```
from math import exp, factorial

def ResultProbs(HomeAttack, HomeDefence, AwayAttack, AwayDefence):
    HomeMean = HomeAttack * AwayDefence
    AwayMean = AwayAttack * HomeDefence
    HomeWin = 0
    Draw = 0
    AwayWin = 0
    for i in range(16):
        for j in range(16):
            if i > j:
                HomeWin += (exp(-HomeMean) * (HomeMean ** i) / factorial(i)) * \
                           (exp(-AwayMean) * (AwayMean ** j) / factorial(j))
            elif i == j:
                Draw += (exp(-HomeMean) * (HomeMean ** i) / factorial(i)) * \
                        (exp(-AwayMean) * (AwayMean ** j) / factorial(j))
            else:
                AwayWin += (exp(-HomeMean) * (HomeMean ** i) / factorial(i)) * \
                           (exp(-AwayMean) * (AwayMean ** j) / factorial(j))
    return HomeWin, Draw, AwayWin
```

```
import pandas as pd
import numpy as np
from scipy.optimize import fsolve
from datetime import datetime


# Results Data CSV
path_to_results_csv = 'results.csv'

# Results To Predict CSV
path_to_predictions_csv = 'PredictionsToMake.csv'

# Parameters
Start_Date = datetime(1999, 12, 31)
End_Date = datetime(2024, 6, 10)


KnockoutTiebreaker = 1

# Import Data from Results CSV
results = pd.read_csv(path_to_results_csv)
results['DATE'] = pd.to_datetime(results['date'])

# Extract Relevant Data
start_index = results[results['DATE'] >= Start_Date].index.min()
end_index = results[results['DATE'] <= End_Date].index.max() + 1
relevant_results = results.loc[start_index:end_index]
```

```
# Select Relevant Teams
RelevantTeams = [
    'Albania', 'Andorra', 'Armenia', 'Austria', 'Azerbaijan', 'Belarus',
    'Belgium', 'Bosnia and Herzegovina', 'Bulgaria', 'Croatia', 'Cyprus',
    'Czech Republic', 'Denmark', 'England', 'Estonia', 'Faroe Islands',
    'Finland', 'France', 'Georgia', 'Germany', 'Gibraltar', 'Greece',
    'Hungary', 'Iceland', 'Republic of Ireland', 'Israel', 'Italy',
    'Kazakhstan', 'Kosovo', 'Latvia', 'Liechtenstein', 'Lithuania',
    'Luxembourg', 'Malta', 'Moldova', 'Montenegro', 'Netherlands',
    'North Macedonia', 'Northern Ireland', 'Norway', 'Poland', 'Portugal',
    'Romania', 'Russia', 'Scotland', 'Serbia', 'Slovakia', 'Slovenia',
    'Spain', 'Sweden', 'Switzerland', 'Turkey', 'Ukraine', 'Wales'
]



# Create Poisson Model
GamesPlayed = np.zeros((len(RelevantTeams), len(RelevantTeams)))
GoalsScoredMatrix = np.zeros((len(RelevantTeams), len(RelevantTeams)))
GoalsScored = np.zeros(len(RelevantTeams))
GoalsConc = np.zeros(len(RelevantTeams))

for _, row in relevant_results.iterrows():
    if row['home_team'] in RelevantTeams and row['away_team'] in RelevantTeams:
        idx_home = RelevantTeams.index(row['home_team'])
        idx_away = RelevantTeams.index(row['away_team'])
        GamesPlayed[idx_home, idx_away] += 1
        GamesPlayed[idx_away, idx_home] += 1
        GoalsScoredMatrix[idx_home, idx_away] += row['home_score']
        GoalsScoredMatrix[idx_away, idx_home] += row['away_score']
        GoalsScored[idx_home] += row['home_score']
        GoalsScored[idx_away] += row['away_score']
