model2022_final

December 6, 2022

1 initial setting

[1]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

[2]:
```
#import
import pandas as pd
import numpy as np
import tensorflow as tf
import warnings
warnings.filterwarnings("ignore")
from sklearn.model_selection import train_test_split
from sklearn.manifold import TSNE, Isomap
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler, Normalizer
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.stats import uniform, randint
import xgboost as xgb
from sklearn.metrics import auc, accuracy_score, mean_absolute_error
from sklearn.model_selection import cross_val_score, RandomizedSearchCV,␣
↪train_test_split, KFold
from sklearn.decomposition import PCA
from sklearn.multioutput import MultiOutputRegressor
import random
import copy
```

[3]:
```
%cd /content/drive/MyDrive/2022/BigData
file_root = './fifa_dataset_with_all_features-no_null_values.csv'
df_old =  pd.read_csv(file_root, encoding='cp949')
```

```
/content/drive/MyDrive/2022/BigData
```

2

[4]:
```
df = pd.read_csv(file_root, encoding='cp949')
#date
#df['date'] = [i.replace('-','') for i in df['date']]
#all_list : tournament,home_team, away_team
all_list = ((df['tournament'].unique()).tolist() + (df['home_team'].unique()).
 ↪tolist() + (df['away_team'].unique()).tolist())

uni_len = '***'.join(all_list)
nat_enc = tf.keras.preprocessing.text.Tokenizer(num_words=len(all_list),
    filters='',
    lower=False,
    split = '***')
nat_enc.fit_on_texts([uni_len])
nat_dic = nat_enc.word_index
```

[5]:
```
#  enumerate
for step, val in enumerate(df['tournament']):
    df['tournament'].iloc[step] = (nat_dic.get(val))
for step, val in enumerate(df['home_team']):
    df['home_team'].iloc[step] = (nat_dic.get(val))
for step, val in enumerate(df['away_team']):
    df['away_team'].iloc[step] = (nat_dic.get(val))
for step, val in enumerate(df['neutral']):
    if df['neutral'].iloc[step] == False:
        df['neutral'].iloc[step] = 0
    if df['neutral'].iloc[step] == True:
        df['neutral'].iloc[step] = 1
```

[6]:
```
df.head()
```

[6]:

	home_team	away_team	home_score	away_score	tournament	neutral	\
0	1	10	2	1	91	1	
1	2	58	1	0	91	1	
2	3	1	1	1	91	0	
3	4	25	0	3	92	1	
4	3	10	1	3	91	0	

	home_team_total_points	away_team_total_points	home_B1_overall	\
0	664.0	710.0	71	
1	849.0	421.0	72	
2	554.0	664.0	72	
3	558.0	461.0	74	
4	576.0	701.0	72	

	home_B2_overall	…	away_B2_Position_Best_Rating	\

```
0                72  …                           72
1                72  …                           59
2                72  …                           72
3                74  …                           75
4                72  …                           72

   away_B3_Position_Best_Rating  away_B4_Position_Best_Rating  \
0                            71                            69
1                            59                            57
2                            70                            70
3                            74                            73
4                            71                            69

   away_M1_Position_Best_Rating  away_M2_Position_Best_Rating  \
0                            78                            77
1                            72                            69
2                            77                            76
3                            79                            78
4                            78                            77

   away_M3_Position_Best_Rating  away_M4_Position_Best_Rating  \
0                            75                            75
1                            67                            63
2                            72                            71
3                            76                            75
4                            75                            75

   away_T1_Position_Best_Rating  away_T2_Position_Best_Rating  \
0                            80                            78
1                            73                            72
2                            76                            74
3                            87                            82
4                            80                            78

   away_K1_Position_Best_Rating
0                            80
1                            66
2                            70
3                            80
4                            80

[5 rows x 316 columns]
```

[7]:
```
#df_new
df_new = df
#len
temp_tor = []
```

```
for i in df_old['tournament']:
    temp_tor.append(len(i))
temp_away = []
for i in df_old['away_team']:
    temp_away.append(len(i))
temp_home = []
for i in df_old['home_team']:
    temp_home.append(len(i))
df_new['tor_len'] = temp_tor
df_new['home_len'] = temp_home
df_new['away_len'] = temp_away

#
df_new = df_new.dropna()

df_label1 = df_new['home_score']
df_label2 = df_new['away_score']
```

[8]:
```
#df_new, df_label   :    (df_new)   (df_label, home_score  away score)
df_new = df_new.drop(['home_score', 'away_score'], axis = 1)
#df_label
df_label = pd.concat([df_label1, df_label2],axis = 1)
df_label.columns = ['home_score', 'away_score']
```

df_label

[9]:
```
#      7948    ?    6547 rows      -> concat          index
df_label
```

[9]:

	home_score	away_score
0	2	1
1	1	0
2	1	1
3	0	3
4	1	3
…	…	…
2540	0	0
2541	0	5
2542	1	1
2543	0	0
2544	1	0

[2545 rows x 2 columns]

[10]:
```
x_train = df_new.to_numpy()
df_train = pd.DataFrame(data=x_train, columns=df_new.columns)
```

df_train

[11]:
```
df_train
```

[11]:

	home_team	away_team	tournament	neutral	home_team_total_points	\
0	1	10	91	1	664.0	
1	2	58	91	1	849.0	
2	3	1	91	0	554.0	
3	4	25	92	1	558.0	
4	3	10	91	0	576.0	
…	…	…	…	…	…	
2540	85	16	92	0	1243.8	
2541	82	27	92	0	961.23	
2542	4	34	92	0	1473.04	
2543	55	6	92	0	1425.59	
2544	56	73	92	0	1384.04	

	away_team_total_points	home_B1_overall	home_B2_overall	home_B3_overall	\
0	710.0	71	72	70	
1	421.0	72	72	71	
2	664.0	72	72	69	
3	461.0	74	74	72	
4	701.0	72	72	69	
…	…	…	…	…	
2540	1335.36	67	66	63	
2541	1434.68	59	56	55	
2542	1405.6	77	75	74	
2543	1509.61	75	73	74	
2544	1341.03	75	73	71	

	home_B4_overall	…	away_M1_Position_Best_Rating	\
0	70	…	78	
1	71	…	72	
2	70	…	77	
3	72	…	79	
4	70	…	78	
…	…	…	…	
2540	59	…	70	
2541	63	…	76	
2542	74	…	78	
2543	73	…	80	
2544	71	…	81	

	away_M2_Position_Best_Rating	away_M3_Position_Best_Rating	\
0	77	75	
1	69	67	
2	76	72	
3	78	76	
4	77	75	

```
…                              …                              …
2540                              69                             69
2541                              76                             75
2542                              74                             72
2543                              77                             76
2544                              74                             71

     away_M4_Position_Best_Rating away_T1_Position_Best_Rating  \
0                                75                             80
1                                63                             73
2                                71                             76
3                                75                             87
4                                75                             80
…                                …                              …
2540                             67                             72
2541                             75                             78
2542                             72                             75
2543                             76                             79
2544                             69                             77

     away_T2_Position_Best_Rating away_K1_Position_Best_Rating tor_len  \
0                                78                             80      28
1                                72                             66      28
2                                74                             70      28
3                                82                             80       8
4                                78                             80      28
…                                …                              …     …
2540                             67                             66       8
2541                             75                             76       8
2542                             73                             83       8
2543                             77                             78       8
2544                             74                             69       8

     home_len away_len
0           4        7
1           5        5
2          12        4
3           6        6
4          12        7
…       …       …
2540       10        8
2541        7        7
2542        6        7
2543        8        5
2544        8       10

[2545 rows x 317 columns]
```

