**DSC510 Introduction to Data Science**

**Course Professor: Dr George Pallis**

# Fantasy Premier League Project

**November 2023**

**Team members:**
Fivos Lympouras
Constantinos Constantinou
Andreas Papadopoulos
Iosif Pintirishis

# Introduction

In the dynamic and unpredictable world of football, the English Premier League stands out as a beacon of high-level competition and excitement. This project delves into the challenging yet fascinating task of predicting the points collected for Premier League teams. Utilizing what we have learned so far of statistical analysis, machine learning algorithms, and a fair understanding of football dynamics, our aim is to forecast the total points a player will score in the next game week. Through this procedure, we explore the effect of various factors of a players performance. This will hopefully help the user in shaping the ultimate team, achieve high ranking in the game and become a legend in the worlds most recognized football league.

## Fantasy Premier League Description

Fantasy Premier League is an online game in which players collect points based on how real-life footballers perform each week. Simply put: you choose a virtual team of Premier League footballers, and if their real-life counterparts do well, you get points. This is hard to achieve, because you need to create a complete team covering all positions with well performing players. This is not as simple as it seems, because you have a limited budget. You can't simply buy the best players as that will be too expensive. You need to balance your team with hotshots and cheaper buys. This is where the challenge begins because you have to make the right choices, of players in positions, to be more dominant in the league. So it is critical to know which transfers will optimize your team for maximum point acquisition.

## Our Data Set

We collected our data from GitHub, from a repository which collects the data of each player, creating a csv each game week. We combined all 38 game weeks of season 2022-23 to make a large data set of 26505 rows and 40 columns. We consider our data to be highly representative, since it is taken from the real life matches played each week. The link for the dataset is `https://github.com/vaastav/Fantasy-Premier-League/tree/master/data/2022-23/gws`

## Problem Formulation

Our problem formulation centers on predicting the performance points for players in the Fantasy Premier League, aiming to forecast the number of points a player is likely to score in upcoming games. This predictive model seeks to encapsulate the various contributing factors such as a player's past performance, team form, opposition strength, and potential for scoring goals, providing assists, or keeping clean sheets. The objective is to create a robust tool that supports fantasy league participants in optimizing their team selections and making strategic decisions about player transfers, with a keen eye on maximizing point yield each week.

## Exploratory Data Analysis

Our data consists of the players of the premier league (rows) and their statistics of performance for the specific week (columns). Since we cannot list all of the columns here, using our knowledge of the game we will list a few of the more important ones, such as:

- Player Position (Goalkeeper, Defender etc.)
- Goals Scored
- Assists
- Total Points (Points scored in the gameweek)

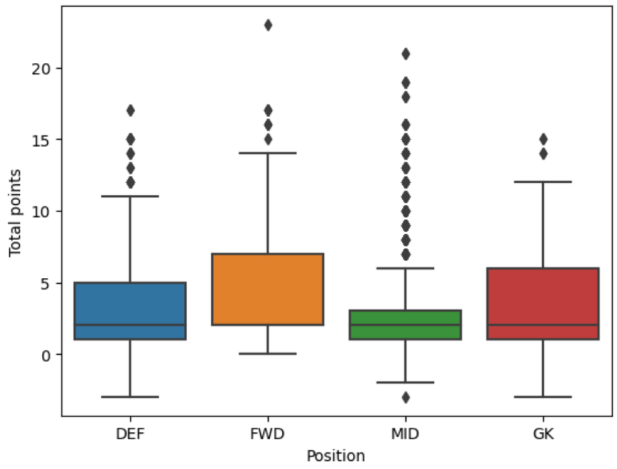


Figure 1: Boxplots for total points per position

From the boxplots of (figure 1) we can see equal medians for all positions except forwards. Moreover, we can see that they have different IQR and midfielders have a lot of outliers. This indicates that we may need to train our model for players of each position separately. Moreover, we can see that there are many outliers and that maybe cause us some problems. From the table of Figure 2, we have Q1, Q2, Q3 to be zero in almost everything and we have a very low mean for the goals, assists and total points. A possible reason is that our data set includes many players who play rarely.