        GoalsConc[idx_home] += row['away_score']
        GoalsConc[idx_away] += row['home_score']

Checkcount = 1
P = GamesPlayed
GS = GoalsScoredMatrix
Objective = lambda x: np.abs((np.block([[P, np.zeros((54, 54))], [np.zeros((54, 54)), P]]) @ x) *
                             np.block([x[54:], x[:54]]) - np.block([GoalsScored, GoalsConc]))
Values = np.block([GoalsScored / np.sum(GoalsScored), GoalsConc / np.sum(GoalsConc)])


while np.sum(Objective(Values) ** 2) > 0.001:
    Values = fsolve(Objective, Values)
    print(np.sum(Objective(Values) ** 2))
Attacks = Values[54:]
Defences = Values[:54]


# Calculate Predictions
preds = pd.read_csv(path_to_predictions_csv)

Home_Team = preds.iloc[:, 0]
Away_Team = preds.iloc[:, 1]
Match_Type = preds.iloc[:, 2]

Home_Win_Prob = np.zeros(len(Home_Team))
Draw_Prob = np.zeros(len(Home_Team))
Away_Win_Prob = np.zeros(len(Home_Team))

for n in range(len(Home_Team)):
    if Home_Team[n] not in RelevantTeams or Away_Team[n] not in RelevantTeams:
        Home_Win_Prob[n] = -1
        Draw_Prob[n] = -1
        Away_Win_Prob[n] = -1
        print(f'Error in Predicting Match Number {n}')
    else:
        idx_home = RelevantTeams.index(Home_Team[n])
        idx_away = RelevantTeams.index(Away_Team[n])
        # Predicting Result After 90 minutes
        HomeWin, Draw, AwayWin = ResultProbs(Attacks[idx_home], Defences[idx_home],
                                             Attacks[idx_away], Defences[idx_away])
        if Match_Type[n] == 'Knockout':
            # Accounting for extra time
            if KnockoutTiebreaker == 1:
                HomeWinET, DrawET, AwayWinET = ResultProbs(Attacks[idx_home] / 3, Defences[idx_home] / 3,
                                                            Attacks[idx_away] / 3, Defences[idx_away] / 3)
                HomeWin += Draw * HomeWinET
                AwayWin += Draw * AwayWinET
                Draw *= DrawET
            # Accounting for penalties
            HomeWin += 0.5 * Draw
            AwayWin += 0.5 * Draw
            Draw = 0
        Home_Win_Prob[n] = HomeWin
        Away_Win_Prob[n] = AwayWin
        Draw_Prob[n] = Draw

Output = pd.DataFrame({'Home_Team': Home_Team, 'Away_Team': Away_Team, 'Match_Type': Match_Type,
                       'Home_Win_Prob': Home_Win_Prob, 'Draw_Prob': Draw_Prob, 'Away_Win_Prob': Away_Win_Prob})
Output.to_csv('PredictionsMade.csv', index=False)
```

```
2.5280078174017458e-11
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_minpack_py.py:177: RuntimeWarning: The iteration is not making good progress, as
  improvement from the last ten iterations.
  warnings.warn(msg, RuntimeWarning)
```

```
from datetime import date
from matplotlib import pyplot as plt
import numpy as np
from numpy.polynomial.polynomial import polyfit
import pandas as pd
from itertools import product

class Team:

    def __init__(self, name):
        self.name = name
        self.elo = 1500  # Set the initial Elo value of 1500

    def expected_outcome(self, other, neutral, homeadv, divisor, base):
        if neutral == True:  # Is there a home advantage to consider?
