# Football Prediction

SOL DANIELA CELY DUARTE 20212020026

SERGIO SANTIAGO DUARTE 20211020013

FRANCO JOSÉ GUZMÁN 20211020155

DATA SCIENCE INTRODUCTION

Instructor: Carlos Sierra

Group: 020-81

District University Francisco José De Caldas

Systems Engineering

Bogotá D.C

July 16, 2024

## 1. Introducción:

In this report, we will demonstrate the engineering exercise conducted to apply data science techniques to the topic of football match prediction. The primary objective of this document is to provide a detailed step-by-step account of how we developed our classification model to determine the outcome of a football match. Additionally, as a guide, we will explain the reasoning behind the decisions made throughout this process.

Finally, we will make observations about the results obtained, explain the reasons behind them, discuss possible choices that could have positively impacted the project, and propose improvements for our approach.

{Keywords:} Football, Classification, Data Science, Machine Learning.

## 2. Data Collection:

Data collection is the first step of our work and it is relatively easy because we obtained the dataset from a Kaggle competition titled "Football Match Probability Prediction." This dataset provides us with over one hundred thousand records and 190 columns, which are divided into columns that describe the game (Table 1) and columns that detail the history of the last 10 games of both teams (Table 2).

| Column name | Description |
|---|---|
| target | The variable you have to predict the probabilities only available in the train set. |
| home_team_name | The name of the Home the team. Hidden in test set, see this discussion |
| away_team_name | The name of the Away the team. Hidden in test set, see this discussion |
| match_date | The match date (UTC). |
| league_name | The league name. |
| league_name | The league id. Note that league names can be identical for two differents id. |
| is_cup | If the value is 1 the match is played for a cup competition. |

| | |
|---|---|
| home_team_coach_id | The id of the Home team coach. |
| away_team_coach_id | The id of the Away team coach. |

Tabla 1: Descriptive Columns

| Nombre de la columna | Descripción |
|---|---|
| home/away_team_history_match_date_{i} | The date of the last i-th match played by Home/Away team. |
| home/away_team_history_is_play_home_{i} | If 1, the Home/Away team played home. |
| home/away_team_history_is_cup_{i} | If 1, the match was a cup competition. |
| home/away_team_history_goal_{i} | The number of goals scored by the Home/Away team on its last i-th match. |
| home/away_team_history_opponent_goal_{i} | The number of goals conceded by the Home/Away team on its last i-th match. |
| home/away_team_history_rating_{i} | The rating of the Home/Away team on its last i-th match (pre match rating). |
| home/away_team_history_opponent_rating_{i} | The rating of the opponent team on Home/Away team last i-th match (pre match rating). |
| home/away_team_history_coach_{i} | The coach id of the Home/Away team on its last i-th match. |
| home/away_team_history_league_id_{i} | The league name id by the Home/Away team on its last i-th match. |

Tabla 2: Historical home/away team features

The data comes in CSV format, which is downloaded from the Kaggle website and then read with pandas, along with the target data that comes in another CSV file. The following code shows the process of loading these data, along with their merging and index assignment.

```
import pandas as pd
import warnings
warnings.filterwarnings('ignore')

# get data from kaggle
football_train_df = pd.read_csv('./data/football_train.csv')
football_target_df = pd.read_csv('./data/football_target.csv')
```

```
football_target_df.set_index('id', inplace=True)
football_train_df.set_index('id', inplace=True)

football_target_df = football_target_df[['score']]
football_train_df = football_train_df.join(football_target_df)
```

With this code, we would have a DataFrame with 110,938 rows and 190 columns. These data will go through the cleaning process.

3. **Data Cleaning:**

When loading the data, we have 1,717,256 missing values, which are mainly distributed among those matches for which we do not have a good history of past matches, along with a large number of missing data corresponding to the coaches' history for each team.