| | goals_scored | assists | total_points |
|---|---|---|---|
| count | 26505.000000 | 26505.000000 | 26505.000000 |
| mean | 0.039162 | 0.034975 | 1.196906 |
| std | 0.215718 | 0.197954 | 2.355236 |
| min | 0.000000 | 0.000000 | -4.000000 |
| 25% | 0.000000 | 0.000000 | 0.000000 |
| 50% | 0.000000 | 0.000000 | 0.000000 |
| 75% | 0.000000 | 0.000000 | 1.000000 |
| max | 3.000000 | 3.000000 | 23.000000 |

Figure 2: Summary for Descriptive Statistics

| name | max | min | mean | median |
|---|---|---|---|---|
| Kieran Trippier | 12.0 | 0.0 | 5.210526 | 5.5 |
| Benjamin White | 8.0 | 0.0 | 4.105263 | 5.0 |
| Trent Alexander-Arnold | 17.0 | 0.0 | 4.105263 | 2.5 |
| Gabriel dos Santos Magalhães | 15.0 | 0.0 | 3.842105 | 2.0 |
| Ben Mee | 14.0 | 0.0 | 3.763158 | 2.0 |

Figure 3: Top 5 Defenders

The table above is sorted in descending order of the mean to highlight the players with the highest average performance. Trent Alexander-Arnold stands out with the highest maximum points of 17 and a mean of around 4.1, although his median is only 2.5, indicating a few high-scoring instances are pushing up his average. Kieran Trippier shows a more consistent performance with a mean of 5.2 and a median of 5.5, suggesting he regularly scores close to his average. Gabriel dos Santos Magalhes and Ben Mee, with lower means, indicate less overall impact in terms of points. This type of analysis is crucial for fantasy football managers to assess player reliability and potential for scoring high points in upcoming matches.

## Preprocessing

A large part of our preprocessing, was about changing the column values to the correct type, for easier use. After fixing the data types we checked for any missing or NA values. Firstly, we created any additional columns that we deemed necessary, and the "next game week points" (or **NGWP**) column which will also be our target value for this project, since we will attempt to estimate the total points a player will score in the next week. Later our target value will be the total points and we are going to remove the features of this gameweek and we will add features from previous gameweeks. More details about this in the Evaluation methodology section.

Our next step was to check for any outliers. When creating the violin plot (figure 4) for the total of the NGWP column we see that there is a very large number of players that have 0 total points. This is odd but at the same time it can be explained since many players in the league don't often play.

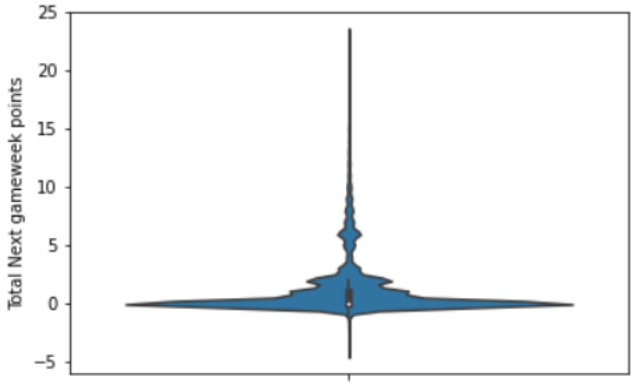


Figure 4: Violin Plot for Next Game Week Points

We removed any of the players that had an average game time less than 60 minutes. This means we are talking about the starting players and players with high impact in the game. As shown in the box plot of (figure 5), we have a much better view of what is going on, and we have fewer outliers now since we can also see that our box plot has moved up as well.

