

# Football Match Probability Prediction

---

## Table of Contents

<b>Task 1: Please train a model to predict the results of matches in test.csv. The output format is the sample in sample_submission.csv .....</b>	<b>1</b>
<b>1. Overview and my approach .....</b>	<b>1</b>
<b>2. Implement .....</b>	<b>2</b>
2.1. FAML + XGBoost .....	2
<b>Task 2: Based on the historical data, write code to include results of last 5 matches between two teams.....</b>	<b>9</b>
<b>Task 3: We need to predict the probability of scores. For example, given a match data, output the probability .....</b>	<b>10</b>
Testing.....	12
<b>Improvement Approaches.....</b>	<b>13</b>
<b>Conclusion.....</b>	<b>13</b>

**Candidate: Nguyen Quoc Khanh**

**Email: [nqkdeveloper@gmail.com](mailto:nqkdeveloper@gmail.com)**

**Note:** I'm a perfectionist so I always want people to understand what I'm saying and writing. So my report has a theory behind everything I say. So whether you're an AI expert or not, if you're uncomfortable with it, just ignore it or give me a signal next time.

---

**Task 1: Please train a model to predict the results of matches in test.csv. The output format is the sample in sample\_submission.csv**

### **1. Overview and my approach**

In this problem, we have to predict the probability of the home team winning, probability of the away team winning and the probability of a draw so this is **regression** problem.

There are many types of regression models and it is difficult to test them all with large amounts of data. So I will use AutoML (Automated Machine Learning) which is a method of using tools and techniques that automate the process of building and deploying machine learning models. AutoML's goal is to reduce complexity and optimize the workflow of data analysts and developers, while enhancing the efficiency and accuracy of machine learning models.

**AutoML** automates tasks such as:

- Data preprocessing: Includes missing data processing, variable encoding, data normalization, and other data related handling.
- Model selection: Automatically search and select suitable machine learning models for the prediction problem.
- Auto-tuning of hyperparameters: Automatically search and adjust model's hyper-parameters to optimize performance.
- Automatic evaluation and comparison: Automatically perform the evaluation and comparison process between different models to choose the best model.

Another approach to this problem is to use the Recurrent Neural Network (RNN) family of models (**I do not mention in this solution because of the time limit and the length of the report**). RNN is a model class in Machine Learning designed for sequential and predictive data processing in situations involving order or sequence. Specific models have been applied and topped the leaderboard in the competition such as LSTM, Bidirectional LSTM, GRU.

Especially the LSTM (Long Short-Term Memory). LSTM is used for regression problems (continuous value prediction) in Machine Learning because it has the ability to process sequential data and capture long-term dependencies in the data..

With its ability to process sequential data, LSTM can understand and learn correlation and dependency patterns between data points in time series. Data features such as order and time can be modeled using LSTMs, which capture complex and time-varying relationships in the data..

In addition, LSTM is also capable of long-term data processing and solves the vanishing or exploding gradient problem that traditional RNN models often encounter. With gate mechanisms in LSTM such as input gate, forget gate and output gate, it is possible to regulate the flow of information and decide which data should be stored and which should be forgotten. This helps the LSTM to maintain and use important information from the past to predict future value.

## **2. Implement**

### **2.1. FAML + XGBoost**

FLAML is a lightweight Python library that finds accurate machine learning models automatically, efficiently and economically. It frees users from selecting learners and hyperparameters for each learner.

This is first my approach for AutoML since for common machine learning tasks like classification and regression, it quickly finds quality models for user-provided data with low computational resources.

Full source code for FAML in [FAML AutoML.ipynb](#)

After handle with missing values and nan values, training with FAML

## Result

```
print('Best ML learner:', automl.best_estimator)
print('Best hyperparameter config:', automl.best_config)
print('Best log_loss on validation data: {0:.4g}'.format(automl.best_loss))
print('Training duration of best run: {0:.4g} s'.format(automl.best_config_train_time))
```

```
Best ML learner: xgboost
Best hyperparameter config: {'n_estimators': 396, 'max_leaves': 11, 'min_child_weight': 36.982
6704053053, 'learning_rate': 0.11980646960410847, 'subsample': 0.9048064340763577, 'colsample
_bylevel': 0.7089206179282082, 'colsample_bytree': 0.6165783556569796, 'reg_alpha': 0.1289766
547000545, 'reg_lambda': 0.019843909838391162}
Best log_loss on validation data: 1.017
Training duration of best run: 58.79 s
```

We can see the best ML learner is XGBoost with its configurations.

