1 Predicting the 2024 UEFA Euro Using Poisson Distribution

Football predictions have always been a fascinating topic for data scientists and enthusiasts alike. In this project, we leverage the power of the Poisson distribution to forecast the outcomes of the 2024 UEFA European Championship. This method is particularly well-suited for predicting football scores due to the nature of the game, where goals are relatively rare events.

In [1]:
```
#intalling the necesary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.stats import poisson
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_curve, roc_auc_score
import matplotlib.colors as mcolors
import warnings
pd.set_option('display.max_rows', 10000)
pd.set_option('display.max_columns', 500)
warnings.filterwarnings('ignore')
```

2 Data Collection

The first step in our analysis involved gathering historical match data. The dataset sourced from Kaggle contains the results of international football matches starting from the very first official match in 1872 up to 2024.

In [2]:
```
#Reading the
df = pd.read_csv('results.csv',sep=',',)
#Dropping matches without final result
df=df[~df['home_score'].isna()]
```

In [3]:
```
1 df
```

Out[3]:

	date	home_team	away_team	home_score	away_score	tournament	city	cour
0	1872-11-30	Scotland	England	0.0	0.0	Friendly	Glasgow	Scotl
1	1873-03-08	England	Scotland	4.0	2.0	Friendly	London	Engl
2	1874-03-07	Scotland	England	2.0	1.0	Friendly	Glasgow	Scotl
3	1875-03-06	England	Scotland	2.0	2.0	Friendly	London	Engl
4	1876-03-04	Scotland	England	3.0	0.0	Friendly	Glasgow	Scotl
...	...	...	...	...	...	...	...	
47365	2024-07-02	Costa Rica	Paraguay	2.0	1.0	Copa América	Austin	Un Sta
47366	2024-07-04	Argentina	Ecuador	1.0	1.0	Copa América	Houston	Un Sta
47367	2024-07-05	Germany	Spain	1.0	2.0	UEFA Euro	Stuttgart	Germ
47368	2024-07-05	Portugal	France	0.0	0.0	UEFA Euro	Hamburg	Germ
47369	2024-07-05	Venezuela	Canada	1.0	1.0	Copa América	Arlington	Un Sta

47370 rows × 9 columns

3 ELO Ranking System

The Elo ranking system is a method for calculating the relative skill levels of teams. It is useful for ranking teams based on their performance in previous matches. The Elo rating is dynamic, meaning it changes based on the results of matches, rewarding teams for wins and penalizing them for losses, while also considering the strength of the opponents.

To create our dataset, we need to define some parameters, such as the importance of each tournament in the points exchanged by the Elo ranking system.

In [4]:

```
#Auxiliary funtions for update the ratings
confederation_tournaments=['AFC Asian Cup','African Cup of Nations','U
confederation_clasification=['UEFA Euro qualification','African Cup o
def k_value(tournament):
    k=5
    if tournament == 'Friendly':
        k=10
    elif tournament in confederation_clasification:
        k=20
    elif tournament == 'FIFA World Cup qualification':
        k=25
    elif tournament in confederation_tournaments:
        k=40
    elif tournament == 'FIFA World Cup':
        k=50
    return k

def expected_result(loc,aw):
    dr=loc-aw
    we=(1/(10**(-dr/400)+1))
    return [np.round(we,3),1-np.round(we,3)]

def actual_result(loc,aw):
    if loc<aw:
        wa=1
        wl=0
    elif loc>aw:
        wa=0
        wl=1
    elif loc==aw:
        wa=0.5
        wl=0.5
    return [wl,wa]

def calculate_elo(elo_l,elo_v,local_goals,away_goals,tournament):

    k=k_value(tournament)
    wl,wv=actual_result(local_goals,away_goals)
    wel,wev=expected_result(elo_l,elo_v)

    elo_ln=elo_l+k*(wl-wel)
    elo_vn=elo_v+k*(wv-wev)

    return elo_ln,elo_vn,wel,wev
```

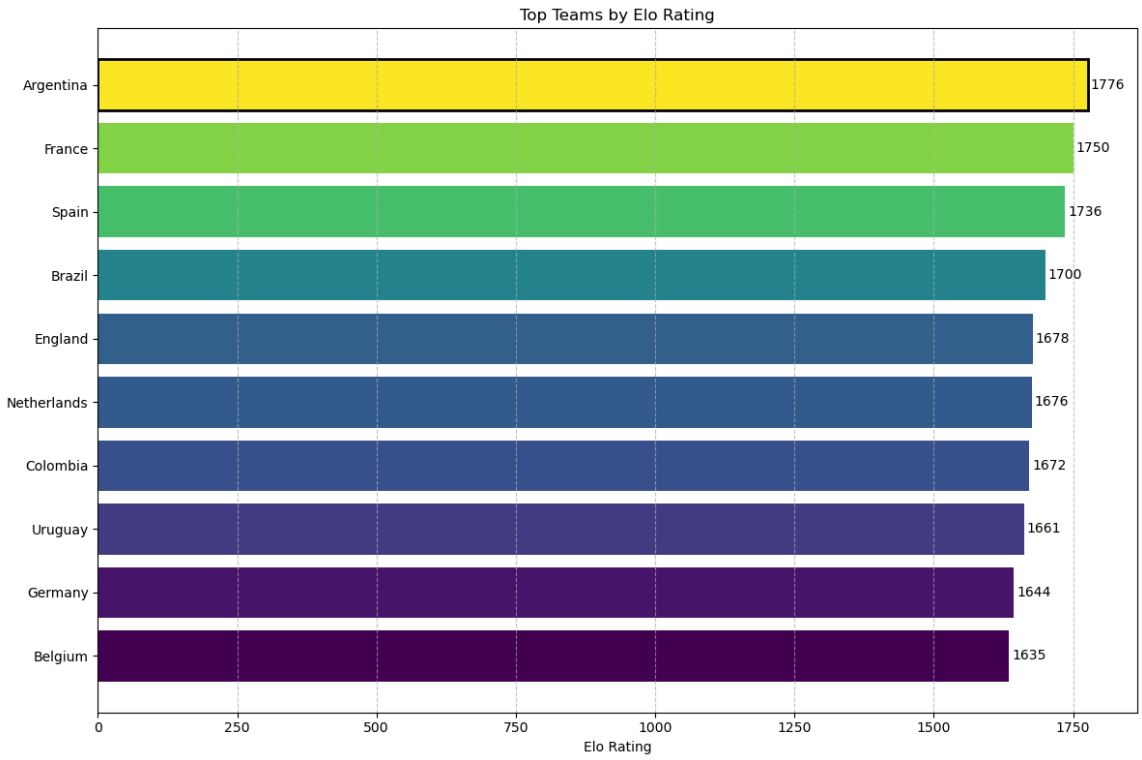
In [5]:
```
#Calculating the elo points for all historical internatinal matches
current_elo={}
for idx,row in df.iterrows():

    local=row['home_team']
    away=row['away_team']
    local_goals=row['home_score']
    away_goals=row['away_score']
    tournament = row['tournament']


    if local not in current_elo.keys():
        current_elo[local]=1300

    if away not in current_elo.keys():
        current_elo[away]=1300

    elo_l=current_elo[local]
    elo_v=current_elo[away]
    elo_ln,elo_vn, wel,wev=calculate_elo(elo_l,elo_v,local_goals,away_

    current_elo[local]=elo_ln
    current_elo[away]=elo_vn

    df.loc[idx,'Elo_h_after']=elo_ln
    df.loc[idx,'Elo_a_after']=elo_vn
    df.loc[idx,'Elo_h_before']=elo_l
    df.loc[idx,'Elo_a_before']=elo_v
    df.loc[idx,'probH']=wel
    df.loc[idx,'probA']=wev
```

Once we have processed the historical data of international matches, we can calculate the final Elo points for each team. This gives us the current Elo ranking, which reflects the latest performance levels of the teams.

In [6]:
```
elos=pd.concat([df[['date','home_team','Elo_h_after']].rename(columns
elos.sort_values(by='date', ascending=False,inplace=True)
elos.drop_duplicates('Team',inplace=True)
elos.sort_values(by='Elo', ascending=False, inplace=True)
elos.reset_index(drop=True, inplace=True)
elos['position']=elos.index+1
```

In [7]:

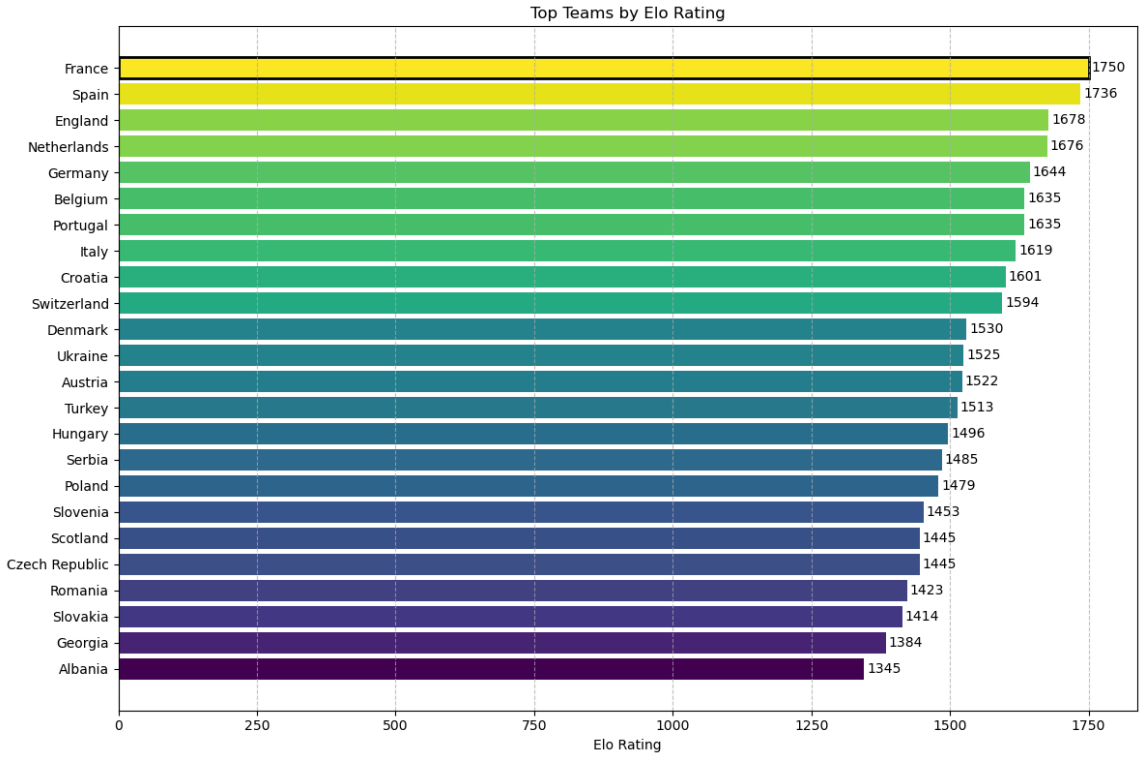
```python
def plot_top_elo(elos, n=10):
    """
    Plots the top n teams by Elo rating in a horizontal bar chart.

    Parameters:
    elos (pd.DataFrame): DataFrame containing 'Team' and 'Elo' columns
    n (int): Number of top teams to display.
    """
    # Sort elos DataFrame by Elo rating in descending order
    elos.sort_values(by='Elo', ascending=False, inplace=True)

    # Select only the top n teams
    top_teams = elos.head(n)

    # Create a color map
    cmap = plt.get_cmap('viridis')
    norm = mcolors.Normalize(vmin=min(top_teams['Elo']), vmax=max(top_
    colors = cmap(norm(top_teams['Elo']))

    # Plotting
    plt.figure(figsize=(12, 8))
    bars = plt.barh(top_teams['Team'], top_teams['Elo'], color=colors

    # Add text annotations
    for bar in bars:
        plt.text(
            bar.get_width() + 5,
            bar.get_y() + bar.get_height() / 2,
            f'{bar.get_width():.0f}',
            ha='left',
            va='center',
            fontsize=10
        )

    # Highlighting the top team
    bars[0].set_edgecolor('black')
    bars[0].set_linewidth(2)

    # Customizing the plot
    plt.xlabel('Elo Rating')
    plt.title('Top Teams by Elo Rating')
    plt.gca().invert_yaxis()  # Invert y-axis to display the highest l
    plt.grid(axis='x', linestyle='--', alpha=0.7)

    # Show plot
    plt.tight_layout()
    plt.show()

# Call the function
plot_top_elo(elos, 10)

```



We will focus only on the teams participating in the 2024 Euro. Therefore, the ranking would be as follows:

In [8]:

```
eurocup_teams =[
    "Albania", "Scotland", "Hungary", "Romania",
    "Germany", "Slovakia", "England", "Czech Republic",
    "Austria", "Slovenia", "Italy", "Serbia",
    "Belgium", "Spain", "Netherlands", "Switzerland",
    "Croatia", "France", "Poland", "Turkey",
    "Denmark", "Georgia", "Portugal", "Ukraine"
]

elos_ranking_euro = elos[elos.Team.isin(eurocup_teams)].copy()
plot_top_elo(elos_ranking_euro,24)
```



Based on this ranking, we can say that France is the favorite to win the Euro, while Albania is the weakest team in the tournament.

One of the problems with international matches is that we don't have the statistics of a league, making it difficult to determine which team has a better attack because it depends on the quality of the opponent. To solve this, and using the same idea as the Elo ranking system above, we can create a ranking for attacking power and defensive power.

