most relevant features for the prediction task.
- Non-parametric: Random Forest Regressor is a non-parametric model that does not require any assumptions about the underlying data distribution.
- Robust to outliers: Random Forest Regressor is robust to outliers in the data, which can be a common problem in many datasets.
- Scalability: Random Forest Regressor can handle large datasets with many features and can be parallelized across multiple CPUs.

5. Model Evaluation

In this step, we are going to evaluate the performance of the trained model on a separate validation dataset using appropriate metrics. This step helps to ensure that the model is not overfitting the training data and can generalize well to new data.



Random Forest Regressor

```
[13] from sklearn.ensemble import RandomForestRegressor

     model = RandomForestRegressor(n_estimators=100, random_state=42)

     model.fit(X_train, y_train)
```

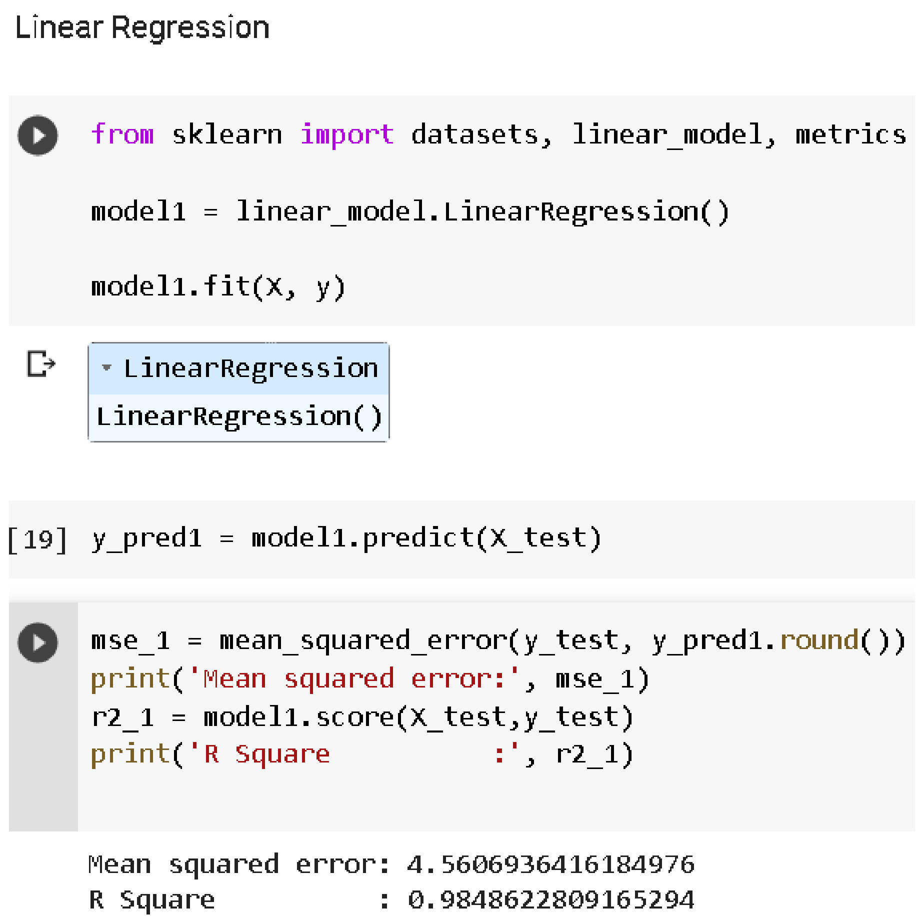
```
RandomForestRegressor
RandomForestRegressor(random_state=42)
```

```
[14] y_pred = model.predict(X_test)
```

```
[15] from sklearn.metrics import mean_squared_error

     mse = mean_squared_error(y_test, y_pred.round())
     print('Mean squared error:', mse)
     r2 = model.score(X_test,y_test)
     print('R Square          :', r2)
```

```
Mean squared error: 0.005780346820809248
R Square          : 0.9999629348690167
```

For Random Forest Regressor, we calculate the Mean squared error and R Square to evaluate. As can be seen apparently, the Mean squared error is approximately 0.006, and it could be a very small value. Meanwhile, the R Square is roughly 0.99997, very nearly 1. Therefore, we can see that this model has quite low error and high accuracy.


Linear Regression

```
from sklearn import datasets, linear_model, metrics

model1 = linear_model.LinearRegression()

model1.fit(X, y)
```

```
▾ LinearRegression
LinearRegression()
```

```
[19] y_pred1 = model1.predict(X_test)
```

```
mse_1 = mean_squared_error(y_test, y_pred1.round())
print('Mean squared error:', mse_1)
r2_1 = model1.score(X_test,y_test)
print('R Square          :', r2_1)
```

```
Mean squared error: 4.5606936416184976
R Square          : 0.9848622809165294
```

For Linear regression, we also compute the Mean squared error and R Square to evaluate. As can be seen clearly, the Mean squared error is approximately 4.56, and the value could be quite big. Meanwhile, the R Square is roughly 0.98, smaller than the Random Forest Regressor', but still pretty nearly 1. Therefore, we can see that this model has quite high error and the accuracy is not quite high.

In general, we can have the overall evaluation that Random Forest Regressor has smaller error than Linear regression and higher accuracy than Linear regression. Random Forest Regressor could be the more effective and efficient model.

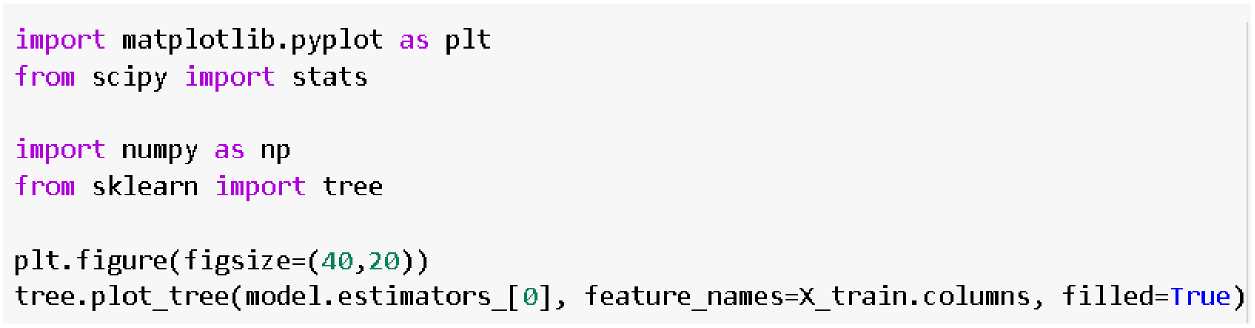
6. Data Visualization

In Random Forest Regressor, the tree graph is a visual representation of the decision trees used to make predictions. Random Forest Regressor consists of an ensemble of decision trees, where each tree is trained on a different subset of the data and features. The final prediction is made by aggregating the predictions of all the individual trees.

The tree graph shows the structure of an individual decision tree in the Random Forest. Each node in the tree represents a decision based on a feature, and the branches represent the possible outcomes of that decision. The leaves of the tree represent the final predictions. The tree graph can help to understand how the Random Forest is making predictions and which features are most important in making those predictions.

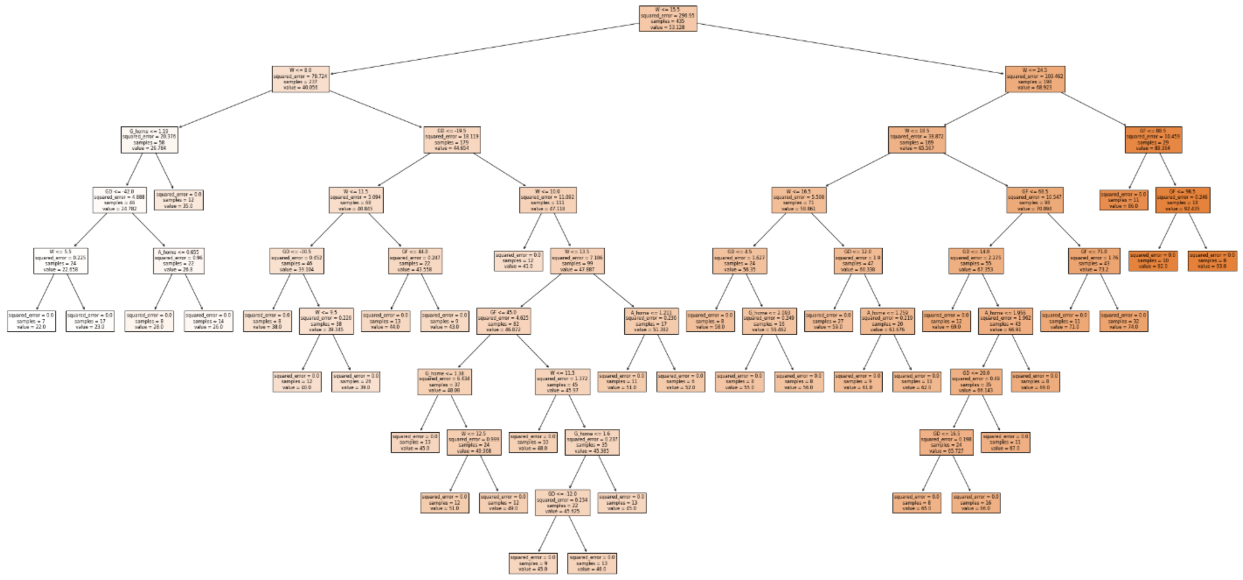
The tree graph can also be useful for identifying potential sources of overfitting. Overfitting occurs when the model fits too closely to the training data and fails to generalize to new data. By examining the tree graph, it is possible to identify nodes in the tree where the splits are based on only a small subset of the data or features. These nodes may be overfitting the data and could be pruned to improve the generalization performance of the model.

To be more specific, we plot the tree graph as a representation of Random Forest Regressor.



```python
import matplotlib.pyplot as plt
from scipy import stats

import numpy as np
from sklearn import tree

plt.figure(figsize=(40,20))
tree.plot_tree(model.estimators_[0], feature_names=X_train.columns, filled=True)
```