I actually switched over to EvalML which is the best rated platform in the library with AutoML but i stuck with some unheard errors and also had previous research results and same results with FAML ( XGBoost) so I'll go straight to the model implementation.

My approach is to construct sets of features to feed into the XGBoost algorithm for training, and see how far it can go.

## Sub utils function for for convenience and better speed of manipulation

```
def is_cup_conversion(df):
    '''convert is_cup column from boolean to 0/1'''
    df.is_cup = df.is_cup.apply(lambda x: np.multiply(x, 1) )
    df['is_cup'].fillna(0, inplace=True) # missing value filled with 0, ie non-cup game
    return df
```

Converting the history columns into **long form**, for convenience and better speed of manipulation. The long-form data structure makes it easier for us to manipulate and analyze the data during the processing phase, such as calculating averages, sorting, and filtering data based on conditions.

```
def convert_to_long_form(df, feature_columns, history_columns):
    '''converting the history columns into long form, for convenience and better speed of manipulation'''
    stubname=[
        'home_team_history_match_date',
        'home_team_history_is_play_home',
        'home_team_history_is_cup',
        'home_team_history_goal',
        'home_team_history_opponent_goal',
        'home_team_history_rating',
        'home_team_history_opponent_rating',
        'home_team_history_coach',
        'home_team_history_league_id',
        'away_team_history_match_date',
        'away_team_history_is_play_home',
        'away_team_history_is_cup',
        'away_team_history_goal',
        'away_team_history_opponent_goal',
        'away_team_history_rating',
        'away_team_history_opponent_rating',
        'away_team_history_coach',
        'away_team_history_league_id']
    df_long=pd.wide_to_long(df[feature_columns+history_columns], stubnames=stubname, i='id', j='match', sep='_')
    df_long=df_long.reset_index().sort_values(by=['id','match'])
    return df_long
```

**Wide Format**

Team	Points	Assists	Rebounds
A	88	12	22
B	91	17	28
C	99	24	30
D	94	28	31

**Long Format**

Team	Variable	Value
A	Points	88
A	Assists	12
A	Rebounds	22
B	Points	91
B	Assists	17
B	Rebounds	28
C	Points	99
C	Assists	24
C	Rebounds	30
D	Points	94
D	Assists	28
D	Rebounds	31

*Diff. Between wide format and long format*

Add features indicating the time gaps between the match days of previous matches for the home and away teams, and how long they played 3 matches and 6 matches.

```
def add_match_interval(df):
    '''add features indicating the time gaps between the match days of previous matches for the home and away teams,
    and how long they played 3 matches and 6 matches, as a reflection of fatigue factor'''
    for i in [1,2,5]:
        home_team_history_match_interval = (df.groupby('id').match_date.first().apply(pd.Timestamp) \
        - df.query('match ==1').groupby('id').home_team_history_match_date.first().apply(pd.Timestamp)) / np.timedelta64(1, "D")
        df=df.join(home_team_history_match_interval.to_frame(name=f'home_team_history_match_interval_{i}'), on='id')

        away_team_history_match_interval = (df.groupby('id').match_date.first().apply(pd.Timestamp) \
        - df.query('match ==1').groupby('id').away_team_history_match_date.first().apply(pd.Timestamp)) / np.timedelta64(1, "D")
        df=df.join(away_team_history_match_interval.to_frame(name=f'away_team_history_match_interval_{i}'), on='id')

    return df
```

These time gaps reflect the interval between consecutive matches and can provide insights into the fatigue factor of the teams.

**Calculate goal difference and the point derived from it (3 points for win, 1 point for draw and 0 point for lose) of each past match**

```
def result_point(goal_difference):
    if goal_difference > 0:
        return 3
    elif goal_difference==0:
        return 1
    elif goal_difference < 0:
        return 0
    else:
        return np.NaN
```

Next **adding form features for home and away teams**, including rating difference and goal difference for each previous match, and the average points got, average goal scored, average goal conceded and average goal difference up to that match as previous n-match form. (See code in my notebook, too long to snap here)