In [9]:
```
elos_ranking_euro
```

Out[9]:

	date	Team	Elo	position
1	2024-07-05	France	1750.370	2
2	2024-07-05	Spain	1735.715	3
4	2024-06-30	England	1677.695	5
5	2024-07-02	Netherlands	1675.520	6
8	2024-07-05	Germany	1643.990	9
9	2024-07-01	Belgium	1634.765	10
10	2024-07-05	Portugal	1634.685	11
11	2024-06-29	Italy	1619.060	12
13	2024-06-24	Croatia	1600.850	14
14	2024-06-29	Switzerland	1594.235	15
22	2024-06-29	Denmark	1529.990	23
23	2024-06-26	Ukraine	1524.660	24
24	2024-07-02	Austria	1521.590	25
28	2024-07-02	Turkey	1513.125	29
32	2024-06-23	Hungary	1496.465	33
35	2024-06-25	Serbia	1485.295	36
37	2024-06-25	Poland	1479.350	38
51	2024-07-01	Slovenia	1452.545	52
55	2024-06-23	Scotland	1445.070	56
56	2024-06-26	Czech Republic	1444.845	57
62	2024-07-02	Romania	1423.000	63
64	2024-06-30	Slovakia	1413.625	65
72	2024-06-30	Georgia	1384.090	73
82	2024-06-24	Albania	1345.260	83

In [10]:
```
def calculate_attdef(Attack_l,Attack_v,Deffend_l,Deffend_v,localGoals,
    k=k_value(tournament)
    if neutral==False:
        c=1.27
    else:
        c=1
    ehg=Attack_l*Deffend_v*c
    eag=Attack_v*Deffend_l/c
    Attack_ln=Attack_l+(k/2000)*(localGoals-ehg)
    Attack_vn=Attack_v+(k/2000)*(awayGoals-eag)
    Deffend_ln=Deffend_l+(k/2000)*(awayGoals-eag)
    Deffend_vn=Deffend_v+(k/2000)*(localGoals-ehg)

    return Attack_ln,Attack_vn,Deffend_ln,Deffend_vn, ehg, eag
```

In [11]:

```
current_att={}
current_def={}
for idx,row in df.iterrows():

    local=row['home_team']
    away=row['away_team']
    local_goals=row['home_score']
    away_goals=row['away_score']
    tournament=row['tournament']
    neutral=row['neutral']

    if local not in current_att.keys():
        current_att[local]=1.3
    if away not in current_att.keys():
        current_att[away]=1.3
    if local not in current_def.keys():
        current_def[local]=1.3
    if away not in current_def.keys():
        current_def[away]=1.3

    att_l=current_att[local]
    att_v=current_att[away]
    def_l=current_def[local]
    def_v=current_def[away]

    att_ln,att_vn, def_ln, def_vn, ehg, eag=calculate_attdef(att_l,at

    current_att[local]=att_ln
    current_att[away]=att_vn
    current_def[local]=def_ln
    current_def[away]=def_vn

    df.loc[idx,'att_h_after']=att_ln
    df.loc[idx,'att_a_after']=att_vn
    df.loc[idx,'att_h_before']=att_l
    df.loc[idx,'att_a_before']=att_v
    df.loc[idx,'def_h_after']=def_ln
    df.loc[idx,'def_a_after']=def_vn
    df.loc[idx,'def_h_before']=def_l
    df.loc[idx,'def_a_before']=def_v
    df.loc[idx,'XGhome']=ehg
    df.loc[idx,'XGaway']=eag
```

In [12]:

```
attdef=df.copy()
attdef=pd.concat([df[['date','home_team','att_h_after','def_h_after']
attdef.sort_values(by='date', ascending=False,inplace=True)
attdef.drop_duplicates('Team',inplace=True)
attdef.sort_values(by='def', ascending=True,inplace=True)
attdef.reset_index(drop=True, inplace=True)
attdef['def_position']=attdef.index+1
attdef.sort_values(by='att', ascending=False,inplace=True)
attdef.reset_index(drop=True, inplace=True)
attdef['att_position']=attdef.index+1
```

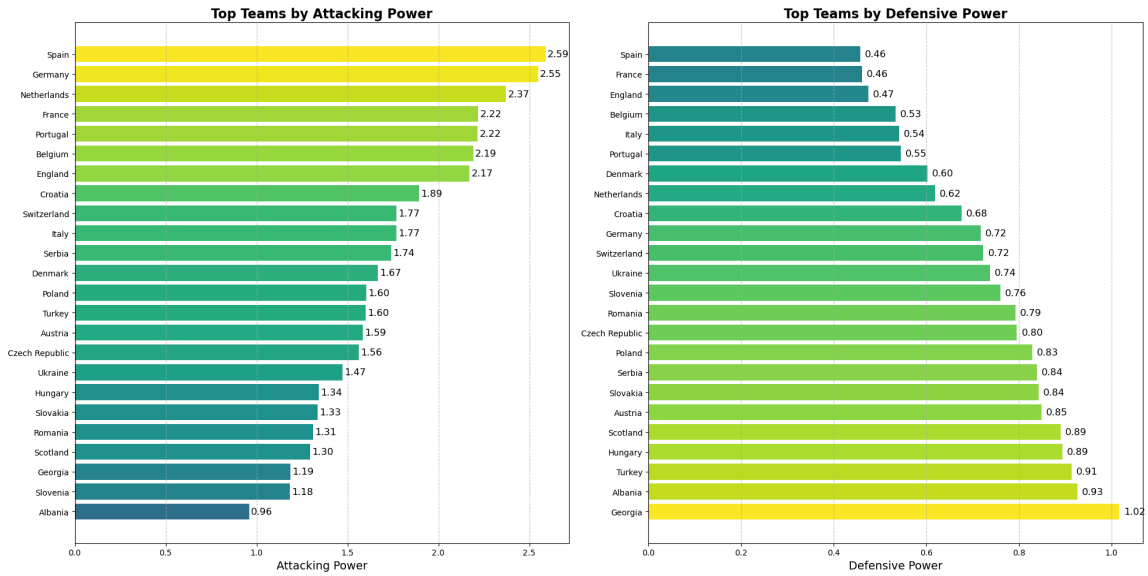
In [13]:

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

def plot_top_att_def(elos, n=10):
    """
    Plot the top teams by attacking and defensive power.

    Parameters:
    elos (pd.DataFrame): DataFrame containing team ratings.
    n (int): Number of top teams to display.
    """
    # Sort elos DataFrame by attacking and defensive ratings
    elos_sorted_att = elos.sort_values(by='att', ascending=False).hea
    elos_sorted_def = elos.sort_values(by='def', ascending=True).head

    # Create a color map for both plots
    colors_att = plt.cm.viridis(elos_sorted_att['att'] / max(elos_sor
    colors_def = plt.cm.viridis(elos_sorted_def['def'] / max(elos_sor

    # Create subplots
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 10))

    # Plot for attacking ratings
    bars_att = ax1.barh(elos_sorted_att['Team'], elos_sorted_att['att
    ax1.set_xlabel('Attacking Power', fontsize=14)
    ax1.set_title('Top Teams by Attacking Power', fontsize=16, weight
    ax1.invert_yaxis()  # Invert y-axis to display the highest rating
    ax1.grid(axis='x', linestyle='--', alpha=0.7)

    # Add text annotations for attacking ratings
    for bar in bars_att:
        ax1.text(
            bar.get_width() + 0.01,
            bar.get_y() + bar.get_height() / 2,
            f'{bar.get_width():.2f}',
            ha='left',
            va='center',
            fontsize=12,
            color='black'
        )

    # Plot for defensive ratings
    bars_def = ax2.barh(elos_sorted_def['Team'], elos_sorted_def['def
    ax2.set_xlabel('Defensive Power', fontsize=14)
    ax2.set_title('Top Teams by Defensive Power', fontsize=16, weight
    ax2.invert_yaxis()  # Invert y-axis to display the highest rating
    ax2.grid(axis='x', linestyle='--', alpha=0.7)

    # Add text annotations for defensive ratings
    for bar in bars_def:
        ax2.text(
            bar.get_width() + 0.01,
            bar.get_y() + bar.get_height() / 2,
            f'{bar.get_width():.2f}',
            ha='left',
            va='center',
            fontsize=12,
            color='black'
        )

    # Adjust layout
```

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

```

In [14]:

```
attdef_ranking_europe = attdef[attdef.Team.isin(eurocup_teams)]
plot_top_att_def(attdef_ranking_europe,24)
```



In [15]:
```
1 attdef_ranking_europe
```

Out[15]:

	date	Team	att	def	def_position	att_position
0	2024-07-05	Spain	2.589579	0.457570	3	1
1	2024-07-05	Germany	2.548609	0.717336	36	2
3	2024-07-02	Netherlands	2.371421	0.618433	18	4
6	2024-07-05	France	2.219333	0.461540	4	7
7	2024-07-05	Portugal	2.216051	0.545238	11	8
8	2024-07-01	Belgium	2.191970	0.533718	9	9
9	2024-06-30	England	2.171419	0.474628	6	10
12	2024-06-24	Croatia	1.892096	0.676163	26	13
16	2024-06-29	Switzerland	1.769522	0.723214	38	17
17	2024-06-29	Italy	1.769313	0.540849	10	18
18	2024-06-25	Serbia	1.740022	0.839128	63	19
25	2024-06-29	Denmark	1.665533	0.602726	17	26
33	2024-06-25	Poland	1.604044	0.828754	59	34
34	2024-07-02	Turkey	1.599998	0.913647	77	35
35	2024-07-02	Austria	1.585274	0.848737	67	36
39	2024-06-26	Czech Republic	1.561265	0.795722	51	40
48	2024-06-26	Ukraine	1.474243	0.737801	41	49
79	2024-06-23	Hungary	1.342773	0.893565	75	80
83	2024-06-30	Slovakia	1.334461	0.842814	64	84
111	2024-07-02	Romania	1.310627	0.792474	49	112
133	2024-06-23	Scotland	1.295363	0.889733	74	134
208	2024-06-30	Georgia	1.187143	1.016745	92	209
210	2024-07-01	Slovenia	1.182072	0.760566	45	211
272	2024-06-24	Albania	0.958692	0.926319	79	273

Based on this ranking, we can conclude that the teams with the best attack are Germany and Spain. The teams with the best defense are England, Spain, and Italy.

4 Goals Expected and Poisson Distribution

The best aspect of this ranking is that we can use it to calculate the expected goals in a match. The expected goals for a team can be calculated as:

$$ExpectedGoals = AttackingPowerTeam \times DefensivePowerOpponent$$

Additionally, in non-neutral matches, we have to set a parameter for the advantage of the home team. This parameter adjusts the expected goals to account for the home field advantage.

Let see if the results adjust with Expected goals we predicted, for doing that we consider the results of European tournaments of the last 14 years.

In [16]:
```
#Creating the reference dataset
df2 = df[(df.date>"2010-09-01") & (df.tournament.isin(["UEFA Euro","U
# Define the bins
bins = np.array([1.5,2.25, 2.5, 2.75, 3,10])  # adjust this to fit yo
df2['total_score'] = df2['away_score'] + df2['home_score']
df2['total_xG']=df2['XGhome']+df2['XGaway']
df2['total_xG_bin'] = pd.cut(df2['total_xG'], bins)
# Group by the bin and calculate the mean of the 'Goals' column
binned_means = df2.groupby('total_xG_bin')['total_score'].agg(["mean"
print(binned_means)
```

```
                  mean   count
total_xG_bin
(1.5, 2.25]   2.249322     369
(2.25, 2.5]   2.366234     385
(2.5, 2.75]   2.515406     357
(2.75, 3.0]   2.923767     223
(3.0, 10.0]   3.505988     334
```

In [17]:
```
df2
```

Out[17]:

	date	home_team	away_team	home_score	away_score	tournament	
33959	2010-09-02	Israel	Malta	3.0	1.0	UEFA Euro qualification	Rama
33961	2010-09-03	Andorra	Russia	0.0	2.0	UEFA Euro qualification	Sant J
33962	2010-09-03	Armenia	Republic of Ireland	0.0	1.0	UEFA Euro qualification	Ye
33963	2010-09-03	Belgium	Germany	0.0	1.0	UEFA Euro qualification	Br
33965	2010-09-03	England	Bulgaria	4.0	0.0	UEFA Euro qualification	L
33966	2010-09-03	Estonia	Italy	1.0	2.0	UEFA Euro qualification	
33967	2010-09-03	Faroe Islands	Serbia	0.0	3.0	UEFA Euro qualification	Tór

In [18]:
```
df2.columns

```

Out[18]:
```
Index(['date', 'home_team', 'away_team', 'home_score', 'away_score',
       'tournament', 'city', 'country', 'neutral', 'Elo_h_after',
       'Elo_a_after', 'Elo_h_before', 'Elo_a_before', 'probH', 'probA',
       'att_h_after', 'att_a_after', 'att_h_before', 'att_a_before',
       'def_h_after', 'def_a_after', 'def_h_before', 'def_a_before', 'XGho
me',
       'XGaway', 'total_score', 'total_xG', 'total_xG_bin'],
      dtype='object')
```

In [19]:
```
df2['year']=df2['date'].apply(lambda x: int(x[:4]))
df2
```

Out[19]:

	date	home_team	away_team	home_score	away_score	tournament	
33959	2010-09-02	Israel	Malta	3.0	1.0	UEFA Euro qualification	Rama
33961	2010-09-03	Andorra	Russia	0.0	2.0	UEFA Euro qualification	Sant J
33962	2010-09-03	Armenia	Republic of Ireland	0.0	1.0	UEFA Euro qualification	Ye
33963	2010-09-03	Belgium	Germany	0.0	1.0	UEFA Euro qualification	Br
33965	2010-09-03	England	Bulgaria	4.0	0.0	UEFA Euro qualification	L
33966	2010-09-03	Estonia	Italy	1.0	2.0	UEFA Euro qualification	
33967	2010-09-03	Faroe Islands	Serbia	0.0	3.0	UEFA Euro qualification	Tór