We can say that there are 9% of missing data overall, similarly distributed in 20% of the records where the number of matches played by a team is less than 10. In this regard, the first cleaning process is to remove the teams with fewer than 10 matches.

```
# remove every team that has less than 10 matches

teams_less_10_matches                                                    =
football_train_df['home_team_name'].value_counts() < 10

teams_less_10_matches                                                    =
teams_less_10_matches[teams_less_10_matches].index

# remove teams from teams_less_10_matches

football_train_df                                                        =
football_train_df[~football_train_df['home_team_name'].isin(teams
_less_10_matches)]

football_train_df                                                        =
football_train_df[~football_train_df['away_team_name'].isin(teams
_less_10_matches)]
```

We extract the names of the clubs since this information is redundant compared to the league IDs.

```
league_dict = football_train_df[['league_id',
'league_name']].drop_duplicates().set_index('league_id').to_dict(
)['league_name']
```

This approach would eliminate 28% of the data. However, a couple of less invasive processes with the data are programmed, which can be used for different training.

One of these simply fills all the nulls with 0 (which corresponds to the easy case). Optional filters are to remove the coach columns to reduce the amount of nulls that will be filled.

```
football_train_df =
football_train_df[football_train_df.columns.drop(list(football_tr
ain_df.filter(regex='coach', axis=1)))]
```

Finally, we also remove the rows that have more than 70% missing values.

```
football_train_df =
football_train_df.dropna(thresh=0.7*len(football_train_df.columns
), axis=0)
```

4. **Data Preparation:**

We change to date format for all columns that have a date in them. To do this, we use regular expressions to find the columns to change:

```
# filter date columns and change datatype

for col in football_train_df.filter(regex='date',
axis=1).columns:

   football_train_df[col]= pd.to_datetime(football_train_df[col])
```

We use a label encoder to transform the string values in the "target" column to a numerical value, and the same for the "is_cup" parameter.

```
# Label encoding target

le = LabelEncoder()

football_train_df['target'] =
le.fit_transform(football_train_df['target'])

football_train_df['is_cup'] =
le.fit_transform(football_train_df['is_cup'])
```

We also separate the score (which was in #-# format) into two columns for each score value, converting the values to integers.

```
# Separate score column # - # in two columns home_score and
away_score and merge it with football_train_df

football_train_df['home_score'] =
football_train_df['score'].str.split('-',
expand=True)[0].astype(int)

football_train_df['away_score'] =
football_train_df['score'].str.split('-',
expand=True)[1].astype(int)

football_train_df.drop('score', axis=1, inplace=True)
```

## 5. Feature Engineering

For feature engineering, we define a series of values that could enhance the performance of the models we are training:

- **home_days_between_last_match:** Number of days since the home team's last match.
- **home_avg_home/opponent_goal_last_10:** Average goals scored/conceded by the home team in their last 10 matches.
- **away_avg_home/opponent_goal_last_10:** Average goals scored/conceded by the away team in their last 10 matches.
- **month:** The month in which the match took place.
- **away_wins_last_10:** Number of wins by the away team in their last 10 matches.
- **away_draws_last_10:** Number of draws by the away team in their last 10 matches.
- **away_losses_last_10:** Number of losses by the away team in their last 10 matches.
- **home_wins_last_10:** Number of wins by the home team in their last 10 matches.
- **home_draws_last_10:** Number of draws by the home team in their last 10 matches.
- **home_losses_last_10:** Number of losses by the home team in their last 10 matches.
- **home_avg_rating_last_10:** Average rating of the home team in their last 10 matches.
- **away_avg_rating_last_10:** Average rating of the away team in their last 10 matches.

```
# find the days between the last match of the home team

football_reduced_df['home_days_betwent_last_match'] =
(football_reduced_df['match_date'] -
football_train_df['home_team_history_match_date_1']).dt.days
```

```
football_reduced_df['home_days_betwent_last_match'].head(10)


# home average goals in last 10 matches (columns
home_team_history_goal_1 to home_team_history_goal_10)

football_reduced_df['home_avg_home_goal_last_10'] =
football_train_df.filter(regex='home_team_history_goal',
axis=1).mean(axis=1)

football_reduced_df['home_avg_opponent_goal_last_10'] =
football_train_df.filter(regex='home_team_history_opponent_goal',
axis=1).mean(axis=1)


football_reduced_df['away_avg_home_goal_last_10'] =
football_train_df.filter(regex='away_team_history_goal',
axis=1).mean(axis=1)

football_reduced_df['away_avg_opponent_goal_last_10'] =
football_train_df.filter(regex='away_team_history_opponent_goal',
axis=1).mean(axis=1)


# create columns by month based on date

football_reduced_df['month'] =
football_reduced_df['match_date'].dt.month


# how many matches did win the home/away team in the last 10 matches

home_history_goals =
football_train_df.filter(regex='home_team_history_goal',
axis=1).to_numpy()

home_history_oponent_goals =
football_train_df.filter(regex='home_team_history_opponent_goal',
axis=1).to_numpy()

football_reduced_df['home_wins_last_10'] = (home_history_goals >
home_history_oponent_goals).sum(axis=1)

football_reduced_df['home_dawn_last_10'] = (home_history_goals ==
home_history_oponent_goals).sum(axis=1)

football_reduced_df['home_lose_last_10'] = (home_history_goals <
home_history_oponent_goals).sum(axis=1)
```

```
away_history_goals =
football_train_df.filter(regex='away_team_history_goal',
axis=1).to_numpy()

away_history_oponent_goals =
football_train_df.filter(regex='away_team_history_opponent_goal',
axis=1).to_numpy()

football_reduced_df['away_wins_last_10'] = (away_history_goals >
away_history_oponent_goals).sum(axis=1)

football_reduced_df['away_dawn_last_10'] = (away_history_goals ==
away_history_oponent_goals).sum(axis=1)

football_reduced_df['away_lose_last_10'] = (away_history_goals <
away_history_oponent_goals).sum(axis=1)


# mean team rating and oponent rating

football_reduced_df['home_avg_rating_last_10'] =
football_train_df.filter(regex='home_team_history_rating',
axis=1).mean(axis=1)

football_reduced_df['home_avg_opponent_rating_last_10'] =
football_train_df.filter(regex='home_team_history_opponent_rating',
axis=1).mean(axis=1)


football_reduced_df['away_avg_rating_last_10'] =
football_train_df.filter(regex='away_team_history_rating',
axis=1).mean(axis=1)

football_reduced_df['away_avg_opponent_rating_last_10'] =
football_train_df.filter(regex='away_team_history_opponent_rating',
axis=1).mean(axis=1)
```

## 6. Modelos

We propose 4 models that are evaluated with different data filters presented in the data cleaning and preparation section. To summarize the testing process and avoid listing more than 10 dataset variants used to train the models, we will directly use the dataset that performed best among them. This dataset includes:

- Removing teams with fewer than 10 matches played.
- Cleaning rows with less than 70% of the data.
- Removing coach data, which represents the majority of missing values.

The four proposed models are K-Nearest Neighbors (KNN), Random Forest, XGBoost, and Neural Network.

**KNN (K Nearest Neighbors):** K Nearest Neighbor was the first algorithm we worked with, starting with an accuracy of 31%. After making thoughtful changes to the dataframe, we achieved 44%. With data scaling, we further improved to 47% accuracy.

**Random Forest:** This algorithm showed good performance from the beginning with an accuracy of 46%, which improved to 48% with standard scaling.

**XGBoost:** Hyperparameter optimization was performed for this algorithm using RandomizedSearchCV with the following parameter grid:

```
from xgboost import XGBClassifier
from sklearn.model_selection import RandomizedSearchCV
xgb = XGBClassifier()
param_grid = {
   'learning_rate': [0.01, 0.05, 0.1, 0.2],
   'max_depth': [3, 5, 7, 9],
   'n_estimators': [50, 100, 200, 500],
   'subsample': [0.5, 0.7, 0.9, 1.0],
   'colsample_bytree': [0.5, 0.7, 0.9, 1.0],
   'gamma': [0, 0.1, 0.2, 0.3, 0.4],
   'reg_alpha': [0, 0.1, 1, 10],
   'reg_lambda': [0, 0.1, 1, 10]
}
random_search = RandomizedSearchCV(xgb, param_grid, n_iter=20, scoring='accuracy', cv=2, verbose=1, n_jobs=-1)
random_search.fit(X_train, Y_train)


params = {'subsample': 0.9, 'reg_lambda': 10, 'reg_alpha': 0.1, 'n_estimators': 200, 'max_depth': 5, 'learning_rate': 0.1, 'gamma': 0.4, 'colsample_bytree': 0.5}
```

