FIFA World Cup Analysis

Domain - Sports

Problem Statment:

With FIFA is in the blood of many people of the world. You are tasked to tell the story of unsung analysts who put great efforts to provide accurate data to answer every question of fans. The FIFA World Cup is a global football competition contested by the various football-playing nations of the world. It is contested every four years and is the most prestigious and important trophy in the sport of football. The World Cups dataset shows all information about all the World Cups in history, while the World Cup Matches dataset shows all the results from the matches contested as part of the cups. Find key metrics and factors that influence the World Cup win. Do your own research and come up with your findings.

Table of Contents

Table of Contents

8. ### Coaching Impact
9. ### Defensive Analysis
10. ### Match Environment
11. ### Game Dynamics
12. ### Historical Comparisons
13. ### Attendance Analysis
14. ### Regional Performance
15. ### Preparing Data and Machine Learning Models
    - ### Correlation Matrix
    - ### Split Train & Test
    - ### Linear Regression
    - ### Standard Scaler
    - ### Label Encoder
    - ### MinMaxScaler
    - ### Logistic Regression

Import Necessary Libraries

In [1]:
```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

from sklearn.preprocessing import LabelEncoder, MinMaxScaler
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score


from sklearn.metrics import mean_squared_error, r2_score
from scipy.stats import ttest_ind
```

Exploratory Data Analysis

Description of each Column Represents

WorldCupMatches Dataset

Year:The year the match took place.

Datetime:The date and time of the match.

Stage:The stage of the tournament

Stadium: The name of the stadium where the match was played.

City:The city where the match was held.

Home Team Name:The name of the home team.

Home Team Goals: The number of goals scored by the home team.

Away Team Goals:The number of goals scored by the away team.

Win Conditions:Conditions under which the match was won

Attendance:The number of people who attended the match.

Half Time Away Goals: The number of goals scored by the away team by half time.

Referee: The name of the referee.

Assistant 1:The name of the first assistant referee.

Assistant 2: The name of the second assistant referee.

Round ID: A unique identifier for the round.

Match ID:A unique identifier for the match.

Home Team Initials: The initials of the home team.

Away Team Initials: The initials of the away team.

In [2]:
```
#loading the dataframe
matches = pd.read_csv('WorldCupMatches.csv')
```

In [3]:
```
#show the columns presnt if the dataframe
matches.columns
```

Out[3]:
```
Index(['Year', 'Datetime', 'Stage', 'Stadium', 'City', 'Home Team Name',
       'Home Team Goals', 'Away Team Goals', 'Away Team Name',
       'Win conditions', 'Attendance', 'Half-time Home Goals',
       'Half-time Away Goals', 'Referee', 'Assistant 1', 'Assistant 2',
       'RoundID', 'MatchID', 'Home Team Initials', 'Away Team Initials'],
      dtype='object')
```

In [4]:
```
#shows the 1st 5 rows of the dataframe
matches.head()
```

Out[4]:

	Year	Datetime	Stage	Stadium	City	Home Team Name	Home Team Goals	Away Team Goals	Away Team Name	Win conditions	Attendance	Half-time Home Goals	Half-time Away Goals	Referee	Ass
0	1930.0	13 Jul 1930 - 15:00	Group 1	Pocitos	Montevideo	France	4.0	1.0	Mexico		4444.0	3.0	0.0	LOMBARDI Domingo (URU)	CRIS Henr
1	1930.0	13 Jul 1930 - 15:00	Group 4	Parque Central	Montevideo	USA	3.0	0.0	Belgium		18346.0	2.0	0.0	MACIAS Jose (ARG)	MAT Fr
2	1930.0	14 Jul 1930 - 12:45	Group 2	Parque Central	Montevideo	Yugoslavia	2.0	1.0	Brazil		24059.0	2.0	0.0	TEJADA Anibal (URU)	VALL
3	1930.0	14 Jul 1930 - 14:50	Group 3	Pocitos	Montevideo	Romania	3.0	1.0	Peru		2549.0	1.0	0.0	WARNKEN Alberto (CHI)	LAN Jea
4	1930.0	15 Jul 1930 - 16:00	Group 1	Parque Central	Montevideo	Argentina	1.0	0.0	France		23409.0	0.0	0.0	REGO Gilberto (BRA)	SAI Ulise

In [5]:
```
# getting total number of rows and column in the dataframe
print(f" Shape of the dataframe = {matches.shape}")
totalrows = matches.shape[0]
print(f" Total number of rows in the dataset =  {totalrows}")
```

```
 Shape of the dataframe = (4572, 20)
 Total number of rows in the dataset =  4572
```

In [6]:
```
#all information about the dataframe
matches.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4572 entries, 0 to 4571
Data columns (total 20 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   Year                  852 non-null    float64
 1   Datetime              852 non-null    object 
 2   Stage                 852 non-null    object 
 3   Stadium               852 non-null    object 
 4   City                  852 non-null    object 
 5   Home Team Name        852 non-null    object 
 6   Home Team Goals       852 non-null    float64
 7   Away Team Goals       852 non-null    float64
 8   Away Team Name        852 non-null    object 
 9   Win conditions        852 non-null    object 
 10  Attendance            850 non-null    float64
 11  Half-time Home Goals  852 non-null    float64
 12  Half-time Away Goals  852 non-null    float64
 13  Referee               852 non-null    object 
 14  Assistant 1           852 non-null    object 
 15  Assistant 2           852 non-null    object 
 16  RoundID               852 non-null    float64
 17  MatchID               852 non-null    float64
 18  Home Team Initials    852 non-null    object 
 19  Away Team Initials    852 non-null    object 
dtypes: float64(8), object(12)
memory usage: 714.5+ KB
```

In [7]:
```
#summary analysis of the dataframe
matches.describe().T
```

Out[7]:

	count	mean	std	min	25%	50%	75%	max
Year	852.0	1.985089e+03	2.244882e+01	1930.0	1970.00	1990.0	2002.00	2014.0
Home Team Goals	852.0	1.811033e+00	1.610255e+00	0.0	1.00	2.0	3.00	10.0
Away Team Goals	852.0	1.022300e+00	1.087573e+00	0.0	0.00	1.0	2.00	7.0
Attendance	850.0	4.516480e+04	2.348525e+04	2000.0	30000.00	41579.5	61374.50	173850.0
Half-time Home Goals	852.0	7.089202e-01	9.374141e-01	0.0	0.00	0.0	1.00	6.0
Half-time Away Goals	852.0	4.284038e-01	6.912519e-01	0.0	0.00	0.0	1.00	5.0
RoundID	852.0	1.066177e+07	2.729613e+07	201.0	262.00	337.0	249722.00	97410600.0
MatchID	852.0	6.134687e+07	1.110572e+08	25.0	1188.75	2191.0	43950059.25	300186515.0

WorldCupPlayers Dataset

Round Id: A unique identifier for the round.

Match ID: A unique identifier for the match.

Team Initials: The initials of the team.

Coach Name: The name of the team's coach.

Line Up: Indicates if the player was in the starting lineup or a substitute.

Shirt Number: The player's shirt number.

Player Name: The name of the player.

Position: The player's position on the field

Event: special event involving the player

In [8]:
```
#loading the dataframe
players = pd.read_csv('WorldCupPlayers.csv')
```

In [9]:
```
#shows the columns present in the dataframe
players.columns
```

Out[9]:
```
Index(['RoundID', 'MatchID', 'Team Initials', 'Coach Name', 'Line-up',
       'Shirt Number', 'Player Name', 'Position', 'Event'],
      dtype='object')
```

In [10]:
```
#shows the 1st 5 rows of the dataframe
players.head()
```

Out[10]:

	RoundID	MatchID	Team Initials	Coach Name	Line-up	Shirt Number	Player Name	Position	Event
0	201	1096	FRA	CAUDRON Raoul (FRA)	S	0	Alex THEPOT	GK	NaN
1	201	1096	MEX	LUQUE Juan (MEX)	S	0	Oscar BONFIGLIO	GK	NaN
2	201	1096	FRA	CAUDRON Raoul (FRA)	S	0	Marcel LANGILLER	NaN	G40'
3	201	1096	MEX	LUQUE Juan (MEX)	S	0	Juan CARRENO	NaN	G70'
4	201	1096	FRA	CAUDRON Raoul (FRA)	S	0	Ernest LIBERATI	NaN	NaN

In [11]:
```
# getting total number of rows and column in the dataframe
print(f" Shape of the dataframe = {players.shape}")
totalrows = players.shape[0]
print(f" Total number of rows in the dataset =  {totalrows}")
```

```
 Shape of the dataframe = (37784, 9)
 Total number of rows in the dataset =  37784
```

In [12]:
```
#all information abput the dataframe
players.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 37784 entries, 0 to 37783
Data columns (total 9 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   RoundID        37784 non-null  int64 
 1   MatchID        37784 non-null  int64 
 2   Team Initials  37784 non-null  object
 3   Coach Name     37784 non-null  object
 4   Line-up        37784 non-null  object
 5   Shirt Number   37784 non-null  int64 
 6   Player Name    37784 non-null  object
 7   Position       4143 non-null   object
 8   Event          9069 non-null   object
dtypes: int64(3), object(6)
memory usage: 2.6+ MB
```

In [13]:
```
#summary analysis
players.describe().T
```

Out[13]:

	count	mean	std	min	25%	50%	75%	max
RoundID	37784.0	1.105647e+07	2.770144e+07	201.0	263.0	337.0	255931.0	97410600.0
MatchID	37784.0	6.362233e+07	1.123916e+08	25.0	1199.0	2216.0	97410003.0	300186515.0
Shirt Number	37784.0	1.072602e+01	6.960138e+00	0.0	5.0	11.0	17.0	23.0

WorldCups Dataset

Year: The year the World Cup took place.

Country: country where the World Cup was held.

Winner: The team that won the World Cup.

Runners-up: The team that was the runner-up.