            return 1 / (1 + base ** ((other.elo - self.elo) / divisor))
        else:
            return 1 / (1 + base ** ((other.elo - (self.elo + homeadv)) / divisor))

    def update_elo(self,other,goals_self,goals_other,neutral,homeadv,divisor,base,K):

        # Calculate the actual outcome W
        goal_diff = goals_self - goals_other
        if goal_diff > 0:
            W = 1
        elif goal_diff == 0:
            W = 0.5
        else:
            W = 0

        # Calculate the expected outcome W_e
        W_e = self.expected_outcome(other, neutral, homeadv, divisor, base)

        # Calculate the factor to multiply with K based on the goal difference
        # K_factor = abs(goal_diff) * K_factor

        # Calculate K_final
        # K_final = K * (1 + K_factor)

        # Update both Elo ratings
        self.elo = self.elo + K * (W - W_e)
        other.elo = other.elo + K * (W_e - W)



def elo_prob(HELO:float, AELO:float, homeadv:int=0, base:int=7, divisor:int=250):
    return 1 / (1 + base ** ((AELO - (HELO + homeadv)) / divisor))

def get_rating_own_elo(team, elos):
    return elos.loc[elos['Team'] == team, 'Elo'].iloc[0]
```

```
def identify_rel_matches(team:str, matches:pd.DataFrame, ELO:float, elos:pd.DataFrame, ELO_range=200):
    matches['Opponent'] = matches.apply(lambda x: x.away_team if x.home_team == team else x.home_team, axis=1)
    matches['ELO Opponent'] = matches.apply(lambda x: get_rating_own_elo(x['Opponent'], elos), axis=1)
    matches['Similar Opponent'] = (matches['ELO Opponent'] < ELO + ELO_range) & (matches['ELO Opponent'] > ELO - ELO_range)
    return matches

def last_n_matches_team(team:str, n_games:int, due_date=str(date.today())):
    df_matches = pd.read_csv('results.csv', parse_dates=['date'])
    df_matches = df_matches[df_matches["date"] < due_date]
    df_matches_team = df_matches[(df_matches['home_team'] == team) | (df_matches['away_team'] == team)]
    df_matches_team = df_matches_team.sort_values('date', ascending=False)
    return df_matches_team.head(n_games)

def mean_goals(df:pd.DataFrame, team:str, n_games:int=10):

    if isinstance(df, pd.DataFrame) == False:
        df = last_n_matches_team(team, n_games)
    sum_goals = df['home_score'].sum() + df['away_score'].sum()
    return sum_goals / df.shape[0]

def result(team_home, team_away, elos, home_advantage=False, n_output_rows=5, n_games=10, sort_col='EP', ELO_range=150, due_date=str(date.toda


    # Retrieve current ELO ratings of both teams
    ELO_home = get_rating_own_elo(team_home, elos)
    ELO_away = get_rating_own_elo(team_away, elos)
    print('ELO', team_home, ':', format(ELO_home, ".0f"))
    print('ELO', team_away, ':', format(ELO_away, ".0f"))
    print('\n')

    # Calculate estimate for the mean number of goals for each team
    # --> Maximum Likelihood Estimate for the Poisson parameter lambda
    team_home_last_matches = last_n_matches_team(team_home, n_games, due_date)
    team_away_last_matches = last_n_matches_team(team_away, n_games, due_date)


    team_home_last_matches_rel = identify_rel_matches(team_home, team_home_last_matches, ELO_away, elos, ELO_range)
    team_away_last_matches_rel = identify_rel_matches(team_away, team_away_last_matches, ELO_home, elos, ELO_range)

    ELO_range_home = ELO_range
    ELO_range_away = ELO_range

    while team_home_last_matches_rel["Similar Opponent"].sum() < 3:
        ELO_range_home = ELO_range_home + 20
        team_home_last_matches_rel = identify_rel_matches(team_home, team_home_last_matches, ELO_away, elos, ELO_range_home)

    while team_away_last_matches_rel["Similar Opponent"].sum() < 3:
        ELO_range_away = ELO_range_away + 20