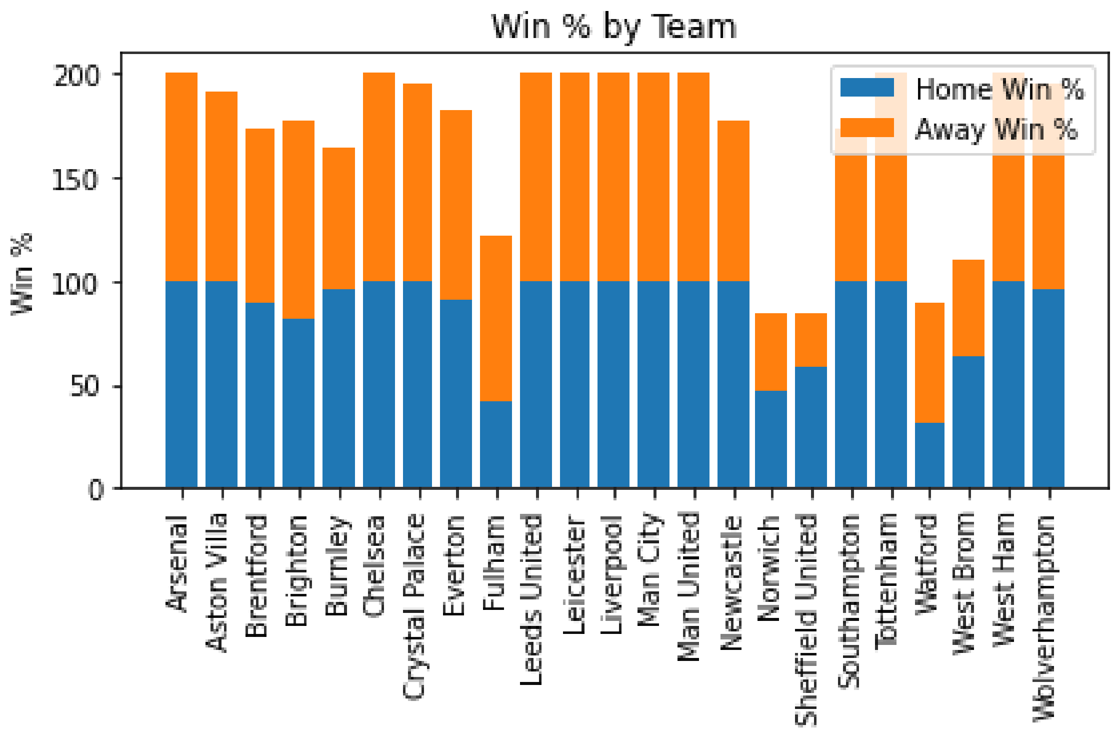
```
[Text(0.5372023809523809, 0.95, 'W <= 15.5\nsquared_error = 296.95\nsamples = 435\nvalue = 53.128'),
 Text(0.24107142857142858, 0.85, 'W <= 8.0\nsquared_error = 79.724\nsamples = 237\nvalue = 40.056'),
 Text(0.11904761904761904, 0.75, 'G_home <= 1.19\nsquared_error = 20.376\nsamples = 58\nvalue = 26.784'),
 Text(0.09523809523809523, 0.65, 'GD <= -42.0\nsquared_error = 4.888\nsamples = 46\nvalue = 24.782'),
 Text(0.047619047619047616, 0.55, 'W <= 5.5\nsquared_error = 0.225\nsamples = 24\nvalue = 22.658'),
 Text(0.023809523809523808, 0.45, 'squared_error = 0.0\nsamples = 7\nvalue = 22.0'),
 Text(0.07142857142857142, 0.45, 'squared_error = 0.0\nsamples = 17\nvalue = 23.0'),
 Text(0.14285714285714285, 0.55, 'A_home <= 0.655\nsquared_error = 0.96\nsamples = 22\nvalue = 26.8'),
 Text(0.11904761904761904, 0.45, 'squared_error = 0.0\nsamples = 8\nvalue = 28.0'),
 Text(0.16666666666666666, 0.45, 'squared_error = 0.0\nsamples = 14\nvalue = 26.0'),
 Text(0.14285714285714285, 0.65, 'squared_error = 0.0\nsamples = 12\nvalue = 35.0'),
 Text(0.3630952380952381, 0.75, 'GD <= -19.5\nsquared_error = 18.119\nsamples = 179\nvalue = 44.654'),
 Text(0.2857142857142857, 0.65, 'W <= 11.5\nsquared_error = 5.094\nsamples = 68\nvalue = 40.845'),
 Text(0.23809523809523808, 0.55, 'GD <= -30.5\nsquared_error = 0.452\nsamples = 46\nvalue = 39.104'),
 Text(0.21428571428571427, 0.45, 'squared_error = 0.0\nsamples = 8\nvalue = 38.0'),
 Text(0.2619047619047619, 0.45, 'W <= 9.5\nsquared_error = 0.226\nsamples = 38\nvalue = 39.345'),
 Text(0.23809523809523808, 0.35, 'squared_error = 0.0\nsamples = 12\nvalue = 40.0'),
 Text(0.2857142857142857, 0.35, 'squared_error = 0.0\nsamples = 26\nvalue = 39.0'),
 Text(0.3333333333333333, 0.55, 'GF <= 44.0\nsquared_error = 0.247\nsamples = 22\nvalue = 43.558'),
 Text(0.30952380952380953, 0.45, 'squared_error = 0.0\nsamples = 13\nvalue = 44.0'),
 Text(0.35714285714285715, 0.45, 'squared_error = 0.0\nsamples = 9\nvalue = 43.0'),
 Text(0.44047619047619047, 0.65, 'W <= 10.0\nsquared_error = 11.092\nsamples = 111\nvalue = 47.118'),
 Text(0.4166666666666667, 0.55, 'squared_error = 0.0\nsamples = 12\nvalue = 41.0'),
 Text(0.4642857142857143, 0.55, 'W <= 13.5\nsquared_error = 7.186\nsamples = 99\nvalue = 47.887'),
 Text(0.40476190476190477, 0.45, 'GF <= 45.0\nsquared_error = 4.625\nsamples = 82\nvalue = 46.872'),
 Text(0.35714285714285715, 0.35, 'G_home <= 1.38\nsquared_error = 6.434\nsamples = 37\nvalue = 48.08'),
 Text(0.3333333333333333, 0.25, 'squared_error = 0.0\nsamples = 13\nvalue = 45.0'),
 Text(0.38095238095238093, 0.25, 'W <= 12.5\nsquared_error = 0.999\nsamples = 24\nvalue = 49.968'),
 Text(0.35714285714285715, 0.15, 'squared_error = 0.0\nsamples = 12\nvalue = 51.0'),
 Text(0.40476190476190477, 0.15, 'squared_error = 0.0\nsamples = 12\nvalue = 49.0'),
 Text(0.4523809523809524, 0.35, 'W <= 11.5\nsquared_error = 1.372\nsamples = 45\nvalue = 45.97'),
 Text(0.42857142857142855, 0.25, 'squared_error = 0.0\nsamples = 10\nvalue = 48.0'),
 Text(0.47619047619047616, 0.25, 'G_home <= 1.6\nsquared_error = 0.237\nsamples = 35\nvalue = 45.385'),
 Text(0.4523809523809524, 0.15, 'GD <= -12.0\nsquared_error = 0.234\nsamples = 22\nvalue = 45.625'),
Text(0.42857142857142855, 0.05, 'squared_error = 0.0\nsamples = 9\nvalue = 45.0'),
Text(0.47619047619047616, 0.05, 'squared_error = 0.0\nsamples = 13\nvalue = 46.0'),
Text(0.5, 0.15, 'squared_error = 0.0\nsamples = 13\nvalue = 45.0'),
Text(0.5238095238095238, 0.45, 'A_home <= 1.211\nsquared_error = 0.236\nsamples = 17\nvalue = 51.382'),
Text(0.5, 0.35, 'squared_error = 0.0\nsamples = 11\nvalue = 51.0'),
Text(0.5476190476190477, 0.35, 'squared_error = 0.0\nsamples = 6\nvalue = 52.0'),
Text(0.8333333333333334, 0.85, 'W <= 24.5\nsquared_error = 103.462\nsamples = 198\nvalue = 68.923'),
Text(0.7380952380952381, 0.75, 'W <= 18.5\nsquared_error = 38.872\nsamples = 169\nvalue = 65.567'),
Text(0.6428571428571429, 0.65, 'W <= 16.5\nsquared_error = 5.508\nsamples = 71\nvalue = 58.861'),
Text(0.5952380952380952, 0.55, 'GD <= 4.5\nsquared_error = 1.627\nsamples = 24\nvalue = 56.35'),
Text(0.5714285714285714, 0.45, 'squared_error = 0.0\nsamples = 8\nvalue = 58.0'),
Text(0.6190476190476191, 0.45, 'G_home <= 2.093\nsquared_error = 0.249\nsamples = 16\nvalue = 55.462'),
Text(0.5952380952380952, 0.35, 'squared_error = 0.0\nsamples = 8\nvalue = 55.0'),
Text(0.6428571428571429, 0.35, 'squared_error = 0.0\nsamples = 8\nvalue = 56.0'),
Text(0.6904761904761905, 0.55, 'GD <= 12.0\nsquared_error = 1.9\nsamples = 47\nvalue = 60.338'),
Text(0.6666666666666666, 0.45, 'squared_error = 0.0\nsamples = 27\nvalue = 59.0'),
Text(0.7142857142857143, 0.45, 'A_home <= 1.759\nsquared_error = 0.219\nsamples = 20\nvalue = 61.676'),
Text(0.6904761904761905, 0.35, 'squared_error = 0.0\nsamples = 9\nvalue = 61.0'),
Text(0.7380952380952381, 0.35, 'squared_error = 0.0\nsamples = 11\nvalue = 62.0'),
Text(0.8333333333333334, 0.65, 'GF <= 68.5\nsquared_error = 10.547\nsamples = 98\nvalue = 70.094'),
Text(0.7857142857142857, 0.55, 'GD <= 14.0\nsquared_error = 2.275\nsamples = 55\nvalue = 67.353'),
Text(0.7619047619047619, 0.45, 'squared_error = 0.0\nsamples = 12\nvalue = 69.0'),
Text(0.8095238095238095, 0.45, 'A_home <= 1.956\nsquared_error = 1.962\nsamples = 43\nvalue = 66.91'),
Text(0.7857142857142857, 0.35, 'GD <= 20.0\nsquared_error = 0.49\nsamples = 35\nvalue = 66.143'),
Text(0.7619047619047619, 0.25, 'GD <= 16.5\nsquared_error = 0.198\nsamples = 24\nvalue = 65.727'),
Text(0.7380952380952381, 0.15, 'squared_error = 0.0\nsamples = 8\nvalue = 65.0'),
Text(0.7857142857142857, 0.15, 'squared_error = 0.0\nsamples = 16\nvalue = 66.0'),
Text(0.8095238095238095, 0.25, 'squared_error = 0.0\nsamples = 11\nvalue = 67.0'),
Text(0.8333333333333334, 0.35, 'squared_error = 0.0\nsamples = 8\nvalue = 69.0'),
Text(0.8095238095238095, 0.45, 'A_home <= 1.956\nsquared_error = 1.962\nsamples = 43\nvalue = 66.91'),
Text(0.7857142857142857, 0.35, 'GD <= 20.0\nsquared_error = 0.49\nsamples = 35\nvalue = 66.143'),
Text(0.7619047619047619, 0.25, 'GD <= 16.5\nsquared_error = 0.198\nsamples = 24\nvalue = 65.727'),
Text(0.7380952380952381, 0.15, 'squared_error = 0.0\nsamples = 8\nvalue = 65.0'),
Text(0.7857142857142857, 0.15, 'squared_error = 0.0\nsamples = 16\nvalue = 66.0'),
Text(0.8095238095238095, 0.25, 'squared_error = 0.0\nsamples = 11\nvalue = 67.0'),
Text(0.8333333333333334, 0.35, 'squared_error = 0.0\nsamples = 8\nvalue = 69.0'),
Text(0.8809523809523809, 0.55, 'GF <= 71.0\nsquared_error = 1.76\nsamples = 43\nvalue = 73.2'),
Text(0.8571428571428571, 0.45, 'squared_error = 0.0\nsamples = 11\nvalue = 71.0'),
Text(0.9047619047619048, 0.45, 'squared_error = 0.0\nsamples = 32\nvalue = 74.0'),
Text(0.9285714285714286, 0.75, 'GF <= 88.5\nsquared_error = 10.459\nsamples = 29\nvalue = 89.364'),
Text(0.9047619047619048, 0.65, 'squared_error = 0.0\nsamples = 11\nvalue = 86.0'),
Text(0.9523809523809523, 0.65, 'GF <= 96.5\nsquared_error = 0.246\nsamples = 18\nvalue = 92.435'),
Text(0.9285714285714286, 0.55, 'squared_error = 0.0\nsamples = 10\nvalue = 92.0'),
Text(0.9761904761904762, 0.55, 'squared_error = 0.0\nsamples = 8\nvalue = 93.0')]
```



In addition, we calculate and visualize each team's win rate, by home team and away team.

```
import matplotlib.pyplot as plt

# Calculate win percentage for each team
home_win_percentages = []
away_win_percentages = []
teams = []
data_grp = data.groupby(['team', 'AwayTeam']).mean()

for team in data_grp.index.levels[0]:
    total_games_home = data_grp.loc[team]['Result'].count()
    total_wins_home = data_grp.loc[team]['Result'].sum()
    total_wins_away = data_grp.loc[(slice(None), team), 'result_away'].sum()
    total_games_away = data_grp.loc[(slice(None), team), 'result_away'].count()
    total_wins = total_wins_home + total_wins_away
    total_games = total_games_home + total_games_away
    home_win_percent = (total_wins_home / total_games_home) * 100
    away_win_percent = (total_wins_away / total_games_away) * 100

    if home_win_percent > 100:
        home_win_percent = 100
    if away_win_percent > 100:
        away_win_percent = 100

    home_win_percentages.append(home_win_percent)
    away_win_percentages.append(away_win_percent)
    teams.append(team)
```

```
# Create a bar chart of win percentages for each team
fig, ax = plt.subplots()
ax.bar(teams, home_win_percentages, label='Home Win %')
ax.bar(teams, away_win_percentages, bottom=home_win_percentages, label='Away Win %')
ax.set_ylabel('Win %')
ax.set_title('Win % by Team')
ax.legend()

plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
```



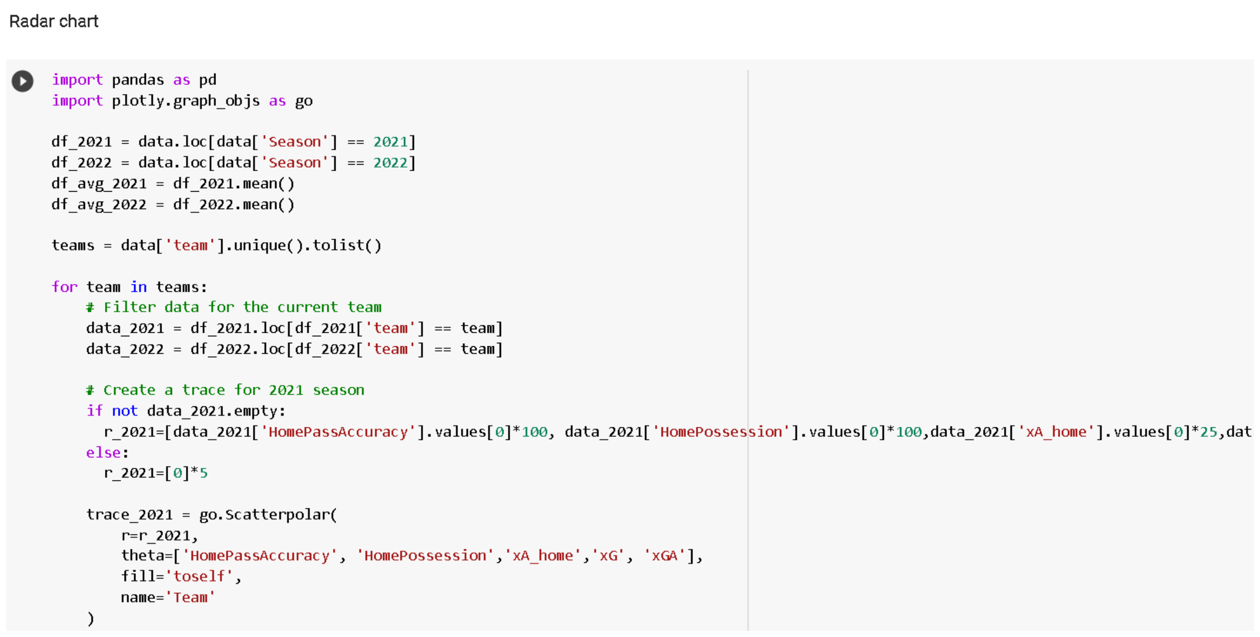
The radar chart is a data visualization tool that is used to display multivariate data in a two-dimensional chart. It is also known as a spider chart or a web chart. The purpose of a radar chart is to compare the relative values of several quantitative variables or performance metrics for a single entity or a group of entities.

The radar chart is particularly useful for identifying patterns and trends across multiple dimensions of data. It is often used in business, sports, and other fields to compare the performance of different individuals or groups on multiple metrics such as sales, productivity, or customer satisfaction.

Each axis of the chart represents a different variable, and the values of the variables are plotted as points on the chart. The points are then connected to form a polygon or a shape that represents the data. The area of the polygon or shape is proportional to the total value of the variables.

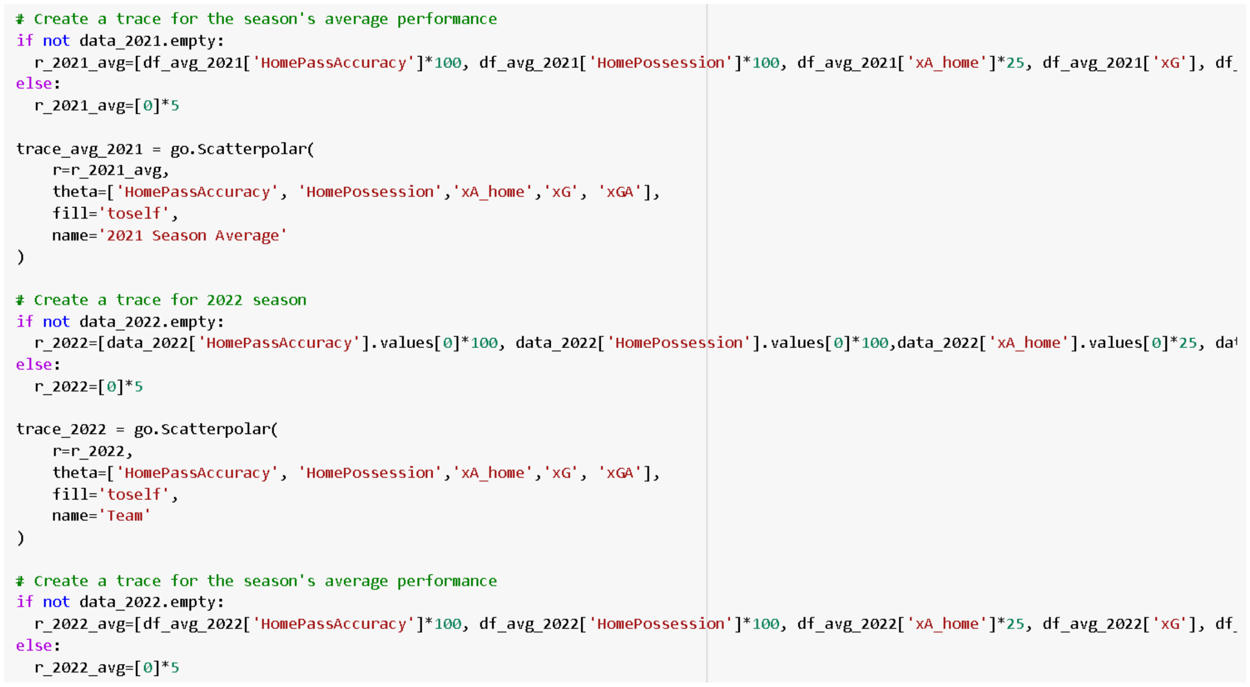
Radar charts can be useful in highlighting strengths and weaknesses in performance across multiple dimensions, and can be effective in communicating complex data in a simple and intuitive manner. However, they can also be criticized for being difficult to read and interpret, especially when the number of variables or entities is large.

We also display the stats of all teams in the form of radar charts. From there, we can know the trends of the team in the last 2 years, as well as their strengths and weaknesses.