In [20]:
```
1 df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1668 entries, 33959 to 47368
Data columns (total 29 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   date          1668 non-null   object
 1   home_team     1668 non-null   object
 2   away_team     1668 non-null   object
 3   home_score    1668 non-null   float64
 4   away_score    1668 non-null   float64
 5   tournament    1668 non-null   object
 6   city          1668 non-null   object
 7   country       1668 non-null   object
 8   neutral       1668 non-null   bool
 9   Elo_h_after   1668 non-null   float64
 10  Elo_a_after   1668 non-null   float64
 11  Elo_h_before  1668 non-null   float64
 12  Elo_a_before  1668 non-null   float64
 13  probH         1668 non-null   float64
 14  probA         1668 non-null   float64
 15  att_h_after   1668 non-null   float64
 16  att_a_after   1668 non-null   float64
 17  att_h_before  1668 non-null   float64
 18  att_a_before  1668 non-null   float64
 19  def_h_after   1668 non-null   float64
 20  def_a_after   1668 non-null   float64
 21  def_h_before  1668 non-null   float64
 22  def_a_before  1668 non-null   float64
 23  XGhome        1668 non-null   float64
 24  XGaway        1668 non-null   float64
 25  total_score   1668 non-null   float64
 26  total_xG      1668 non-null   float64
 27  total_xG_bin  1668 non-null   category
 28  year          1668 non-null   int64
dtypes: bool(1), category(1), float64(20), int64(1), object(6)
memory usage: 368.4+ KB
```

In [21]:
```
df2['home_diff_elo']=df2['Elo_h_after']-df2['Elo_a_after']
df2['away_diff_elo']=df2['Elo_a_after']-df2['Elo_h_after']
df2
```

Out[21]:

	date	home_team	away_team	home_score	away_score	tournament	
33959	2010-09-02	Israel	Malta	3.0	1.0	UEFA Euro qualification	Rama
33961	2010-09-03	Andorra	Russia	0.0	2.0	UEFA Euro qualification	Sant J
33962	2010-09-03	Armenia	Republic of Ireland	0.0	1.0	UEFA Euro qualification	Ye
33963	2010-09-03	Belgium	Germany	0.0	1.0	UEFA Euro qualification	Br
33965	2010-09-03	England	Bulgaria	4.0	0.0	UEFA Euro qualification	L
33966	2010-09-03	Estonia	Italy	1.0	2.0	UEFA Euro qualification	
33967	2010-09-03	Faroe Islands	Serbia	0.0	3.0	UEFA Euro qualification	Tór

In [22]:

```
aP_teams = {}
for idx, match in df2.iterrows():
    home_team = match["home_team"]
    away_team = match["away_team"]
    home_score = match["home_score"]
    away_score = match["away_score"]

    match_year = int(match['date'][:4])
    if match_year == 2024:
        match_year = 2023

#     if home_team == 'Northern Ireland':
#         home_team = 'Republic of Ireland'

#     if away_team == 'Northern Ireland':
#         away_team = 'Republic of Ireland'

    home_elo_series = df2[(df2['year'] == match_year) & (df2['home_tea
    away_elo_series = df2[(df2['year'] == match_year) & (df2['away_tea

    try:
        home_elo = home_elo_series.tolist()[0]
        away_elo = away_elo_series.tolist()[0]
    except:
        continue

    home_diff = home_elo - away_elo
    away_diff = away_elo - home_elo

    if home_team in eurocup_teams:
        if home_team not in aP_teams:
            aP_teams[home_team] = [home_score, 1, [home_score], away_s
        else:
            aP_teams[home_team][0] += home_score
            aP_teams[home_team][1] += 1
            aP_teams[home_team][2].append(home_score)
            aP_teams[home_team][3] += away_score
            aP_teams[home_team][4].append(away_score)
            aP_teams[home_team][5].append(home_diff)

    if away_team in eurocup_teams:
        if away_team not in aP_teams:
            aP_teams[away_team] = [away_score, 1, [away_score], home_s
        else:
            aP_teams[away_team][0] += away_score
            aP_teams[away_team][1] += 1
            aP_teams[away_team][2].append(away_score)
            aP_teams[away_team][3] += home_score
            aP_teams[away_team][4].append(home_score)
            aP_teams[away_team][5].append(away_diff)
```

In [23]:

```
# Summary of aP and Code Function
# aP: This dictionary ends up containing the average number of goals s
# Print Output: For each team, it prints the team name, the average go
import statistics
aP = {}
for i in aP_teams.keys():
    aP[i] = aP_teams[i][0] / aP_teams[i][1]

    print(f"{i},{aP[i]},{statistics.stdev(aP_teams[i][2])}")
```

```
Belgium,2.3714285714285714,1.7870016063789165
Germany,2.288135593220339,1.848119710423572
England,2.0985915492957745,1.8059815659355425
Italy,1.7625,1.4861704248320815
Serbia,1.5789473684210527,1.223977246827685
France,1.8333333333333333,1.9890727127006333
Georgia,1.2580645161290323,1.213793455719087
Turkey,1.4696969696969697,1.3268783141667742
Croatia,1.6428571428571428,1.3623904214488174
Spain,2.3947368421052633,1.7669480080517068
Scotland,1.576271186440678,1.499464155215507
Portugal,2.066666666666667,1.833128571693857
Romania,1.303030303030303,1.2021736335577509
Albania,1.1071428571428572,1.1548879745526117
Netherlands,2.208955223880597,1.8630157486674404
Slovakia,1.1940298507462686,1.1446688811967767
Slovenia,1.359375,1.337697470958783
Hungary,1.492537313432836,1.3070106341683057
Austria,1.6153846153846154,1.3996908999434894
Czech Republic,1.373134328358209,1.056521176708736
Denmark,1.671875,1.4148272371958037
Switzerland,1.878787878787879,1.6503160677709963
Poland,1.5833333333333333,1.5762180296000163
Ukraine,1.3166666666666667,1.2418093233182887
```

In [24]:
```
# Structure of aP_teams
# Key: Team name.
# Value: A list containing:
# Total goals scored by the team ([0]).
# Total number of matches ([1]).
# A list of goals scored in each match ([2]).
# Total goals conceded ([3]).
# A list of goals conceded in each match ([4]).
# A list of ELO rating differences for each match ([5]).


import pprint # Will be used later for some nicer prints
pprint.pprint(aP_teams["England"])
```

```
[149.0,
 71,
 [4.0,
  3.0,
  0.0,
  2.0,
  2.0,
  3.0,
  1.0,
  2.0,
  1.0,
  3.0,
  1.0,
  0.0,
  2.0,
  5.0,
  1.0,
  3.0,
  4.0,
```

In [25]:
```
mega = [[], []]
for idx, i in enumerate(list(aP_teams.values())):
    mega[1] += (np.array(i[2]) - np.array(len(i[2])*[aP[list(aP_teams
    mega[0] += i[5]
```

In [26]:
```
from sklearn.linear_model import LinearRegression

X, y = np.array(mega[0]).reshape(-1, 1), np.array(mega[1]).reshape(-1


reg = LinearRegression()
reg.fit(X, y)


print(reg.score(X, y))

print(reg.coef_, reg.intercept_)
```

```
0.1575846986689724
[[0.00279906]] [-0.28575836]
```

In [27]:
```
max(X)
```

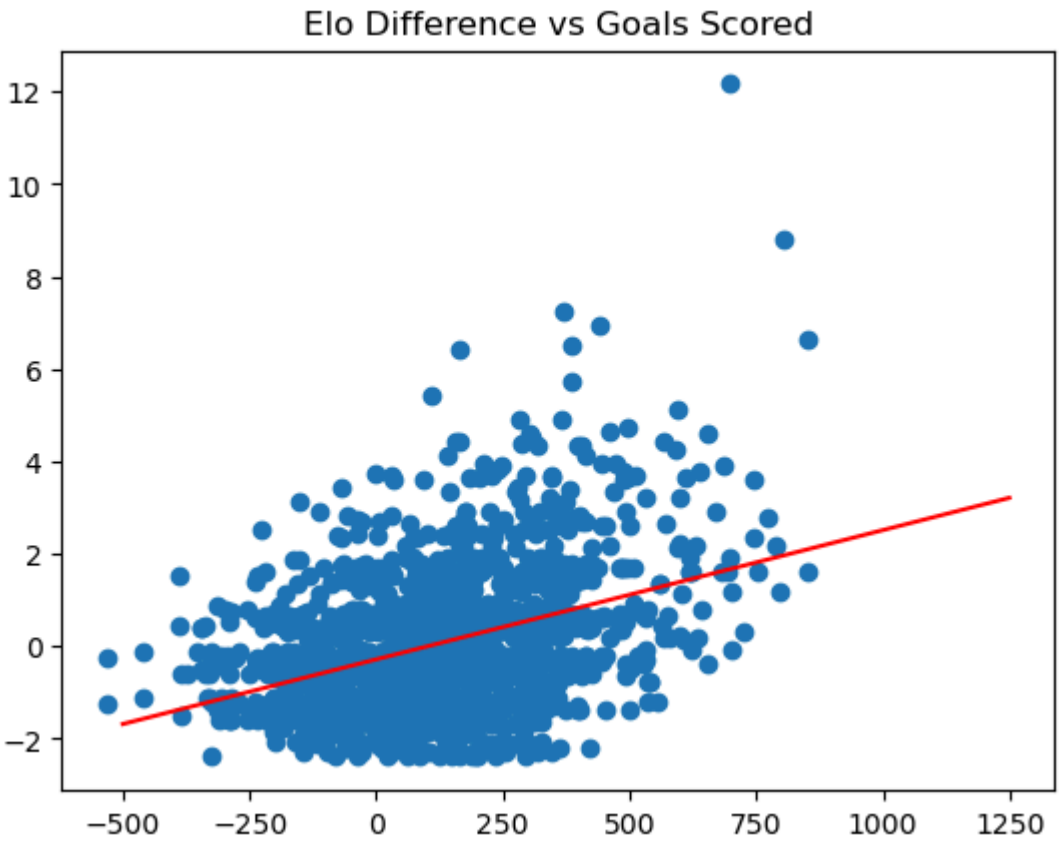
Out[27]:
```
array([852.45])
```

In [28]:
```
1 min(X)
```

Out[28]:
```
array([-531.88])
```

In [29]:
```
1 plt.title('Elo Difference vs Goals Scored')
2 plt.scatter(mega[0], mega[1])
3 x = np.arange(-500, 1250)
4 y = x*reg.coef_[0] + reg.intercept_[0]
5 plt.plot(x, y, color='red')
```

Out[29]:
```
[<matplotlib.lines.Line2D at 0x2711fbe0610>]
```



We can see that the expected goals we created adjust very well to the real number of goals in the matches. So we can use the poisson distribution to predict the results of the matches

The Poisson distribution is defined by the probability mass function:

$P(X=k)=\lambda ke-\lambda k!$

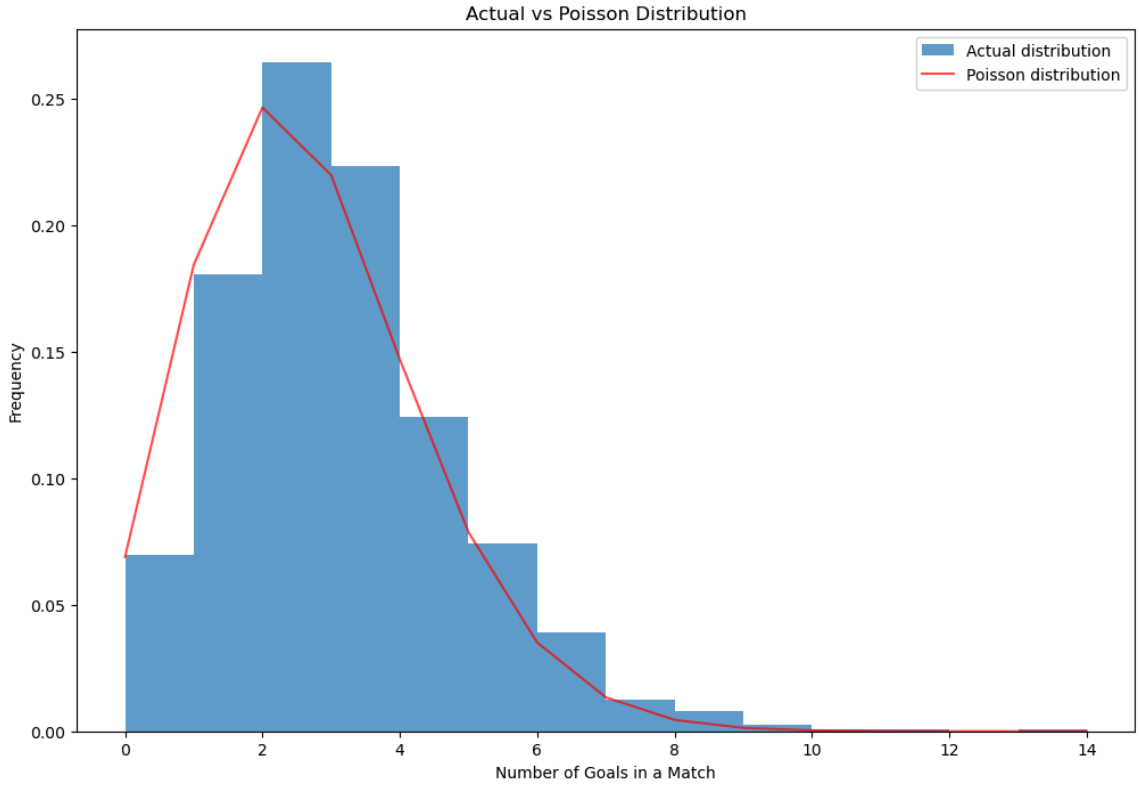
where:

$P(X=k)$ is the probability of k goals being scored, λ is the average number of goals scored in a match.

In our context, λ represents the average number of goals a team is expected to score.

In [30]:

```
data = df2['total_score']

# Calculate the mean of the data as it will be the lambda (rate parame
mu = data.mean()

# Create a range of numbers from 0 to the maximum number of goals in d
k = np.arange(0, data.max()+1)

# Create a Poisson distribution with the mean obtained
poisson_pmf = poisson.pmf(k, mu)

plt.figure(figsize=(12,8))

# Plot the actual distribution of goals
plt.hist(data, bins=k, density=True, alpha=0.7, label='Actual distribu

# Plot the Poisson distribution
plt.plot(k, poisson_pmf, 'r-', alpha=0.7, label='Poisson distribution

plt.title('Actual vs Poisson Distribution')
plt.xlabel('Number of Goals in a Match')
plt.ylabel('Frequency')
plt.legend()

plt.show()
```



With the utilization of the given distribution, we can use the probabilities to predict the matches.