**Add features about whether the teams have recent coach change ?**

In the context of the analysis, when a team undergoes a coach change, it implies that there has been a shift in leadership and management strategies. This change can sometimes bring about a positive impact on the team's dynamics, motivation, and gameplay. The phrase "rebound of form from the bottom" refers to the idea that a team that was previously struggling or performing poorly might experience a resurgence or improvement in their performance after a coach change.

```
def has_coach_change(df):
    '''add features indicating whether home and away teams have changed coach'''

    df['home_has_coach_change'] = df.apply(lambda r: any([(r['home_team_coach_id']!=r['home_team_history_coach_{i}'])
                                                         & (np.isnan(r['home_team_history_coach_{i}'])==False) for i in range(1,11) ]), axis=1)
    df['home_has_coach_change'] = df['home_has_coach_change'].apply(lambda x: np.multiply(x, 1) )
    df['home_has_coach_change'].fillna(0, inplace=True)
    df['away_has_coach_change'] = df.apply(lambda r: any([(r['away_team_coach_id']!=r['away_team_history_coach_{i}'])
                                                         & (np.isnan(r['away_team_history_coach_{i}'])==False) for i in range(1,11) ]), axis=1)
    df['away_has_coach_change'] = df['away_has_coach_change'].apply(lambda x: np.multiply(x, 1) ).fillna(0, inplace=True)
    df['away_has_coach_change'].fillna(0, inplace=True)
    return df
```

And no matter how I kept adding features, the most salient ones remain the medium rating differences of past 10 matches of the away and home team. Although the host of this competition published an article and notebook on Elo ratings, examination of rating changes between matches for a few teams shows that the ratings are not Elo ratings for the team, at least not directly used. And the host confirmed they are calculated pairwise for the home and away teams, and combining the attack strength of one side and defense strength of the other side. In other notebook of other user, they have shown the ratings are most likely the expected goals scored by the team of the match.

The rating differences of the past matches are informative as a rough indicator of the general strength of a team, but cannot be directly carried to the match in interest. The host does not provide the ratings for the matches to be predicted for the obvious reason. Though the host said the ratings can be calculated based on past ratings, without knowing what is the model used, it is a hard code to crack. The series of ratings in each row are combining the offensive strength of the team and the defensive strength of different opposing teams, so doing an autocorrelation on the series of ratings does not look hopeful. Statisticians of football results modeling usually fit their models with seasons of data of one or more leagues, it is not certain whether it is the appropriate approach trying to be learnt and applied for the case here.

So I decided to construct a simple model of expected goal as a feature. A simple idea used by David Spiegelhalter, a Cambridge statistician, and some pundits goes like this: first you get

**The average goals scored by an average team in a league or competition, then the offensive strength of a team is the the average goals scored by that team divided by the league average, and the defensive strength of the team is its average goals conceded divided by the league average.** With the average goals scored by a home team and the away team in the league or competition, for every match we have:

- Expected goals scored by the home team = Average goals scored by a home team in the league or competition \* Offensive strength of the home team \* Defensive strength of the away team
- Expected goals scored by the away team = Average goals scored by an away team in the league or competition \* Offensive strength of the away team \* Defensive strength of the home team

→ Pundits would feed these expected goals data into some Poisson distribution model to get the probabilities of every possible score line and the prediction of match result, but I leave it for XGBoost to figure out modeling.

```
def add_attack_defense_strength_features(df):
    '''getting goal rates of the league/competition, and calculate the attack and defense strength of a team relative to the league/competition,
    and the expected goals based on this'''

    home_history_columns=['home_team_history_match_date',
                          'home_team_history_is_play_home',
                          'home_team_history_is_cup',
                          'home_team_history_goal',
                          'home_team_history_opponent_goal',
                          'home_team_history_rating',
                          'home_team_history_opponent_rating',
                          'home team history coach']
```

### **Full code implement in add\_attack\_defense\_strength\_features() function**

The average goals data of every league in this dataset can be found by pooling the history feature columns data and groupby league id.

As for the team goals data, it should be aligned with the league id of the match to be predicted, as different tournaments have different competition levels. But I thought within the 10 previous matches data, cup matches normally only feature a couple of times to be reliable, so in the first trial I used data of all ten matches without regard to the league id.

To collect more league goals data of a team beyond the 10 past matches, normally we should use the team identifiers. But as the team names are masked in the test set following concern about data leak, and no team ids are provided, the coach id becomes an imperfect proxy, as coach records can be coming from different teams for coaches who have changed teams. So after setting up this set of features by groupby coach id and league id and starting the XGBoost training process, I did not have much expectation and did internet surfing elsewhere.