Third: The team that finished in third place.

Fourth: The team that finished in fourth place.

GoalsScored: The total number of goals scored in the tournament.

QualifiedTeams: The number of teams that qualified for the tournament.

MatchesPlayed: The total number of Played

Attendance: The total attendance for the tournament.

In [14]:
```
#loading the dataframe
worldcups = pd.read_csv('WorldCups.csv')
```

In [15]:
```
#shows the 1st 5 rows of the dataframe
worldcups.head()
```

Out[15]:

	Year	Country	Winner	Runners-Up	Third	Fourth	GoalsScored	QualifiedTeams	MatchesPlayed	Attendance
0	1930	Uruguay	Uruguay	Argentina	USA	Yugoslavia	70	13	18	590.549
1	1934	Italy	Italy	Czechoslovakia	Germany	Austria	70	16	17	363.000
2	1938	France	Italy	Hungary	Brazil	Sweden	84	15	18	375.700
3	1950	Brazil	Uruguay	Brazil	Sweden	Spain	88	13	22	1.045.246
4	1954	Switzerland	Germany FR	Hungary	Austria	Uruguay	140	16	26	768.607

In [16]:
```
# getting total number of rows and column in the dataframe
print(f" Shape of the dataframe = {worldcups.shape}")
totalrows = worldcups.shape[0]
print(f" Total number of rows in the dataset =  {totalrows}")
```

```
 Shape of the dataframe = (20, 10)
 Total number of rows in the dataset =  20
```

In [17]:
```
#shows the columns present in the dataframe
worldcups.columns
```

Out[17]:
```
Index(['Year', 'Country', 'Winner', 'Runners-Up', 'Third', 'Fourth',
       'GoalsScored', 'QualifiedTeams', 'MatchesPlayed', 'Attendance'],
      dtype='object')
```

In [18]:
```
#infromation about the dataframe
worldcups.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20 entries, 0 to 19
Data columns (total 10 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   Year            20 non-null     int64 
 1   Country         20 non-null     object
 2   Winner          20 non-null     object
 3   Runners-Up      20 non-null     object
 4   Third           20 non-null     object
 5   Fourth          20 non-null     object
 6   GoalsScored     20 non-null     int64 
 7   QualifiedTeams  20 non-null     int64 
 8   MatchesPlayed   20 non-null     int64 
 9   Attendance      20 non-null     object
dtypes: int64(4), object(6)
memory usage: 1.7+ KB
```

In [19]:
```
worldcups.describe().T
```

Out[19]:

	count	mean	std	min	25%	50%	75%	max
Year	20.0	1974.80	25.582889	1930.0	1957.0	1976.0	1995.00	2014.0
GoalsScored	20.0	118.95	32.972836	70.0	89.0	120.5	145.25	171.0
QualifiedTeams	20.0	21.25	7.268352	13.0	16.0	16.0	26.00	32.0
MatchesPlayed	20.0	41.80	17.218717	17.0	30.5	38.0	55.00	64.0

In [20]:

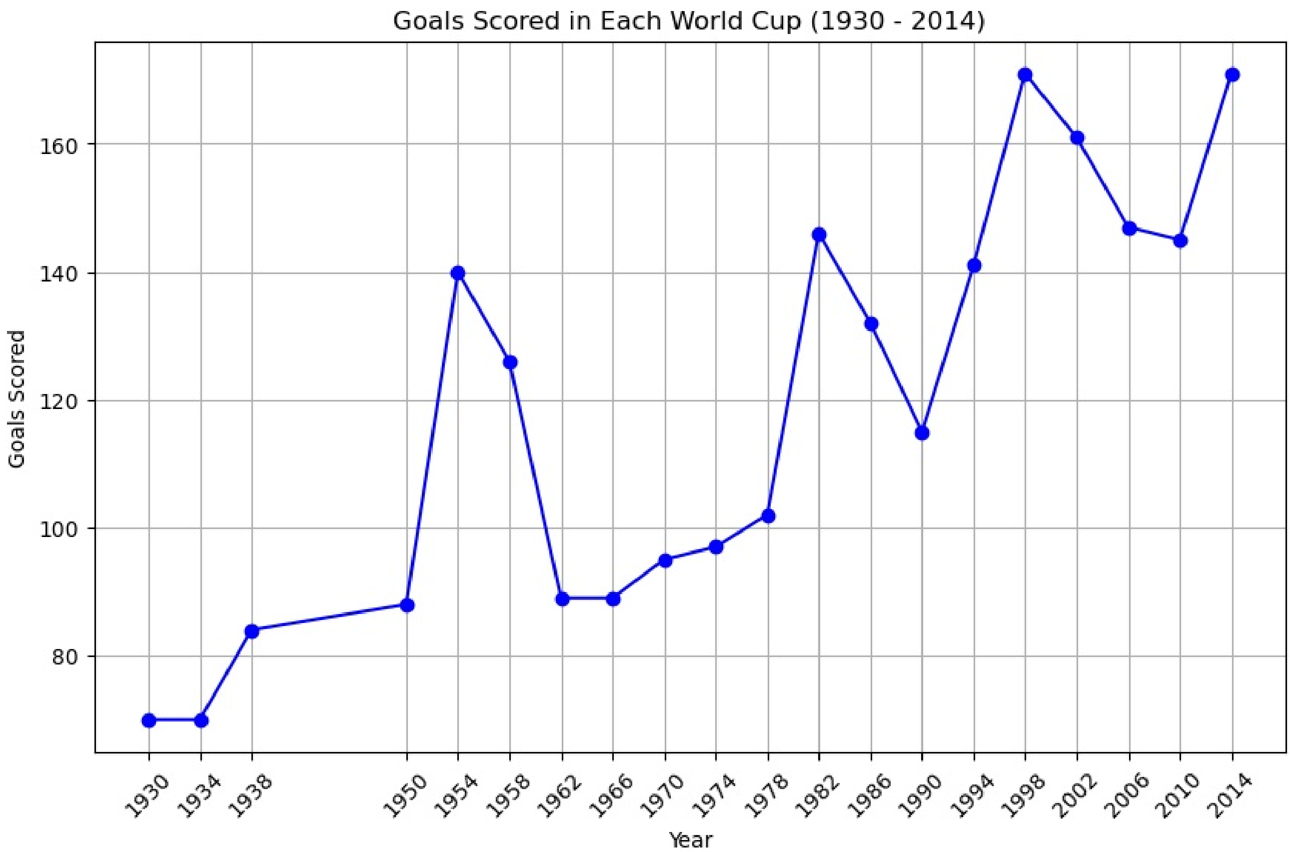
```
matches['Datetime'] = pd.to_datetime(matches['Datetime'])
```

1. **Historical Performance:**

Number of Goals Scored per World Cup changed over the years

In [21]:

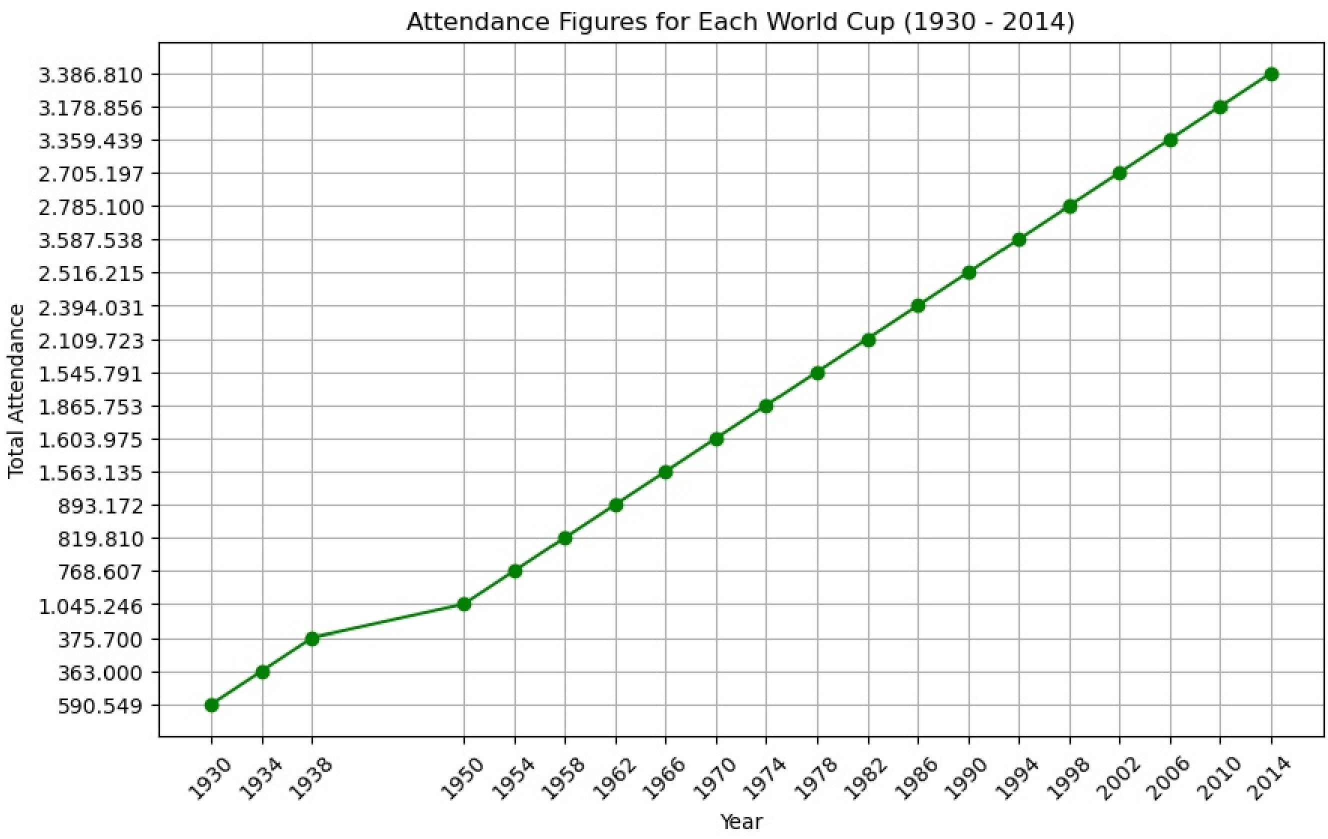
```
#Extract the Year and GoalsScored
worldcups_goals = worldcups[['Year', 'GoalsScored']]

# Plot the number of goals scored in each World Cup
plt.figure(figsize=(10, 6))
plt.plot(worldcups_goals['Year'], worldcups_goals['GoalsScored'], marker='o', linestyle='-', color='b')
plt.title('Goals Scored in Each World Cup (1930 - 2014)')
plt.xlabel('Year')
plt.ylabel('Goals Scored')
plt.grid(True)
plt.xticks(worldcups_goals['Year'], rotation=45)
plt.show()
```