In [31]:

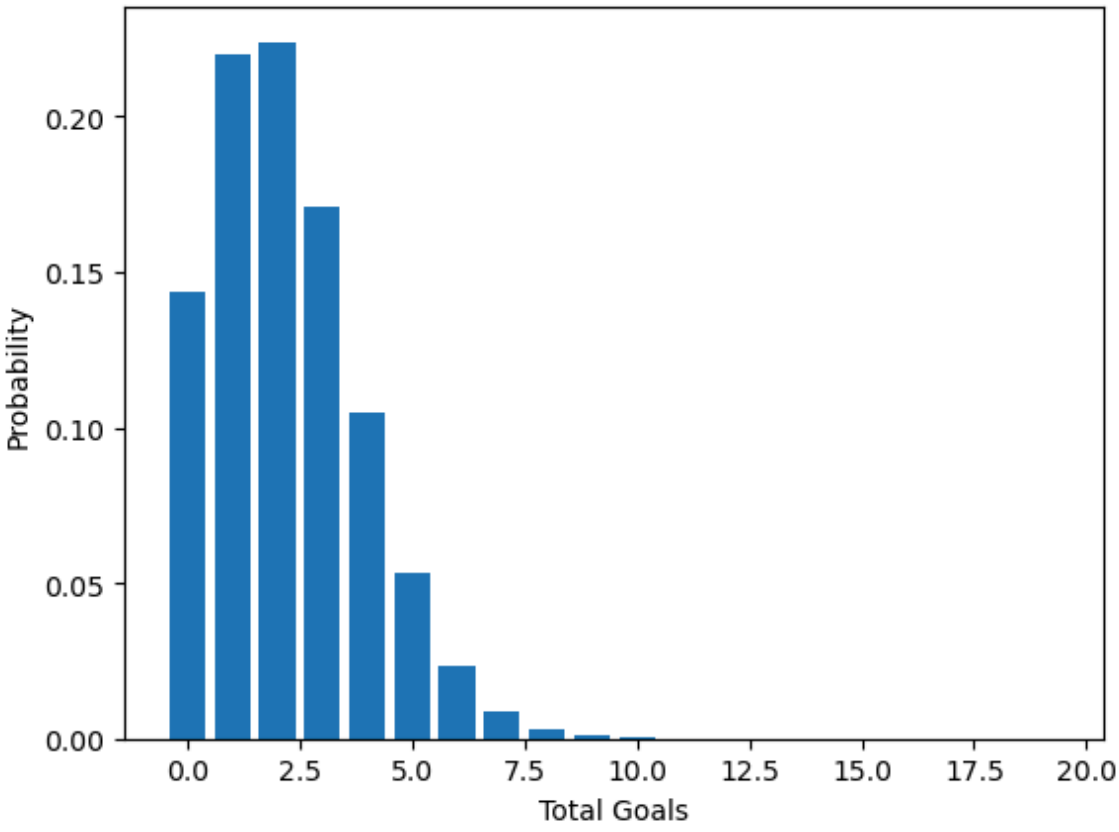
```
def calculate_match_probabilities(team_a_lambda, team_b_lambda, max_g
    # Create a matrix of zeros
    matrix = np.zeros((max_goals + 1, max_goals + 1))

    # Populate the matrix with probabilities
    for i in range(max_goals + 1):
        for j in range(max_goals + 1):
            matrix[i, j] = poisson.pmf(i, team_a_lambda) * poisson.pm

    # Calculate the total goals probabilities
    total_goals_prob = [np.sum(np.diag(matrix[::-1], k)) for k in rang

    # Calculate the probability for the specific result
    draw_prob = np.sum(np.diag(matrix))
    away_prob = np.sum(np.triu(matrix, 1))
    local_prob = np.sum(np.tril(matrix, -1))

    df_match_goals=pd.DataFrame(matrix)
    highest_value = df_match_goals.max().max()
    max_position = df_match_goals.stack().idxmax()
    h_goals=max_position[0]
    a_goals=max_position[1]


    return total_goals_prob, draw_prob, away_prob, local_prob,h_goals

```

For instance, let's consider the inaugural match of the 2024 Euro between Germany and Scotland. Germany boasts an offensive power of 2.58, while Scotland 1.31.

In [32]:

```
# Example usage
# Albania   1342.140
# France    1752.570
# Georgia   1.177952    0.987056
# Spain 2.548556    0.458797

team_a_lambda = 2.548556 * 0.987056
team_b_lambda = 1.177952 * 0.458797
total_goals_prob, draw_prob, away_prob, local_prob,h_goals,a_goals = (

# Print the results
'''print("Total goals probabilities:")
for goals, prob in enumerate(total_goals_prob):
    print(f'Probability of total {goals} goals: {prob}')'''


print(f"Probability for France: {round(local_prob,2)*100} %")
print(f"Probability for Albania: {round(away_prob,2)*100} %")
print(f"Probability for draw: {round(draw_prob,2)*100} %")

import matplotlib.pyplot as plt
# Plot the distribution of possible total goals
plt.bar(range(len(total_goals_prob)), total_goals_prob)
plt.xlabel('Total Goals')
plt.ylabel('Probability')
plt.show()
```

```
Probability for France: 80.0 %
Probability for Albania: 6.0 %
Probability for draw: 14.000000000000002 %
```



In [33]:
```
h_goals
```

Out[33]: 2

In [34]:
```
a_goals
```

Out[34]: 0

In [35]:
```
max(total_goals_prob)
```

Out[35]: 0.22392614125034904

5 Adjusted Goals Expected with ELO form and Poisson Distribution

In [36]:
```
# phase1=pd.read_excel('Prediction_template.xlsx',sheet_name='Match pr
# phase1=phase1[['Home Team','Away Team','Home_goal','Away_Goal']]
# phase1=phase1.iloc[:36,:]
# phase1
```

In [37]:
```
# home_goals = phase1.groupby('Home Team').agg(goals_scored=('Home_goa
# home_goals.rename(columns={'Home Team': 'Team'}, inplace=True)
# away_goals = phase1.groupby('Away Team').agg(goals_scored=('Away_Goa
# away_goals.rename(columns={'Away Team': 'Team'}, inplace=True)
# teams_pred = pd.concat([home_goals, away_goals]).groupby('Team').sum
# teams_pred
```

In [38]:
```
# teams_pred.columns
```

In [ ]:
```

```

5.1 Adding euro form

In [39]:
```
1 import pandas as pd
2 euros24_group=pd.read_excel('euroresults.xlsx')
3 euros24_group
```

Out[39]:

	Team	goals_scored	goals_against	points
0	Spain	5	0	9
1	Germany	8	2	7
2	Austria	6	4	6
3	Turkey	5	5	6
4	Portugal	5	3	6
5	England	2	1	5
6	France	2	1	5
7	Switzerland	5	3	5
8	Italy	3	3	4
9	Romania	4	3	4
10	Belgium	2	1	4
11	Netherlands	4	4	4
12	Slovakia	3	3	4
13	Georgia	4	4	4
14	Ukraine	2	4	4
15	Hungary	2	5	3
16	Slovenia	2	2	3
17	Denmark	2	2	3
18	Serbia	1	2	2
19	Croatia	3	6	2
20	Poland	3	6	1
21	Scotland	2	7	1
22	Albania	3	5	1
23	Czech Republic	3	5	1

In [40]:
```
group16=[
    'Austria',
'England',
'Germany',
'Portugal',
'Romania',
'Spain',
'Belgium',
'Denmark',
'France',
'Italy',
'Switzerland',
'Turkey',
    'Georgia',
'Netherlands',
'Slovakia',
'Slovenia']

group16_eurosteams=euros24_group[euros24_group['Team'].isin(group16)]
group16_eurosteams_sorted = group16_eurosteams.sort_values(by=['points
group16_eurosteams_sorted['euro_form'] = (group16_eurosteams_sorted['
group16_eurosteams_sorted

```

Out[40]:

	Team	goals_scored	goals_against	points	euro_form
0	Spain	5	0	9	2.000000
1	Germany	8	2	7	1.555556
2	Austria	6	4	6	1.333333
3	Turkey	5	5	6	1.333333
4	Portugal	5	3	6	1.333333
5	England	2	1	5	1.111111
6	France	2	1	5	1.111111
7	Switzerland	5	3	5	1.111111
8	Italy	3	3	4	0.888889
9	Romania	4	3	4	0.888889
10	Belgium	2	1	4	0.888889
11	Netherlands	4	4	4	0.888889
12	Slovakia	3	3	4	0.888889
13	Georgia	4	4	4	0.888889
16	Slovenia	2	2	3	0.666667
17	Denmark	2	2	3	0.666667

In [41]:
```
group16_eurosteams_sorted['points'].median()
```

Out[41]: 4.5

In [ ]:
```
1
```

In [42]:
```
# group16_eurosteams_sorted_predict = pd.merge(group16_eurosteams_sor
# group16_eurosteams_sorted_predict['diff_goals_scored']=group16_euros
# group16_eurosteams_sorted_predict['diff_goals_agaisnt']=group16_eur
# group16_eurosteams_sorted_predict['avg_goals_scored']=group16_eurost
# group16_eurosteams_sorted_predict['avg_goals_agaisnt']=group16_euros
# group16_eurosteams_sorted_predict['euro_form']=group16_eurosteams_s
# group16_eurosteams_sorted_predict['form_goals_scored']=group16_euros
# group16_eurosteams_sorted_predict['form_goals_agaisnt']=group16_eur
# group16_eurosteams_sorted_predict


```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [43]:
```
def calculate_match_probabilities_elo(team_a_lambda, team_b_lambda,hor
    # Create a matrix of zeros
    matrix = np.zeros((max_goals + 1, max_goals + 1))

    # Populate the matrix with probabilities
    for i in range(max_goals + 1):
        for j in range(max_goals + 1):
            matrix[i, j] = poisson.pmf(i, team_a_lambda) * poisson.pm

    # Calculate the total goals probabilities
    total_goals_prob = [np.sum(np.diag(matrix[::-1], k)) for k in rang

    # Calculate the probability for the specific result
    draw_prob = np.sum(np.diag(matrix))
    away_prob = np.sum(np.triu(matrix, 1))
    local_prob = np.sum(np.tril(matrix, -1))

    df_match_goals=pd.DataFrame(matrix)
    highest_value = df_match_goals.max().max()
    max_position = df_match_goals.stack().idxmax()
    #array results so need to convert to int
    h_goals_adjust=int(reg.predict(np.array(home_diff).reshape(-1, 1)
    a_goals_adjust=int(reg.predict(np.array(away_diff).reshape(-1, 1)
    h_goals=max_position[0]
    a_goals=max_position[1]

    fin_h_goals=h_goals+h_goals_adjust
    fin_a_goals=a_goals+a_goals_adjust
    if (fin_a_goals==-1):
        adj_sum=abs(h_goals_adjust)+abs(a_goals_adjust)
        fin_h_goals+=adj_sum
        fin_a_goals+=1
    elif (fin_h_goals==-1):
        adj_sum=abs(h_goals_adjust)+abs(a_goals_adjust)
        fin_a_goals+=adj_sum
        fin_h_goals+=1


    return total_goals_prob, draw_prob, away_prob, local_prob,h_goals
```

In [44]:
```
float(elos_ranking_euro[elos_ranking_euro['Team']=='Germany']['Elo']
```

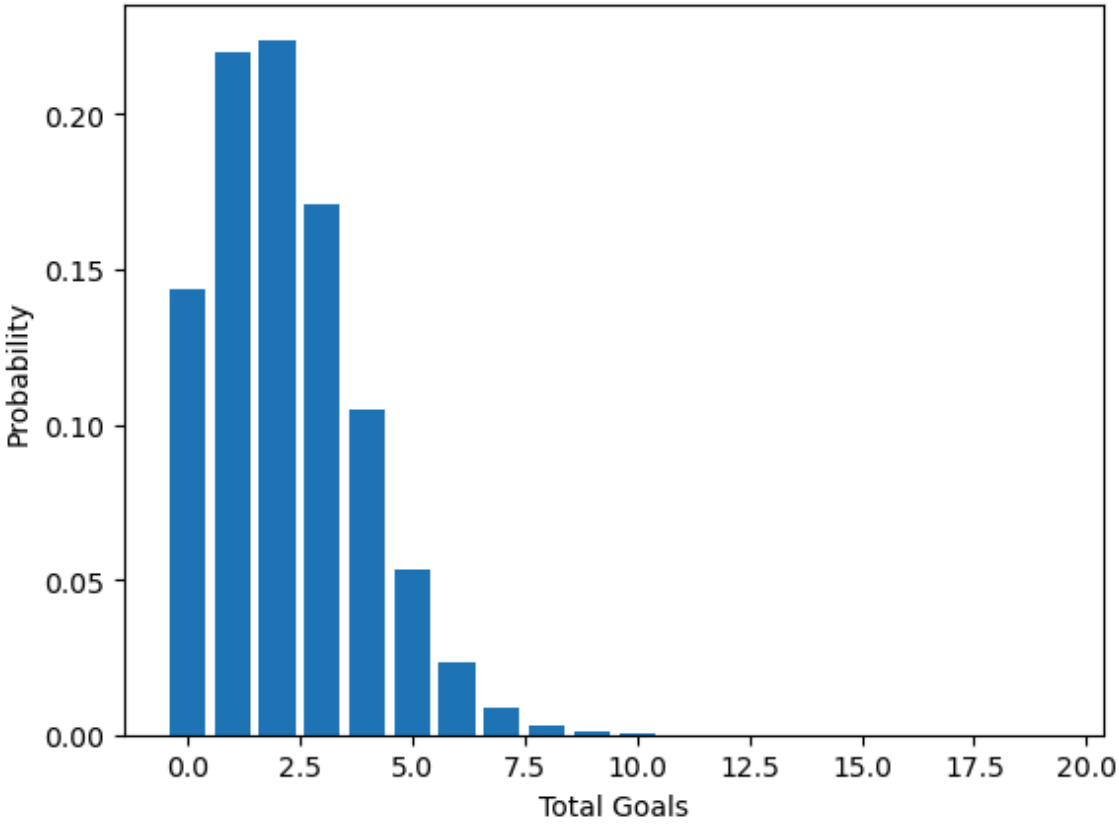
Out[44]: 1643.9900000000023

In [45]:
```

# home_diff = float(elos_ranking_euro[elos_ranking_euro['Team']=='Ger
# away_diff = float(elos_ranking_euro[elos_ranking_euro['Team']=='Sco
# print(f'home_diff : {home_diff}')
# print(f'away_diff : {away_diff}')
```

