In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np
import seaborn as sns
from scipy.stats import poisson
import statsmodels.api as sm
import statsmodels.formula.api as smf
import itertools
from IPython.display import display, HTML
```

In [2]:
```
matches = pd.read_csv('../input/international-football-results-from-1872-to-2017/results.csv')
matches.head()
```

Out[2]:

	date	home_team	away_team	home_score	away_score	tournament	city	count
0	1872-11-30	Scotland	England	0.0	0.0	Friendly	Glasgow	Scotla
1	1873-03-08	England	Scotland	4.0	2.0	Friendly	London	Engla
2	1874-03-07	Scotland	England	2.0	1.0	Friendly	Glasgow	Scotla
3	1875-03-06	England	Scotland	2.0	2.0	Friendly	London	Engla
4	1876-03-04	Scotland	England	3.0	0.0	Friendly	Glasgow	Scotla

In [3]:
```
matches = matches.astype({'date':'datetime64[ns]'})
```

In [4]:
```
home = matches[['home_team', 'home_score']].rename(columns={'home_team':'team', 'home_score':'score'})
away = matches[['away_team', 'away_score']].rename(columns={'away_team':'team', 'away_score':'score'})
# merge it into one
team_score = home.append(away).reset_index(drop=True)
# make an aggregation of the the score column group by the team
country_info = team_score.groupby('team')['score'].agg(['sum','count','mean']).reset_index()
country_info = country_info.rename(columns={'sum':'nb_goals', 'count':'nb_matches', 'mean':'goal_avg'})

del home, away
```

In [5]:
```
means = matches[['home_score','away_score']].mean()
means
```

Out[5]:
```
home_score    1.744297
away_score    1.186175
dtype: float64
```

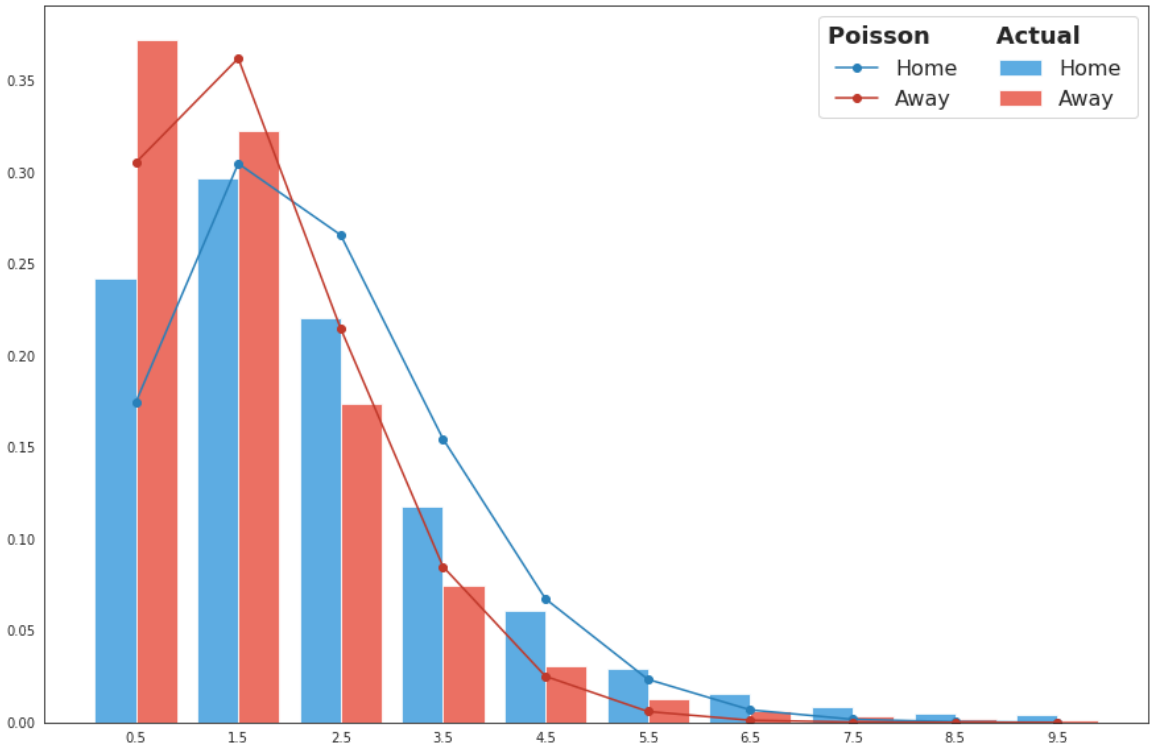
In [6]:

```
plt.figure(figsize=(15,10))
sns.set_style("white")
# construct Poisson  for each mean goals value
poisson_pred = np.column_stack([[poisson.pmf(k, means[j]) for k in rang
e(10)] for j in range(2)])
# plot histogram of actual goals
plt.hist(matches[['home_score', 'away_score']].values, range(11), alpha
=0.8,
         label=['Home', 'Away'],density=True, color=["#3498db", "#e74c3
c"])

# add lines for the Poisson distributions
pois1, = plt.plot([i-0.5 for i in range(1,11)], poisson_pred[:,0],
                  linestyle='-', marker='o',label="Home", color = '#298
0b9')
pois2, = plt.plot([i-0.5 for i in range(1,11)], poisson_pred[:,1],
                  linestyle='-', marker='o',label="Away", color = '#c03
92b')

leg=plt.legend(loc='upper right', fontsize=16, ncol=2)
leg.set_title("Poisson        Actual      ", prop = {'size':'18', 'weig
ht':'bold'})

plt.xticks([i-0.5 for i in range(1,11)],[i for i in range(11)])
plt.xlabel("Goals per Match",size=18)
plt.ylabel("Proportion of Matches",size=18)
plt.title("Number of Goals per Match",size=20,fontweight='bold')
plt.show()
```

```
---------------------------------------------------------------------
------
ValueError                                Traceback (most recent call
last)
<ipython-input-6-a083fce4aa0a> in <module>
     16 leg.set_title("Poisson        Actual      ", prop = {'size':'
18', 'weight':'bold'})
     17
---> 18 plt.xticks([i-0.5 for i in range(1,11)],[i for i in range(1
1)])
     19 plt.xlabel("Goals per Match",size=18)
     20 plt.ylabel("Proportion of Matches",size=18)

/opt/conda/lib/python3.7/site-packages/matplotlib/pyplot.py in xticks
(ticks, labels, **kwargs)
   1781         labels = ax.get_xticklabels()
   1782     else:
-> 1783         labels = ax.set_xticklabels(labels, **kwargs)
   1784     for l in labels:
   1785         l.update(kwargs)

/opt/conda/lib/python3.7/site-packages/matplotlib/axes/_base.py in wr
apper(self, *args, **kwargs)
     71
     72         def wrapper(self, *args, **kwargs):
---> 73             return get_method(self)(*args, **kwargs)
     74
     75         wrapper.__module__ = owner.__module__

/opt/conda/lib/python3.7/site-packages/matplotlib/_api/deprecation.py
in wrapper(*args, **kwargs)
    469                 "parameter will become keyword-only %(remova
l)s.",
    470                 name=name, obj_type=f"parameter of {func.__na
me__}()")
--> 471         return func(*args, **kwargs)
    472
    473     return wrapper

/opt/conda/lib/python3.7/site-packages/matplotlib/axis.py in _set_tic
klabels(self, labels, fontdict, minor, **kwargs)
   1777         if fontdict is not None:
   1778             kwargs.update(fontdict)
-> 1779         return self.set_ticklabels(labels, minor=minor, **kwa
rgs)
   1780
   1781     def set_ticks(self, ticks, *, minor=False):

/opt/conda/lib/python3.7/site-packages/matplotlib/axis.py in set_tick
labels(self, ticklabels, minor, **kwargs)
   1699             if len(locator.locs) != len(ticklabels) and len(t
icklabels) != 0:
   1700                 raise ValueError(
-> 1701                     "The number of FixedLocator locations"
   1702                     f" ({len(locator.locs)}), usually from a
call to"
```