[12]:
```
#x_train = df_new.to_numpy()
Norm = Normalizer() #            1
x_temp = Norm.fit_transform(x_train)
df_temp = pd.DataFrame(data=x_temp, columns=df_new.columns)
```

df_temp : df_train x_train , df_temp x_train normalize x_temp

[13]:
```
df_temp
```

[13]:

	home_team	away_team	tournament	neutral	home_team_total_points	\
0	0.000738	0.007380	0.067162	0.000738	0.490059	
1	0.001523	0.044177	0.069313	0.000762	0.646666	
2	0.002362	0.000787	0.071636	0.000000	0.436114	
3	0.003353	0.020959	0.077129	0.000838	0.467806	
4	0.002285	0.007618	0.069326	0.000000	0.438810	
…	…	…	…	…	…	
2540	0.042079	0.007921	0.045544	0.000000	0.615739	
2541	0.042346	0.013943	0.047510	0.000000	0.496393	
2542	0.001776	0.015093	0.040841	0.000000	0.653911	
2543	0.024010	0.002619	0.040163	0.000000	0.622346	
2544	0.026164	0.034107	0.042984	0.000000	0.646647	

	away_team_total_points	home_B1_overall	home_B2_overall	\
0	0.524008	0.052401	0.053139	
1	0.320667	0.054841	0.054841	
2	0.522706	0.056679	0.056679	
3	0.386485	0.062039	0.062039	
4	0.534037	0.054851	0.054851	
…	…	…	…	
2540	0.661065	0.033168	0.032673	
2541	0.740889	0.030468	0.028919	
2542	0.623973	0.034182	0.033294	
2543	0.659025	0.032741	0.031868	
2544	0.626552	0.035041	0.034107	

	home_B3_overall	home_B4_overall	…	away_M1_Position_Best_Rating	\
0	0.051663	0.051663	…	0.057567	
1	0.054079	0.054079	…	0.054841	
2	0.054317	0.055105	…	0.060615	
3	0.060362	0.060362	…	0.066231	
4	0.052566	0.053328	…	0.059422	
…	…	…	…	…	
2540	0.031188	0.029208	…	0.034653	
2541	0.028403	0.032534	…	0.039247	
2542	0.032850	0.032850	…	0.034626	
2543	0.032305	0.031868	…	0.034924	
2544	0.033172	0.033172	…	0.037845	

```
      away_M2_Position_Best_Rating  away_M3_Position_Best_Rating  \
0                         0.056829                      0.055353
1                         0.052556                      0.051033
2                         0.059828                      0.056679
3                         0.065392                      0.063716
4                         0.058660                      0.057137
…                            …                             …
2540                      0.034158                      0.034158
2541                      0.039247                      0.038731
2542                      0.032850                      0.031962
2543                      0.033615                      0.033178
2544                      0.034574                      0.033172

      away_M4_Position_Best_Rating  away_T1_Position_Best_Rating  \
0                         0.055353                      0.059043
1                         0.047986                      0.055603
2                         0.055892                      0.059828
3                         0.062877                      0.072938
4                         0.057137                      0.060946
…                            …                             …
2540                      0.033168                      0.035643
2541                      0.038731                      0.040280
2542                      0.031962                      0.033294
2543                      0.033178                      0.034488
2544                      0.032238                      0.035976

      away_T2_Position_Best_Rating  away_K1_Position_Best_Rating   tor_len  \
0                         0.057567                      0.059043  0.020665
1                         0.054841                      0.050271  0.021327
2                         0.058253                      0.055105  0.022042
3                         0.068746                      0.067069  0.006707
4                         0.059422                      0.060946  0.021331
…                            …                             …        …
2540                      0.033168                      0.032673  0.003960
2541                      0.038731                      0.039247  0.004131
2542                      0.032406                      0.036845  0.003551
2543                      0.033615                      0.034051  0.003492
2544                      0.034574                      0.032238  0.003738

      home_len  away_len
0     0.002952  0.005166
1     0.003808  0.003808
2     0.009447  0.003149
3     0.005030  0.005030
4     0.009142  0.005333
…        …         …
```

```
2540  0.004950  0.003960
2541  0.003615  0.003615
2542  0.002664  0.003107
2543  0.003492  0.002183
2544  0.003738  0.004672

[2545 rows x 317 columns]
```

[14]:
```
#df_corr = df_temp.corr() #
```

3

df_label df_temp(normalize)

[15]:
```
y_es = df_label.to_numpy()
x_es = df_temp.to_numpy()
x_t, x_val , y_t, y_val = train_test_split(x_es, y_es, random_state=28)
```

[16]:
```
params = {
    "estimator__colsample_bylevel": uniform(0.7, 0.3),
    "estimator__gamma": uniform(0, 0.5),
    "estimator__learning_rate": uniform(0.003, 0.3),
    "estimator__max_depth": randint(2, 6),
    "estimator__n_estimators": randint(100, 500),
    "estimator__subsample": uniform(0.6, 0.4)
}
```

: XGBRegressor

[17]:
```
#Multioutput regression are regression problems that involve predicting two or␣
 ↪more numerical values given an input example.
xgb_model = MultiOutputRegressor(xgb.XGBRegressor(objective='reg:squarederror',␣
 ↪eval_metric=['mae'], random_state = 1, use_label_encoder=False))
```

[18]:
```
#search :
search = RandomizedSearchCV(xgb_model, param_distributions=params, random_state␣
 ↪=1, n_iter=5, cv=3,
                            verbose=0, n_jobs=1, return_train_score=True)
search.fit(x_t, y_t)

best_param = search.best_params_
print('-')
print(search.best_score_)
print(best_param)
print('-')
```

```
-
0.08699591513110606
{'estimator__colsample_bylevel': 0.9147911547905092, 'estimator__gamma':
0.40137875196868245, 'estimator__learning_rate': 0.030840242592221373,
'estimator__max_depth': 3, 'estimator__n_estimators': 101,
'estimator__subsample': 0.9460081007915934}
-
```

[19]:
```
#RandomizedSearchCV    best param
xgb_model = MultiOutputRegressor(xgb.XGBRegressor(objective='reg:
 ↪squarederror',eval_metric=['mae'], random_state = 1,
                                use_label_encoder=False, **best_param)).fit(x_t,␣
 ↪y_t)
```

[20]:
```
#MAE : MSE    loss function
#     loss function         ?
print('MAE: ',mean_absolute_error(y_val, xgb_model.predict(x_val)))
```

```
MAE:  0.8795515228431303
```

[21]:
```
#K-fold    :
#       K      ,              .
#                             .
#K-fold                                  .
folds = KFold(n_splits = 5, shuffle = True, random_state = 100)
scores = cross_val_score(xgb_model, x_val, y_val,␣
 ↪scoring='neg_mean_absolute_error', cv=folds, verbose=0)
print('KFold_Neg_MAE: ',np.mean(scores))
```

```
KFold_Neg_MAE:  -0.9389189271674322
```

