In [1]:
```
import os

import numpy as np
import pandas as pd

from scipy.stats import poisson, skellam, kstest, chisquare
from scipy.optimize import minimize
import scipy.stats

import plotly.express as px
import plotly.graph_objects as go
import plotly.offline as pyo
pyo.init_notebook_mode()

import plotly.io as pio
pio.renderers.default = "notebook+pdf"

import matplotlib.pyplot as plt
import seaborn as sns
```

CUSTOM FUNCTIONS

In [4]:
```
def test_poisson_dist(df, home_col = 'home_score', away_col='away_score', sa
    print(f"\nTest {sample} sample for Poisson distribution")

    # VIZUAL ANALYSIS
    # Define parameters
    high_percentile = 0.99
    max_home_score = df[home_col].quantile(high_percentile)
    max_away_score = df[away_col].quantile(high_percentile)
    poisson_range = int(np.ceil(max(max_home_score, max_away_score))) + 1  #

    # Calculate mean values for home and away scores
    mean_home_score = df[home_col].mean()
    mean_away_score = df[away_col].mean()

    # Construct poisson distribution
    poisson_home = [poisson.pmf(i, mean_home_score) for i in range(poisson_r
    poisson_away = [poisson.pmf(i, mean_away_score) for i in range(poisson_r

    # Make histogram
    hist_home = np.histogram(df[home_col], bins=range(poisson_range + 1), de
    hist_away = np.histogram(df[away_col], bins=range(poisson_range + 1), de

    # Plot
    fig = go.Figure()

    # home score hist
    fig.add_trace(go.Bar(
        x=hist_home[1][:-1],
        y=hist_home[0],
        name='Actual Home Scores',
```

```
        marker_color='#FFA07A',
        opacity=0.7
    ))

    # away score hist
    fig.add_trace(go.Bar(
        x=hist_away[1][:-1],
        y=hist_away[0],
        name='Actual Away Scores',
        marker_color='#20B2AA',
        opacity=0.7
    ))

    # Poisson dist for home score
    fig.add_trace(go.Scatter(
        x=list(range(poisson_range)),
        y=poisson_home,
        mode='lines+markers',
        name='Poisson Home',
        line=dict(color='#CD5C5C'),
        marker=dict(symbol='circle')
    ))

    # Poisson dist for away score
    fig.add_trace(go.Scatter(
        x=list(range(poisson_range)),
        y=poisson_away,
        mode='lines+markers',
        name='Poisson Away',
        line=dict(color='#006400'),
        marker=dict(symbol='circle')
    ))

    # Adjust plot appearance
    fig.update_layout(
        title="Scores Distribution",
        xaxis_title="Scores",
        yaxis_title="Proportion of Matches",
        barmode='overlay',
        legend=dict(title="Legend", font=dict(size=12)),
        template="plotly_white"
    )

    return fig

def remove_outliers(df, column, q1 = 0.25, q3 = 0.75):
    print(f'Outliers detecting for {column}...')

    fig, ax = plt.subplots()

    sns.boxplot(x=column, data=df, ax=ax)

    # IQR outliers detecting
    Q1 = df[column].quantile(q1)
    Q3 = df[column].quantile(q3)
    IQR = Q3 - Q1
```

```
    filtered_df = df[~((df[column] < (Q1 - 1.5 * IQR)) | (df[column] > (Q3 +
    outliers = df[((df[column] < (Q1 - 1.5 * IQR)) | (df[column] > (Q3 + 1.5

#     df.loc[(df[column] < (Q1 - 1.5 * IQR)), column] = int(round(Q1 - 1.5 *
#     df.loc[(df[column] > (Q3 + 1.5 * IQR)), column] = int(round(Q3 + 1.5 *

    sns.stripplot(x=column, data=outliers, color='red', ax=ax)

    plt.title(f'{column} Boxplot with Outliers Highlighted')
    plt.xlabel(column)
    plt.show()

    return filtered_df  # df
```

INPUT DATA

The raw data includes 3 sets on different pairs of teams. To analyze the data, let's combine the sets into one dataframe by setting a new attribute as a pair identifier col_name = 'pair'.

Moreover, let's implement 4 basic asserts for the correctness of the initial data check:

- Presence of columns in the datasets necessary for the script playback
- Scores data integrity
- Presence of obvious unmasked omissions in the data
- Absence of errors in the data, in particular negative score values.

If necessary, the set of input data asserts can be changed and expanded based on business requirements, this set includes only basic checks to demonstrate this possibility.

In [5]:
```
ROOT_PATH = '/Users/olesyamba/PycharmProjects/betby_test_task_ds'
DATA_FOLDER = os.path.join(ROOT_PATH, 'data')
```

In [6]:
```
files = ("test_data_first_pair.csv", "test_data_second_pair.csv", "test_data

file_paths = {
    f"{i}_pair": os.path.join(DATA_FOLDER, file) for i, file in enumerate(fi
}
```

In [7]:
```
data_frames = {name: pd.read_csv(path).assign(pair=i+1) for i, (name, path)
full_data = pd.concat(data_frames.values(), ignore_index=True)
del DATA_FOLDER, files, file_paths, data_frames
```

In [8]:
```
assert 'home_score' in full_data.columns and 'away_score' in full_data.colum
assert (full_data.dtypes[['home_score', 'away_score']] == 'int64').all(), "I
assert full_data[['home_score', 'away_score']].isnull().sum().sum() == 0, "E
assert (full_data['home_score'] >= 0).all() and (full_data['away_score'] >=
```

In [9]:
```
full_data.head(5)
```

Out[9]:

	event_id	home_score	away_score	pair
0	1	3	2	1
1	2	1	1	1
2	3	3	2	1
3	4	1	0	1
4	5	1	1	1

EDA

First of all, let's examine the dataset we're working with. Let's display the number of columns and rows, the total information, the number of empty and unique values by column, the descriptive statistics of the total set and the set for each pair, check for duplicates, and create two variables total_score and score_difference to look at the data from a different perspective.