## Hyperparameter tuning on an XGBoost classifier










```
# Define the parameter grid
param_grid = {
    'max_depth': [3, 5, 7],
    'learning_rate': [0.01, 0.1, 0.5],
    'subsample': [0.6, 0.8, 1.0],
    'colsample_bytree': [0.6, 0.8, 1.0]
}

# Create an XGBoost classifier
xgb_model = xgb.XGBClassifier(objective='multi:softprob', num_class=3, eval_metric='mlogloss', use_label_encoder=False)

# Perform grid search
grid_search = GridSearchCV(estimator=xgb_model, param_grid=param_grid, cv=3)
grid_search.fit(X_train, y_train)

# Print the best parameters and best score
print("Best Parameters: ", grid_search.best_params_)
print("Best Score: ", grid_search.best_score_)
```

This final version has a public score of 0.99853 and private score of 0.99302.

#	△	Team	Members	Score	Entries	Last	Solution
1	—	Sze Yeung		0.97185	22	1y	
		Bookmakers		0.97884			
2	↗ 3	Fernando SeraQueVence		0.99115	72	1y	
3	↗ 1	NoviceDL		0.99129	58	1y	
4	↗ 3	Tang Tun Yu		0.99154	41	1y	
5	↗ 4	Treble Chance 12X		0.99167	92	1y	
6	↗ 2	BOOBA		0.99168	45	1y	
7	↗ 5	Himanshu Nayal		0.99173	86	1y	
8	↗ 3	Hamza Ouammou		0.99173	13	1y	

LeaderBoard for Problem (1 year ago)

Result for XGBoost:

- Private Score: 0.99302
- Public Score: 0.99853

According to the results of the XGBoost model, the results are still not optimized compared to the rankings of a year ago, we need to improve further or go a different direction.



**Task 2: Based on the historical data, write code to include results of last 5 matches between two teams.**

```
# Sort the DataFrame by match_date in ascending order
train = train.sort_values('match_date')

# Create columns for the results of the last 5 matches between home and away teams
for i in range(1, 6):
    home_column = f'home_recent_match_{i}'
    away_column = f'away_recent_match_{i}'

    # Get the previous match results for the home team
    train[home_column] = train.groupby('home_team_name')['target'].shift(i)

    # Get the previous match results for the away team
    train[away_column] = train.groupby('away_team_name')['target'].shift(i)

# Update the home and away match results based on team switching roles
for index, row in train.iterrows():
    home_team = row['home_team_name']
    away_team = row['away_team_name']
    match_result = row['target']

    # Check if the previous match involved the same teams but with switched roles
    previous_home_result = row[home_column]
    previous_away_result = row[away_column]
    if previous_home_result == match_result and previous_away_result == match_result:
        # Update the home match result
        train.at[index, home_column] = away_team
        # Update the away match result
        train.at[index, away_column] = home_team

# Drop rows with missing values (first 5 rows will have NaN values for recent match columns)
train = train.dropna(subset=[f'home_recent_match_{i}' for i in range(1, 6)] + [f'away_recent_match_{i}' for i in range(1, 6)])
```

Here is a summary of the algorithm:

1. The DataFrame **train** is sorted by the **match\_date** column in ascending order to ensure the matches are ordered chronologically.
2. For each of the last 5 matches (indexed as **i** from 1 to 5), new columns named **home\_recent\_match\_i** and **away\_recent\_match\_i** are created.
3. The code then iterates over the rows of the DataFrame and assigns the previous match results for the home and away teams in the respective columns.
4. Additionally, it checks if the previous match involved the same teams but with switched roles (i.e., the home team became the away team and vice versa). If this condition is met, it updates the match result columns accordingly.
5. Finally, the rows with missing values in the recent match columns (the first 5 rows) are dropped.

The resulting DataFrame will contain the original columns (**id**, **target**, **home\_team\_name**, **away\_team\_name**) along with the updated columns representing the results of the last 5 matches between the teams.

```
columns = ['id', 'target', 'home_team_name', 'away_team_name'] + [f'home_recent_match_{i}' for i in range(1, 6)] + [f'away_recent_match_{i}' for i in range(1, 6)]

# Display the selected columns
train_subset = train.loc[:, columns]
train_subset
```

	id	target	home_team_name	away_team_name	home_recent_match_1	home_recent_match_2	home_recent_match_3
7042	11984658	home	Liverpool	Everton	Everton	home	home
7254	11989931	away	Manchester United	Manchester City	home	home	draw
7342	11989312	away	Xanthi	Aris	home	draw	Aris
7347	11989929	draw	Leicester City	Aston Villa	home	away	draw
8714	11876498	home	Burton Albion	Milton Keynes Dons	home	away	home
...	...	...	...	...	...	...	...
110932	17948949	away	New York RB II	Hartford Athletic	Hartford Athletic	Hartford Athletic	home
110933	18030016	draw	Zamora Fútbol Club	Hermanos Colmenares	draw	Hermanos Colmenares	home
110935	17715497	draw	São Bernardo	Água Santa	home	home	draw
110936	17944153	away	Everton	La Serena	away	draw	draw
110937	17786297	home	Colón	Arsenal de Sarandí	draw	draw	draw