4 predict_score

[22]:
```
file_WC_team = './input_test_with_14_features.csv'
df_teams = pd.read_csv(file_WC_team, encoding='cp949')
```

[23]:
```
#inp_list:       home_team, away team
def predict_score(inp_list, autofill, Norm, seed, model):
    # ind_list = []
    # for i in inp_list:
    #     ind_list.append(nat_dic.get(i))
    # seed[:,1] = ind_list[0]
    # seed[:,2] = ind_list[1]
    x_test = []
    qatar_flag = False
    home_away_chagned_flag = False
```

```
home_team = inp_list[0]
away_team = inp_list[1]

# Netural == False
if (home_team == 'Qatar' or away_team == 'Qatar'):
    qatar_flag = True
    if(away_team == 'Qatar'):
        home_away_chagned_flag = True
        home_team, away_team = away_team, home_team

home_data = df_teams.loc[df_teams['team'] == home_team]
away_data = df_teams.loc[df_teams['team'] == away_team]


# Home_team norm
x_test.append(nat_dic.get(home_team))
# Away_team norm
x_test.append(nat_dic.get(away_team))

# Tournament norm
x_test.append(nat_dic.get('FIFA World Cup'))

# Neutral
if (qatar_flag):
    x_test.append(0)
else:
    x_test.append(1)

# Rank
x_test.append(float(home_data.iloc[0]['total_points']))
x_test.append(float(away_data.iloc[0]['total_points']))

#
feature_num = 14                    # TODO:   Feature
for i in range(0, feature_num):
  x_test += home_data.iloc[0][2+(i*11):13+(i*11)].to_list()
  x_test += away_data.iloc[0][2+(i*11):13+(i*11)].to_list()

# Tournament   len
x_test.append(len('FIFA World Cup'))
# Home_team   len
x_test.append(len(home_team))
# Away_team   len
x_test.append(len(away_team))
```

```
    # Norm
    x_test_pd = pd.DataFrame(data=np.array(x_test).reshape(-1, len(x_test)),␣
↪columns=df_new.columns)
    x_test_norm = Norm.fit_transform(x_test_pd)
    pred = model.predict(x_test_norm)

    output = np.ravel(pred).tolist()

    if (home_away_chagned_flag):
        output[0], output[1] = output[1], output[0]

    return np.ravel(pred).tolist()
```

5 def

[24]:
```
import math
```

[25]:
```
def create_params():
  seed = x_train[random.randint(0,x_train.shape[0])].reshape(1,-1)
  params = {
    'autofill': True,
    'Norm': Norm,
    'seed': seed,
    'model': xgb_model
  }
  return params
```

[26]:
```
def create_input(home_team, away_team):
  #   home_team, away_team     ?            ?
  ind_list = [home_team, away_team]
  return ind_list#input_vector
```

[27]:
```
def select_winning_team(ans, qual): #probability_array
    pred_ans = copy.deepcopy(ans)
    #  :    x,
    if qual == True :
      #
      pred_ans[0] = round(pred_ans[0])
      pred_ans[1] = round(pred_ans[1])

    #  #
    """
    else :

      pred_ans[0] = round(pred_ans[0], 1)
```

```
      pred_ans[1] = round(pred_ans[1], 1)
    """

    if (pred_ans[0]>pred_ans[1]):
          out=0 # home
    elif (pred_ans[0]<pred_ans[1]):
          out=1 # away
    elif (pred_ans[0]==pred_ans[1]):
          out=2 #

    return out, pred_ans
```

6

[28]:
```
#
Group_A= ["Netherlands","Senegal","Ecuador","Qatar"]
Group_B= ["England","United States","Iran","Wales"] # United States     ;;
Group_C= ["Argentina","Poland","Mexico", "Saudi Arabia"]
Group_D= ["France","Australia","Tunisia","Denmark"]
Group_E= ["Japan","Spain","Germany","Costa Rica"]
Group_F= ["Morocco","Croatia","Belgium","Canada"]
Group_G= ["Brazil","Switzerland","Cameroon","Serbia"]
Group_H= ["Portugal","South Korea","Uruguay","Ghana"] # South Korea     ;;
Groups={"Group A":Group_A,"Group B":Group_B,"Group C":Group_C,"Group D":
 ↪Group_D,"Group E":Group_E,"Group F":Group_F,"Group G":Group_G,"Group H":
 ↪Group_H}
```