In [10]:
```
full_data.shape
```

Out[10]:
```
(1650, 4)
```

In [11]:
```
full_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1650 entries, 0 to 1649
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   event_id    1650 non-null   int64
 1   home_score  1650 non-null   int64
 2   away_score  1650 non-null   int64
 3   pair        1650 non-null   int64
dtypes: int64(4)
memory usage: 51.7 KB
```

In [12]:
```
full_data.isnull().sum()
```

Out[12]:
```
event_id      0
home_score    0
away_score    0
pair          0
dtype: int64
```

In [13]:
```
full_data.nunique().sum()
```

Out[13]:
```
568
```

In [14]:
```
full_data[['home_score', 'away_score']].describe()
```

Out[14]:

	home_score	away_score
count	1650.000000	1650.000000
mean	2.093333	1.269697
std	1.279504	1.065184
min	0.000000	0.000000
25%	1.000000	0.000000
50%	2.000000	1.000000
75%	3.000000	2.000000
max	7.000000	6.000000

In [15]:
```
full_data.groupby('pair')[['home_score', 'away_score']].describe()
```

Out[15]:

	home_score								away_score		
	count	**mean**	**std**	**min**	**25%**	**50%**	**75%**	**max**	**count**	**mean**	
pair											
1	550.0	2.154545	1.309809	0.0	1.0	2.0	3.0	6.0	550.0	1.723636	1.134
2	550.0	1.783636	1.218597	0.0	1.0	2.0	3.0	7.0	550.0	1.334545	0.988
3	550.0	2.341818	1.247028	0.0	1.0	2.0	3.0	7.0	550.0	0.750909	0.815

In [16]:
```
full_data.duplicated().sum()
```

Out[16]:
```
0
```

In [17]:
```
full_data['total_score'] = full_data['home_score'] + full_data['away_score']
full_data['score_difference'] = full_data['home_score'] - full_data['away_sc
```

The data is loaded with the correct types, the full set contains 1650 records. The data has no skips or duplicates, which makes our work much easier and looks more like a sweet dream than like a real data :)

Descriptive statistics for the whole set show an excess of average home scores over average away scores. The standard deviation on away scores is smaller in absolute terms than the standard deviation on home scores, but when evaluated as a percentage of the mean, it suggests that away teams play more unpredictably. Let's take this into the account, but let's not jump to conclusions, actually because we don't know anything about the distribution yet.

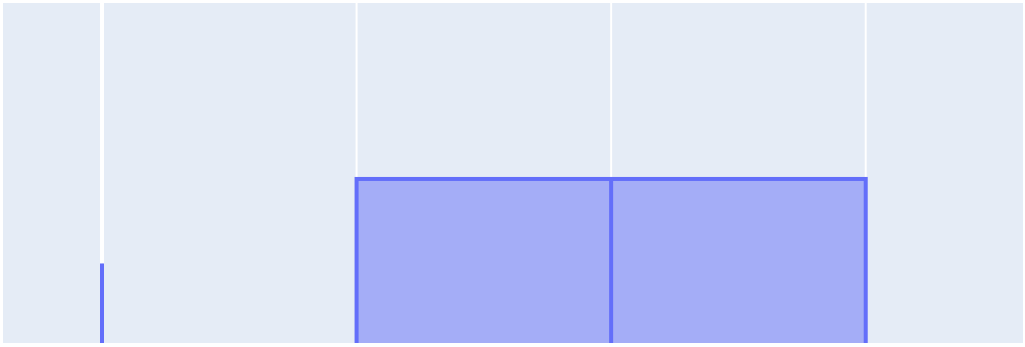
The descriptive statistics grouped by pairs show that there are significant differences in the stats between the different pairs. However, the general hypotheses identified at the

stage of analyzing the overall descriptive statistics are supported.

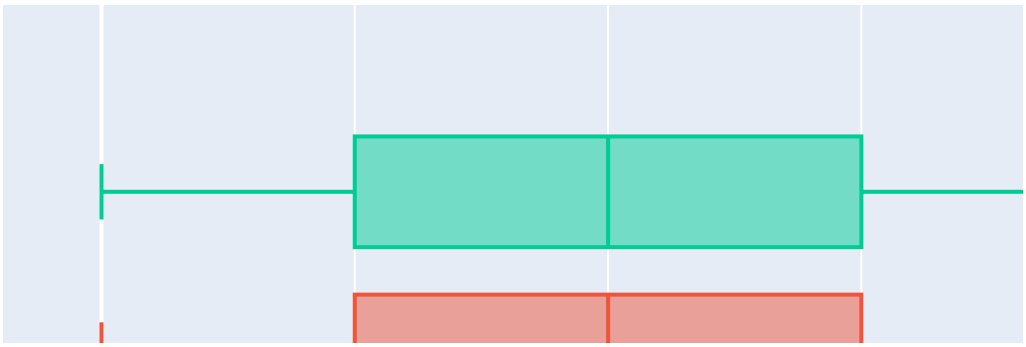
PLOTS, PLOTS and PLOTS

In [18]:
```
# BOX PLOTS
for col in full_data.columns:
    if col != 'event_id' and col != 'pair':
        fig = go.Figure(px.box(data_frame = full_data,
                               x = col,
                               title=f"Boxplot | {col} | full sample"
            ))
        fig.show()

        fig = go.Figure(px.box(data_frame = full_data,
                               x = col,
                               color='pair',
                               title=f"Boxplot | {col} | colored by pair"
            ))
        fig.show()

del col, fig
```