In [46]:

```
# Example usage
# Albania   1342.140
# France    1752.570
# Georgia   1.177952    0.987056
# Spain 2.548556    0.458797

team_a_lambda = 2.548556 * 0.987056
team_b_lambda = 1.177952 * 0.458797
home_diff = float(elos_ranking_euro[elos_ranking_euro['Team']=='Spain
away_diff = float(elos_ranking_euro[elos_ranking_euro['Team']=='Georg
total_goals_prob, draw_prob, away_prob, local_prob,h_goals,a_goals,fi




# Print the results
'''print("Total goals probabilities:")
for goals, prob in enumerate(total_goals_prob):
    print(f'Probability of total {goals} goals: {prob}')'''


print(f"Probability for France: {round(local_prob,2)*100} %")
print(f"Probability for Albania: {round(away_prob,2)*100} %")
print(f"Probability for draw: {round(draw_prob,2)*100} %")

import matplotlib.pyplot as plt
# Plot the distribution of possible total goals
plt.bar(range(len(total_goals_prob)), total_goals_prob)
plt.xlabel('Total Goals')
plt.ylabel('Probability')
plt.show()
```

```
Probability for France: 80.0 %
Probability for Albania: 6.0 %
Probability for draw: 14.000000000000002 %
```



In [47]:
```
1 h_goals,a_goals
```

Out[47]: (2, 0)

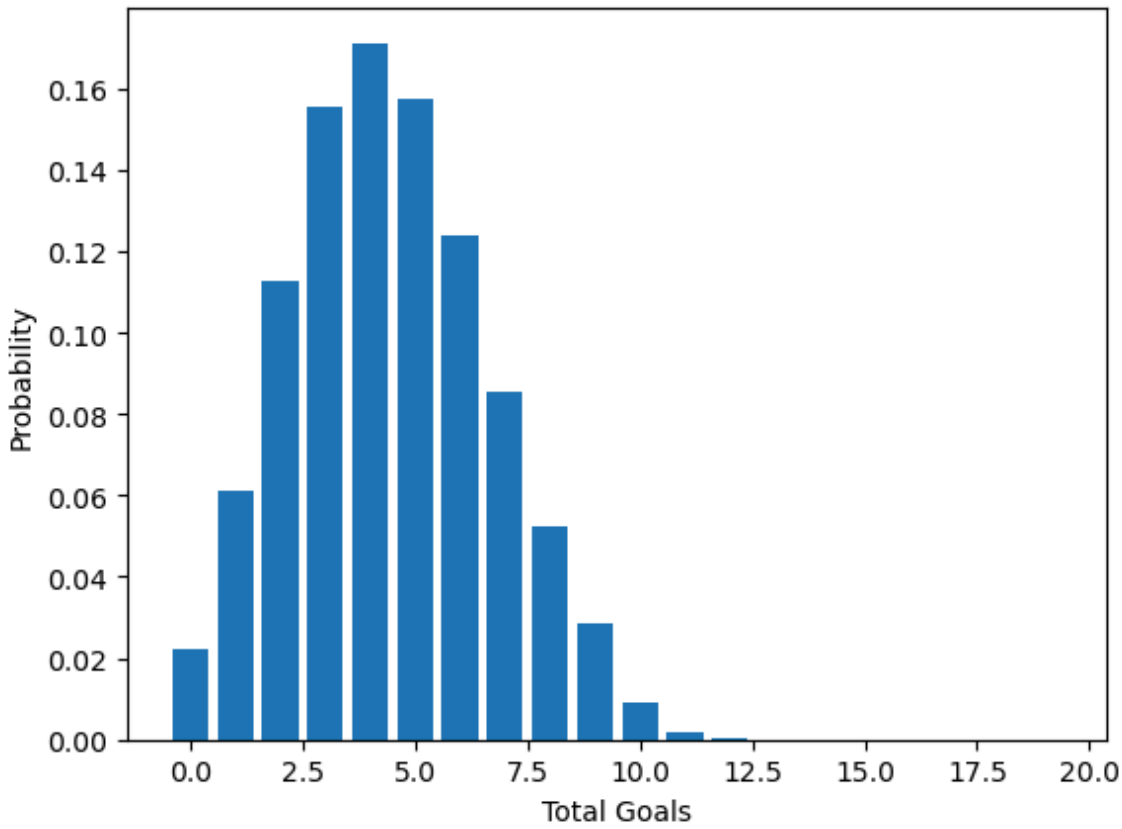
In [48]:
```
1 fin_h_goals,fin_a_goals
```

Out[48]: (3, 0)

In [49]:

```
# Example usage
# Georgia   1389.450
# Spain 1713.675
# Georgia   1.177952    0.987056
# Spain 2.548556    0.458797

team_a_lambda = 2.548556 * 0.987056*2.000000
team_b_lambda = 1.177952 * 0.458797*0.888889
home_diff = float(elos_ranking_euro[elos_ranking_euro['Team']=='Spain
away_diff = float(elos_ranking_euro[elos_ranking_euro['Team']=='Georg
total_goals_prob, draw_prob, away_prob, local_prob,h_goals,a_goals,fi


# Print the results
'''print("Total goals probabilities:")
for goals, prob in enumerate(total_goals_prob):
    print(f'Probability of total {goals} goals: {prob}')'''


print(f"Probability for France: {round(local_prob,2)*100} %")
print(f"Probability for Albania: {round(away_prob,2)*100} %")
print(f"Probability for draw: {round(draw_prob,2)*100} %")

import matplotlib.pyplot as plt
# Plot the distribution of possible total goals
plt.bar(range(len(total_goals_prob)), total_goals_prob)
plt.xlabel('Total Goals')
plt.ylabel('Probability')
plt.show()
```

```
Probability for France: 96.0 %
Probability for Albania: 1.0 %
Probability for draw: 2.0 %
```



In [50]:
```
1 h_goals,a_goals
```

Out[50]: (5, 0)

In [51]:
```
1 fin_h_goals,fin_a_goals
```

Out[51]: (6, 0)

In [ ]:
```
1
```

In [ ]:
```
1
```

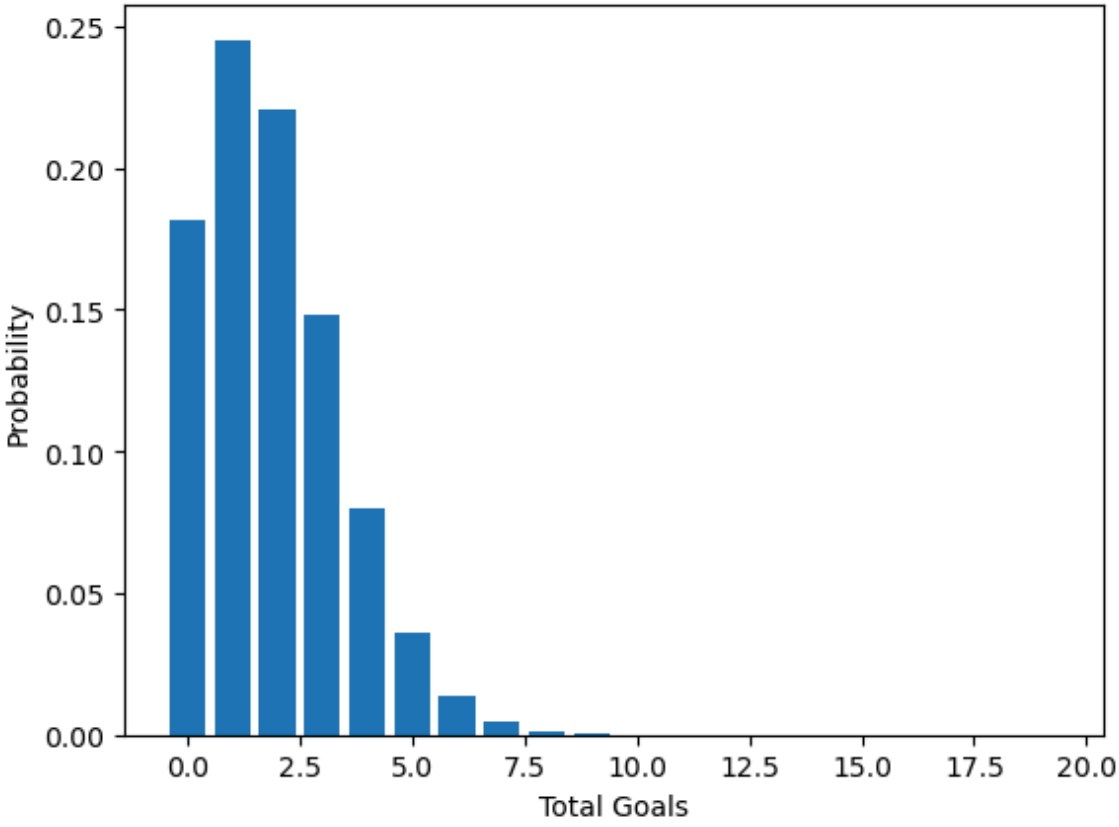
In [52]:
```
1 for x in range(-532,853):
2     h_goals_adjust=int(reg.predict(np.array(x).reshape(-1, 1)))
3     print(h_goals_adjust,x)
```

```
-1 -532
-1 -531
-1 -530
-1 -529
-1 -528
-1 -527
-1 -526
-1 -525
-1 -524
-1 -523
-1 -522
-1 -521
-1 -520
-1 -519
-1 -518
-1 -517
-1 -516
-1 -515
-1 -514
```

In [53]:

```
# Example usage
# Albania   1342.140
# France    1752.570
# Albania   0.929605    0.945796
# France    2.316062    0.543255


team_b_lambda = 2.316062 * 0.945796
team_a_lambda = 0.543255 * 0.929605
away_diff = float(elos_ranking_euro[elos_ranking_euro['Team']=='France
home_diff = float(elos_ranking_euro[elos_ranking_euro['Team']=='Alban
total_goals_prob, draw_prob, away_prob, local_prob,h_goals,a_goals,fi



# Print the results
'''print("Total goals probabilities:")
for goals, prob in enumerate(total_goals_prob):
    print(f'Probability of total {goals} goals: {prob}')'''


print(f"Probability for France: {round(local_prob,2)*100} %")
print(f"Probability for Albania: {round(away_prob,2)*100} %")
print(f"Probability for draw: {round(draw_prob,2)*100} %")

import matplotlib.pyplot as plt
# Plot the distribution of possible total goals
plt.bar(range(len(total_goals_prob)), total_goals_prob)
plt.xlabel('Total Goals')
plt.ylabel('Probability')
plt.show()
```

```
Probability for France: 7.000000000000001 %
Probability for Albania: 76.0 %
Probability for draw: 17.0 %
```



In [54]:
```
1 h_goals,a_goals
```

Out[54]: (0, 2)

In [55]:
```
1 fin_h_goals,fin_a_goals
```

Out[55]: (0, 3)

In [56]:
```
1 team_a_lambda = 2.316062 * 0.945796
2 team_b_lambda = 0.543255 * 0.929605
3 total_goals_prob, draw_prob, away_prob, local_prob,h_goals,a_goals,fi
4
```

In [57]:
```
1 h_goals,a_goals
```

Out[57]: (2, 0)

In [58]:
```
1 fin_h_goals,fin_a_goals
```

Out[58]: (5, 0)

In [59]:
```
1 team_a_lambda = 2.316062 * 0.945796
2 team_b_lambda = 0.543255 * 0.929605
3 total_goals_prob, draw_prob, away_prob, local_prob,h_goals,a_goals,fi
4
```

In [60]:
```
1 h_goals,a_goals
```

Out[60]: (0, 2)

In [61]:
```
1 fin_h_goals,fin_a_goals
```

Out[61]:
```
(0, 5)
```

Now it is time to validate if the probabilities given by the poisson distribution are accurate or not

In [62]:
```
1 df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1668 entries, 33959 to 47368
Data columns (total 31 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   date           1668 non-null   object
 1   home_team      1668 non-null   object
 2   away_team      1668 non-null   object
 3   home_score     1668 non-null   float64
 4   away_score     1668 non-null   float64
 5   tournament     1668 non-null   object
 6   city           1668 non-null   object
 7   country        1668 non-null   object
 8   neutral        1668 non-null   bool
 9   Elo_h_after    1668 non-null   float64
 10  Elo_a_after    1668 non-null   float64
 11  Elo_h_before   1668 non-null   float64
 12  Elo_a_before   1668 non-null   float64
 13  probH          1668 non-null   float64
 14  probA          1668 non-null   float64
 15  att_h_after    1668 non-null   float64
 16  att_a_after    1668 non-null   float64
 17  att_h_before   1668 non-null   float64
 18  att_a_before   1668 non-null   float64
 19  def_h_after    1668 non-null   float64
 20  def_a_after    1668 non-null   float64
 21  def_h_before   1668 non-null   float64
 22  def_a_before   1668 non-null   float64
 23  XGhome         1668 non-null   float64
 24  XGaway         1668 non-null   float64
 25  total_score    1668 non-null   float64
 26  total_xG       1668 non-null   float64
 27  total_xG_bin   1668 non-null   category
 28  year           1668 non-null   int64
 29  home_diff_elo  1668 non-null   float64
 30  away_diff_elo  1668 non-null   float64
dtypes: bool(1), category(1), float64(22), int64(1), object(6)
memory usage: 394.4+ KB
```