Trends Attendance Figures Across Different World Cups

In [22]:

```
# Extract Year and Attendance
attendance_data = worldcups[['Year', 'Attendance']]

# Plot the attendance figures for each World Cup
plt.figure(figsize=(10, 6))
plt.plot(attendance_data['Year'], attendance_data['Attendance'], marker='o', linestyle='-', color='g')
plt.title('Attendance Figures for Each World Cup (1930 - 2014)')
plt.xlabel('Year')
plt.ylabel('Total Attendance')
plt.grid(True)
plt.xticks(attendance_data['Year'], rotation=45)
plt.show()
```



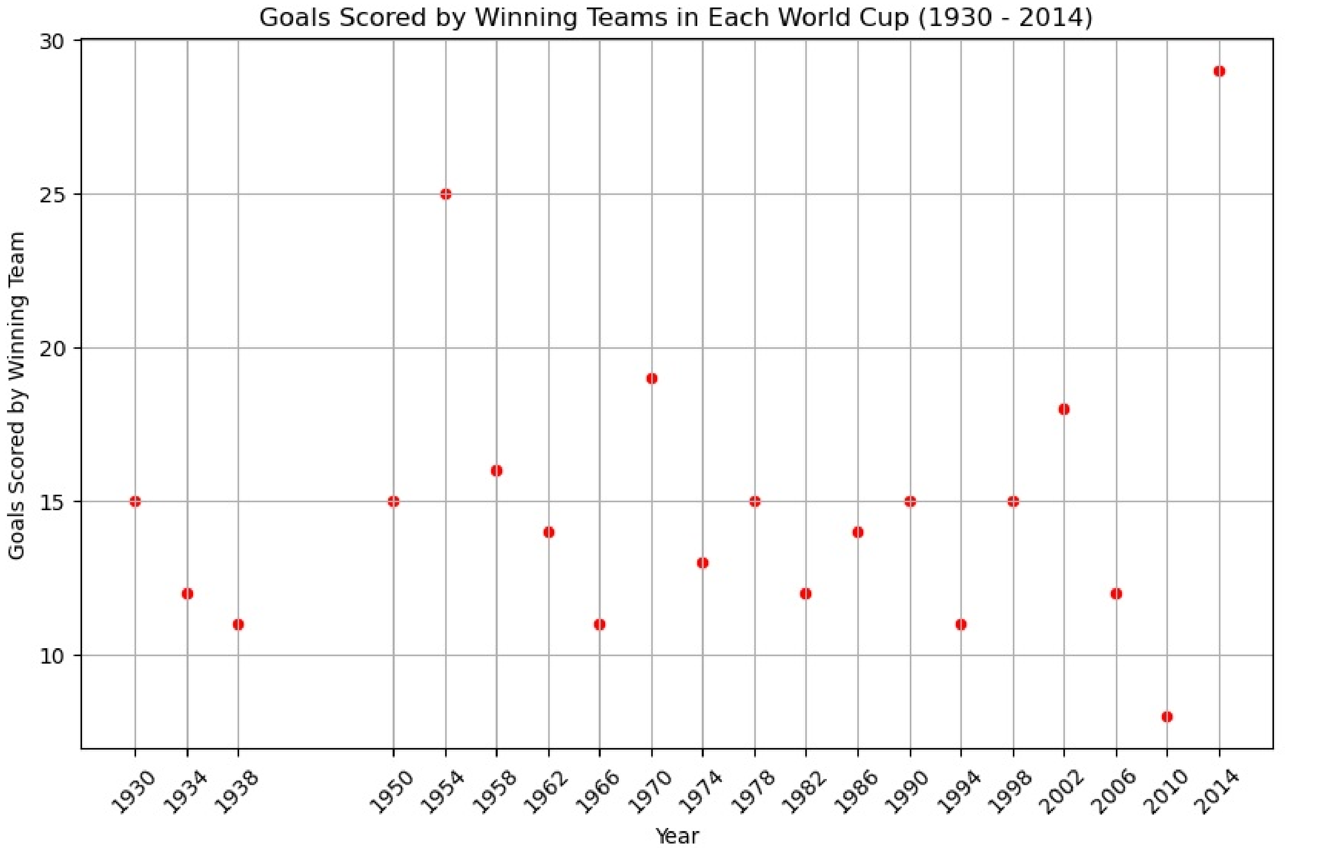
2. **Winning Factors:**

Correlation between the Number of Goals Scored by a team and winning the World Cup

In [23]:

```
# Extract Year, Winner, GoalsScored
winners_data = worldcups[['Year', 'Winner', 'GoalsScored']]

# Calculate the total goals scored by the winning team in each World Cup
winners_goals = []
for year in winners_data['Year']:
    winner = winners_data[winners_data['Year'] == year]['Winner'].values[0]
    matches_in_year = matches[matches['Year'] == year]
    total_goals = matches_in_year[matches_in_year['Home Team Name'] == winner]['Home Team Goals'].sum() + match
    winners_goals.append(total_goals)

# Add the total goals scored by winners to the dataset
winners_data['WinnerGoals'] = winners_goals

# Plot the relationship between the year and goals scored by winning teams
plt.figure(figsize=(10, 6))
sns.scatterplot(data=winners_data, x='Year', y='WinnerGoals', marker='o', color='r')
plt.title('Goals Scored by Winning Teams in Each World Cup (1930 - 2014)')
plt.xlabel('Year')
plt.ylabel('Goals Scored by Winning Team')
plt.grid(True)
plt.xticks(winners_data['Year'], rotation=45)
plt.show()

# Analyze the correlation
correlation = winners_data['GoalsScored'].corr(winners_data['WinnerGoals'])
print(f'Correlation between total goals scored in a World Cup and goals scored by the winning team: {correlatio
```

```
C:\Users\ELCOT\AppData\Local\Temp\ipykernel_10024\2352605515.py:13: 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#ret
urning-a-view-versus-a-copy
  winners_data['WinnerGoals'] = winners_goals
```




```
Correlation between total goals scored in a World Cup and goals scored by the winning team: 0.32988169360101666
```

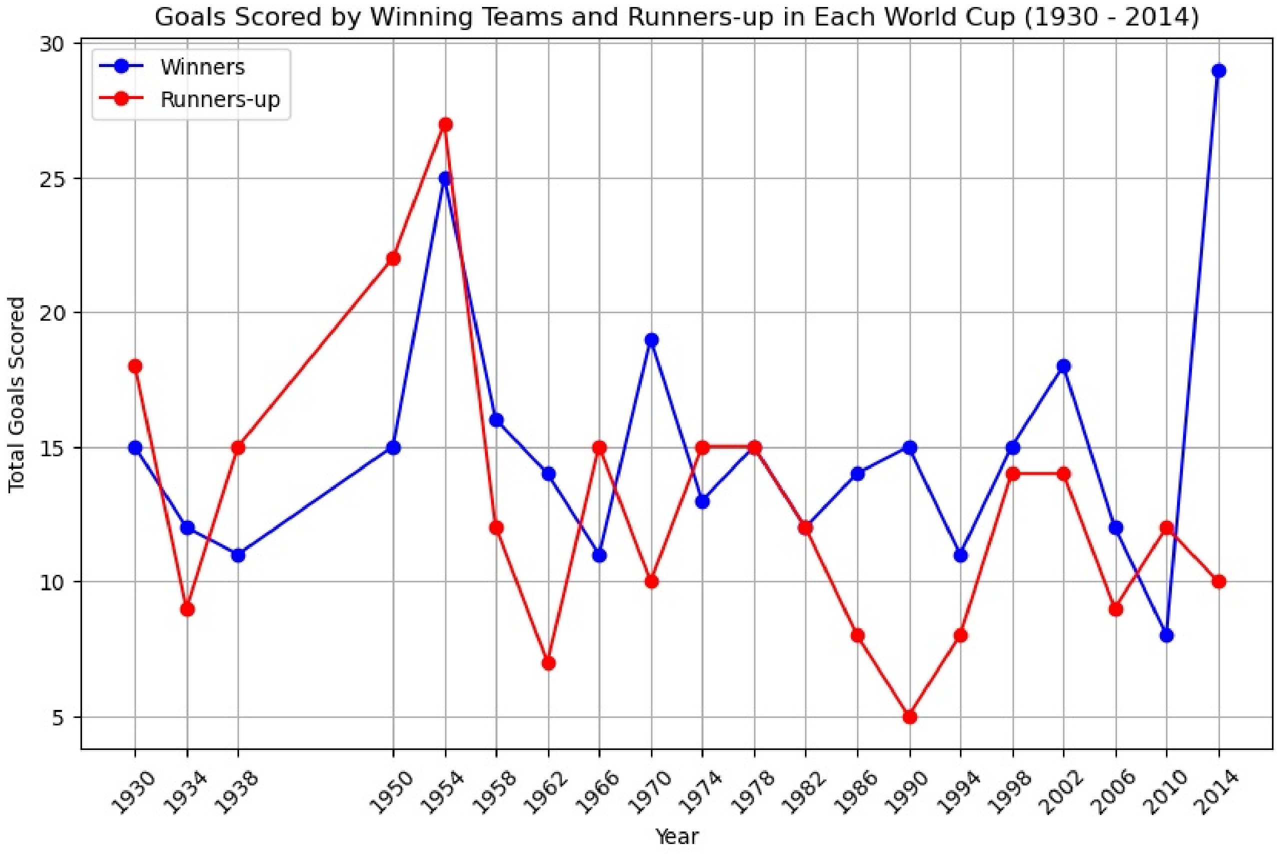
Average Number of Goals Scored by the Winning Team Compare to the Runners-Up