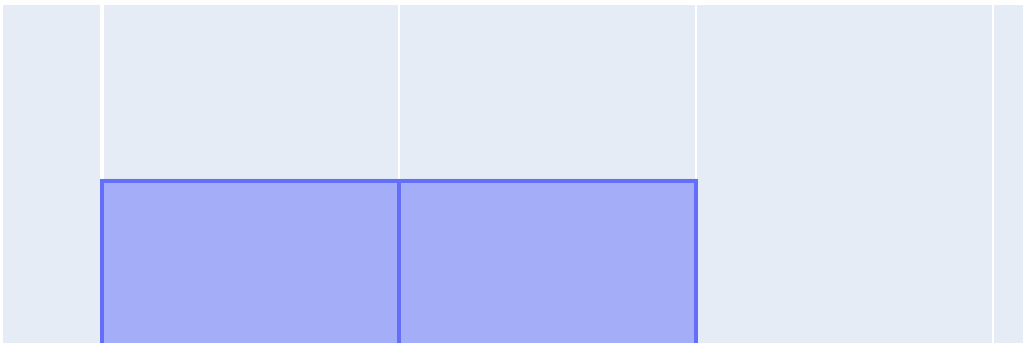
Boxplot | home_score | full sample



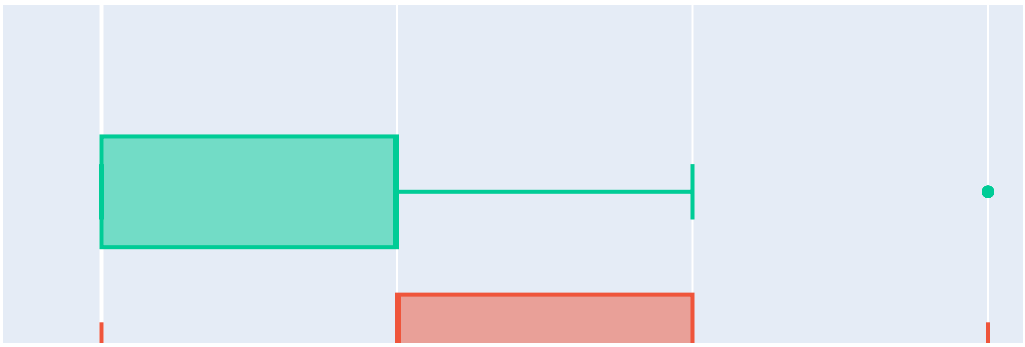
Boxplot | home_score | colored by pair



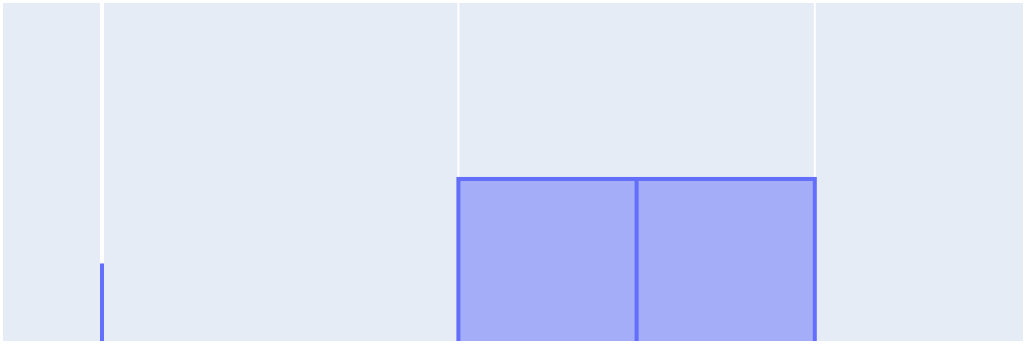
Boxplot | away_score | full sample



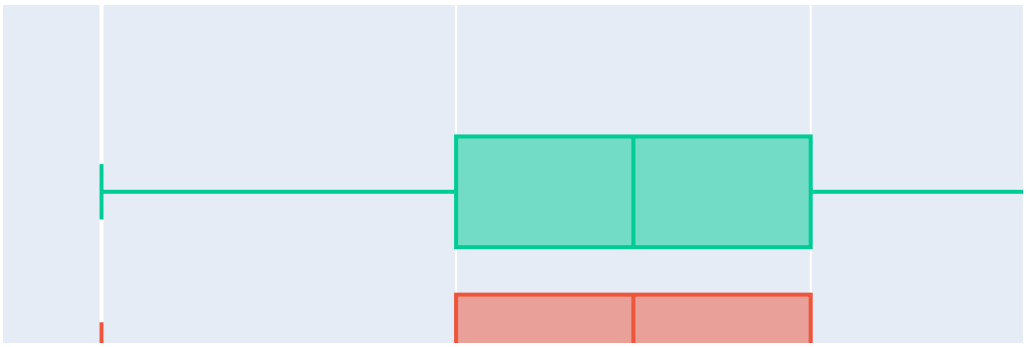
Boxplot | away_score | colored by pair



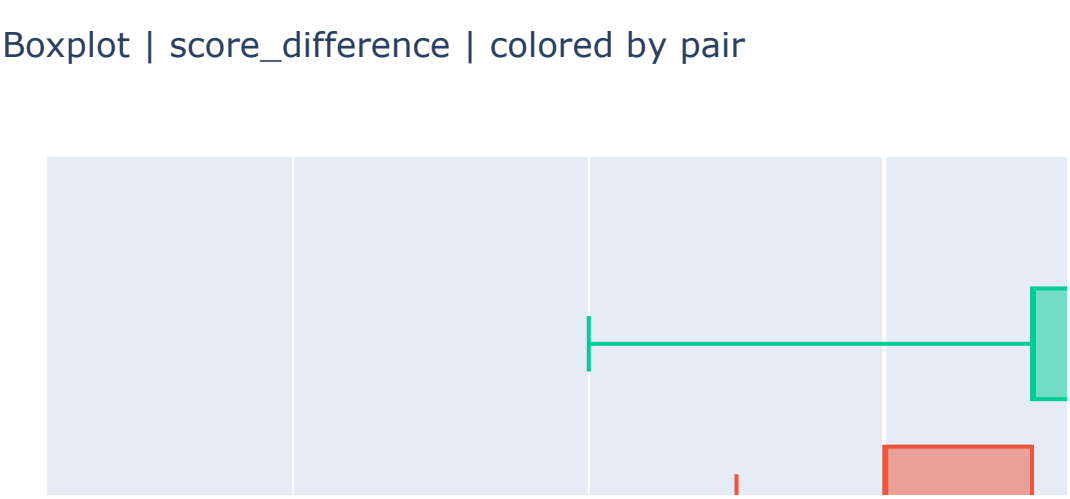
Boxplot | total_score | full sample



Boxplot | total_score | colored by pair



Boxplot | score_difference | full sample