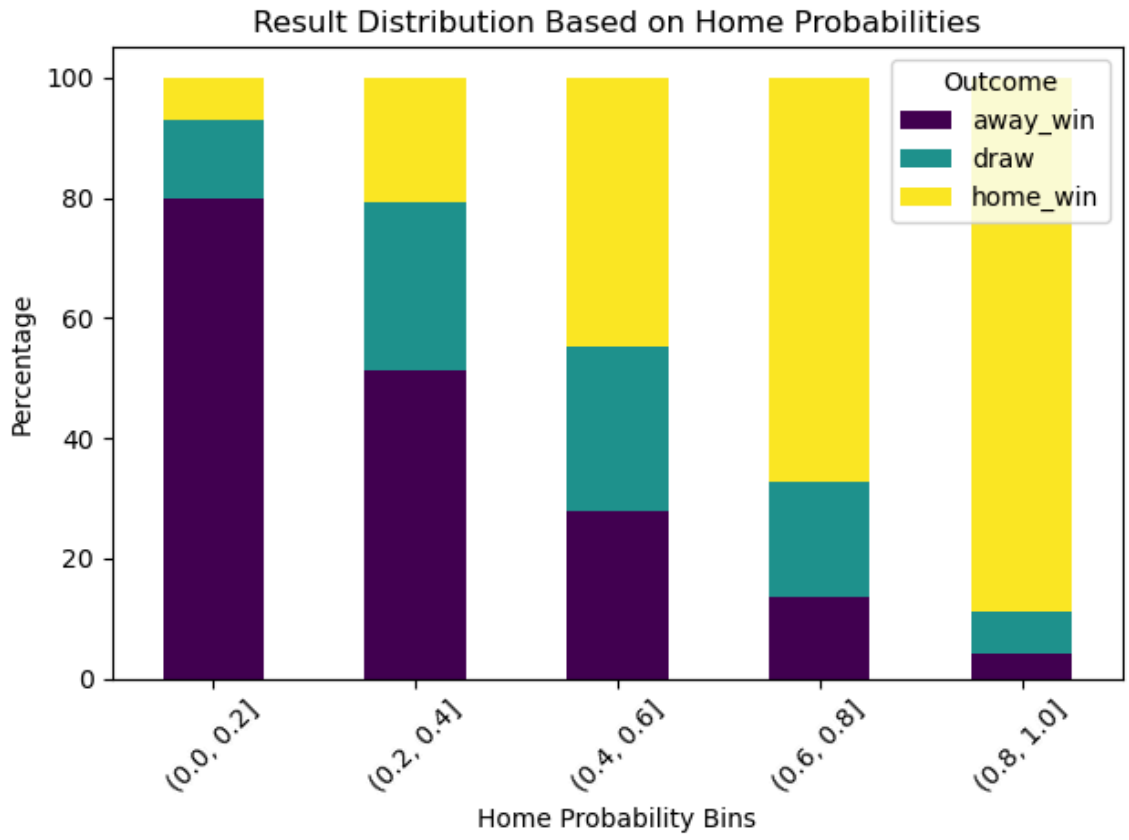
In [63]:

```
df_euro=df2.copy()
# Function to apply to each row of the DataFrame
def apply_calculation_elo(row):
#     home_diff = float(elos_ranking_euro[elos_ranking_euro['Team']==
#     away_diff = float(elos_ranking_euro[elos_ranking_euro['Team']==
    total_goals_prob, draw_prob, away_prob, local_prob,h_goals,a_goals
    return pd.Series({
        'total_goals_prob': total_goals_prob,
        'draw_prob': draw_prob,
        'away_prob': away_prob,
        'local_prob': local_prob,
        'home_goals': h_goals,
        'away_goals': a_goals,
         'home_goals_elo': fin_h_goals,
        'away_goals_elo': fin_a_goals,
    })

# Apply the function to each row
calculated_variables = df_euro.apply(apply_calculation_elo, axis=1)

# Concatenate the result with the original DataFrame
matches_df = pd.concat([df_euro, calculated_variables], axis=1)

df_euro[['total_goals_prob', 'draw_prob', 'away_prob', 'local_prob','l

# Iterate over each row in the DataFrame
for index, row in df_euro.iterrows():
    # Iterate over each element in the 'total_goals_prob' list
    for i, prob in enumerate(row['total_goals_prob']):
        # Create new column with the name 'prob_index_goals_i' and ass
        df_euro.at[index, f'prob_index_goals_{i}'] = prob
```

In [64]:

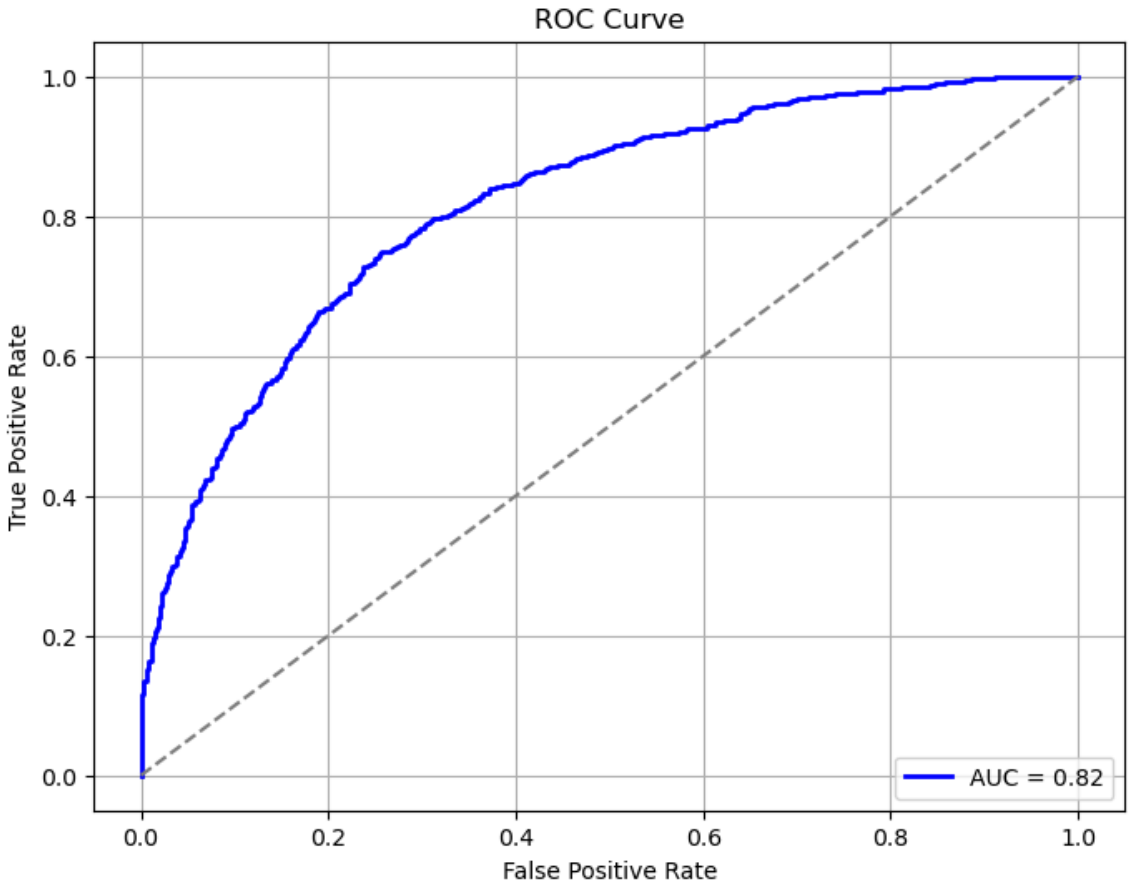
```
# Compute the actual outcome based on the home and away scores
df_euro['result'] = np.where(df_euro['home_score'] > df_euro['away_sc
                             np.where(df_euro['home_score'] == df_

df_euro['home_win_flag'] = np.where(df_euro['result']=='home_win', 1,
df_euro['draw_flag'] = np.where(df_euro['result']=='draw', 1,0)
df_euro['away_win_flag'] = np.where(df_euro['result']=='away_win', 1,
# Define bins for grouping based on predicted probabilities (adjust as
bins = [0, 0.2, 0.4,0.6, 0.8,1]

# Create a new column indicating the bin for each match based on predi
df_euro['probability_bin'] = pd.cut(df_euro['local_prob'], bins=bins)

# Group the DataFrame by the probability bins and compute the percenta
result_distribution = df_euro.groupby('probability_bin')['result'].va

# Plotting the result distribution based on home probabilities
plt.figure(figsize=(12, 6))
result_distribution.plot(kind='bar', stacked=True, cmap='viridis')
plt.title('Result Distribution Based on Home Probabilities')
plt.xlabel('Home Probability Bins')
plt.ylabel('Percentage')
plt.xticks(rotation=45)
plt.legend(title='Outcome')
plt.tight_layout()
```

```
<Figure size 1200x600 with 0 Axes>
```



In [65]:

```
from sklearn.metrics import roc_curve, roc_auc_score
import matplotlib.pyplot as plt

# Compute ROC curve
fpr, tpr, thresholds = roc_curve(df_euro['home_win_flag'], df_euro['l

# Compute AUC
auc = roc_auc_score(df_euro['home_win_flag'], df_euro['local_prob'])

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label=f'AUC = {auc:.2f}')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()

print(f"AUC: {auc:.2f}")
```




```
AUC: 0.82
```

It can be seen that the probabilities for home winning are working good.

In [66]:
```
# Create a new column indicating the bin for each match based on pred
bins = [0, 0.15, 0.3,0.6,1]
df_euro['probability_bin'] = pd.cut(df_euro['away_prob'], bins=bins)

# Group the DataFrame by the probability bins and compute the percent
result_distribution = df_euro.groupby('probability_bin')['result'].va
print(result_distribution)
# Compute AUC
auc_away = roc_auc_score(df_euro['away_win_flag'], df_euro['away_prob

print(f"AUC: {auc_away:.2f}")
```

```
result            away_win       draw   home_win
probability_bin
(0.0, 0.15]       8.007449  14.152700  77.839851
(0.15, 0.3]      25.223214  27.455357  47.321429
(0.3, 0.6]       53.053435  26.717557  20.229008
(0.6, 1.0]       87.421384   7.547170   5.031447
AUC: 0.81
```

It can be said the same for the away probabilities, so we can conclude that this predictions are relative good for the matches in Euro.

6 2024 Euro Prediction

Once we have the probabilities, we can finally use it to predict the results of the 2024 Euro. Let see it.

In [88]:

```
matches_data = [
    # Grupo 16
    {"Grupo": "phase3", "Home_Team": "Spain", "Away_Team": "France"},
    {"Grupo": "phase3", "Home_Team": "Netherlands", "Away_Team": "Engl
    {"Grupo": "phase3", "Home_Team": "Spain", "Away_Team": "England"}
#     {"Grupo": "16", "Home_Team": "Spain", "Away_Team": "Georgia"},
#     {"Grupo": "16", "Home_Team": "France", "Away_Team": "Belgium"},
#     {"Grupo": "16", "Home_Team": "Portugal", "Away_Team": "Slovenia
#     {"Grupo": "16", "Home_Team": "Romania", "Away_Team": "Netherlan
#     {"Grupo": "16", "Home_Team": "Austria", "Away_Team": "Turkey"},

#     #Grupo 8
#     {"Grupo": "8", "Home_Team": "Italy", "Away_Team": "England"},
#     {"Grupo": "8", "Home_Team": "Germany", "Away_Team": "Spain"},
#     {"Grupo": "8", "Home_Team": "France", "Away_Team": "Portugal"},
#     {"Grupo": "8", "Home_Team": "Netherlands", "Away_Team": "Austri




#     {"Grupo": "16", "Home_Team": "Slovenia", "Away_Team": "Denmark"
#     {"Grupo": "16", "Home_Team": "Serbia", "Away_Team": "England"},
#     {"Grupo": "16", "Home_Team": "Slovenia", "Away_Team": "Serbia"},
#     {"Grupo": "16", "Home_Team": "Denmark", "Away_Team": "England"},
#     {"Grupo": "16", "Home_Team": "England", "Away_Team": "Slovenia"
#     {"Grupo": "16", "Home_Team": "Denmark", "Away_Team": "Serbia"},
#     # Grupo D
#     {"Grupo": "16", "Home_Team": "Poland", "Away_Team": "Netherlands
#     {"Grupo": "16", "Home_Team": "Austria", "Away_Team": "France"},
#     {"Grupo": "16", "Home_Team": "Poland", "Away_Team": "Austria"},
#     {"Grupo": "16", "Home_Team": "Netherlands", "Away_Team": "France
#     {"Grupo": "D", "Home_Team": "Netherlands", "Away_Team": "Austri
#     {"Grupo": "D", "Home_Team": "France", "Away_Team": "Poland"},
#     # Grupo E
#     {"Grupo": "E", "Home_Team": "Romania", "Away_Team": "Ukraine"},
#     {"Grupo": "E", "Home_Team": "Belgium", "Away_Team": "Slovakia"},
#     {"Grupo": "E", "Home_Team": "Slovakia", "Away_Team": "Ukraine"},
#     {"Grupo": "E", "Home_Team": "Belgium", "Away_Team": "Romania"},
#     {"Grupo": "E", "Home_Team": "Slovakia", "Away_Team": "Romania"},
#     {"Grupo": "E", "Home_Team": "Ukraine", "Away_Team": "Belgium"},
#     # Grupo F
#     {"Grupo": "F", "Home_Team": "Turkey", "Away_Team": "Georgia"},
#     {"Grupo": "F", "Home_Team": "Portugal", "Away_Team": "Czech Repu
#     {"Grupo": "F", "Home_Team": "Georgia", "Away_Team": "Czech Repub
#     {"Grupo": "F", "Home_Team": "Turkey", "Away_Team": "Portugal"},
#     {"Grupo": "F", "Home_Team": "Georgia", "Away_Team": "Portugal"},
#     {"Grupo": "F", "Home_Team": "Czech Republic", "Away_Team": "Turk
]

# Create a dataframe
matches_df = pd.DataFrame(matches_data)
```