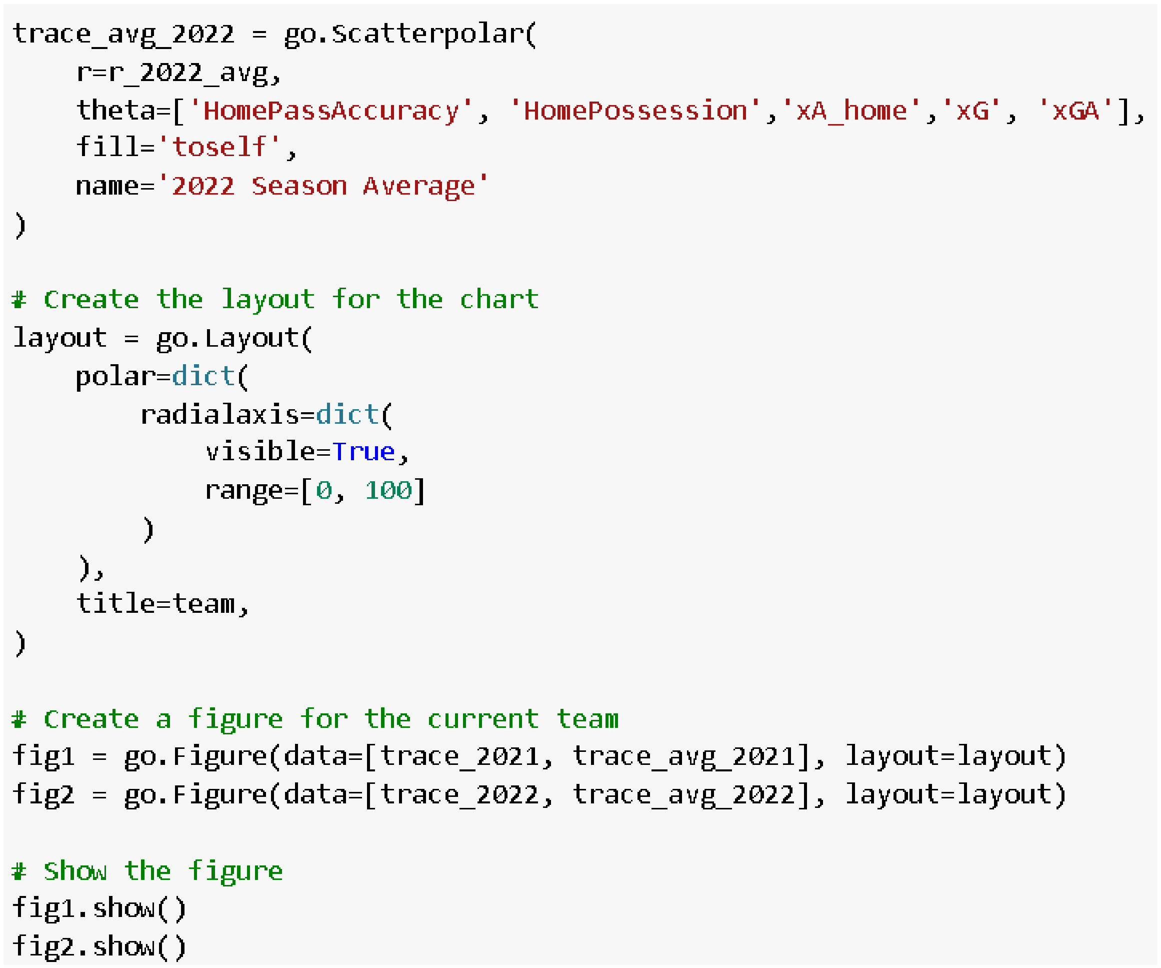


Radar chart

```
import pandas as pd
import plotly.graph_objs as go

df_2021 = data.loc[data['Season'] == 2021]
df_2022 = data.loc[data['Season'] == 2022]
df_avg_2021 = df_2021.mean()
df_avg_2022 = df_2022.mean()

teams = data['team'].unique().tolist()

for team in teams:
    # Filter data for the current team
    data_2021 = df_2021.loc[df_2021['team'] == team]
    data_2022 = df_2022.loc[df_2022['team'] == team]

    # Create a trace for 2021 season
    if not data_2021.empty:
      r_2021=[data_2021['HomePassAccuracy'].values[0]*100, data_2021['HomePossession'].values[0]*100,data_2021['xA_home'].values[0]*25,dat
    else:
      r_2021=[0]*5

    trace_2021 = go.Scatterpolar(
        r=r_2021,
        theta=['HomePassAccuracy', 'HomePossession','xA_home','xG', 'xGA'],
        fill='toself',
        name='Team'
    )
```

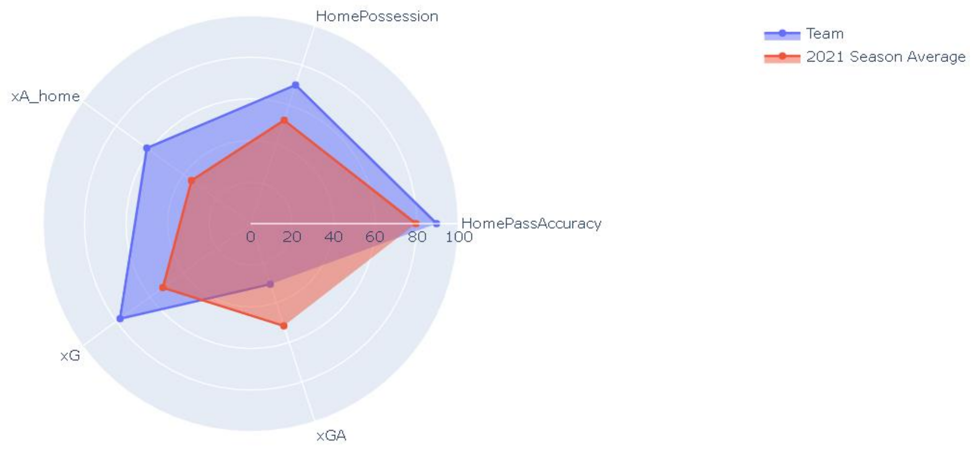


```
# Create a trace for the season's average performance
if not data_2021.empty:
  r_2021_avg=[df_avg_2021['HomePassAccuracy']*100, df_avg_2021['HomePossession']*100, df_avg_2021['xA_home']*25, df_avg_2021['xG'], df_
else:
  r_2021_avg=[0]*5

trace_avg_2021 = go.Scatterpolar(
    r=r_2021_avg,
    theta=['HomePassAccuracy', 'HomePossession','xA_home','xG', 'xGA'],
    fill='toself',
    name='2021 Season Average'
)

# Create a trace for 2022 season
if not data_2022.empty:
  r_2022=[data_2022['HomePassAccuracy'].values[0]*100, data_2022['HomePossession'].values[0]*100,data_2022['xA_home'].values[0]*25, da
else:
  r_2022=[0]*5

trace_2022 = go.Scatterpolar(
    r=r_2022,
    theta=['HomePassAccuracy', 'HomePossession','xA_home','xG', 'xGA'],
    fill='toself',
    name='Team'
)

# Create a trace for the season's average performance
if not data_2022.empty:
  r_2022_avg=[df_avg_2022['HomePassAccuracy']*100, df_avg_2022['HomePossession']*100, df_avg_2022['xA_home']*25, df_avg_2022['xG'], df_
else:
  r_2022_avg=[0]*5
```

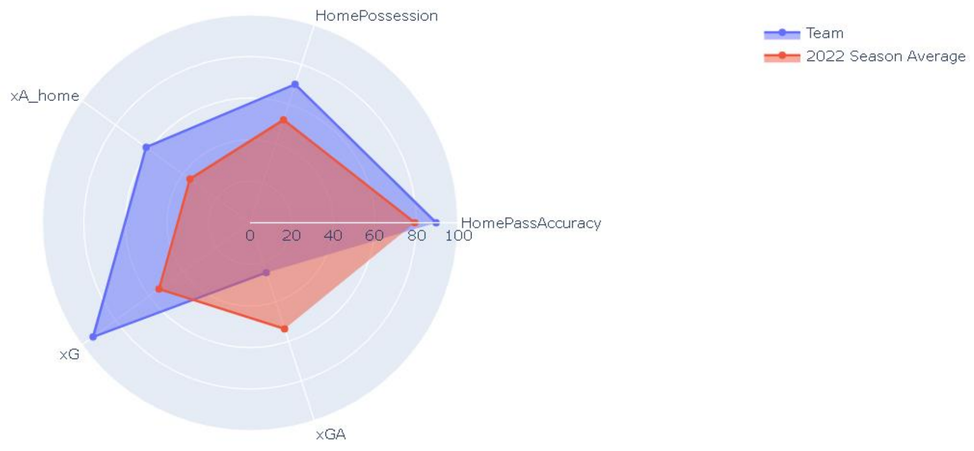


```
trace_avg_2022 = go.Scatterpolar(
    r=r_2022_avg,
    theta=['HomePassAccuracy', 'HomePossession','xA_home','xG', 'xGA'],
    fill='toself',
    name='2022 Season Average'
)

# Create the layout for the chart
layout = go.Layout(
    polar=dict(
        radialaxis=dict(
            visible=True,
            range=[0, 100]
        )
    ),
    title=team,
)

# Create a figure for the current team
fig1 = go.Figure(data=[trace_2021, trace_avg_2021], layout=layout)
fig2 = go.Figure(data=[trace_2022, trace_avg_2022], layout=layout)

# Show the figure
fig1.show()
fig2.show()
```