7 32

[29]:
```
##Group stage Matches
print("========Qualifying Games=========")
Group_standings={}
game_id=0
game_win_predicted={}
game_win_predicted_team={}
for grp_name in list(Groups.keys()):
    print(f"[{grp_name} Matches]")
    #
    probable_countries=Groups[grp_name]

    team_wins_dct={}
    goal_scored_dct={}
    goal_against_dct={}
```

```
    win_dct={}
    draw_dct={}
    lost_dct={}

    for i in range(len(probable_countries)):
        #team i: team1, team j : team2
        j=i+1
        team_1=probable_countries[i]

        team_wins=0

        while j<len((probable_countries)):
            team_2=probable_countries[j]
            team_lst=[team_1,team_2]

            #   home_team, away_team     ?           ?
            input=create_input(team_1, team_2)#np.
↪array([[year,stadium_num,team_1_num,team_2_num]])
            params = create_params()
            #ans :    ex 1.3 : 1.5
            ans = predict_score(input, **params)

            #qual :    true,    false
            qual = True

            win, prob_lst=select_winning_team(ans, qual)
            #print("applied_score : ", prob_lst,", predicted_score: ", ans)

            #
            goal_scored_dct[team_1] = goal_scored_dct.get(team_1,0)+prob_lst[0]
            goal_scored_dct[team_2] = goal_scored_dct.get(team_2,0)+prob_lst[1]
            #
            goal_against_dct[team_1] = goal_against_dct.
↪get(team_1,0)+prob_lst[1]
            goal_against_dct[team_2] = goal_against_dct.
↪get(team_2,0)+prob_lst[0]

            game_id+=1

            try:
                print(game_id," [win]: [", win, "]")
                game_win_predicted[game_id]=win
                if(win==2):
                    game_win_predicted_team[game_id]='draw'
                else:
                    game_win_predicted_team[game_id]=team_lst[win]
                print(f" {team_1} vs {team_2} \n  {team_lst[win]} wins \n")
```

```
            # print(str(prob_lst[0]) + " : " + str(prob_lst[1]))
            # print()

            #team1
            if (win)==0:
                team_wins_dct[team_1] = team_wins_dct.get(team_1,0)+2
                team_wins_dct[team_2] = team_wins_dct.get(team_2,0)
                #team1 1
                win_dct[team_1] = win_dct.get(team_1,0)+1
                win_dct[team_2] = win_dct.get(team_2,0)
                #team2 1
                lost_dct[team_2] = lost_dct.get(team_2,0)+1
                lost_dct[team_1] = lost_dct.get(team_1,0)
                #
                draw_dct[team_2] = draw_dct.get(team_2,0)
                draw_dct[team_1] = draw_dct.get(team_1,0)

            elif (win)==1:
                team_wins_dct[team_2] = team_wins_dct.get(team_2,0)+2
                team_wins_dct[team_1] = team_wins_dct.get(team_1,0)
                #team2 1
                win_dct[team_2] = win_dct.get(team_2,0)+1
                win_dct[team_1] = win_dct.get(team_1,0)
                #team1 1
                lost_dct[team_1] = lost_dct.get(team_1,0)+1
                lost_dct[team_2] = lost_dct.get(team_2,0)
                #
                draw_dct[team_1] = draw_dct.get(team_1,0)
                draw_dct[team_2] = draw_dct.get(team_2,0)

        except IndexError:
            #
            print(f"{team_1} vs {team_2} \n  Match Draw \n")
            team_wins_dct[team_1] = team_wins_dct.get(team_1,0)+1
            team_wins_dct[team_2] = team_wins_dct.get(team_2,0)+1

            #
            draw_dct[team_1] = draw_dct.get(team_1,0)+1
            draw_dct[team_2] = draw_dct.get(team_2,0)+1
            #
            win_dct[team_1] = win_dct.get(team_1,0)
            lost_dct[team_1] = lost_dct.get(team_1,0)
            #
            win_dct[team_2] = win_dct.get(team_2,0)
            lost_dct[team_2] = lost_dct.get(team_2,0)
```

```
            j=j+1

    
↪group_results=[win_dct,draw_dct,lost_dct,team_wins_dct,goal_scored_dct,goal_against_dct]
    Group_standings[grp_name]=group_results
```

```
========Qualifying Games=========
[Group A Matches]
1  [win]: [ 0 ]
 Netherlands vs Senegal
  Netherlands wins

2  [win]: [ 0 ]
 Netherlands vs Ecuador
  Netherlands wins

3  [win]: [ 0 ]
 Netherlands vs Qatar
  Netherlands wins

4  [win]: [ 0 ]
 Senegal vs Ecuador
  Senegal wins

5  [win]: [ 0 ]
 Senegal vs Qatar
  Senegal wins

6  [win]: [ 0 ]
 Ecuador vs Qatar
  Ecuador wins

[Group B Matches]
7  [win]: [ 0 ]
 England vs United States
  England wins

8  [win]: [ 0 ]
 England vs Iran
  England wins

9  [win]: [ 0 ]
 England vs Wales
  England wins

10  [win]: [ 0 ]
 United States vs Iran
```

```
  United States wins

11  [win]: [ 0 ]
 United States vs Wales
  United States wins

12  [win]: [ 2 ]
Iran vs Wales
  Match Draw

[Group C Matches]
13  [win]: [ 0 ]
 Argentina vs Poland
  Argentina wins

14  [win]: [ 0 ]
 Argentina vs Mexico
  Argentina wins

15  [win]: [ 0 ]
 Argentina vs Saudi Arabia
  Argentina wins

16  [win]: [ 0 ]
 Poland vs Mexico
  Poland wins

17  [win]: [ 0 ]
 Poland vs Saudi Arabia
  Poland wins

18  [win]: [ 0 ]
 Mexico vs Saudi Arabia
  Mexico wins

[Group D Matches]
19  [win]: [ 0 ]
 France vs Australia
  France wins

20  [win]: [ 0 ]
 France vs Tunisia
  France wins

21  [win]: [ 0 ]
 France vs Denmark
  France wins
```

```
22  [win]: [ 0 ]
 Australia vs Tunisia
  Australia wins

23  [win]: [ 2 ]
Australia vs Denmark
  Match Draw

24  [win]: [ 0 ]
 Tunisia vs Denmark
  Tunisia wins

[Group E Matches]
25  [win]: [ 2 ]
Japan vs Spain
  Match Draw

26  [win]: [ 0 ]
 Japan vs Germany
  Japan wins

27  [win]: [ 0 ]
 Japan vs Costa Rica
  Japan wins

28  [win]: [ 0 ]
 Spain vs Germany
  Spain wins

29  [win]: [ 0 ]
 Spain vs Costa Rica
  Spain wins

30  [win]: [ 0 ]
 Germany vs Costa Rica
  Germany wins

[Group F Matches]
31  [win]: [ 2 ]
Morocco vs Croatia
  Match Draw

32  [win]: [ 2 ]
Morocco vs Belgium
  Match Draw

33  [win]: [ 0 ]
 Morocco vs Canada
```

```
  Morocco wins

34  [win]: [ 2 ]
Croatia vs Belgium
  Match Draw

35  [win]: [ 0 ]
 Croatia vs Canada
  Croatia wins

36  [win]: [ 0 ]
 Belgium vs Canada
  Belgium wins

[Group G Matches]
37  [win]: [ 0 ]
 Brazil vs Switzerland
  Brazil wins

38  [win]: [ 0 ]
 Brazil vs Cameroon
  Brazil wins

39  [win]: [ 0 ]
 Brazil vs Serbia
  Brazil wins

40  [win]: [ 0 ]
 Switzerland vs Cameroon
  Switzerland wins

41  [win]: [ 0 ]
 Switzerland vs Serbia
  Switzerland wins

42  [win]: [ 0 ]
 Cameroon vs Serbia
  Cameroon wins

[Group H Matches]
43  [win]: [ 0 ]
 Portugal vs South Korea
  Portugal wins

44  [win]: [ 0 ]
 Portugal vs Uruguay
  Portugal wins
```

```
45  [win]: [ 0 ]
 Portugal vs Ghana
  Portugal wins

46  [win]: [ 0 ]
 South Korea vs Uruguay
  South Korea wins

47  [win]: [ 0 ]
 South Korea vs Ghana
  South Korea wins

48  [win]: [ 0 ]
 Uruguay vs Ghana
  Uruguay wins
```

8 ACCURACY

[30]:
```
file_qualifying_label='./qualifying_label.csv'
df_qualifying_label = pd.read_csv(file_qualifying_label)

list_label_win=df_qualifying_label['label_win'].tolist()
game_win_label = {}
game_id_label=1
for i in range(48):
  game_win_label[game_id_label]=list_label_win[i]
  game_id_label +=1

#game_win_label
```

[31]:
```
game_win_label
```

[31]:
```
{1: 0,
 2: 2,
 3: 0,
 4: 0,
 5: 0,
 6: 0,
 7: 2,
 8: 0,
 9: 0,
 10: 0,
 11: 2,
 12: 0,
 13: 0,
```

```
 14: 0,
 15: 1,
 16: 2,
 17: 0,
 18: 0,
 19: 0,
 20: 1,
 21: 0,
 22: 0,
 23: 0,
 24: 2,
 25: 0,
 26: 0,
 27: 1,
 28: 2,
 29: 0,
 30: 0,
 31: 2,
 32: 0,
 33: 0,
 34: 2,
 35: 0,
 36: 0,
 37: 0,
 38: 1,
 39: 0,
 40: 0,
 41: 0,
 42: 2,
 43: 1,
 44: 0,
 45: 0,
 46: 2,
 47: 1,
 48: 0}
```

[32]:
```
for i in range(len(game_win_predicted)):
    df_qualifying_label['predicted_win'][i]=int(game_win_predicted[i+1])
    df_qualifying_label['predicted_win_team'][i]=game_win_predicted_team[i+1]
```

[33]:
```
num_correct=0
for i in range(len(game_win_predicted)):
  if(game_win_predicted[i+1] == game_win_label[i+1]):
    df_qualifying_label['accuracy'][i]=True
    num_correct+=1
  else:
    df_qualifying_label['accuracy'][i]=False
```

```
accuracy_rate = num_correct/len(game_win_predicted)*100
```