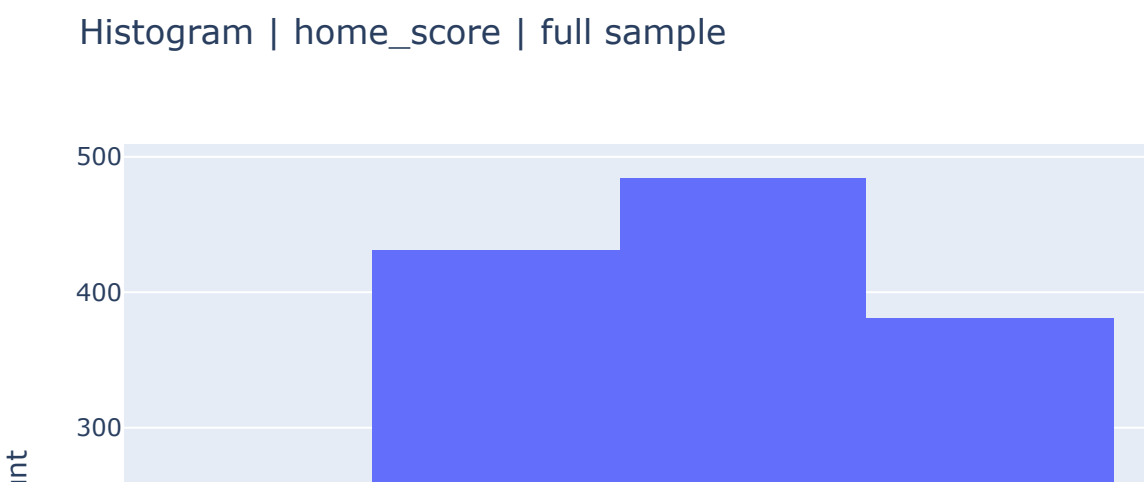


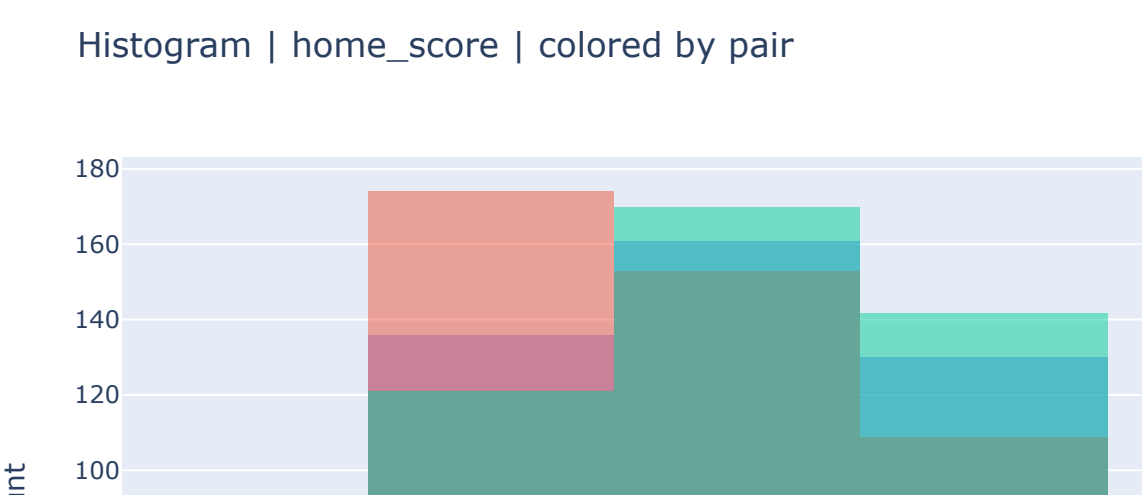
Boxplots show a stable score of home teams from all three pairs and the presence of a significant deviation in the results of away teams. Interestingly, the range of total score differs insignificantly from the home and away scores. So a situation where both the home and away team simultaneously score a large number of scores on historical data is almost impossible. This is an interesting conclusion.