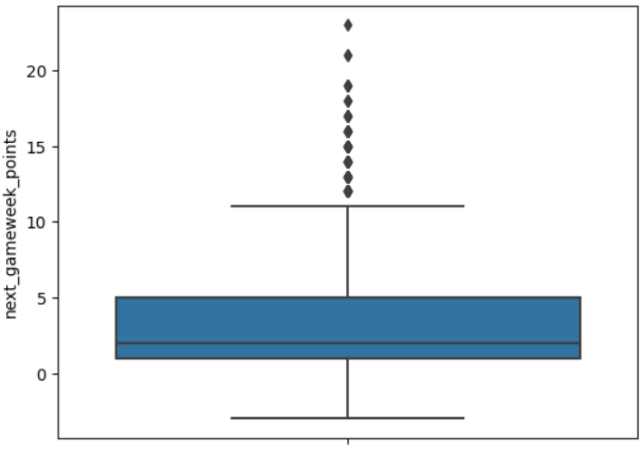


Figure 5: Boxplot for the Totals of Starting players NGWP

## Encoding

In our project we used One Hot Encoding on the "next game week home away" column. This columns purpose is to tell us whether the next game that the player plays, he will be in the home or away team. We wanted to encode and look at the influence this has on the scored points of a player, since we suspect that it does influence the players performance. Moreover, we used One Hot Encoding on the "Position" column, in order to turn the positions into a numeric format.

| position_GK | position_DEF | position_MID | position_FWD |
|---|---|---|---|
| 1.0 | 0.0 | 0.0 | 0.0 |
| 1.0 | 0.0 | 0.0 | 0.0 |

Figure 6: New Columns for Each Position

| next_gameweek_home_away_True | next_gameweek_home_away_False |
|---|---|
| 1.0 | 0.0 |
| 0.0 | 1.0 |

Figure 7: New Columns for Next Game Home and Away

# Main Objective

With all the data preprocessing out of the way our goal, was to use the game week data (goals, assists, etc) to try and predict the points each player will score in his **next** game week. This is why we created the NGWP column as well as other columns for the NGW opponent or if the NGW game would be played home or away.

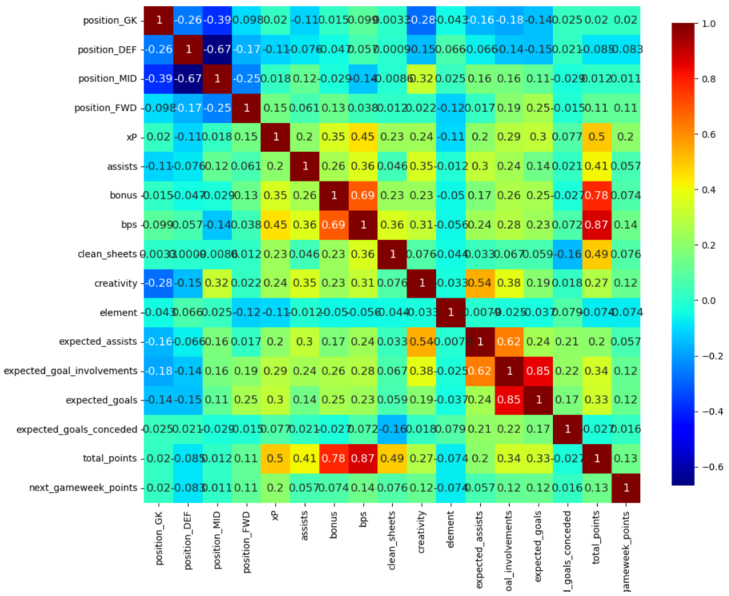


Figure 8: Correlation matrix for dataset features

In the heatmap of (figure 8) we do not see many variables relate to each other. Moreover, the Y variable of NGWP also has no correlation with the other variables and that means that we do not expect very good results later.

# Evaluation methodology

## Random Forest Regression

Using the aforementioned NGWP column as our target value, we first attempted a random forest regression and our results were bad cause we had a really bad R2 score. This led us to give our data a harder look and here we realised that our data might be better in a classification algorithm.

## Classification

We decided to classify our data on our target value NGWP using the following format.

- Players that scored 0 or less points go to **class 1**
- Players that scored 1 point go to **class 2**
- Players that scored 2 points go to **class 3**
- Players that scored 3 to 6 points go to **class 4**
- Players that scored 7 to 9 points go to **class 5**
- Players that scored 10 or more points go to **class 6**

Having classified our data the next step would be to do a feature selection process. We used these three methods for feature selection, Random forest, Naive Bayes and Decision trees. After feature selection we selected the best features and we got the results that you see in (figure 9). F1 score is low and based on the (figure 1 on page 1) we decided to keep only the defenders. Similarly we can model our data for the players in the other positions.