```
1703                    " set_ticks, does not match"
```

```
ValueError: The number of FixedLocator locations (10), usually from a
call to set_ticks, does not match the number of ticklabels (11).
```



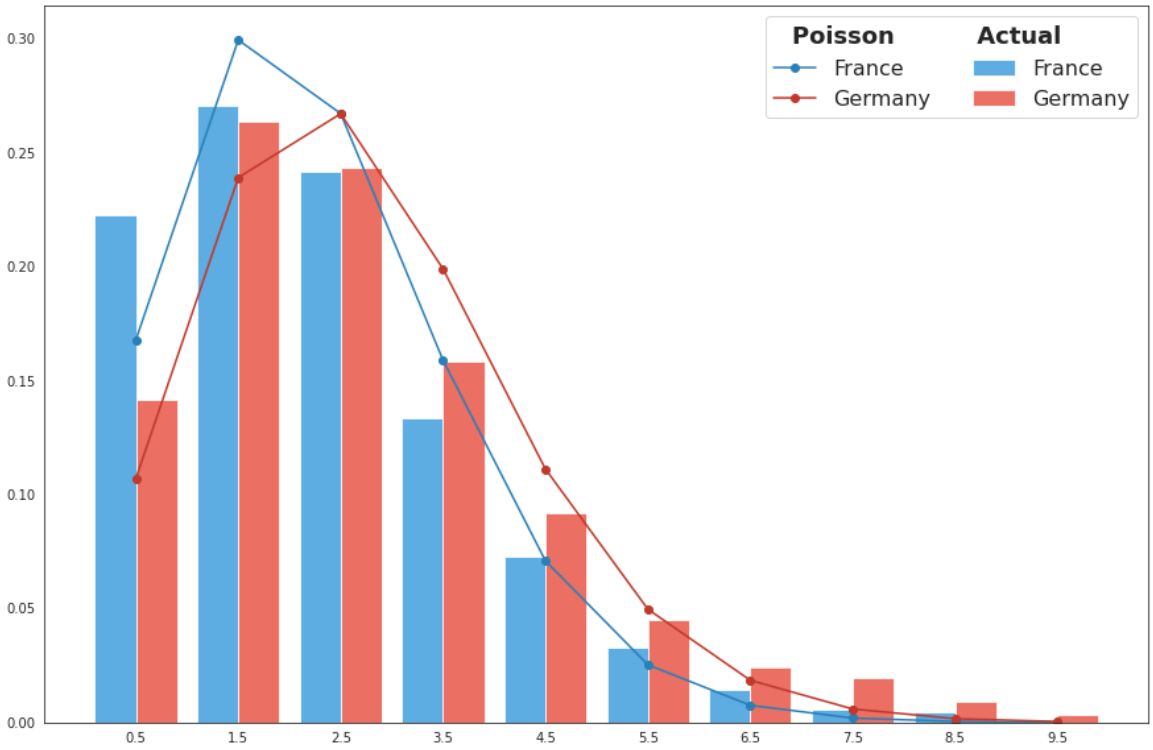
In [7]:

```
plt.figure(figsize=(15,10))
sns.set_style("white")
team1, team2 = "France", "Germany"
matches_t1 = team_score[team_score['team'] == team1]
matches_t2 = team_score[team_score['team'] == team2]

mean_t1 = matches_t1['score'].mean()
mean_t2 = matches_t2['score'].mean()

# construct Poisson  for each mean goals value
poisson_pred_t1 = [poisson.pmf(k, mean_t1) for k in range(10)]
poisson_pred_t2 = [poisson.pmf(k, mean_t2) for k in range(10)]

# plot histogram of actual goals
plt.hist([matches_t1['score'].values, matches_t2['score'].values], rang
e(11), alpha=0.8,
         label=[team1, team2],density=True, color=["#3498db", "#e74c3
c"])