In [89]:

```
matches_df
```

Out[89]:

	Grupo	Home_Team	Away_Team
0	phase3	Spain	France
1	phase3	Netherlands	England
2	phase3	Spain	England

In [90]:
```
elos
```

Out[90]:

	date	Team	Elo	position
0	2024-07-04	Argentina	1776.345	1
1	2024-07-05	France	1750.370	2
2	2024-07-05	Spain	1735.715	3
3	2024-07-02	Brazil	1699.620	4
4	2024-06-30	England	1677.695	5
5	2024-07-02	Netherlands	1675.520	6
6	2024-07-02	Colombia	1671.785	7
7	2024-07-01	Uruguay	1661.435	8
8	2024-07-05	Germany	1643.990	9
9	2024-07-01	Belgium	1634.765	10
10	2024-07-05	Portugal	1634.685	11

In [91]:
```
# merged_df = pd.merge(matches_df, attdef[["Team", "att", "def"]].add_
# merged_df_elo = pd.merge(merged_df, elos[["Team","Elo"]].add_prefix(
# merged_df_elo_form = pd.merge(merged_df_elo, group16_eurosteams_sor
# group_matches_elo = pd.merge(merged_df_elo_form, attdef[["Team", "a
# group_matches_form= pd.merge(group_matches_elo, elos[["Team","Elo"]
# group_matches = pd.merge(group_matches_form, group16_eurosteams_sor
# # group_matches['form_Home_att']=group_matches['Home_att']+group_ma
# # group_matches['form_Home_def']=group_matches['Home_def']+group_ma
# # group_matches['form_Away_att']=group_matches['Away_att']+group_ma
# # group_matches['form_Away_def']=group_matches['Away_def']+group_ma
# group_matches
```

In [92]:
```
merged_df = pd.merge(matches_df, attdef[["Team", "att", "def"]].add_p
merged_df_elo = pd.merge(merged_df, elos[["Team","Elo"]].add_prefix('
merged_df_elo_form = pd.merge(merged_df_elo, group16_eurosteams_sorte
group_matches_elo = pd.merge(merged_df_elo_form, attdef[["Team", "att
group_matches_form= pd.merge(group_matches_elo, elos[["Team","Elo"]].
group_matches = pd.merge(group_matches_form, group16_eurosteams_sorte

group_matches
```

Out[92]:

	Grupo	Home_Team	Away_Team	Home_att	Home_def	Home_Elo	Home_euro_form	Awa
0	phase3	Spain	France	2.589579	0.457570	1735.715	2.000000	2.21
1	phase3	Netherlands	England	2.371421	0.618433	1675.520	0.888889	2.17
2	phase3	Spain	England	2.589579	0.457570	1735.715	2.000000	2.17

In [93]:
```
#Computing the expected goals
group_matches['XGhome']=group_matches['Home_att']*group_matches['Away_
group_matches['XGaway']=group_matches['Home_def']*group_matches['Away_
# group_matches['XGhome_form']=group_matches['Home_att']*group_matches
# group_matches['XGaway_form']=group_matches['Home_def']*group_matches
group_matches['home_diff_elo']=group_matches['Home_Elo']-group_matches
group_matches['away_diff_elo']=group_matches['Away_Elo']-group_matches

group_matches[['total_goals_prob', 'draw_prob', 'away_prob', 'local_pr
group_matches.drop("total_goals_prob",axis=1, inplace=True)
```

In [94]:
```

group_matches
```

Out[94]:

o	Away_euro_form	XGhome	XGaway	home_diff_elo	away_diff_elo	draw_prob	away_prob	l
0	1.111111	2.390389	1.128333	-14.655	14.655	0.182667	0.164327	
5	1.111111	1.000482	1.492086	-2.175	2.175	0.260620	0.485769	
5	1.111111	2.458173	1.103973	58.020	-58.020	0.176375	0.153105	

In [74]:

```
# Define a function to determine the result based on probabilities
def determine_result_random(row):
#     np.random.seed(1)  # Note: Setting the seed inside the function
    random_number = np.random.random()

    if random_number < row["draw_prob"]:
        return pd.Series(["Draw", random_number])
    elif random_number < row["draw_prob"] + row["away_prob"]:
        return pd.Series([row["Away_Team"], random_number])
    else:
        return pd.Series([row["Home_Team"], random_number])

def determine_result(row):
#     max_prob = max(row["draw_prob"], row["away_prob"], row["local_pr

    if row["home_goals_elo"] == row["away_goals_elo"]:
        return "Draw"
    elif row["away_goals_elo"] > row["away_goals_elo"]:
        return row["Away_Team"]
    else:
        return row["Home_Team"]

def determine_result_knockout(row):
    if row["away_prob"] >row["home_prob"]:
        return row["Away_Team"]
    else:
        return row["Home_Team"]

def determine_result_knockout_random(row):
    random_number = np.random.random()
    if random_number < row["draw_prob"]:
        random_number2 = np.random.random()
        if random_number < 0.5:
            return row["Away_Team"]
        else:
            return row["Home_Team"]
    elif random_number < row["draw_prob"] + row["away_prob"]:
        return row["Away_Team"]
    else:
        return row["Home_Team"]

# Apply the function to create the "Results" column
group_matches["Result"] = group_matches.apply(determine_result, axis=
```

We can see the predictions for each match, based on the expexted goals, we simulate the number of goals scored by each team in a match.

In [75]:
```
for x in range(-532,853):
    h_goals_adjust=int(reg.predict(np.array(x).reshape(-1, 1)))
    print(h_goals_adjust,x)
```

```
-1 -532
-1 -531
-1 -530
-1 -529
-1 -528
-1 -527
-1 -526
-1 -525
-1 -524
-1 -523
-1 -522
-1 -521
-1 -520
-1 -519
-1 -518
-1 -517
-1 -516
-1 -515
-1 -514
```

In [76]:
```
group_matches[group_matches['away_diff_elo']<-243]
```

Out[76]:

	Grupo	Home_Team	Away_Team	Home_att	Home_def	Home_Elo	Home_euro_form	Away
1	16	England	Slovakia	2.171419	0.474628	1677.695	1.111111	1.33
3	16	Spain	Georgia	2.589579	0.457570	1735.715	2.000000	1.18

In [77]:
```
group_matches[group_matches['home_diff_elo']<-243]
```

Out[77]:

	Grupo	Home_Team	Away_Team	Home_att	Home_def	Home_Elo	Home_euro_form	Away
6	16	Romania	Netherlands	1.310627	0.792474	1423.0	0.888889	2.37

In [78]:
```
group_matches[group_matches.Grupo=="A"][["Home_Team","Away_Team", "lo
```

Out[78]:

	Home_Team	Away_Team	local_prob	draw_prob	away_prob	home_goals	away_goals	hor

In [79]:
```
group_matches[group_matches.Grupo=="B"][["Home_Team","Away_Team", "lo
```

Out[79]:

	Home_Team	Away_Team	local_prob	draw_prob	away_prob	home_goals	away_goals	hor

In [80]:
```
group_matches[group_matches.Grupo=="C"][["Home_Team","Away_Team", "lo
```

Out[80]:

Home_Team	Away_Team	local_prob	draw_prob	away_prob	home_goals	away_goals	hor

In [81]:
```
group_matches[group_matches.Grupo=="D"][["Home_Team","Away_Team", "lo
```

Out[81]:

Home_Team	Away_Team	local_prob	draw_prob	away_prob	home_goals	away_goals	hor

In [82]:
```
group_matches[group_matches.Grupo=="E"][["Home_Team","Away_Team", "lo
```

Out[82]:

Home_Team	Away_Team	local_prob	draw_prob	away_prob	home_goals	away_goals	hor

In [83]:
```
group_matches[group_matches.Grupo=="F"][["Home_Team","Away_Team", "lo
```

Out[83]:

Home_Team	Away_Team	local_prob	draw_prob	away_prob	home_goals	away_goals	hor

By running numerous simulations, we can estimate the probabilities of different outcomes for each match. We simulate each match multiple times to predict the likely outcomes and the points each team might accumulate. The table presented below illustrates the predicted probabilities for each team to conclude the group stage in a particular position.

In [84]:
```
group_matches[group_matches['Result']=='Draw']
```

Out[84]:

	Grupo	Home_Team	Away_Team	Home_att	Home_def	Home_Elo	Home_euro_form	Awa
0	16	Switzerland	Italy	1.769522	0.723214	1594.235	1.111111	1.7
7	16	Austria	Turkey	1.585274	0.848737	1521.590	1.333333	1.5
10	8	France	Portugal	2.219333	0.461540	1750.370	1.111111	2.2
11	8	Netherlands	Austria	2.371421	0.618433	1675.520	0.888889	1.5

In [85]:
```
group_matches.columns
```

Out[85]:
```
Index(['Grupo', 'Home_Team', 'Away_Team', 'Home_att', 'Home_def', 'Home_El
o',
       'Home_euro_form', 'Away_att', 'Away_def', 'Away_Elo', 'Away_euro_fo
rm',
       'XGhome', 'XGaway', 'home_diff_elo', 'away_diff_elo', 'draw_prob',
       'away_prob', 'local_prob', 'home_goals', 'away_goals', 'home_goals_
elo',
       'away_goals_elo', 'Result'],
      dtype='object')
```

In [86]:

```python
def calculate_points(result, home_team, team):
    if result == "Draw":
        return 1
    elif result == home_team:
        return 3
    elif result == team:
        return 0
```

In [87]:

```
# Simulate match results and calculate standings
num_simulations = 10000  # Number of simulations
standings_list = []  # List to store standings for each simulation

for _ in range(num_simulations):
    # Apply the function to create the "Results" column
    group_matches[["Random_Result",'Random_number']] = group_matches.a
    group_matches["Result"] = group_matches.apply(determine_result, a
    # Apply the function to create the "Points" column for both Home
    group_matches["Points_Home"] = group_matches.apply(lambda row: ca
    group_matches["Points_Away"] = group_matches.apply(lambda row: ca
    # Create a dataframe to store points for each teams
    standings = pd.DataFrame(columns=["Grupo","Team", "Points"])
    # Concatenate Home and Away points by group
    for group_name, group_data in group_matches.groupby("Grupo"):
        group_points_df = pd.DataFrame(columns=["Team", "Points"])
        for team in set(group_data["Home_Team"]).union(set(group_data
            home_points = group_data.loc[group_data["Home_Team"] == t
            away_points = group_data.loc[group_data["Away_Team"] == t
            total_points = home_points + away_points
            group_points_df = group_points_df._append({"Team": team,
        group_points_df = group_points_df.sort_values(by=["Points"], a
        group_points_df["Grupo"] = group_name
        group_points_df["Rank"] = range(1, len(group_points_df) + 1)
        group_points_df["Qualified_1"]=np.where(group_points_df["Rank
        group_points_df["Qualified_2"]=np.where(group_points_df["Rank
        group_points_df["Qualified_3"]=np.where(group_points_df["Rank
        group_points_df["Qualified_4"]=np.where(group_points_df["Rank
        standings = pd.concat([standings, group_points_df], ignore_in
        standings_list.append(standings)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
File C:\ProgramData\anaconda3\Lib\site-packages\pandas\core\indexes\base.py:3653, in Index.get_loc(self, key)
   3652 try:
-> 3653     return self._engine.get_loc(casted_key)
   3654 except KeyError as err:

File C:\ProgramData\anaconda3\Lib\site-packages\pandas\_libs\index.pyx:147, in pandas._libs.index.IndexEngine.get_loc()

File C:\ProgramData\anaconda3\Lib\site-packages\pandas\_libs\index.pyx:176, in pandas._libs.index.IndexEngine.get_loc()

File pandas\_libs\hashtable_class_helper.pxi:7080, in pandas._libs.hashtable.PyObjectHashTable.get_item()

File pandas\_libs\hashtable_class_helper.pxi:7088, in pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 'Rank'

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
File C:\ProgramData\anaconda3\Lib\site-packages\pandas\core\frame.py:4110, in DataFrame._set_item_mgr(self, key, value)
   4109 try:
-> 4110     loc = self._info_axis.get_loc(key)
   4111 except KeyError:
   4112     # This item wasn't present, just insert at end

File C:\ProgramData\anaconda3\Lib\site-packages\pandas\core\indexes\base.py:3655, in Index.get_loc(self, key)
   3654 except KeyError as err:
-> 3655     raise KeyError(key) from err
   3656 except TypeError:
   3657     # If we have a listlike key, _check_indexing_error will raise
   3658     #  InvalidIndexError. Otherwise we fall through and re-raise
   3659     #  the TypeError.

KeyError: 'Rank'

During handling of the above exception, another exception occurred:

KeyboardInterrupt                         Traceback (most recent call last)
Cell In[87], line 25
     23 group_points_df = group_points_df.sort_values(by=["Points"], ascending=False).reset_index(drop=True)
     24 group_points_df["Grupo"] = group_name
---> 25 group_points_df["Rank"] = range(1, len(group_points_df) + 1)
     26 group_points_df["Qualified_1"]=np.where(group_points_df["Rank"]==1,1,0)
     27 group_points_df["Qualified_2"]=np.where(group_points_df["Rank"]==2,1,0)

File C:\ProgramData\anaconda3\Lib\site-packages\pandas\core\frame.py:3950,
```

```
in DataFrame.__setitem__(self, key, value)
   3947     self._setitem_array([key], value)
   3948 else:
   3949     # set column
-> 3950     self._set_item(key, value)

File C:\ProgramData\anaconda3\Lib\site-packages\pandas\core\frame.py:4156,
in DataFrame._set_item(self, key, value)
   4153         if isinstance(existing_piece, DataFrame):
   4154             value = np.tile(value, (len(existing_piece.columns),
1)).T
-> 4156 self._set_item_mgr(key, value)

File C:\ProgramData\anaconda3\Lib\site-packages\pandas\core\frame.py:4113,
in DataFrame._set_item_mgr(self, key, value)
   4110     loc = self._info_axis.get_loc(key)
   4111 except KeyError:
   4112     # This item wasn't present, just insert at end
-> 4113     self._mgr.insert(len(self._info_axis), key, value)
   4114 else:
   4115     self._iset_item_mgr(loc, value)

File C:\ProgramData\anaconda3\Lib\site-packages\pandas\core\internals\mana
gers.py:1419, in BlockManager.insert(self, loc, item, value)
   1417 else:
   1418     self._insert_update_mgr_locs(loc)
-> 1419     self._insert_update_blklocs_and_blknos(loc)
   1421 self.axes[0] = new_axis
   1422 self.blocks += (block,)

File C:\ProgramData\anaconda3\Lib\site-packages\pandas\core\internals\mana
gers.py:1456, in BlockManager._insert_update_blklocs_and_blknos(self, loc)
   1453 if loc == self.blklocs.shape[0]:
   1454     # np.append is a lot faster, let's use it if we can.
   1455     self._blklocs = np.append(self._blklocs, 0)
-> 1456     self._blknos = np.append(self._blknos, len(self.blocks))
   1457 elif loc == 0:
   1458     # np.append is a lot faster, let's use it if we can.
   1459     self._blklocs = np.append(self._blklocs[::-1], 0)[::-1]

File <__array_function__ internals>:200, in append(*args, **kwargs)

File C:\ProgramData\anaconda3\Lib\site-packages\numpy\lib\function_base.p
y:5499, in append(arr, values, axis)
   5497     values = ravel(values)
   5498     axis = arr.ndim-1
-> 5499 return concatenate((arr, values), axis=axis)

File <__array_function__ internals>:200, in concatenate(*args, **kwargs)

KeyboardInterrupt:
```