        team_away_last_matches_rel = identify_rel_matches(team_away, team_away_last_matches, ELO_home, elos, ELO_range_away)

    print('Last', n_games, 'matches of', team_home, ':')
    print(team_home_last_matches_rel[['date', 'home_team', 'away_team', 'home_score', 'away_score', 'tournament', 'Similar Opponent']])
    print('\n')
    print('Last', n_games, 'matches of', team_away, ':')
    print(team_away_last_matches_rel[['date', 'home_team', 'away_team', 'home_score', 'away_score', 'tournament', 'Similar Opponent']])
    print('\n')

    # Select only matches where opponent had a similar ELO rating (i.e. opponent ELO +-300)
    team_home_rel_matches = team_home_last_matches_rel[team_home_last_matches_rel['Similar Opponent'] == True]
    team_away_rel_matches = team_away_last_matches_rel[team_away_last_matches_rel['Similar Opponent'] == True]

    lambda_home = mean_goals(df=team_home_rel_matches , team = team_home)
    lambda_away = mean_goals(df=team_away_rel_matches , team = team_away)
    lambda_total = (lambda_home + lambda_away) / 2
    print('Mean number of goals with', team_home, 'involved:', lambda_home)
    print('Mean number of goals with', team_away, 'involved:', lambda_away)
    print('\n')

    # Calculate winning probability of home team
    win_prob = elo_prob(ELO_home, ELO_away, home_advantage)
    print(win_prob)

    # Incorporate the winning probability into the Poisson parameters for both teams
    lambda_home_elo = lambda_total * win_prob
    lambda_away_elo = lambda_total * (1-win_prob)

    # Simulate results and find the probabilities for each result
    df_results = result_probs(lambda_home_elo, lambda_away_elo)

    return df_results.sort_values(sort_col, ascending=False).head(n_output_rows).style.format({'Prob':'{:.2%}'})

def result_probs(lambda_home, lambda_away, n_sims:int=1000000):

    goals_home = np.random.poisson(lambda_home, n_sims)
    goals_away = np.random.poisson(lambda_away, n_sims)
    df = pd.DataFrame(np.hstack((goals_home[:,None], goals_away[:,None])), columns=["Goals Home", "Goals Away"])
    df_counts = df.value_counts(subset=["Goals Home", "Goals Away"], normalize=True)
    return df_counts.to_frame(name="Prob")


def simulate_elos(df, homeadv:int, divisor:int, base:int, K:float, due_date=str(date.today()), plot_linreg=False):


    Home_ELO_New = []
    Home_ELO_Old = []
    Away_ELO_New = []
    Away_ELO_Old = []
    team_dict = {}

    for index, row in df.iterrows():
      for team in [row["home_team"], row["away_team"]]:
          if team not in team_dict:
              team_dict[team] = Team(team)
      # Print Elo ratings before the update
      # print("Before update:", team_dict[row["home_team"]].elo, team_dict[row["away_team"]].elo)
      Home_ELO_Old.append(team_dict[row["home_team"]].elo)
      Away_ELO_Old.append(team_dict[row["away_team"]].elo)
      team_dict[row["home_team"]].update_elo(
          team_dict[row["away_team"]],
          row["home_score"],
          row["away_score"],
          row["neutral"],
          homeadv,
          divisor,
          base,
          K
      )
      Home_ELO_New.append(team_dict[row["home_team"]].elo)
      Away_ELO_New.append(team_dict[row["away_team"]].elo)

    Home_ELO_Old_series = pd.Series(Home_ELO_Old, index=df.index)
    Away_ELO_Old_series = pd.Series(Away_ELO_Old, index=df.index)
    Home_ELO_New_series = pd.Series(Home_ELO_New, index=df.index)
    Away_ELO_New_series = pd.Series(Away_ELO_New, index=df.index)

# Assign Series to DataFrame columns
    df["home_team_elo_old"] = Home_ELO_Old_series
    df["away_team_elo_old"] = Away_ELO_Old_series