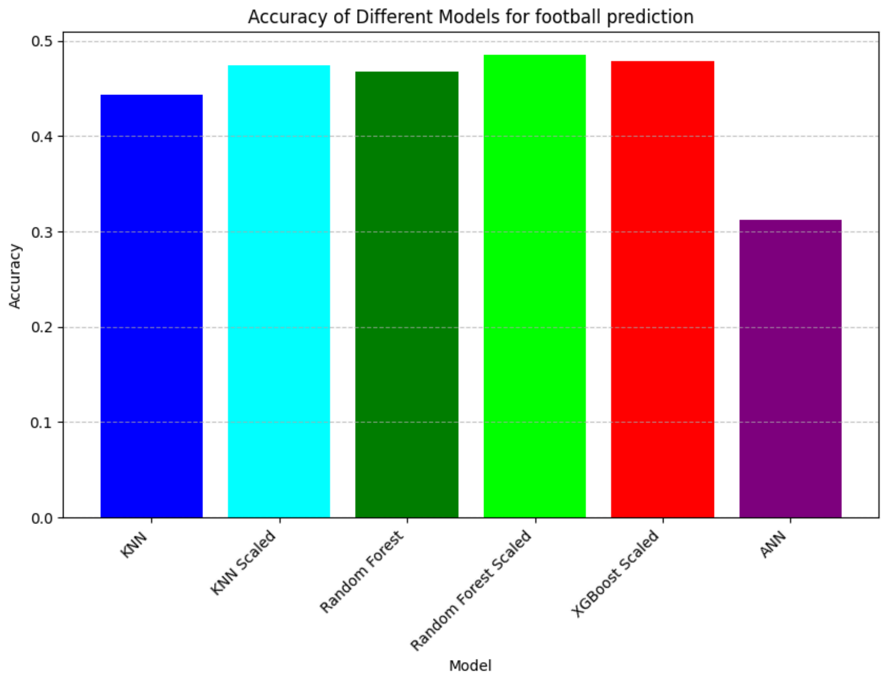
Upon training the model, an accuracy of 48% was achieved.

**Neural Network:** A neural network was implemented as follows:

```
# One hot encoding in Y
Y_train = to_categorical(Y_train)
Y_test = to_categorical(Y_test)

ann = Sequential()
ann.add(Dense(units=8, activation='tanh',
input_dim=X_train.shape[1]))
ann.add(Dense(units=8, activation='softmax'))
ann.add(Dense(units=8, activation='relu'))
ann.add(Dense(units=8, activation='softmax'))
ann.add(Dense(units=8, activation='relu'))
ann.add(Dense(units=8, activation='relu'))
ann.add(Dense(units=8, activation='tanh'))
ann.add(Dense(units=3, activation='relu'))

ann.compile( optimizer='adam',
           loss='categorical_crossentropy',
           metrics=['accuracy'] )

ann.fit(X_train, Y_train, epochs=20)
```

It performed at 31%, which is lower than the random chance accuracy of 33.3% for correctly predicting outcomes.