# add lines for the Poisson distributions
pois1, = plt.plot([i-0.5 for i in range(1,11)], poisson_pred_t1,
                  linestyle='-', marker='o',label=team1, color = '#2980
b9')
pois2, = plt.plot([i-0.5 for i in range(1,11)], poisson_pred_t2,
                  linestyle='-', marker='o',label=team2, color = '#c039
2b')

leg=plt.legend(loc='upper right', fontsize=16, ncol=2)
leg.set_title("Poisson          Actual      ", prop = {'size':'18', 'we
ight':'bold'})

plt.xticks([i-0.5 for i in range(1,11)],[i for i in range(11)])
plt.xlabel("Goals per Match",size=18)
plt.ylabel("Proportion of Matches",size=18)
plt.title("Number of Goals per Match",size=20,fontweight='bold')
plt.show()
```

```
---------------------------------------------------------------------
------
ValueError                                Traceback (most recent call
last)
<ipython-input-7-4bd6e1a46d5b> in <module>
     25 leg.set_title("Poisson          Actual      ", prop = {'size
':'18', 'weight':'bold'})
     26
---> 27 plt.xticks([i-0.5 for i in range(1,11)],[i for i in range(1
1)])
     28 plt.xlabel("Goals per Match",size=18)
     29 plt.ylabel("Proportion of Matches",size=18)

/opt/conda/lib/python3.7/site-packages/matplotlib/pyplot.py in xticks
(ticks, labels, **kwargs)
   1781         labels = ax.get_xticklabels()
   1782     else:
-> 1783         labels = ax.set_xticklabels(labels, **kwargs)
   1784     for l in labels:
   1785         l.update(kwargs)

/opt/conda/lib/python3.7/site-packages/matplotlib/axes/_base.py in wr
apper(self, *args, **kwargs)
     71
     72         def wrapper(self, *args, **kwargs):
---> 73             return get_method(self)(*args, **kwargs)
     74
     75         wrapper.__module__ = owner.__module__

/opt/conda/lib/python3.7/site-packages/matplotlib/_api/deprecation.py
in wrapper(*args, **kwargs)
    469                 "parameter will become keyword-only %(remova
l)s.",
    470                 name=name, obj_type=f"parameter of {func.__na
me__}()")
--> 471         return func(*args, **kwargs)
    472
    473     return wrapper

/opt/conda/lib/python3.7/site-packages/matplotlib/axis.py in _set_tic
klabels(self, labels, fontdict, minor, **kwargs)
   1777         if fontdict is not None:
   1778             kwargs.update(fontdict)
-> 1779         return self.set_ticklabels(labels, minor=minor, **kwa
rgs)
   1780
   1781     def set_ticks(self, ticks, *, minor=False):

/opt/conda/lib/python3.7/site-packages/matplotlib/axis.py in set_tick
labels(self, ticklabels, minor, **kwargs)
   1699             if len(locator.locs) != len(ticklabels) and len(t
icklabels) != 0:
   1700                 raise ValueError(
-> 1701                     "The number of FixedLocator locations"
   1702                     f" ({len(locator.locs)}), usually from a
call to"
```

```
1703                        " set_ticks, does not match"
```

```
ValueError: The number of FixedLocator locations (10), usually from a
call to set_ticks, does not match the number of ticklabels (11).
```



In [8]:
```
def weight_from_tournament(tournament):
    if 'Cup' in tournament or 'Euro' in tournament:
        return 1
    else :
        return 100

# Create weight column based on the date
matches.loc[:,'weight'] = matches['tournament'].apply(weight_from_tourn
ament)
matches.loc[:,'weight'] = 1 / ((2022 - matches['date'].dt.year.astype('
int64'))*matches['weight'])

# Create model data
matches_model_data = pd.concat([matches[['home_team','away_team','home_
score','weight']].rename(
            columns={'home_team':'team', 'away_team':'opponent','home_s
core':'goals'}),
          matches[['away_team','home_team','away_score','weight']].ren
ame(
            columns={'away_team':'team', 'home_team':'opponent','away_s
core':'goals'})])
```

In [9]:
```
poisson_model = smf.glm(formula="goals ~ team + opponent", data=matches
_model_data,
                        family=sm.families.Poisson(), freq_weights=matc
hes_model_data['weight'].values).fit()
```

In [10]:

```
def get_proba_match(foot_model, team1, team2, max_goals=10):
    # Get the average goal for each team
    t1_goals_avg = foot_model.predict(pd.DataFrame(data={'team': team1,
'opponent': team2}, index=[1])).values[0]
    t2_goals_avg = foot_model.predict(pd.DataFrame(data={'team': team2,
'opponent': team1}, index=[1])).values[0]

    # Get probability of all possible score for each team
    team_pred = [[poisson.pmf(i, team_avg) for i in range(0, max_goals+
1)] for team_avg in [t1_goals_avg, t2_goals_avg]]

    # Do the product of the 2 vectors to get the matrix of the match
    match = np.outer(np.array(team_pred[0]), np.array(team_pred[1]))

    # Get the proba for each possible outcome
    t1_wins = np.sum(np.tril(match, -1))
    draw = np.sum(np.diag(match))
    t2_wins = np.sum(np.triu(match, 1))
    result_proba = [t1_wins, draw, t2_wins]

    # Adjust the proba to sum to one
    result_proba =  np.array(result_proba)/ np.array(result_proba).sum
(axis=0,keepdims=1)
    team_pred[0] = np.array(team_pred[0])/np.array(team_pred[0]).sum(ax
is=0,keepdims=1)
    team_pred[1] = np.array(team_pred[1])/np.array(team_pred[1]).sum(ax
is=0,keepdims=1)
    return result_proba, [np.array(team_pred[0]), np.array(team_pred
[1])]
```

In [11]:
```
def get_match_result(foot_model, team1, team2, elimination=False, max_draw=50, max_goals=10):
    # Get the proba
    proba, score_proba = get_proba_match(foot_model, team1, team2, max_goals)

    # Get the result, if it's an elimination game we have to be sure the result is not draw
    results = pd.Series([np.random.choice([team1, 'draw', team2], p=proba) for i in range(0,max_draw)]).value_counts()
    result = results.index[0] if not elimination or (elimination and results.index[0] != 'draw') else results.index[1]

    # If the result is not a draw game then we calculate the score of the winner from 1 to the max_goals
    # and the score of the looser from 0 to the score of the winner
    if (result != 'draw'):
        i_win, i_loose = (0,1) if result == team1 else (1,0)
        score_proba[i_win] = score_proba[i_win][1:]/score_proba[i_win][1:].sum(axis=0,keepdims=1)
        winner_score = pd.Series([np.random.choice(range(1, max_goals+1), p=score_proba[i_win]) for i in range(0,max_draw)]).value_counts().index[0]
        score_proba[i_loose] = score_proba[i_loose][:winner_score]/score_proba[i_loose][:winner_score].sum(axis=0,keepdims=1)
        looser_score = pd.Series([np.random.choice(range(0, winner_score), p=score_proba[i_loose]) for i in range(0,max_draw)]).value_counts().index[0]
        score = [winner_score, looser_score]
    # If it's a draw then we calculate a score and repeat it twice
    else:
        score = np.repeat(pd.Series([np.random.choice(range(0, max_goals+1), p=score_proba[0]) for i in range(0,max_draw)]).value_counts().index[0],2)
    looser = team2 if result == team1 else team1 if result != 'draw' else 'draw'
    return result, looser, score
```

In [12]:
```
print(get_match_result(poisson_model, 'Gabon', 'Togo'))
print(get_match_result(poisson_model, 'France', 'Togo', elimination=True))
print(get_match_result(poisson_model, 'Argentina', 'Germany'))
print(get_match_result(poisson_model, 'England', 'Morocco'))
print(get_match_result(poisson_model, 'Iran', 'Japan'))
```