**Task 3: We need to predict the probability of scores. For example, given a match data, output the probability**

As I calculated earlier, I already have the parameters

- home\_team\_coach\_attack\_strength
- home\_team\_coach\_defense\_strength
- away\_team\_coach\_attack\_strength
- away\_team\_coach\_defense\_strength
- home\_team\_coach\_expected\_goal
- away\_team\_coach\_expected\_goal

Với các tham số trên, ta đã sẵn sàng để dự đoán tỉ số của trận đấu sử dụng **The Poisson Distribution formula**,

The formula is as follows:

$$P(x; \mu) = \frac{e^{-\mu} \times (\mu^x)}{x!}$$

- P(x; μ): Probability of x events occurring

- e: Euler's number, approximately 2.71828
- $\mu$ : Average rate of occurrence
- x: Number of events

Assume that Team A's scoring probability is 1.623 and Team B's - 0.824. Looking at the matchup between two, we are interested in 0-5 goals for each one. Using the aforementioned tools, we are going to get the following results.

### Team A vs. Team B Poisson Distribution

Goals	0	1	2	3	4	5
Team A	19.73%	32.02%	25.99%	14.06%	5.07%	1.85%
Team B	43.86%	36.14%	14.89%	4.09%	0.84%	0.14%

Code for implement:

```
# Calculate the goal probabilities for each row in subset_task3
for index, row in subset_task3.iterrows():
    team_a_scoring_prob = row['home_team_coach_expected_goal']
    team_b_scoring_prob = row['away_team_coach_expected_goal']

    # Calculate the goal probabilities for Team A
    team_a_goal_probs = []
    for goals in range(6):
        prob = (math.exp(-team_a_scoring_prob) * (team_a_scoring_prob ** goals)) / math.factorial(goals)
        team_a_goal_probs.append(prob)

    # Calculate the goal probabilities for Team B
    team_b_goal_probs = []
    for goals in range(6):
        prob = (math.exp(-team_b_scoring_prob) * (team_b_scoring_prob ** goals)) / math.factorial(goals)
        team_b_goal_probs.append(prob)

    # Add the goal probabilities as columns to the 'subset_task3' DataFrame
    subset_task3.at[index, 'goals_away_0'] = team_b_goal_probs[0]
    subset_task3.at[index, 'goals_away_1'] = team_b_goal_probs[1]
    subset_task3.at[index, 'goals_away_2'] = team_b_goal_probs[2]
    subset_task3.at[index, 'goals_away_3'] = team_b_goal_probs[3]
    subset_task3.at[index, 'goals_away_4'] = team_b_goal_probs[4]
    subset_task3.at[index, 'goals_away_5'] = team_b_goal_probs[5]

    subset_task3.at[index, 'goals_home_0'] = team_a_goal_probs[0]
    subset_task3.at[index, 'goals_home_1'] = team_a_goal_probs[1]
    subset_task3.at[index, 'goals_home_2'] = team_a_goal_probs[2]
    subset_task3.at[index, 'goals_home_3'] = team_a_goal_probs[3]
    subset_task3.at[index, 'goals_home_4'] = team_a_goal_probs[4]
    subset_task3.at[index, 'goals_home_5'] = team_a_goal_probs[5]
```