*---------- CLASSIFICATION MODEL PERFORMANCE EVALUATION ----------*

[93]:

| | Model | Cross Val Score | Test Accuracy | Average_Accuracy | Precision | Recall | F1 Score |
|---|---|---|---|---|---|---|---|
| 5 | AdaBoostClassifier | 0.352163 | 0.3428 | 0.347481 | 0.392001 | 0.243961 | 0.208615 |
| 1 | SVC | 0.346310 | 0.3418 | 0.344055 | 0.161906 | 0.220056 | 0.157236 |
| 3 | RandomForestClassifier | 0.297455 | 0.3052 | 0.301328 | 0.257092 | 0.255983 | 0.252142 |
| 2 | DecisionTreeClassifier | 0.290585 | 0.3032 | 0.296893 | 0.249590 | 0.256242 | 0.246491 |
| 4 | SGDClassifier | 0.243511 | 0.3418 | 0.292656 | 0.178640 | 0.226260 | 0.183049 |
| 0 | GaussianNB | 0.214249 | 0.2228 | 0.218525 | 0.138928 | 0.244341 | 0.154758 |

Figure 9: Classification Model Performance Evaluation

After doing the same procedure we saw a clear improvement of the F1 scores for defenders only (figure 10) and in both tables we have as our best classifiers as far as concerned the F1 scores the Random Forest and Decision Tree.

*---------- CLASSIFICATION MODEL PERFORMANCE EVALUATION ----------*

[67]:

| | Model | Cross Val Score | Test Accuracy | Average_Accuracy | Precision | Recall | F1 Score |
|---|---|---|---|---|---|---|---|
| 3 | RandomForestClassifier | 0.386565 | 0.4007 | 0.393633 | 0.336321 | 0.333281 | 0.329838 |
| 2 | DecisionTreeClassifier | 0.386579 | 0.3941 | 0.390339 | 0.323051 | 0.327529 | 0.322425 |
| 5 | AdaBoostClassifier | 0.258543 | 0.3388 | 0.298672 | 0.249384 | 0.244671 | 0.237207 |
| 1 | SVC | 0.305884 | 0.2899 | 0.297892 | 0.118371 | 0.187061 | 0.118444 |
| 0 | GaussianNB | 0.290311 | 0.2834 | 0.286855 | 0.277633 | 0.227315 | 0.208001 |
| 4 | SGDClassifier | 0.229368 | 0.0945 | 0.161934 | 0.117518 | 0.222390 | 0.081564 |

Figure 10: Classification Model Performance Evaluation with only Defenders

We decided to continue with defenders only. From now on our target value will be 'total points' and we are going to remove the columns from this week, the ones that we don't know from before (e.g. goals scored, assists etc.) and with rolling statistics we will create the previous stats for the players. We used rolling statistics for specific columns: clean sheets, expected goals conceded, goals conceded, goals scored and assists. The idea behind these columns are that they were selected because clean sheets, expected goals conceded,

goals conceded were among the important features in our previous feature selection for defenders only for all the classifiers that we did try sequential forward selection. Additionally, we added goals scored and assists because based on our knowledge on the FPL game are very critical variables.

For the rolling statistics, we used 3 time spaces: The "previous", the "recent" and the "seasonal" for specific columns like goals,assists etc. The "previous" refers to the last gameweek, "recent" is about the mean of the last 4 gameweeks and "seasonal" for the mean of the last 16 gameweeks. Moreover, we observe that using this method our data set has lot of NA, because the first gameweek haven't a previous one. Finally, we replace those NA with zero, in order to calculate the means afterwards.