In [24]:

```
# Initialize dictionaries to hold total goals for winners and runners-up
winners_goals = {}
runners_up_goals = {}

# Calculate total goals for each team in each World Cup
for year in worldcups['Year']:
    winner = worldcups[worldcups['Year'] == year]['Winner'].values[0]
    runner_up = worldcups[worldcups['Year'] == year]['Runners-Up'].values[0]
    matches_in_year = matches[matches['Year'] == year]

    winner_goals = matches_in_year[matches_in_year['Home Team Name'] == winner]['Home Team Goals'].sum() + matc
    runner_up_goals = matches_in_year[matches_in_year['Home Team Name'] == runner_up]['Home Team Goals'].sum() 

    winners_goals[year] = winner_goals
    runners_up_goals[year] = runner_up_goals

# Convert dictionaries to DataFrames
winners_goals_df = pd.DataFrame(list(winners_goals.items()), columns=['Year', 'Goals'])
runners_up_goals_df = pd.DataFrame(list(runners_up_goals.items()), columns=['Year', 'Goals'])

# Calculate the average goals scored by winners and runners-up
average_winner_goals = winners_goals_df['Goals'].mean()
average_runner_up_goals = runners_up_goals_df['Goals'].mean()

print(f'Average goals scored by winning teams: {average_winner_goals}')
print(f'Average goals scored by runners-up teams: {average_runner_up_goals}')

# Plot the comparison
plt.figure(figsize=(10, 6))
plt.plot(winners_goals_df['Year'], winners_goals_df['Goals'], marker='o', linestyle='-', color='b', label='Winn
plt.plot(runners_up_goals_df['Year'], runners_up_goals_df['Goals'], marker='o', linestyle='-', color='r', label
plt.title('Goals Scored by Winning Teams and Runners-up in Each World Cup (1930 - 2014)')
plt.xlabel('Year')
plt.ylabel('Total Goals Scored')
plt.legend()
plt.grid(True)
```

```
plt.xticks(winners_goals_df['Year'], rotation=45)
plt.show()
```

```
Average goals scored by winning teams: 15.0
Average goals scored by runners-up teams: 12.85
```

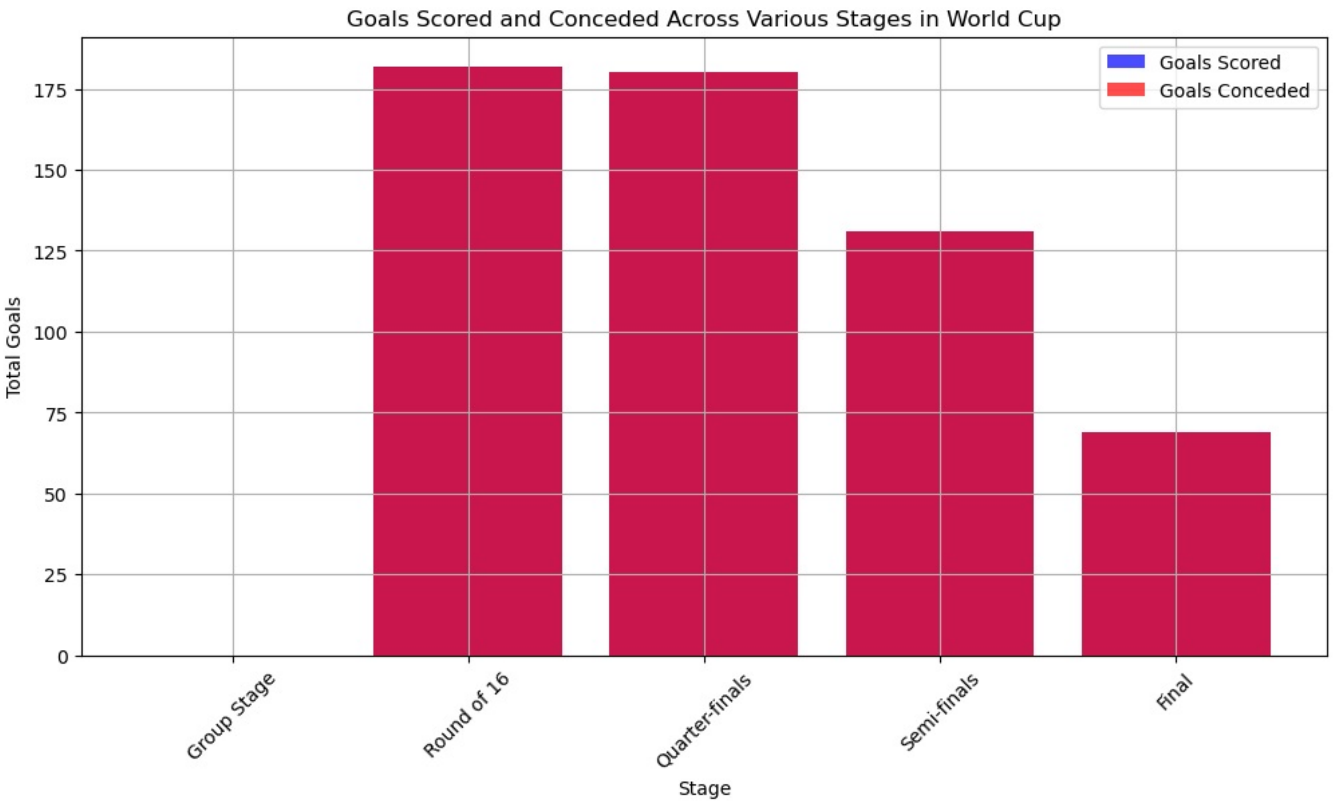


3. **Stage Analysis:**

Goals Scored and Conceded differ Across Various Stages

In [25]:

```
# Define the stages
stages = ['Group Stage', 'Round of 16', 'Quarter-finals', 'Semi-finals', 'Final']

# Initialize dictionaries to hold goals scored and conceded for each stage
goals_scored = {stage: 0 for stage in stages}
goals_conceded = {stage: 0 for stage in stages}

# Calculate the goals scored and conceded for each stage
for stage in stages:
    matches_in_stage = matches[matches['Stage'] == stage]
    goals_scored[stage] = matches_in_stage['Home Team Goals'].sum() + matches_in_stage['Away Team Goals'].sum()
    goals_conceded[stage] = matches_in_stage['Home Team Goals'].sum() + matches_in_stage['Away Team Goals'].sum

# Convert dictionaries to DataFrames
goals_scored_df = pd.DataFrame(list(goals_scored.items()), columns=['Stage', 'Goals Scored'])
goals_conceded_df = pd.DataFrame(list(goals_conceded.items()), columns=['Stage', 'Goals Conceded'])

# Plot the comparison
plt.figure(figsize=(12, 6))
plt.bar(goals_scored_df['Stage'], goals_scored_df['Goals Scored'], color='b', alpha=0.7, label='Goals Scored')
plt.bar(goals_conceded_df['Stage'], goals_conceded_df['Goals Conceded'], color='r', alpha=0.7, label='Goals Con
plt.title('Goals Scored and Conceded Across Various Stages in World Cup')
plt.xlabel('Stage')
plt.ylabel('Total Goals')
plt.legend()
plt.grid(True)
plt.xticks(rotation=45)
plt.show()
```



Average Attendance for Different Stages of the World Cup

In [26]:
```
# Group by stage and calculate the average attendance
average_attendance = matches.groupby('Stage')['Attendance'].mean().reset_index()

# Print the results
print(average_attendance)
```

```
                       Stage    Attendance
0                      Final  76383.650000
1                First round  16120.333333
2                    Group 1  41664.064516
3                    Group 2  34241.762712
4                    Group 3  34271.410714
5                    Group 4  25915.745455
6                    Group 5  35354.000000
7                    Group 6  65658.583333
8                    Group A  54321.350000
9                    Group B  51367.866667
10                   Group C  45514.937500
11                   Group D  45427.333333
12                   Group E  45675.916667
13                   Group F  42840.875000
14                   Group G  48533.166667
15                   Group H  46747.033333
16     Match for third place  50847.866667
17  Play-off for third place  68034.000000
18         Preliminary round  16875.000000
19            Quarter-finals  45697.484848
20               Round of 16  52346.842857
21               Semi-finals  59053.333333
22               Third place  57741.500000
```

4. **Home Advantage:**

Host Countries reach the Semifinals or Finals

In [27]:
```
# Filter matches where the host country participated
host_matches = matches[(matches['Home Team Initials'] == matches['Home Team Initials'].mode()[0]) | (matches['A

# Filter matches where the host country reached the semifinals or finals
host_semifinals_finals = host_matches[(host_matches['Stage'] == 'Semi-finals') | (host_matches['Stage'] == 'Fin

# Count the number of times the host country reached the semifinals or finals
times_host_reached_semifinals_finals = len(host_semifinals_finals)

# Count the total number of tournaments where the host country participated
total_tournaments = len(matches['Year'].unique())

# Calculate the percentage of tournaments where the host country reached the semifinals or finals
percentage = (times_host_reached_semifinals_finals / total_tournaments) * 100
```

```
print(f"The host country reached the semifinals or finals in {times_host_reached_semifinals_finals} out of {tot
```

The host country reached the semifinals or finals in 12 out of 21 tournaments, which is approximately 57.14% of the time.