subset_task3																
	id	home_team_coach_expected_goal	away_team_coach_expected_goal	goals_away_0	goals_away_1	goals_away_2	goals_away_3	goals_away_4	goals_away_5	goals_home_0	goals_home_1	goals_home_2	goals_home_3	goals_home_4	goals_home_5	
0	11913628	1.053336	1.339833	0.262939	0.351243	0.234001	0.104482	0.034886	0.009320	0.184419	0.296832	0.287841	0.181541	0.072774	0.028279	
1	11913633	1.558596	2.879091	0.068628	0.183854	0.346581	0.219938	0.147307	0.078830	0.210431	0.327877	0.235582	0.132788	0.051741	0.018129	
2	11913634	1.975901	1.194090	0.302980	0.381768	0.216002	0.085976	0.025686	0.006129	0.138636	0.273932	0.270631	0.178247	0.088049	0.034795	
5	11913637	1.496778	2.954680	0.032101	0.133935	0.227407	0.223964	0.185430	0.097755	0.223880	0.338054	0.250751	0.125106	0.048814	0.014014	
6	11913639	1.823686	2.444048	0.088809	0.212185	0.259270	0.211223	0.129080	0.063086	0.148071	0.288991	0.270266	0.173301	0.083343	0.032066	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
72701	18450091	0.972917	0.798980	0.449963	0.359338	0.145481	0.091844	0.007625	0.001218	0.377979	0.387742	0.178891	0.058015	0.014111	0.003749	
72704	18450334	3.141749	1.108582	0.330023	0.365881	0.202795	0.074939	0.020769	0.004805	0.043207	0.138748	0.213240	0.223316	0.175400	0.110213	
72706	18450343	1.043195	1.120240	0.328302	0.395424	0.204881	0.075431	0.021435	0.004798	0.352327	0.387548	0.191711	0.068884	0.017398	0.003627	
72709	18450640	1.762708	1.150076	0.318632	0.384132	0.209379	0.080283	0.023076	0.006308	0.171580	0.300445	0.288881	0.188623	0.069020	0.024332	
72710	18450641	1.528582	1.244143	0.288188	0.359547	0.223042	0.092499	0.028770	0.007159	0.217281	0.331894	0.233175	0.128829	0.049188	0.015011	

41310 rows x 15 columns

Multiply the corresponding probabilities for each team to get the joint probability of a specific score combination. In this case, example I want to calculate the probability of a 3-1 score, which corresponds to Team A scoring 3 goals and Team B scoring 1 goal.

- Probability of Team A scoring 3 goals: 14.06%
- Probability of Team B scoring 1 goal: 36.14%

Multiply the probabilities together to get the overall probability of the specific score combination:

- Probability of A:B - 3:1 = 14.06% \* 36.14%

So, the probability of the match result being A:B - 3:1 is the product of these two probabilities.

## Testing

```
# Main program
if __name__ == "__main__":
    # Input match ID and predicted score
    match_id = int(input("Enter match ID: "))
    predicted_score = input("Enter predicted score (e.g., 3-1 (Range Scores: 0-5))": ")

    # Call the predict_match_results function
    predict_match_results(match_id, predicted_score, subset_task3)
```

```
Enter match ID: 11913633
Enter predicted score (e.g., 3-1 (Range Scores: 0-5)): 1-1
Goal probabilities for 1-1:
Home team: 32.79774%
Away team: 18.38540%
Overall goal probabilities: 6.03000%
```

## Theory Reference:

- <https://www.bettingwell.com/sports-betting-guide/football-bettors-guide/how-predict-score-football-betting>

## Improvement Approaches

- **Data Augmentation:** Generate synthetic data points or augment the existing data to increase the diversity and quantity of the training set. Some additional factors that could be taken into account are:
  - Angle, height and direction of a shot.
  - Player's and goalkeeper's skill.
  - Type of assist (cross, pass, rebound, etc).
  - Number of touches before the shot.

References for this: <https://xgscore.io/xg-statistics/explained>

- **Time Series Analysis:** Consider the temporal aspect of the data by applying time series analysis techniques. Explore seasonality patterns, trends, and other time-dependent factors that can impact match outcomes. Incorporate time series models like ARIMA, SARIMA, GRU or LSTM (BI-LSTM) (Long Short-Term Memory) to capture and utilize this temporal information.
- **Transfer Learning:** Explore the potential of transfer learning by leveraging pre-trained models or embeddings from related domains, such as player performance in other sports, to extract useful representations and features for the Football Match Probability Prediction problem.

## Conclusion

In conclusion, this technical assessment test for the Football Match Probability Prediction project demonstrates my understanding and ability to work with football match data, apply machine learning techniques, and predict match outcomes and scores. The source code and report provide an overview of the approach, methodologies, and results obtained during the analysis.

I believe that my work showcases my technical skills, problem-solving abilities, and dedication to delivering high-quality solutions. I appreciate the opportunity to participate in this assessment and look forward to any feedback or further discussions.

Please feel free to reach out to me at [nqkdeveloper@gmail.com](mailto:nqkdeveloper@gmail.com) if you have any questions or require additional information. Thank you for considering my application.