*---------- CLASSIFICATION MODEL PERFORMANCE EVALUATION ----------

[47]:

| | Model | Cross Val Score | Test Accuracy | Average_Accuracy | Precision | Recall | F1 Score |
|---|---|---|---|---|---|---|---|
| 2 | DecisionTreeClassifier | 0.380137 | 0.4295 | 0.404818 | 0.348620 | 0.349182 | 0.345340 |
| 3 | RandomForestClassifier | 0.376156 | 0.4295 | 0.402828 | 0.437167 | 0.379999 | 0.388292 |
| 5 | AdaBoostClassifier | 0.310464 | 0.2853 | 0.297882 | 0.198221 | 0.203492 | 0.189788 |
| 0 | GaussianNB | 0.294777 | 0.2978 | 0.296289 | 0.256863 | 0.201546 | 0.178702 |
| 1 | SVC | 0.263337 | 0.2790 | 0.271168 | 0.046499 | 0.166667 | 0.072712 |
| 4 | SGDClassifier | 0.207696 | 0.2790 | 0.243348 | 0.138624 | 0.190880 | 0.130730 |

Figure 11: Model performance for defenders using rolling statistics

With rolling statistics there is a slight improvement on the F1 scores (figure 11) with Random Forest and Decision tree to be again the 2 best classifiers for modeling our data. Thus, we continue with rolling statistics only for defenders. Later, we tried PCA and SVD with scaled data for dimensionality reduction despite they are not optimal for classification just to clarify that for our case. A reason of that is that they focus on capturing the directions of maximum variance in the data and not on separating different classes. Indeed, the results were worse than the ones that we had in the (figure 11).
Our goal to predict which of the 6 classes a player would go seems to be a bit difficult , according to our results. So we thought about dividing even more our classes to 2. Our problem from now on would be a Binary Classification.

## Binary Classification

In the binary classification for the Fantasy Premier League, we divided the players into two distinct classes based on their total points. Class 1 includes players scoring 4 points or below, representing standard or below-average performances. Class 2, meanwhile, consists of players scoring above 4 points, indicating significant contributions like goals or assists. This classification is particularly useful for fantasy league strategies, especially when considering player transfers. In the FPL game, when you transfer a player without having free transfer you penalized with minus 4 points. Therefore, ensuring a player is likely to score above 4 points is crucial to know to be able to decide if a transfer is worth it. This system aids in making more informed decisions about player transfers and optimizing team performance.

- Players that scored 4 points or below go to **class 1: 0**
- Players that scored above 4 points go to **class 2: 1**

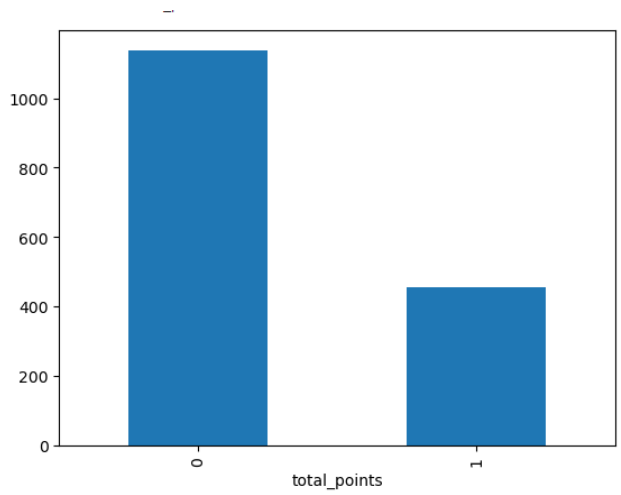


Figure 12: Value counts of the classes

The bar chart shows an imbalanced target value, with a greater number of players falling into Class 0 than Class 1. Given this imbalance, the F1 score, which considers both precision and recall, is a more appropriate metric than accuracy as it avoids bias towards the majority class. Using a weighted F1 score can further adjust for the skew by assigning more significance to the underrepresented class, ensuring a more balanced evaluation of the model's predictive performance.