Win Rate of Host Countries Compared to Non-Host Countries

In [28]:

```
# Filter matches where the host country participated
host_matches = matches[(matches['Home Team Initials'] == matches['Home Team Initials'].mode()[0]) | (matches['A

# Filter matches where the host country won
host_wins = host_matches[(host_matches['Home Team Initials'] == host_matches['Home Team Initials'].mode()[0]) &
                         (host_matches['Away Team Initials'] == host_matches['Away Team Initials'].mode()[0]) &

# Filter matches where the host country lost
host_losses = host_matches[(host_matches['Home Team Initials'] == host_matches['Home Team Initials'].mode()[0])
                           (host_matches['Away Team Initials'] == host_matches['Away Team Initials'].mode()[0])

# Count the total number of matches where the host country participated
total_host_matches = len(host_matches)

# Count the number of matches where the host country won and lost
host_wins_count = len(host_wins)
host_losses_count = len(host_losses)

# Calculate the win rate of the host country
win_rate_host = (host_wins_count / total_host_matches) * 100

# Calculate the total number of matches where the host country did not participate
total_non_host_matches = len(matches) - total_host_matches

# Calculate the number of matches where the non-host country won and lost
non_host_wins_count = len(matches[(matches['Home Team Initials'] != matches['Home Team Initials'].mode()[0]) &
                                  (matches['Away Team Initials'] != matches['Away Team Initials'].mode()[0]) &
non_host_losses_count = len(matches[(matches['Home Team Initials'] != matches['Home Team Initials'].mode()[0])
                                    (matches['Away Team Initials'] != matches['Away Team Initials'].mode()[0])

# Calculate the win rate of non-host countries
win_rate_non_host = (non_host_wins_count / total_non_host_matches) * 100

print(f"Win rate of host countries: {win_rate_host:.2f}%")
print(f"Win rate of non-host countries: {win_rate_non_host:.2f}%")
```

```
Win rate of host countries: 56.03%
Win rate of non-host countries: 14.12%
```

5. **Team Performance:**

Teams have Consistently reached the Semifinals over Multiple World Cups

In [29]:

```
# Filter matches to include only semifinals
semifinal_matches = matches[matches['Stage'] == 'Semi-finals']

# Identify the teams that participated in the semifinals
semifinal_teams = semifinal_matches[['Home Team Name', 'Away Team Name']]

# Flatten the DataFrame to a list of teams
semifinal_teams_list = pd.concat([semifinal_teams['Home Team Name'], semifinal_teams['Away Team Name']])

# Count the occurrences of each team reaching the semifinals
semifinal_teams_count = semifinal_teams_list.value_counts()

# Display the results
print(semifinal_teams_count)
```

```
Brazil            9
Germany FR        7
Italy             7
Germany           6
France            5
Argentina         5
Uruguay           4
Netherlands       4
Sweden            3
England           2
Czechoslovakia    2
Portugal          2
Hungary           2
Yugoslavia        2
Austria           2
Belgium           1
Turkey            1
Korea Republic    1
Croatia           1
USA               1
Soviet Union      1
Chile             1
Bulgaria          1
Poland            1
Spain             1
dtype: int64
```

Average Number of Goals Scored by Teams in their Winning Years

In [30]:

```python
# Initialize a dictionary to hold total goals for winners
winner_goals = {}

# Iterate through each World Cup year to calculate total goals for the winning team
for year in worldcups['Year']:
    winner = worldcups[worldcups['Year'] == year]['Winner'].values[0]
    matches_in_year = matches[matches['Year'] == year]

    # Calculate goals scored by the winning team
    winner_goals[year] = matches_in_year[matches_in_year['Home Team Name'] == winner]['Home Team Goals'].sum()

# Convert the dictionary to a DataFrame
winner_goals_df = pd.DataFrame(list(winner_goals.items()), columns=['Year', 'Goals'])

# Calculate the average number of goals scored by the winning teams
average_winner_goals = winner_goals_df['Goals'].mean()

print(f'Average number of goals scored by teams in their winning years: {average_winner_goals:.2f}')
```

```
Average number of goals scored by teams in their winning years: 15.00
```

6. **Match Analysis:**

Distribution of Match Outcomes team Won World Cup

In [31]:

```python
# Initialize a dictionary to hold match outcomes for winners
outcomes = {'Team': [], 'Win': [], 'Lose': [], 'Draw': []}

# Iterate through each World Cup year to analyze match outcomes for the winning team
for year in worldcups['Year']:
    winner = worldcups[worldcups['Year'] == year]['Winner'].values[0]
    matches_in_year = matches[matches['Year'] == year]

    # Initialize counts for outcomes
    win_count = 0
    lose_count = 0
    draw_count = 0

    # Analyze each match involving the winning team
    for index, match in matches_in_year.iterrows():
        home_team = match['Home Team Name']
        away_team = match['Away Team Name']
        home_goals = match['Home Team Goals']
        away_goals = match['Away Team Goals']

        if home_team == winner or away_team == winner:
            if home_goals == away_goals:
                draw_count += 1
            elif (home_team == winner and home_goals > away_goals) or (away_team == winner and away_goals > hom
                win_count += 1
            else:
                lose_count += 1

    outcomes['Team'].append(winner)
    outcomes['Win'].append(win_count)
    outcomes['Lose'].append(lose_count)
```

```
    outcomes['Draw'].append(draw_count)

# Convert the dictionary to a DataFrame
outcomes_df = pd.DataFrame(outcomes)

# Calculate the total outcomes
total_wins = outcomes_df['Win'].sum()
total_loses = outcomes_df['Lose'].sum()
total_draws = outcomes_df['Draw'].sum()
total_matches = total_wins + total_loses + total_draws

# Calculate the distribution of match outcomes
win_percentage = (total_wins / total_matches) * 100
lose_percentage = (total_loses / total_matches) * 100
draw_percentage = (total_draws / total_matches) * 100

print(f'Distribution of match outcomes for World Cup-winning teams:')
print(f'Wins: {win_percentage:.2f}%')
print(f'Losses: {lose_percentage:.2f}%')
print(f'Draws: {draw_percentage:.2f}%')
```

```
Distribution of match outcomes for World Cup-winning teams:
Wins: 82.81%
Losses: 3.12%
Draws: 14.06%
```

Number of Goals Scored in the First Half Compare to the Second Half

In [32]:

```
# Calculate total goals scored in the first half and the second half
first_half_goals = matches['Half-time Home Goals'].sum() + matches['Half-time Away Goals'].sum()
second_half_goals = (matches['Home Team Goals'].sum() - matches['Half-time Home Goals'].sum()) + (matches['Away

# Print the results
print(f'Total goals scored in the first half: {first_half_goals}')
print(f'Total goals scored in the second half: {second_half_goals}')

# Calculate the average goals per half
total_matches = len(matches)
average_first_half_goals = first_half_goals / total_matches
average_second_half_goals = second_half_goals / total_matches

print(f'Average goals per match in the first half: {average_first_half_goals:.2f}')
print(f'Average goals per match in the second half: {average_second_half_goals:.2f}')
```

```
Total goals scored in the first half: 969.0
Total goals scored in the second half: 1445.0
Average goals per match in the first half: 0.21
Average goals per match in the second half: 0.32
```

7. **Player Performance:**

Players have Scored the Most Goals in World Cup history

In [33]:

```
# Filter out only goal events
goal_events = players[players['Event'].str.contains('G', na=False)]

# Count the number of goals scored by each player
top_scorers = goal_events['Player Name'].value_counts().head(10)

# Print the top scorers
print(top_scorers)
```

```
RONALDO                                13
KLOSE                                  12
Gerd MUELLER                            9
Uwe SEELER                              9
M�LLER                                  9
Grzegorz LATO                           8
PEL� (Edson Arantes do Nascimento)      8
JAIRZINHO                               8
Helmut RAHN                             8
Just FONTAINE                           6
Name: Player Name, dtype: int64
```

Positions are Most Associated with Top Goal Scorers

In [34]:

```
# Filter out only goal events
goal_events = players[players['Event'].str.contains('G', na=False)]

# Count the number of goals scored by each player
top_scorers = goal_events['Player Name'].value_counts().head(10).index

# Extract positions of top goal scorers
top_scorers_positions = players[players['Player Name'].isin(top_scorers)]['Position']
```

```
# Count the frequency of each position
position_counts = top_scorers_positions.value_counts()

# Print the position counts
print(position_counts)
```

```
C    19
Name: Position, dtype: int64
```

8. **Coaching Impact:**

Correlation between the Experience of a Coach and the Team's Performance

In [35]:

```
# Extract coach and match information
coach_matches = players[['MatchID', 'Team Initials', 'Coach Name']].drop_duplicates()

# Count the number of matches each coach has coached
coach_experience = coach_matches['Coach Name'].value_counts()

# Merge with match outcomes
match_outcomes = matches[['MatchID', 'Home Team Initials', 'Home Team Goals', 'Away Team Goals', 'Away Team Ini

# Define a function to determine the result of a match
def get_match_result(row, team):
    if row['Home Team Initials'] == team:
        if row['Home Team Goals'] > row['Away Team Goals']:
            return 'Win'
        elif row['Home Team Goals'] < row['Away Team Goals']:
            return 'Lose'
        else:
            return 'Draw'
    elif row['Away Team Initials'] == team:
        if row['Away Team Goals'] > row['Home Team Goals']:
            return 'Win'
        elif row['Away Team Goals'] < row['Home Team Goals']:
            return 'Lose'
        else:
            return 'Draw'
    return 'Unknown'

# Calculate win rate for each coach
coach_performance = {}
for coach in coach_experience.index:
    coach_teams = coach_matches[coach_matches['Coach Name'] == coach]
    wins, losses, draws = 0, 0, 0
    for _, row in coach_teams.iterrows():
        match = match_outcomes[match_outcomes['MatchID'] == row['MatchID']].iloc[0]
        result = get_match_result(match, row['Team Initials'])
        if result == 'Win':
            wins += 1
        elif result == 'Lose':
            losses += 1
        elif result == 'Draw':
            draws += 1
    total_matches = wins + losses + draws
    win_rate = wins / total_matches if total_matches > 0 else 0
    coach_performance[coach] = {'Experience': coach_experience[coach], 'Win Rate': win_rate}

# Convert to DataFrame
coach_performance_df = pd.DataFrame(coach_performance).T

# Calculate the correlation between experience and win rate
correlation = coach_performance_df['Experience'].corr(coach_performance_df['Win Rate'])

print(f'Correlation between coach experience and team win rate: {correlation:.2f}')
```

```
Correlation between coach experience and team win rate: 0.42
```