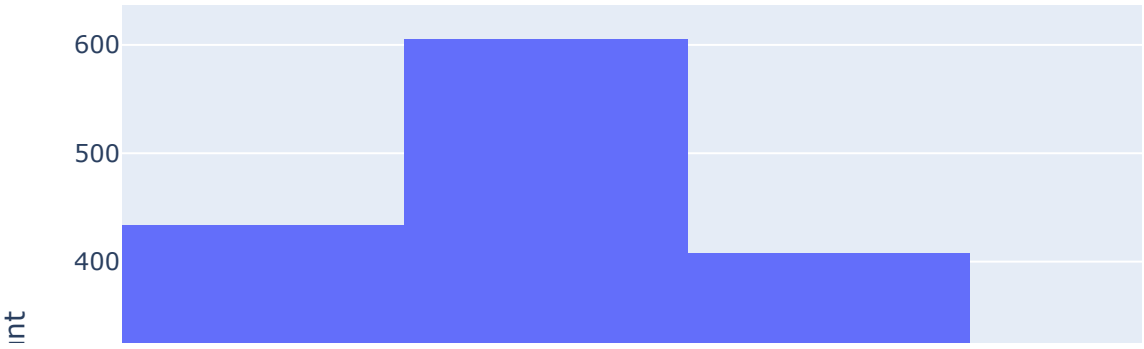
The graphical analysis also confirms the presence of statistical outliers. It is the choice of each researcher whether to remove these values, vinzorize them, fill them with the mean or zero etc. In my opinion, the key point is that the graphs show that there are no obvious errors in the data, no incorrectly entered values that would seem inadequate (e.g. 10000 scores per match). 7 scores scored may be possible and occur in only 1% of cases, but in my opinion it would be incorrect to exclude these cases from the study and bias the probability estimation. Since such events have happened in history, it means that hypothetically, although with a negligible probability, they can happen again. If you wish, you can process outliers using the custom function remove_outliers, which I placed in cell 2 of the notebook. Usefull code for it is placed in above cell.

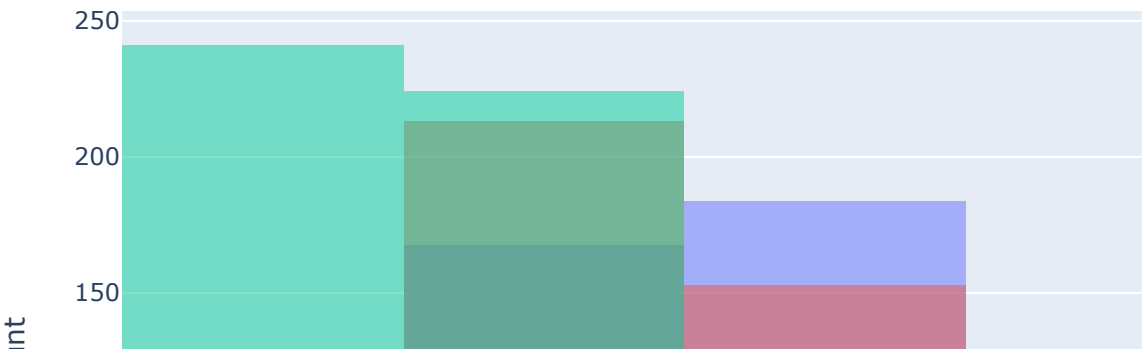
In [19]:
```
# full_data = remove_outliers(df=full_data, column='home_score')
# full_data = remove_outliers(df=full_data, column='away_score')
```

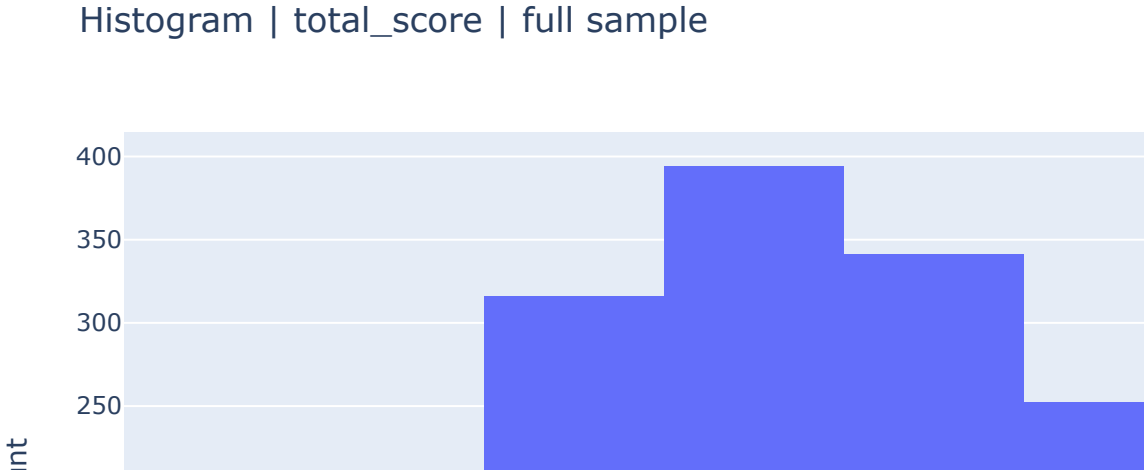
In [20]:
```
# HIST PLOTS

for col in full_data.columns:
    if col != 'event_id' and col != 'pair':
        fig = go.Figure(px.histogram(data_frame = full_data,
                                     x = col,
                                     title=f"Histogram | {col} | full sample
                                     nbins = full_data[col].nunique()
            ))
        fig.show()

        fig = go.Figure(px.histogram(data_frame = full_data,
                                     x = col,
                                     color='pair',
#                                      facet_col='pair',
                                     title=f"Histogram | {col} | colored by
                                     nbins = full_data[col].nunique(),
                                     barmode="overlay"
            ))
        fig.show()

del col, fig
```