    df["home_team_elo_new"] = Home_ELO_New_series
    df["away_team_elo_new"] = Away_ELO_New_series


    df_2000 = df[
        (df["date"] > "1999-12-31")
        & (df["date"] < due_date)
        & ((df["tournament"] == "UEFA Euro") | (df["tournament"] == "FIFA World Cup"))
    ].copy()
    df_2000["diff_score"] = df["home_score"] - df["away_score"]
    df_2000["diff_elo"] = df["home_team_elo_old"] - df["away_team_elo_old"]

    if plot_linreg:
        b, m = polyfit(df_2000["diff_elo"], df_2000["diff_score"], 1)
        fig, ax = plt.subplots(1, 1, figsize=(9, 6))
        plt.scatter(df_2000["diff_elo"], df_2000["diff_score"])
        plt.plot(range(-600, 600, 10), [b + m * x for x in range(-600, 600, 10)], )

        plt.title("The Relationship between Rating and Goal Differences")
        plt.xlabel("Elo Rating Difference")
        plt.ylabel("Goals Difference")

        plt.show()
        return None

    ratings = pd.DataFrame(
        {
```

```
        {
            "Team": list(team_dict.keys()),
            "Elo": [team.elo for team in team_dict.values()],
        }
    )
    ratings = ratings.sort_values(by="Elo", ascending=False).reset_index(drop=True)

    return ratings, df_2000
```

```
ratings , df_t = simulate_elos(df=relevant_results, divisor = 400,  base = 10,  K= 40,  homeadv = 50, plot_linreg=False)
```

```
<ipython-input-5-cff9330cb688>:178: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-cc
  df["home_team_elo_old"] = Home_ELO_Old_series
<ipython-input-5-cff9330cb688>:179: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-cc
  df["away_team_elo_old"] = Away_ELO_Old_series
<ipython-input-5-cff9330cb688>:180: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-cc
  df["home_team_elo_new"] = Home_ELO_New_series
<ipython-input-5-cff9330cb688>:181: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-cc
  df["away_team_elo_new"] = Away_ELO_New_series
```

```
euro_teams = [
    "Germany",
    "Belgium",
    "France",
    "Portugal",
    "Scotland",
    "Spain",
    "Turkey",
    "Czech Republic",
    "Austria",
    "England",
    "Hungary",
    "Albania",
    "Denmark",
    "Netherlands",
    "Romania",
    "Switzerland",
    "Serbia",
    "Italy",
    "Slovenia",
    "Slovakia",
    "Croatia",
    "Georgia",
    "Ukraine",
    "Poland",
]
for index , i in ratings.iterrows():
  if i['Team'] in euro_teams:
    print(i['Team'] , ' ' , i['Elo'])
```

```
France   1962.4592567511822
Spain   1943.0765796057538
Belgium   1930.2084862227564
England   1925.9825899549805
Portugal   1911.4963984265746
Italy   1908.4249171975289
Netherlands   1904.0818287463119
Croatia   1882.4377886187353
Germany   1859.8693762522457
```

```
Austria   1818.2936533844324
Ukraine   1812.1603254084512
Hungary   1807.4632606364232
Denmark   1795.6128686741224
Serbia   1777.4468613667725
Switzerland   1769.8937938572296
Czech Republic   1744.330665350831
Slovenia   1736.7994740179354
Turkey   1733.7522658214866
Poland   1719.282077514393
Scotland   1703.1118147049137
Georgia   1650.2075234432393
Romania   1642.2810435586036
Slovakia   1627.9761773445655
Albania   1576.4348724839401
```

```
from ipywidgets import interact

def predict_score(team_home, team_away, matchday):

    ratings, _ = simulate_elos(df=relevant_results, divisor = 400,  base = 10,  K= 40,  homeadv = 50, plot_linreg=False)
    display(result(team_home, team_away, elos=ratings, home_advantage=True, n_output_rows=10, n_games=12, sort_col="Prob", due_date=matchday
```

```
date_list = [str(x) for x in pd.date_range(start="2024-06-11",end="2024-07-11").date]
interact(predict_score, team_home=euro_teams, team_away=euro_teams, matchday=date_list);
```