9. **Defensive Analysis:**

Clean Sheets (no goals conceded) do winning teams

In [36]:

```
# Determine if a match was a clean sheet for the winning team
matches['Winning Team'] = matches.apply(lambda row: row['Home Team Name'] if row['Home Team Goals'] > row['Away
matches['Clean Sheet'] = matches.apply(lambda row: 1 if (row['Home Team Goals'] > row['Away Team Goals'] and ro

# Calculate the total number of clean sheets by winning teams
clean_sheets_by_winning_teams = matches[matches['Winning Team'] != 'Draw']['Clean Sheet'].sum()

# Calculate the total number of matches won
total_wins = len(matches[matches['Winning Team'] != 'Draw'])

# Average clean sheets by winning teams
```

```
average_clean_sheets = clean_sheets_by_winning_teams / total_wins

print(f"Total Clean Sheets by Winning Teams: {clean_sheets_by_winning_teams}")
print(f"Total Wins: {total_wins}")
print(f"Average Clean Sheets by Winning Teams: {average_clean_sheets:.2f}")
```

```
Total Clean Sheets by Winning Teams: 346
Total Wins: 662
Average Clean Sheets by Winning Teams: 0.52
```

Average Number of Goals by teams reach the semifinals

In [37]:
```
# Filter matches to only include semifinal matches
semifinal_matches = matches[matches['Stage'] == 'Semi-finals']

# Initialize variables to keep track of total goals conceded and number of teams
total_goals_conceded = 0
total_teams = 0

# Calculate goals conceded for teams in semifinal matches
for _, match in semifinal_matches.iterrows():
    total_goals_conceded += match['Home Team Goals'] + match['Away Team Goals']
    total_teams += 2  # Both teams in the match reached the semifinals

# Calculate average goals conceded
average_goals_conceded = total_goals_conceded / total_teams

print(f"Average number of goals conceded by teams that reach the semifinals: {average_goals_conceded:.2f}")
```

```
Average number of goals conceded by teams that reach the semifinals: 1.82
```

10. **Match Environment:**

Location (continent) of the World Cup impact the Performance of Teams

In [38]:
```
country_to_continent = {'Brazil': 'South America', 'Argentina': 'South America', 'Germany': 'Europe', 'France':
worldcups['Host Continent'] = worldcups['Country'].map(country_to_continent)
matches['Home Team Continent'] = matches['Home Team Name'].map(country_to_continent)
matches['Away Team Continent'] = matches['Away Team Name'].map(country_to_continent)
matches = matches.merge(worldcups[['Year', 'Host Continent']], on='Year')

performance = matches.groupby(['Host Continent', 'Home Team Continent']).apply(lambda x: (x['Home Team Goals'] 
print(performance)
```

```
Home Team Continent    Europe  South America
Host Continent                              
Europe               0.684211        0.62069
South America        0.750000        0.68750
```

11. **Game Dynamics:**

Average Goal-Scoring Time in World Cup Matches

In [39]:
```
# Filter events for goals
goals = players[players['Event'].str.contains('G', na=False)]

# Extract the goal-scoring time from the event column
goals['Time'] = goals['Event'].str.extract(r'(\d+)').astype(int)

# Calculate the average goal-scoring time
average_goal_time = goals['Time'].mean()

print(f'Average Goal-Scoring Time: {average_goal_time:.2f} minutes')
```

```
Average Goal-Scoring Time: 46.32 minutes
```

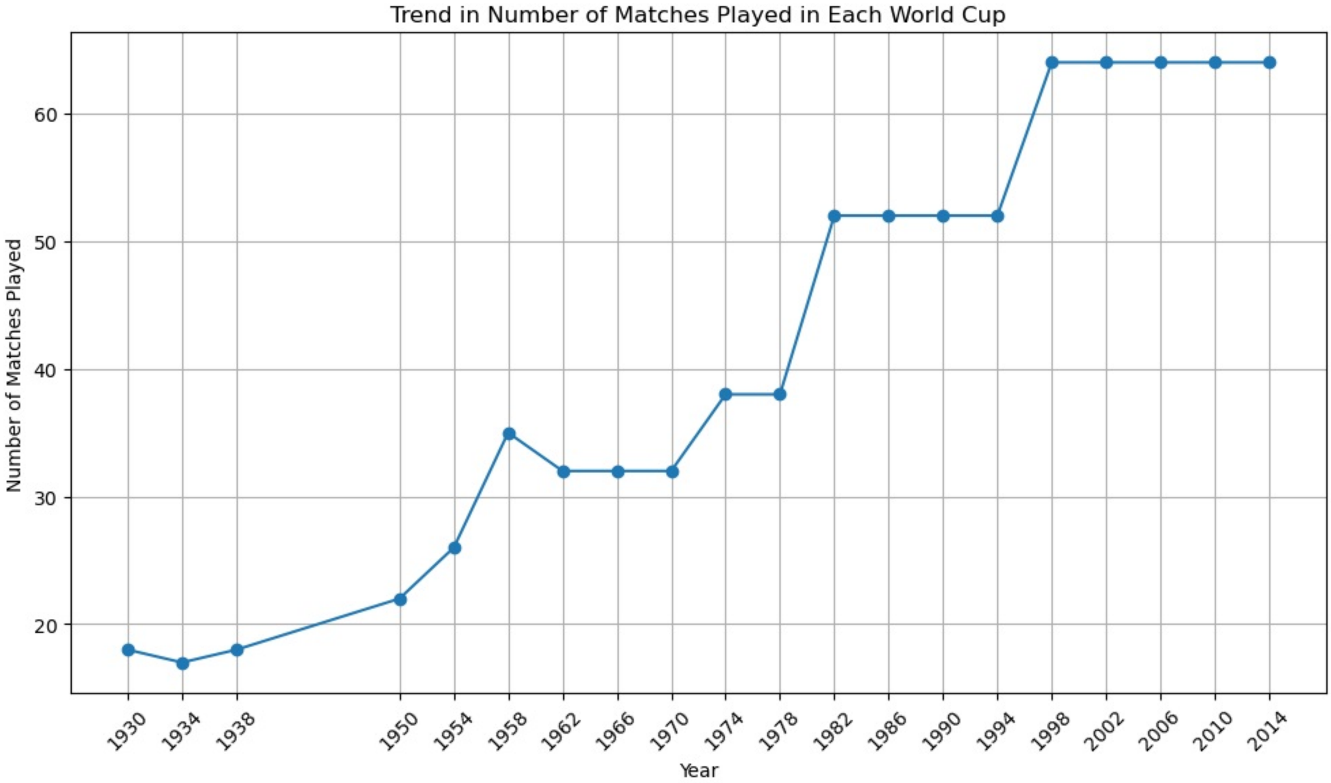
```
C:\Users\ELCOT\AppData\Local\Temp\ipykernel_10024\1001645918.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
urning-a-view-versus-a-copy
  goals['Time'] = goals['Event'].str.extract(r'(\d+)').astype(int)
```

12. **Historical Comparisons:**

Trend Number of Matches Played in each World Cup</font

In [40]:
```
# Plot the trend in the number of matches played in each World Cup
plt.figure(figsize=(10, 6))
```

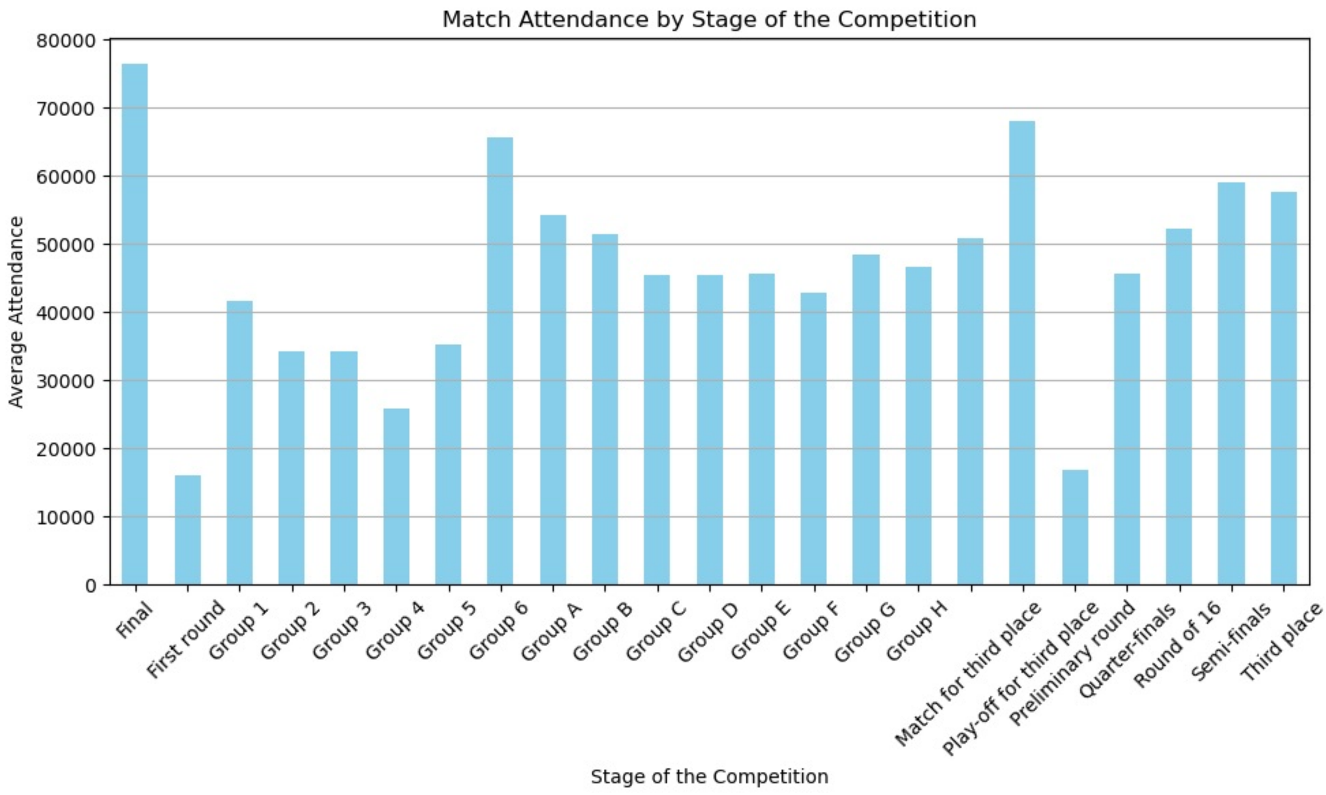
```
plt.plot(worldcups['Year'], worldcups['MatchesPlayed'], marker='o', linestyle='-')
plt.title('Trend in Number of Matches Played in Each World Cup')
plt.xlabel('Year')
plt.ylabel('Number of Matches Played')
plt.grid(True)
plt.xticks(worldcups['Year'], rotation=45)
plt.tight_layout()
plt.show()
```