## 7. Model Comparation

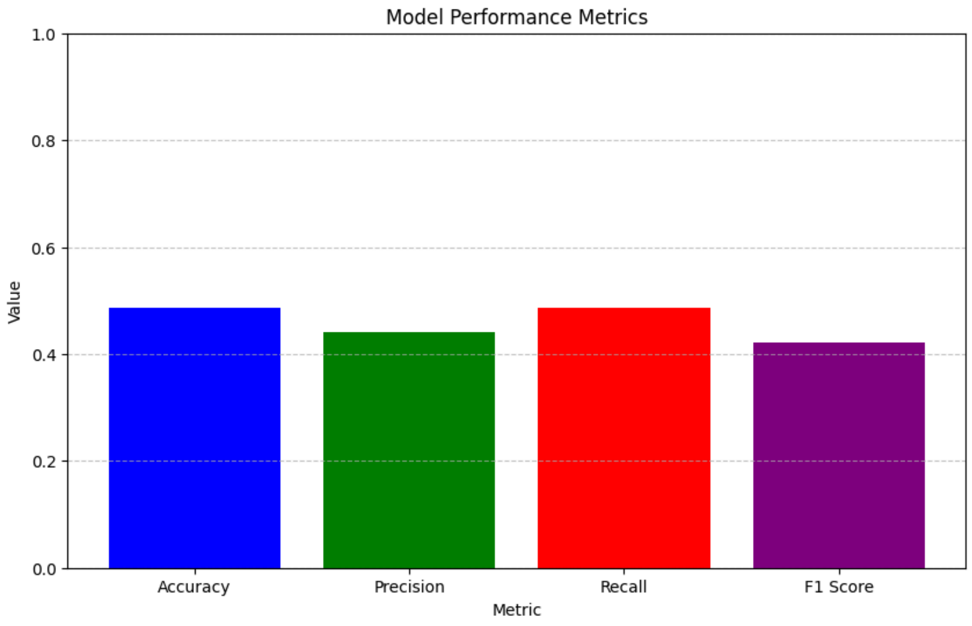


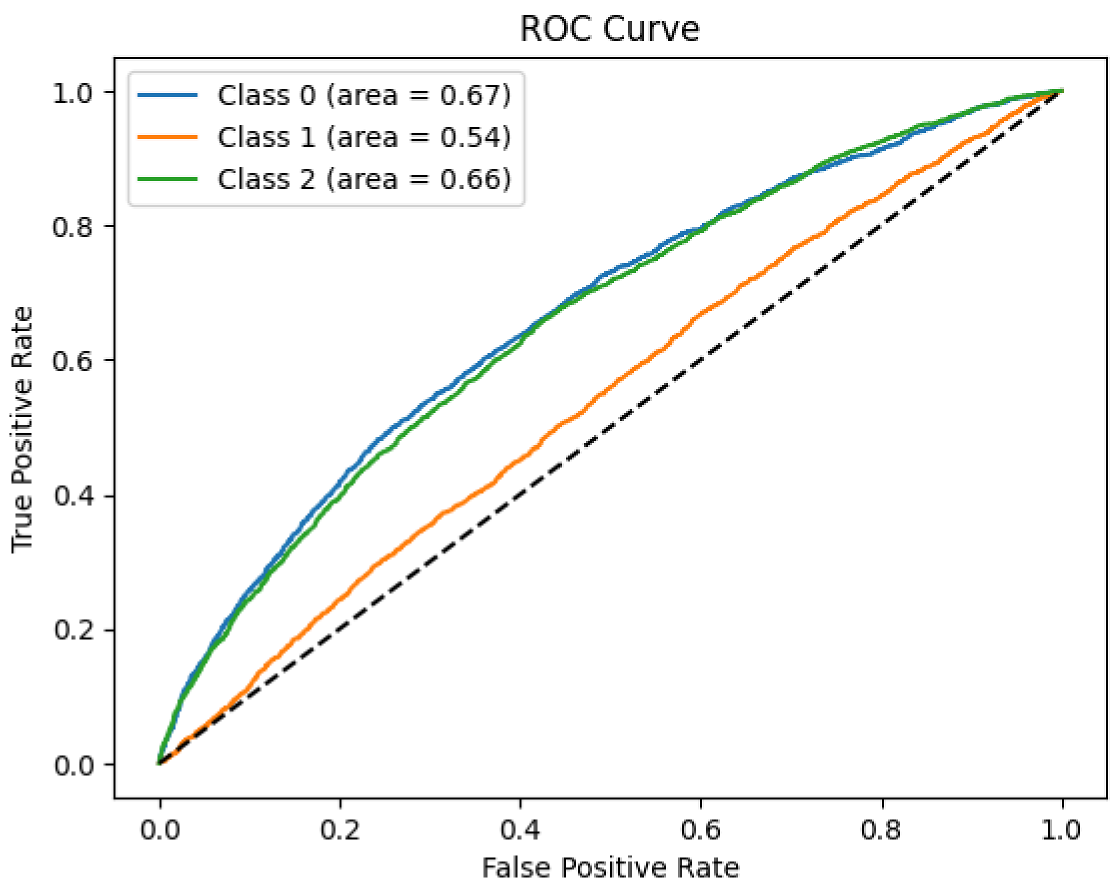
With these results in mind, we choose the Random Forest algorithm, which had the best performance despite being low.