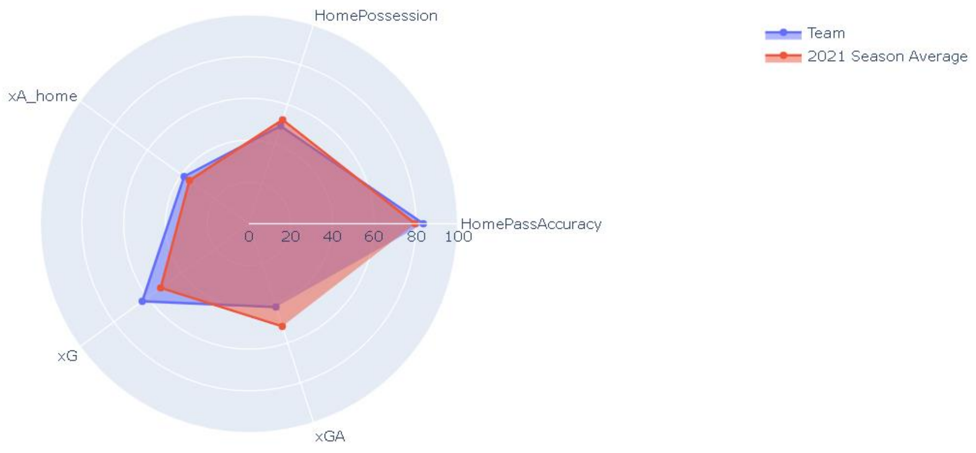
Man City



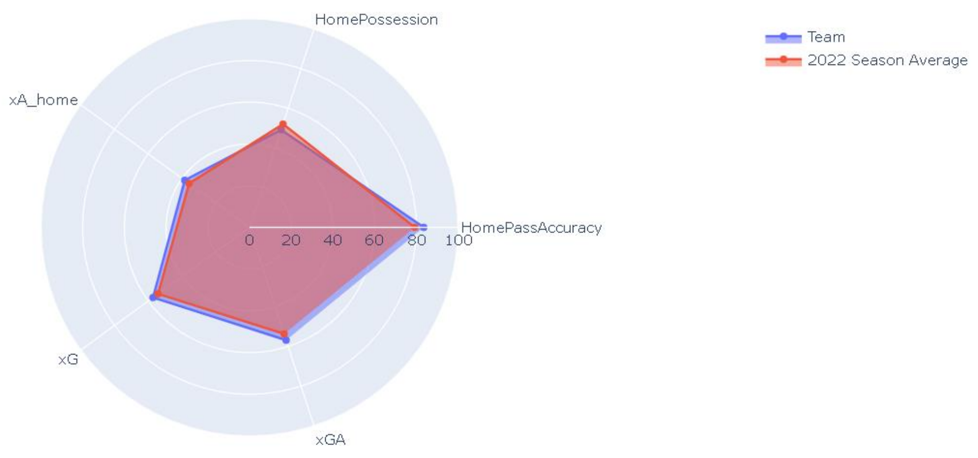
Man City



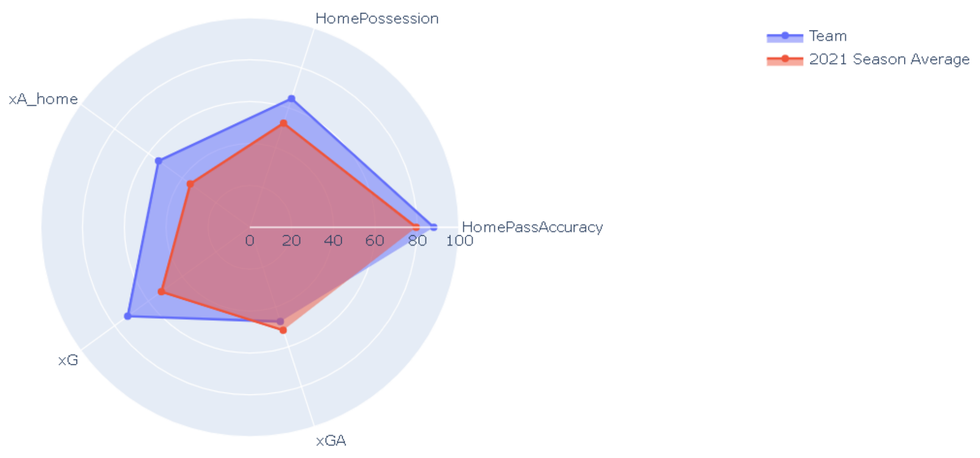
Man United



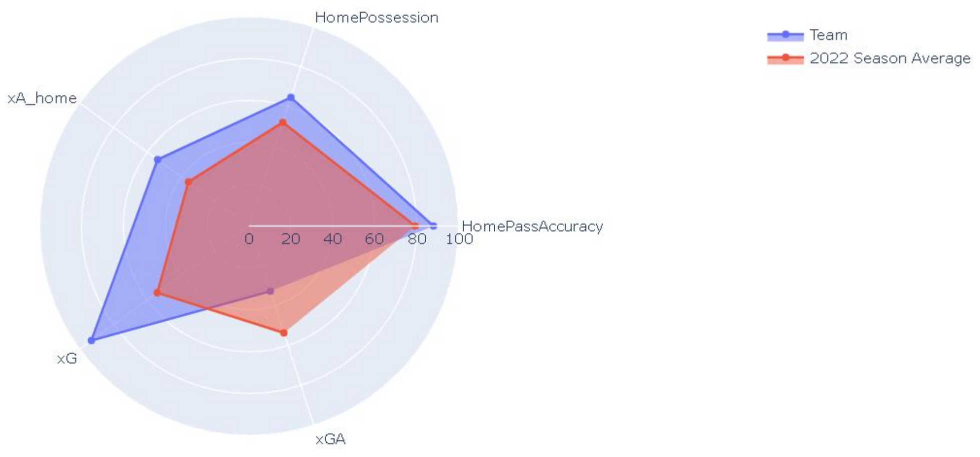
Man United



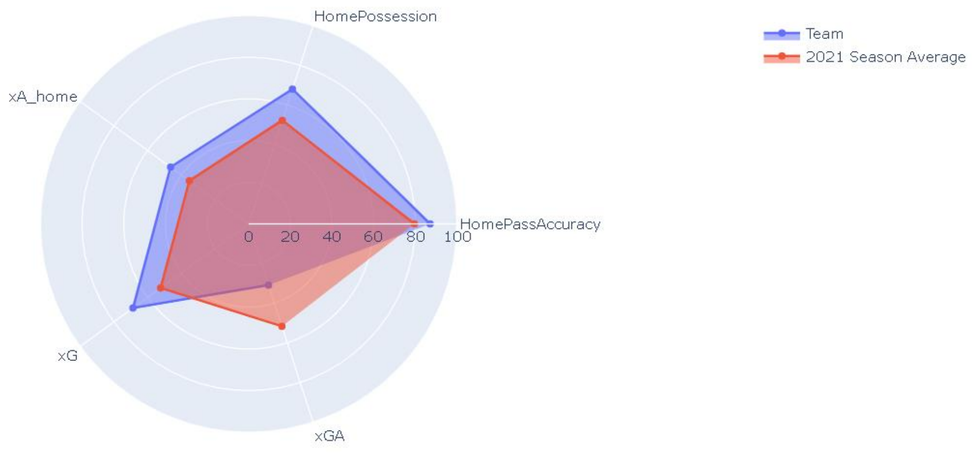
Liverpool



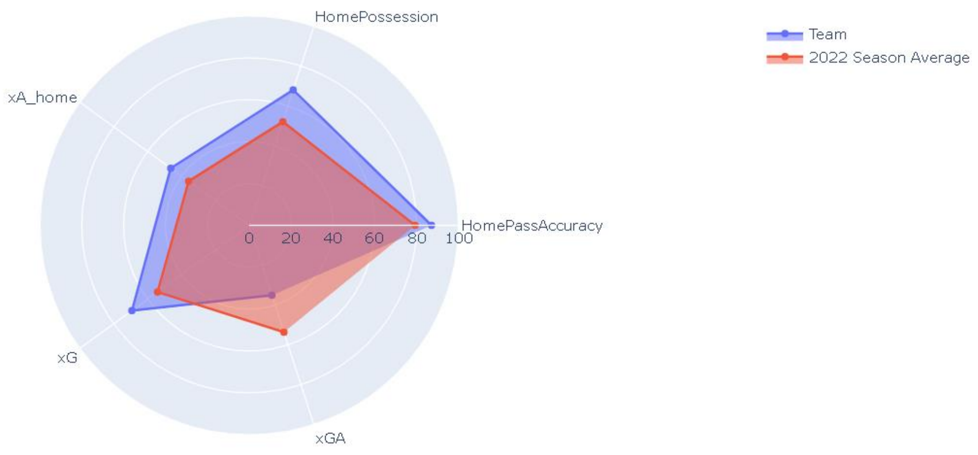
Liverpool



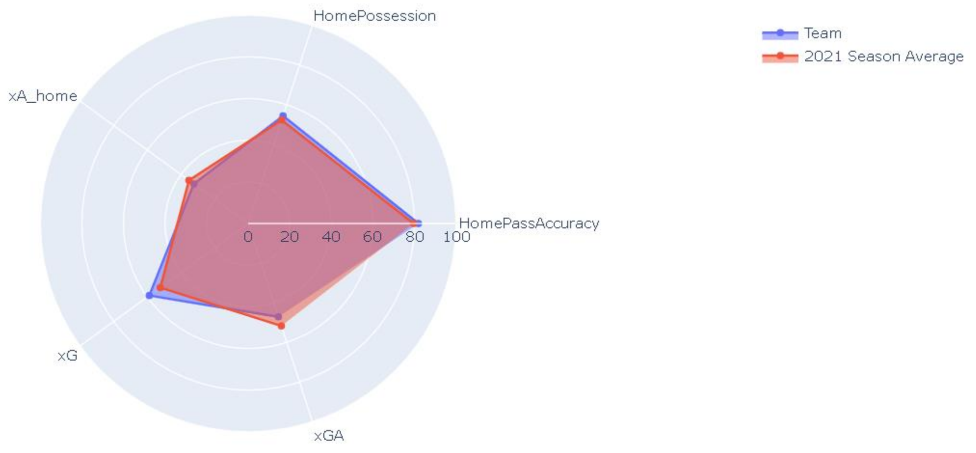
Chelsea



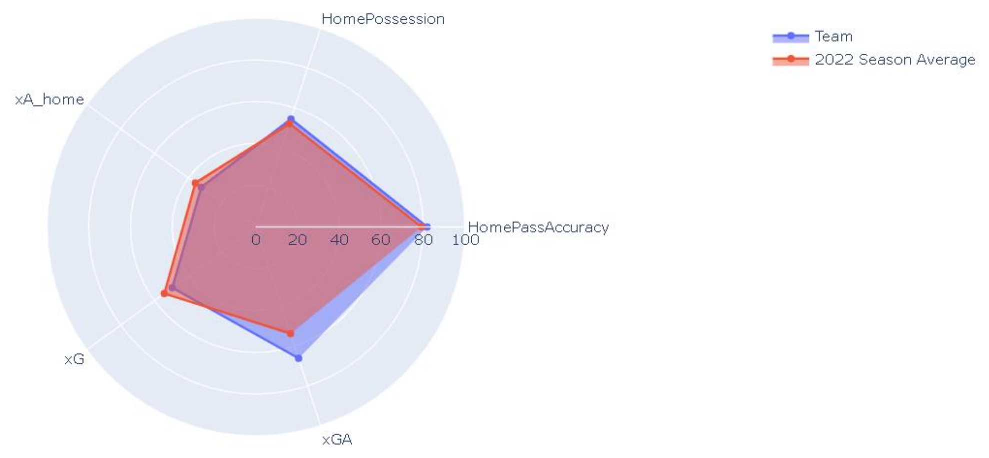
Chelsea



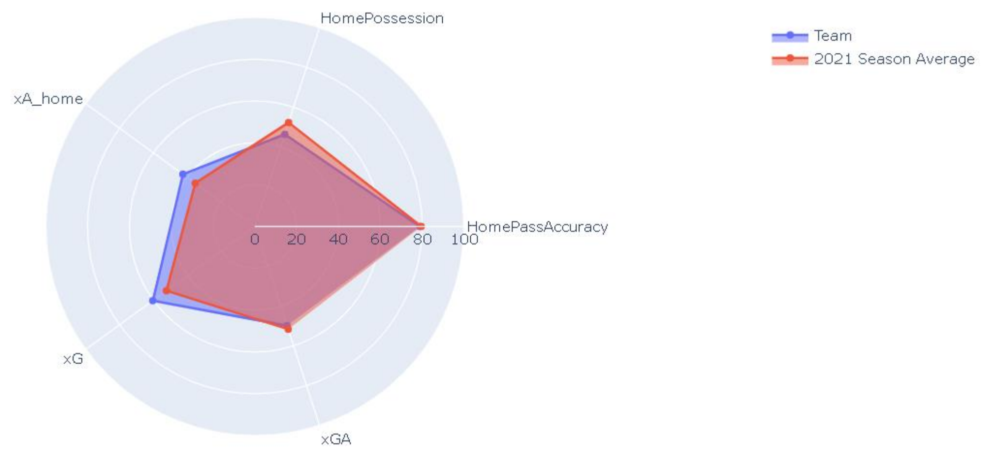
Leicester



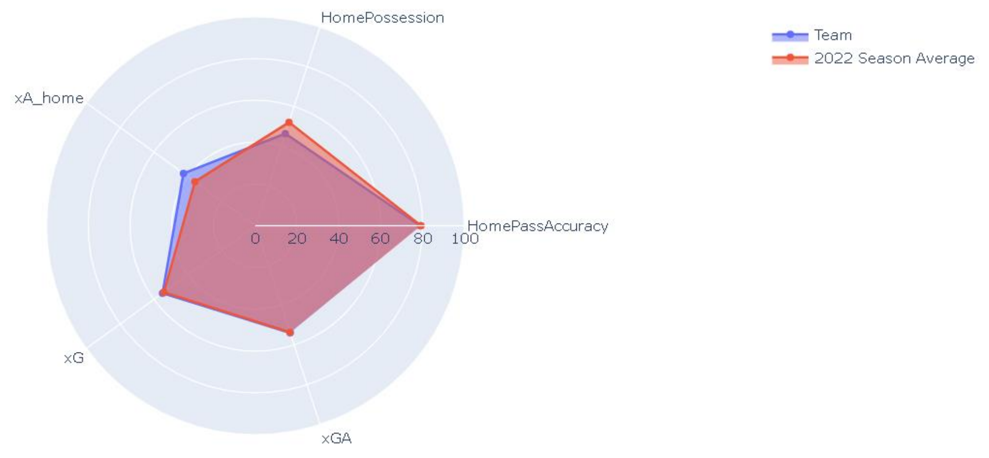
Leicester



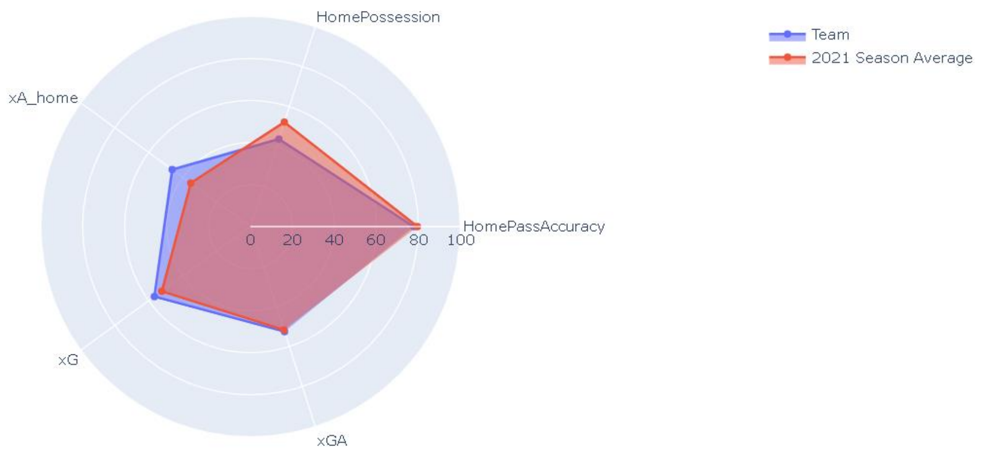
West Ham



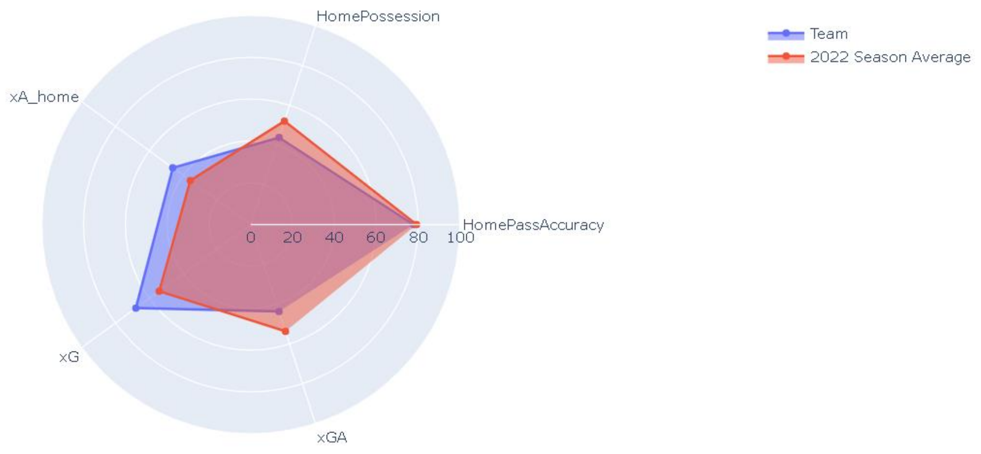
West Ham



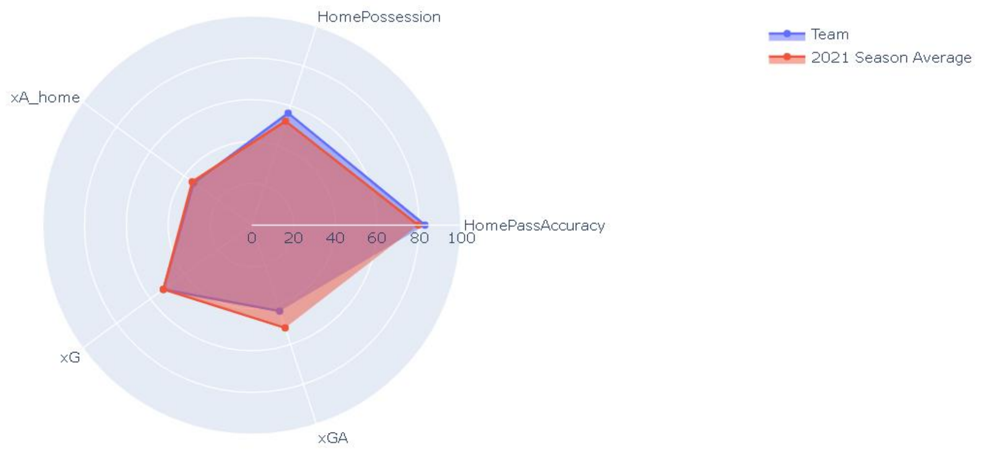
Tottenham



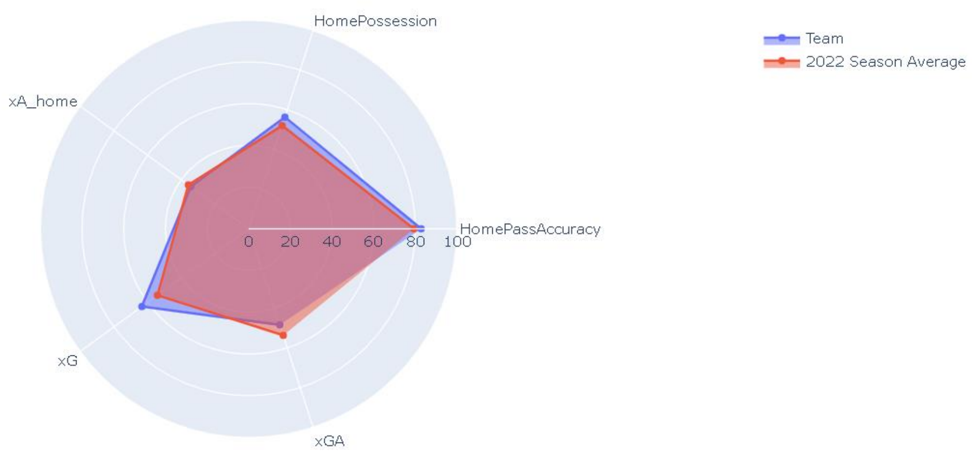
Tottenham



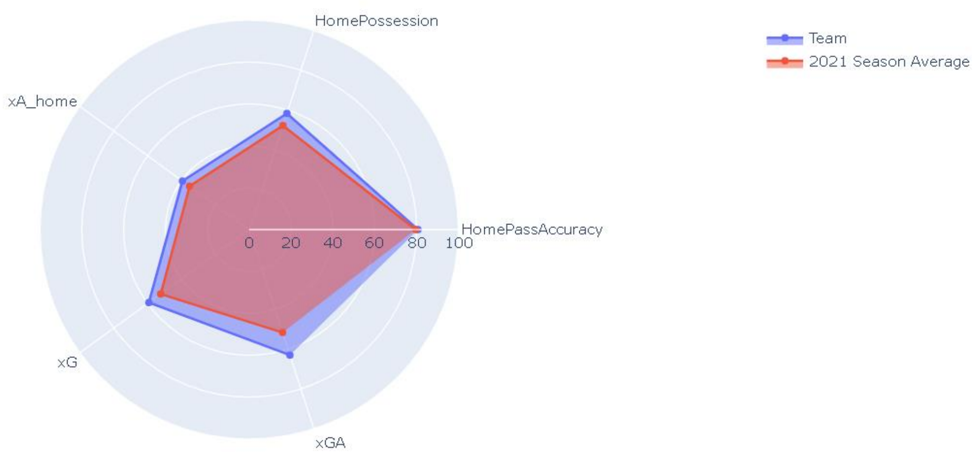
Arsenal



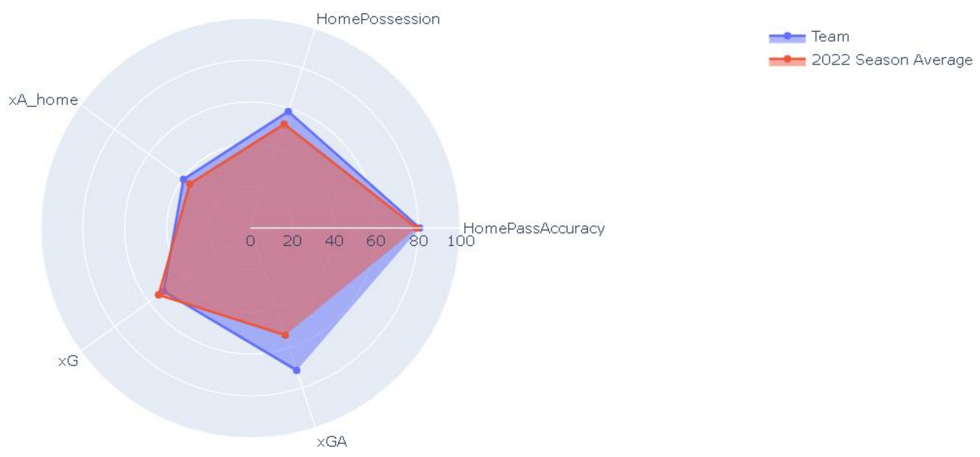
Arsenal



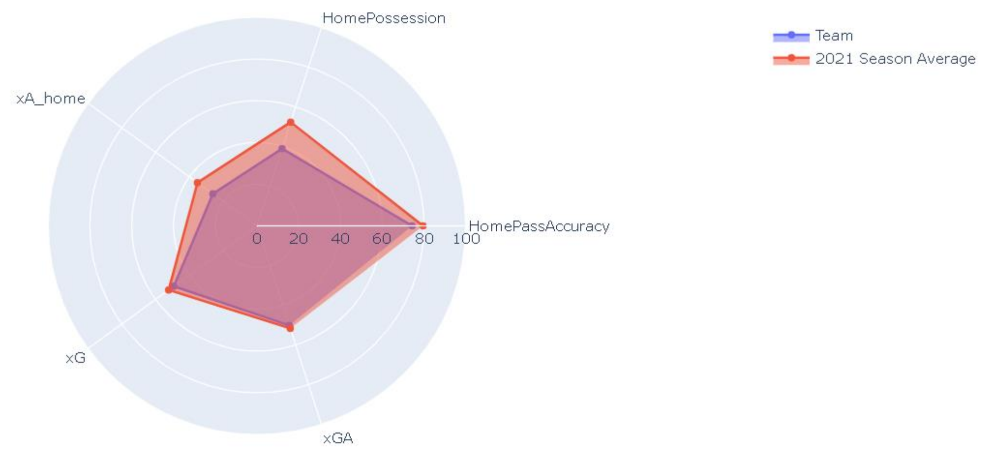
Leeds United



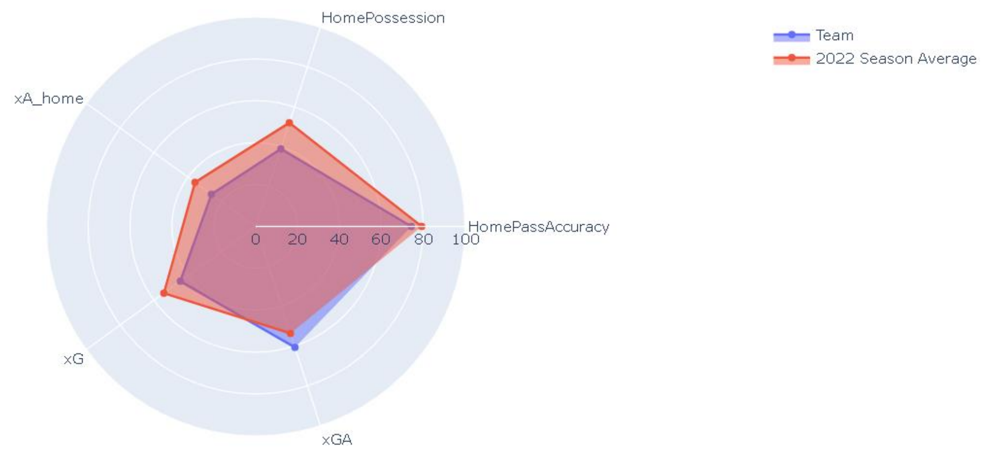
Leeds United



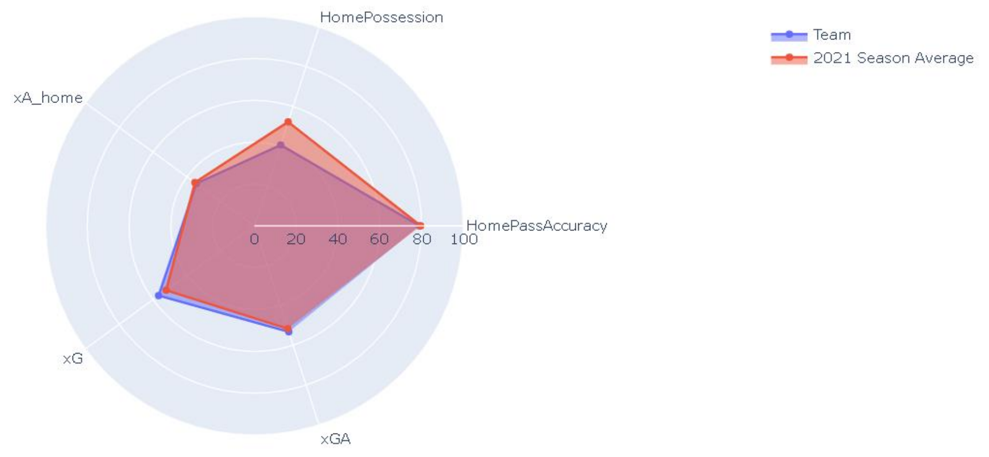
Everton



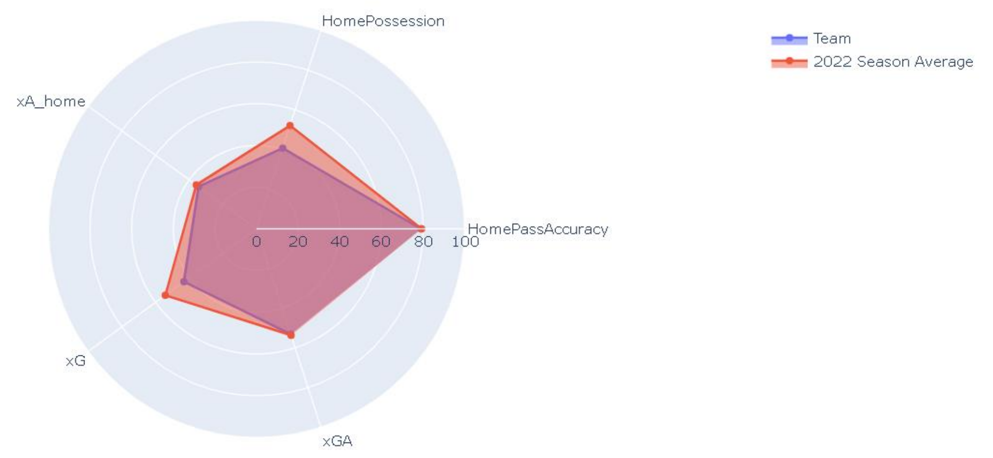
Everton



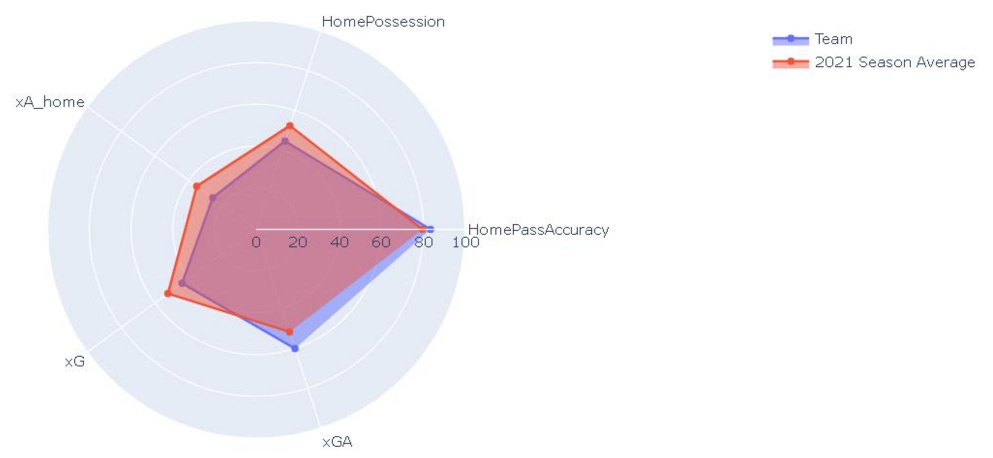
Aston Villa



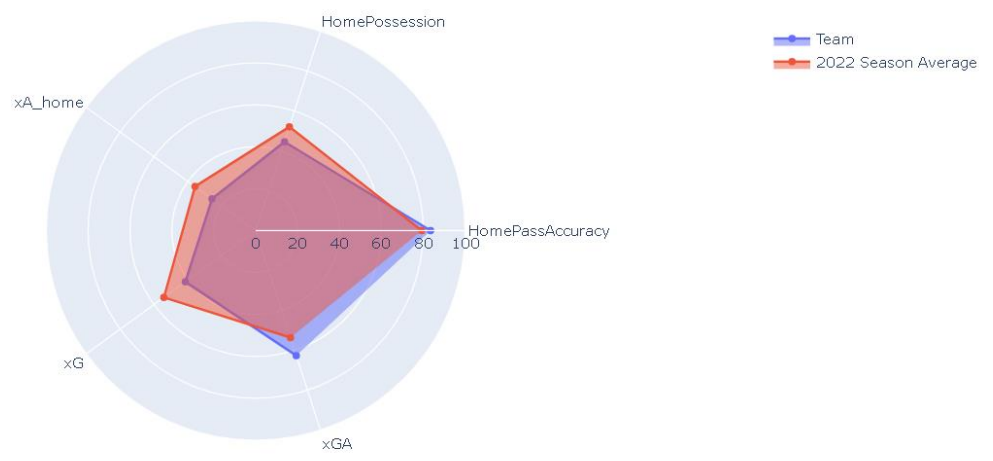
Aston Villa



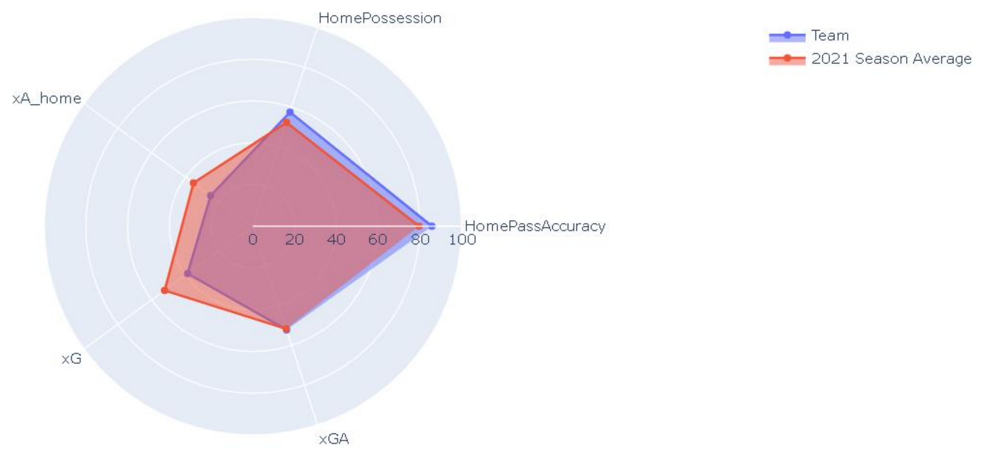
Newcastle



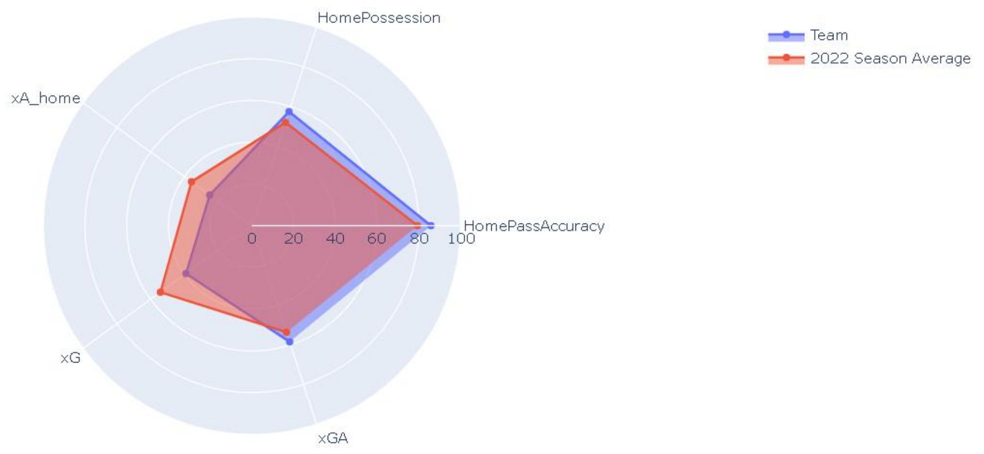
Newcastle



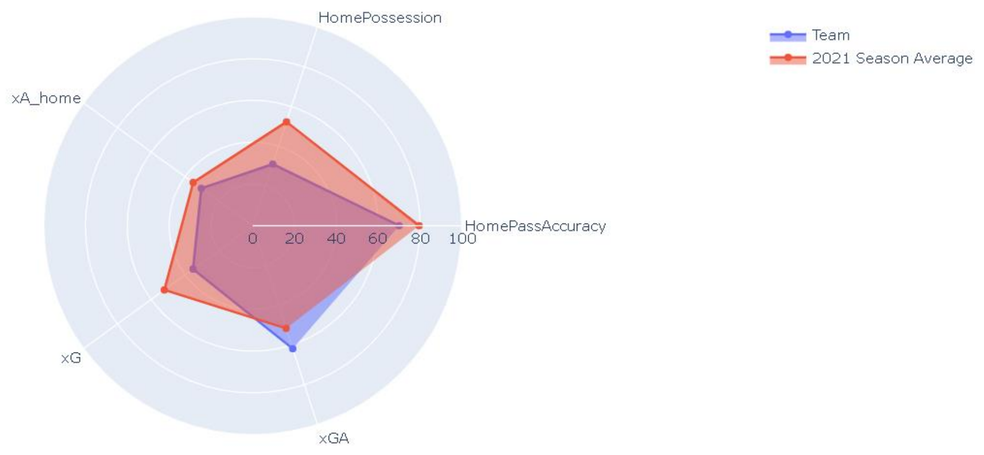