## Feature selection

We proceed with Feature selection using the two algorithms that they outperformed the other algorithms in our previous tests. The two algorithms are Random Forest and Decision Tree Classifiers. Random Forest is a strong choice for binary classification due to its ensemble learning approach, which combines multiple decision trees to enhance accuracy and robustness. It effectively handles over-fitting, a common issue in decision trees, by averaging results from various trees. One of the advantages of using tree-based algorithms, like Random Forest is that they do not require feature scaling. This is the reason that we did not use scaled data.

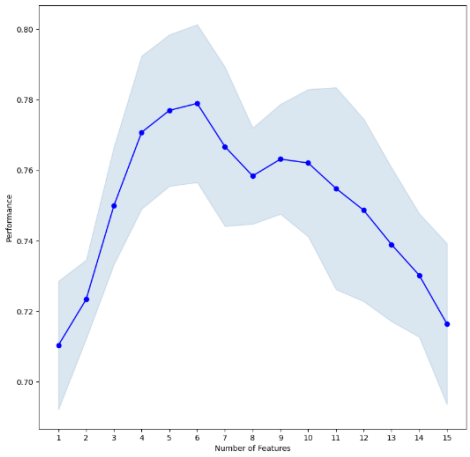


Figure 13: Feature Selection using Random Forest

The feature selection method that we used is forward sequential selection. The best features are round, recent clean sheets, prev clean sheets, prev expected goals conceded, prev goals conceded and prev assists. These features are reasonable to be important for our case as we are trying to predict the classes for defenders only. Below, we did the same for Decision Tree Classifier which is well-suited for binary classification. It divides the dataset into subsets based on the value of the most significant attributes, leading to clear binary splits. It is also a tree-based algorithm, so it does not require scaling.

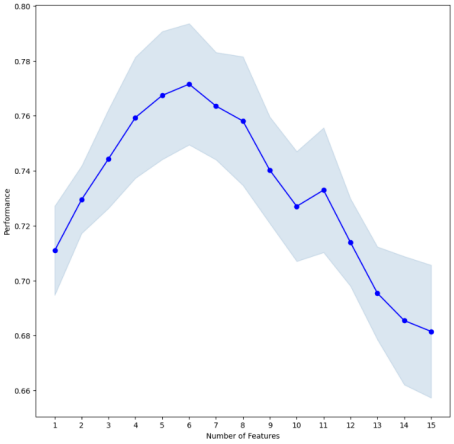


Figure 14: Feature Selection using Decision Tree

Same as before, the feature selection method that we used is forward sequential selection. The best features are round, recent clean sheets, prev clean sheets, prev expected goals conceded, prev goals conceded and prev goals scored. The difference between the two algorithms best features are that decision tree forward sequential method selected prev goals scored, instead of prev assists. After trying classifications algorithms with many combinations of these features we went ahead and used these features for our model: round, recent clean sheets, prev clean sheets, prev expected goals conceded, prev goals conceded and prev assists.
After splitting the dataset with test size to be 20% of our data we created again the Classification Model Performance Evaluation. XGBoost is highly effective for binary classification tasks, it is robust to over-fitting and capable of handling various types of data. Also, it is a tree-based algorithm, so it does not require scaled data. It builds an ensemble of decision trees in a sequential manner, where each tree corrects the errors of the previous ones, leading to improved accuracy. That's why we added XGBoost Classifier in our tries.

*---------- CLASSIFICATION MODEL PERFORMANCE EVALUATION ----------*

[25]:

| | Model | Cross Val Score | Test Accuracy | Average_Accuracy | Precision | Recall | Avg Precision Recall | F1 Score |
|---|---|---|---|---|---|---|---|---|
| 0 | XGBClassifier | 0.753955 | 0.7962 | 0.775078 | 0.783429 | 0.796238 | 0.653557 | 0.782559 |
| 6 | RandomForestClassifier | 0.739850 | 0.8056 | 0.772725 | 0.795063 | 0.805643 | 0.662219 | 0.795560 |
| 5 | DecisionTreeClassifier | 0.739087 | 0.7837 | 0.761394 | 0.777090 | 0.783699 | 0.490184 | 0.779694 |
| 8 | AdaBoostClassifier | 0.704528 | 0.7398 | 0.722164 | 0.702604 | 0.739812 | 0.403286 | 0.659460 |
| 4 | SVC | 0.709246 | 0.7335 | 0.721373 | 0.538084 | 0.733542 | 0.259205 | 0.620792 |
| 1 | LogisticRegression | 0.707677 | 0.7335 | 0.720589 | 0.672394 | 0.733542 | 0.325786 | 0.626508 |
| 2 | KNeighborsClassifier | 0.697546 | 0.7304 | 0.713973 | 0.713417 | 0.730408 | 0.424562 | 0.719437 |
| 3 | GaussianNB | 0.700603 | 0.7022 | 0.701401 | 0.611095 | 0.702194 | 0.346295 | 0.632189 |
| 7 | SGDClassifier | 0.636380 | 0.7210 | 0.678690 | 0.581091 | 0.721003 | 0.291080 | 0.620047 |

Figure 15: Binary Classification with outliers

*---------- CLASSIFICATION MODEL PERFORMANCE EVALUATION ----------*

| | Model | Cross Val Score | Test Accuracy | Average_Accuracy | Precision | Recall | Avg Precision Recall | F1 Score |
|---|---|---|---|---|---|---|---|---|
| 0 | XGBClassifier | 0.760076 | 0.8089 | 0.784488 | 0.799372 | 0.808917 | 0.626189 | 0.792651 |
| 6 | RandomForestClassifier | 0.758508 | 0.7930 | 0.775754 | 0.780055 | 0.792994 | 0.541322 | 0.780234 |
| 5 | DecisionTreeClassifier | 0.742540 | 0.7739 | 0.758220 | 0.760617 | 0.773885 | 0.446842 | 0.763953 |
| 4 | SVC | 0.724311 | 0.7325 | 0.728406 | 0.536533 | 0.732484 | 0.257219 | 0.619380 |
| 1 | LogisticRegression | 0.726698 | 0.7229 | 0.724799 | 0.534643 | 0.722930 | 0.295407 | 0.614691 |
| 2 | KNeighborsClassifier | 0.725086 | 0.7102 | 0.717643 | 0.672996 | 0.710191 | 0.377376 | 0.682304 |
| 8 | AdaBoostClassifier | 0.720337 | 0.7102 | 0.715268 | 0.562876 | 0.710191 | 0.329844 | 0.613646 |
| 3 | GaussianNB | 0.717949 | 0.6943 | 0.706125 | 0.548938 | 0.694268 | 0.314385 | 0.605267 |
| 7 | SGDClassifier | 0.556178 | 0.7293 | 0.642739 | 0.663105 | 0.729299 | 0.296669 | 0.643831 |

Figure 16: Binary Classification without outliers

From (figure 15) and (figure 16), we can see that the two tables have similar F1 scores. We observe that the best 3 algorithms are: Random Forest Classifier, XGBoost Classifier and Decision Tree Classifier. Random Forest and Decision Tree Classifiers are slightly better without removing the outliers. XGBoost is better after removing the outliers.