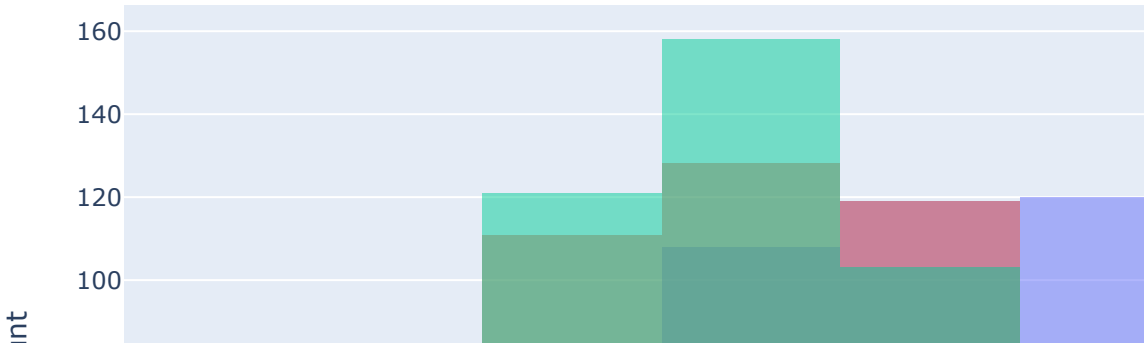


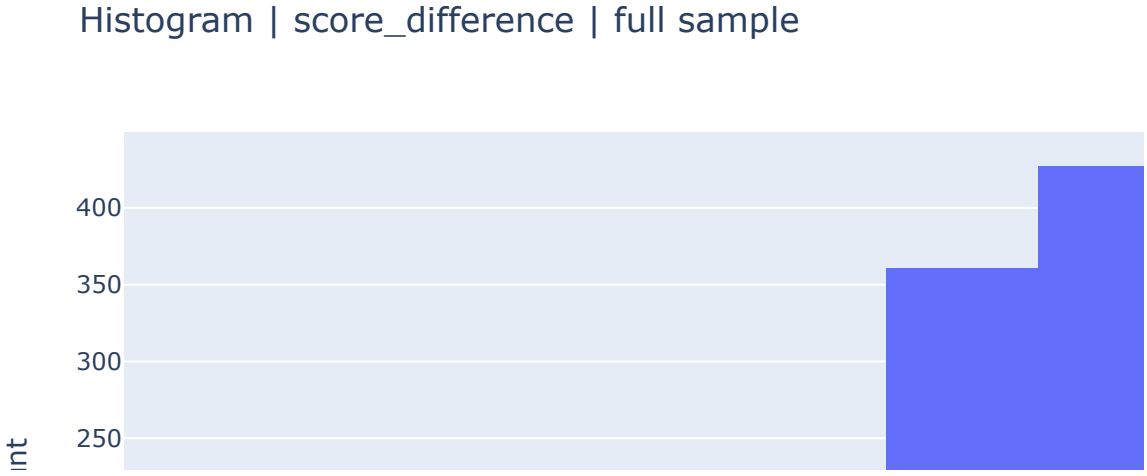

Histogram | home_score | colored by pair
180
160
140
120
100
nt

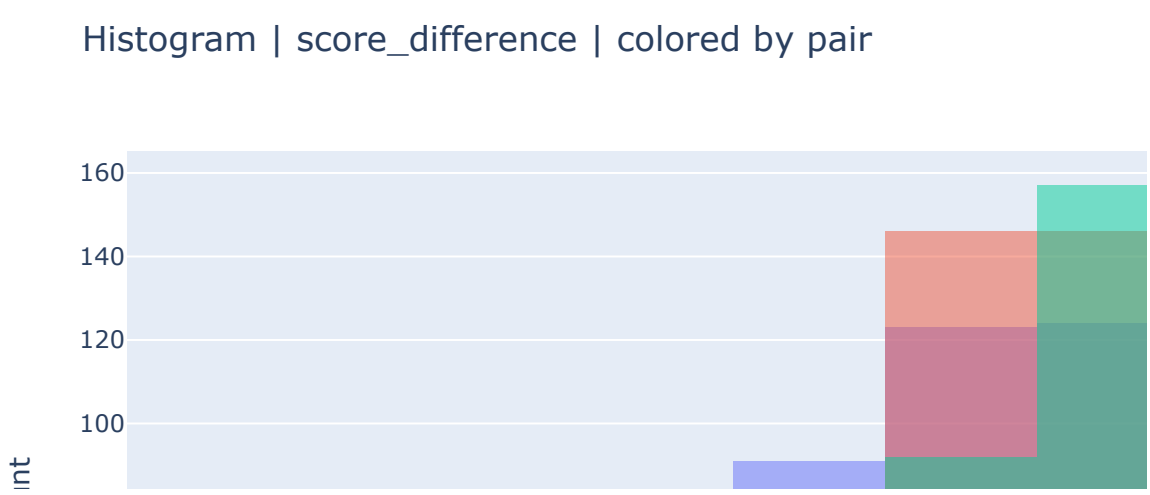

Histogram | away_score | full sample
600
500
400
nt


Histogram | away_score | colored by pair
250
200
150
nt


Histogram | total_score | full sample
400
350
300
250
nt


Histogram | total_score | colored by pair
160
140
120
100
nt


Histogram | score_difference | full sample
400
350
300
250
int



Analysis of the histograms of the score distributions shows an obvious non-normality of the data distribution. This suggests that if the data are Poisson distributed and the scores of home and away teams are independent, then a Poisson model for predicting probabilities may be appropriate.