13. **Attendance Analysis:**

Match Attendance vary by the Stage of the Competition</font

In [41]:
```
# Calculate the average attendance by stage of the competition
attendance_by_stage = matches.groupby('Stage')['Attendance'].mean()

# Plot the variation of match attendance by stage of the competition
plt.figure(figsize=(10, 6))
attendance_by_stage.plot(kind='bar', color='skyblue')
plt.title('Match Attendance by Stage of the Competition')
plt.xlabel('Stage of the Competition')
plt.ylabel('Average Attendance')
plt.xticks(rotation=45)
plt.grid(axis='y')
plt.tight_layout()
plt.show()
```



14. **Regional Performance:**

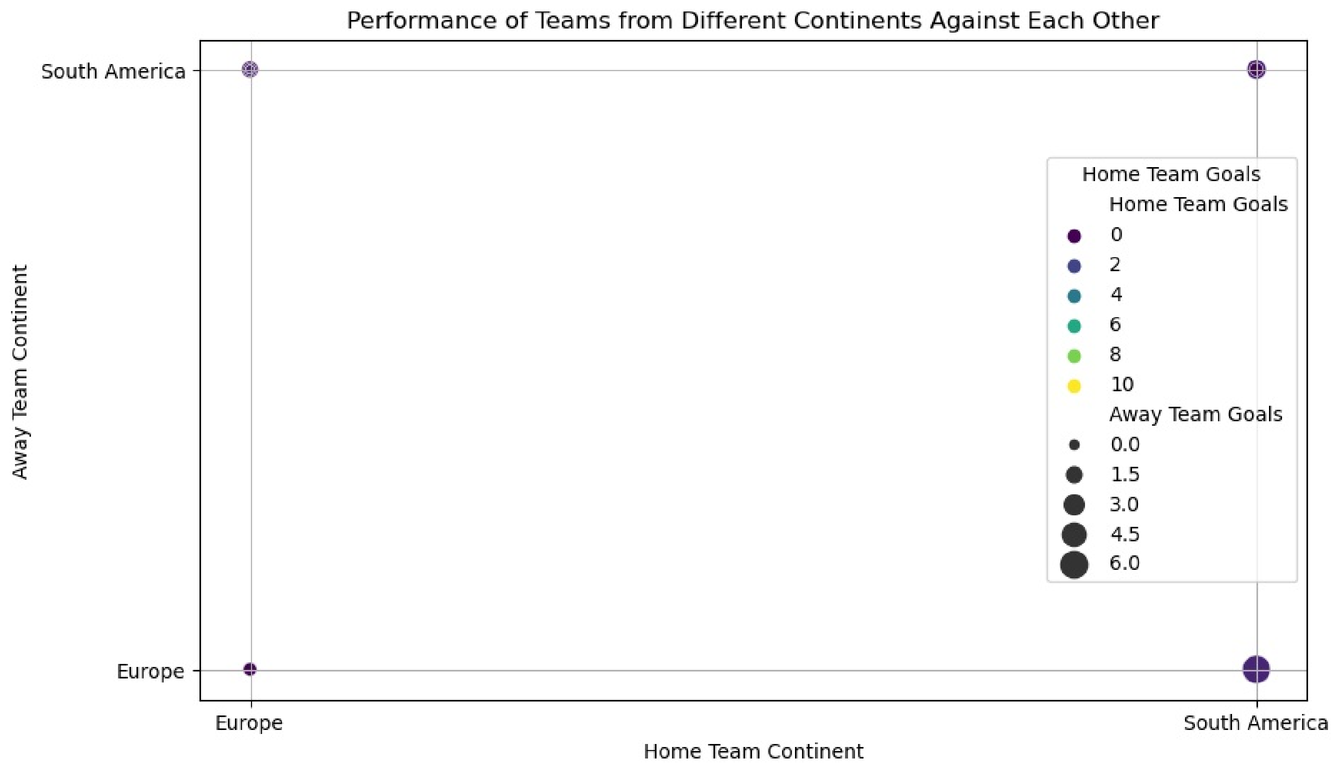
Continent has Produced Most World Cup Winners

In [42]:
```
# Count the number of World Cup wins by continent
winners_by_continent = worldcups['Winner'].value_counts()

# Display the continent with the most World Cup winners
most_winners_continent = winners_by_continent.idxmax()
most_winners_count = winners_by_continent.max()

print(f"Continent with the most World Cup winners: {most_winners_continent} ({most_winners_count} winners)")
```

```
Continent with the most World Cup winners: Brazil (5 winners)
```

Different Continents Perform against each Other</font

In [43]:
```
# Create a DataFrame to store the data
data = pd.DataFrame({
    'Home Team Continent': matches['Home Team Continent'],
    'Away Team Continent': matches['Away Team Continent'],
    'Home Team Goals': matches['Home Team Goals'],
    'Away Team Goals': matches['Away Team Goals']
})

# Plot the performance of teams from different continents against each other
plt.figure(figsize=(10, 6))
sns.scatterplot(data=data, x='Home Team Continent', y='Away Team Continent', hue='Home Team Goals', size='Away 
plt.title('Performance of Teams from Different Continents Against Each Other')
plt.xlabel('Home Team Continent')
plt.ylabel('Away Team Continent')
plt.legend(title='Home Team Goals')
plt.grid(True)
plt.show()
```



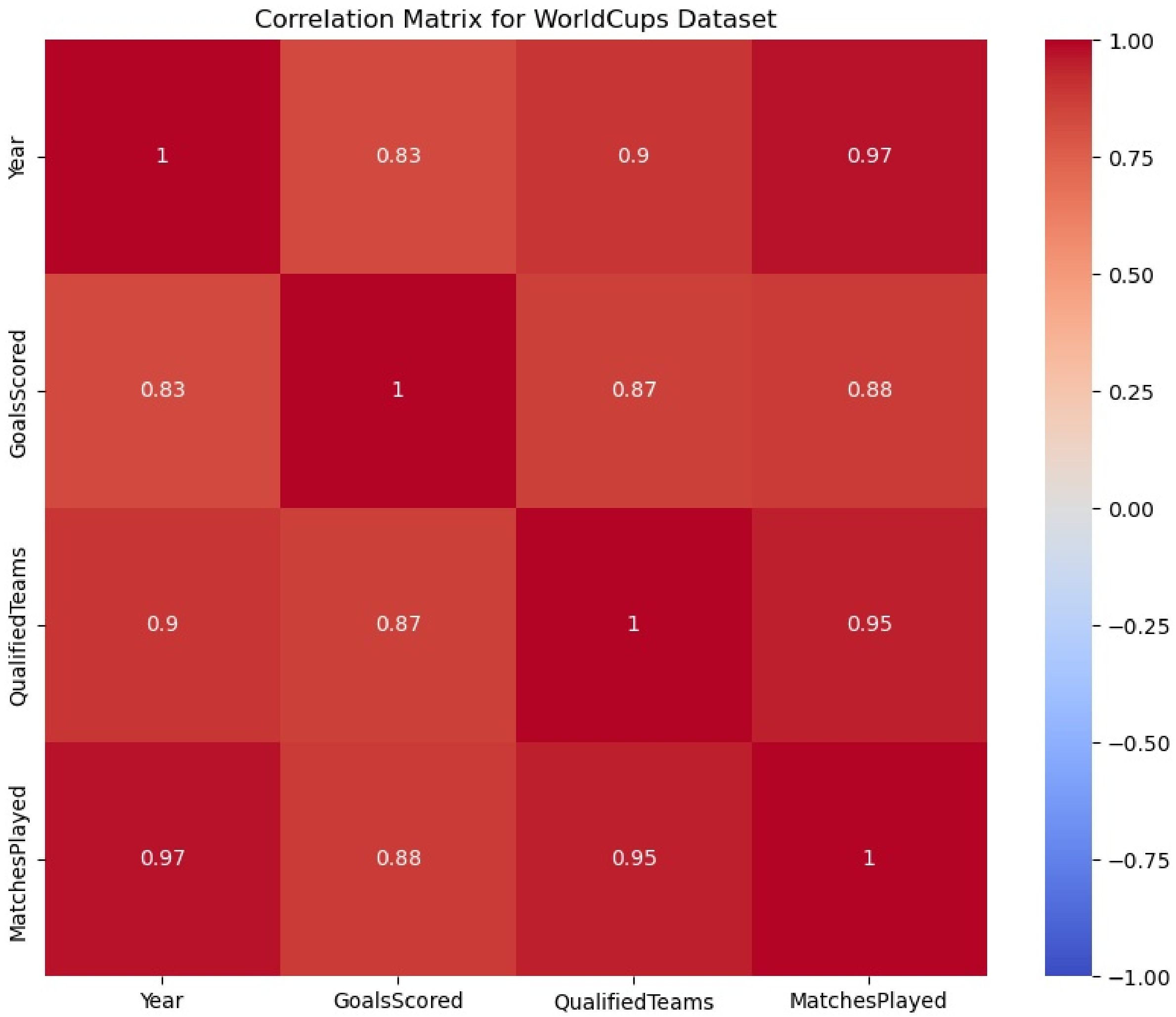
15. **Preparing Data and Machine Learning Models:**