[34]:
```
df_qualifying_label
```

[34]:

	game_id	group	team_1	team_2	label_win	predicted_win
0	0	A	Netherlands	Senegal	0	0.0
1	1	A	Netherlands	Ecuador	2	0.0
2	2	A	Netherlands	Qatar	0	0.0
3	3	A	Senegal	Ecuador	0	0.0
4	4	A	Senegal	Qatar	0	0.0
5	5	A	Ecuador	Qatar	0	0.0
6	6	B	England	United States	2	0.0
7	7	B	England	Iran	0	0.0
8	8	B	England	Wales	0	0.0
9	9	B	United States	Iran	0	0.0
10	10	B	United States	Wales	2	0.0
11	11	B	Iran	Wales	0	2.0
12	12	C	Argentina	Poland	0	0.0
13	13	C	Argentina	Mexico	0	0.0
14	14	C	Argentina	Saudi Arabia	1	0.0
15	15	C	Poland	Mexico	2	0.0
16	16	C	Poland	Saudi Arabia	0	0.0
17	17	C	Mexico	Saudi Arabia	0	0.0
18	18	D	France	Australia	0	0.0
19	19	D	France	Tunisia	1	0.0
20	20	D	France	Denmark	0	0.0
21	21	D	Australia	Tunisia	0	0.0
22	22	D	Australia	Denmark	0	2.0
23	23	D	Tunisia	Denmark	2	0.0
24	24	E	Japan	Spain	0	2.0
25	25	E	Japan	Germany	0	0.0
26	26	E	Japan	Costa Rica	1	0.0
27	27	E	Spain	Germany	2	0.0
28	28	E	Spain	Costa Rica	0	0.0
29	29	E	Germany	Costa Rica	0	0.0
30	30	F	Morocco	Croatia	2	2.0
31	31	F	Morocco	Belgium	0	2.0
32	32	F	Morocco	Canada	0	0.0
33	33	F	Croatia	Belgium	2	2.0
34	34	F	Croatia	Canada	0	0.0
35	35	F	Belgium	Canada	0	0.0
36	36	G	Brazil	Switzerland	0	0.0
37	37	G	Brazil	Cameroon	1	0.0
38	38	G	Brazil	Serbia	0	0.0
39	39	G	Switzerland	Cameroon	0	0.0
40	40	G	Switzerland	Serbia	0	0.0

41	41	G	Cameroon	Serbia	2	0.0
42	42	H	Portugal	South Korea	1	0.0
43	43	H	Portugal	Uruguay	0	0.0
44	44	H	Portugal	Ghana	0	0.0
45	45	H	South Korea	Uruguay	2	0.0
46	46	H	South Korea	Ghana	1	0.0
47	47	H	Uruguay	Ghana	0	0.0

	accuracy	labe_win_team	predicted_win_team
0	True	Netherlands	Netherlands
1	False	Draw	Netherlands
2	True	Netherlands	Netherlands
3	True	Senegal	Senegal
4	True	Senegal	Senegal
5	True	Ecuador	Ecuador
6	False	Draw	England
7	True	England	England
8	True	England	England
9	True	United States	United States
10	False	Draw	United States
11	False	Iran	draw
12	True	Argentina	Argentina
13	True	Argentina	Argentina
14	False	Saudi Arabia	Argentina
15	False	Draw	Poland
16	True	Poland	Poland
17	True	Mexico	Mexico
18	True	France	France
19	False	Tunisia	France
20	True	France	France
21	True	Australia	Australia
22	False	Australia	draw
23	False	Draw	Tunisia
24	False	Japan	draw
25	True	Japan	Japan
26	False	Costa Rica	Japan
27	False	Draw	Spain
28	True	Spain	Spain
29	True	Germany	Germany
30	True	Draw	draw
31	False	Morocco	draw
32	True	Morocco	Morocco
33	True	Draw	draw
34	True	Croatia	Croatia
35	True	Belgium	Belgium
36	True	Brazil	Brazil
37	False	Cameroon	Brazil

38	True	Brazil	Brazil
39	True	Switzerland	Switzerland
40	True	Switzerland	Switzerland
41	False	Draw	Cameroon
42	False	South Korea	Portugal
43	True	Portugal	Portugal
44	True	Portugal	Portugal
45	False	Draw	South Korea
46	False	Ghana	South Korea
47	True	Uruguay	Uruguay

[35]:
```
accuracy_rate
```

[35]:
```
62.5
```

[36]:
```
df_qualifying_label
```

[36]:

	game_id	group	team_1	team_2	label_win	predicted_win	\
0	0	A	Netherlands	Senegal	0	0.0	
1	1	A	Netherlands	Ecuador	2	0.0	
2	2	A	Netherlands	Qatar	0	0.0	
3	3	A	Senegal	Ecuador	0	0.0	
4	4	A	Senegal	Qatar	0	0.0	
5	5	A	Ecuador	Qatar	0	0.0	
6	6	B	England	United States	2	0.0	
7	7	B	England	Iran	0	0.0	
8	8	B	England	Wales	0	0.0	
9	9	B	United States	Iran	0	0.0	
10	10	B	United States	Wales	2	0.0	
11	11	B	Iran	Wales	0	2.0	
12	12	C	Argentina	Poland	0	0.0	
13	13	C	Argentina	Mexico	0	0.0	
14	14	C	Argentina	Saudi Arabia	1	0.0	
15	15	C	Poland	Mexico	2	0.0	
16	16	C	Poland	Saudi Arabia	0	0.0	
17	17	C	Mexico	Saudi Arabia	0	0.0	
18	18	D	France	Australia	0	0.0	
19	19	D	France	Tunisia	1	0.0	
20	20	D	France	Denmark	0	0.0	
21	21	D	Australia	Tunisia	0	0.0	
22	22	D	Australia	Denmark	0	2.0	
23	23	D	Tunisia	Denmark	2	0.0	
24	24	E	Japan	Spain	0	2.0	
25	25	E	Japan	Germany	0	0.0	
26	26	E	Japan	Costa Rica	1	0.0	
27	27	E	Spain	Germany	2	0.0	
28	28	E	Spain	Costa Rica	0	0.0	

29	29	E	Germany	Costa Rica	0	0.0
30	30	F	Morocco	Croatia	2	2.0
31	31	F	Morocco	Belgium	0	2.0
32	32	F	Morocco	Canada	0	0.0
33	33	F	Croatia	Belgium	2	2.0
34	34	F	Croatia	Canada	0	0.0
35	35	F	Belgium	Canada	0	0.0
36	36	G	Brazil	Switzerland	0	0.0
37	37	G	Brazil	Cameroon	1	0.0
38	38	G	Brazil	Serbia	0	0.0
39	39	G	Switzerland	Cameroon	0	0.0
40	40	G	Switzerland	Serbia	0	0.0
41	41	G	Cameroon	Serbia	2	0.0
42	42	H	Portugal	South Korea	1	0.0
43	43	H	Portugal	Uruguay	0	0.0
44	44	H	Portugal	Ghana	0	0.0
45	45	H	South Korea	Uruguay	2	0.0
46	46	H	South Korea	Ghana	1	0.0
47	47	H	Uruguay	Ghana	0	0.0