However, the graphical analysis shows meaningful differences between pairs of teams. In my opinion, it would be incorrect to make predictions without differentiating between pairs of teams. The average estimation may give a fairer understanding of the generalized probability tendencies, but in our case the accuracy of the forecast for a particular pair is more important, so we will leave the generalized estimation only to make the understanding of trends and features in the data deeper.

CORRELATION ANALYSIS

To check if the Poisson model fits us, we need to analyze the correlation between the number of home scores and the number of away scores and check the distribution of these variables.

In [21]:
```
correlation = full_data['home_score'].corr(full_data['away_score'])
print(f"Correlation between home and away scores through all pairs: {correla
```

```
Correlation between home and away scores through all pairs: -0.05
```

In [22]:
```
for pair, group in full_data.groupby('pair'):
    correlation = group['home_score'].corr(group['away_score'])
    print(f"orrelation between home and away scores through the pair №{pair}
```

```
orrelation between home and away scores through the pair №1: -0.07
orrelation between home and away scores through the pair №2: 0.01
orrelation between home and away scores through the pair №3: -0.02
```

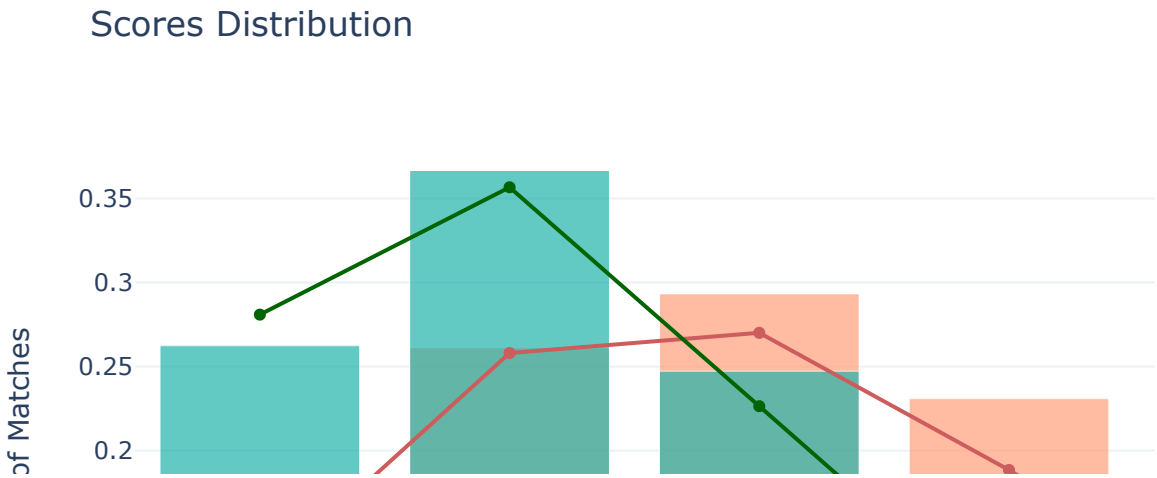
Correlation analysis allows us to find a linear relationship. The value calculated for the total sample as well as the correlation coefficients calculated for each individual pair indicate that there is no significant relationship between the variables and removes the corresponding restriction on the use of the Poisson model.

TEST POISSON DISTRIBUTION

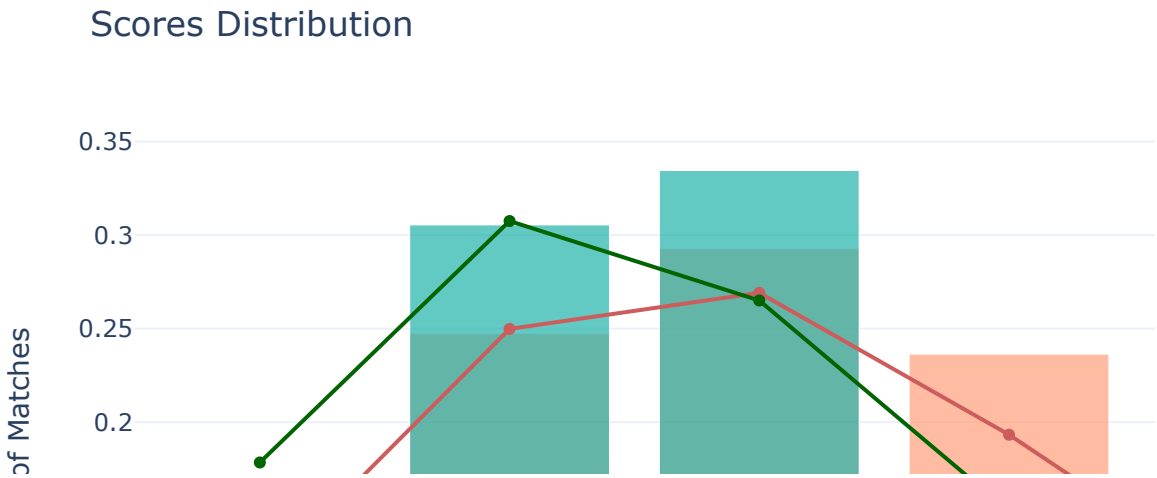
To test the distribution of the variables of interest, a custom function test_poisson_dist is used in the following code. Functions depicts plots for vizual analysis of distribution and make some statistical tests in order to make consistent decision. For more, look into the second cell of this notebook.