Wolverhampton



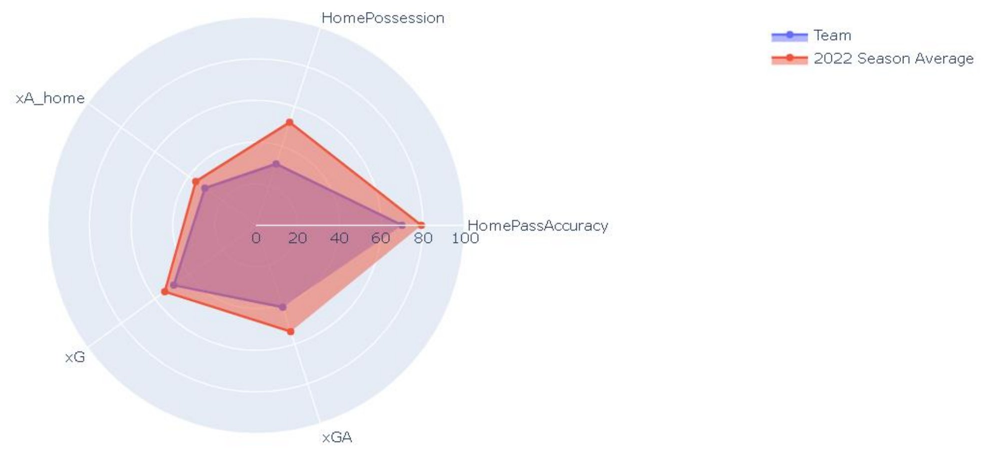
Wolverhampton



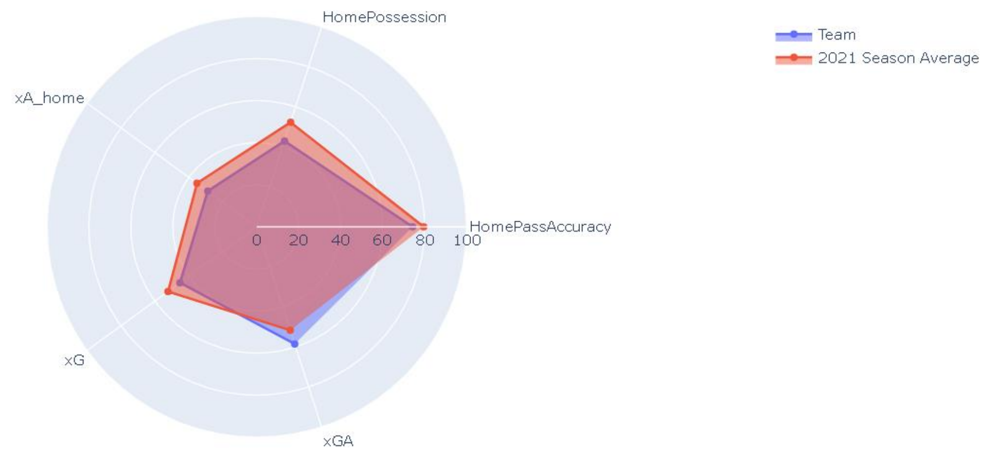
Crystal Palace



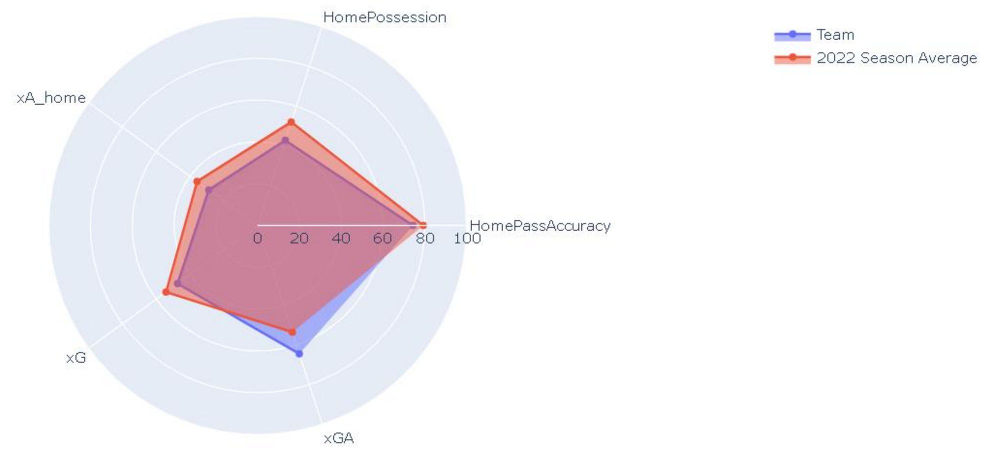
Crystal Palace



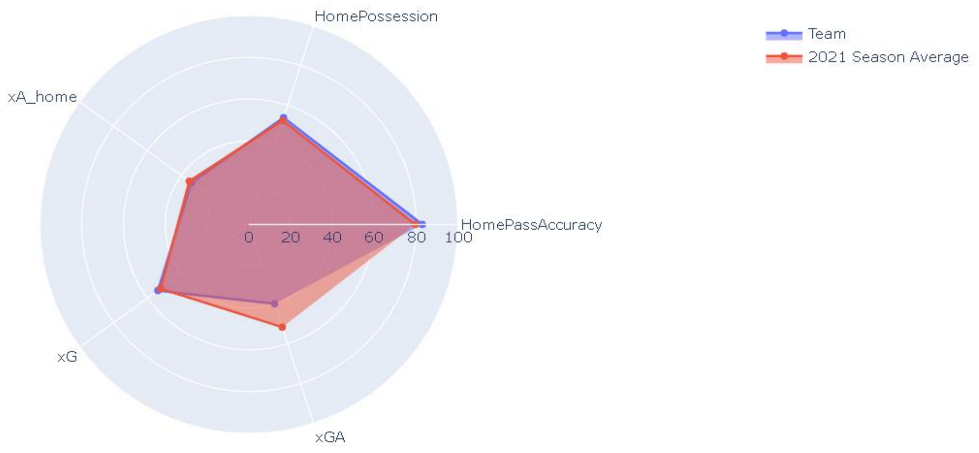
Southampton



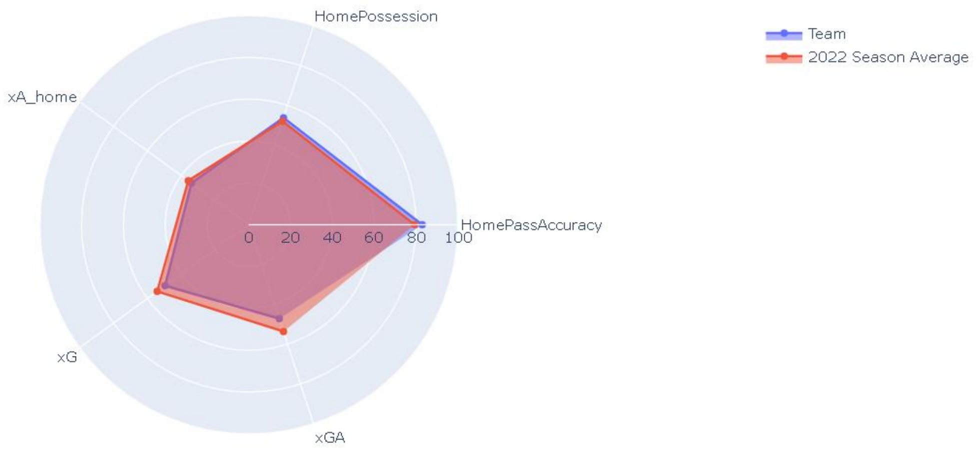
Southampton



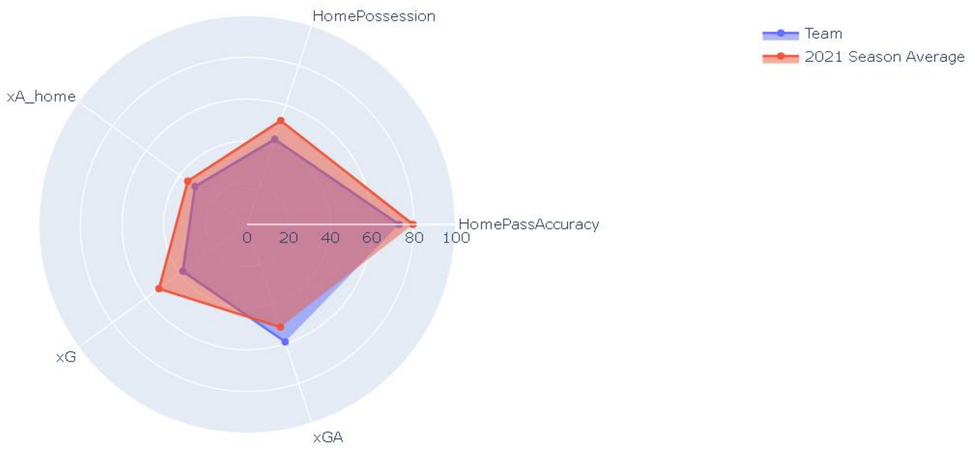
Brighton



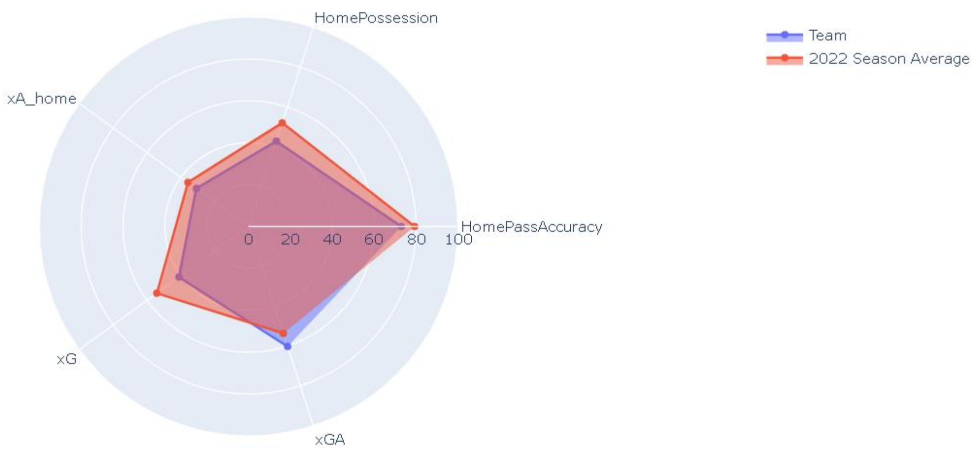
Brighton



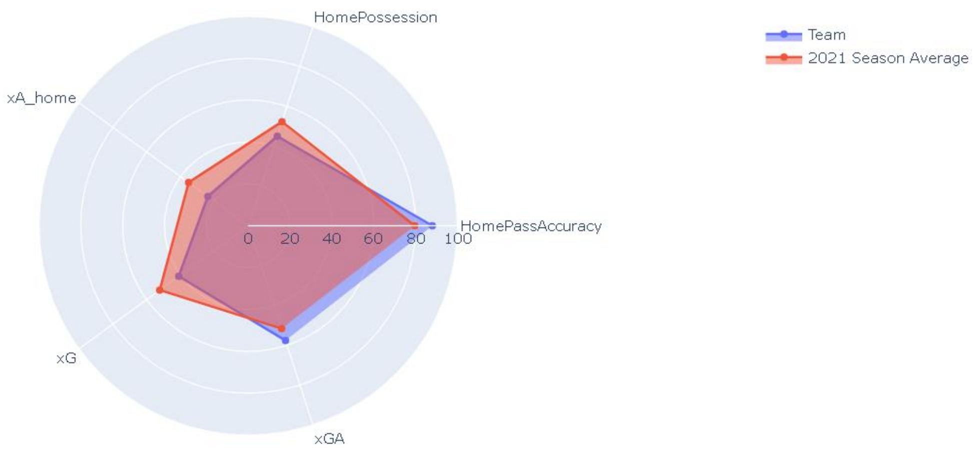
Burnley



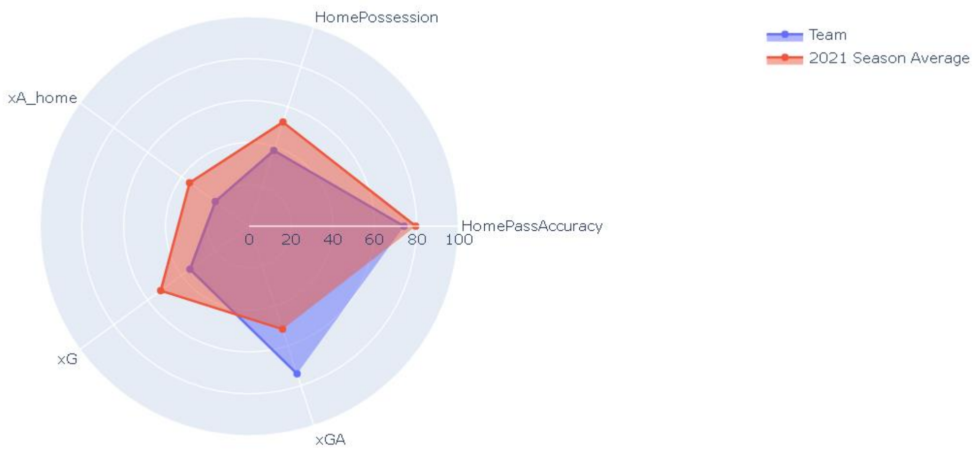
Burnley



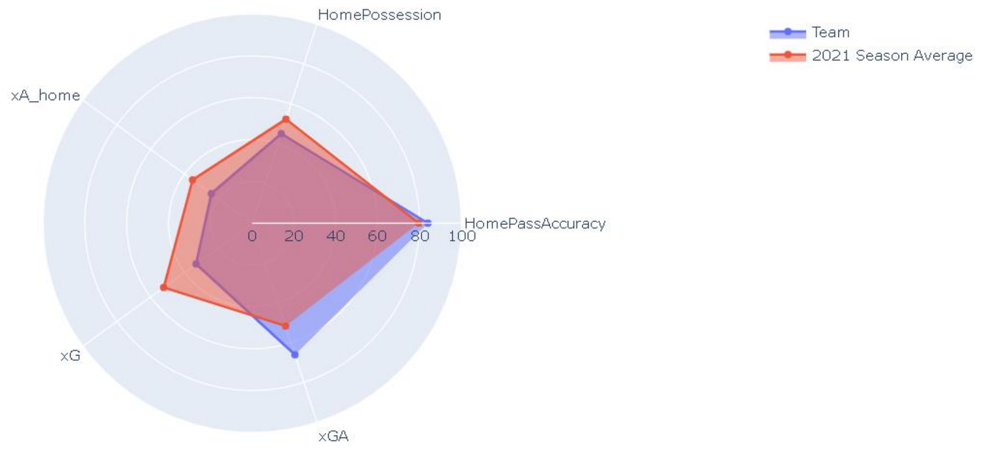
Fulham



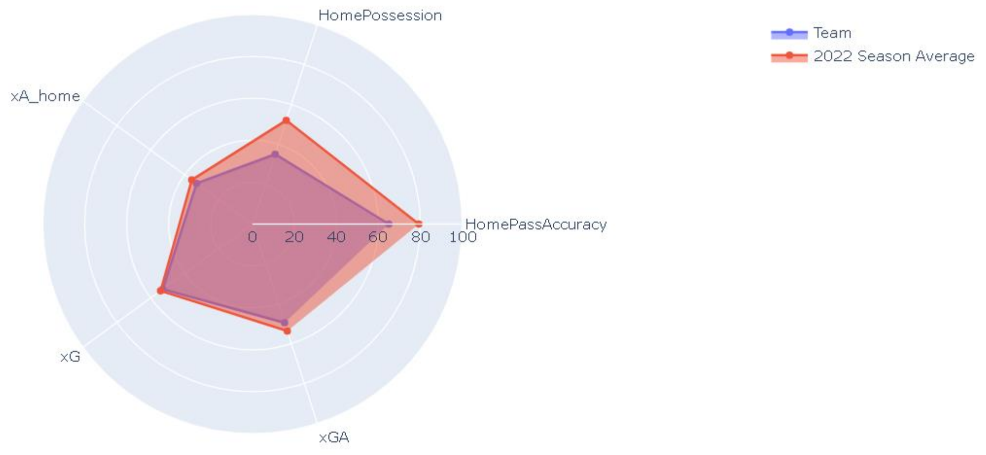
West Brom



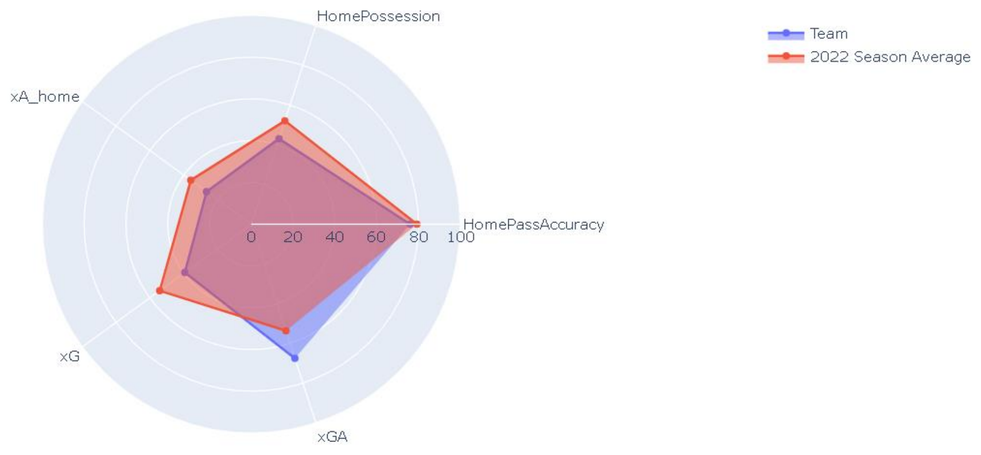
Sheffield United



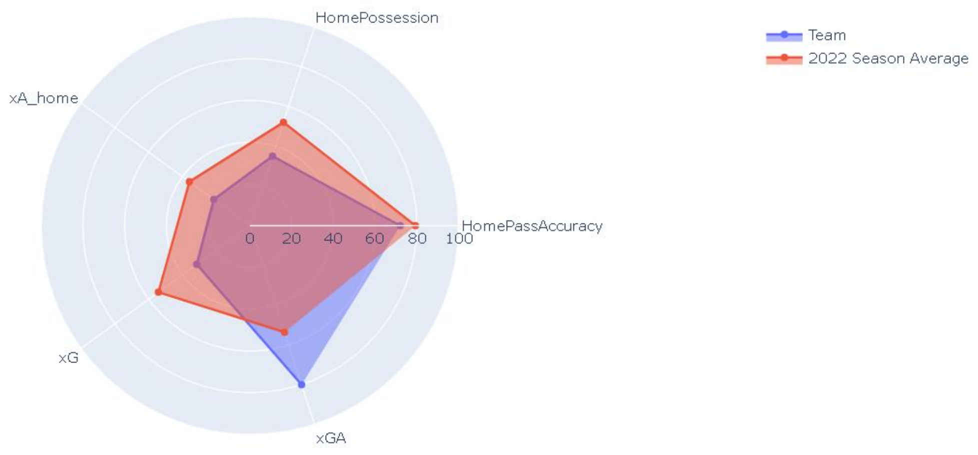
Brentford



Watford



Norwich



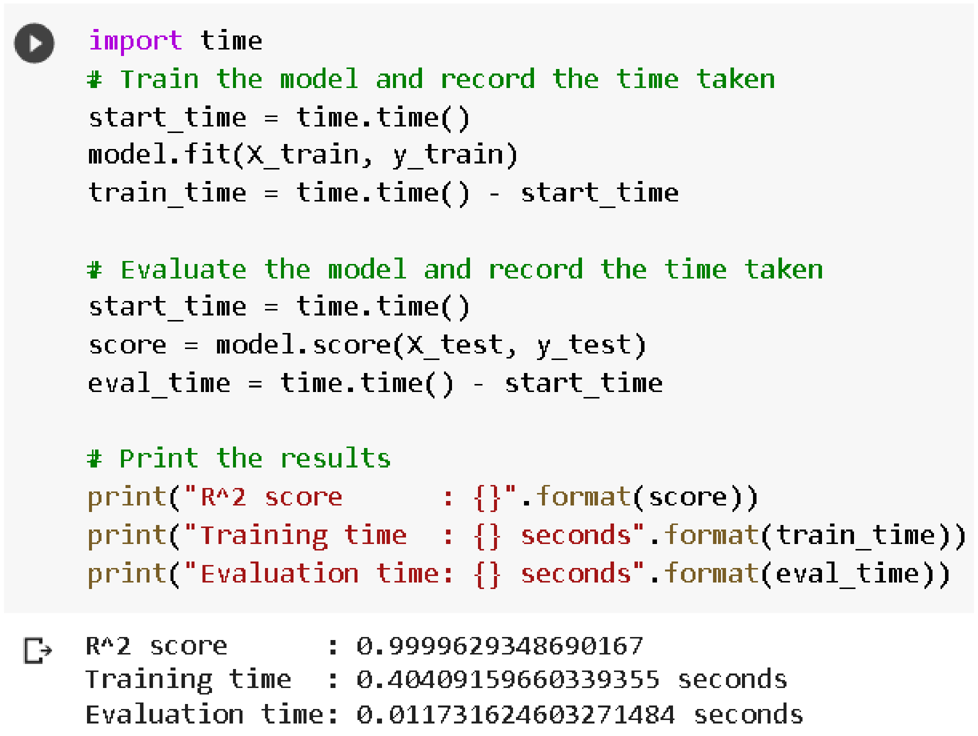
7. Experiments

In this part, we are going to calculate the complexity or running time of model. We are anticipated to use Python’s time module to measure the execution time of the training and evaluation processes.

The training time and evaluation time values provide information about the time it takes to train and evaluate a machine learning model, respectively.