	accuracy	labe_win_team	predicted_win_team
0	True	Netherlands	Netherlands
1	False	Draw	Netherlands
2	True	Netherlands	Netherlands
3	True	Senegal	Senegal
4	True	Senegal	Senegal
5	True	Ecuador	Ecuador
6	False	Draw	England
7	True	England	England
8	True	England	England
9	True	United States	United States
10	False	Draw	United States
11	False	Iran	draw
12	True	Argentina	Argentina
13	True	Argentina	Argentina
14	False	Saudi Arabia	Argentina
15	False	Draw	Poland
16	True	Poland	Poland
17	True	Mexico	Mexico
18	True	France	France
19	False	Tunisia	France
20	True	France	France
21	True	Australia	Australia
22	False	Australia	draw
23	False	Draw	Tunisia
24	False	Japan	draw
25	True	Japan	Japan

```
26    False     Costa Rica              Japan
27    False           Draw              Spain
28     True          Spain              Spain
29     True        Germany            Germany
30     True           Draw               draw
31    False        Morocco               draw
32     True        Morocco            Morocco
33     True           Draw               draw
34     True        Croatia            Croatia
35     True        Belgium            Belgium
36     True         Brazil             Brazil
37    False       Cameroon             Brazil
38     True         Brazil             Brazil
39     True    Switzerland        Switzerland
40     True    Switzerland        Switzerland
41    False           Draw           Cameroon
42    False    South Korea           Portugal
43     True       Portugal           Portugal
44     True       Portugal           Portugal
45    False           Draw        South Korea
46    False          Ghana        South Korea
47     True        Uruguay            Uruguay
```

[37]:
```python
#game_win_label  game_win_predicted
num_correct=0
correct_game_id=[]
num_wrong=0
wrong_game_id=[]
for i in range(1, len(game_win_label)+1):
  if(game_win_label[i] == game_win_predicted[i]):
    num_correct+=1
    correct_game_id.append(i)
  else:
    num_wrong+=1
    wrong_game_id.append(i)
```

9 16

[45]:
```python
print("[PREDICTION]")

for grp_name in list(Group_standings.keys()):

    team_wins_dct= dict(sorted(Group_standings[grp_name][3].items()))
    goal_scored_dct=dict(sorted(Group_standings[grp_name][4].items()))
    goal_against_dct=dict(sorted(Group_standings[grp_name][5].items()))
```

```
    win_dct=dict(sorted(Group_standings[grp_name][0].items()))
    draw_dct=dict(sorted(Group_standings[grp_name][1].items()))
    lost_dct=dict(sorted(Group_standings[grp_name][2].items()))

    lst_teams=list(team_wins_dct.keys())

    win_lst=list(win_dct.values())
    draw_lst=list(draw_dct.values())
    lost_lst=list(lost_dct.values())

    lst_win_count=list(team_wins_dct.values())
    goal_scored=list(goal_scored_dct.values())
    goal_against=list(goal_against_dct.values())
    goal_differance=[goal_scored[i]-goal_against[i] for i in range␣
↪(len(goal_scored))]
    ranking_table=pd.
↪DataFrame(list(zip(lst_teams,win_lst,draw_lst,lost_lst,goal_scored,goal_against,goal_differ
↪Scored","Goal Against","Goal Differance","Points"])
    ranking_table=ranking_table.sort_values("Points",ascending=False).
↪reset_index(drop=True)
    ranking_table.index = ranking_table.index + 1
    print(f"\n\n{grp_name} Final Rankings")
    print(ranking_table.to_markdown())
```

[PREDICTION]

```
Group A Final Rankings
|    | Team        |   Wins |   Draw |   Lost |   Goal Scored |   Goal Against |
Goal Differance |   Points |
|---:|:------------|-------:|-------:|-------:|--------------:|---------------:|
-----------------:|---------:|
|  1 | Netherlands |      3 |      0 |      0 |            12 |              7 |
5 |        6 |
|  2 | Senegal     |      2 |      0 |      1 |            10 |             10 |
0 |        4 |
|  3 | Ecuador     |      1 |      0 |      2 |             9 |             11 |
-2 |        2 |
|  4 | Qatar       |      0 |      0 |      3 |             9 |             12 |
-3 |        0 |


Group B Final Rankings
|    | Team          |   Wins |   Draw |   Lost |   Goal Scored |   Goal Against
|   Goal Differance |   Points |
|---:|:--------------|-------:|-------:|-------:|--------------:|---------------
```

```
:|------------------:|---------:|
|  1 | England       |      3 |      0 |      0 |            12 |              6
|                 6 |        6 |
|  2 | United States |      2 |      0 |      1 |             8 |              8
|                 0 |        4 |
|  3 | Iran          |      0 |      1 |      2 |             7 |             10
|                -3 |        1 |
|  4 | Wales         |      0 |      1 |      2 |             7 |             10
|                -3 |        1 |
```

Group C Final Rankings

```
|    | Team         |   Wins |   Draw |   Lost |   Goal Scored |   Goal Against
|   Goal Differance |   Points |
|---:|:-------------|-------:|-------:|-------:|--------------:|--------------:
|------------------:|---------:|
|  1 | Argentina    |      3 |      0 |      0 |            12 |              6
|                 6 |        6 |
|  2 | Poland       |      2 |      0 |      1 |            10 |              8
|                 2 |        4 |
|  3 | Mexico       |      1 |      0 |      2 |             8 |             10
|                -2 |        2 |
|  4 | Saudi Arabia |      0 |      0 |      3 |             6 |             12
|                -6 |        0 |
```

Group D Final Rankings

```
|    | Team      |   Wins |   Draw |   Lost |   Goal Scored |   Goal Against |
Goal Differance |   Points |
|---:|:----------|-------:|-------:|-------:|--------------:|---------------:|--
----------------:|---------:|
|  1 | France    |      3 |      0 |      0 |            13 |              7 |
6 |        6 |
|  2 | Australia |      1 |      1 |      1 |             9 |             11 |
-2 |        3 |
|  3 | Tunisia   |      1 |      0 |      2 |             8 |             10 |
-2 |        2 |
|  4 | Denmark   |      0 |      1 |      2 |            10 |             12 |
-2 |        1 |
```

Group E Final Rankings

```
|    | Team       |   Wins |   Draw |   Lost |   Goal Scored |   Goal Against |
Goal Differance |   Points |
|---:|:-----------|-------:|-------:|-------:|--------------:|---------------:|-
-----------------:|---------:|
|  1 | Japan      |      2 |      1 |      0 |            11 |              8 |
3 |        5 |
```

2	Spain	2	1	0	11	7	4	5
3	Germany	1	0	2	9	10	-1	2
4	Costa Rica	0	0	3	6	12	-6	0

Group F Final Rankings

	Team	Wins	Draw	Lost	Goal Scored	Goal Against	Goal Difference	Points
1	Belgium	1	2	0	10	8	2	4
2	Croatia	1	2	0	10	8	2	4
3	Morocco	1	2	0	9	8	1	4
4	Canada	0	0	3	6	11	-5	0

Group G Final Rankings

	Team	Wins	Draw	Lost	Goal Scored	Goal Against	Goal Difference	Points
1	Brazil	3	0	0	12	7	5	6
2	Switzerland	2	0	1	10	8	2	4
3	Cameroon	1	0	2	9	10	-1	2
4	Serbia	0	0	3	6	12	-6	0