```
('Gabon', 'Togo', [1, 0])
('France', 'Togo', [2, 0])
('Argentina', 'Germany', [1, 0])
('England', 'Morocco', [1, 0])
('draw', 'draw', array([1, 1]))
```

In [13]:
```
groupA = ['Argentina', 'Bolivia','Chile','Paraguay', 'Uruguay']
groupB = ['Brazil', 'Colombia', 'Ecuador', 'Peru','Venezuela']
groups = [groupA, groupB]
```

In [14]:
```
def get_group_result(foot_model, group):
    ranking = pd.DataFrame({'points':[0,0,0,0,0], 'diff':[0,0,0,0,0], '
goals':[0,0,0,0,0]}, index=group)
    for team1, team2 in itertools.combinations(group, 2):
        result, looser, score = get_match_result(foot_model, team1, tea
m2)
        #print(result, '-', looser,':', score)
        if result == 'draw':
            ranking.loc[[team1, team2], 'points'] += 1
            ranking.loc[[team1, team2], 'goals'] += score[0]
        else:
            ranking.loc[result, 'points'] += 3
            ranking.loc[result, 'goals'] += score[0]
            ranking.loc[looser, 'goals'] += score[1]
            ranking.loc[result, 'diff'] += score[0]-score[1]
            ranking.loc[looser, 'diff'] -= score[0]-score[1]

    return ranking.sort_values(by=['points','diff','goals'], ascending=
False)
```

In [15]:
```
groups_ranking = []
for group in groups:
    groups_ranking.append(get_group_result(poisson_model, group))
```

In [16]:
```
for group_rank in groups_ranking:
    display(group_rank)
```

	points	diff	goals
Argentina	12	5	6
Chile	6	0	3
Uruguay	6	0	2
Paraguay	4	-1	2
Bolivia	1	-4	1

	points	diff	goals
Brazil	12	9	9
Colombia	9	3	4
Ecuador	6	-2	2
Peru	3	-4	1
Venezuela	0	-6	0

In [17]:
```
def get_final_result(foot_model, groups_result):
    quarter_finals = []
    semi_finals = []

    # SIMULATE QUATER FINALS
    quarter_finals.append(get_match_result(foot_model, groups_result
[0].index[0], groups_result[1].index[3], elimination=True))
    quarter_finals.append(get_match_result(foot_model, groups_result
[0].index[1], groups_result[1].index[2], elimination=True))
    quarter_finals.append(get_match_result(foot_model, groups_result
[0].index[2], groups_result[1].index[1], elimination=True))
    quarter_finals.append(get_match_result(foot_model, groups_result
[0].index[3], groups_result[1].index[0], elimination=True))

    # SIMULATE SEMI FINALS
    semi_finals.append(get_match_result(foot_model, quarter_finals
[0][0], quarter_finals[2][0], elimination=True))
    semi_finals.append(get_match_result(foot_model, quarter_finals
[1][0], quarter_finals[3][0], elimination=True))

    # SIMULATE 3RD PLACE MATCH
    little_final = get_match_result(foot_model, semi_finals[0][1], semi
_finals[1][1], elimination=True)

    # SIMULATE FINAL
    final = get_match_result(foot_model, semi_finals[0][0], semi_finals
[1][0], elimination=True)

    return quarter_finals, semi_finals, little_final, final
```

In [18]:
```
quarter_finals, semi_finals, little_final, final = get_final_result(poi
sson_model, groups_ranking)
```

In [19]:
```
quarter_finals
```

Out[19]:
```
[('Argentina', 'Peru', [1, 0]),
 ('Chile', 'Ecuador', [1, 0]),
 ('Colombia', 'Uruguay', [1, 0]),
 ('Brazil', 'Paraguay', [2, 0])]
```

In [20]:
```
semi_finals
```

Out[20]:
```
[('Argentina', 'Colombia', [1, 0]), ('Brazil', 'Chile', [1, 0])]
```

In [21]:
```
little_final
```

Out[21]:
```
('Chile', 'Colombia', [1, 0])
```

In [22]:
```
final
```

Out[22]:
```
('Brazil', 'Argentina', [1, 0])
```