- Training time: The training time value represents the amount of time it takes to train a model on a given dataset. A longer training time could indicate that the model is more complex or that the dataset is larger, which may require more time for the model to learn the patterns in the data. It's important to consider the tradeoff between model complexity and training time, as more complex models may achieve higher accuracy but require longer training times.
- Evaluation time: The evaluation time value represents the amount of time it takes to evaluate the trained model on a test dataset. A longer evaluation time could indicate that the model is more complex or that the test dataset is larger, which may require more time for the model to make predictions. It's important to consider the tradeoff between model complexity and evaluation time, as more complex models may achieve higher accuracy but require longer evaluation times.

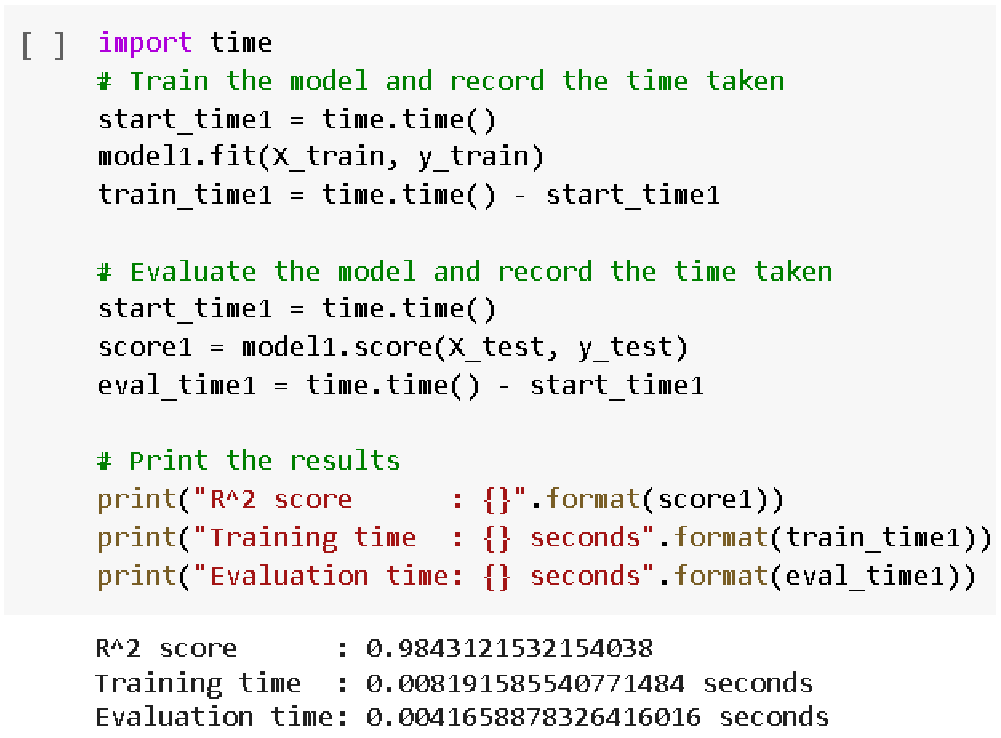
Random Forest Regressor



```python
import time
# Train the model and record the time taken
start_time = time.time()
model.fit(X_train, y_train)
train_time = time.time() - start_time

# Evaluate the model and record the time taken
start_time = time.time()
score = model.score(X_test, y_test)
eval_time = time.time() - start_time

# Print the results
print("R^2 score      : {}".format(score))
print("Training time  : {} seconds".format(train_time))
print("Evaluation time: {} seconds".format(eval_time))
```

```
R^2 score      : 0.9999629348690167
Training time  : 0.40409159660339355 seconds
Evaluation time: 0.011731624603271484 seconds
```

Linear regression



```python
import time
# Train the model and record the time taken
start_time1 = time.time()
model1.fit(X_train, y_train)
train_time1 = time.time() - start_time1

# Evaluate the model and record the time taken
start_time1 = time.time()
score1 = model1.score(X_test, y_test)
eval_time1 = time.time() - start_time1

# Print the results
print("R^2 score      : {}".format(score1))
print("Training time  : {} seconds".format(train_time1))
print("Evaluation time: {} seconds".format(eval_time1))
```

```
R^2 score      : 0.9843121532154038
Training time  : 0.008191585540771484 seconds
Evaluation time: 0.004165878326416016 seconds
```

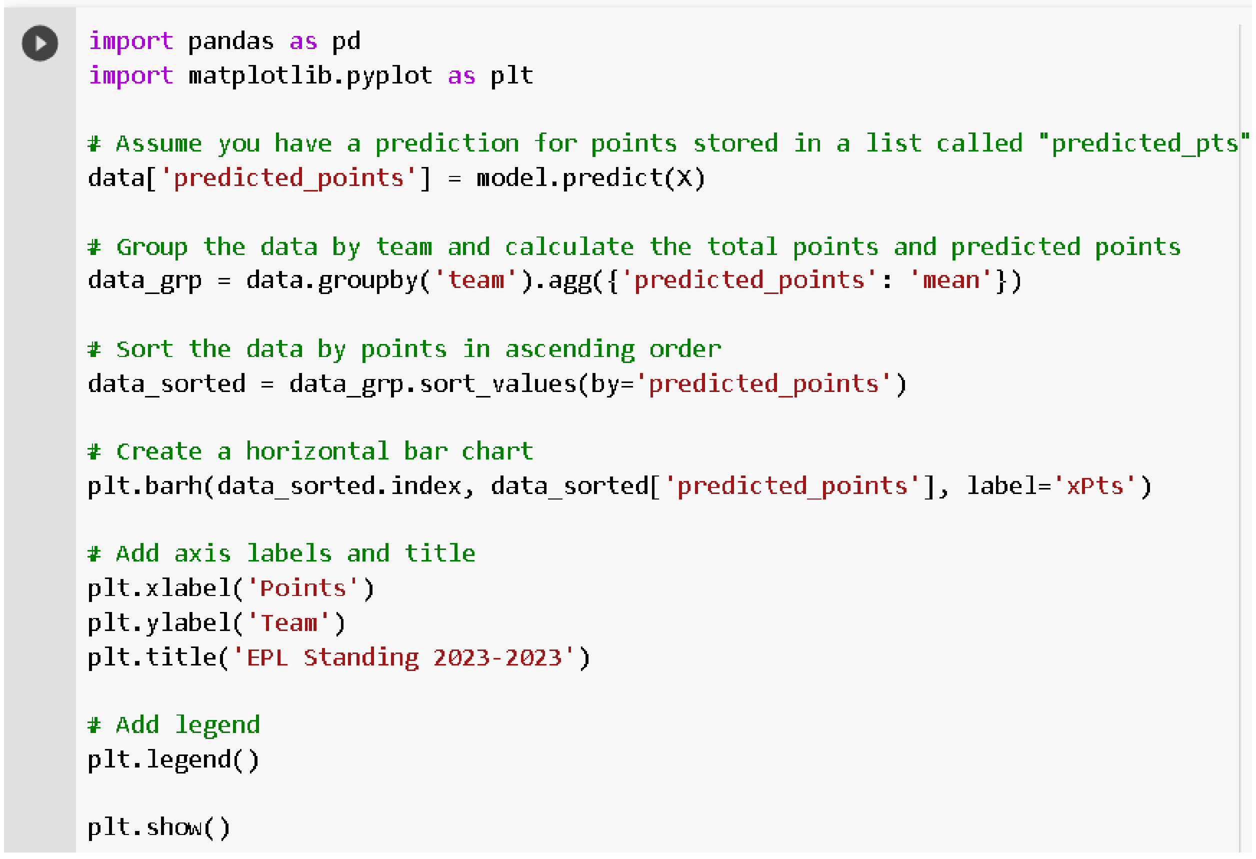
As can be seen apparently, Random Forest Regressor has a bigger value of training time and evaluation time than Linear regression. Hence, Random Forest Regressor could be more complex and achieve higher accuracy, but require longer training & evaluation time than Linear regression.

8. Prediction

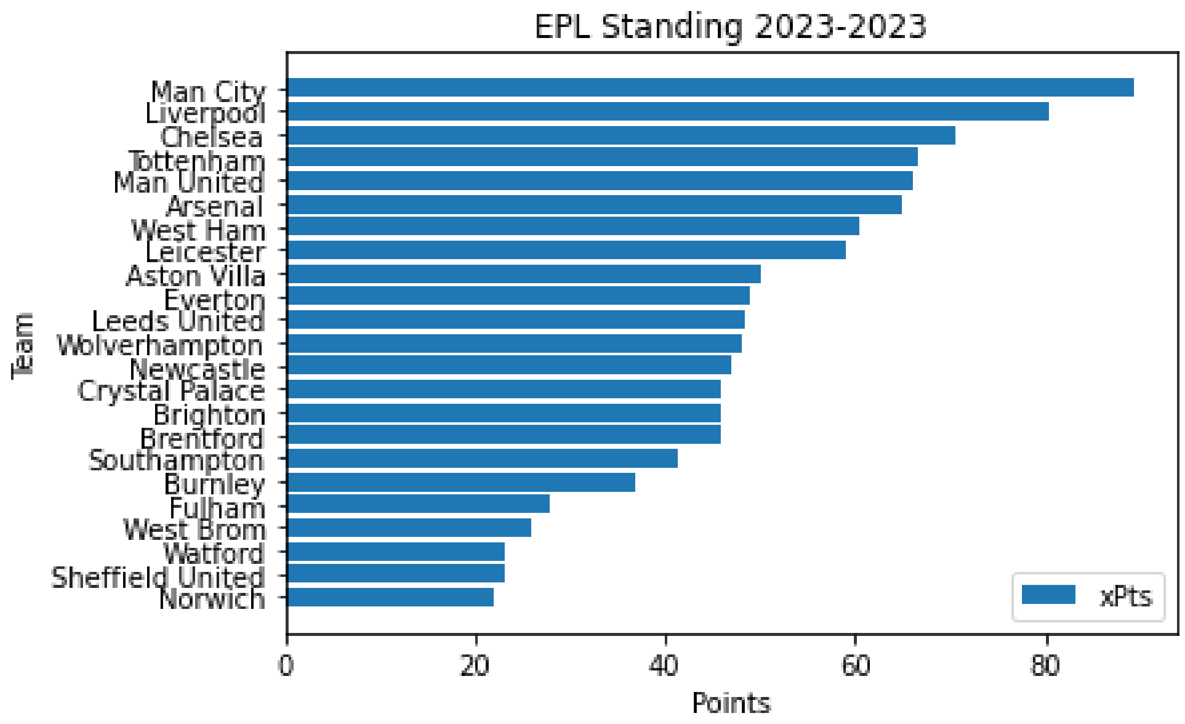
In this part, we are going to use the trained and validated model to make predictions on new data.