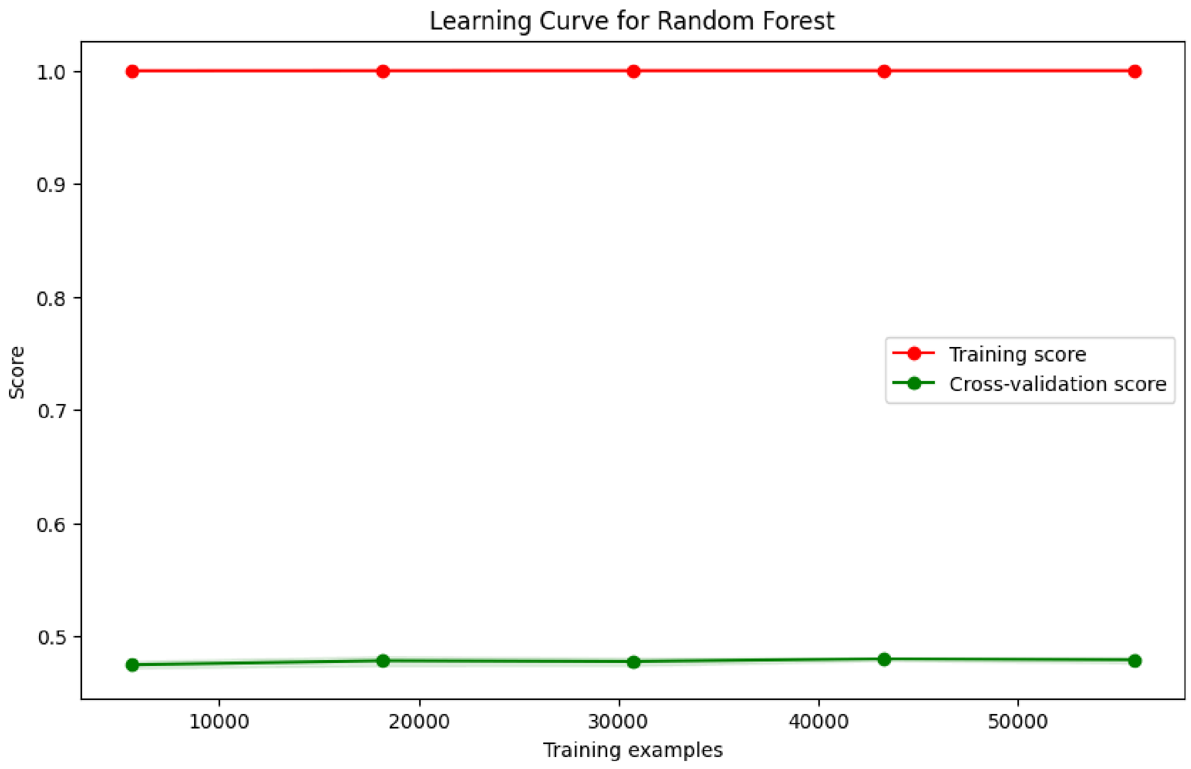
## 8. Evaluación del modelo

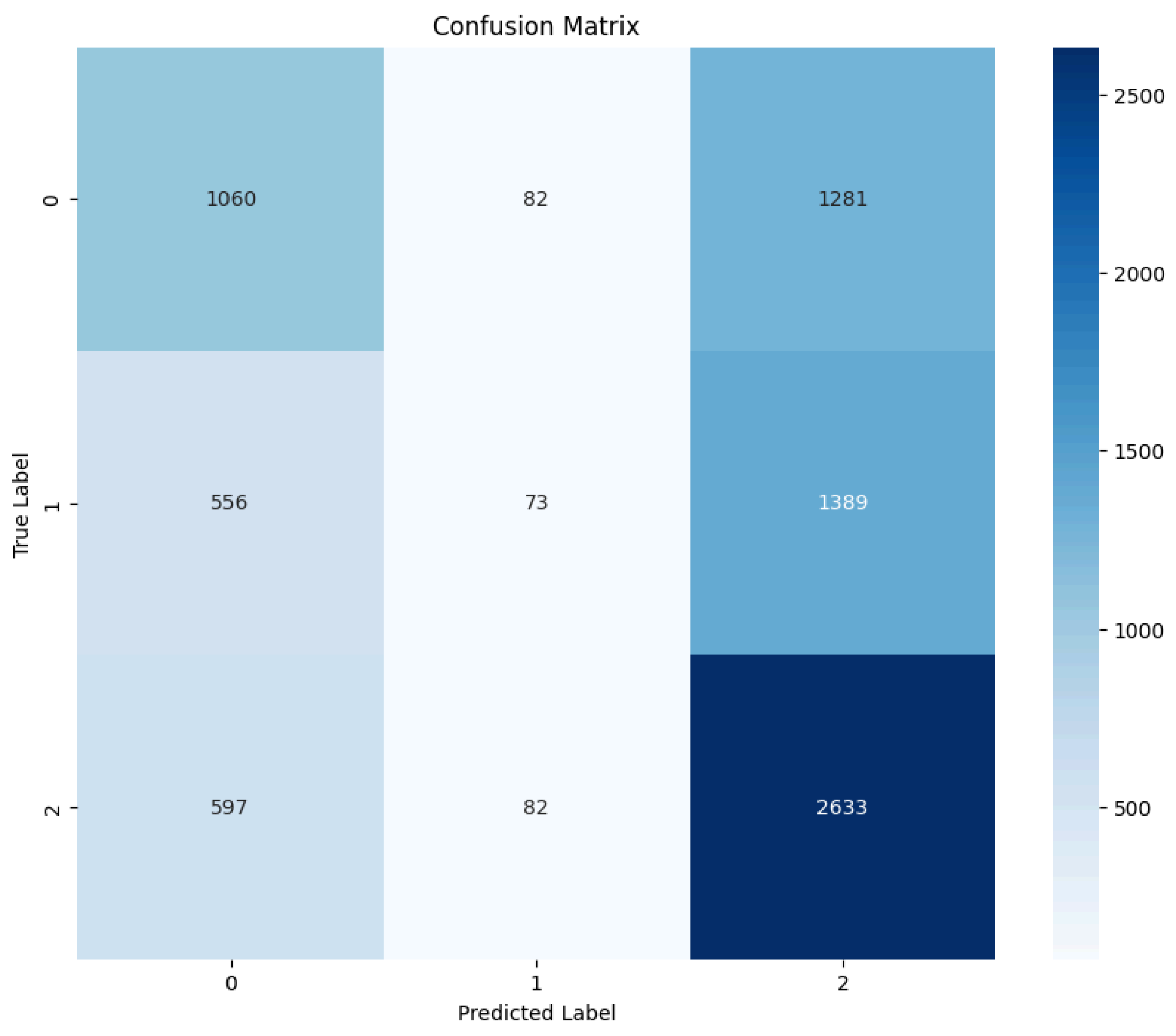
For the evaluation of our model (Random Forest), we proceeded to calculate another set of metrics, which are shown in the following table:

| Métrica | Valor |
|---|---|
| Accuracy | 0.4857 |
| Precision | 0.4419 |
| Recall | 0.4857 |
| F1 Score | 0.4208 |