Group H Final Rankings

	Team	Wins	Draw	Lost	Goal Scored	Goal Against	Goal Difference	Points
1	Portugal	3	0	0	10	6	4	6
2	South Korea	2	0	1	10	8	2	4
3	Uruguay	1	0	2	9	9		

```
0 |        2 |
|  4 | Ghana       |      0 |      0 |      3 |             6 |             12 |
-6 |        0 |
```

10 16

[46]:
```
##Round of 16 Section_1

qualified_teams_1=[]
standings=list(Group_standings.keys())
i=0
print(f"Round of 16\n")
while i < (len(standings)):
    A_team= sorted(Group_standings[standings[i]][3].items(), key=lambda x:␣
 ↪x[1], reverse=True)
    team_1=A_team[0][0]
    B_team= sorted(Group_standings[standings[i+1]][3].items(), key=lambda x:␣
 ↪x[1], reverse=True)
    team_2=B_team[1][0]

    #team_1_num=label_encoder.transform([team_1])[0]
    #team_2_num=label_encoder.transform([team_2])[0]
    team_lst=[team_1,team_2]

    #Input_vector=np.array([[year,host_num,team_1_num,team_2_num]])
    input=create_input(team_1, team_2)
    params = create_params()
    #res=model.predict(Input_vector)
    ans = predict_score(input, **params)
    #win,_=select_winning_team(res)
    qual = False
    win, prob_lst=select_winning_team(ans, qual)

    try:
            #print("applied_score : ", prob_lst,", predicted_score: ", ans)
            print(f"{team_1} vs {team_2} \n  {team_lst[win]} wins \n")

            # print(str(prob_lst[0]) + "  :  " + str(prob_lst[1]))
            # print()

            #print(f"    {team_lst[win]} into the Quater-Finals  \n")
            qualified_teams_1.append(team_lst[win])
    except IndexError:
            print(f"{team_1} vs {team_2} \n  [ERROR]Match Draw ")
            """
            winning_team=random.choice(team_lst)
```

```
            print(f"   {winning_team} wins at Penaly Shoot-Out ")
            print(f"    {winning_team} into the Quater-Finals  \n")
            qualified_teams_1.append(winning_team)
            """
    i=i+2

##Round of 16 Section_2
qualified_teams_2=[]
standings=list(Group_standings.keys())
i=0
while i < (len(standings)):
    A_team= sorted(Group_standings[standings[i]][3].items(), key=lambda x:␣
 ↪x[1], reverse=True)
    team_1=A_team[1][0]
    B_team= sorted(Group_standings[standings[i+1]][3].items(), key=lambda x:␣
 ↪x[1], reverse=True)
    team_2=B_team[0][0]

    #team_1_num=label_encoder.transform([team_1])[0]
    #team_2_num=label_encoder.transform([team_2])[0]
    team_lst=[team_1,team_2]

    #Input_vector=np.array([[year,host_num,team_1_num,team_2_num]])
    input=create_input(team_1, team_2)
    params = create_params()
    #res=model.predict(Input_vector)
    ans = predict_score(input, **params)
    #win,_=select_winning_team(res)
    qual = False
    win, prob_lst=select_winning_team(ans, qual)

    try:
            #print("applied_score : ", prob_lst,", predicted_score: ", ans)
            print(f"{team_1} vs {team_2} \n  {team_lst[win]} wins \n")
            # print(str(prob_lst[0]) + "  :  " + str(prob_lst[1]))
            # print()
            #print(f"    {team_lst[win]} into the Quater-Finals  \n")
            qualified_teams_2.append(team_lst[win])

    except IndexError:
            print(f"{team_1} vs {team_2} \n  [ERROR]Match Draw ")
            """
            winning_team=random.choice(team_lst)
            print(f"   {winning_team} wins at Penaly Shoot-Out ")
            #print(f"    {winning_team} into the Quater-Finals  \n")
            qualified_teams_2.append(winning_team)
            """
```

```
    i=i+2
```

```
Round of 16

Netherlands vs United States
  Netherlands wins

Argentina vs Australia
  Argentina wins

Japan vs Croatia
  Japan wins

Brazil vs South Korea
  Brazil wins

Senegal vs England
  Senegal wins

Poland vs France
  Poland wins

Spain vs Morocco
  Spain wins

Switzerland vs Portugal
  Switzerland wins
```

11 8

[47]:
```
#Quarter Finals

Semifinal_teams=[]
i=0
print(f"Quater Final Matches\n")
while i < (len(qualified_teams_1))-1:
    team_1= qualified_teams_1[i]
    team_2= qualified_teams_1[i+1]

    #team_1_num=label_encoder.transform([team_1])[0]
    #team_2_num=label_encoder.transform([team_2])[0]
    team_lst=[team_1,team_2]

    #Input_vector=np.array([[year,host_num,team_1_num,team_2_num]])
    input=create_input(team_1, team_2)
```

```
    params = create_params()
    #res=model.predict(Input_vector)
    ans = predict_score(input, **params)
    #win,_=select_winning_team(res)
    qual = False
    win, prob_lst=select_winning_team(ans, qual)

    try:
            #print("applied_score : ", prob_lst,", predicted_score: ", ans)
            print(f"{team_1} vs {team_2} \n  {team_lst[win]} wins \n")
            #print(f"    {team_lst[win]} into the Semi-Finals \n")
            Semifinal_teams.append(team_lst[win])

    except IndexError:
            print(f"{team_1} vs {team_2} \n  [ERROR]Match Draw")
            """
            winning_team=random.choice(team_lst)
            print(f"   {winning_team} wins at Penaly Shoot-Out  ")
            print(f"    {winning_team} into the Semi-Finals   \n")
            Semifinal_teams.append(winning_team)
            """
    i=i+2

i=0
while i < (len(qualified_teams_2))-1:
    team_1= qualified_teams_2[i]
    team_2= qualified_teams_2[i+1]
    #team_1_num=label_encoder.transform([team_1])[0]
    #team_2_num=label_encoder.transform([team_2])[0]
    team_lst=[team_1,team_2]

    #Input_vector=np.array([[year,host_num,team_1_num,team_2_num]])
    input=create_input(team_1, team_2)
    params = create_params()
    #res=model.predict(Input_vector)
    ans = predict_score(input, **params)
    #win,_=select_winning_team(res)
    qual = False
    win, prob_lst=select_winning_team(ans, qual)

    try:
            #print("applied_score : ", prob_lst,", predicted_score: ", ans)
            print(f"{team_1} vs {team_2} \n  {team_lst[win]} wins \n")
            #print(f"    {team_lst[win]} into the Semi-Finals  \n")
            Semifinal_teams.append(team_lst[win])

    except IndexError:
```

```
            print(f"{team_1} vs {team_2} \n  [ERROR]Match Draw ")
            """
            winning_team=random.choice(team_lst)
            print(f"   {winning_team} wins at Penaly Shoot-Out  ")
            print(f"    {winning_team} into the Semi-Finals   \n")
            Semifinal_teams.append(winning_team)
            """
    i=i+2
```

```
Quater Final Matches

Netherlands vs Argentina
  Netherlands wins

Japan vs Brazil
  Japan wins

Senegal vs Poland
  Senegal wins

Spain vs Switzerland
  Spain wins
```