We are going to demonstrate two predicted results about the final ranking order of all clubs in the next season 2022-2023 by using two kinds of models: Random Forest Regressor & Linear regression.

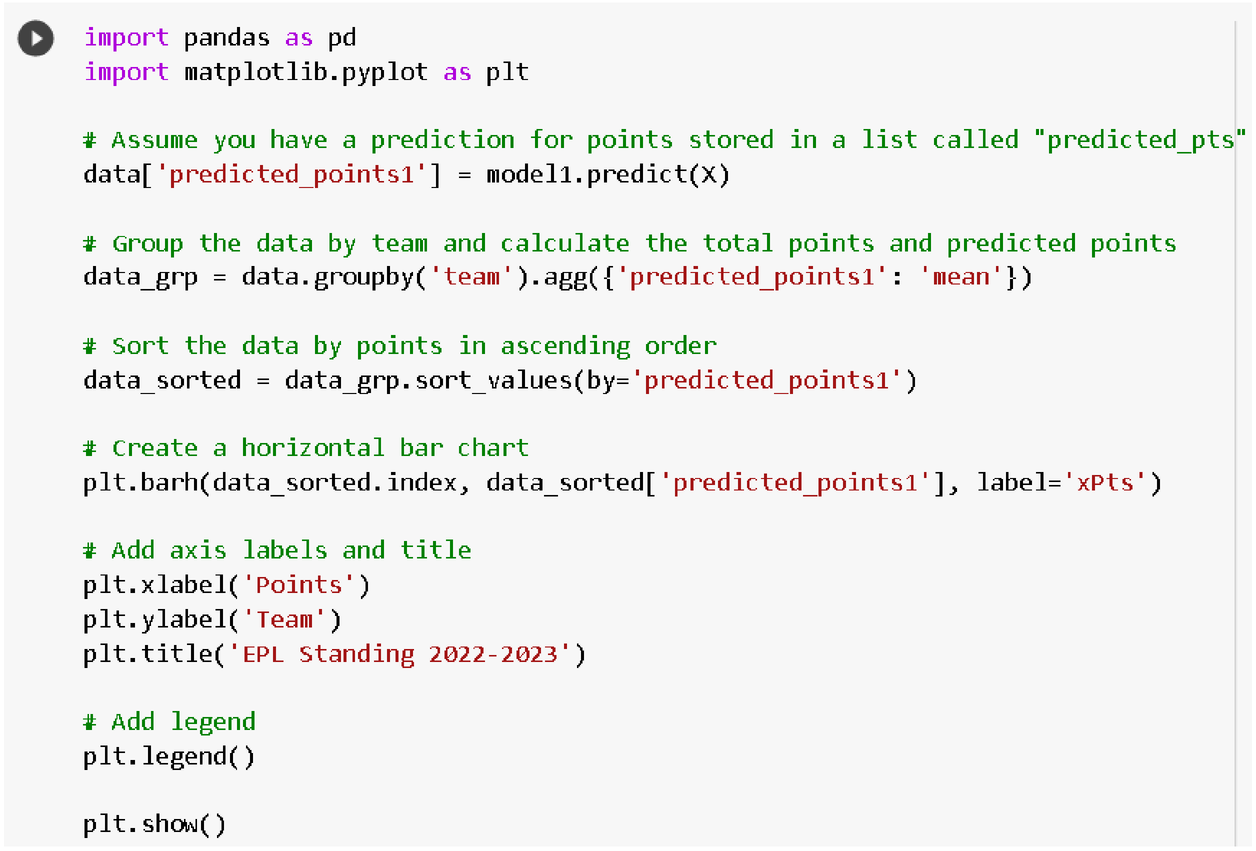
Random Forest Regressor



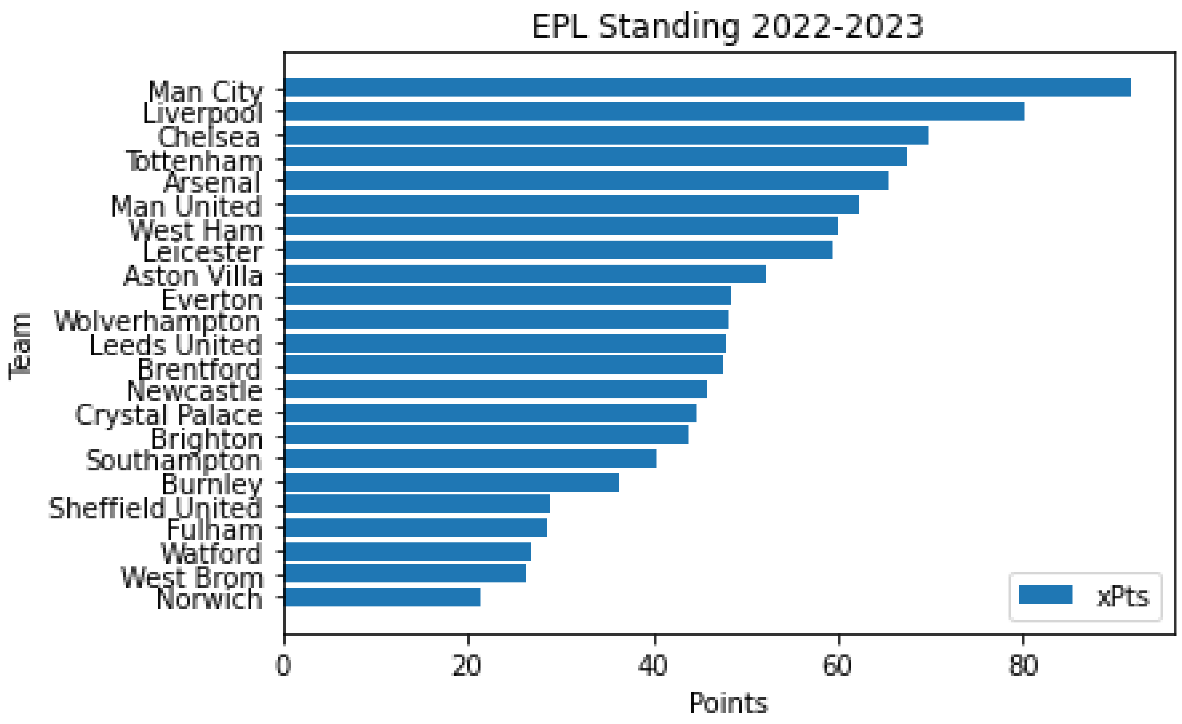
```
import pandas as pd
import matplotlib.pyplot as plt

# Assume you have a prediction for points stored in a list called "predicted_pts"
data['predicted_points'] = model.predict(X)

# Group the data by team and calculate the total points and predicted points
data_grp = data.groupby('team').agg({'predicted_points': 'mean'})

# Sort the data by points in ascending order
data_sorted = data_grp.sort_values(by='predicted_points')

# Create a horizontal bar chart
plt.barh(data_sorted.index, data_sorted['predicted_points'], label='xPts')

# Add axis labels and title
plt.xlabel('Points')
plt.ylabel('Team')
plt.title('EPL Standing 2023-2023')

# Add legend
plt.legend()

plt.show()
```



Linear regression



```
import pandas as pd
import matplotlib.pyplot as plt

# Assume you have a prediction for points stored in a list called "predicted_pts"
data['predicted_points1'] = model1.predict(X)

# Group the data by team and calculate the total points and predicted points
data_grp = data.groupby('team').agg({'predicted_points1': 'mean'})

# Sort the data by points in ascending order
data_sorted = data_grp.sort_values(by='predicted_points1')

# Create a horizontal bar chart
plt.barh(data_sorted.index, data_sorted['predicted_points1'], label='xPts')

# Add axis labels and title
plt.xlabel('Points')
plt.ylabel('Team')
plt.title('EPL Standing 2022-2023')

# Add legend
plt.legend()

plt.show()
```



As can be seen clearly, the predicted results from two models are different. From my point of view, the results from Random Forest Regressor should be considered the reference because of higher accuracy.

IV. RESULT AND DISCUSSION

1. Result

After about a month doing the project, in this section, we are going to present the results of our experiment in which we developed and evaluated various machine learning models to predict the final standings of the English Premier League (EPL) for the 2022-2023 season.

Generally speaking, we predict that the club in first place is Manchester City and the top 6 EPL includes the following teams: Liverpool, Chelsea, Tottenham, Arsenal, Manchester United.

2. Discussion

Besides, we have some difficulties and problematic matters when we do the project.

Firstly, on the beginning days doing the project, it was difficult for us to get the right direction for the project. We have not done this kind of machine learning project like this together before and this subject is quite hard for us because we have to research a lot while doing the project.

Secondly, Data Collection is really a problem for us. In that period of time, we are not able to collect much more data. We also have to collect data by hand because the dataset we need is usually not available on the internet. Therefore, the data we collect for the project could be not much and it may be not highly precise.

Thirdly, it could be Model Selection. Choosing the right machine learning model can be difficult, especially if we are not familiar with the various models and their strengths and weaknesses. This can impact the accuracy of our predictions and the overall success of our project.

Fourthly, it may be Feature Selection. Identifying the right features to include in our model can be challenging. We need to determine which features are most relevant to predicting EPL standings, and which ones are less important or even irrelevant.

Fifthly, it is not really simple for us to handle Model Performance. Even with the right data and features, our model may not perform as well as expected. This can be due to overfitting, underfitting, or other issues that may require further exploration and refinement.

Sixthly, Interpretation of Results could be a problem for us. Once we have obtained results from your model, it can be difficult to interpret them and understand what they mean in the context of EPL standings. This requires a deep understanding of the underlying data and models.

Seventhly, in the section “Prediction”, we have not predicted the standing of exact 20 clubs playing in EPL 2022-2023. Because we also think that three promoted teams do not affect the championship race. Although, if we do that, the accuracy of prediction definitely decreases and we have not had the solution for this problematic issue yet.

Overall, conducting a Machine Learning 2 project on EPL standing predictions can be challenging but also rewarding. Our predicted results could not have high accuracy because of all following reasons. With careful planning, attention to detail, and a deep understanding of the underlying data and models, we can overcome these challenges and produce meaningful results that can help inform decision-making in the EPL.

V. FUTURE WORK

In the future, we have to overcome all the difficulties as I mentioned in the DISCUSSION we face. Besides, we need to pay attention to these following problems to improve our Machine Learning & Data Mining project as well as the accuracy of prediction:

- Model Improvement: There may be opportunities to further improve the accuracy of your predictions by refining the models we have developed. For instance, we could experiment with different feature selections or try out new machine learning models.
- Data Collection and Integration: It may be possible to incorporate additional data sources into our project, such as player statistics or weather conditions. This could provide new insights into team performance and lead to more accurate predictions.
- Real-Time Predictions: We could explore the possibility of developing a real-time prediction system that could be used to inform live betting or other decision-making in the EPL.
- Visualizations and Dashboards: Creating interactive visualizations and dashboards could help users better understand and interpret the predictions we have generated. This could make the insights more accessible and usable for EPL teams, fans, and analysts.
- Multi-Season Analysis: Expanding our analysis beyond a single season could help identify trends and patterns that are not apparent when looking at a single season in isolation. This could provide more robust and reliable predictions.
- Model Deployment: Finally, we could explore ways to deploy our model in a production environment, such as through a web application or API. This would make the predictions more widely available and could have practical applications for EPL teams, fans, and other stakeholders.

Ultimately, there are many exciting avenues for future work in the area of EPL standing predictions for Machine Learning 2 projects. By continuing to refine and improve the models, integrating new data sources, and exploring novel approaches to visualization and deployment, there is significant potential to generate valuable insights and inform decision-making in the world of English football.

If our machine learning model for predicting EPL results proves to be accurate and effective, it could have a significant impact on the strategic decision-making processes of coaches, managers, and other stakeholders in the league. Here are a few ways in which this might happen:

- Tactical planning: With access to accurate predictions about upcoming matches and potential outcomes, coaches and managers could adjust their tactics and strategies accordingly. For example, if the machine learning model predicts that a certain opponent is likely to play defensively, the team could adapt their game plan to counter this.
- Transfer strategies: Teams could use the predictions from the machine learning model to identify potential transfer targets that would help them address specific weaknesses or gaps in their squad. For example, if the model predicts that a team is likely to struggle defensively, the team could prioritize signing a strong defender.
- Squad selection: Coaches and managers could use the predictions from the machine learning model to help inform their decisions about which players to select for upcoming matches. For example, if the model predicts that a certain player is likely to perform well against a particular opponent, the coach may choose to start them.
- Fan engagement: Accurate predictions about upcoming matches and outcomes could also be used to engage fans and build excitement around the sport. Fans may be more likely to attend matches or tune in to broadcasts if they have a sense of what to expect.

In conclusion, the use of machine learning to predict EPL results could help teams make more informed decisions and potentially gain a competitive advantage. However, it is also important to consider the potential drawbacks and ethical implications of relying too heavily on technology in sport.

VI. REFERENCES

1. PYTHON CHARTS. Retrieved 10/3/2023, from https://python-charts.com/ranking/radar-chart-plotly/?fbclid=IwAR2hwAfEdeCjuLCJX707nHZ0Rdo2O9Q8YWkq_3-L29GbltZ-LXdr2R-Wklk
2. Understat. Retrieved 10/3/2023, from https://understat.com/league/EPL?fbclid=IwAR1AEgrV9UD-3QPz3ANAuiiGcgKcPyRZVrLPfOTminDMTRr8Y6TTPvuiyjM
3. WhoScored.com. Retrieved 11/03/2023, from https://1xbet.whoscored.com/Regions/252/Tournaments/2/Seasons/9075/Stages/20934/Show/England-Premier-League-2022-2023?fbclid=IwAR3wvb6ALshciV3AKjKqhUoQ0AjUEsc-Sjh7iDbl2VhsKP05nFdDyjTPUtc
4. IBM. Retrieved 12/03/2023, from https://www.ibm.com/topics/random-forest?fbclid=IwAR2mISC2wvZJJvaFZ2czOu20d9cQxVDd2XIi-MXj4TkZTC-7WcdTEwvtQKk#:~:text=Random%20forest%20is%20a%20commonly,both%20classification%20and%20regression%20problems
5. Java T point. Retrieved 12/03/2023, from https://www.javatpoint.com/multiple-linear-regression-in-machine-learning?fbclid=IwAR1jDOSnH-_2XzYWa7rL4KAKsB-kXVqUoe0kbnIQIhvQQg_Ss5VrYnac6xI