Correlation Matrix

In [44]:
```
# Calculate correlation matrix for the WorldCups dataset
correlation_matrix = worldcups.corr()
print(correlation_matrix)

# Visualize the correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', vmin=-1, vmax=1)
plt.title('Correlation Matrix for WorldCups Dataset')
plt.show()
```

```
C:\Users\ELCOT\AppData\Local\Temp\ipykernel_10024\2258091173.py:2: FutureWarning: The default value of numeric_
only in DataFrame.corr is deprecated. In a future version, it will default to False. Select only valid columns
or specify the value of numeric_only to silence this warning.
  correlation_matrix = worldcups.corr()
```

```
                   Year  GoalsScored  QualifiedTeams  MatchesPlayed
Year           1.000000     0.829886        0.895565       0.972473
GoalsScored    0.829886     1.000000        0.866201       0.876201
QualifiedTeams 0.895565     0.866201        1.000000       0.949164
MatchesPlayed  0.972473     0.876201        0.949164       1.000000
```



Split Train & Test

In [45]:
```
# Remove commas and periods from the 'Attendance' column and convert to float
worldcups['Attendance'] = worldcups['Attendance'].astype(str).str.replace(',', '').str.replace('.', '').astype(
```

```
C:\Users\ELCOT\AppData\Local\Temp\ipykernel_10024\657489174.py:2: FutureWarning: The default value of regex wil
l change from True to False in a future version. In addition, single character regular expressions will *not* b
e treated as literal strings when regex=True.
  worldcups['Attendance'] = worldcups['Attendance'].astype(str).str.replace(',', '').str.replace('.', '').astyp
e(float)
```

In [46]:
```
# Prepare the data for linear regression
X = worldcups[['MatchesPlayed', 'Attendance']]
```

```
y = worldcups['GoalsScored']
```

In [47]:
```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

LinearRegression

In [48]:
```
# Create and train the linear regression model
linear_regressor = LinearRegression()
linear_regressor.fit(X_train, y_train)
```

Out[48]:
```
▾ LinearRegression
LinearRegression()
```

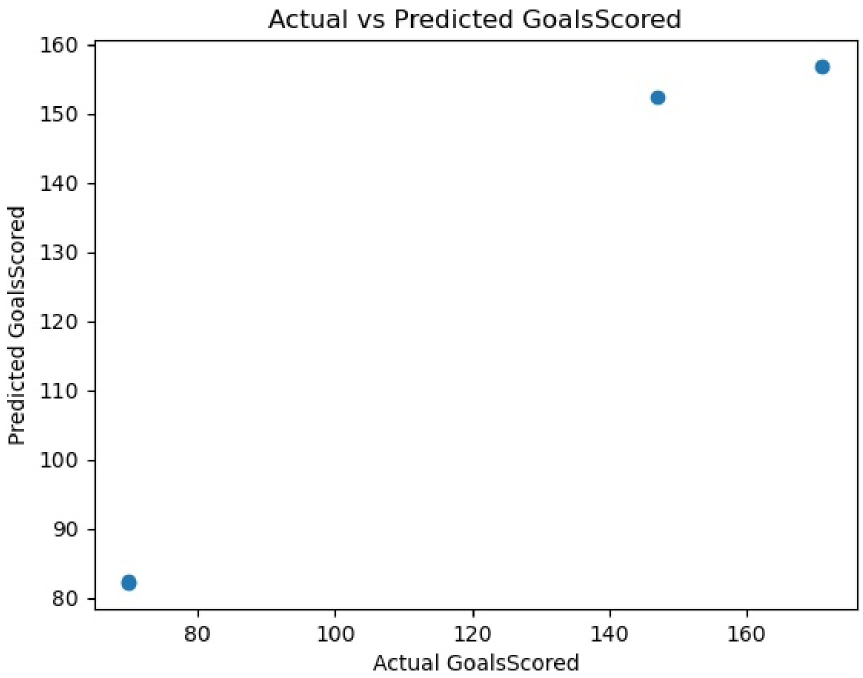
In [49]:
```
# Predict and evaluate the model
y_pred = linear_regressor.predict(X_test)
print('Coefficients:', linear_regressor.coef_)
print('Intercept:', linear_regressor.intercept_)
```

```
Coefficients: [ 1.99300432e+00 -7.84587942e-06]
Intercept: 51.11909565585704
```

In [50]:
```
# Calculate the model performance
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print('Mean Squared Error:', mse)
print('R^2 Score:', r2)
```

```
Mean Squared Error: 132.43767983246127
R^2 Score: 0.9354670825520959
```

In [51]:
```
# Visualize the results
plt.scatter(y_test, y_pred)
plt.xlabel('Actual GoalsScored')
plt.ylabel('Predicted GoalsScored')
plt.title('Actual vs Predicted GoalsScored')
plt.show()
```



Standard Scaler

In [52]:
```
# Initialize the standard scaler
scaler = StandardScaler()

# Fit and transform the features
X_scaled = scaler.fit_transform(X)

# Display the first few rows of the scaled features
print(pd.DataFrame(X_scaled, columns=['MatchesPlayed', 'Attendance']).head())
```

```
   MatchesPlayed  Attendance
0      -1.418125   -1.227463
1      -1.477710   -1.445275
2      -1.418125   -1.433118
3      -1.179784   -0.792222
4      -0.941444   -1.057024
```

Label Encoder

In [53]:
```
# Encode the categorical target variable 'Winner' using LabelEncoder
label_encoder = LabelEncoder()
worldcups['Winner'] = label_encoder.fit_transform(worldcups['Winner'])
```

In [54]:
```
# Prepare the features and target variable
X = worldcups[['Year', 'GoalsScored', 'MatchesPlayed', 'Attendance']]
y = worldcups['Winner']
```

MinMaxScaler

In [55]:
```
# Scale the features using MinMaxScaler
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)
X_scaled
```

Out[55]:
```
array([[0.        , 0.        , 0.0212766 , 0.07056794],
       [0.04761905, 0.        , 0.        , 0.        ],
       [0.0952381 , 0.13861386, 0.0212766 , 0.00393855],
       [0.23809524, 0.17821782, 0.10638298, 0.21157946],
       [0.28571429, 0.69306931, 0.19148936, 0.12578763],
       [0.33333333, 0.55445545, 0.38297872, 0.14166681],
       [0.38095238, 0.18811881, 0.31914894, 0.16441797],
       [0.42857143, 0.18811881, 0.31914894, 0.3721882 ],
       [0.47619048, 0.24752475, 0.31914894, 0.38485358],
       [0.52380952, 0.26732673, 0.44680851, 0.46603668],
       [0.57142857, 0.31683168, 0.44680851, 0.36680945],
       [0.61904762, 0.75247525, 0.74468085, 0.54169714],
       [0.66666667, 0.61386139, 0.74468085, 0.62986729],
       [0.71428571, 0.44554455, 0.74468085, 0.66775923],
       [0.76190476, 0.7029703 , 0.74468085, 1.        ],
       [0.80952381, 1.        , 1.        , 0.75114637],
       [0.85714286, 0.9009901 , 1.        , 0.72636669],
       [0.9047619 , 0.76237624, 1.        , 0.92926149],
       [0.95238095, 0.74257426, 1.        , 0.87325874],
       [1.        , 1.        , 1.        , 0.93774984]])
```

LogisticRegression

In [56]:
```
# Create and train the logistic regression model
logistic_regression = LogisticRegression()
logistic_regression.fit(X_train, y_train)
```

Out[56]:
```
▾ LogisticRegression
LogisticRegression()
```

In [57]:
```
# Predict the labels for the test set
y_pred = logistic_regression.predict(X_test)
```

In [58]:
```
print("Classification Report:")
print(classification_report(y_test, y_pred))

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Classification Report:
              precision    recall  f1-score   support

          70       0.00      0.00      0.00       2.0
         141       0.00      0.00      0.00       0.0
         147       0.00      0.00      0.00       1.0
         171       0.00      0.00      0.00       1.0

    accuracy                           0.00       4.0
   macro avg       0.00      0.00      0.00       4.0
weighted avg       0.00      0.00      0.00       4.0

Confusion Matrix:
[[0 2 0 0]
 [0 0 0 0]
 [0 1 0 0]
 [0 1 0 0]]
Accuracy: 0.0
```

```
C:\Users\ELCOT\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Pre
cision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division
` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\ELCOT\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Rec
all and F-score are ill-defined and being set to 0.0 in labels with no true samples. Use `zero_division` parame
ter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\ELCOT\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Pre
cision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division
` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\ELCOT\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Rec
all and F-score are ill-defined and being set to 0.0 in labels with no true samples. Use `zero_division` parame
ter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\ELCOT\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Pre
cision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division
` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\ELCOT\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Rec
all and F-score are ill-defined and being set to 0.0 in labels with no true samples. Use `zero_division` parame
ter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```