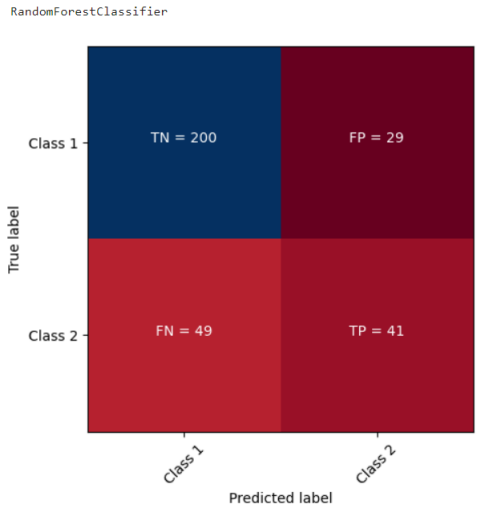


Figure 17: Plot confusion matrix for Random Forest

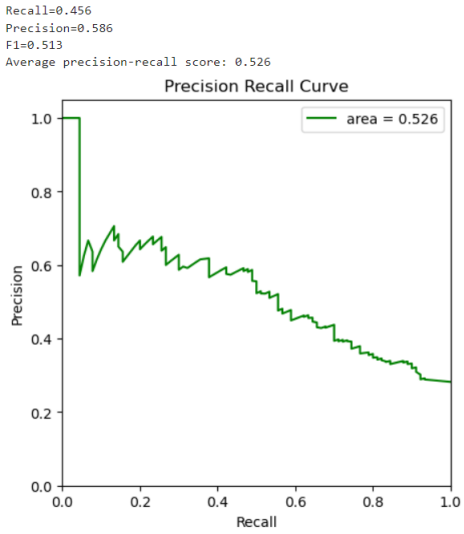


Figure 18: Precision Recall Curve for Random Forest

The confusion matrix (figure 17 on the previous page) and Precision-Recall Curve (figure 18) displayed here indicate the performance of the Random Forest classifier on our binary classification problem. In the confusion matrix, the high number of true negatives (TN = 200) and the relatively low number of true positives (TP = 41) suggest the model may be more inclined to predict Class 1, possibly due to class imbalance. The Precision-Recall Curve reflects a similar story. An area of 0.526 indicates a modest ability to discriminate between classes, with the precision of 0.586 and recall of 0.456 showing that the model is relatively conservative in predicting Class 2.

## Grid Search CV

We continue with Grid Search CV using the three best classifier algorithms for our problem. The three classifiers are Random Forest, Decision Tree and XGBoost. We tried Grid Search CV with $cv = 5$, using many hyperparameters and different combinations of their values. Firstly, we see the results of the Grid Search CV of Decision Tree classifier.
Optimized Model

---

Final accuracy score on the testing data: 0.7743
Final Weighted F1 score on the testing data: 0.7696
The hyperparameters of the optimized model were

```
max_depth=30,
max_features=0.5.
```

Secondly, for the XGBoost classifier we used the dataset without outliers because we had a better result in the Classification Model Performance Evaluation.
Optimized XGB Model

---

Final accuracy score on the testing data: 0.7994
Final Weighted F1 score on the testing data: 0.7844
The hyperparameters of the optimized model were

```
colsample_bytree=0.8,
enable_categorical=False,
eval_metric='mlogloss',
learning_rate=0.2, max_depth=7,
min_child_weight=1, missing=nan,
n_estimators=400
```

## Best model

Lastly, we did grid Search CV for Random Forest Classifier.
Unoptimized model

---

Accuracy score on testing data: 0.8213
Weighted F1 score on testing data: 0.8093

Optimized Model

---

Final accuracy score on the testing data: 0.8245
Final Weighted F1 score on the testing data: 0.8121
The hyperparameters of the optimized model were

```
criterion='entropy', max_depth=20,
n_estimators=400
```

The optimized model, which employs a Random Forest Classifier with the 'entropy' criterion, a maximum depth of 20, and 400 estimators, shows a minor improvement over the unoptimized model. The final accuracy score has increased from 0.8213 to 0.8245, and the weighted F1 score has seen a slight rise from 0.8093 to 0.8121. These improvements, while relatively small, suggest that the hyperparameter tuning has made the model slightly more effective at generalizing to the test data. The increased number of estimators and controlled depth likely contributed to the improved performance by enhancing the model's ability to learn from the data without overfitting. The use of 'entropy' as a criterion for splitting also suggests that the model might be better at handling complex patterns in the data, leading to a more accurate classification outcome.

## Conclusion

In conclusion, our investigation into Fantasy Premier League data has established that classification models, especially when applied to specific player positions like defenders, outperform regression models for predicting player performance. The strategy of segmenting the data by position and leveraging rolling statistics greatly improved the accuracy of our predictions, indicating the importance of considering the recent form in player performance analysis. Despite experimenting with PCA and SVD for dimensionality reduction, these techniques did not enhance the model's predictive strength, suggesting the original features hold essential information for accurate predictions. Additionally, the binary classification approach yielded more robust results, reaffirming its suitability for this domain where the focus is often on distinguishing between high and low performers rather than predicting exact points.