In [ ]:
```
1 group_matches[group_matches['Result']!=group_matches['Random_Result']
```

In [ ]:
```
standings = pd.concat(standings_list).groupby(["Team","Grupo"])[["Qua
standings_style = standings.style.background_gradient(cmap='Greens')
standings_style
```

In [ ]:
```
group_matches['home_goals_elo'].sum()+group_matches['away_goals_elo']
```

In [95]:
```


# Calculate total home goals for each team
home_goals = group_matches.groupby('Home_Team')['home_goals_elo'].sum

# Calculate total away goals for each team
away_goals = group_matches.groupby('Away_Team')['away_goals_elo'].sum

# Combine home and away goals to get the total goals for each team
total_goals = home_goals.add(away_goals, fill_value=0)

# Convert the Series to DataFrame for better presentation
total_goals_df = total_goals.reset_index()
total_goals_df.columns = ['Team', 'Total_Goals']
total_goals_df.sort_values('Total_Goals',ascending=False,inplace=True
total_goals_df
```

Out[95]:

	Team	Total_Goals
3	Spain	4.0
0	England	2.0
1	France	1.0
2	Netherlands	1.0

7 Top Goal scorer

In [96]:
```
euro_players=pd.read_excel('euro_players.xlsx')
euro_players
```

Out[96]:

	Pos.	Player	Date of birth	age	Caps	Goals	G/C	Club	Countr
0	FW	Niclas Füllkrug	1993-02-09	31	15	11	0.733333	Germany Borussia Dortmund	Germar
1	FW	Romelu Lukaku	1993-05-13	31	114	83	0.728070	Italy Roma	Belgiu
2	FW	Harry Kane (captain)	1993-07-28	30	90	63	0.700000	Germany Bayern Munich	Englan
3	FW	Gonçalo Ramos	2001-06-20	22	12	8	0.666667	France Paris Saint-Germain	Portug
4	FW	Tomáš Chorý	1995-01-26	29	3	2	0.666667	Czech Republic Viktoria Plzeň	Czec Republ
5	FW	Aleksandar Mitrović	1994-09-16	29	90	57	0.633333	Saudi Arabia Al Hilal	Serbi

In [97]:
```
1 euro_players.columns
```

Out[97]:
```
Index(['Pos.', 'Player', 'Date of birth ', 'age', 'Caps', 'Goals', 'G/C',
       'Club', 'Country', 'Manager', 'Group'],
      dtype='object')
```

In [98]:
```
1 sorted_df = euro_players.groupby('Country').apply(lambda x: x.sort_va
2 sorted_df.reset_index(drop=True, inplace=True)
3 sorted_df['Rank'] = sorted_df.groupby('Country')['G/C'].rank(method='
4 final_players=sorted_df[sorted_df['Rank'] <=5][['Player','Country','G/
5 final_players
```

Out[98]:

	Player	Country	G/C	Rank
0	Jasir Asani	Albania	0.333333	1.0
1	Ernest Muçi	Albania	0.300000	2.0
2	Armando Broja	Albania	0.250000	3.0
3	Rey Manaj	Albania	0.212121	4.0
4	Mirlind Daku	Albania	0.200000	5.0
27	Christoph Baumgartner	Austria	0.378378	1.0
28	Maximilian Entrup	Austria	0.333333	2.0
29	Marko Arnautović	Austria	0.321429	3.0
30	Michael Gregoritsch	Austria	0.277778	4.0
31	Marcel Sabitzer	Austria	0.217949	5.0
56	Romelu Lukaku	Belgium	0.728070	1.0

In [99]:

```
final_players_sort = pd.merge(final_players, total_goals_df, left_on=
final_players_sort
# Calculate proportional goals based on G/C ratio
# Normalize the G/C ratio within each country to distribute goals
# final_players_sort['G/C Normalized'] = final_players_sort.groupby('
# final_players_sort['Player Goals'] = (final_players_sort['G/C Normal

# final_players_sort.sort_values('Player Goals',ascending=False,inplac

```

Out[99]:

	Player	Country	G/C	Rank	Team	Total_Goals
0	Harry Kane (captain)	England	0.700000	1.0	England	2.0
1	Ivan Toney	England	0.500000	2.0	England	2.0
2	Bukayo Saka	England	0.343750	3.0	England	2.0
3	Cole Palmer	England	0.333333	4.0	England	2.0
4	Ollie Watkins	England	0.250000	5.0	England	2.0
5	Kylian Mbappé (captain)	France	0.597403	1.0	France	1.0
6	Warren Zaïre-Emery	France	0.500000	2.0	France	1.0
7	Olivier Giroud	France	0.435115	3.0	France	1.0
8	Antoine Griezmann	France	0.346457	4.0	France	1.0
9	Randal Kolo Muani	France	0.200000	5.0	France	1.0
10	Memphis Depay	Netherlands	0.488889	1.0	Netherlands	1.0
11	Cody Gakpo	Netherlands	0.391304	2.0	Netherlands	1.0
12	Georginio Wijnaldum	Netherlands	0.307692	3.0	Netherlands	1.0
13	Wout Weghorst	Netherlands	0.290323	4.0	Netherlands	1.0
14	Steven Bergwijn	Netherlands	0.250000	5.0	Netherlands	1.0
15	Joselu	Spain	0.500000	1.0	Spain	4.0
16	Álex Baena	Spain	0.500000	1.0	Spain	4.0
17	Álvaro Morata (captain)	Spain	0.478873	2.0	Spain	4.0
18	Ferran Torres	Spain	0.450000	3.0	Spain	4.0
19	Lamine Yamal	Spain	0.333333	4.0	Spain	4.0
20	Mikel Oyarzabal	Spain	0.250000	5.0	Spain	4.0

In [100]:
```
final_players_sort['G/C_Normalized'] = final_players_sort.groupby('Co
final_players_sort
```

Out[100]:

	Player	Country	G/C	Rank	Team	Total_Goals	G/C_Normalized
0	Harry Kane (captain)	England	0.700000	1.0	England	2.0	0.329089
1	Ivan Toney	England	0.500000	2.0	England	2.0	0.235064
2	Bukayo Saka	England	0.343750	3.0	England	2.0	0.161606
3	Cole Palmer	England	0.333333	4.0	England	2.0	0.156709
4	Ollie Watkins	England	0.250000	5.0	England	2.0	0.117532
5	Kylian Mbappé (captain)	France	0.597403	1.0	France	1.0	0.287355
6	Warren Zaïre-Emery	France	0.500000	2.0	France	1.0	0.240503
7	Olivier Giroud	France	0.435115	3.0	France	1.0	0.209293
8	Antoine Griezmann	France	0.346457	4.0	France	1.0	0.166648
9	Randal Kolo Muani	France	0.200000	5.0	France	1.0	0.096201
10	Memphis Depay	Netherlands	0.488889	1.0	Netherlands	1.0	0.282888
11	Cody Gakpo	Netherlands	0.391304	2.0	Netherlands	1.0	0.226422
12	Georginio Wijnaldum	Netherlands	0.307692	3.0	Netherlands	1.0	0.178041
13	Wout Weghorst	Netherlands	0.290323	4.0	Netherlands	1.0	0.167991
14	Steven Bergwijn	Netherlands	0.250000	5.0	Netherlands	1.0	0.144659
15	Joselu	Spain	0.500000	1.0	Spain	4.0	0.199028
16	Álex Baena	Spain	0.500000	1.0	Spain	4.0	0.199028
17	Álvaro Morata (captain)	Spain	0.478873	2.0	Spain	4.0	0.190619
18	Ferran Torres	Spain	0.450000	3.0	Spain	4.0	0.179125
19	Lamine Yamal	Spain	0.333333	4.0	Spain	4.0	0.132685
20	Mikel Oyarzabal	Spain	0.250000	5.0	Spain	4.0	0.099514

In [101]:
```
final_players_sort['Player_Goals'] = np.floor(final_players_sort['G/C_

unique_total_goals = final_players_sort.drop_duplicates(subset=['Coun

# Calculate remaining goals after initial assignment
assigned_goals = final_players_sort['Player_Goals'].sum()
remaining_goals = unique_total_goals - assigned_goals

remaining_goals
```

Out[101]: 8.0

In [102]:

```
# Distribute remaining goals starting from the players with the highes
sorted_indices = final_players_sort.sort_values(by=['G/C_Normalized',

# Ensure we do not exceed the number of available players
num_players = len(sorted_indices)
to_distribute = min(remaining_goals, num_players)

# Loop to increment goals for the top ranked players based on remainir
for i in range(int(to_distribute)):
    final_players_sort.loc[sorted_indices[i], 'Player_Goals'] += 1

final_players_sort
```

Out[102]:

	Player	Country	G/C	Rank	Team	Total_Goals	G/C_Normalized	Player_
0	Harry Kane (captain)	England	0.700000	1.0	England	2.0	0.329089	
1	Ivan Toney	England	0.500000	2.0	England	2.0	0.235064	
2	Bukayo Saka	England	0.343750	3.0	England	2.0	0.161606	
3	Cole Palmer	England	0.333333	4.0	England	2.0	0.156709	
4	Ollie Watkins	England	0.250000	5.0	England	2.0	0.117532	
5	Kylian Mbappé (captain)	France	0.597403	1.0	France	1.0	0.287355	
6	Warren Zaïre-Emery	France	0.500000	2.0	France	1.0	0.240503	
7	Olivier Giroud	France	0.435115	3.0	France	1.0	0.209293	
8	Antoine Griezmann	France	0.346457	4.0	France	1.0	0.166648	
9	Randal Kolo Muani	France	0.200000	5.0	France	1.0	0.096201	
10	Memphis Depay	Netherlands	0.488889	1.0	Netherlands	1.0	0.282888	
11	Cody Gakpo	Netherlands	0.391304	2.0	Netherlands	1.0	0.226422	
12	Georginio Wijnaldum	Netherlands	0.307692	3.0	Netherlands	1.0	0.178041	
13	Wout Weghorst	Netherlands	0.290323	4.0	Netherlands	1.0	0.167991	
14	Steven Bergwijn	Netherlands	0.250000	5.0	Netherlands	1.0	0.144659	
15	Joselu	Spain	0.500000	1.0	Spain	4.0	0.199028	
16	Álex Baena	Spain	0.500000	1.0	Spain	4.0	0.199028	
17	Álvaro Morata (captain)	Spain	0.478873	2.0	Spain	4.0	0.190619	
18	Ferran Torres	Spain	0.450000	3.0	Spain	4.0	0.179125	
19	Lamine Yamal	Spain	0.333333	4.0	Spain	4.0	0.132685	
20	Mikel Oyarzabal	Spain	0.250000	5.0	Spain	4.0	0.099514	

In [103]:
```
1 final_players_sort['Player_Goals'].sum()
```

Out[103]: 8

In [104]:
```
final_players_sort.sort_values('Player_Goals',ascending=False,inplace
final_players_sort[:10]
```

Out[104]:

	Player	Country	G/C	Rank	Team	Total_Goals	G/C_Normalized	Player_
0	Harry Kane (captain)	England	0.700000	1.0	England	2.0	0.329089	
6	Warren Zaïre-Emery	France	0.500000	2.0	France	1.0	0.240503	
15	Joselu	Spain	0.500000	1.0	Spain	4.0	0.199028	
11	Cody Gakpo	Netherlands	0.391304	2.0	Netherlands	1.0	0.226422	
1	Ivan Toney	England	0.500000	2.0	England	2.0	0.235064	
7	Olivier Giroud	France	0.435115	3.0	France	1.0	0.209293	
10	Memphis Depay	Netherlands	0.488889	1.0	Netherlands	1.0	0.282888	
5	Kylian Mbappé (captain)	France	0.597403	1.0	France	1.0	0.287355	
8	Antoine Griezmann	France	0.346457	4.0	France	1.0	0.166648	
9	Randal Kolo Muani	France	0.200000	5.0	France	1.0	0.096201	

In [105]:
```
"Niclas Füllkrug"
```

Out[105]: 'Niclas Füllkrug'

As we can see, the teams that are more likely to win its respecctive group are Portugal, Belgium and England. And the team which have more difficult to pass the round is Albania.

8 Conclusion

Predicting football match outcomes is inherently uncertain, but statistical models like the Poisson distribution offer a structured approach to quantify this uncertainty. Our predictions for the 2024 UEFA Euro provide a glimpse into possible outcomes based on historical performance data. As with all models, the key is to continually refine our approach as new data becomes available.