12 4

[48]:
```
final_teams=[]
third_place_match_teams=[]
i=0
print(f"Semi Final Matches\n")
while i < (len(Semifinal_teams))-1:
    team_1= Semifinal_teams[i]
    team_2= Semifinal_teams[i+1]

    #team_1_num=label_encoder.transform([team_1])[0]
    #team_2_num=label_encoder.transform([team_2])[0]
    team_lst=[team_1,team_2]

    #Input_vector=np.array([[year,host_num,team_1_num,team_2_num]])
    input=create_input(team_1, team_2)
    params = create_params()
    #res=model.predict(Input_vector)
    ans = predict_score(input, **params)
    #win,_=select_winning_team(res)
    win, prob_lst=select_winning_team(ans, qual)
```

```
    try:
            #print("applied_score : ", prob_lst,", predicted_score: ", ans)
            print(f"{team_1} vs {team_2} \n  {team_lst[win]} wins \n")
            #print(f"    {team_lst[win]} into the FiIFA-Finals   \n")
            final_teams.append(team_lst[win])
            third_place_match_teams.append(team_lst[(win+1)%2])


    except IndexError:
            print(f"{team_1} vs {team_2} \n  [ERROR] Match Draw ")
            """
            winning_team=random.choice(team_lst)
            print(f"   {winning_team} wins at Penaly Shoot-Out  ")
            print(f"    {winning_team} into the FIFA-Finals   \n")
            final_teams.append(winning_team)
            team_lst.remove(winning_team)
            third_place_match_teams.append(team_lst[0])
            """
    i=i+2
```

```
Semi Final Matches

Netherlands vs Japan
  Netherlands wins

Senegal vs Spain
  Senegal wins
```

13

[50]:
```
#Finals and Third Place match

print(f"FIFA FINAl\n")
team_1= final_teams[1]
team_2= final_teams[0]

#team_1_num=label_encoder.transform([team_1])[0]
#team_2_num=label_encoder.transform([team_2])[0]
team_lst=[team_1,team_2]

#Input_vector=np.array([[year,host_num,team_1_num,team_2_num]])
input=create_input(team_1, team_2)
params = create_params()
#res=model.predict(Input_vector)
ans = predict_score(input, **params)
```

```
#win,_=select_winning_team(res)
win, prob_lst=select_winning_team(ans, qual)

try:
    #print("applied_score : ", prob_lst,", predicted_score: ", ans)
    if (ans[0] > ans[1]):
        print("[predicted_score] ", round(ans[0]), ":", round(ans[1])-1)
    else:
        print("[predicted_score] ", round(ans[0]-1), ":", round(ans[1]))
    print(f"{team_1} vs {team_2} \n  {team_lst[win]} are the Winners \n\n")
    winner=team_lst[win]
    place_2=team_lst[(win+1)%2]

except IndexError:
    print(f"{team_1} vs {team_2} \n  [ERROR]Match Draw ")
    """
    winning_team=random.choice(team_lst)
    print(f"   {winning_team} wins at Penaly Shoot-Out  ")
    print(f"    {winning_team} are the Winners   \n\n")
    winner=winning_team

    team_lst.remove(winning_team)
    place_2=team_lst[0]
    """

print(f"Third Place match\n")
team_1= third_place_match_teams[1]
team_2= third_place_match_teams[0]

#team_1_num=label_encoder.transform([team_1])[0]
#team_2_num=label_encoder.transform([team_2])[0]
team_lst=[team_1,team_2]

#Input_vector=np.array([[year,host_num,team_1_num,team_2_num]])
input=create_input(team_1, team_2)
params = create_params()
#res=model.predict(Input_vector)
ans = predict_score(input, **params)
#win,_=select_winning_team(res)
win, prob_lst=select_winning_team(ans, qual)

try:
    print(f"{team_1} vs {team_2} \n  {team_lst[win]} Wins the 3rd Place \n")
    place_3=team_lst[win]

except IndexError:
    print(f"{team_1} vs {team_2} \n  [ERROR]Match Draw ")
```

```
    """
    winning_team=random.choice(team_lst)
    print(f"   {winning_team} wins at Penaly Shoot-Out   ")
    print(f"    {winning_team} Wins the 3rd Place   \n")
    place_3=winning_team
    """



print(f"\n\nWinner is {winner} ")
print(f"Runner-up is {place_2} ")
print(f"3rd Place is {place_3} ")
```

```
FIFA FINAl

[predicted_score]  3 : 2
Senegal vs Netherlands
  Senegal are the Winners


Third Place match

Spain vs Japan
  Spain Wins the 3rd Place



Winner is Senegal
Runner-up is Netherlands
3rd Place is Spain
```

[43]:
```
!apt-get update
!apt-get install texlive texlive-xetex texlive-latex-extra pandoc
!pip install pypandoc
```

```
Ign:1 https://developer.download.nvidia.com/compute/machine-
learning/repos/ubuntu1804/x86_64  InRelease
Hit:2 https://developer.download.nvidia.com/compute/cuda/repos/ubuntu1804/x86_64
InRelease
Hit:3 https://cloud.r-project.org/bin/linux/ubuntu bionic-cran40/ InRelease
Hit:4 https://developer.download.nvidia.com/compute/machine-
learning/repos/ubuntu1804/x86_64  Release
Get:5 http://security.ubuntu.com/ubuntu bionic-security InRelease [88.7 kB]
Hit:6 http://ppa.launchpad.net/c2d4u.team/c2d4u4.0+/ubuntu bionic InRelease
Hit:7 http://archive.ubuntu.com/ubuntu bionic InRelease
Hit:9 http://archive.ubuntu.com/ubuntu bionic-updates InRelease
Hit:10 http://ppa.launchpad.net/cran/libgit2/ubuntu bionic InRelease
```

```
Get:11 http://archive.ubuntu.com/ubuntu bionic-backports InRelease [83.3 kB]
Hit:12 http://ppa.launchpad.net/deadsnakes/ppa/ubuntu bionic InRelease
Hit:13 http://ppa.launchpad.net/graphics-drivers/ppa/ubuntu bionic InRelease
Fetched 172 kB in 2s (88.3 kB/s)
Reading package lists… Done
Reading package lists… Done
Building dependency tree
Reading state information… Done
pandoc is already the newest version (1.19.2.4~dfsg-1build4).
texlive is already the newest version (2017.20180305-1).
texlive-latex-extra is already the newest version (2017.20180305-2).
texlive-xetex is already the newest version (2017.20180305-1).
The following package was automatically installed and is no longer required:
  libnvidia-common-460
Use 'apt autoremove' to remove it.
0 upgraded, 0 newly installed, 0 to remove and 19 not upgraded.
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
Requirement already satisfied: pypandoc in /usr/local/lib/python3.8/dist-
packages (1.10)
```

[44]:
```
!jupyter nbconvert --to PDF '/content/drive/MyDrive/2022/BigData/Code/
↪model2022_final.ipynb'
```

```
[NbConvertApp] Converting notebook
/content/drive/MyDrive/2022/BigData/Code/model2022_final.ipynb to PDF
[NbConvertApp] Writing 127387 bytes to ./notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', './notebook.tex', '-quiet']
[NbConvertApp] Running bibtex 1 time: ['bibtex', './notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely because there were no
citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 100654 bytes to
/content/drive/MyDrive/2022/BigData/Code/model2022_final.pdf
```

[44]:
```

```