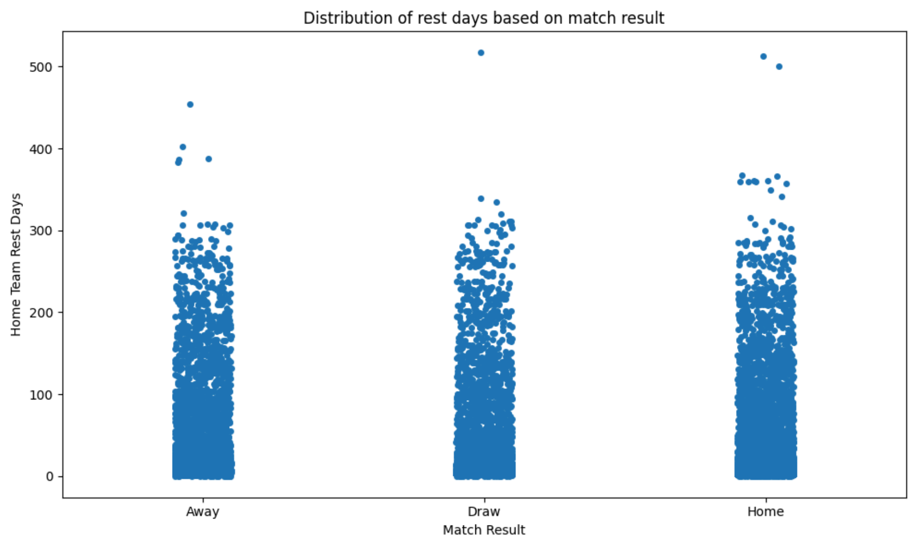

ROC Curve
Class 0 (area = 0.67)
Class 1 (area = 0.54)
Class 2 (area = 0.66)
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0


Learning Curve for Random Forest
Training score
Cross-validation score
Score
Training examples
1.0
0.9
0.8
0.7
0.6
0.5
10000
20000
30000
40000
50000

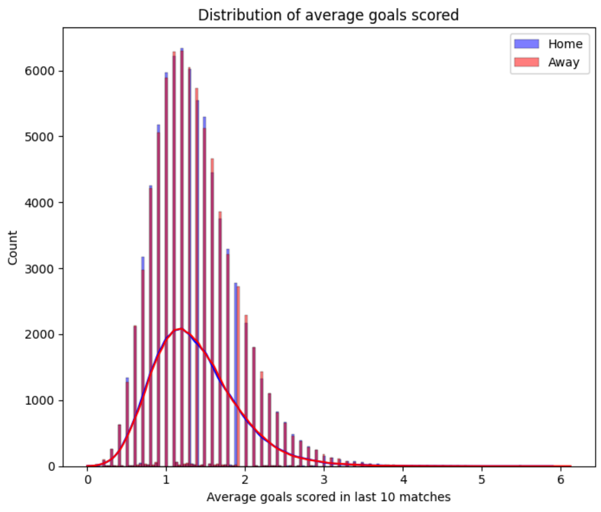


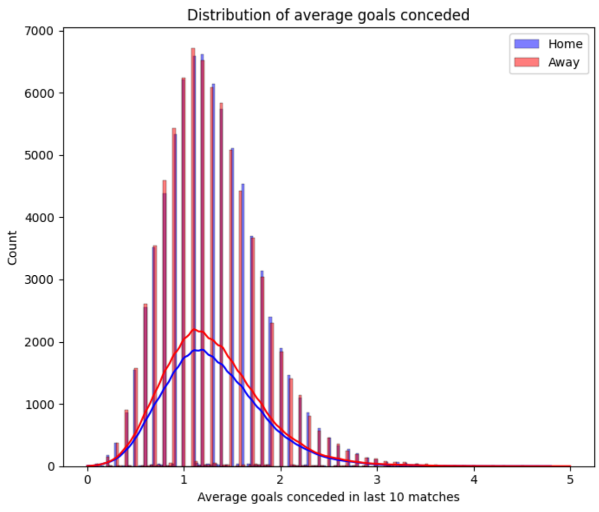
## 9. Business Questions:

Is there any relation between the rest days and the game result?



What is the distribution between the average goals scored and average goals conceded?


Distribution of average goals scored
Home
Away
Count
Average goals scored in last 10 matches


Distribution of average goals conceded
Home
Away
Count
Average goals conceded in last 10 matches