In [23]:
```
fig = test_poisson_dist(full_data, sample = 'full')
fig.show()

for pair, group in full_data.groupby('pair')[['home_score', 'away_score']]:
    fig = test_poisson_dist(group, sample = pair)
    fig.show()
```

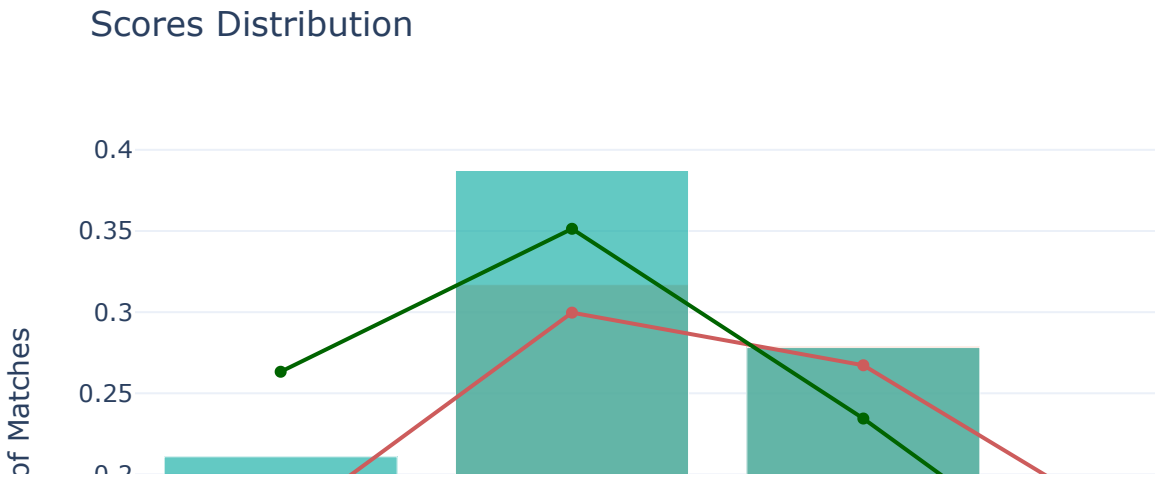
```
Test full sample for Poisson distribution
```



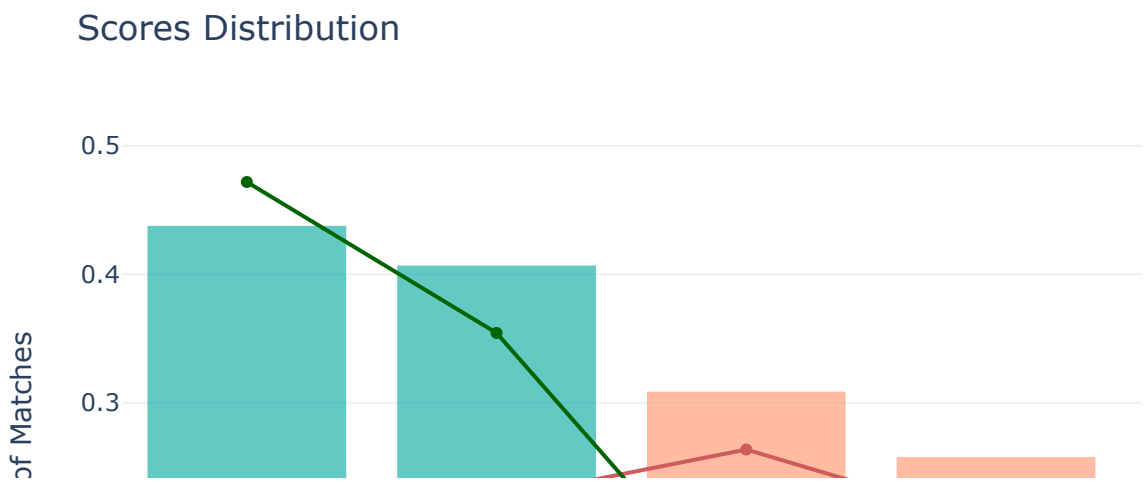
Test 1 sample for Poisson distribution



Test 2 sample for Poisson distribution



Test 3 sample for Poisson distribution



The graphical analysis shows that the actual and theoretical distributions are very close, the patterns are extremely similar, and there are no catastrophic divergences, which means that it is likely that the datasets columns can indeed fit a Poisson distribution.

Since we found out that there is no distribution mismatch, correlation dependence is not confirmed, so we can use the Poisson model to construct probability matrices.

However, it should be noted that at each stage of our exploratory analysis we noted the differences between the pairs of teams and the presence of home team advantage. This should be taken into account in the model. The attached file Model.py calculates not only the overall probability matrix, but also separate matrices for each pair, and also contains a parameter that takes into account the home team advantage. To get the results without this adjustment, you should set the parameter = False when initializing the class

In [27]:
```
!jupyter nbconvert --to webpdf --allow-chromium-download report_eda.ipynb
```

```
[NbConvertApp] Converting notebook report_eda.ipynb to webpdf
[NbConvertApp] Building PDF
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 272116 bytes to report_eda.pdf
```

In [ ]: